AddShare+: An Efficient Additive Secret Sharing Approach for Private Federated Learning

by

Bernard Asare

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Computer Science, Concentration in Applied Artificial Intelligence
in
School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Examining Committee

The following served on the Examining Committee for this thesis.

Internal Member(s): Burak Kantarci
Full Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Paria Shirani
Assistant Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Supervisor(s): Paula Branco
Assistant Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Iluju Kiringa
Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Federated Learning (FL) has emerged as a transformative approach for collaboratively training Machine Learning (ML) models without the need to share raw data. Despite its benefits, FL is susceptible to significant privacy threats, such as inference attacks that allow adversaries to reconstruct training data from shared model updates and eavesdropping attacks where malicious actors intercept communications to access sensitive information. This thesis addresses these challenges by introducing the AddShare and AddShare+ frameworks, which leverage additive secret sharing and advanced encryption techniques as means of protection during the FL process.

The AddShare framework transforms model updates into multiple shares distributed among clients, rendering it impossible for any single client or adversary to reconstruct the original model. AddShare offers three primary advantages: enhanced privacy through additive secret sharing, secure data transmission using Rivest–Shamir–Adleman (RSA) encryption, and improved efficiency via client grouping. By organizing clients into groups and distributing shares within these groups, AddShare significantly reduces communication overhead while maintaining robust privacy guarantees.

Building on these foundations, AddShare+ introduces two key enhancements. First, it employs selective weight sharing, where only a subset of model weights are shared, reducing computational and communication costs. AddShare+ utilizes a magnitude-based weight selection strategy, focusing on the most impactful weights to optimize the balance between model performance and privacy. Second, it replaces RSA with the more efficient Elliptic Curve Integrated Encryption Scheme (ECIES), enabling faster and more secure share distribution. These frameworks ensure strong privacy preservation while maintaining model accuracy, making them suitable for sensitive applications such as healthcare, finance, and the Internet of Things (IoT).

Comprehensive experimental evaluations demonstrate the efficacy of these frameworks. For instance, AddShare+ achieves comparable model accuracy to traditional FL methods on the Fashion - Modified National Institute of Standards and Technology (F-MNIST)

dataset, maintaining a Server-side Accuracy (SA) of 78.9%, compared to 79.3% with Federated Averaging (FedAvg), while reducing Federated Learning Time (FLT) by 13% relative to AddShare.

The thesis concludes with a case study on crop yield prediction, showcasing the practical application of AddShare+ in a smart farming context. While aggregated farming data holds immense potential for improving agricultural productivity and sustainability through ML and FL, farmers are often reluctant to share their operational data due to concerns about its impact on loan negotiations, insurance rates, land valuations, and competitive advantages in local markets. This real-world scenario demonstrates how the proposed framework can be applied to sensitive agricultural data, balancing the need for collaborative learning with data privacy concerns. These frameworks significantly enhance the privacy and security of FL systems against inference and eavesdropping attacks, promoting their broader adoption in various critical applications. Future research directions include further optimization of these frameworks and exploration of additional privacy-preserving techniques to enhance the security and efficiency of decentralized ML systems.

Acknowledgements

As I reflect on this transformative journey of pursuing my master's degree at the University of Ottawa, I am filled with profound gratitude for the numerous individuals who have contributed to my growth and the successful completion of this thesis. First and foremost, I give thanks to El-Elyon for His unwavering presence, guidance, and strength throughout this academic endeavor.

I extend my deepest and most heartfelt appreciation to my supervisor, Professor Paula Branco. Her role in my academic journey has been nothing short of transformative. Her support extended far beyond the realm of academic guidance; she became a pillar of strength during times when unforeseen challenges threatened to derail my master's journey. Her unwavering commitment to my success led her to consistently support me through various challenges, ensuring the continuity of my studies. Her meticulous attention to detail, consistent encouragement, and a genuine care for my well-being, has been instrumental not just in shaping this research, but in shaping my entire academic and personal growth.

I am also deeply grateful to Professor Iluju Kiringa and Professor Tet Yeap for their valuable insights, constructive feedback, and support throughout this process. Your expertise and guidance have significantly enhanced the quality and depth of this research. My sincere thanks to the Cyber Range Research Team for their collaborative spirit and technical expertise, with special gratitude to Dominica Amanfo for her invaluable support. To my family, your unconditional love and support have been my anchor. This achievement is as much yours as it is mine. I acknowledge the University of Ottawa for providing the resources and environment conducive to this research. Lastly, I appreciate all who have contributed to this work in various ways. This journey of personal and professional growth was made possible by your collective support.

Table of Contents

List of Tables

List of Figures

Abbreviations

AES Advanced Encryption Standard 43, 44

ARFED Attack-Resistant Federated Averaging 13

ASST Average Secret Sharing Time 63, 68, 70, 78, 86, 87, 89, 90, 95–97, 99, 113, 115, 136

ATT Average Training Time 63, 68, 71, 99, 136

CAA Client-side Average Accuracy 63, 68, 69, 78–82, 84, 99, 100, 136

CNN Convolutional Neural Network 71, 104, 110–112

CSS Combined Security Score 54, 55

DP Differential Privacy 17–19, 21, 26, 120

DSAA Data Science and Advanced Analytics 6

ECDLP Elliptic Curve Discrete Logarithm Problem 49

ECIES Elliptic Curve Integrated Encryption Scheme iv, 4, 6, 7, 12, 42, 43, 49, 61, 77, 78, 82–85, 118

ESORICS European Symposium on Research in Computer Security 6

F-MNIST Fashion - Modified National Institute of Standards and Technology iv, 64–66, 78–80, 83, 86, 89, 99, 100, 118, 136

FedAvg Federated Averaging v, 10, 29, 30, 36, 61, 63, 79–82, 84, 85, 113, 115, 118, 136

FHE Full Homomorphic Encryption 20, 21

FL Federated Learning iv, v, xiv, xv, 1–7, 9–30, 32–34, 36–41, 43, 47, 51, 52, 58, 59, 61, 64–72, 76–79, 85, 89–94, 98, 100–102, 105, 107, 111, 113–119, 136

FLT Federated Learning Time v, 63, 68, 70, 78, 86, 87, 89, 90, 95–97, 99, 100, 113, 115, 136

Gboard Google Keyboard 2

GDPR General Data Protection Regulation 1

GIS Geographic Information Systems 107

HE Homomorphic Encryption 17, 20, 21, 26, 120

HIPAA Health Insurance Portability and Accountability Act 1

IID Independent and Identically Distributed 24, 64, 67, 68

IoT Internet of Things iv, 2, 4, 28

KDF Key Derivation Function 43, 44

LSTM Long Short Term Memory 24

MAE Mean Absolute Error 111–113

MAPE Mean Absolute Percentage Error 111–114

MI Mutual Information 48, 51

MIA Membership Inference Attack 15, 16

ML Machine Learning iv, v, 1, 2, 11, 24, 64–66, 103, 105, 117

MNIST Modified National Institute of Standards and Technology 64, 65, 78, 79, 82, 86–89, 97, 99, 100, 118, 136

MSE Mean Squared Error 111

NIR Near-Infrared 108

OBD Optimal Brain Damage 55–57, 94–98, 101

OSF Ottawa Smart Farm 106

PHE Partial Homomorphic Encryption 20, 21

PIA Property inference attacks 16

PIPEDA Personal Information Protection and Electronic Documents Act 1

RGB Red-Green-Blue 108, 110

RMSE Root Mean Square Error 111–114

RNN Recurrent Neural Networks 104

RSA Rivest–Shamir–Adleman iv, 4, 6, 7, 12, 29, 34–36, 40, 42, 43, 61, 77, 78, 82–85

SA Server-side Accuracy v, 63, 68, 69, 78–84, 88, 94–97, 99, 100, 136

SecFedSA Secure Federated Stability Assessment 19, 20

SLURM Simple Linux Utility for Resource Management 77

SMC Secure Multiparty Computation 7, 13, 17, 21, 23, 26, 30

SVHN Street View House Numbers 64, 66, 78, 79, 82, 84, 86, 89, 90, 99, 100, 118, 136

UAV Unmanned Aerial Vehicles 103, 104, 108, 110, 112

Chapter 1

Introduction

1.1 Background

FL, a distributed ML approach that enables multiple parties to collaboratively train models while keeping their data local has transformed the ML landscape by fundamentally altering how distributed information is utilized for model development [79]. Here, ML refers to computational systems that learn and improve from experience to make predictions or decisions without being explicitly programmed.

Traditional centralized ML approaches aggregate data from multiple sources into a central source for model training. This method poses substantial privacy concerns such as data exposure and breaches, which can occur when data is transferred to a central source for training. Additionally, it faces legal and regulatory challenges such as General Data Protection Regulation (GDPR) [30], Personal Information Protection and Electronic Documents Act (PIPEDA) [37], and Health Insurance Portability and Accountability Act (HIPAA) [83], which often make it practical impossible to access and use sensitive data without violating privacy standards. FL addresses these issues by allowing the model training to occur locally on client devices. Instead of transferring raw data to a central server, only the model updates are shared and aggregated to form a global model. This method

ensures that raw data remains local, enabling the use of data from multiple sources that would otherwise be inaccessible due to privacy concerns or regulatory barriers. By decentralizing the learning process, FL can help align with stringent data privacy regulations and reduce the risk of data breaches associated with centralized data storage. Although FL addresses centralized ML challenges such as data exposure and breaches by preventing the need for raw data transfer and centralization, it is important to note that FL is not immune to privacy risks. Techniques such as inference attacks can still jeopardize data privacy, as malicious actors may attempt to reconstruct original data from the shared model updates [59]. Therefore, while FL offers a means to work with decentralized data, additional measures are often required to enhance data security.

The applications of FL span various domains where data sharing is typically infeasible due to privacy concerns. One of the most widely recognized applications of FL is in improving the text suggestions for Google Keyboard (Gboard) for most Android and iOS devices [101]. FL allows Gboard to learn from the typing patterns of millions of users without collecting their private data on central servers. By training models directly on users' devices, Gboard can enhance its predictive text and query suggestions while ensuring that sensitive information, such as personal messages and search queries, remains on the user's device. In healthcare, FL enables collaborative training of predictive models on patient data distributed across hospitals while ensuring sensitive patient data is never shared with a third party [10, 25, 55, 64]. This approach helps in developing robust models for disease diagnosis and treatment without compromising patient confidentiality. In the financial sector, FL is applied for the detection of fraudulent transactions by training models on transaction data from multiple financial institutions, enhancing fraud detection capabilities without exposing sensitive financial data [6, 12, 85]. The IoT sector leverages FL to improve the performance and security of smart devices, where devices collaboratively learn from local data to improve functionalities such as predictive maintenance and anomaly detection [3]. Additionally, FL is instrumental in autonomous vehicles, where it allows car manufacturers to collaboratively train models on driving data collected from multiple vehicles, enhancing the safety and efficiency of autonomous driving systems while preserving

user privacy [67]. These applications underscore the versatility and critical importance of FL. In all applications, the computational power of edge devices is leveraged, relieving the burden and cost on a central server.

Despite the inherent advantages of FL, research highlights significant privacy risks associated with the sharing of model updates [59, 105]. Adversaries with access to these updates may be able to infer sensitive information about the training data, potentially compromising the privacy of individual clients. This is a critical concern, particularly in sensitive domains such as healthcare, finance, and agriculture, where data privacy is of utmost importance. Several attack vectors have been identified in FL systems, including model inversion attacks [65], membership inference attacks [33, 60], and reconstruction attacks [59]. Model inversion attacks aim to reconstruct representative samples of the training data by exploiting the shared model updates. Membership inference attacks attempt to determine whether specific data samples were part of the training set. Reconstruction attacks leverage the shared updates to recover portions of the original training data, posing a significant threat to data privacy. Addressing these privacy concerns is crucial for the practical adoption and deployment of FL in real-world scenarios. Failing to mitigate these risks can undermine the trust and confidence of clients in the FL system, hindering its widespread adoption and limiting its potential benefits. As such, developing robust privacy-preserving techniques for FL has become a critical area of research, aiming to strike a balance between preserving data privacy and maintaining the accuracy and performance of the trained models.

The agricultural sector presents unique privacy challenges that make it an ideal domain for testing privacy-preserving FL approaches. While aggregated farming data holds immense potential for improving agricultural productivity and sustainability, farmers are often reluctant to share their operational data due to legitimate business concerns [78]. Crop yield data, which directly reflects a farm's productivity and profitability, can significantly impact farmers' negotiating positions for loans, insurance rates, and land valuations when exposed to financial institutions [32, 90, 103]. Moreover, detailed yield information combined with farming practices represents proprietary operational knowledge developed

over generations, where sharing could compromise competitive advantages in local agricultural markets. This sensitivity around agricultural data sharing, particularly in regions where farmers compete for land leases or market share, creates a compelling case for privacy-preserving machine learning approaches that can enable collaborative learning while protecting individual farm data.

1.2 Motivation

The motivation behind this research is driven by the pressing need to address privacy vulnerabilities in FL, especially given its widespread applications in sensitive domains such as healthcare, finance, and IoT, where data privacy is paramount. The AddShare and AddShare+ frameworks introduced in this thesis specifically target these privacy challenges by integrating robust privacy-preserving techniques, such as additive secret sharing and advanced encryption mechanisms. These frameworks transform model updates into shares distributed across multiple clients, making it practically impossible for any single client or adversary to reconstruct the original model. This approach significantly enhances data privacy, ensuring that even if an attacker intercepts some shares, they cannot derive meaningful information without access to all distributed shares. Moreover, AddShare uses RSA encryption, while AddShare+ employs the more efficient ECIES to secure the communication of these shares. This dual-layer security strategy not only protects against inference attacks but also safeguards data during transmission against eavesdropping and unauthorized access. The frameworks effectively neutralize the threats posed by sophisticated inference attacks, thereby maintaining the confidentiality of the training data. The significance of our research lies in its dual focus on enhancing privacy and maintaining efficiency, addressing the critical need for secure FL systems that do not compromise on performance. By developing and implementing these frameworks, this research aims to foster greater trust and confidence in FL systems, facilitating their broader adoption in sensitive and critical applications. This research not only advances the field of privacy-

preserving FL but also sets a foundation for future innovations aimed at further enhancing the security and efficiency of decentralized machine learning systems.

1.3 Objectives

The primary objective of this research is to develop a secure and efficient approach for FL that preserves the privacy of clients' data while maintaining the accuracy and performance of the trained models. To achieve this goal we defined the following sub-goals listed below:

- **Sub-Goal 1: Develop an efficient lightweight secure mechanism for preserving privacy in FL using selective additive weight sharing mechanism that reduces communication overhead.** The proposed AddShare+ framework successfully achieved this sub-goal through the implementation of selective weight sharing, optimized client grouping strategies, and a lightweight encryption scheme. These innovations significantly reduced communication overhead while maintaining privacy and accuracy in the FL process. We conduct a comprehensive experimental evaluation and analysis to assess the performance of the proposed approach across multiple datasets and configurations.
- **Sub-Goal 2: Investigate the practical application of the proposed approach in the domain of crop yield prediction, demonstrating its potential real-world impact and effectiveness in a real-world scenario.** We demonstrate this by applying AddShare+ to a smart farming environment composed of multiple farms using on high-quality real-world data acquired from AreaX.O. The experimental evaluation conducted confirms the effectiveness of AddShare+ in this real-world setting.

By addressing these objectives, this research aims to contribute a novel and practical solution for privacy-preserving FL that balances the trade-offs between privacy, accuracy,

and efficiency, enabling the broader adoption and deployment of FL in various domains where data privacy is a critical concern.

1.4 Research Contribution

This thesis advances the field of privacy-preserving FL through several significant contributions that address key challenges in data privacy and system efficiency in FL. Our research introduces novel frameworks and methodologies that enhance both the security and practical implementation of FL systems, as detailed in the following contributions.

- AddShare, our initial framework, introduces a privacy-preserving approach that combines additive secret sharing with RSA encryption to protect model updates during the FL process while maintaining model accuracy. This framework implements client groups where clients can form smaller groups (3, 5, 10) for weight sharing using server-based group formation. The development and validation of AddShare were published in European Symposium on Research in Computer Security (ESORICS) 2023 International Workshops, demonstrating its effectiveness in preserving privacy in federated learning environments.

- Building on these foundations, AddShare+ enhances the initial framework by introducing selective weight sharing and ECIES encryption, significantly improving computational efficiency while preserving privacy guarantees. AddShare+ maintains the client grouping capability but expands it to include both server-based and client-based group formation options. This enhanced framework was presented at IEEE International Conference on Data Science and Advanced Analytics (DSAA) 2024, validating its improvements in efficiency and practicality. A comparison of these frameworks is shown in Table 1.1.

- Through extensive experimental validation across multiple datasets, complemented by real-world implementation in smart farming scenarios using AreaX.O data, our

Table 1.1: Key Contribution between AddShare and AddShare+

Framework	AddShare	AddShare+
Additive Secret Sharing	All weights	Portion of weights
Encryption	RSA encryption	ECIES encryption
Client groups (Clients can form smaller groups (3, 5, 10) for weight sharing)	Server-based group formation	Server-based group formation
		Client-based group formation

research demonstrates the frameworks' effectiveness in both controlled and practical settings. These contributions collectively advance the field of privacy-preserving federated learning and establish a foundation for secure, efficient collaborative learning in sensitive domains.

1.5 Organization of Thesis

The thesis is structured as follows: Chapter 2 provides an extensive background and literature review in the area of FL. It explores the threats and attacks faced by FL systems and the various privacy-preserving approaches proposed to mitigate these challenges, with particular emphasis on Secure Multiparty Computation (SMC) techniques, specifically additive secret sharing. This chapter also outlines the current challenges and drawbacks of existing research in this domain, highlighting the need for further improvements, and discusses open security issues in the application of FL in smart farming. Chapter 3 delves into the complete system design and the various components proposed in this thesis. It provides a detailed explanation of how additive secret sharing is employed to carry out privacy preservation in FL, and describes different strategies for client grouping, including server-side and client-side group formation techniques, as well as different encryption approaches such as RSA and ECIES. Chapter 4 presents the experimental settings, results, and analysis, discussing the empirical findings obtained from evaluating the proposed approach across multiple datasets and configurations. This chapter introduces the performance evaluation metrics used to assess the proposed approach and includes an in-depth analysis of the accu-

racy, efficiency, and privacy trade-offs observed during the experiments and examines the impact of various factors, such as client grouping strategies and selective weight sharing, on the overall performance and privacy guarantees. Chapter 5 focuses on the practical application of the proposed framework in the domain of crop yield prediction. It describes the agricultural dataset used, the experimental setup, and the specific configurations employed for this case study, presenting and analyzing the results obtained from applying the proposed approach to the crop yield prediction task, highlighting its effectiveness and potential real-world implications. Finally, Chapter 6 concludes the thesis by summarizing the key findings and contributions of this research work, underscoring the direct impact of the proposed approach on improving crop yield prediction and, consequently, enhancing food security and agricultural productivity. Moreover, this chapter reviews potential future research directions and outlines the next steps to advance knowledge in this domain further, addressing any limitations or open challenges identified during the study.

1.6 Publications

The work in Chapter 3 and Chapter 4 is based on the following two publications:

- Bernard Atiemo Asare, Paula Branco, Iluju Kiringa, and Tet Yeap. AddShare: A Privacy-Preserving Approach for Federated Learning. In: *Computer Security. ESORICS 2023 International Workshops*, pages 299-309, Springer, Cham, 2024
- Bernard Atiemo Asare, Paula Branco, Iluju Kiringa, and Tet Yeap. "AddShare+: Efficient Selective Additive Secret Sharing Approach for Private Federated Learning," *2024 IEEE 11th International Conference on Data Science and Advanced Analytics (DSAA)*, San Diego, CA, USA, 2024, pages. 1-10

Chapter 2

Background and Literature Review

2.1 Introduction

This chapter provides background information and a review of the literature related to the privacy and security of FL systems. It covers the fundamental concepts of FL, including the FL process, security threats, potential attacks, and approaches to privacy preservation. Understanding this background is crucial for addressing the privacy and security challenges in applying FL to emerging domains like smart farming.

2.2 Federated Learning

FL has demonstrated its potential in preserving privacy while enabling collaboration among multiple parties to train machine learning models by allowing the exchange of model-related data while keeping the raw data of the parties private [79]. The FL process begins by distributing a model architecture with default initialized weight, denoted as M_0, from a central server to all n client devices. Each client, indexed by i, then trains this model locally using its own private data, denoted as D_i, without exchanging any raw data samples,

generating an updated local model, denoted as M_i'. These local model updates are then shared with the central server. The central server aggregates the local model updates, typically through a weighted average using FedAvg [57] algorithm in order to obtain an improved global model, denoted as M_{global}, using the formula:

$$M_{\text{global}} = \frac{1}{n} \sum_{i=1}^{n} M_i' \tag{2.1}$$

The federated global model M_{global} is then shared back with all client devices, and the cycle continues for multiple rounds until reaching an accurately trained model. This iterative process converges over multiple rounds, refining the global model based on the insights gained from the diverse local datasets without exposing raw data to the central server. The detailed steps of the FL process, including the initialization, local training, and aggregation phases, are presented in Algorithm 2.1.

Algorithm 2.1 Federated Learning Process

Input: Initial global model M_0, number of clients n, number of iterations T

Central Server:
Initialize global model $M_{global} \leftarrow M_0$
for each round $t = 1$ to T **do**
    Distribute M_{global} to all n clients
    Clients:
    for each client $i = 1$ to n **in parallel do**
        Receive M_{global}
        Train M_{global} locally using private data D_i
        Generate updated local model M_i^t
        Send M_i^t to central server
    end for
    Central Server:
    Aggregate local models to update global model:
    $M_{global} \leftarrow \frac{1}{n} \sum_{i=1}^{n} M_i^t$
end for
Output: Trained global model M_{global}

2.3 Federated Learning Security Threat Model

FL systems face complex security challenges that require a comprehensive threat modeling approach to understand and mitigate potential risks and attacks. Some of these threats and attacks can compromise the confidentiality, integrity, and availability of FL systems. This section presents a structured analysis of the security landscape in FL, examining the assets requiring protection, capabilities of potential adversaries or attackers, and the necessary security measures to maintain system integrity.

2.3.1 Critical Assets and Protection Requirements

The FL architecture contains several critical assets that require protection. The global model stands as the primary asset, where its integrity and performance directly impact the system's effectiveness. Client training data represents another crucial asset, containing potentially sensitive information that must remain confidential. The model update transmission process, server aggregation mechanisms, and client participation patterns also constitute vital assets requiring protection against unauthorized access or manipulation.

2.3.2 Adversary Types and Capabilities

FL enables collaborative ML model development across distributed devices without centralized data storage. The decentralized architecture introduces significant security vulnerabilities from both trusted participants and external actors, each possessing distinct capabilities and attack vectors. This section examines the comprehensive threat landscape, encompassing both insider and outsider threats that can compromise the integrity and security of FL systems. Insider threats emerge when fundamental trust relationships within FL systems become potential attack vectors. These manifest primarily through compromised aggregation servers and malicious clients. Aggregation servers, which coordinate the FL process, can exploit their privileged position to manipulate global models, extract

sensitive information from updates, or orchestrate targeted attacks. Similarly, participating clients can compromise system integrity by submitting manipulated model updates that degrade global model performance or introduce targeted attacks. The external threat landscape is equally concerning, as the FL architecture necessitates the exchange of model updates between clients and servers or among clients themselves.

External attackers can intercept or manipulate these updates in transit through various techniques, including eavesdropping [95] on network traffic [102] and man-in-the-middle attacks [15]. These attack vectors potentially expose sensitive information contained within the model updates. To mitigate external threats, robust security measures incorporating encryption, authentication, and secure communication protocols are essential [1]. Asymmetric encryption serves as a particularly effective countermeasure for protecting weight shares exchanged among clients. Implementation of RSA [71] encryption and its faster alternative, ECIES [17], ensures that weight shares remain protected during transit. By encrypting these shares with the recipient client's public key, the system guarantees that only intended clients can decrypt and access the information, thereby maintaining both privacy and integrity of model updates against external compromise.

Understanding the technical capabilities, motivations, and potential impact of both insider and outsider threats is essential for developing comprehensive security measures that preserve the trustworthiness of FL systems. This dual threat landscape necessitates a multilayered security approach that addresses both the privileged access of insiders and the interception capabilities of external adversaries.

2.3.2.1 Malicious Clients and Servers

Malicious clients pose a significant threat to the accuracy, integrity, and security of collaborative models in FL environments [7, 19, 53]. These attackers, who are legitimate system participants, intentionally send inaccurate or harmful model updates to the central server [18, 31], undermining the effectiveness of the FL process. The primary objective of malicious clients is to compromise the model in various ways, such as injecting biased

updates that favor specific patterns or classes, thereby reducing the model's fairness and generalization capabilities. They can also insert hidden triggers or patterns into models that can later induce undesirable behavior or weaken the system [94]. In addition, malicious clients might exploit vulnerabilities in the FL environment to steal or modify sensitive data from other clients, leading to privacy invasions and data breaches. Another severe insider threat in FL arises from the potential compromise of the aggregating server. If a malicious actor gains control over the server, they can manipulate the aggregation process, compromising the global model's utility and data privacy. This can involve steering the model in a biased direction through gradient manipulation or exploiting access to aggregated data to extract sensitive information, thereby violating user privacy guarantees [74, 92]. This threat is particularly concerning in contexts involving highly sensitive data, such as healthcare or financial sectors. To mitigate these risks, the FL architecture can implement various defense mechanisms and one of such effective methods is Attack-Resistant Federated Averaging (ARFED) which is based on Outlier Elimination [44]. ARFED employs a robust aggregation technique that includes outlier detection to identify and exclude model updates that deviate significantly from the norm. This helps detect and prevent potential gradient manipulation from being incorporated into the global model, enhancing the security and reliability of the FL process.

2.3.2.2 Honest but Curious Clients and Servers

A significant threat to the privacy and security of model updates in FL comes from honest but curious servers [45] where the server may follow the prescribed protocol but is still interested in analyzing the model updates to learn more about the training data and devices. The server can potentially reveal sensitive information about the training data and devices by analyzing the model updates. One approach to protecting the privacy of model updates is through the use of cryptographic techniques such as SMC.

2.3.2.3 Passive Eavesdropper

While insider threats in FL raise concerns about malicious participants, outsider attacks pose an equally significant risk. One such outsider threat is the passive eavesdropper, a malevolent actor who intercepts data transmissions between devices and the central server without directly participating in the training process [86]. These eavesdroppers can exploit vulnerabilities in communication channels to listen in on the exchange of model updates, potentially gleaning sensitive information. An eavesdropper might intercept gradients and analyze these gradients in order to reconstruct the local model on participating devices, potentially inferring sensitive training data or even the original training dataset itself [97]. This poses a significant privacy risk, as it could expose personal information or intellectual property embedded within the training data.

2.3.2.4 Active Eavesdropper

Another prominent threat is the active eavesdropper. Such an attacker strategically positions themselves on the network, intercepting data exchanged between participants. Although this eavesdropper could passively observe the data, potentially gleaning sensitive information or model insights, they actively tamper with the data, injecting noise or manipulating gradients to mislead the training process and compromise the resulting model [80]. Defending against active eavesdroppers requires robust cryptographic measures to ensure data confidentiality and integrity throughout the FL lifecycle.

