EFFICIENT PRIVACY-PRESERVING DEEP NEURAL NETWORK TRAINING PROTOCOL IN FEDERATED LEARNING

by

Gaurav Vinay Uttarkar

Bachelor of Engineering, Vidyavardhaka College of Engineering, 2020

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor:	Kalikinkar Mandal, Ph.D., Computer Science
Examining Board:	Michael Fleming, Ph.D., Computer Science, Chair
	Roozbeh Razavi-Far, Ph.D., Computer Science
	Clodualdo Aranas, Ph.D., Mechanical Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

June, 2023

Abstract

Machine learning is being used in large sectors such as healthcare and financial services. This raises privacy concerns regarding user data and model privacy. As a result, federated learning (FL) was introduced. It empowers users to combine their models through a centralized server. In FL, since a user computes their model locally, the user input is not directly threatened, whereas the privacy risk of model misuse is high.

In this thesis, we propose a robust, efficient privacy-preserving DNN training protocol, built with PrivFL as its foundation. Our private DNN training protocol consists of a secure and efficient local gradient computation protocol and a secure aggregation protocol. We develop an optimized two-party local gradient computation protocol using fully homomorphic encryption and garbled circuit. The essence of our secure multiparty aggregation is computing the global gradient of DNNs. We analyze the security against semi-honest adversaries and implement it on real-world datasets.

Dedication

To,

My parents - Mary and Vinay Uttarkar,

My brother - Dhanush Uttarkar,

My late grandmother - Lily Karunakaran

Acknowledgements

Completing this master's thesis has been a herculean yet rewarding journey, and I would like to express my heartfelt gratitude to all those who have supported me throughout this process.

First and foremost, I would like to express my sincere gratitude to my thesis supervisor and mentor, Dr. Kalikinkar Mandal. His guidance, expertise, and encouragement have been invaluable in shaping this work. He always challenged me to think critically, offered insightful feedback, and provided the support I needed to push through the difficult times in all walks of my life these two years.

I would also like to extend my gratitude to the faculty members at the University of New Brunswick (UNB) and Canadian Institute of Cybersecurity (CIC) for their support and contributions to my research.

I am indebted to have parents and a brother who provided unwavering support, love, technical assistance, knowledge and encouragement throughout this journey. Their belief in me and my abilities kept me motivated and focused, even when the work felt overwhelming.

I appreciate the staff at UNB and CIC, in particular, the librarians and research support team, who helped me navigate the complexities of academic research and assisted me. Finally, I would like to acknowledge my fellow students, friends and colleagues, who provided valuable feedback and support. Their perspectives, insights, and encouragement were helpful in shaping this work.

Table of Contents

List of Tables

List of Figures

List of Symbols, Nomenclature or Abbreviations

ML	Machine Learning
MLaaS	Machine Learning as a Service
NN	Neural Network
DNN	Deep Neural Network
BNN	Binarized Neural Network
CNN	Convolutional Neural Network
SVM	Support Vector Machines
PCA	Principal Component Analysis
EDA	Exploratory Data Analysis
GD	Gradient Descent
SGD	Stochastic Gradient Descent
MBGD	Mini-