2.3.3 Attack Vectors in Federated Learning

The distributed nature of FL systems exposes multiple attack vectors that adversaries can exploit to compromise system security and privacy. Network communication channels represent a primary vulnerability, where both passive and active adversaries can intercept model updates during transmission between clients and servers [15, 95]. The model update

protocol itself presents another critical attack surface, where malicious participants can manipulate update parameters or inject poisoned gradients to compromise model performance [7]. The aggregation process, central to FL's operation, becomes vulnerable when adversaries exploit the server's privileged position to manipulate the global model or extract sensitive information from aggregated updates [23, 74]. Within the training data pipeline, adversaries can compromise local training processes to introduce targeted vulnerabilities or backdoors that persist through model iterations [7]. These diverse attack vectors necessitate a comprehensive security framework that addresses vulnerabilities at each layer of the FL architecture, implementing appropriate countermeasures to maintain system integrity and participant privacy.

2.4 Potential Attacks in Federated Learning

Building upon the established threat models outlined in Section 2.3, this section delves into the specific attacks that these threats can leverage within the decentralized training paradigm of FL.

2.4.1 Membership Inference Attacks

While FL promises privacy-preserving collaboration, its distributed nature introduces unique vulnerabilities. One such vulnerability is Membership Inference Attack (MIA), where an adversary aims to determine if a specific data point was used to train the model [23]. As innocuous as it seemingly appears to be, this attack can have severe consequences when individuals within a training set are identified. Attackers can potentially reidentify individuals by linking their data to other sources or manipulate the model's behavior for targeted attacks. MIAs exploit the inherent statistical properties of the model, analyzing its predictions or gradients on crafted inputs to distinguish members from non-members [23, 74]. Different attack strategies exist, leveraging various data characteristics and model properties. Gradient-based methods analyze the model's sensitivity to specific inputs [87], while

prediction-based approaches compare the model's confidence on member and non-member data [54]. The effectiveness of these attacks depends on several factors, including the size and diversity of the training data, the model architecture, and the attacker's access to auxiliary information. Understanding and mitigating MIAs is crucial for ensuring the true privacy of individuals participating in FL, safeguarding their data from unauthorized exposure and potential harm.

2.4.2 Property Inference Attacks

Beyond the well-known threats of MIA, FL also faces the challenge of Property inference attacks (PIA). Here, attackers aim to extract broader, aggregate characteristics of the training data, even without pinpointing specific individuals or attributes [88]. This can include inferring demographics like age or location, the presence of specific categories like medical conditions, or even the overall size or composition of the training datasets. These attacks leverage the inherent leakage of information in the model updates exchanged during federated training. While not directly revealing individual data points, they can still pose significant privacy risks by disclosing sensitive group characteristics or enabling further targeted attacks. Understanding and mitigating PIA is crucial for ensuring the privacy-preserving nature of FL, particularly in domains where sensitive data is involved.

2.4.3 Data Reconstruction Attacks

Attackers can still exploit information leakage during the training process of FL through data reconstruction attacks which are particularly concerning, as they attempt to reverse engineer individual training data points from shared model updates or gradients [99]. These attacks often target intermediate features learned within local models, which can be more informative than the global model itself. The attacker's success depends on various factors, including the model architecture, training data properties, and the aggregation mechanism used. Recent advancements have shown attacks capable of reconstructing high-fidelity

data, even in scenarios with secure aggregation protocols [26]. The potential consequences of successful data reconstruction are significant, ranging from exposing sensitive attributes to enabling further targeted attacks. Therefore, understanding and mitigating these vulnerabilities is crucial for ensuring the long-term viability of FL in privacy-sensitive domains.

2.5 Privacy Preservation

There is a motivation to research privacy-preserving techniques for FL due to the potential leakage of private information about training data [105]. The major approaches to tackling the various threats and attacks discussed in Section 2.3 and 2.4 are discussed next, they include: Differential Privacy (DP), Homomorphic Encryption (HE) and SMC. DP protects privacy by adding statistical noise to either model updates [106] or local training data to obscure the contribution of any single individual's data. Despite the perks of this approach, model performance can be impacted negatively [82] therefore there is a need to balance privacy and utility by employing methods such as the injection of noise only into a subset of model updates in order to improve utility [81]. On the other hand, HE enables computations on encrypted data without needing to decrypt it first, providing robust privacy protection [42]. However, HE remains computationally intensive, requires key management and distribution mechanisms [52], and can significantly slow down the training process. This trade-off between security and efficiency is a critical consideration in deploying HE in FL systems.

Additionally, SMC allows multiple parties to jointly compute a function while keeping their individual inputs private. A commonly used SMC technique is additive secret sharing [96], which allows multiple parties to jointly compute a function while keeping their inputs private as illustrated in Figure 2.1. In this technique, when applied in FL, the model weights of each client are split into random additive shares and exchanged among clients. Each individual client computes the sum of the shares received which is then sent to the server. The server aggregates the summed-up shares received from each client, yielding

the aggregate model update, without learning any information about the individual shares or the underlying data. Additive secret sharing [96] provides strong privacy guarantees as long as the server does not have access to all the shares since it is impossible to recover the original weight without all the shares. This makes it an effective solution for mitigating the threat of an honest but curious server. These privacy-preserving techniques are essential for enhancing the security and privacy of FL systems. Table 2.1 summarizes some related approaches to privacy preservation for FL.

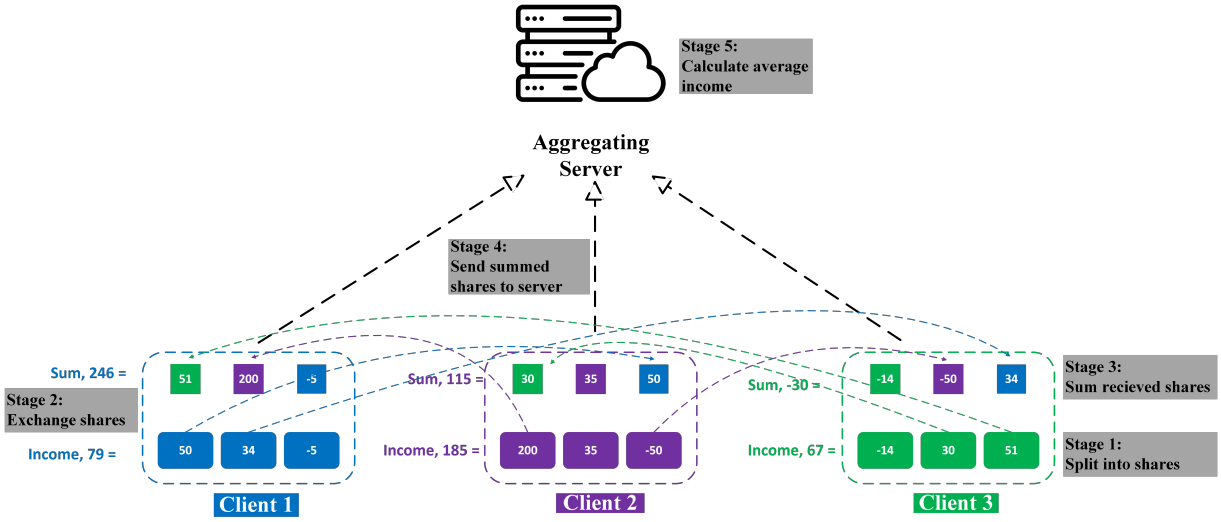

Figure 2.1: Illustration of the application of additive secret sharing in finding the average income of 3 homes.

2.5.1 Differential Privacy

At its core, DP guarantees that the inclusion or exclusion of any single data point will not significantly impact query output distributions [36]. As such, adversaries cannot infer with high confidence whether any individual contributed data or not [27]. DP protects privacy by adding calibrated statistical noise to either model weights or dataset [28] as well as restricting outputs [47].

Table 2.1: Privacy Preservation Approaches

Related Work	Differential Privacy	Homomorphic Encryption	Secure Multi-Party Computation
LDP-FL [77]	✓	×	×
LDP-Fed [81]	✓	×	×
SecFedSA [70]	✓	×	×
HybridAlpha [98]	✓	✓	×
BatchCrypt [104]	✓	×	×
xMK-CKKS [56]	✓	×	×
SecAgg [14]	×	×	✓
SecAgg+ [8]	×	×	✓
Scotch [61]	×	×	✓
FedShare [29]	×	×	✓
AddShare (our proposal) [11]	×	×	✓
AddShare+ (our proposal) [4]	×	×	✓

Formally, (ϵ, δ)-DP guarantees that for any two datasets D and D' differing in at most one client's data, and for any subset of outputs S, the probability of observing an output in S is approximately the same, up to a small multiplicative factor e^{ϵ}, and an exception probability δ. The definition is stated as follows:

$$\Pr[\mathcal{A}(\mathcal{D}) \in S] \leq e^{\epsilon} \Pr[\mathcal{A}(\mathcal{D}') \in S] + \delta \tag{2.2}$$

The privacy guarantee given by DP is controlled by the privacy budget (ϵ, δ), where smaller values of ϵ and δ provide stronger privacy. In the context of FL, DP is typically achieved by adding carefully calibrated noise to the model updates from clients.

Truex et al [77] introduced LDP-Fed, a FL framework designed to offer formal privacy guarantees to clients by leveraging local DP techniques. This framework allows for efficient training of complex neural networks while ensuring user privacy through local noise injection before sharing model updates within the FL process. The degree of noise applied dictates the strength of the differential privacy guarantee, where higher noise levels enhance privacy but may compromise model accuracy and slow convergence. Striking the optimal balance between privacy, accuracy, and convergence speed depends on the specific use case, target performance metrics, and iteration constraints. Addressing the challenges of stability assessment in smart cyber-physical grids, Ren et al [70] proposed Secure Federated Stability Assessment (SecFedSA), a secure and scalable federated learning framework. By

integrating a differential privacy mechanism, specifically the Gaussian mechanism, their approach ensures data privacy in distributed system architectures. Their experiments on benchmark systems demonstrated the effectiveness of SecFedSA, showing its potential applicability in real-world scenarios such as power companies, grid operators, and industries reliant on complex, interconnected networks.

2.5.2 Homomorphic Encryption

HE allows a third party to perform computations on encrypted data while preserving the underlying data's structure [107]. This is achieved through a ring homomorphism transformation from the message space M to the cipher space C by a map:

$$\sigma : M \to C \tag{2.3}$$

For any $x, y \in M$, this transformation satisfies:

$$\sigma(x) + \sigma(y) = \sigma(x + y) \quad \text{and} \quad \sigma(x) * \sigma(y) = \sigma(x * y). \tag{2.4}$$

In other words, given two messages m_1 and m_2, if the encryption function E is a ring homomorphism, then:

$$E(m_1) + E(m_2) = E(m_1 + m_2) \quad \text{or} \quad E(m_1) * E(m_2) = E(m_1 * m_2). \tag{2.5}$$

If the encryption function supports both additive and multiplicative homomorphism, it is commonly known as Full Homomorphic Encryption (FHE). FHE enables arbitrary computations on encrypted data, providing strong privacy guarantees because any mathematical operation can be performed on ciphertexts without decryption [42]. However, FHE is computationally intensive and often impractical for many FL applications due to its significant overhead in terms of time and computational resources. Partial Homomorphic

Encryption (PHE), on the other hand, supports only one of these operations—either addition or multiplication. PHE offers better performance with reduced computational costs, making it more feasible for practical applications. For example, some PHE schemes enable encrypted dot products for model updates, balancing privacy and efficiency more effectively than FHE [20]. When implemented carefully and effectively, partial homomorphism can provide cryptographic privacy with manageable overheads [40].

Some frameworks, like HybridAlpha [98] apply best of both worlds by combining HE with DP in order to guard against inference attacks in FL. Although it provides certain guarantees, the time and computational overhead cannot be ignored. To further reduce these costs, BatchCrypt was proposed, which encrypts batches of quantized gradients into a long integer in a single operation, rather than encrypting individual gradients with full precision [104]. This method significantly reduces computation and communication overhead in cross-silo FL. Another advancement is xMK-CKKS [56], an improved version of the MK-CKKS multi-key HE protocol. It encrypts model updates using an aggregated public key, thereby preventing privacy leakage and resisting collusion between devices and the server. Unlike MK-CKKS, xMK-CKKS allows the server to decrypt only the aggregation of shared encrypted model updates, further enhancing privacy protection in FL.

2.5.3 Secure Multi-Party Computation

SMC uses cryptographic techniques to enable multiple parties to jointly compute a function over their inputs while keeping those inputs private. A prominent method in SMC is secret sharing, specifically additive secret sharing, which is used to distribute model updates into shares that can be aggregated privately [96]. In this technique, each participant splits their data into multiple shares and distributes these shares among other participants. No single participant can reconstruct the original data from their share alone, preserving the privacy of the data.

Formally, suppose we have n participants and a value v that a participant wants to

share. The participant generates n random values $v_1, v_2, \ldots, v_{n-1}$ such that:

$$\begin{aligned} v &= v_1 + v_2 + \cdots + v_{n-1} + v_n \\ v_n &= v - (v_1 + v_2 + \cdots + v_{n-1}) \end{aligned} \tag{2.6}$$

In practice, each client keeps their own i-th share and distributes the other shares to the respective clients. This process is repeated for all clients, resulting in each client holding n shares, one share of their own data and $n-1$ shares from other clients. To reconstruct the aggregated value across all clients, each client performs a local summation of all the shares they have received, including their own share. Mathematically, if we denote the j-th share of the i-th client's value as $v_{i,j}$, then each client i computes:

$$S_i = \sum_{j=1}^{n} v_{j,i} \tag{2.7}$$

where S_i is the local sum computed by client (i). This sum includes the i-th share of their own data ($v_{i,i}$) and the i-th shares received from all other clients. The beauty of this approach lies in its aggregation property. When all clients compute their local sums S_i, the sum of these local sums equals the sum of all original values:

$$\sum_{i=1}^{n} S_i = \sum_{i=1}^{n} v_i \tag{2.8}$$

This property allows for secure aggregation of values across all participants without revealing individual contributions.

FL schemes like FedShare [29] and SCOTCH [61] employ additive secret sharing for secure aggregation. These protocols introduce intermediary servers to handle shares. Clients split their local model updates into n shares and send one share to each of the n aggregating servers. The servers then sum their shares and return the aggregated result, allowing clients to reconstruct the global model without revealing individual updates. In FedShare [29], each client creates n additive-secret shares of its locally trained model weights such that

the sum of these shares equals the original model weights. Each client sends one share to each of the n aggregator servers. Each server then computes a weighted sum of the shares it received from all clients. Finally, the servers send their weighted sums to a lead server, which computes the true aggregated model by summing up the weighted sums received from all servers and sends the aggregated model back to the clients. Conversely, SCOTCH [61] involves splitting the local model weights from each client into n additive secret shares. Each of the n shares is then sent to one of the n aggregator servers. The aggregator servers add up the shares they receive from all clients and divide by n to obtain their share of the federated average model. The servers send their shares back to the clients, who can reconstruct the full federated average model by summing the shares from all servers.

AddShare [11] simplifies this process by applying additive secret sharing without the need for intermediary servers. SecAgg [14] and its enhanced version SecAgg+ [8] enable secure FL through input vector masking and SMC. In these protocols, clients mask their input vectors with random pairwise masks shared via Diffie-Hellman key agreement and Shamir secret sharing. The aggregating server sums the masked vectors to compute an aggregate model update without learning individual inputs. If a client drops out, the server can reconstruct their masks from shared key shares to remove them from the aggregate. SecAgg+ improves upon SecAgg by using a sparse random communication graph, reducing the number of shares each client must manage from linear to logarithmic, thereby decreasing computation and communication overhead per client while maintaining security guarantees. Table 2.2 summarizes the various SMC techniques used in these protocols.

Table 2.2: Main FL Secret Sharing Algorithms from the literature.

Algorithm	Year	Sharing	Single Server	Multi-Server	Uses Percentage of Weights
SecAgg [14]	2017	Shamir	✓	×	×
SecAgg+ [8]	2020	Shamir	✓	×	×
Scotch [61]	2022	Additive	×	✓	×
FedShare [29]	2023	Additive	×	✓	×
AddShare (our proposal) [11]	2023	Additive	✓	×	×
AddShare+ (our proposal) [4]	2024	Additive	✓	×	✓

2.6 Open Security Issues in the Application of Federated Learning to Smart Farming

In FL, model training process is distributed across multiple devices or entities with each entity training their model locally using their own dataset. At no point is the raw private data from each entity shared which makes FL well-suited for smart farming scenarios involving multiple farms. No individual source needs to sacrifice competitive advantages or reveal protected intellectual property. FL thereby aligns well to the distributed yet data-rich information landscape in next-generation smart farming.

Several recent studies have explored the feasibility and value of using FL approaches in agricultural settings. Siniosoglou et al. [76] developed an animal welfare Long Short Term Memory (LSTM) model to predict future farm conditions through a FL method that leveraged agricultural time series-based datasets from some farms. By training the model on the collective data from multiple sites, they were able to achieve better predictive performance compared to models trained on any individual plantation's dataset alone. This showcases how FL enables collaborative ML development to produce superior models by sharing knowledge across distributed private datasets. A recent study by [2] proposed a FL framework for rice leaf disease classification that allows collaborative model development while preserving data privacy across clients. They implemented a lightweight deep learning architecture distributed across client devices and a central server which aggregated model updates. The system was tested on both Independent and Identically Distributed (IID) and non-IID datasets containing images of diseased rice leaves to simulate real-world data heterogeneity. Their FL approach achieved 99% accuracy on IID data and 95% accuracy on non-IID data, nearly matching baseline performance of a centralized EfficientNetB3 model trained on the full dataset. These initial results demonstrate FL's potential for plant disease diagnosis while keeping sensitive farm data decentralized. The authors highlight that their framework's ability to work with limited resource edge devices makes it accessible for widespread use, while preserving data privacy and security. However, further evaluation

is still needed regarding communication efficiency and cost.

While FL shows immense promise for collaborative data-driven FL application in smart farming, the exploration of privacy and security issues related to its use in this domain remains at early stages. A review of current literature uncovers little analysis or discussions of risks around potential misuse, leakage, or breaches with smart farming based FL systems thus far. However, insights from applications in other industries provide an informative perspective. A study [59] demonstrated that FL systems can unintentionally leak sensitive participant data and defenses against such inference attacks are currently inadequate, highlighting urgent need for constraints in learning and governance to address the risk that collaborative modeling may expose rather than protect private data. Wei et al. [89] also demonstrated the ability to reconstruct clients' private training data in FL systems by analyzing the parameter updates they share. Even though FL keeps raw data decentralized, the researchers showed that gradient or weight updates alone still leak enough information for extraction attacks. Through formal analysis and experiments, the authors highlighted multiple attack vectors that intrude on presumed client data privacy despite only intermediate model outputs being transmitted. They also surfaced that common efficiency techniques like gradient compression as well as typical FL hyperparameters can influence both attack feasibility and costs for adversaries. By systematically evaluating the various client privacy leakage threats possible strictly through inspection of shared model updates, this work underscores the inadequate protections for decentralized data in current FL approaches.

Although the data types and threat models certainly differ in smart farming, insights like these warrant more proactive analysis around potential attack surfaces early in design before real risks emerge downstream. Developing robust frameworks and technically secure system architectures suited to unique smart farming challenges can help preemptively safeguard both privacy and productivity goals as FL expands across the smart farming ecosystem.

2.7 Summary

Significant privacy risks persist in existing FL systems despite keeping raw data private. Recent work has shown model updates leaked through gradient inspection, membership inference attacks, and other vectors can unintentionally expose sensitive participant information. This underscores the inadequate protections for FL data in current approaches.

This chapter provided a comprehensive analysis of existing privacy-preserving approaches for curtailing these issues in FL using DP, HE, and SMC. The review also demonstrates growing efforts towards using SMC secret sharing schemes like Shamir's and additive secret sharing [96] to enable secure aggregation for privacy-preserving FL. Protocols like SecAgg [14] and SecAgg+ [8] have made progress in leveraging techniques like input vector masking and distributed randomness to facilitate joint computation on private data. However, all these prevailing methods exchange full model weight updates between clients and servers, incurring major efficiency costs for bandwidth, computation, and communication. Techniques like SCOTCH [61] and FedShare [29] predominantly rely on additional intermediary servers for share distribution and aggregation as well, further increasing infrastructure demands.

While the literature has validated FL's potential for collaborative modeling without compromising sensitive decentralized data, substantial barriers around efficiency and affordability must still be addressed for widespread practical adoption in areas such as smart farming where there has been little proactive analysis around potential attack surfaces and governance mechanisms. As the exploration of privacy and security issues related to FL's use in smart farming remains in the early stages, effective privacy preservation is imperative for FL to deliver on its potential benefits without compromising data sensitivity. There is a definitive need for communication-efficient and lightweight techniques optimized for the systemic constraints in emerging FL-powered smart farming ecosystems.To address this need, we present in the next chapter our proposed AddShare+ framework which uses an optimized selective additive secret sharing [96] approach. The AddShare+ methodology and an extensive empirical evaluation of its performance will be covered next.

Chapter 3

Efficient Additive Secret Sharing Solutions

3.1 Introduction

In this chapter, we present the details of AddShare and AddShare+, the two frameworks developed for privacy-preserving FL based on additive secret sharing. Addshare is a privacy-preserving framework designed to protect the confidentiality of local model updates during the FL aggregation process. In Addshare, each client splits its local model updates into multiple additive secret shares, which are then exchanged among clients. These clients locally aggregate the shares received and send the aggregated shares to the central FL server, allowing the aggregation of all weight updates into a global model without revealing individual client updates (weights). Addshare+ builds upon the foundations of Addshare by introducing enhancements that improve security, scalability, and efficiency. It selectively shares model weights and incorporates faster cryptographic techniques and optimizations to further safeguard the aggregation process and mitigate potential vulnerabilities.

This chapter is structured to provide a comprehensive understanding of the proposed solutions, Addshare and Addshare+. It begins with the general architecture of FL, followed

by detailed sections on the design and implementation of Addshare and Addshare+. The chapter also includes a discussion on the security guarantees provided by AddShare+ as well as the impacts that additive secret sharing has on the model's accuracy. The key points discussed in each section are summarized at the end, highlighting the unique contributions and significance of the proposed methods.

3.2 Federated Learning Architecture

FL architecture is composed of several key components that work together to enable decentralized machine learning while preserving data privacy. At the core of FL are the clients and the central aggregating server.

3.2.1 Clients

Clients are the individual devices or entities that participate in the FL training process by contributing their local data and computational resources. These clients, also referred to as participants, nodes, or local devices, can include smartphones, IoT devices, or computers that generate and process data locally. Each client trains a local model on its private dataset and periodically sends the model updates to the server.

3.2.2 Central Server

The central server in FL plays a crucial role as the central coordinating entity that manages the FL process. Often referred to as the central server, aggregating server, aggregator, or coordinator, it is responsible for collecting and aggregating the model updates from all participating clients. The aggregation server performs aggregation to ensure that individual updates are computed into one global model. This server not only aggregates the updates but also distributes the updated global model back to the clients, ensuring continuous improvement of the model across all devices. The global model server, as it is

sometimes called, maintains the global model and orchestrates the entire training process by facilitating communication between clients and managing the iterative update process. The server's role is critical in ensuring the efficiency, and accuracy of the FL prcoess.

3.3 AddShare

AddShare [11] consists of four main components:

1. Additive Secret Sharing
2. Group Formation
3. Encryption
4. Federated Averaging

The key idea of AddShare is to avoid any client and server knowing any client's local weights and to defend against an attacker listening to all communications to recover the client weights. The four components work together in a coordinated workflow to achieve this. Additive secret sharing forms the foundation by allowing clients to split their model updates into shares that individually reveal no information. The group formation component organizes clients into smaller groups to optimize communication efficiency, ensuring that secret sharing occurs within manageable subsets rather than across all participants. Encryption, implemented through RSA, secures the transmission of these shares between clients within their assigned groups. Finally, FedAvg [57] aggregates the reconstructed model updates from all groups into a global model. This architecture as illustrated in Figure 3.2 allows AddShare to maintain privacy throughout the entire training process from the initial splitting of model updates to the final model aggregation while managing computational and communication overhead through strategic grouping.

3.3.1 Additive Secret Sharing Algorithm

In secret sharing [73], a secret value x is divided into n shares x_1, x_1, ... x_n. The shares are chosen in such a way that, when they are combined, they reveal the original secret value, but any subset of shares less than a threshold k does not reveal any information about the secret. This method can be thought of as a (k,n)-threshold secret sharing scheme, where k is the threshold number of shares necessary for secret retrieval and n is the number of shares. We employ additive secret sharing [21], a variant of secret sharing whereby all the shares are necessary for secret retrieval. This method can also be thought of as an (n,n)-threshold secret sharing scheme.

This SMC technique is applied as a privacy preservation protocol in our FL framework. After each client has trained a model with their local data, instead of simply sharing their original weights with the aggregating server, each client splits the local model weights W into n shares as w_i, where n is equal to the number of participating clients in the sharing process. Algorithm 3.1 provides the pseudo-code for the steps involved in the additive secret sharing procedure of model weights. These weight shares are exchanged among the clients. This ensures that each client receives a share of the model weights of other clients, and no client has access to the original model weights of other clients. Each client assembles the model weights using their local share and the $n-1$ shares received from the other clients. The reconstructed model weights of client i, which we represent as W' are sent to the aggregating server, where they are aggregated using FedAvg [57]. Where:

$$W = w_1 + w_2 + \cdots + w_{n-1} + w_n \tag{3.1}$$

$$W' = \sum_{i=1}^{n} w_i \tag{3.2}$$

Finally, all clients receive the updated model, μ from the server. Figure 3.1 displays this process for 3 clients. We must highlight that the example in Figure 3.1 assumes all clients participate in one sharing process, i.e., the total number of clients is the same as the number

Algorithm 3.1 Generate n shares of a multidimensional array

Input: n: number of clients entering the sharing process; arr, multidimensional array of shape $(m_1, m_2, \ldots, m_k)$ with the client's local weights.

Output: $shares$: list of size n containing arrays of shape $(m_1, m_2, \ldots, m_k)$ such that $\sum_{k=1}^{n} shares[k] = arr$.

1: $rand\_arr \leftarrow$ array of $n-1$ randomly generated arrays of shape $(m_1, m_2, \ldots, m_k)$ ▷ where each element of $rand\_arr[i]$ is drawn from a uniform distribution between $-|arr[i]|$ and $|arr[i]|$.
2: $nth\_share \leftarrow arr - \sum_{i=1}^{n-1} rand\_arr[i]$
3: $shares \leftarrow (rand\_arr, nth\_share)$
4: Return $shares$

Algorithm 3.2 Reconstruct a multidimensional array from its shares

Input: $shares$, an array of shape $(n, m_1, m_2, \ldots, m_k)$ such that $\sum_{i=1}^{n} shares[i] = arr$.

Output: The original array arr of shape $(m_1, m_2, \ldots, m_k)$.

1: $arr \leftarrow$ an array of shape $(m_1, m_2, \ldots, m_k)$ initialized to zero.
2: **for** $i = 1$ to n **do**
3: $\quad arr \leftarrow arr + shares[i]$
4: **end for**
5: Return arr

of clients sharing the weights. In the group component proposed, we discuss a strategy where a total of n clients is considered but the sharing process is carried out in groups where only k clients participate.

Algorithms 3.1 and 3.2 form the core of AddShare's secret sharing mechanism. Algorithm 3.1 handles the generation of shares by taking an input array of model weights and splitting it into n shares, where n represents the number of participating clients. The algorithm first generates n-1 random arrays matching the shape of the input weights, with values uniformly distributed between the negative and positive of each weight value. The nth share is then computed by subtracting the sum of these random arrays from the original weights, ensuring the shares sum to the original values. Algorithm 3.2 complements

this by performing the reconstruction process, taking the distributed shares as input and systematically adding them together to recover the original weight values. This pair of algorithms ensures that while individual shares reveal no information about the original weights, the complete set of shares can perfectly reconstruct the model parameters.

3.3.2 Group Formation

The group formation component in AddShare organizes clients into smaller, manageable groups to optimize communication efficiency and maintain system scalability. Before each FL round, the server executes a deterministic algorithm that randomly shuffles the list of participating clients and systematically constructs groups of size k. The selection of the group size k, which is discussed in detail in Section 3.8.1.4, is a critical parameter that balances communication overhead with privacy guarantees as smaller groups reduce communication costs but must be carefully sized to maintain security. The algorithm employs a cyclic iterator to ensure fair distribution of clients across groups while preventing duplicate assignments within the same group. Each client is guaranteed assignment to at least one group, with the algorithm continuing until all clients are included in the grouping structure. Algorithm 3.3 provides a pseudocode of the algorithm.

Initialization and Shuffling:
The server begins by receiving the list of participating clients. This list is randomly shuffled to ensure that the group formation is not biased by the original order of clients. Shuffling introduces randomness, which is crucial for enhancing the privacy aspects of the FL process.

Creating a Repeating Iterator:
An iterator is created over the shuffled list of clients. This iterator is cyclic, meaning that once the end of the list is reached, it starts again from the beginning. This cycle ensures that the group formation process can continue seamlessly until all clients are included in at least one group.

Group Selection and Tracking:

The algorithm iteratively selects groups of clients by taking the next k clients from the cyclic iterator. A set is maintained to keep track of which clients have been selected. This set helps ensure that each client is eventually included in a group. Selected groups are appended to a list of selected groups, forming the final output of the group formation process.

Completion:

The process continues until the set of selected clients includes all clients, ensuring comprehensive participation. The resulting list of groups, where each group consists of k clients, is returned.

Algorithm 3.3 Group Formation Algorithm

Input: List of clients, group size k
Output: List of groups selected_groups

```
shuffled_clients ← shuffle(clients)                          ▷ Shuffle the list of clients
client_iterator ← cycle(shuffled_clients)   ▷ Create a cyclic iterator over the shuffled clients
selected_clients ← {}                       ▷ Initialize an empty set to track selected clients
selected_groups ← [ ]                       ▷ Initialize an empty list to store selected groups
while |selected_clients| < |clients| do
    group ← next_group(client_iterator, k)                   ▷ Select the next group of clients
    selected_clients ← selected_clients ∪ group              ▷ Update the set of selected clients
    selected_groups ← selected_groups ∪ [group]  ▷ Append the selected group to the list of selected groups
end while
return selected_groups
```

Practical Considerations of Group Formation

1. By shuffling the clients list before forming groups, the algorithm ensures that group composition varies in each FL round, enhancing security by reducing the likelihood of pattern recognition or targeted attacks.

2. The size of the groups, k, is a critical parameter that balances communication efficiency and privacy. Larger group sizes may reduce the communication overhead but could introduce more complexity in managing the secret shares.

3. Although the process is deterministic in nature (given the same initial list and random seed, the output will be consistent), the incorporation of random shuffling makes the group formation dynamic and adaptive to different client lists in each round.

3.3.3 Encryption

The encryption component is a pivotal element in AddShare that ensures the security and privacy of FL processes. Asymmetric encryption, is employed to provide a robust layer of protection. Each client in the system generates a unique pair of public and private keys, which are securely stored and managed. This design choice addresses several critical security concerns. Notably, in the event of a private key being compromised, the breach is isolated to the affected client, with the remaining clients' private keys remaining secure. This containment of risk is a significant advantage over symmetric encryption, where a single key compromise could jeopardize the entire system's security. Moreover, asymmetric encryption ensures that only the intended recipient can decrypt and access the shared model weights, effectively preventing eavesdropping and unauthorized access during transmission. This capability is crucial for maintaining the confidentiality and integrity of the data shared in the FL process. Additionally, the use of public-key cryptography simplifies key management and secure communication among multiple clients. Each client encrypts data with the recipient's public key and decrypts received data with their private key, aligning seamlessly with the distributed nature of FL. This method simplifies key exchange and management compared to symmetric encryption systems, which require secure key distribution channels.

AddShare uses the RSA [71] encryption algorithm to secure communications between clients. RSA, a widely adopted asymmetric encryption method, leverages the mathematical

properties of prime numbers and modular arithmetic to secure communication and data encryption. A public key is used for encryption, and a private key is used for decryption. The strength of RSA resides in the computational complexity of factoring huge composite numbers into their prime components, making it a cornerstone of modern encryption. When employed in AddShare, client i encrypts the shares that will be sent to client j using client j's public key. This allows client j to receive the shares, decrypt them, and aggregate them before sending them to the central server.

The key generation process of RSA encryption involves the following steps:

1. **Prime Number Selection**: Selection of two large prime numbers, p and q.
2. **Modulus Computation**: Computation of the modulus n as the product of p and q.
3. **Computation of Euler's totient function**:

$$\varphi(n) = (p-1)(q-1) \tag{3.3}$$

4. **Public Key Selection**: Selection of an integer e such that;

$$1 < e < \varphi(n) \tag{3.4}$$

$$\gcd(e, \varphi(n)) = 1 \tag{3.5}$$

   thus e that is coprime with $\varphi(n)$

5. **Private Key Selection**: The private key d is computed as the multiplicative inverse of e modulo $\varphi(n)$.

To encrypt the *shares* using RSA encryption, we follow two steps:

1. Public Key Acquisition: the recipient's public key (e, n) is obtained

2. Ciphertext Computation: ciphertext, C is computed using:

$$C = share^{e}(modn) \tag{3.6}$$

To decrypt the ciphertext C and retrieve the original message, two steps must be completed:

1. Private Key Usage: the private key (d, n) is employed
2. Plaintext Computation: Plaintext, $shares$ is computed using

$$shares = C^{d}(modn) \tag{3.7}$$

To mitigate the threat of eavesdroppers, RSA encryption is employed as a safeguard. By encrypting the additive share sent to each client using the client's public key, RSA ensures that only clients possessing the corresponding private key can decrypt and access that model share, preserving the confidentiality and integrity of the FL process. In this framework, client keys are generated offline, and each client securely stores its private keys while sharing the corresponding public keys with other clients prior to the FL process.

3.3.4 Federated Averaging

FedAvg [58] is a fundamental algorithm in FL. The process involves clients periodically communicating their model updates to a central server, which aggregates these updates to form a global model. FedAvg [57] ensures that the global model benefits from the data and computations performed locally by each participating client while maintaining data privacy.

Overview of the FedAvg Process

1. **Local Training**: Each client i trains a local model using its private dataset. Instead of sharing raw data, the client generates model updates, represented as weight parameters

$\mathbf{W'}_i$

2. **Additive Secret Sharing and Exchange**: To preserve privacy, each client i divides its local model weights $\mathbf{W'}_i$ into i additive shares using the additive secret sharing algorithm. The client retains one share and sends the remaining $n-1$ shares to the other participating clients. Each client now possesses a complete set of shares from all clients, enabling the reconstruction of the original weights locally without revealing them to the central server.

3. **Reconstruction and Submission**: Each client aggregates model weights using its own share and the shares received from other clients. This reconstructed weight, denoted as $\mathbf{W'}_i$ is sent to the central server.

4. **Global Aggregation**: The server aggregates the updates from all clients by computing a weighted average of the received model parameters. The weight assigned to each client's update is proportional to the size of the client's local dataset. This ensures that clients with more data have a proportionally greater influence on the global model update. Mathematically, the new global model parameters μ are calculated as follows:

$$\mu = \sum_{i=1}^{n} \frac{d_i}{d} \mathbf{W'}_i \tag{3.8}$$

where d_i is the number of data points at client i, and d is the total number of data points across all clients.

The aggregation step is pivotal in FL as it balances the contributions from all clients, enabling the global model to be trained effectively while preserving the privacy of each client's local data. This collaborative approach leverages diverse datasets to enhance the model's performance across different data distributions and environments.

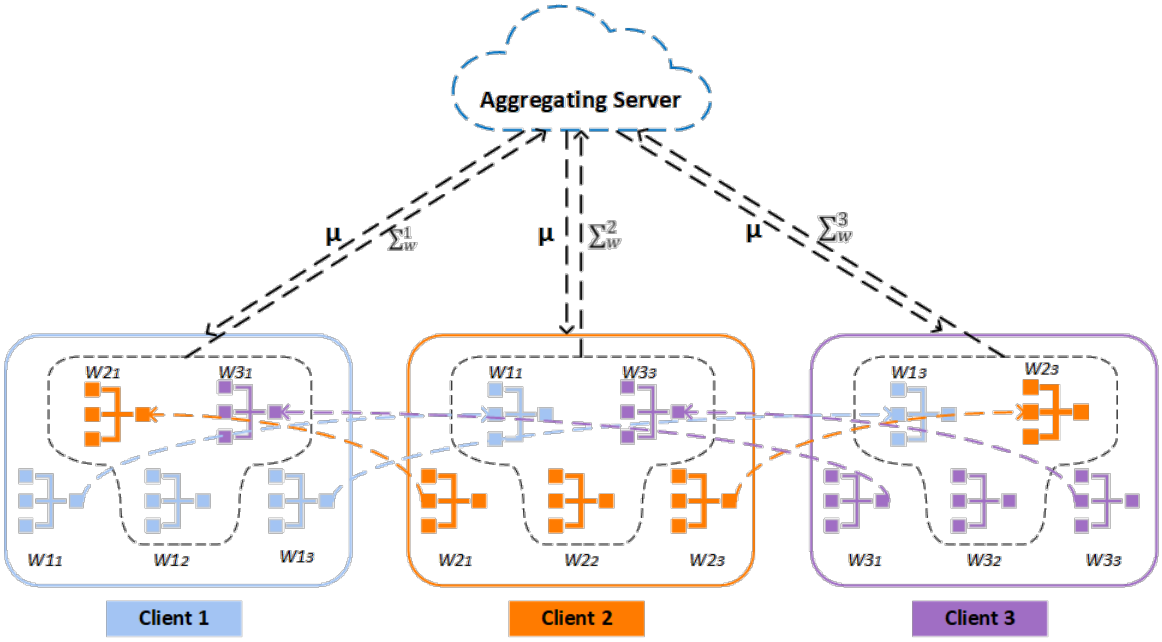

Figure 3.1: Overview of the proposed AddShare Algorithm using additive secret sharing for model weights sharing in FL. (Considering a 3-client FL setup, Client 1 builds 3 shares from its local model weights ($W11$, $W12$, and $W13$), keeps one share ($W11$), and sends $W12$ and $W13$ to clients 2 and 3, while also receiving one share from the other clients. The shares are aggregated and sent to the central server.)

3.3.5 AddShare Variants

3.3.5.1 Additive Secret Sharing Federated Learning

This variant only applies additive secret sharing in the FL process. No encryption is used and no groups of clients are formed. For the n clients, each $i-th$ client splits their model weights into n additive shares using Algorithm 3.1. Each $i-th$ client keeps one share of their n additive shares and exchanges the $n-1$ shares left with the other $n-1$ clients without encrypting the weights. At the end of this exchange process, each $i-th$ client reconstructs the model weights using its local share and the $n-1$ shares received from the other clients. The reconstructed model weights are sent to the central server for aggregation

into a global model.

3.3.5.2 Encrypted additive secret sharing Federated Learning

Building upon the previous variant, in this case, we combine the benefits of both encryption and additive secret sharing. The same additive secret sharing process is followed but the clients encrypt the shares before transmitting them to the other clients using each client's public key. This ensures the weight's privacy during the FL process.

3.3.5.3 Additive Secret Sharing Federated Learning with Groups

This variant introduces the concept of subgroup-based additive secret-sharing FL. Instead of performing additive secret sharing among all n clients, participants are divided into smaller subgroups of size k (e.g., subgroups of 3, 5, and 10) executed by the aggregating server. The additive secret sharing is performed only within each subgroup. This decreases the number of communications as instead of building n shares to send to $n-1$ clients, only k shares are needed and $k-1$ shares are sent, this is discussed in detail in Section 3.8.1.4. The resulting model updates are then sent to the central server for aggregation into a global model. Figure 3.2 illustrates this.

3.3.5.4 Encrypted additive secret sharing Federated Learning with groups

Extending the previous variant, this approach combines all components of AddShare including subgroup-based additive secret-sharing FL with encryption. Clients encrypt their additive shares and transmit them securely within their subgroups.

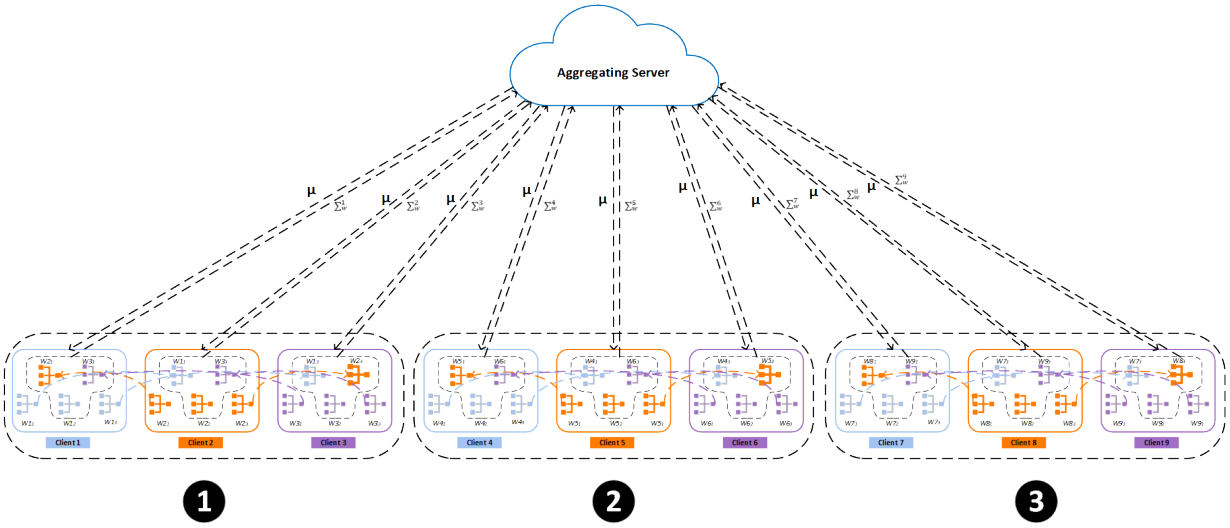

Figure 3.2: Addshare with groups of 3

3.4 AddShare+: Improvements

While AddShare [11] provides privacy preservation in FL, it incurs significant communication overhead from exchanging full model weight updates between clients as well as the application of RSA encryption as discussed in Chapter 4. Moreover, this solution involves an accuracy loss of the models due to the additive secret sharing process. To address these limitations, we propose selective weight sharing, i.e., selectively sharing only a subset of model weights during the aggregated model training. This reduces the amount of model information exchanged between clients, improving communication efficiency and scalability. At the same time, sharing partial weight updates provides privacy benefits by making the reconstruction of the full client models more difficult, as different model weights are being modified at each FL round. Finally, sharing only a part of the clients' weights allows to minimize the performance loss associated with using additive secret sharing as discussed in Section 3.8.3.

Let us suppose that each client has a given neural network architecture and will train a model M with its local dataset. Let the neural network model M consist of L layers, with

layer l containing N_l nodes. Let w_{lj} represent the weight for node j in layer l. For selected layers, additive secret sharing is performed on the layer weight tensors before exchange.

Each client shares a percentage p of the N_l nodes in each layer l. This subset is denoted by S_l. The remaining nodes U_l are unshared. With

$$|S_l| = p * N_l \tag{3.9}$$

$$U_l = N_l - S_l \tag{3.10}$$

For example, for a layer with 1000 nodes, with $p = 50\%$, the client would share 500 nodes in that layer. The nodes to be shared are selected by the central server before each round, and this information is sent to each client. This procedure is repeated at each FL round, i.e., at each round the clients will share different weights.

For each node $j \in S_l$ to be shared, the client generates k additive shares of w_{lj} using a linear additive secret sharing scheme: w_{lj1}, w_{lj2}, ..., w_{ljk} where:

$$w_{lj} = w_{lj1} + w_{lj2} + ... + w_{ljk}. \tag{3.11}$$

The client keeps one share w_{lj1} locally and distributes shares w_{lj2}, ..., w_{ljk} among the other $k-1$ clients. For unshared nodes $i \in U_l$, the original weight w_{li} is kept on the client. Each client receives $k-1$ shares and reconstructs (or aggregates) the corresponding shares received for each weight of each layer as shown in Equation 3.12.

$$w_{lj} = w_{lj1} + \sum_{k \neq 1} w_{ljk}, \forall j \in S_l \tag{3.12}$$

The reconstructed shared weights and original unshared weights(U_l), ΔW_i comprise the update submitted to the central server for aggregation. The central server aggregates all δN received from each client and sends an FL update ΔW_{agg} back to all clients. This selective sharing addresses two key issues: (i) reduces communication costs and (ii) minimizes

accuracy loss. The architecture of AddShare+ is illustrated in Figure 3.3 below.

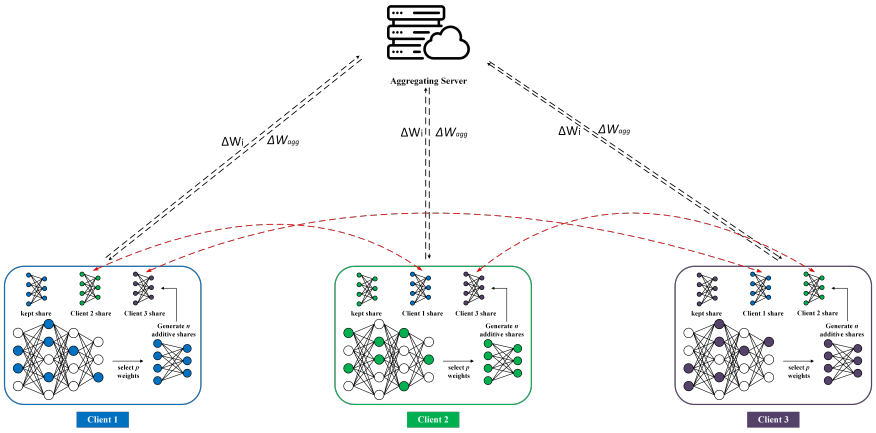

Figure 3.3: An illustration of the AddShare+ protocol with k=3 participants. Each participant generates k additive shares for a subset of their model parameters. They distribute k-1 shares to others, keeping one. Participants reconstruct the shared weights using received shares and their own. Both reconstructed shared parameters and original unshared parameters are sent to a central aggregator to update the global model. This partial weight sharing approach reduces communication overhead while preserving privacy.

3.5 AddShare+ with Elliptic Curve Integrated Encryption Scheme

ECIES serves as a critical enhancement in AddShare+, replacing the RSA encryption used in the original AddShare framework. When clients perform selective weight sharing in AddShare+, they encrypt their weight shares using ECIES before transmission to other group members. This integration brings several advantages. The shorter key lengths in ECIES reduce the communication overhead during share exchange, complementing AddShare+'s selective weight sharing strategy. In addition, the hybrid encryption approach of ECIES provides faster encryption/decryption operations during the frequent weight exchanges between clients, improving the overall training efficiency. Lastly, the

elliptic curve cryptography maintains strong security guarantees while better aligning with AddShare+'s focus on reduced computational overhead. These benefits directly support AddShare+'s core objective of optimizing the privacy-efficiency trade-off in FL. The details of ECIES are presented below.

ECIES is a hybrid encryption scheme that combines elliptic curve asymmetric (public-key) cryptography for key exchange with symmetric encryption for encrypting data [34, 35]. Each user generates a public-private key pair using an elliptic curve algorithm that relies on the algebraic properties of elliptic curves over finite fields. This method allows security equivalent to public key systems like RSA but with much shorter keys resulting in improved efficiency. In ECIES, the sender uses the recipient's public key along with a freshly generated ephemeral key pair to derive a shared secret. This shared secret is then fed through a Key Derivation Function (KDF) to derive symmetric encryption keys unique to that message. The actual plaintext is encrypted with the symmetric key using an algorithm like Advanced Encryption Standard (AES) [66], and the ciphertext is transmitted alongside the ephemeral public key to the recipient. The recipient uses their private key with the received ephemeral public key to derive the same shared secret and decryption key in order to decrypt the message. The elliptic curve key exchange provides semantic security while the symmetric encryption efficiently provides confidentiality. Thus, ECIES is able to achieve the security of a public-key cryptosystem along with the speed of a symmetric cryptosystem. A more formalized description is given below.

Mathematically, let G be a cyclic group of prime order q generated by a point P on an elliptic curve defined over a finite field F_p, denoted as $E(F_p)$. Let n be the bit-length of the order of G.

Key Generation:
Each user generates a public-private key pair by selecting a random integer d in the interval $[1, q-1]$ as the private key and compute the corresponding public key:

$$Q = dP \tag{3.13}$$

Encryption:
(a) Key Exchange: The sender selects a random integer k in the interval $[1, q-1]$ and computes the ephemeral public key:

$$K = kP \tag{3.14}$$

The shared secret is computed as:

$$S = kQ \tag{3.15}$$

(b) Key Derivation: The shared secret S is passed through a KDF to derive symmetric encryption keys unique to the message, denoted as K_{enc} and K_{mac}
(c) Symmetric Encryption: The plaintext M is encrypted using a symmetric encryption algorithm like AES with the derived encryption key K_{enc}, resulting in ciphertext C.
(d) Transmission: Transmit the ciphertext C along with the ephemeral public key K to the recipient.

Decryption:
(a) Shared Secret Derivation: The recipient computes the shared secret $S' = dK$, using their private key d and the received ephemeral public key K.
(b) Key Derivation: The shared secret S' is passed through the same KDF to derive the symmetric encryption keys K'_{enc} and K'_{mac}.
(c) Symmetric Decryption: The recipient decrypts the ciphertext C using the symmetric decryption key K'_{enc} to retrieve the original plaintext message M.

3.6 AddShare+ with Groups

Similar to Addshare [11], smaller groups of participants are formed for secret sharing rather than sharing secrets among all clients. In this subgroup-based approach, the aggregating server forms subgroups of clients of size k, such as 3, 5, or 10 members each. Additive secret

sharing occurs only within each local subgroup prior to sending masked model updates to the central server. Creating shares for distribution to $k-1$ subgroup members rather than $n-1$ total clients reduces communication overhead. After additive sharing locally, subgroups transmit their obscured model updates to the aggregation server which combines them into an updated global model. Grouping clients alleviates privacy risks from sharing intermediate model information across the entire client population. Another alternative form of group formation explored in AddShare+ is the client-side group formation. In this variant, each client employs a decentralized and stochastic group formation protocol, whereby they independently sample from the set of all participating clients, excluding themselves, to construct a group of size k - 1 according to a uniform random distribution. This randomized and distributed approach to group construction mitigates the potential for a central server to manipulate group formation, thereby fortifying the privacy guarantees against a malicious server.

3.7 Security Guarantees of AddShare+

The AddShare+ provides robust security guarantees through a combination of information-theoretic security, threshold security, and computational security. This section presents an analysis of these security aspects and their quantifiable impacts on the overall security of the framework.

3.7.1 Information-Theoretic Security

AddShare+ achieves information-theoretic security through its additive secret sharing mechanism, ensuring that individual model updates remain private regardless of an adversary's computational capabilities. Unlike traditional cryptographic approaches that rely on computational hardness assumptions, additive secret sharing provides unconditional security guarantees that are mathematically provable.

Perfect Secrecy Proof

Theorem 3.1 (Perfect Secrecy of AddShare+)**.** *Let W be a client's model weights, and $\{S_1, S_2, ..., S_n\}$ be the n shares generated through additive secret sharing. For any subset of shares $S' \subset \{S_1, S_2, ..., S_n\}$ where $|S'| < n$, the mutual information $I(W; S') = 0$.*

Proof. We prove this through the following steps:

1. For a given weight tensor W, each share S_i (except the last) is generated independently from a uniform distribution over the domain of W:

$$S_i \sim \mathcal{U}(-|W|, |W|), \quad i \in \{1, ..., n-1\}$$

2. The final share S_n is computed deterministically:

$$S_n = W - \sum_{i=1}^{n-1} S_i$$

3. For any subset S' containing $k < n$ shares:

   - The joint distribution of these shares is uniform over their domain
   - For any possible value of W, all possible combinations of share values in S' are equally likely
   - Therefore, $P(W|S') = P(W)$

4. This independence implies zero mutual information:

$$I(W; S') = H(W) - H(W|S') = 0$$

   where $H(\cdot)$ denotes Shannon entropy

□

In an FL setting, an adversary poses a potential threat by attempting a targeted reconstruction attack on the model weight W_i of a specific client. To counter this, each client employs additive secret sharing on their local model weights, splitting it into n shares where $\{S_1, S_2, ..., S_{n-1}\}$ of these shares are then exchanged between clients. Each clients aggregates the shares received from other clients locally. Local aggregation results in an updated model weight represented as:

$$\Delta W_i = S_i^n + \sum_{j=1}^{n-1} S^j. \tag{3.16}$$

where S_i^n is the share each client keeps after splitting its own weights and S^j are the shares it received from other clients. Participants transmit their model updates ΔW_i to the central server which is aggregated as global model update:

$$\Delta W_{\text{agg}} = \sum_{i=1}^{n} \Delta W_i \tag{3.17}$$

Privacy Guarantees Against Specific Threats

Based on our threat model in Section 2.3, AddShare+ through additive secret sharing provides the following concrete security guarantees:

1. **Honest-but-Curious Server Protection:** For a server with access to locally aggregated updates $\{\Delta W_i\}$, we guarantee:

   $$I(W_i; \Delta W_i) = 0$$

   where W_i represents a client's true weight and ΔW_i is what the server observes which is made up of shares from other clients and only 1 share of ΔW_i.

2. **Collusion Resistance:** For any coalition $\mathcal{C}$ of $t < n$ participants:

$$H(W|\mathcal{C}) = H(W)$$

where $H(W|\mathcal{C})$ represents the entropy of weight W given the coalition's knowledge.

3. **External Adversary Protection:** For any eavesdropper intercepting up to $k < n$ shares:

$$P(W|S_1, ..., S_k) = P(W)$$

ensuring that partial share knowledge provides no information about W.

Information theoretic security guarantees that ΔW_i is independent of the original W_i of each client and the Mutual Information (MI) between them is 0. An honest but curious server lacks the necessary information to reconstruct W_i since ΔW_i contains only 1 share of W_i. Lacking access to all subsets of a participant's model weights W_i, the central server is incapable of deducing W_i in order to lunch a reconstruction attack and recover a participants P_i data. These guarantees hold regardless of computational power, ensuring robust privacy protection against both current and future adversarial capabilities. The security properties are maintained even if the adversary possesses quantum computing capabilities, as they rely on information-theoretic principles rather than computational hardness assumptions.

3.7.2 Threshold Security

AddShare+ uses a (n, n)-threshold secret sharing scheme, meaning all n shares are required to reconstruct the original secret. In the (n, n)-threshold scheme, an adversary who obtains $k < n$ shares should gain no information about the secret. Formally, let $\mathcal{A}$ be an adversary who has obtained a subset of shares $S_k = \{s_1, s_2, ..., s_k\}$ where $k < n$. The probability of successfully reconstructing the secret W can be expressed as:

$$P(\text{reconstruct } W|S_k) = 0, \forall k < n \tag{3.18}$$

where P denotes probability. To prove this guarantee, consider an adversary attempting partial reconstruction with k shares. Let $\mathcal{W}$ be the space of all possible secrets and $\mathcal{S}_k$ be the space of all possible combinations of k shares. For any secret $w \in \mathcal{W}$:

$$P(W = w|S_k) = P(W = w) \tag{3.19}$$

This equality demonstrates that knowledge of k shares provides no additional information about the secret, maintaining perfect secrecy against partial reconstruction attempts. With proper implementation, reconstructing the secret S with fewer than n shares is computationally infeasible. The security of this threshold property can be quantified for b-bit numbers, an attacker would need to test $2^{b(n-k)}$ possibilities to recover the secret using only k shares, where $k < n$. This computational barrier provides concrete security guarantees that align with information-theoretic security principles [9]. In other words, it's impossible to reconstruct the secret unless you have all n shares.

3.7.3 Computational Security

AddShare+ incorporates ECIES to provide computational security for the weight shares exchanged among clients. ECIES combines the security of elliptic curve cryptography with the efficiency of symmetric key encryption, providing robust protection against attacks relying on computational power. The computational security of ECIES is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). The security strength can be quantified by the key length of l bits, where the work factor W for a brute-force attack is approximately:

$$W \approx 2^l \text{ operations} \tag{3.20}$$

For instance, with a 256-bit key used in AddShare+, $W \approx 2^{256}$ operations is required, which

is computationally infeasible with current and foreseeable technology. This ensures that even if an attacker intercepts the encrypted shares, decrypting them without the private key would require an impractically large amount of computational resources.

3.7.4 Combined Security Guarantees

The combined security guarantees for AddShare+ integrates information-theoretic security, threshold security, and computational security into a cohesive model that quantifies the overall security of the framework which provides a comprehensive evaluation of the security guarantees by considering how these three components collectively protect against various attacks and vulnerabilities. This section presents a comprehensive framework for analyzing and quantifying the overall security of the system, including degradation analysis under various attack scenarios.

Let $\Omega(t)$ represent the overall security state of the system at time t. We define $\Omega(t)$ as a function of three primary security components:

$$\Omega(t) = \alpha S_i(t) + \beta S_t(t) + \gamma S_c(t) \tag{3.21}$$

where:

- $S_i(t)$: Information-theoretic security score
- $S_t(t)$: Threshold security score
- $S_c(t)$: Computational security score
- α, β, γ: Weight factors where $\alpha + \beta + \gamma = 1$

Information-theoretic Security Score (S_i)

Information-theoretic security in AddShare+ is represented by the MI I between the original secret S and any subset of shares S^i. The ideal scenario is that any subset of shares smaller than n reveals no information about the original data.

$$S_i = \begin{cases} 100 & \text{if } I(S; S_1, S_2, \ldots, S_k) = 0 \quad \forall k < n \\ 0 & \text{if } I(S; S_1, S_2, \ldots, S_n) > 0 \end{cases} \tag{3.22}$$

In this context, $S_i = 100$ indicates perfect information-theoretic security, meaning the shares provide zero information to an adversary.

Partial Score Adjustments: If the information-theoretic security is not perfect, we can introduce a penalty for information leakage:

$$S_i = 100 - \epsilon \times \frac{I(S; S_1, S_2, \ldots, S_k)}{H(S)} \quad \text{where } \epsilon \text{ is a sensitivity factor} \tag{3.23}$$

This formula adjusts S_i based on the proportion of information revealed relative to the total information content $H(S)$.

For a perfect security where $S_i = 100$, lets consider a FL system with three clients sharing the weight $w = 0.75$ of one client. Using additive secret sharing, w is split into three shares: $s_1 = 2.41$, $s_2 = -1.83$, and $s_3 = 0.17$, such that $w = s_1 + s_2 + s_3 = 0.75$. Since these shares are generated using random values uniformly distributed over $\mathbb{R}$, obtaining any subset of shares provides zero information about w. With sensitivity factor $\epsilon = 100$,

$$\begin{aligned} S_i &= 100 - 100 \times \frac{I(S; S_1, S_2, \ldots, S_k)}{H(w)} \\ &= 100 - 100 \times \frac{0}{H(w)} \\ &= 100 \end{aligned} \tag{3.24}$$

In contrast, lets consider an imperfect security where $S_i < 100$ in which an FL system's neural network weights must be constrained between -1 and 1 for model stability. If $w = 0.75$ is split into shares and an adversary obtains $s_1 = 0.9$, they can deduce that w must be positive since the remaining shares must sum with s_1 to produce a value in [-1,1]. This partial information leakage results in $I(w; s_1) > 0$, leading to:

$$\begin{aligned} S_i &= 100 - 100 \times \frac{I(S; S_1, S_2, \ldots, S_n)}{H(w)} \\ &= 100 - 100 \times \frac{H(w)}{H(w)} = 0 \\ &= 100 \end{aligned} \tag{3.25}$$

Threshold Security Score (S_t)

Threshold security quantifies the ability of the system to prevent data reconstruction unless all n shares are available. This is represented as:

$$S_t = \begin{cases} 100 & \text{if } P(\text{reconstruct } S \mid k < n) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

If threshold security is not perfectly met, the score can be adjusted to reflect the probability of reconstruction given fewer than n shares:

$$S_t = 100 - \lambda \times P(\text{reconstruct } S \mid k < n) \quad \text{where } \lambda \text{ is a weighting factor} \tag{3.27}$$

This penalty function reflects the probability of reconstruction based on the number of shares available.

Consider a system with $n = 5$ clients sharing a neural network layer with 1000 parameters. For each parameter w_i, additive secret sharing creates 5 shares: $w_i = s_{i1}+s_{i2}+s_{i3}+s_{i4}+s_{i5}$. If 3 clients attempt reconstruction: $\hat{w}_i = s_{i1} + s_{i2} + s_{i3}$, the reconstruction error $E = |w_i - \hat{w}_i| = |s_{i4}+s_{i5}|$ represents missing shares. Since s_{i4} and s_{i5} are random in $[-|w_i|, |w_i|]$, their sum is independent of w_i, giving $P(\hat{w}_i = w_i) = 0$ and thus $S_t = 100$.

Computational Security (S_c)

Computational security is defined based on the difficulty of breaking the encryption scheme used for protecting the shares. It is measured by the work factor W, the computational effort required to break the encryption:

$$W = 2^l \tag{3.28}$$

where l is the key length in bits. We normalize S_c as:

$$S_c = \frac{W}{W_{\max}} \times 100 \tag{3.29}$$

Here, $W_{\max}$ is the maximum possible work factor given the current computational limits (e.g., a 256-bit key length).

Practical Adjustment: If the encryption strength is lower than the desired level, a penalty factor δ based on the key length can be introduced:

$$S_c = 100 - \delta \times \left(\frac{W_{\min}}{W} \right) \tag{3.30}$$

This formula penalizes S_c if the work factor W is significantly lower than the required minimum $W_{\min}$.

Combined Security Score

A Combined Security Score (CSS) can be computed as a weighted aggregation of S_i, S_t, and S_c. The CSS can be defined using the following formula:

$$\text{CSS} = \alpha S_i + \beta S_t + \gamma S_c \tag{3.31}$$

where α, β, and γ are the weights assigned to each security component based on their relative importance.

- **Weight α:** Reflects the importance of information-theoretic security, crucial when dealing with highly sensitive data.
- **Weight β:** Reflects the importance of threshold security, ensuring that no partial set of clients can reconstruct the data.
- **Weight γ:** Reflects the importance of computational security, relevant for protecting against external attackers who may intercept data.

These weights can be chosen such that $\alpha+\beta+\gamma = 1$. For sensitive applications, a higher weight might be given to information-theoretic security (e.g., $\alpha = 0.5, \beta = 0.3, \gamma = 0.2$).

Interpretation of CSS

- **CSS = 100:** Indicates perfect security across all three components, ensuring the highest level of protection against all types of attacks.
- **CSS = 80-99:** Reflects high security, with minor weaknesses in one or more components.
- **CSS = 60-79:** Indicates moderate security, suggesting potential vulnerabilities that need addressing.

- **CSS $<$ 60:** Signifies significant security weaknesses, requiring immediate improvements to avoid potential breaches.

3.8 Impacts and Trade-offs of Addshare+

3.8.1 Weight selection method impact

To optimize the trade-off between privacy, efficiency, and model accuracy, different strategic methods for selecting the subset of weights to be shared during the training process can be explored. These methods, typically employed for model pruning, can be repurposed to intelligently identify the most critical weights for sharing.

3.8.1.1 Random Weight Selection

This straightforward method randomly selects weights for sharing. While its simplicity is advantageous in terms of implementation and computational efficiency, it may neglect important weights crucial for model performance, potentially hindering convergence and accuracy.

3.8.1.2 Magnitude-based Weight Selection

This approach prioritizes weights based on their absolute magnitude, assuming larger values exert a more significant influence on the model's output [39]. The key advantages of magnitude-based weight pruning are its simplicity and computational efficiency. Unlike methods such as Optimal Brain Damage (OBD), which require computing second-order derivatives, magnitude-based pruning only involves sorting the weights based on their magnitudes, which is a relatively inexpensive operation. However, it is important to note that magnitude-based pruning is a heuristic approach and does not have a theoretical foundation like OBD. It has been empirically shown to work well in practice as shown in the work

of Li et al [51] which explored magnitude-based pruning for yield forecasting, demonstrated significant reductions in communication overhead while maintaining, and even improving, model accuracy compared to unpruned baselines.

3.8.1.3 Optimal Brain Damage

OBD is a weight pruning method proposed by Yann LeCun et al. [50] for neural networks. The goal of this method is to remove redundant connections (weights) from the network while minimizing the increase in the training error. This is achieved by computing the saliency of each weight in the network and then removing the weights with the lowest saliencies. The saliency of a weight is defined as the second-order derivative of the error function with respect to that weight, which provides an estimate of the change in the error function when that weight is removed. While offering efficient weight selection with potential benefits for communication and accuracy, OBD comes at the cost of computationally expensive second-order derivative calculations.

The OBD method can be described as follows:
(a) Train the neural network to convergence on the training data.
(b) Compute the saliencies of all the weights in the network using the following formula:

$$s_i = \frac{1}{2} \cdot \left(\frac{\partial^2 E}{\partial w_i^2} \right) \tag{3.32}$$

where s_i is the saliency of the ith weight w_i, and E is the error function of the network.
(c) Sort the weights in ascending order of their saliencies.
(d) Remove a fraction of the weights with the smallest saliencies from the network.
(e) Retrain the pruned network to convergence on the training data.

The key idea behind OBD is that weights with small saliencies have a relatively small impact on the error function, and thus can be removed with minimal impact on the net-

work's performance. By iteratively removing the least salient weights and retraining the network, OBD can significantly reduce the number of connections in the network while maintaining its accuracy

3.8.1.4 L1 Regularization

L1 weight pruning is a weight pruning method for neural networks that leverages L1 regularization. The key idea behind this method is to introduce an L1 regularization term in the loss function during training, which encourages the network to learn sparse weight representations. The L1 regularization term adds the sum of the absolute values of the weights to the loss function, effectively pushing smaller weights towards zero. The pruning process then involves removing the weights that are below a certain threshold. The method can be described as follows:
(a) Modify the loss function of the neural network by adding an L1 regularization term:

$$\text{Loss} = \text{original\_loss} + \lambda \cdot \sum_i |w_i| \tag{3.33}$$

where: original_loss is the original loss function of the neural network, λ is the regularization strength hyperparameter, w_i are the weights of the neural network and $\sum_i |w_i|$ represents the sum of the absolute values of all weights.

(b) Train the neural network with the modified loss function.
(c) After training, sort the weights in ascending order based on their magnitudes (absolute values).
(d) Remove a fraction of the weights with the smallest magnitudes from the network. The fraction of weights to be pruned is p that can be tuned based on the desired trade-off between model size and performance.

3.8.2 Communication Overhead Reduction

One of the key advantages of the AddShare+ approach with groups is the reduction in communication overhead during the additive secret sharing process. In traditional FL where all n clients participate, each client needs to generate n shares and send $n-1$ shares to every other client. This leads to a total of $n(n-1)$ messages being communicated across the system. However, by dividing the clients into smaller groups of size k, AddShare+ can significantly cut down on this communication cost. When operating within a group of k clients, each client only needs to generate k shares and send $k-1$ shares to the other clients in the same group. Across all the groups of size k, the total number of messages reduces to $n(k-1)$. Since k is smaller than n, this grouped approach decreases the communication overhead by a factor of $\frac{n-1}{k-1}$ compared to involving all n clients in the sharing process. This reduction in messages translates to lower bandwidth usage and faster secret sharing, thereby improving the overall efficiency and scalability of the FL system. The formalization of this communication overhead reduction is as follows:

With n being the total number of clients for additive (n, n) secret sharing, each client needs to generate n shares. Each client sends $n-1$ shares to the other $n-1$ clients as such, each client sends $(n-1)$ messages. The total number of messages exchanged among the clients can be represented as T_n.

$$T_n = n \times (n-1) \tag{3.34}$$

Considering communication rounds with groups of size k where $k < n$, clients are divided into $g = \frac{n}{k}$ groups of size k each. In each group, additive (k, k) secret sharing is performed whereby each client generates k shares and sends $k-1$ shares to the other $k-1$ clients in the same group. Within its group, each client sends $(k-1)$ messages with the total messages sent within one group being T_k.

$$T_k = k \times (k-1) \tag{3.35}$$

Across all g groups, the total number of messages sent can be represented as T_g.

$$\begin{aligned} T_g &= g \times T_k \\ T_g &= g \times k \times (k-1) \\ T_g &= \tfrac{n}{k} \times k \times (k-1) \\ T_g &= n \times (k-1) \end{aligned} \tag{3.36}$$

Since $k < n$, we have $(k-1) < (n-1)$. Therefore, $n \times (k-1) < n \times (n-1)$.

So performing AddShare within groups of size k reduces the total number of messages sent during the additive secret sharing process across all clients by a factor of $\frac{n-1}{k-1}$ compared to not using groups.

Optimal Group Size Selection and Worst-Case Analysis

The selection of group size k requires careful optimization across multiple system constraints and security requirements. First, k should be selected such that $\frac{n}{k}$ results in an integer number of groups to ensure even distribution of clients. A practical minimum value for k is 3, as a smaller group size of 2 provide insufficient privacy guarantees. The upper bound for k is typically constrained by communication overhead where larger groups require more message exchanges between clients, with communication cost growing quadratically as $O(k^2)$ within each group. The worst-case scenario manifests when $k \to n$, effectively eliminating the communication benefits of grouping and reverting to the communication complexity of the basic additive secret sharing scheme. Additionally, if k is too large relative to network conditions and client capabilities, it increases the risk of synchronization failures during share exchange.

3.8.3 Precision Loss and Rounding Errors

Leveraging additive secret sharing for privacy-preserving FL necessitates splitting model updates into independent shares. Only the aggregated sum reveals the true update, ensur-

ing individual contributions remain cloaked. However, this approach incurs a performance penalty, introducing rounding errors and compromising precision. Floating-point numbers have inherent limitations in precision, meaning they cannot represent all real numbers exactly. If the original weight has more significant digits than the shares can accommodate, information gets lost during the splitting process. In addition, each additive share embodies a fraction of the original weight as a floating-point number. Reconstructing the weight involves repeated additions of these floating-point values, each susceptible to rounding errors due to the finite number of bits used for representation. With more shares (using more clients), these errors accumulate, causing a larger deviation from the original value, further fueling inaccuracy. These accumulated errors can significantly impact the final model's accuracy.

Let ϵ_{split} be the maximum error introduced when splitting the original model weight W (a floating-point number) into n shares due to rounding to fit the floating-point representation. Let w_i represent a single share when W of a client i is split into n additive shares. The share w_i is a floating-point approximation of $\frac{W}{n}$ with limited precision due to the finite number of bits used to represent floating-point numbers. After sharing and reconstruction, the reconstructed weight can be represented as $W' = \sum_{i=1}^{n} w_i$. Each addition $w_i + w_{i+1}$ can introduce a rounding error ϵ_{add} due to floating-point arithmetic. Let ϵ_{total} be the total rounding error accumulated when reconstructing W' from the n shares. We can then formally state:

$$\epsilon_{\text{total}} \leq \epsilon_{\text{split}} + (n-1)\epsilon_{\text{add}} \tag{3.37}$$

When using all n clients, the number of additive shares is maximum, so ϵ_{total} is higher due to more accumulation of ϵ_{add} terms. However, when using subgroups of k clients ($k < n$), we have:

$$\epsilon_{\text{total}} \leq \epsilon_{\text{split}} + (k-1)\epsilon_{\text{add}} \tag{3.38}$$

Since $k < n$, the accumulation of ϵ_{add} terms is reduced, leading to a lower ϵ_{total}. Hence, by using subgroups the AddShare+ approach can control the accumulation of rounding errors, improving precision while still providing privacy guarantees.

3.9 Summary

This chapter presented a comprehensive overview of the AddShare and AddShare+ frameworks. We began by outlining the general architecture of FL, highlighting the roles of clients and the central server. The AddShare framework was then introduced, detailing its four main components: additive secret sharing, group formation, encryption, and FedAvg [57]. Building upon AddShare, we introduced AddShare+, which addresses the limitations of its predecessor by implementing selective weight sharing. This approach significantly reduces communication overhead and minimizes accuracy loss associated with the additive secret sharing process. We explored the implementation of the Elliptic Curve Integrated Encryption Scheme ECIES in AddShare+, offering improved efficiency over the RSA encryption used in AddShare. We also delved into the security guarantees provided by AddShare+, demonstrating how additive secret sharing ensures information-theoretic security against potential threats, including honest-but-curious servers and eavesdroppers. We also discussed the impacts and trade-offs of AddShare+, exploring various weight selection methods and their implications on model performance and privacy. Furthermore, we analyzed the communication overhead reduction achieved through client grouping and examined the precision loss and rounding errors inherent in the additive secret sharing process. These provide crucial insights into the balance between privacy, efficiency, and accuracy in the proposed frameworks.

As we move into Chapter 4, we will build upon this theoretical foundation to present the experimental settings, results, and analysis of the proposed approaches. The upcoming chapter will provide empirical evidence of the effectiveness of AddShare and AddShare+ across multiple datasets and configurations, offering a comprehensive evaluation of their

performance in terms of accuracy, efficiency, and privacy trade-offs.

Chapter 4

Experimental Design and Evaluation

4.1 Introduction

In this chapter, we present a comprehensive experimental evaluation of the AddShare and AddShare+ protocols. We assess the effectiveness, efficiency, and privacy-preserving capabilities of these approaches through a series of well-designed experiments and analyses. This chapter outlines the experimental methodology, including dataset selection, model architectures, evaluation metrics, and experimental setup. We examine the performance of these protocols across multiple benchmark datasets, comparing AddShare and AddShare+ against baseline methods such as FedAvg [57], SCOTCH [61] and FedShare [29]. A significant portion of this chapter is dedicated to explaining our evaluation metrics, which include SA, Client-side Average Accuracy (CAA), FLT, Average Secret Sharing Time (ASST), and Average Training Time (ATT). These metrics collectively provide a holistic view of the performance, efficiency, and privacy trade-offs. Furthermore, we conduct in-depth ablation studies to investigate the impact of key components in AddShare+, such as the percentage of weights sampled and different weight selection approaches. These studies provide valuable insights into the robustness and adaptability of our proposed methods. This chapter concludes with a comprehensive discussion of our findings, highlighting the significant

advantages of the protocols in maintaining high model performance while substantially reducing communication and computational overhead. We also discuss the implications of our results for real-world applications of privacy-preserving FL.

4.2 Datasets

The selection of datasets is a pivotal component of the experimental design, as it ensures the evaluation of AddShare and AddShare+ techniques across diverse data types and complexities. For this study, we have chosen four widely-used datasets: CIFAR-10 [48], Modified National Institute of Standards and Technology (MNIST) [24], F-MNIST [93], and Street View House Numbers (SVHN) [62]. These datasets are representative of different challenges in image classification tasks and provide a comprehensive benchmark for assessing the performance and privacy-preserving capabilities of our FL models. Each dataset is distributed equally among 50 clients in an IID manner to simulate a FL environment.

4.2.1 CIFAR-10

The CIFAR-10 dataset [48] is a widely utilized benchmark dataset in computer vision and ML research. It comprises 60,000 color images, each with dimensions of 32x32 pixels, categorized into 10 distinct classes. The dataset is partitioned into 50,000 training images and 10,000 test images, with each class represented by 6,000 images. This balanced distribution ensures equitable representation across all categories. The classes encompass a diverse range of objects and scenes, including airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks, providing a comprehensive representation of real-world visual categories. The relatively small image size coupled with the variety of subjects presents a significant challenge for image classification algorithms. A summary of the CIFAR-10 dataset's key attributes can be found in Table 4.1. In the context of our FL experiments, preprocessing involved normalizing pixel values to the range [0, 1] and implementing one-hot encoding for labels, optimizing the data for distributed model training and evaluation.

The CIFAR-10 dataset is chosen for its complexity and the variety of classes it offers, which are instrumental in evaluating the robustness of our FL models under AddShare and AddShare+ protocols.

4.2.2 MNIST

The MNIST [24] dataset is a foundational resource in the field of digit recognition. It consists of 70,000 grayscale images of handwritten digits, each sized at 28x28 pixels. The dataset is divided into 60,000 training images and 10,000 test images, providing a robust basis for model development and evaluation. MNIST encompasses digits from 0 to 9, with each class well-represented in the dataset. The handwritten nature of these digits introduces natural variations in style, slant, and thickness, simulating real-world scenarios and challenging recognition algorithms to develop robustness and adaptability. Despite its apparent simplicity, MNIST continues to serve as a fundamental benchmark in ML, particularly for testing novel algorithms and architectural innovations. For our FL framework, preprocessing of the MNIST data involved normalizing pixel values to the range [0, 1] and employing one-hot encoding for labels, ensuring optimal compatibility with the distributed learning architecture. The simplicity of the MNIST dataset makes it an ideal starting point for testing the basic functionality and performance of the AddShare and AddShare+ techniques in a controlled environment. For a concise overview of the MNIST dataset's properties, refer to Table 4.1

4.2.3 F-MNIST

F-MNIST [93], developed by Zalando Research, presents an enhanced alternative to the traditional MNIST dataset. This collection comprises 70,000 grayscale images, each sized at 28x28 pixels, maintaining consistency with the MNIST format. The dataset is equally divided into 60,000 training images and 10,000 test images. F-MNIST categorizes images into 10 classes of fashion items, including T-shirts/tops, trousers, pullovers, dresses, coats,

sandals, shirts, sneakers, bags, and ankle boots. This diversity in fashion items introduces a higher level of complexity compared to digit recognition, requiring models to differentiate between sometimes subtle variations in clothing styles and accessories. The grayscale format and consistent image size make F-MNIST an ideal dataset for benchmarking ML algorithms, particularly in scenarios where increased complexity is desired without significantly increasing computational requirements. The key characteristics of the F-MNIST dataset are summarized in Table 4.1 In our FL experiments, preprocessing involved normalizing pixel values to the range [0, 1] and converting labels to one-hot encoding, optimizing the data for effective distributed learning and evaluation.

4.2.4 SVHN

The SVHN [62] dataset is derived from Google Street View images, this dataset contains 630,420 color images of house numbers, each sized at 32x32 pixels. The dataset is structured with 73,257 digits for training, 26,032 for testing, and an additional 531,131 less difficult samples. Unlike controlled datasets, SVHN captures digits in their natural context, complete with real-world challenges such as varying illumination, occlusions, and distortions. The images span the 10 digit classes (0 through 9), often featuring multiple digits in a single image, which adds complexity to the recognition task. This real-world aspect makes SVHN particularly valuable for developing robust models capable of performing in uncontrolled environments. In our FL setup, preprocessing of the SVHN images involved normalizing pixel values to the range [0, 1] and employing one-hot encoding for labels, ensuring compatibility with our framework while preserving the dataset's inherent challenges. The distinguishing features of the SVHN dataset, including its size and class distribution, are detailed in Table 4.1.

Table 4.1: Characteristics of the datasets used in our experiments.

Reference	Dataset	# Classes	Dimensions	Train Size (per client)	Total Train Size	Total Test Size
[48]	CIFAR10	10	32x32	1000	50000	10000
[24]	MNIST	10	28x28	1200	60000	10000
[93]	F-MNIST	10	28x28	1200	60000	10000
[62]	SVHN	10	32x32	1465	73257	26032

4.2.5 Federated Dataset Distribution

To ensure an equal and IID distribution of the datasets among 50 clients, we implemented a method that randomly shuffles the dataset indices and then partitions them equally. This process is crucial for simulating an FL environment where each client holds a balanced subset of the entire dataset, thus maintaining statistical consistency across clients.

Independent: In this context, it means that the selection of data samples for one client is independent of the selection for another client. This is achieved by randomly shuffling the dataset indices before partitioning them, ensuring no overlap or bias between the data assigned to different clients.

Identically Distributed: This means that each client's data subset comes from the same overall distribution as the original dataset. By partitioning the shuffled indices equally among the clients, we ensure that each subset retains the same statistical properties as the entire dataset, such as class distribution and feature representation.

First, we generate an array of indices corresponding to the training samples. This array is then randomly shuffled to ensure that the data is independently distributed among the clients. Subsequently, the shuffled indices are divided into 50 equal parts, with each part assigned to a client, ensuring identical distribution. This guarantees that each client receives an equal number of training samples that are representative of the overall dataset. The index assignments for each client are then assigned to them. The detailed implementation of this process in Python is presented in Algorithm 4.1.

Algorithm 4.1 IID Balanced Data Distribution

function IID_BALANCED(client_number, train_size, dataset)
    rand_array ← np.arange(train_size)
    np.random.shuffle(rand_array)
    clients ← [[] for i in range(client_number)]
    for i in range(client_number) **do**
        clients[i] ← rand_array[$\lfloor i \times \frac{\text{train\_size}}{\text{client\_number}} \rfloor$: $\lfloor (i+1) \times \frac{\text{train\_size}}{\text{client\_number}} \rfloor$]
    end for
    save indices
end function

By utilizing this IID distribution method, we ensure that each client in the FL setup has an equally balanced subset of the dataset, which is crucial for maintaining the integrity and validity of the experimental results. This approach guarantees that the subsets are both independent and identically distributed, aligning with the principles of IID. The IID distribution assumption aligns well with many real-world enterprise environments where organizations follow standardized data collection and labeling protocols, ensuring consistent data distributions across different branches or departments. This is particularly true in regulated industries like healthcare and finance where strict data standards and quality control measures help maintain uniform data distributions across different locations or units

4.3 Evaluation Metrics

The evaluation of FL models, particularly those employing privacy-preserving techniques such as AddShare and AddShare+, requires a comprehensive set of metrics to measure both performance and efficiency. In this section, we detail the evaluation metrics used in our experiments, which include Server-side Accuracy (SA), Client-side Average Accuracy (CAA), Federated Learning Time (FLT), Average Secret Sharing Time (ASST), and Average Training Time (ATT). These metrics provide a holistic view of the effectiveness and

practicality of the models in various scenarios.

4.3.1 Server-side Accuracy

SA is a critical metric that evaluates the performance of the global model aggregated at the central server. It measures how accurately the combined model, which incorporates weight updates from multiple clients, performs on a designated test dataset. This metric is crucial because it reflects the overall effectiveness of the FL process in producing a robust and generalizable model. Higher SA indicates better model performance and successful aggregation of client contributions. It demonstrates the collaborative power of FL, showing how well the combined efforts of all clients improve the model. SA was specifically chosen as the primary evaluation metric due to the balanced nature of our datasets, where each class is equally represented across all clients. This balanced distribution ensures that accuracy provides an unbiased measure of model performance, as it gives equal weight to the correct classification of each class. SA is computed by testing the aggregated global model on a central test dataset and recording the proportion of correctly classified instances.

$$\text{SA} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Test Cases}} \tag{4.1}$$

4.3.2 Client-side Average Accuracy

CAA measures the average accuracy of local models trained on individual client datasets. This metric provides insight into the performance of each client's model before weight updates are shared with the central server. CAA helps to understand the variability in model performance across different clients and datasets. It highlights the effectiveness of local training and the consistency of model performance across clients, which is essential for fair and balanced FL. CAA is obtained by averaging the accuracy of all local models

on their respective local test datasets.

$$\text{CAA} = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{Number of Correct Predictions}_i}{\text{Total Number of Test Cases}_i} \tag{4.2}$$

where n is the number of clients.

4.3.3 Federated Learning Time

FLT is the average total time taken for one complete round of FL. This includes the local training time on clients, the time taken for communication of updates between clients and the server, secret sharing time, and the aggregation time at the server. It provides a measure of the time efficiency of the FL process, which is critical for applications requiring timely updates. FLT is the sum of the time taken for local training, communication latency, secret sharing time, and server-side aggregation, averaged over multiple rounds.

$$\text{FLT} = T_{\text{train}} + T_{\text{ss}} + T_{\text{comm}} + T_{\text{agg}} \tag{4.3}$$

where T_{train} is the local training time, T_{ss} is the secret sharing time, T_{comm} is the communication time, and T_{agg} is the aggregation time.

4.3.4 Average Secret Sharing Time

ASST measures the average time taken by each client to perform the additive secret sharing process. This process involves splitting the local model weights into multiple shares and securely exchanging them with other clients. It assesses the overhead introduced by the privacy-preserving mechanism, helping to understand its impact on the overall learning time. ASST is computed by timing the secret sharing process for each client and averaging it across all clients.

$$\text{ASST} = \frac{1}{N}\sum_{i=1}^{N} T_{\text{ss}}^{i} \tag{4.4}$$

where N is the number of clients and T_{ss}^{i} is the secret sharing time for client i.

4.3.5 Average Training Time

ATT measures the average time taken by each client to train their local model. This metric focuses on the computational load imposed on the clients due to local model training. It helps to gauge the computational efficiency of the FL approach, ensuring that local training is not excessively burdensome for clients with limited resources. ATT is calculated by timing the local training process for each client and averaging it across all clients.

$$\text{ATT} = \frac{1}{N} \sum_{i=1}^{N} T_{\text{train}}^{i} \tag{4.5}$$

where N is the number of clients and T_{train}^{i} is the training time for client i.

These evaluation metrics collectively provide a detailed assessment of the performance and efficiency of the AddShare and AddShare+ techniques in FL. By analyzing these metrics, we can draw meaningful conclusions about the trade-offs between model accuracy, computational efficiency, and privacy preservation, offering insights into the practical applicability of these methods in real-world scenarios.

4.4 Model Architecture

The architecture of a neural network model describes its structural design. This encompasses the arrangement and characteristics of its layers, including the quantity and type of layers employed, the number of neurons or filters within each layer, and the activation functions utilized. The interconnectivity of these layers is also defined by the model architecture. This architecture plays a pivotal role in dictating the model's capacity to learn and tackle specific tasks or problems. The model architecture used in this study is LeNet-5 [49], a pioneering Convolutional Neural Network (CNN) developed by Yann LeCun and

his colleagues in the late 1990s. LeNet-5 [49] is designed primarily for handwritten and machine-printed character recognition tasks, specifically for the MNIST [24] dataset, but its architecture has proven versatile enough to be applied to various other image classification tasks as well. In the FL setup, each client locally trains a LeNet-5 model on their private dataset shard. LeNet-5 [49] consists of the following layers:

1. Input Layer: The input to LeNet-5 [49] is a 32x32 pixel grayscale image. For the datasets used in this study, such as MNIST (which provides 28x28 pixel images), the images are typically zero-padded to fit the required input dimensions.

2. Convolutional Layer C1: This layer has six 5x5 convolutional kernels (filters) with a stride of 1, producing six feature maps of size 28x28. The convolutional operation captures local features such as edges and textures.

3. Subsampling Layer S2: This layer performs average pooling (subsampling) with a 2x2 filter and a stride of 2, resulting in six 14x14 feature maps. This reduces the spatial resolution and provides translation invariance.

4. Convolutional Layer C3: In this layer, there are sixteen 5x5 convolutional kernels. Each kernel is connected to all six of the S2 layer's feature maps, yielding sixteen 10x10 feature maps. This layer learns more complex features by combining the lower-level features detected by C1 and S2.

5. Subsampling Layer S4: Similar to S2, this layer performs average pooling with a 2x2 filter and a stride of 2, resulting in sixteen 5x5 feature maps.

6. Convolutional Layer C5: This layer has 120 5x5 convolutional kernels, but because the input feature maps are also 5x5, the result is 120 1x1 feature maps. Effectively, each of the 120 outputs is a result of a full connection with the previous layer's feature maps, acting more like a fully connected layer.

7. Fully Connected Layer F6: This layer consists of 84 units, each fully connected to the 120 units of C5. This layer combines the features extracted by the convolutional and subsampling layers to form higher-level representations.

8. Output Layer: The final layer is a softmax layer that produces a probability distribution over the ten class labels (for a ten-class classification task like all our datasets used).

4.4.1 Model Hyperparameters

Hyperparameter tuning is essential for optimizing the performance of the LeNet-5 [49] model. Below are detailed definitions, descriptions, and formalizations for each key hyperparameter. Table 4.2 shows the hyperparamter values for the LeNet-5 [49] model used.

Table 4.2: Hyperparameters for the LeNet-5 configuration

Hyperparameter	**Value**
Batch Size	32
epochs	2
learning rate	0.001
Number of CNN Block	3
Dropout	0.2
Stride	2
Number of Filters	64, 8

Learning Rate: The learning rate determines the step size during the update of the model's parameters. It controls how quickly or slowly a model learns by adjusting the weights with respect to the loss gradient. The learning rate (η) is crucial because a high learning rate can cause the model to converge too quickly to a suboptimal solution. Whereas a low learning rate can make the training process very slow and may get stuck in a local minimum. The weight update rule in gradient descent is given by:

$$W' = W - \eta \frac{\partial L}{\partial W} \tag{4.6}$$

where W represents the weights, η is the learning rate, and $\frac{\partial L}{\partial W}$ is the gradient of the loss function L with respect to the weights.

Batch Size: The batch size is the number of training samples processed in one forward and backward pass of the training algorithm. Smaller batch sizes (e.g., 16) lead to more frequent updates and can help in regularizing the model, but they make the training process slower. Larger batch sizes (e.g., 128) make training faster due to parallel processing but can result in less stable updates due to reduced gradient noise. The number of batches B required to cover the entire dataset is:

$$B = \left\lfloor \frac{N}{m} \right\rfloor \tag{4.7}$$

where N is the total number of samples in the dataset, and m is the batch size.

Epochs: The number of epochs refers to the number of complete passes through the entire training dataset. Too few epochs can lead to underfitting as the model might not learn enough and too many epochs can cause overfitting where the model learns the noise in the training data. In each epoch, the model parameters are updated once per batch for all batches in the dataset.

Dropout Rate: Dropout rate is a regularization technique where randomly selected neurons are ignored during training, which helps prevent overfitting. Low dropout rates (e.g., 0.1) may not effectively prevent overfitting. High dropout rates (e.g., 0.4) can lead to underfitting by not allowing the model to learn sufficient features. Dropout is typically applied to the neurons' output as:

$$\text{Dropout}(x) = x \cdot \text{Bernoulli}(p) \tag{4.8}$$

where x is the input, and $\text{Bernoulli}(p)$ is a Bernoulli random variable with probability p of retaining a neuron.

Stride: This refers to the number of pixels the filter moves over the input image during convolution. A stride of 1 maintains high spatial resolution but increases computational load. Larger strides (e.g., 3) reduce the feature map size and computational load but may lose important spatial information. The output dimension of a convolutional layer is given by:

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - F + 2P}{S} \right\rfloor + 1 \tag{4.9}$$

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} - F + 2P}{S} \right\rfloor + 1 \tag{4.10}$$

where W_{in} and H_{in} are the input width and height, F is the filter size, P is the padding, and S is the stride.

Kernel Size: Kernel size refers to the dimensions of the filter used in the convolutional layer, typically represented as a square matrix (e.g., 3×3). Larger kernels can capture more complex features but are computationally expensive. Smaller kernels focus on local features and require less computation. The receptive field of a neuron in a convolutional layer with kernel size $k \times k$ is $k \times k$ pixels of the input image.

Number of Filters: The number of filters in a convolutional layer determines the depth of the output feature maps. More filters (e.g., 64) allow the model to learn more complex features but increase the risk of overfitting and computational cost. Fewer filters (e.g., 16) may not capture sufficient detail. If the input has depth D and there are K filters of size $F \times F$, the output feature map will have depth K. The number of parameters in a convolutional layer is:

$$\text{Parameters} = K \times (F \times F \times D + 1) \tag{4.11}$$

where the $+1$ accounts for the bias term in each filter.

4.5 Model Selection Rationale

The selection of LeNet-5 [49] as our model architecture was driven by practical considerations rather than performance optimization goals. The primary objective was to demonstrate that AddShare and AddShare+ protocols could maintain model performance while enhancing privacy and efficiency, rather than to optimize the underlying model architecture itself. LeNet-5's [49] established track record, straightforward architecture, and widespread use in the field make it an ideal candidate for this purpose. This approach allows us to isolate and analyze the impact of our privacy-preserving mechanisms without the confounding effects of complex model architectures.

During the initial stages of framework development, several contemporary deep learning architectures were evaluated, including ResNet-18 [41], VGG-16 [75], and MobileNetV2 [72]. However, with our experimental setup involving 50 clients and 10 communication rounds, these models presented significant computational challenges. For instance, training ResNet-18 [41] across all clients sequentially required approximately 72 hours to complete one FL round, while VGG-16 [75] needed even longer (roughly 96 hours per round). These extended training periods made comprehensive testing and validation of the AddShare and AddShare+ protocols impractical within our research timeframe.

The initial FL setup operated sequentially, where clients would train one after another until all clients completed their local training before aggregation could occur. This sequential approach, while straightforward to implement, significantly extended the overall training duration. To address this limitation, we later enhanced our framework by implementing a parallel training architecture using FastAPI [69] servers. This redesign allowed all clients to train simultaneously and communicate with each other and the aggregating server through REST APIs, substantially reducing the total training time.

In this context, LeNet-5 [49] emerged as the optimal choice for our research objectives. Its relatively small architecture, requiring only around 3 hours per FL round with sequential training and approximately 45 minutes with parallel training, enabled us to

conduct comprehensive experiments while maintaining statistical validity. This efficiency was crucial for performing multiple ablation studies and analyzing various aspects of our protocols. Furthermore, LeNet-5's [49] simpler architecture helped ensure that any performance variations could be clearly attributed to our privacy-preserving mechanisms rather than to complexities in the model architecture itself. This aligns perfectly with our goal of demonstrating that AddShare and AddShare+ can be effectively implemented without degrading model performance, regardless of the underlying architecture.

4.6 Experimental Setup

The experimental setup includes the specific hardware and software environments used to conduct the experiments, as well as the configurations and initial conditions necessary for reproducibility. This ensures that our results are reliable and can be replicated by other researchers in the field. The development of the AddShare and AddShare+ protocols was carried out using TensorFlow Keras and the Python cryptography package. TensorFlow Keras, a high-level API for building and training neural networks, enabled the efficient construction and tuning of complex deep learning models. Its modular design facilitated rapid development and experimentation with different network architectures. The Python cryptography package was crucial for implementing robust encryption mechanisms used i.e., RSA and ECIES, ensuring data privacy and security in the FL process. This package provides secure cryptographic primitives and recipes, allowing the protocol to handle sensitive data securely and maintain user privacy across distributed environments. The computational infrastructure used for training the AddShare+ model was the Linux Béluga cluster, managed by the Digital Research Alliance of Canada. The training jobs were submitted using Simple Linux Utility for Resource Management (SLURM) jobs to Béluga with specific maximumn resource requests: five nodes, 30 CPUs per task, 35 GB of memory, and a wall-time of six days. Béluga's powerful and diverse compute nodes, including those equipped with NVIDIA V100 GPUs, provided the necessary computational

power to handle intensive training tasks.

4.7 Experimental Results

The experimental evaluation of AddShare and AddShare+ was conducted to thoroughly assess their performance in a FL setting, focusing on three critical aspects: accuracy, efficiency, and privacy preservation. As they introduce innovative mechanisms such as selective weight sharing, group formation, and encryption (RSA and ECIES), it is essential to understand how these techniques influence the overall performance compared to widely recognized FL baselines such as FedAvg, SCOTCH and FedShare. The evaluation aims to demonstrate that not only do they maintains competitive accuracy but also provides significant advantages in privacy protection and communication efficiency, making it a viable solution for real-world applications where data privacy and resource constraints are paramount.

To validate AddShare+, we designed a comprehensive set of experiments using four benchmark datasets: CIFAR-10, MNIST, F-MNIST, and SVHN. These datasets span different levels of complexity and variability, ensuring that the results are representative of diverse application scenarios. Various metrics were employed to evaluate performance, including SA, CAA, FLT, and ASST. These metrics collectively provide insights into the trade-offs between privacy, accuracy, and efficiency in both client and server operations. The experiments were structured to compare AddShare+ against baseline methods under identical conditions, assessing the impact of privacy-preserving features such as encryption and weight sharing on model accuracy, system efficiency, and the overall FL process. Special emphasis was placed on understanding how different group sizes (G3, G5, G10) and group formation strategies (server-side vs. client-side) affect the system's performance in various configurations. By isolating key factors, we aim to highlight the strengths of AddShare+ and its suitability for applications in privacy-sensitive domains such as healthcare, finance, and edge computing environments.

4.7.1 Accuracy Analysis: Balancing Accuracy and Privacy

In this section, we examine the accuracy of AddShare+ in comparison with other FL baselines, specifically FedAvg, FedShare, and Scotch. Across different datasets (MNIST, F-MNIST, CIFAR-10, and SVHN), AddShare+ demonstrates competitive accuracy while maintaining privacy through selective weight sharing and encryption mechanisms. The results suggest that the accuracy trade-offs are minimal, reinforcing the practicality of AddShare+ for real-world, privacy-sensitive applications. A plot showing the performance is shown in Figure 4.1.

Vanilla AddShare and AddShare+ Performance

The first major insight is from the MNIST dataset, where we see a general trend of high accuracy across all methods. FedAvg [57] achieves the highest SA at 95.8%, but Vanilla (unencrypted) AddShare+ follows closely with an SA of 95.7%, showing a negligible difference of just 0.001. Vanilla AddShare lags slightly with an SA of 95.2%, reflecting the impact of its simpler weight-sharing mechanism. When compared to FedShare 95.0% and SCOTCH 94.4%, AddShare+ outperforms these models while introducing privacy-preserving features like selective weight sharing. This result is crucial because it demonstrates that privacy mechanisms in AddShare+ do not significantly degrade accuracy in a relatively simple dataset like MNIST. CAA results remain consistent, with AddShare+ and FedAvg [57] both achieving 94.2%, while SCOTCH and FedShare trail slightly at 94.2% and 94.2%, respectively. This is shown in Table 4.3.

Table 4.3: Performance results of vanilla versions of AddShare and AddShare+ against other baselines on the MNIST dataset

Test	Server		Client		
	SA	FLT (s)	CAA	CTT (s)	ASST (s)
Scotch	0.944	8.283	0.942	1.876	1.487
FedShare	0.950	404.625	0.942	1.867	1.139
FedAvg	0.958	354.818	0.942	1.927	*
AddShare	0.952	400.673	0.937	1.935	0.925
Addshare+	0.957	347.539	0.942	1.595	0.407

On the more challenging F-MNIST dataset, which presents a greater challenge, FedAvg [57] still leads with an SA of 79.3%. AddShare+ follows closely with an SA of 78.9%, reflecting a minimal trade-off in accuracy for added privacy. Vanilla AddShare performs worse, achieving only 74.3%, suggesting that its simpler privacy methods are not as effective in more complex datasets. FedShare records an SA of 76.1%, while SCOTCH lags behind at 70.6%, further highlighting the competitive edge of AddShare+ in balancing privacy and accuracy. What's particularly notable in this dataset is the small difference between AddShare+ and FedAvg, which suggests that privacy-preserving mechanisms, such as selective weight sharing, do not hinder performance significantly. Moreover, AddShare+ achieves a CAA of 76.4%, nearly identical to FedAvg's [57] 76.5%, while outperforming FedShare (76.2%) and SCOTCH (76.2%), which shows that client-side accuracy is maintained despite the introduction of privacy mechanisms.

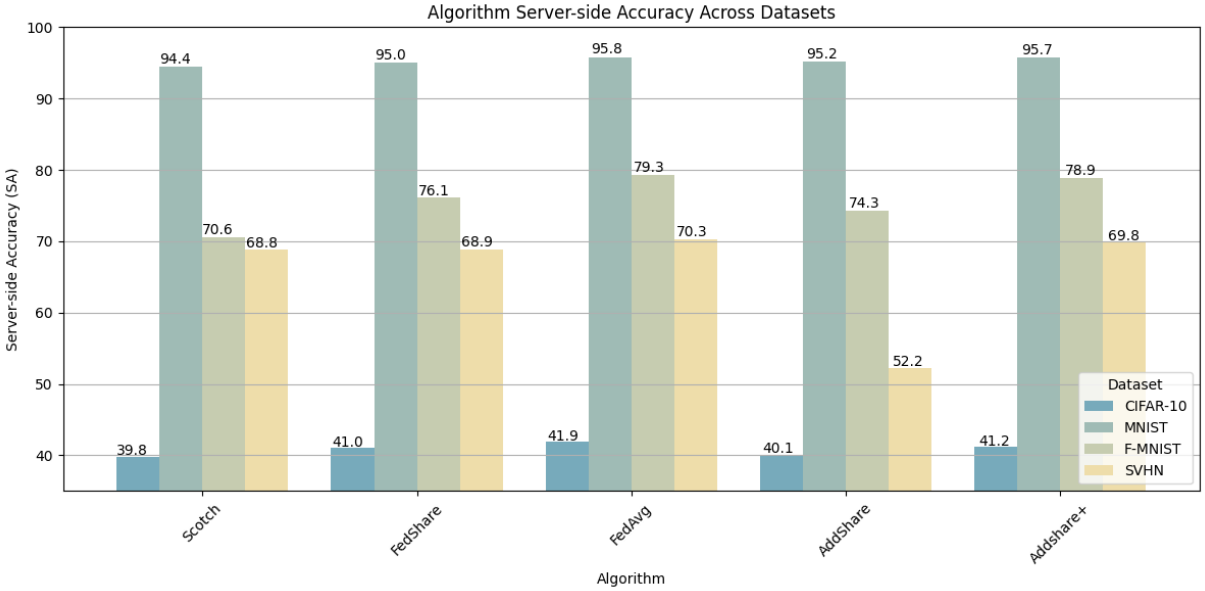

Figure 4.1: Performance of algorithms on four different datasets.

The CIFAR-10 dataset, known for its complexity and high-dimensional data, paints a similar picture. FedAvg [57] achieves an SA of 41.9%, with AddShare+ closely behind at 41.2%. This minimal 0.7% difference reinforces the conclusion that AddShare+ preserves accuracy even under more demanding conditions. Vanilla AddShare records a lower SA of 40.1%, indicating that its simpler approach struggles with the complexity of the dataset. FedShare and SCOTCH, with SAs of 41.0% and 39.8%, respectively, perform comparably to AddShare+, showing that all these models face challenges in maintaining high accuracy on complex datasets. However, AddShare+ maintains an edge due to its privacy mechanisms, achieving a CAA of 37.6%, in line with FedAvg (37.7%), SCOTCH (37.7%) and FedShare (37.7%). The minimal drop in accuracy between AddShare+ and the baselines, coupled with its privacy-preserving features, suggests that it offers a viable solution for complex datasets without compromising much in terms of accuracy. This is shown in Table 4.4

Table 4.4: Table showing the performance of vanilla versions of AddShare and AddShare+ against other baselines on the CIFAR-10 dataset

Test	Server		Client		
	SA	FLT (s)	CAA	CTT (s)	ASST (s)
Scotch	0.398	8.431	0.377	1.871	2.584
FedShare	0.410	430.711	0.377	1.574	1.704
FedAvg	0.419	353.599	0.377	1.521	*
AddShare	0.401	417.811	0.363	1.595	1.138
Addshare+	0.412	356.533	0.376	2.005	0.364

Lastly, the SVHN dataset, which is another complex dataset with real-world images, further highlights the strengths of AddShare+. FedAvg [57] achieves an SA of 70.3%, while AddShare+ closely follows with 69.8%, a difference of just 0.5%. This small margin indicates that AddShare+ effectively balances privacy and performance. Vanilla AddShare, however, struggles more on this dataset, achieving an SA of only 52.2%. FedShare (68.9%) performs comparably to AddShare+, while SCOTCH lags slightly at 68.8%. In terms of CAA, AddShare+ achieves 66.6%, very close to FedAvg's 66.7%, and again outperforms FedShare (65.3%) and SCOTCH (66.7%). The results show that even in real-world datasets, where accuracy is paramount, AddShare+ manages to maintain competitive performance while also preserving privacy.

Comparing Baselines and Encrypted Versions of AddShare and AddShare+

The introduction of encryption adds another layer to the analysis. When comparing FedAvg [57] and its RSA-encrypted counterpart, both maintain the same SA across all datasets, indicating that encryption does not affect accuracy directly but comes at a significant computational cost (as discussed below). Similarly, AddShare+ with ECIES encryption closely follows FedAvg in terms of accuracy, with only slight decreases across all datasets.

Starting again with MNIST, both FedAvg [57] and its RSA-encrypted variant achieve

the highest SA of 95.8%. In comparison, AddShare+ with ECIES encryption records an SA of 95.7%, effectively matching the performance of FedAvg despite the added privacy features. This slight difference of 0.1% confirms that the more efficient ECIES encryption scheme used in AddShare+ preserves accuracy while offering robust data protection. Vanilla AddShare, when combined with RSA encryption, experiences a noticeable drop in accuracy to 95.1%. This decline can be attributed to the heavier computational load introduced by RSA encryption, which hampers the model's ability to maintain precision during weight-sharing and reconstruction. The outcomes of these experiments are illustrated in Figure 4.2, which presents the performance of the encrypted algorithm variants on the MNIST dataset across multiple training rounds.

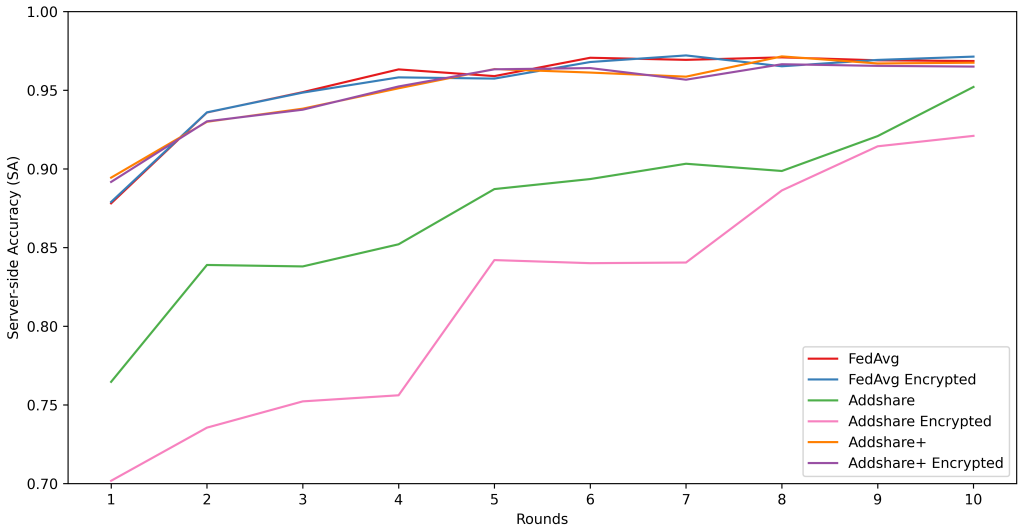

Figure 4.2: Comparison of algorithm performance on MNIST dataset per round, highlighting the effects of encryption on accuracy. The figure shows how different encryption variants impact the convergence of accuracy over training rounds.

The F-MNIST results further illustrate the strength of AddShare+ with ECIES encryption. While the RSA-encrypted version of AddShare sees a substantial drop in SA to 71.5%,

AddShare+ with ECIES achieves an SA of 78.9% (nearly matching FedAvg's [57] 79.3%). The consistency in CAA across AddShare+ (76.4%) and FedAvg [57] (76.5%) underscores that AddShare+ can maintain client-side performance even when strong encryption is applied, positioning it as a highly practical choice for privacy-sensitive applications where client accuracy must be preserved. This is shown in Table 4.5.

Table 4.5: Table showing the performance of encrypted versions of AddShare and AddShare+ against other baselines on the F-MNIST dataset

Test	Server		Client		
	SA	FLT (s)	CA	CTT (s)	ASST (s)
FedAvg	0.793	358.384	0.765	1.914	*
FedAvg Encrypted (RSA)	0.793	985.338	0.765	1.891	*
AddShare	0.743	387.922	0.762	1.782	0.893
AddShare Encrypted (RSA)	0.715	15145.941	0.765	1.800	1.566
Addshare+	0.789	337.261	0.764	1.558	0.298
Addshare+ Encrypted (ECIES)	0.789	432.197	0.764	2.772	0.375

For the CIFAR-10 dataset, the encrypted models further reveal the impact of different encryption schemes. FedAvg [57] with RSA encryption manages to retain the same SA as its unencrypted version 41.9%, but at the cost of significant computational complexity. AddShare+ with ECIES encryption records an SA of 41.2%, closely trailing FedAvg [57], while RSA-encrypted AddShare drops to 0.378, highlighting the performance hit associated with RSA. This stark contrast between ECIES and RSA suggests that while encryption is necessary for privacy, the choice of encryption scheme significantly affects model accuracy and overall system performance.

The SVHN dataset continues this trend. FedAvg [57] with RSA encryption maintains an SA of 70.2%, equivalent to its unencrypted counterpart, but again, the computational cost is substantial. AddShare+ with ECIES encryption, on the other hand, achieves an SA of 69.8%, only 0.4% lower than FedAvg [57]. The RSA-encrypted AddShare version fares much worse, dropping to 51.7%, reinforcing the conclusion that RSA encryption imposes

a significant accuracy cost on models that do not use optimized sharing and encryption techniques. This is shown in Table 4.6.

Table 4.6: Table showing the performance of encrypted versions of AddShare and AddShare+ against other baselines on the SVHN dataset

Test	Server		Client		
	SA	FLT (s)	CA	CTT (s)	ASST (s)
FedAvg	0.703	530.480	0.667	3.373	*
FedAvg Encrypted (RSA)	0.702	1587.473	0.667	3.318	*
AddShare	0.522	536.093	0.660	2.073	1.146
AddShare Encrypted (RSA)	0.517	21040.402	0.667	3.565	1.509
Addshare+	0.698	585.478	0.666	2.093	0.389
Addshare+ Encrypted (ECIES)	0.698	593.149	0.665	2.132	0.392

Conclusion: The performance of both vanilla and encrypted versions of AddShare and AddShare+ tells a clear story of balancing privacy and accuracy in FL. Vanilla AddShare+ consistently performs near or on par with FedAvg [57], demonstrating that privacy-preserving mechanisms like selective weight sharing can be integrated without significant sacrifices in accuracy. This contrasts with vanilla AddShare, which struggles in more complex datasets, highlighting the importance of optimized techniques like those in AddShare+. When encryption is introduced, AddShare+ with ECIES continues to maintain competitive accuracy across all datasets, showing that strong privacy can coexist with high performance, whereas RSA-encrypted AddShare suffers from computational overhead and reduced accuracy in complex scenarios. AddShare+ proves to be resilient, outperforming FedShare and SCOTCH in balancing privacy and accuracy, making it a forward-looking solution for privacy-sensitive FL environments.

4.7.2 Impact of Group Size and Group Formation on Performance and Time Efficiency

Group size and the method of group formation (client-side vs. server-side) play a significant role in determining the performance, time efficiency, and privacy trade-offs. The number of clients within a group (G3, G5, G10) affects both accuracy and the speed of training, while the strategy used to form these groups—whether managed by the clients themselves or by the server further impacts the FLT, ASST, and communication overhead. This section explores how these factors influence performance across four datasets: MNIST, F-MNIST, CIFAR-10, and SVHN.

Client-Side vs. Server-Side Group Formation

The results from the AddShare+ experiments provide clear evidence of the substantial impact that group formation strategy—whether client-side or server-side—has on the overall performance, time efficiency, and privacy trade-offs. Server-side group formation consistently outperforms client-side group formation in terms of FLT, thanks to its centralized and optimized approach to assigning clients to groups. For example, with SVHN, AddShare+ with server-side group formation and G10 achieves an FLT of 578.325 seconds, while client-side G10 results in a longer FLT of 598.817 seconds. This disparity highlights the efficiency advantage of server-side group formation, where the server manages the group assignments and ensures efficient communication, resulting in faster convergence times and reduced overall training time. This pattern is even more pronounced in CIFAR-10, where server-side G10 achieves an FLT of 364.764 seconds, while client-side G10 requires 424.333 seconds to complete the same task. The difference of nearly 59 seconds in FLT reflects the added complexity and inefficiencies associated with client-side group formation. In this decentralized setup, clients independently select their group members, which introduces delays and inconsistencies in communication. These delays arise because clients must communicate and coordinate with their peers without the server's centralized control, leading

to suboptimal group configurations and increased communication overhead. Consequently, client-side group formation often results in longer training times, despite offering better privacy by allowing clients to retain control over their data-sharing processes.

Further supporting these findings, the MNIST dataset shows similar trends. Server-side group formation with G10 achieves the fastest FLT at 408.412 seconds, compared to client-side G10, which records an FLT of 414.190 seconds as can be seen in Figure 4.3. Although the difference is smaller in this simpler dataset, the server-side formation still demonstrates more efficient training due to the centralized group management and reduced communication overhead. The ASST also shows consistent advantages for server-side group formation. For example, in the MNIST dataset, server-side G10 achieves an ASST of 0.023 seconds, compared to 0.052 seconds for client-side G10. This difference highlights the additional computational burden placed on clients during client-side group formation. In client-side group formation, each client autonomously selects which other clients it will share its model weights with. Some clients may be selected by multiple peers, meaning they receive and reconstruct more weight shares than others. This creates an imbalance, where certain clients have disproportionate influence on the global model, potentially skewing the aggregated updates towards their local data distribution. In contrast, server-side group formation ensures a more even distribution of influence by carefully managing the group assignments, leading to more balanced and accurate model updates.

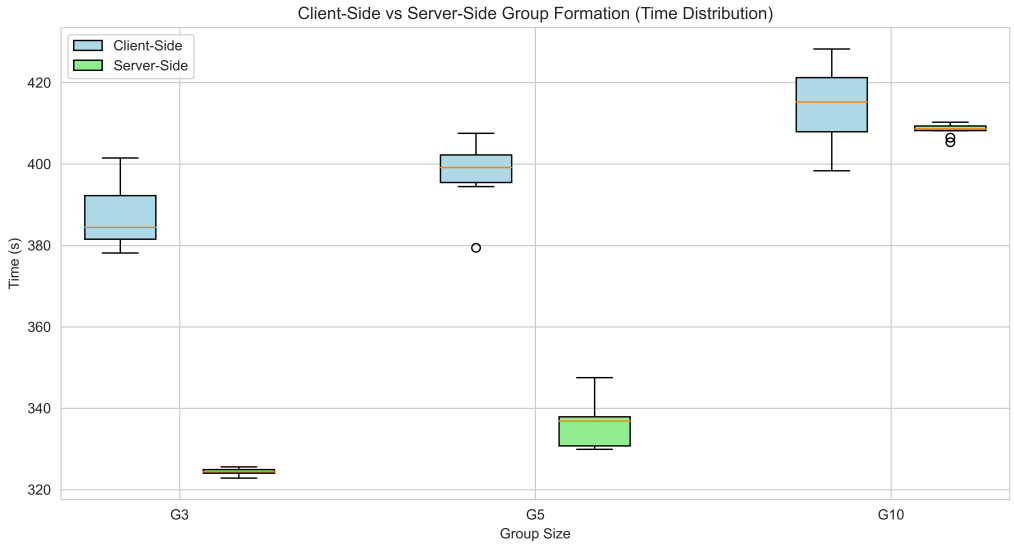

Figure 4.3: Client-Side vs. Server-Side Group Formation Time Distribution for AddShare+ on the MNIST Dataset. The plot illustrates the FLT on both the client-side (blue) and server-side (green) for three different group sizes: G3, G5, and G10.

The influence of group size on accuracy also varies based on the group formation strategy. In server-side group formation, AddShare+ achieves consistent SA across group sizes. For example, in the MNIST dataset, AddShare+ records an SA of 95.8% for G3, G5, and G10, indicating that group size does not significantly impact accuracy when groups are formed by the server. However, in client-side group formation, larger groups tend to show a slight decline in accuracy. For instance, in CIFAR-10, AddShare+ records an SA of 41.2% for G3, but this decreases slightly to 40.7% for G10. This drop in accuracy can be attributed to the imbalance created by larger groups in client-side formation, where some clients dominate the model updates due to receiving and reconstructing more weight shares than others. As group size increases, the diversity of updates decreases, leading to reduced generalization and lower accuracy. This imbalance is less pronounced in server-side formation, where the server ensures that all clients contribute evenly to the model aggregation process.

Time Efficiency and Group Size

The impact of group size on time efficiency is where smaller groups truly shine. As the number of clients in a group decreases, the overall communication rounds required for the model to converge decrease, resulting in faster training times. This is reflected in the FLT across datasets. For MNIST, AddShare+ with G3 (server-side) achieves an FLT of 324.684 seconds, compared to 408.412 seconds for G10. In CIFAR-10, the efficiency gains are even more pronounced. G3 achieves an FLT of 328.481 seconds, significantly lower than G10's 364.764 seconds. This reduction in training time for smaller groups demonstrates that, in more complex datasets, group size has a substantial impact on the speed of the FL process. Similar patterns are observed in F-MNIST, where G3 achieves an FLT of 324.616 seconds compared to G10's 415.437 seconds, and in SVHN, where G3 reduces FLT to 452.760 seconds, compared to G10's 578.325 seconds.

The ASST is another critical factor that varies with group size. Smaller groups generally benefit from reduced ASST because the burden of secret sharing is distributed across less clients. This reduction in ASST can significantly enhance the system's overall efficiency, especially in large-scale deployments. For example, in MNIST, AddShare+ with G3 achieves an ASST of 0.009 seconds, compared to 0.023 seconds for G10. A similar trend is observed across other datasets. In SVHN, G3 records an ASST of 0.011 seconds, while G10 takes 0.023 seconds. These efficiency improvements demonstrate that smaller groups are not only beneficial for reducing training time but also for optimizing the computational overhead associated with secure weight sharing. However, in datasets like CIFAR-10, the efficiency gains in ASST with smaller groups are slightly more modest. G3 achieves an ASST of 0.012 seconds, compared to 0.014 seconds for G10. While these improvements are less pronounced, they still contribute to faster overall convergence in smaller groups.

While larger groups like G10 reduce communication rounds needed for convergence, the higher per-round communication costs of secret sharing in larger groups can offset these benefits. Larger groups aggregate more updates per round, allowing the model to approach its target performance more rapidly in theory. However, the significant per-round commu-

nication cost, particularly during the secret sharing phase, undermines the speed advantage of larger groups. In contrast, smaller groups incur lower per-round communication costs, which leads to faster FLT. The reduction in communication overhead can be quantified by Equation 3.36, as discussed in Sub-section 3.8.1.4.

The trade-off between communication overhead and FLT is crucial to understand. While larger groups require fewer rounds to converge, making them suitable in environments with bandwidth constraints, they also result in longer overall training times due to the increased per-round communication cost. On the other hand, smaller groups like G3 improve overall time efficiency, particularly in complex datasets such as CIFAR-10 or SVHN, by drastically reducing the number of communication rounds and minimizing per-round costs. This makes smaller groups an ideal choice for time-sensitive or large-scale FL tasks, where faster training and resource optimization are paramount, even if it comes at the expense of a slightly higher convergence rate.

Conclusion: The analysis of group size and group formation reveals that both factors significantly influence the performance, time efficiency, and communication overhead. Server-side group formation consistently outperforms client-side formation due to its centralized approach, which leads to more efficient communication and faster convergence, particularly in larger groups. Smaller groups (e.g., G3) exhibit faster FLT and ASST due to reduced per-round communication costs, making them ideal for time-sensitive or resource-constrained environments. While larger groups (e.g., G10) require fewer communication rounds to converge, their higher per-round communication costs can undermine their potential time efficiency. The trade-off between communication overhead and FLT emphasizes that smaller groups are generally more efficient for complex datasets, such as CIFAR-10 or SVHN, where the need for faster training and resource optimization outweighs the slight increase in convergence rate. Ultimately, selecting the optimal group size and formation strategy depends on the specific goals of the FL task, balancing accuracy, efficiency, and privacy needs

4.7.3 Ablation Studies: Understanding the Impact of Key Components

In this section, we conduct a series of ablation studies to evaluate the impact of individual components of AddShare+, including selective weight sharing, percentage of weight shared and precision settings. These studies aim to isolate and analyze each element's effect on system performance, efficiency, and privacy. Through targeted modifications of specific components, we seek to gain a deeper understanding of the trade-offs inherent in the design of AddShare+, enabling us to provide more informed recommendations for real-world applications.

Precision Loss and Rounding Errors

In the AddShare+ framework, additive secret sharing is employed to ensure privacy by splitting each client's model weights into multiple shares, which are then distributed among other clients. This technique enables secure collaboration in FL without revealing individual data. However, the transformation of weights into additive shares and their subsequent reconstruction introduces potential precision loss and rounding errors, which can impact the overall accuracy of the federated model. In this section, we systematically investigate these effects.

To assess the impact of precision loss and rounding errors, we performed an ablation study using three MNIST pre-trained LeNet-5 models, each representing an individual client. The first two convolutional layers of each model were selected, and the weights were split into three additive shares for distribution. After the shares were exchanged among clients and the weights reconstructed, we compared the federated averages of the original weights with those of the reconstructed weights to identify any discrepancies. The results revealed small but measurable differences, which we attributed to the following factors:

- **Floating-point representation limitations:** In AddShare+, the splitting of weights into shares is performed using floating-point arithmetic. Floating-point numbers,

while efficient for representing real numbers in computational systems, are inherently limited by the precision of the underlying hardware. Typically, floating-point numbers are stored in a fixed number of bits (e.g., 32-bit or 64-bit floating point). This finite representation means that only a certain number of digits can be stored accurately, leading to truncation or rounding when dealing with very small or very large numbers. When the original weights are split into multiple shares, each share inherits these precision limitations. If the original weight contains more significant digits than the floating-point format can accommodate, truncation occurs. As a result, when the shares are reconstructed and summed, the final value deviates slightly from the original weight. This issue becomes more pronounced when working with more clients and weights of varying magnitudes, especially those that approach the limits of the floating-point range.

- **Cumulative rounding errors:** The additive secret sharing process requires the reconstruction of the original weights by summing the individual shares. However, repeated arithmetic operations, especially with floating-point numbers, introduce rounding errors. This phenomenon is known as cumulative rounding error, where the tiny imprecision from each floating-point operation adds up over multiple calculations. In the context of FL, this effect is particularly important because model weights are reconstructed after multiple rounds of additions across several clients. Each client generates and transmits shares, which are summed at the reconstruction stage. As the number of clients increases, the number of summations grows, amplifying the rounding errors. While the differences between the original and reconstructed weights remain small, these cumulative errors can potentially affect the final model's performance, especially when training involves large-scale models or high-precision tasks.

Trade-offs Between Privacy and Precision: The ablation study highlights an inherent trade-off in the AddShare+ framework: increasing the number of clients and, conse-

quently, the number of shares, enhances privacy by distributing knowledge of the original weights across a larger network. However, this comes at the cost of precision. The more shares generated and reconstructed, the more opportunities arise for rounding errors to accumulate. Therefore, while increasing the number of clients offers stronger privacy guarantees, it can inadvertently introduce precision loss, potentially degrading the performance of the global model. This trade-off is particularly critical in privacy-sensitive applications where both data security and model accuracy are paramount. In such cases, carefully balancing the number of shares and the precision requirements of the model is essential. Techniques to mitigate precision loss, such as adjusting the floating-point precision or the use of group formation, could be explored to minimize the impact of these errors.

Quantitative Analysis of Precision Loss: The quantitative results from our ablation study, shown in Table 4.7, demonstrate the precision loss introduced by the additive secret sharing process. For example, we observed a small deviation in the mean weight values between the original federated average and the reconstructed average. Specifically, the mean value difference was on the order of $1.689 \times 10{-}10$, while the minimum and maximum differences were $5.960{\times}10{-}8$ and $4.967{\times}10{-}8$, respectively. While these differences are extremely small and unlikely to have a significant impact on most tasks, they are still worth noting, particularly for high-precision applications where even minor deviations could affect performance. These results suggest that while the precision loss is not substantial in our experiments with just two layers of the LeNet-5 model, the cumulative impact could become more significant in more complex models or tasks requiring fine-grained accuracy. As such, the decision to implement additive secret sharing in large-scale FL systems should carefully consider the trade-off between privacy protection and precision retention.

Table 4.7: Table showing the difference in FedAvg values of weights after application of AddShare+

Metric	FedAvg	AddShare+	Difference
Mean	-0.02611662	-0.02611663	-1.68931565e-10
Minimum	-0.65246886	-0.65246880	-5.96046447e-08
Maximum	0.43009330	0.43009329	4.96705334e-08

Weight Selections Approaches

In the AddShare+ framework, one of the key challenges is to optimize the selection of model weights to be shared among clients while balancing privacy, efficiency, and model accuracy. Weight selection is crucial because sharing all weights is often leads to communication overhead which is evident in AddShare, especially in large-scale FL systems. Therefore, selecting a subset of the most relevant weights can significantly reduce transmission costs while maintaining strong model performance. In this ablation study, we explored four distinct weight selection methods, each with its own strengths and trade-offs as discussed in Section 3.8.1 for the additive secret sharing process.

1. **Random Weight Selection**: Random weight selection involves arbitrarily choosing a subset of weights to be shared among clients. While this method is straightforward to implement, it operates under the assumption that all weights are equally important. As a result, it may neglect critical weights that have a higher impact on model performance, leading to sub-optimal accuracy. The primary advantage of this method is its simplicity and minimal computational overhead, making it a feasible option in systems where speed is paramount, but model performance may not be the highest priority. However, the results from our experiments show that random selection generally yields lower SA compared to more sophisticated methods. For instance, on the CIFAR-10 dataset, random selection achieved a SA of 36.1%, significantly lower than other methods like magnitude-based or OBD selection. This

indicates that random selection may be suitable only for applications where model accuracy can be compromised for the sake of efficiency.

2. **Magnitude-based Weight Selection**: Magnitude-based selection prioritizes weights based on their absolute value, operating under the assumption that weights with larger magnitudes contribute more significantly to the model's predictions. In this method, the model evaluates the magnitude of each weight and selects those with the highest values to be shared across clients. This approach intuitively makes sense because, in neural networks, larger weights typically have a greater influence on the model's output. Our experimental results demonstrated that magnitude-based selection strikes an excellent balance between model accuracy and computational efficiency. For example, in the CIFAR-10 dataset, this method achieved a SA of 40.2%, a significant improvement over random selection. Moreover, the ASST for this method was 0.301ms, the fastest among them all, making it the most efficient in terms of both computational cost and communication overhead. Similarly, on the MNIST dataset, magnitude-based selection produced a SA of 95.7%, closely matching more complex methods such as OBD and L1 regularization, while maintaining the fastest FLT. These results suggest that magnitude-based selection is a practical and efficient solution for scenarios where both time and accuracy are critical.

3. **Optimal Brain Damage**: OBD is a pruning technique that calculates the saliency of each weight using the second-order derivative of the loss function with respect to the weights. Weights with low saliency are deemed less important and can be pruned or deprioritized in the sharing process. This method offers a more rigorous approach to weight selection by considering the impact of each weight on the model's overall performance, making it a more data-driven method than random or magnitude-based selection. In our experiments, OBD demonstrated slightly higher accuracy than the magnitude-based method, particularly in more complex datasets like CIFAR-10, where it achieved a SA of 41.3%. However, this gain in accuracy comes at a cost: the ASST for OBD was higher (0.366ms), and the FLT also increased due

to the additional computational complexity involved in calculating weight saliency. While OBD may offer some accuracy improvements, the additional computational burden might not justify its use in time-sensitive or resource-constrained environments. It is, however, a good candidate for use cases where model accuracy is of utmost importance, and computational resources are abundant.

4. **L1 Regularization**: L1 regularization adds a penalty term proportional to the absolute value of the weights in the loss function, encouraging the model to learn sparse representations. This method naturally leads to many weights becoming small or zero, making it easier to select and prioritize weights that are most influential. During weight selection, those with absolute values below a certain threshold are pruned, ensuring that only the most significant weights are shared. Similar to OBD, L1 regularization offers a more structured approach to weight selection, focusing on promoting sparsity within the model. Our results showed that L1 regularization achieved the highest accuracy in some cases, particularly on the CIFAR-10 dataset, where it reached a SA of 41.5%, narrowly surpassing OBD. Despite this, the computational overhead of L1 regularization was non-trivial. With an ASST time of 0.386ms and a relatively slower FLT, this method, like OBD, is best suited for environments where model accuracy takes precedence over computational efficiency.

When examining the performance of different weight selection methods across all datasets from the results in Table 4.8, random selection consistently demonstrated inferior performance compared to more sophisticated approaches. For instance, in the CIFAR-10 dataset, random selection achieved only 36.1% SA, significantly lower than magnitude-based (40.2%), OBD (41.3%), and L1 regularization (41.5%). This pattern held true across all datasets, with random selection showing consistently lower performance in both accuracy metrics and time efficiency measures. Among the three structured approaches, L1 regularization demonstrated marginally superior accuracy across all datasets. In CIFAR-10, L1 regularization achieved the highest SA at 41.5%, followed closely by OBD at 41.3%, while magnitude-based selection reached 40.2%. Similar patterns emerged in other

Table 4.8: Results of Weight Selection Alternatives

Dataset	Metric	Random	Magnitude	OBD	L1
CIFAR-10	SA	0.361	0.402	0.413	0.415
	CAA	0.373	0.376	0.366	0.369
	FLT (ms)	382.332	388.514	427.308	403.026
	ASST (ms)	0.490	0.301	0.366	0.386
F-MNIST	SA	0.766	0.789	0.784	0.790
	CAA	0.763	0.770	0.754	0.768
	FLT (ms)	368.323	379.560	392.636	402.919
	ASST (ms)	0.427	0.306	0.336	0.362
MNIST	SA	0.954	0.957	0.958	0.958
	CAA	0.945	0.936	0.941	0.944
	FLT (ms)	366.001	381.035	412.338	381.526
	ASST (ms)	0.414	0.297	0.274	0.425
SVHN	SA	0.677	0.699	0.702	0.703
	CAA	0.647	0.665	0.660	0.665
	FLT (ms)	492.341	514.343	569.337	553.312
	ASST (ms)	0.475	0.292	0.278	0.448

datasets, with L1 regularization maintaining a slight edge in accuracy metrics. However, this superior accuracy came at a substantial computational cost, as evidenced by the FLT. The FLT reveal a critical advantage of magnitude-based selection. In the CIFAR-10 dataset, magnitude-based selection achieved an FLT of 388.514 ms, significantly faster than both OBD (427.308 ms) and L1 regularization (403.026 ms). This efficiency advantage was consistent across all datasets, with magnitude-based selection consistently requiring less computational time while maintaining comparable accuracy. For instance, in the MNIST dataset, while L1 regularization and OBD achieved 95.8% SA, magnitude-based selection reached 95.7%, a negligible difference in practical terms, while maintaining substantially lower FLT.

The ASST further reinforces the efficiency advantage of magnitude-based selection. For CIFAR-10, magnitude-based selection recorded an ASST of 0.301 ms, compared to 0.366 ms for OBD and 0.386 ms for L1 regularization. This significant reduction in processing

time, coupled with competitive accuracy metrics, makes magnitude-based selection particularly attractive for practical implementations. These findings led to the selection of magnitude-based weight selection as the preferred approach for AddShare+. While L1 regularization and OBD demonstrated marginally better accuracy in some cases, the substantial improvements in computational efficiency offered by magnitude-based selection, combined with its competitive accuracy performance, make it the most practical choice for real-world federated learning implementations. This decision prioritizes the practical benefits of reduced computational overhead while maintaining robust model performance, rather than pursuing minimal accuracy improvements at the cost of significantly increased computational burden.

Percentage of Weights Sampled in AddShare+

In this study, we examined the impact of varying the percentage of model weights shared during the FL process (denoted as p) on the performance of the AddShare+ framework. The parameter p represents the proportion of the model's weights selected for sharing between clients in the FL setup, and we explored its effect by evaluating several values: 15%, 25%, 35%, and 45%. The goal was to identify the optimal value of p that balances communication efficiency, privacy, and model accuracy.

Rationale for Weight Sampling: Selective weight sharing is integral to AddShare+ because it reduces the communication overhead and computation time by only transmitting a subset of the model's weights instead of the entire set. This is particularly useful in resource-constrained environments, such as edge devices or distributed systems with limited bandwidth. Sharing fewer weights decreases the volume of data transmitted, but care must be taken to ensure that this reduction does not come at the expense of model accuracy or privacy guarantees. The choice of p directly influences this trade-off: smaller values of p reduce the data sent but may exclude important weights, while larger values increase communication costs without significant accuracy gains.

Experimental Setup and Results: To analyze the sensitivity of AddShare+ to different values of p, we conducted experiments on four datasets: CIFAR-10, F-MNIST, MNIST, and SVHN. For each dataset, we evaluated the performance of AddShare+ across the four selected percentages of weight sharing. The metrics used for evaluation included SA, CAA, FLT, ATT and ASST. The results of these experiments are summarized in Table 4.9.

Table 4.9: Results of AddShare+ with varying sharing percentages of p whereby p is the quantity of nodes to share.

Experiment	p	**SA**	**CAA**	**FLT (ms)**	**ATT/ASST (ms)**
CIFAR10	0.15	0.419	0.382	146.269	1.387 / 0.029
	0.25	0.419	0.381	144.672	1.398 / 0.045
	0.35	0.419	0.382	156.188	1.418 / 0.053
	0.45	0.419	0.382	147.347	1.400 / 0.054
F-MNIST	0.15	0.793	0.750	138.485	1.637 / 0.025
	0.25	0.793	0.750	139.107	1.737 / 0.026
	0.35	0.793	0.748	141.115	1.839 / 0.037
	0.45	0.793	0.750	141.170	1.870 / 0.044
MNIST	0.15	0.958	0.939	139.831	1.594 / 0.030
	0.25	0.958	0.938	139.738	1.576 / 0.036
	0.35	0.958	0.938	140.958	1.587 / 0.038
	0.45	0.958	0.938	141.252	1.597 / 0.042
SVHN	0.15	0.702	0.684	241.748	1.806 / 0.030
	0.25	0.702	0.685	240.037	1.806 / 0.047
	0.35	0.701	0.685	241.316	1.806 / 0.065
	0.45	0.702	0.685	240.754	1.826 / 0.055

For the CIFAR-10 dataset, we found that the SA remained consistent at 41.9% across all tested values of p. However, the $p = 0.25$ configuration achieved the fastest FLT (144.672 ms) and the lowest CAA (38.1%), which suggests that this level of weight sharing strikes the best balance between communication efficiency and model performance. Although sharing 15% of the weights further reduced FLT, the corresponding CAA (38.2%) was only marginally higher, and the additional reduction in communication overhead did not justify the potential risks to privacy from sharing too few weights.

For the F-MNIST dataset, $p = 0.15$ achieved the best balance between accuracy and efficiency, with an SA of 79.3%, a CAA of 75%, and an FLT of 138.485 ms. This suggests that for certain datasets, sharing fewer weights can still yield optimal results, as long as the shared weights capture the most significant information.

In the MNIST dataset, all values of p resulted in the same SA of 95.8%, demonstrating the robustness of the framework across different percentages of weight sharing. However, $p = 0.25$ again stood out by achieving a slightly lower CAA (93.8%) compared to $p = 0.15$ (93.9%), while maintaining the fastest FLT (139.738 ms). This reinforces the finding that $p = 0.25$ offers a reliable balance of accuracy and efficiency.

For the SVHN dataset, the SA reached 70.2% across $p = 0.15$, $p = 0.25$, and $p = 0.45$, demonstrating that accuracy remained consistent regardless of the percentage of weights shared. However, $p = 0.25$ again achieved the lowest FLT (240.037 ms) while maintaining comparable CAA values with the other configurations. This suggests that while higher values of p do not offer substantial accuracy improvements, they incur additional communication and computational costs.

Analysis and Interpretation: The results from these experiments indicate that sharing approximately 25% of the model's weights ($p = 0.25$) generally provides the best trade-off between communication overhead, computational efficiency, and model performance. Across all datasets, $p = 0.25$ consistently resulted in lower FLT while maintaining competitive SA and CAA values compared to larger percentages of weight sharing. This finding is particularly important for real-world FL applications where computational resources and bandwidth are often constrained. By sharing only 25% of the weights, AddShare+ can significantly reduce the communication and processing load without sacrificing the accuracy of the global model. This makes $p = 0.25$ a highly efficient and practical configuration for scenarios where fast iteration and resource optimization are critical, such as in IoT devices or distributed edge systems. However, it is essential to note that sharing too few weights, as seen with $p = 0.15$, could undermine the privacy benefits of the additive secret sharing

mechanism. While the reduction in communication overhead is attractive, the smaller set of shared weights may not provide sufficient noise or data obfuscation, increasing the risk of model inversion or other inference attacks. On the other hand, sharing more weights ($p = 0.35$ or $p = 0.45$) results in diminishing returns in terms of accuracy gains while imposing higher communication and computational costs.

The ablation study on the percentage of weights sampled in AddShare+ demonstrates that $p = 0.25$ is an optimal configuration across a wide range of datasets and scenarios. It balances the need for communication efficiency, computational speed, and model performance, making it well-suited for practical FL applications. Moreover, this configuration maintains strong privacy guarantees by ensuring a sufficient proportion of the model's weights are shared, without unnecessarily increasing the communication burden.

Summary of Ablation Findings

The ablation studies conducted for AddShare+ reveal that each component (precision handling, selective weight sharing, and weight selection percentage) significantly impacts the balance between privacy, accuracy, and efficiency in FL. Precision loss and rounding errors, though minimal, emerge due to floating-point limitations during secret sharing, emphasizing a trade-off between privacy and accuracy. The selective weight-sharing mechanism, particularly the magnitude-based method, proved effective in minimizing communication overhead while maintaining competitive model accuracy, offering a practical solution for resource-constrained environments. More advanced methods, such as OBD and L1 regularization, provided slight accuracy gains but at the cost of increased computational complexity. The study on weight-sharing percentages found that sharing 25% of the model weights strikes the best balance between communication efficiency and model performance, achieving consistent accuracy improvements across datasets without excessive overhead. These findings underscore the flexibility and scalability of AddShare+, making it adaptable to privacy-sensitive applications where computational resources or communication bandwidth may be limited. Overall, AddShare+ demonstrates a careful balance between

privacy preservation and system performance, offering valuable insights for optimizing FL systems in real-world deployments.

4.8 Conclusion

This chapter provided an in-depth evaluation of AddShare and AddShare+, assessing their effectiveness in balancing accuracy, privacy, and efficiency within FL frameworks. Through comprehensive testing on various datasets, we observed that AddShare+ maintains competitive performance while significantly reducing communication overhead and preserving privacy through selective weight sharing and encryption techniques. The experiments highlighted the influence of group size and group formation on system performance, with larger groups and server-side formation offering better time efficiency. Furthermore, the ablation studies provided critical insights into the impact of key components, such as precision handling and weight selection methods, illustrating the trade-offs between privacy, computational efficiency, and model performance. Overall, the findings confirm that AddShare+ is a robust and adaptable solution for real-world, privacy-sensitive FL environments, capable of addressing the challenges of both resource-constrained and high-performance applications.

Chapter 5

Case Study: Crop Yield Prediction

5.1 Introduction

Crop yield prediction plays a pivotal role in modern agriculture, serving as a foundation for food security and economic stability [43]. Accurate prediction of crop yields helps farmers make informed decisions about planting, fertilizing, and harvesting, thereby optimizing resource usage and maximizing productivity [84]. Traditional methods of crop yield prediction, which often rely on historical data and basic statistical techniques, are increasingly seen as inadequate due to their inability to account for the complex and dynamic nature of agricultural ecosystems [38]. With advancements in technology, data-driven approaches such as ML have emerged as powerful tools to enhance the accuracy and reliability of crop yield predictions [5].

Among various ML techniques, neural networks have shown significant promise in crop yield prediction due to their ability to model non-linear relationships and handle large volumes of data [22]. Neural networks, particularly deep learning models, can learn intricate patterns from diverse datasets, including weather conditions, soil properties, crop characteristics, and Unmanned Aerial Vehicles (UAV) imagery [22]. The process typically involves training a neural network on historical data, where the model learns to associate

input features with crop yield outcomes. Once trained, the neural network can predict future yields based on new input data. For instance, CNNs can be used to analyze UAV imagery to assess plant health and estimate yields [63], while Recurrent Neural Networks (RNN)s can handle sequential data such as time-series weather information [46]. By leveraging these sophisticated models, farmers and agricultural stakeholders can achieve more accurate and timely predictions, ultimately leading to better decision-making and increased agricultural productivity.

5.2 Agricultural Data Privacy Challenges

While aggregated farming data holds immense potential for improving agricultural productivity and sustainability, farmers are often hesitant to share their operational data due to several legitimate business and economic considerations [32, 90]. Crop yield data directly reflects a farm's productivity and profitability. When this information becomes available to financial institutions, it can significantly impact farmers' negotiating positions for loans, insurance rates, and land valuations [78]. Banks and lenders may use historical yield data to adjust risk assessments, potentially affecting interest rates or loan terms based on perceived operational performance [103]. Detailed yield information, combined with farming practices and resource utilization data, represents proprietary operational knowledge that farmers have developed over generations. Sharing this information could compromise their competitive edge in local agricultural markets, particularly in regions where farmers compete for land leases or market share [90].

Property valuations and lease negotiations are heavily influenced by demonstrated yield productivity [91]. Farmers may face disadvantageous positions in land transactions if detailed yield records become accessible to real estate investors or competing agricultural operations looking to expand their holdings [32]. Access to granular yield data could enable commodities traders and market speculators to make more informed predictions about local supply levels, potentially leading to price manipulations that disadvantage

producers [16]. This is particularly concerning in regions where a small number of farms contribute significantly to local market supply [78].

The confidentiality concerns surrounding agricultural data make the sector an ideal candidate for demonstrating the practical value of privacy-preserving FL approaches like AddShare and AddShare+ [103]. By enabling collaborative model training without requiring direct data sharing, these protocols address farmers' privacy needs while still allowing the agricultural sector to benefit from ML insights derived from broader data patterns [100]. The success of AddShare+ in maintaining both high predictive accuracy and robust privacy protections in this case study demonstrates its potential for addressing similar challenges in other privacy-sensitive domains.

5.3 Importance of Crop Yield Prediction

Accurate crop yield prediction is a critical endeavor that holds profound implications for food security and the agricultural economy. Reliable yield estimates play a pivotal role in ensuring adequate food production to meet the ever-increasing demands of a growing global population. Furthermore, crop yields exert a direct influence on the economic well-being of farmers, agricultural communities, and nations, as they shape commodity prices, trade dynamics, and overall economic stability. Traditional methods of yield prediction, which heavily rely on farmers' personal experiences and local observations, are often uncertain, inconsistent and susceptible to inaccuracies [68]. These approaches fail to capture the intricate interplay between various factors affecting crop yields, such as climate change, soil degradation, and weather variability, ultimately leading to suboptimal decision-making and inefficient resource allocation. Consequently, there is an urgent need for reliable and data-driven crop yield prediction methodologies that can capture the complex interplay between these variables and provide actionable insights. Precise crop yield predictions offer numerous benefits to various stakeholders in the agricultural sector. For farmers, accurate yield estimates can inform critical decisions regarding crop selection, planting schedules,

and the judicious application of inputs such as fertilizers, irrigation, and pesticides. This not only empower farmers and agricultural organizations to make well-informed decisions regarding crop selection, planting schedules, and the judicious application of inputs such as fertilizers, irrigation, and pesticides but also maximizes crop productivity and profitability [38, 43, 84]. This promotes sustainable agricultural practices, contributing to long-term environmental and economic sustainability.

5.4 Dataset

5.4.1 AreaX.O

The dataset was gathered from a cutting-edge agricultural research facility managed by AreaX.O, which covers approximately 1,866 acres. This facility is outfitted with advanced technologies such as IoT devices, 4G/5G networking, GPS, and specialized agricultural sensors. The primary crop cultivated is corn, with planting commencing in early June and harvesting ending in late September. The Ottawa Smart Farm (OSF) was established at the AreaX.O site in 2019, with the goal of demonstrating the advantages of new data-driven technologies to farmers in Eastern Ontario and facilitating their adoption. This research utilized two types of data:

(a) **Yield data**: This represents the continuous measurement of the final yield obtained at the end of the season
(b) **Field orthomosaics**: These are composite images created by stitching together numerous photos taken by a drone over multiple flights to depict a larger area.

In 2021, the OSF producer grew corn on four demonstration fields and two additional fields within the Ottawa Smart Farm. The OSF collaborates with a diverse group of partners, ranging from local producers to global agricultural technology companies such as Microsoft, FarmersEdge, and Trimble. Table 5.1 lists the various fields and their respective sizes. Given the extensive volume of the datasets collected, only five fields were utilized for

this case study, specifically fields 1, 2, 3, 4, and 9. Fields 1, 2, 3, and 4 were employed to simulate four FL clients, while field 9 served as the testing data on the FL central server.

Table 5.1: Size of fields in acres

Field	Area in acres
1	3.38
2	4.67
3	3.55
4	2.07
9	1.43
11	48.08

5.4.2 Yield Data

Yield data were collected using a John Deere S660 combine harvester equipped with Trimble's yield monitoring and GPS systems. Trimble an American hardware and software company. The yield measurements, recorded in bushels per acre (bu/ac), were captured at a frequency of 1Hz during harvesting. The parameters measured in this data includes:
Dry Yield: Measured in bushels per acre (bu/ac), representing the weight of the grain after accounting for moisture.
Moisture Content: Measured as a percentage, indicating the moisture level of the grain at the time of harvest.
Speed: The operational speed of the harvester, recorded in miles per hour (mph)
Swath Width: The width covered by the harvester, recorded in feet (ft).
Field Boundaries: Geospatial data outlining the field areas, recorded using GPS coordinates.

The data collected by Trimble was stored in shapefiles, a geospatial vector data format utilized in Geographic Information Systems (GIS) softwares. This shapefiles are actually a collection of four different files:

(a) *DryYield.shp*: Holds the geometric data for the point features.
(b) *DryYield.dbf*: Stores the associated attribute data, such as yield values.
(c) *DryYield.shx*: A positional index file that allows for quicker data access.
(d) *DryYield.prj*: It includes essential projection information like the coordinate system, units, and datum used, necessary for accurate rendering and analysis of the shapefile.

5.4.3 UAV Imagery

UAVs flights, operated by Indro Robotics, using an Auto EVO 2 drone initially, later replaced by a DJI M210 drone equipped with RGB cameras were used to capture high-resolution imagery of the fields. And the cameras used were Mica Sense Red-Edge M and Zenmuse X4S. UAV flights were conducted throughout the 2021 growing season to monitor crop health and development. The UAVs captured both Red-Green-Blue (RGB) and multispectral images. The multispectral imagery included bands such as Blue, Near-Infrared (NIR), Red Edge, Green, and Red, essential for analyzing plant health and detecting stress. UAV flights were scheduled bi-weekly, starting from the pre-planting stage in late May and continuing until post-harvest in early October. The drones flew at an altitude of 50 meters, covering approximately 45 acres per flight. The raw images captured by UAVs were processed by a third party for the purpose of generating orthomosaics, which are composite images that provide a comprehensive view of the fields. The fields are shown in Figure 5.1

Figure 5.1: Satellite imagery of the fields utilized for federated learning training. Fields 1, 2, 3, and 4 were used to represent four federated learning clients

5.4.3.1 Tiles Generation

The geospatial data from yield monitoring and UAV imagery were processed using GeoPy, a Python library for geospatial analysis. This included transforming shapefiles to ensure consistent coordinate reference systems. The yield data and UAV imagery were merged into a unified geospatial dataframe using GeoPandas. To facilitate manageable data processing, the large orthomosaic images from UAV flights were downsampled into smaller tiles, with each tile representing a 9x9-meter area of the field, aligned with the spatial resolution of the yield data points with a resolution of 512 x 512 x 3 as shown in Figure 5.2. These tiles were generated using the rasterio and gdal libraries in Python. Finally, the tile data was fused with yield data to create a comprehensive dataset for model training, ensuring spatial accuracy and consistency between the imagery and yield data.

Figure 5.2: A figure of a portion of Field 1, where 9x9 meter square polygons were extracted for model training.

5.5 Experimental Setup

5.5.1 Model architecture & Training Configuration

The 2D CNN model architecture used in this case study is adapted from Bhavesh et al [13]. This architecture was chosen due to its effectiveness in extracting spatial features from this UAV imagery dataset. The 2D CNN model processes input images of size 512 x 512 pixels with three color channels (RGB) and comprises two primary convolutional blocks followed by fully connected layers. The first convolutional block includes a convolutional layer with 32 filters of size 3x3, a ReLU activation function, a max-pooling layer with a pool size of 2x2, and a dropout layer with a dropout rate of 0.25 to prevent overfitting. The second convolutional block contains a convolutional layer with 64 filters of size 3x3, a ReLU activation function, a max-pooling layer with a pool size of 2x2, and a dropout layer with a dropout rate of 0.25. Following these blocks, the model has a flattening layer to convert the 2D matrix data to a 1D vector, a dense layer with 128 units and a ReLU activation function, a dropout layer with a dropout rate of 0.5, and an output layer with a

single neuron and a linear activation function, suitable for regression tasks to predict the yield. The training configuration involved using the Mean Squared Error (MSE) as the loss function, the Adam optimizer for its efficiency in handling large datasets and high-dimensional parameter spaces, a learning rate of 0.001, a batch size of 32, and 50 epochs of training.

5.5.2 Evaluation

To evaluate the performance of our 2D CNN model in predicting corn yield in a privacy preserving FL setting, we used three common regression metrics: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE). Each of these metrics provides a different perspective on the accuracy and reliability of the model's predictions.

5.5.2.1 Root Mean Squared Error

RMSE is another commonly used metric that measures the average magnitude of the errors. Unlike MAE, RMSE gives higher weight to larger errors by squaring the differences between the predicted and actual values before averaging them. This makes RMSE more sensitive to outliers.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.1}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

5.5.2.2 Mean Absolute Percentage Error

MAPE measures the average absolute percentage error between the predicted and actual values. It provides a percentage error, making it easier to interpret and compare across

different datasets or models. However, MAPE can be problematic if any actual values are zero or close to zero.

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{5.2}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

Each of these metrics provides valuable insights into the model's performance. MAE gives a clear measure of average error, RMSE emphasizes larger errors, and MAPE provides an easy-to-understand percentage error. By analyzing these metrics together, we can gain a comprehensive understanding of how well the 2D CNN model predicts corn yield.

5.5.2.3 Mean Absolute Error

MAE quantifies the typical absolute deviation between forecasted values and true values by calculating the mean of the absolute differences, disregarding whether the predictions are overestimated or underestimated. This metric offers a transparent assessment of how proximate the predictions are to the actual observations.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{5.3}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

5.6 Results and Discussion

This case study evaluates the performance of AddShare, AddShare+, FedAvg, SCOTCH, and FedShare, in the context of predicting corn crop yields using UAV imagery and yield data from the AreaX.O dataset. The models were assessed using key metrics such as MAE,

Table 5.2: Performance comparison of FL models in crop yield prediction using UAV imagery and yield data. The table shows metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), Federated Learning Time (FLT), Average Mean Absolute Error (AMAE), Average Root Mean Squared Error (ARMSE), Average Mean Absolute Percentage Error (AMAPE), Average Training Time (ATT) and Average Secret Sharing Time (ASST) across different models and configurations, with a focus on privacy-preserving mechanisms and group formation strategies (G2). The FLT for SCOTCH is marked with an asterisk (*) as it could not be calculated due to the model's architecture

Test	Server				Client				
	MAE	RMSE	MAPE	FLT	AMAE	ARMSE	AMAPE	ATT	ASST
FedAvg	39.793	71.542	39.032	330.143	39.088	66.228	59.811	68.830	N/A
FedAvg Encrypted	39.808	71.557	39.023	334.120	40.148	67.354	60.378	70.000	N/A
SCOTCH	104.041	125.023	83.844	*	34.309	68.366	35.715	65.810	3.916
FedShare	103.073	124.172	83.377	352.469	47.500	73.894	62.941	68.759	0.317
AddShare	53.146	82.244	48.232	360.047	41.663	68.535	60.267	69.415	7.503
AddShare Encryption	53.14	82.24	48.23	365.05	42.25	69.2	60.8	69.39	7.498
AddShare Server - G2	53.138	82.23	48.24	370	41.6	68.54	60.3	69.4	7.503
AddShare Server Encryption - G2	53.137	82.233	48.233	368.5	42.3	69.22	60.85	69.42	7.505
AddShare Client - G2	53.143	82.245	48.225	372.8	41.65	68.55	60.35	69.5	7.499
AddShare Client Encryption - G2	53.141	82.241	48.229	370.1	42.4	69.25	60.87	69.55	7.502
AddShare+	39.986	71.678	39.119	334.628	39.862	66.972	59.827	69.010	1.097
AddShare+ Encryption	40.043	71.724	39.125	336.729	39.862	66.972	59.827	69.508	0.991
AddShare+ Server - G2	39.823	71.573	39.012	331.500	39.721	66.829	59.836	68.469	0.359
AddShare+ Server Encryption - G2	39.823	71.573	39.012	331.601	39.721	66.829	59.836	68.519	0.359
AddShare+ Client - G2	39.975	71.715	38.973	363.755	39.650	66.774	59.738	68.381	0.363
AddShare+ Client Encryption - G2	39.595	71.330	39.175	365.342	39.783	66.907	59.701	68.737	9.448

RMSE, MAPE, FLT, and ASST. These metrics provide insights into both the predictive performance and the computational efficiency of the models, particularly when incorporating privacy-preserving mechanisms. The results obtained are shown in Table 5.2

5.6.1 Predictive Performance

From the results, AddShare+ consistently demonstrated superior performance, achieving the lowest MAE (39.986), RMSE (71.678), and MAPE (39.119) when compared to SCOTCH and FedShare, thus indicating its capability to generate highly accurate predictions while incorporating privacy-preserving mechanisms. FedAvg in terms of performance

(MAE: 39.793, RMSE: 71.542) achieves overall best results but lacks the enhanced privacy features inherent in AddShare+, underscoring the advantage of the latter in privacy-sensitive environments.

Models such as SCOTCH and FedShare performed considerably worse in terms of performance, with SCOTCH reporting the highest error metrics (MAE: 104.041, RMSE: 125.023, MAPE: 83.844). This difference suggests that these models are less capable of handling the complexities associated with geospatial and UAV-based data, potentially due to less sophisticated FL mechanisms. AddShare outperforms these models but still lags behind AddShare+, highlighting the value of the additional optimizations and encryption mechanisms introduced in AddShare+ for improving prediction accuracy.

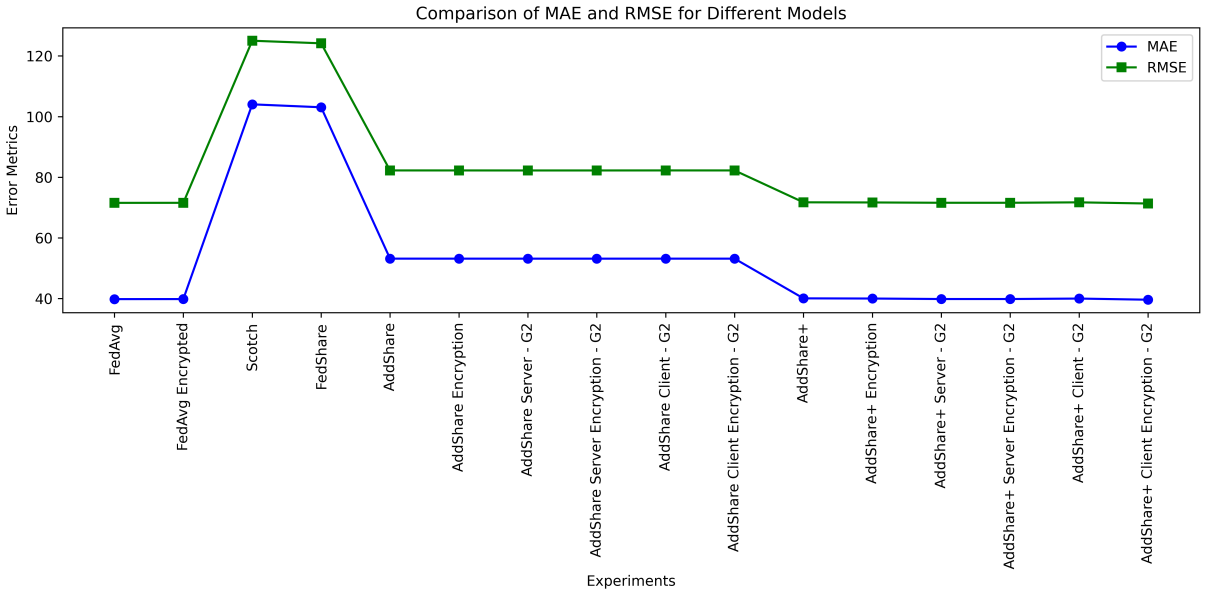

Figure 5.3: Comparison of Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) Across Different Experiments in Crop Yield Prediction.

5.6.2 Time Efficiency and Communication Overhead

In terms of FLT, AddShare+ with server-side G2 group formation achieved a notable improvement, recording an FLT of 331.500 seconds. This is significantly faster compared to other configurations, including FedShare (352.469 seconds) and AddShare (360.047 seconds). Furthermore, it can be observed that AddShare+ with server-side G2 group with encryption follows closely with the second fastest FLT of 331.601 seconds highlighting how the addition of a second layer of protection to AddShare+ only slightly alters its efficiency. Although client-side group formation, while slightly less efficient in terms of FLT (363.755 seconds), offers enhanced privacy protections, as clients maintain control over the weight-sharing process.

The ASST remained relatively low for AddShare+ in both server-side (0.359 seconds) and client-side configurations (0.363 seconds), demonstrating the framework's ability to handle secure data sharing efficiently. Although encryption slightly increased the ASST across board, some difference were substantial (e.g., server-side encryption: 0.359 seconds vs. client-side encryption: 9.448 seconds), affirming that privacy does come at a significant cost when clients form their own groups.

5.6.3 Practical Implications and Insights

The results from this case study clearly demonstrate that AddShare+ offers the best balance between accuracy, time efficiency, and privacy when operating in a FL environment. The strong predictive performance of AddShare+, even with privacy-preserving mechanisms such as encryption, underscores its suitability for privacy-sensitive applications like precision agriculture. Compared to FedAvg, AddShare+ provides nearly identical accuracy but with the added advantage of robust privacy protections, making it highly relevant in real-world agricultural contexts where data security is crucial.

Moreover, the choice of group size (G2) and group formation strategy (server-side vs. client-side) plays a significant role in the overall performance. Server-side group formation

leads to faster convergence and lower FLT, making it an ideal choice for time-sensitive applications. However, client-side group formation offers a slight trade-off in efficiency for enhanced privacy, which can be crucial in scenarios where data control and confidentiality are paramount.

5.7 Summary

The results highlight AddShare+ as the most effective model for crop yield prediction in a FL setting, particularly in terms of balancing accuracy, efficiency, and privacy. The G2 group formation provides an optimal setup for small-scale FL tasks, and the framework's ability to maintain low communication overhead, even with encryption, makes it highly scalable. AddShare+ successfully demonstrates the feasibility of applying FL to real-world agricultural applications, providing both high predictive accuracy and strong privacy protections without significantly compromising performance.

Chapter 6

Conclusion

In an era where data privacy and security are paramount, the advancement of privacy-preserving technologies in ML is crucial. This thesis has addressed these concerns by introducing AddShare and AddShare+, innovative protocols designed to enhance privacy-preserving FL. The primary goal has been to strike an optimal balance between maintaining data privacy and achieving high model performance. Through a comprehensive experimental evaluation, AddShare+ stood out by demonstrating significant improvements over existing methods, highlighting its potential for widespread application in various domains. This concluding chapter summarizes the key contributions, addresses the limitations, and outlines future research directions to further advance the field.

6.1 Summary of Contributions

This thesis presented a comprehensive exploration of privacy-preserving FL through the development and rigorous evaluation of the AddShare and AddShare+ protocols. The contributions of this research are multifaceted and span several key areas. The initial chapter lays the groundwork by articulating the increasing importance of data privacy in ML, especially in FL settings where data is distributed across multiple clients. The chapter

outlines the primary research question and objectives, establishing the need for a protocol like AddShare+ that can safeguard privacy without compromising model performance and the efficiency of the FL process. We introduced AddShare+, a novel protocol designed to enhance privacy-preserving FL. At the core of AddShare+ is the innovative application of selective additive secret sharing, which forms the bedrock for its privacy-preserving capabilities. This protocol ingeniously integrates ECIES to secure data exchanges and protect sensitive information throughout the FL process. This approach addresses the critical need for balancing data privacy with the performance of FL models.

Extensive experimental evaluations were conducted using a diverse array of benchmark datasets, including CIFAR-10, MNIST, F-MNIST, and SVHN. These experiments consistently demonstrated that AddShare+ outperforms traditional FL methods, such as FedAvg, as well as other privacy-preserving baselines. The protocol achieves notable improvements in both accuracy and efficiency, underscoring its effectiveness and robustness. A particularly insightful component of this thesis is the detailed ablation study, which meticulously investigates the impact of different weight-sharing strategies. This study reveals that sharing 25% of model weights (p=0.25) provides an optimal balance between maintaining high accuracy and minimizing communication overhead, while still preserving robust data privacy. This selective weight-sharing approach is pivotal in reducing the overall computational and communication costs associated with FL, making the protocol more scalable and practical for real-world applications. Moreover, in comparison to other additive secret sharing privacy-preserving FL methods, AddShare+ stands out due to its streamlined architecture that does not necessitate the use of multiple aggregating servers. This design choice ensures that AddShare+ makes little to no changes to the original FL architecture, thereby simplifying its integration and deployment in existing systems. This aspect significantly enhances the practicality and applicability of AddShare+, as it avoids the complexities and overheads associated with managing multiple servers, which are typically required in other privacy-preserving schemes. Further enhancing the practical relevance of this work, the thesis demonstrates the application of AddShare+ to a real-world case study focused on crop yield prediction. This case study not only exemplifies

the protocol's versatility but also highlights its potential for significant impact in critical areas such as agriculture, where accurate predictions and data privacy are paramount. The encrypted variant of AddShare+, particularly the configuration involving Server-G2, emerges as a highly effective solution. This underscores the protocol's capability to address the dual challenges of maintaining data privacy and ensuring model performance in FL environments.

6.2 Limitations and Future Works

While the AddShare+ protocol has demonstrated notable success in enhancing privacy-preserving FL, there are several areas that require further exploration. One key limitation is the evaluation of the protocol on static datasets. In real-world applications, data is often dynamic and evolves over time. Future research should focus on extending AddShare+ to support incremental learning, where models can adapt to new data without retraining from scratch. Another area for improvement is the computational burden introduced by the encryption and decryption processes, which may become a bottleneck in resource-limited environments, such as mobile or IoT devices. Optimizing these processes, potentially through the use of lightweight encryption techniques or hardware accelerators, could enhance the protocol's applicability. In addition to these points, given the observed trade-offs between privacy and precision, future work could explore strategies to mitigate precision loss in AddShare+. One potential approach is to employ higher precision floating-point formats (e.g., 128-bit precision), which can accommodate more significant digits and reduce rounding errors. However, this would come at the cost of increased computational overhead and memory usage. Another strategy could involve adaptive precision techniques, where only a subset of the model's most critical weights are shared with higher precision, while less important weights are shared with lower precision. This would reduce the overall computational burden while preserving the precision of the most influential parts of the model.

Furthermore, while the protocol has been tested on homogeneous datasets, many real-world FL applications involve heterogeneous data distributions. Future work should investigate the performance of AddShare+ under these conditions to ensure its robustness across different domains. Finally, combining AddShare+ with other privacy-preserving techniques, such as DP or HE, could further strengthen its privacy guarantees. Collaborative efforts with industry partners for large-scale real-world deployments would provide valuable insights and drive the refinement of the protocol. By addressing these limitations through targeted future research, the full potential of AddShare+ in diverse, privacy-sensitive applications will be realized.

References

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A Survey on Homomorphic Encryption Schemes: Theory and Implementation, October 2017. arXiv:1704.03578 [cs].

[2] Meenakshi Aggarwal, Vikas Khullar, Nitin Goyal, Abdullah Alammari, Marwan Ali Albahar, and Aman Singh. Lightweight federated learning for rice leaf disease classification using non independent and identically distributed images. *Sustainability*, 15(16), 2023.

[3] Jisu Ahn, Younjeong Lee, Namji Kim, Chanho Park, and Jong Beom Jeong. Federated learning for predictive maintenance and anomaly detection using time series data distribution shifts in manufacturing processes. *Sensors (Basel, Switzerland)*, 23, 2023.

[4] Bernard Atiemo Asare, Paula Branco, Iluju Kiringa, and Tet Yeap. AddShare+: Efficient Selective Additive Secret Sharing Approach for Private Federated Learning. Submitted to the 11th IEEE International Conference on Data Science and Advanced Analytics (DSAA) Special Session on Private, Secure, and Trust Data Analytics (PSTDA), 2024.

[5] Authors. Optimizing crop yield prediction: Data-driven analysis and machine learning modeling using usda datasets. *Current Agriculture Research Journal*, 2023.

[6] Tomisin Awosika, Raj Mani Shukla, and Bernardi Pranggono. Transparency and privacy: the role of explainable ai and federated learning in financial fraud detection. *IEEE Access*, 2024.

[7] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International conference on artificial intelligence and statistics*, pages 2938–2948. PMLR, 2020.

[8] James Henry Bell and Bonawitz et al. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[9] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret (extended abstract). In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 251–260, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[10] Cosmin I Bercea, Benedikt Wiestler, Daniel Rueckert, and Shadi Albarqouni. Federated disentangled representation learning for unsupervised brain anomaly detection. *Nature Machine Intelligence*, 4(8):685–695, 2022.

[11] Bernard Atiemo Asare, Paula Branco, Iluju Kiringa, and Tet Yeap. AddShare: a Privacy-Preserving Approach for Federated Learning. *International Workshop on Data Privacy Management, DPM 2023.*, 2023.

[12] Kai Bian and Hong Zheng. Fedavg-dwa: A novel algorithm for enhanced fraud detection in federated learning environment. *2023 4th International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, pages 13–17, 2023.

[13] Bhavesh Singh Bisht, Iluju Kiringa, and Tet Yeap. Corn yield prediction using spatial-temporal data and deep learning. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pages 1792–1798, 2023.

[14] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1175–1191, New York, NY, USA, October 2017. Association for Computing Machinery.

[15] Nader Bouacida and Prasant Mohapatra. Vulnerabilities in federated learning. *IEEE Access*, 9:63229–63249, 2021.

[16] Kelly Bronson. Looking through a responsible innovation lens at uneven engagements with digital farming. *NJAS - Wageningen Journal of Life Sciences*, 90-91:100294, 2019.

[17] D Brown. Standards for efficient cryptography, sec 1: elliptic curve cryptography. *Released Standard Version*, 1, 2009.

[18] Xiaoyu Cao and Neil Zhenqiang Gong. Mpaf: Model poisoning attacks to federated learning based on fake clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3396–3404, 2022.

[19] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Provably secure federated learning against malicious clients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6885–6893, May 2021.

[20] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *BMC medical genomics*, 11:3–12, 2018.

[21] Ashish Choudhury and Arpita Patra. Secret sharing. In *Secure Multi-Party Computation Against PassiveAdversaries*, Synthesis Lectures on Distributed Computing Theory, pages 17–31. Springer International Publishing, 2022.

[22] Snehal S Dahikar and Sandeep V Rode. Agricultural crop yield prediction using artificial neural network approach. *International journal of innovative research in electrical, electronics, instrumentation and control engineering*, 2(1):683–686, 2014.

[23] Saroj Dayal, Dima Alhadidi, Ali Abbasi Tadi, and Noman Mohammed. Comparative analysis of membership inference attacks in federated learning. In *Proceedings of the 27th International Database Engineered Applications Symposium*, pages 185–192, 2023.

[24] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[25] Qi Dou, Tiffany Y So, Meirui Jiang, Quande Liu, Varut Vardhanabhuti, Georgios Kaissis, Zeju Li, Weixin Si, Heather HC Lee, Kevin Yu, et al. Federated deep learning for detecting covid-19 lung abnormalities in ct: a privacy-preserving multinational validation study. *NPJ digital medicine*, 4(1):60, 2021.

[26] Ilias Driouich, Chuan Xu, Giovanni Neglia, Frédéric Giroire, and Eoin Thomas. Local model reconstruction attacks in federated learning and their uses. *ArXiv*, abs/2210.16205, 2022.

[27] Cynthia Dwork. Differential Privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 1–12, Berlin, Heidelberg, 2006. Springer.

[28] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, 2014.

[29] Fazli Khojir et al. FedShare: Secure Aggregation based on Additive Secret Sharing in Federated Learning. In *International Database Engineered Applications Symposium Conference*, pages 25–33, Heraklion, Crete Greece, May 2023. ACM.

[30] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council.

[31] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *29th USENIX security symposium (USENIX Security 20)*, pages 1605–1622, 2020.

[32] Jody L Ferris. Data privacy and protection in the agriculture industry: Is federal regulation necessary? *Minnesota journal of law, science & technology*, 18:309, 2017.

[33] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[34] Víctor Gayoso Martínez, Fernando Hernández Álvarez, Luis Hernández Encinas, and Carmen Sánchez Ávila. Analysis of ecies and other cryptosystems based on elliptic curves, 2011.

[35] Víctor Gayoso Martínez, Luis Hernández Encinas, and Carmen Sánchez Ávila. A survey of the elliptic curve integrated encryption scheme, 2010.

[36] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv*, 2018.

[37] Government of Canada. Personal information protection and electronic documents act, 2000. Accessed: 2024-06-03.

[38] Moon Halder, Ayon Datta, Md Kamrul Hossain Siam, Shakik Mahmud, Md Saem Sarkar, and Md Masud Rana. A systematic review on crop yield prediction using machine learning. In *The International Conference on Intelligent Systems & Networks*, pages 658–667. Springer, 2023.

[39] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[40] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[42] Ehsan Hesamifard, Daniel Takabi, Mehdi Ghasemi, and Rebecca Wright. Privacy-preserving Machine Learning as a Service. *Proceedings on Privacy Enhancing Technologies*, 2018:123–142, June 2018.

[43] Takeshi Horie, Masaharu Yajima, and Hiroshi Nakagawa. Yield forecasting. *Agricultural systems*, 40(1-3):211–236, 1992.

[44] Ece Isik-Polat, Gorkem Polat, and Altan Kocyigit. ARFED: Attack-Resistant Federated averaging based on outlier elimination. *Future Generation Computer Systems*, 141:626–650, April 2023. arXiv:2111.04550 [cs].

[45] Swanand Kadhe and Rajaraman et al. FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning, September 2020. arXiv:2009.11248 [cs, math, stat].

[46] Saeed Khaki, Lizhi Wang, and Sotirios V Archontoulis. A cnn-rnn framework for crop yield prediction. *Frontiers in Plant Science*, 10:492736, 2020.

[47] Daniel Kifer and Ashwin Machanavajjhala. Pufferfish: A framework for mathematical privacy definitions. *ACM Transactions on Database Systems (TODS)*, 39(1):1–36, 2014.

[48] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[49] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. Conference Name: Proceedings of the IEEE.

[50] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[51] Andy Li, Milan Markovic, Peter Edwards, and Georgios Leontidis. Model pruning enables localized and efficient federated learning for yield forecasting and data sharing. *Expert Systems with Applications*, 242:122847, 2024.

[52] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, OSDI'14, pages 583–598, USA, October 2014. USENIX Association.

[53] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning, 2020.

[54] Lan Liu, Yi Wang, Gaoyang Liu, Kai Peng, and Chen Wang. Membership inference attacks against machine learning models via prediction sensitivity. *IEEE Transactions on Dependable and Secure Computing*, 20(3):2341–2347, 2023.

[55] Julian Lo, T Yu Timothy, Da Ma, Pengxiao Zang, Julia P Owen, Qinqin Zhang, Ruikang K Wang, Mirza Faisal Beg, Aaron Y Lee, Yali Jia, et al. Federated learning for microvasculature segmentation and diabetic retinopathy classification of oct data. *Ophthalmology Science*, 1(4):100069, 2021.

[56] Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. Privacy-preserving federated learning based on multi-key homomorphic encryption. *International Journal of Intelligent Systems*, 37(9):5880–5901, 2022.

[57] Brendan McMahan and Moore et al. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, April 2017. ISSN: 2640-3498.

[58] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, April 2017. ISSN: 2640-3498.

[59] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, 2019.

[60] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, 2019.

[61] Yash More and Ramachandran et al. SCOTCH: An Efficient Secure Computation Framework for Secure Aggregation, February 2022. arXiv:2201.07730 [cs].

[62] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[63] Petteri Nevavuori, Nathaniel Narra, and Tarmo Lipping. Crop yield prediction with deep convolutional neural networks. *Computers and electronics in agriculture*, 163:104859, 2019.

[64] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, and Albert Y Zomaya. Federated learning for covid-19 detection with generative adversarial networks in edge cloud computing. *IEEE Internet of Things Journal*, 9(12):10257–10271, 2021.

[65] Pretom Roy Ovi and Aryya Gangopadhyay. A comprehensive study of gradient inversion attacks in federated learning and baseline defense strategies. In *2023 57th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, 2023.

[66] Christof Paar and Jan Pelzl. *The Advanced Encryption Standard (AES)*, pages 87–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[67] Raj Parekh, Nisarg P. Patel, Rajesh Gupta, Nilesh Kumar Jadav, Sudeep Tanwar, Abdullah Alharbi, Amr M. Tolba, Bogdan Constantin Neagu, and Maria Simona Răboacă. Gefl: Gradient encryption-aided privacy preserved federated learning for autonomous vehicles. *IEEE Access*, 11:1825–1839, 2023.

[68] James W Pease, Ernest W Wade, Jerry S Skees, and Chandra M Shrestha. Comparisons between subjective and statistical forecasts of crop yields. *Applied Economic Perspectives and Policy*, 15(2):339–350, 1993.

[69] Sebastián Ramírez. Fastapi. https://fastapi.tiangolo.com, 2018. Version 0.95.2 (or your specific version).

[70] Chao Ren, Han Yu, Rudai Yan, Qiaoqiao Li, Yan Xu, Dusit Niyato, and Zhao Yang Dong. Secfedsa: A secure differential privacy-based federated learning approach for smart cyber-physical grid stability assessment. *IEEE Internet of Things Journal*, pages 1–1, 2023.

[71] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[72] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[73] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[74] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, Los Alamitos, CA, USA, may 2017. IEEE Computer Society.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2015.

[76] Ilias Siniosoglou, Konstantinos Xouveroudis, Vasileios Argyriou, Thomas Lagkas, Dimitrios Margounakis, Alexandros-Apostolos A. Boulogeorgos, and Panagiotis Sarigiannidis. Applying federated learning on decentralized smart farming: A case study. In *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1295–1300, 2023.

[77] Lichao Sun, Jianwei Qian, and Xun Chen. LDP-FL: Practical Private Aggregation in Federated Learning with Local Differential Privacy, May 2021. arXiv:2007.15789 [cs].

[78] Michael Sykuta. Big data in agriculture: Property rights, privacy and competition in ag data services. *The International Food and Agribusiness Management Review*, 19, 06 2016.

[79] Thursday. Federated learning: Collaborative machine learning without centralized training data.

[80] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*, pages 480–501. Springer, 2020.

[81] Stacey Truex, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. LDP-Fed: Federated Learning with Local Differential Privacy, June 2020. arXiv:2006.03637 [cs, stat].

[82] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. Demystifying Membership Inference Attacks in Machine Learning as a Service. *IEEE Transactions on Services Computing*, 14(6):2073–2089, November 2021. Conference Name: IEEE Transactions on Services Computing.

[83] U.S. Congress. Health insurance portability and accountability act of 1996, 1996. Accessed: 2024-06-03.

[84] Thomas Van Klompenburg, Ayalew Kassahun, and Cagatay Catal. Crop yield prediction using machine learning: A systematic literature review. *Computers and Electronics in Agriculture*, 177:105709, 2020.

[85] Sheng Wan, Daning Hu, Jiaqi Yan, Jin Hu, Xuan Yang, et al. Leveraging federated learning for unsecured loan risk assessment on decentralized finance lending platforms. In *2023 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 663–670. IEEE, 2023.

[86] L Wang, S Xu, X Wang, and Q Zhu. Eavesdrop the composition proportion of training labels in federated learning. *arXiv preprint arXiv:1910.06044*, 2019.

[87] Xiaodong Wang, Naiyu Wang, Longfei Wu, Zhitao Guan, Xiaojiang Du, and Mohsen Guizani. Gbmia: Gradient-based membership inference attack in federated learning. *ICC 2023 - IEEE International Conference on Communications*, pages 5066–5071, 2023.

[88] Zhibo Wang, Yuting Huang, Mengkai Song, Libing Wu, Feng Xue, and Kui Ren. Poisoning-assisted property inference attack against federated learning. *IEEE Transactions on Dependable and Secure Computing*, 20(4):3328–3340, 2023.

[89] Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. A framework for evaluating gradient leakage attacks in federated learning. *arXiv preprint arXiv:2004.10397*, 2020.

[90] Leanne Wiseman, Jay Sanderson, Airong Zhang, and Emma Jakku. Farmers and their data: An examination of farmers' reluctance to share their data through the lens of the laws impacting smart farming. *NJAS - Wageningen Journal of Life Sciences*, 90-91:100301, 2019.

[91] Joshua Woodard. Big data and ag-analytics: An open source, open data platform for agricultural & environmental finance, insurance, and risk. *Agricultural Finance Review*, 76:15–26, 05 2016.

[92] Geming Xia, Jian Chen, Chaodong Yu, and Jun Ma. Poisoning attacks in federated learning: A survey. *IEEE Access*, 11:10708–10722, 2023.

[93] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, September 2017. arXiv:1708.07747 [cs, stat].

[94] Chulin Xie, Keli Huang, Pin Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[95] Yuan-Ai Xie, Jiawen Kang, Dusit Niyato, Nguyen Thi Thanh Van, Nguyen Cong Luong, Zhixin Liu, and Han Yu. Securing federated learning: A covert communication-based approach. *IEEE Network*, 37(1):118–124, 2023.

[96] Lizhi Xiong, Wenhao Zhou, Zhihua Xia, Qi Gu, and Jian Weng. Efficient privacy-preserving computation based on additive secret sharing, 2021.

[97] Chuan Xu and Giovanni Neglia. What else is leaked when eavesdropping federated learning? In *CCS workshop Privacy Preserving Machine Learning (PPML)*, 2021.

[98] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, CCS '19. ACM, November 2019.

[99] Haomiao Yang, Mengyu Ge, Kunlan Xiang, and Jingwei Li. Using highly compressed gradients in federated learning for data reconstruction attacks. *IEEE Transactions on Information Forensics and Security*, 18:818–830, 2023.

[100] Qiang Yang and Yang Liu et al. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, 10(2):1–19, 2019.

[101] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *ArXiv*, abs/1812.02903, 2018.

[102] Zhaohui Yang, Mingzhe Chen, Walid Saad, Choong Seon Hong, and Mohammad Shikh-Bahaei. Energy efficient federated learning over wireless communication networks, 2020.

[103] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[104] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 493–506. USENIX Association, July 2020.

[105] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. iDLG: Improved Deep Leakage from Gradients, January 2020. arXiv:2001.02610 [cs, stat].

[106] Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Dusit Niyato, and Kwok-Yan Lam. Local Differential Privacy-Based Federated Learn-

ing for Internet of Things. *IEEE Internet of Things Journal*, 8(11):8836–8853, June 2021. Conference Name: IEEE Internet of Things Journal.

[107] Changshi Zhou and Nirwan Ansari. Securing federated learning enabled nwdaf architecture with partial homomorphic encryption. *IEEE Networking Letters*, 5(4):299–303, 2023.

APPENDICES

Appendix A

All Additional Tables and Plots

A.1 Experimental Results

This appendix presents the detailed results of the main experiments conducted during the evaluation of the AddShare and AddShare+ frameworks, alongside other FL models such as FedAvg, SCOTCH, and FedShare. The results are organized across various datasets, including MNIST, F-MNIST, CIFAR-10, and SVHN, and focus on key performance metrics: SA, FLT, CAA, ATT, and ASST. These results offer a comprehensive overview of each model's predictive accuracy, time efficiency, and the impact of privacy-preserving techniques such as encryption. In the case of SCOTCH, the FLT is marked with an asterisk (*) where it could not be calculated due to the architecture's constraints.

Table A.1: Overall results of the 32 FL variants on CIFAR10 Dataset.

Test	Server		Client		
	SA	**FLT (s)**	**CA**	**ATT (s)**	**ASST**
Scotch	0.398	*	0.377	1.871	2.584
FedShare	0.410	430.711	0.377	1.574	1.704
FedAvg	0.419	353.599	0.377	1.521	N\A
FedAvg Encryption (RSA)	0.419	1296.098	0.377	1.635	N\A
AddShare	0.401	417.811	0.363	1.595	1.138
AddShare Encryption (RSA)	0.378	20653.193	0.360	1.790	1.504
AddShare Server - G3	0.419	380.512	0.377	1.756	0.016
AddShare Server - G5	0.419	368.573	0.377	1.625	0.014
AddShare Server - G10	0.419	349.986	0.377	1.577	0.011
AddShare Client - G3	0.411	458.152	0.377	1.574	0.056
AddShare Client - G5	0.401	450.952	0.375	1.561	0.100
AddShare Client - G10	0.378	455.707	0.374	1.588	0.203
AddShare Encryption (RSA) Server - G3	0.419	382.386	0.377	1.827	0.015
AddShare Encryption (RSA) Server - G5	0.419	381.639	0.377	1.802	0.017
AddShare Encryption (RSA) Server - G10	0.419	364.230	0.377	1.590	0.014
AddShare Encryption (RSA) Client - G3	0.411	2642.547	0.377	1.311	62.619
AddShare Encryption (RSA) Client - G5	0.401	4701.494	0.375	1.611	125.633
AddShare Encryption (RSA) Client - G10	0.378	9999.192	0.374	1.790	283.504
Addshare+	0.412	356.533	0.376	2.005	0.364
Addshare+ Encryption (ECIES)	0.412	411.643	0.377	2.070	0.374
Addshare+ Server - G3	0.419	328.481	0.377	1.390	0.012
Addshare+ Server - G5	0.419	329.389	0.377	2.128	0.031
Addshare+ Server - G10	0.419	364.764	0.377	1.329	0.014
Addshare+ Client - G3	0.412	399.901	0.374	1.414	0.030
Addshare+ Client - G5	0.409	420.695	0.374	1.367	0.037
Addshare+ Client - G10	0.407	424.333	0.374	1.411	0.070
Addshare+ Encryption (ECIES) Server - G3	0.419	399.165	0.377	2.039	0.023
Addshare+ Encryption (ECIES) Server - G5	0.419	343.359	0.377	1.352	0.013
Addshare+ Encryption (ECIES) Server - G10	0.419	402.082	0.376	1.373	0.011
Addshare+ Encryption (ECIES) Client - G3	0.412	406.371	0.374	1.338	0.234
Addshare+ Encryption (ECIES) Client - G5	0.409	418.350	0.374	1.379	0.512
Addshare+ Encryption (ECIES) Client - G10	0.407	477.544	0.374	2.072	1.390

Table A.2: Overall results of the 32 FL variants on F-MNIST Dataset.

Test	Server		Client		
	SA	**FLT (s)**	**CA**	**ATT (s)**	**ASST**
Scotch	0.706	*	0.762	1.916	1.312
FedShare	0.761	430.127	0.762	1.810	1.216
FedAvg	0.793	358.384	0.765	1.914	N\A
FedAvg Encryption (RSA)	0.793	985.338	0.765	1.891	N\A
AddShare	0.743	387.922	0.762	1.782	0.893
AddShare Encryption (RSA)	0.715	15145.941	0.765	1.800	1.566
AddShare Server - G3	0.793	356.023	0.765	1.874	0.012
AddShare Server - G5	0.793	355.315	0.765	1.881	0.012
AddShare Server - G10	0.793	344.022	0.765	1.792	0.010
AddShare Client - G3	0.791	457.627	0.765	1.910	0.053
AddShare Client - G5	0.783	461.054	0.764	1.941	0.093
AddShare Client - G10	0.771	463.702	0.765	1.894	0.175
AddShare Encryption (RSA) Server - G3	0.793	345.016	0.765	2.027	0.017
AddShare Encryption (RSA) Server - G5	0.793	365.423	0.765	2.050	0.017
AddShare Encryption (RSA) Server - G10	0.793	366.956	0.765	2.096	0.018
AddShare Encryption (RSA) Client - G3	0.791	2040.140	0.765	3.077	44.620
AddShare Encryption (RSA) Client - G5	0.783	3467.521	0.764	1.897	89.513
AddShare Encryption (RSA) Client - G10	0.771	7165.941	0.765	1.800	200.057
Addshare+	0.789	337.261	0.764	1.558	0.298
Addshare+ Encryption (ECIES)	0.789	432.197	0.764	2.772	0.375
Addshare+ Server - G3	0.793	324.616	0.765	1.586	0.010
Addshare+ Server - G5	0.793	334.524	0.765	2.630	0.125
Addshare+ Server - G10	0.793	415.437	0.765	1.571	0.260
Addshare+ Client - G3	0.790	403.772	0.763	1.552	0.035
Addshare+ Client - G5	0.789	406.613	0.763	1.602	0.050
Addshare+ Client - G10	0.788	386.524	0.762	1.561	0.060
Addshare+ Encryption (ECIES) Server - G3	0.796	322.990	0.765	2.646	0.006
Addshare+ Encryption (ECIES) Server - G5	0.793	324.194	0.765	1.562	0.011
Addshare+ Encryption (ECIES) Server - G10	0.783	408.796	0.765	1.539	0.022
Addshare+ Encryption (ECIES) Client - G3	0.790	396.265	0.763	1.570	0.175
Addshare+ Encryption (ECIES) Client - G5	0.789	402.681	0.763	1.543	0.367
Addshare+ Encryption (ECIES) Client - G10	0.788	483.902	0.763	2.588	1.109

Table A.3: Overall results of the 32 FL variants on MNIST Dataset.

Test	**Server**		**Client**		
	SA	**FLT (s)**	**CA**	**ATT (s)**	**ASST**
Scotch	0.944	*	0.942	1.876	1.487
FedShare	0.950	404.625	0.942	1.867	1.139
FedAvg	0.958	354.818	0.942	1.927	N\A
FedAvg Encrypyted (RSA)	0.958	983.385	0.942	1.902	N\A
AddShare	0.952	400.673	0.937	1.935	0.925
AddShare Encryption (RSA)	0.951	17959.740	0.930	1.900	1.644
AddShare Server - G3	0.958	358.928	0.942	1.901	0.013
AddShare Server - G5	0.958	352.054	0.942	1.838	0.011
AddShare Server - G10	0.958	345.283	0.942	1.786	0.009
AddShare Client - G3	0.957	453.219	0.941	1.883	0.049
AddShare Client - G5	0.955	459.047	0.942	1.897	0.092
AddShare Client - G10	0.955	463.981	0.942	1.919	0.185
AddShare Encryption (RSA) Server - G3	0.958	363.821	0.942	2.017	0.016
AddShare Encryption (RSA) Server - G5	0.958	366.923	0.942	2.024	0.019
AddShare Encryption (RSA) Server - G10	0.958	351.004	0.942	1.874	0.012
AddShare Encryption (RSA) Client - G3	0.957	2042.896	0.941	3.005	44.783
AddShare Encryption (RSA) Client - G5	0.955	3469.496	0.942	1.904	89.590
AddShare Encryption (RSA) Client - G10	0.955	7154.555	0.942	1.767	199.939
Addshare+	0.957	347.539	0.942	1.595	0.407
Addshare+ Encryption (ECIES)	0.957	438.283	0.942	1.786	0.320
Addshare+ Server - G3	0.958	324.684	0.942	1.632	0.009
Addshare+ Server - G5	0.958	345.455	0.942	2.644	0.013
Addshare+ Server - G10	0.958	408.412	0.942	1.563	0.023
Addshare+ Client - G3	0.957	397.189	0.941	1.651	0.075
Addshare+ Client - G5	0.958	398.093	0.941	1.593	0.053
Addshare+ Client - G10	0.957	414.190	0.941	1.575	0.052
Addshare+ Encryption (ECIES) Server - G3	0.958	320.229	0.942	2.666	0.030
Addshare+ Encryption (ECIES) Server - G5	0.958	324.376	0.942	1.555	0.009
Addshare+ Encryption (ECIES) Server - G10	0.958	324.869	0.942	1.560	0.012
Addshare+ Encryption (ECIES) Client - G3	0.957	392.788	0.941	1.564	0.176
Addshare+ Encryption (ECIES) Client - G5	0.958	395.842	0.941	1.521	0.368
Addshare+ Encryption (ECIES) Client - G10	0.957	493.696	0.941	2.661	1.117

Table A.4: Overall results of the 32 FL variants on SVHN Dataset.

Test	Server		Client		
	SA	**FLT (s)**	**CA**	**ATT (s)**	**ASST**
Scotch	0.688	8.877	0.667	1.750	0.648
FedShare	0.689	593.149	0.653	2.132	0.539
FedAvg	0.703	530.480	0.667	3.373	*
FedAvg Encrypyted (RSA)	0.702	1587.473	0.667	3.318	*
AddShare	0.522	536.093	0.660	2.073	1.146
AddShare Encryption (RSA)	0.517	21040.402	0.667	3.565	1.509
AddShare Server - G3	0.702	509.075	0.667	2.221	0.016
AddShare Server - G5	0.702	491.679	0.667	2.090	0.014
AddShare Server - G10	0.702	441.086	0.667	1.969	0.011
AddShare Client - G3	0.702	566.340	0.669	2.094	0.060
AddShare Client - G5	0.697	616.094	0.669	2.195	0.115
AddShare Client - G10	0.685	616.199	0.667	2.205	0.246
AddShare Encryption (RSA) Server - G3	0.702	538.905	0.667	2.470	0.017
AddShare Encryption (RSA) Server - G5	0.702	550.233	0.667	2.495	0.017
AddShare Encryption (RSA) Server - G10	0.702	505.514	0.667	2.225	0.017
AddShare Encryption (RSA) Client - G3	0.702	2857.585	0.669	3.238	63.309
AddShare Encryption (RSA) Client - G5	0.697	4852.149	0.669	2.144	126.265
AddShare Encryption (RSA) Client - G10	0.685	10040.609	0.667	2.065	281.577
Addshare+	0.698	585.478	0.666	2.093	0.389
Addshare+ Encryption (ECIES)	0.698	593.149	0.665	2.132	0.392
Addshare+ Server - G3	0.702	452.760	0.666	1.870	0.011
Addshare+ Server - G5	0.702	508.526	0.667	2.844	0.021
Addshare+ Server - G10	0.702	578.325	0.667	1.717	0.023
Addshare+ Client - G3	0.704	499.881	0.667	1.758	0.042
Addshare+ Client - G5	0.705	597.409	0.668	1.796	0.056
Addshare+ Client - G10	0.704	598.817	0.667	1.733	0.074
Addshare+ Encryption (ECIES) Server - G3	0.702	591.512	0.666	2.902	0.011
Addshare+ Encryption (ECIES) Server - G5	0.702	470.661	0.667	1.759	0.012
Addshare+ Encryption (ECIES) Server - G10	0.702	521.877	0.667	1.750	0.025
Addshare+ Encryption (ECIES) Client - G3	0.704	512.364	0.667	1.693	0.230
Addshare+ Encryption (ECIES) Client - G5	0.705	512.339	0.668	1.719	0.482
Addshare+ Encryption (ECIES) Client - G10	0.704	625.027	0.667	2.875	1.384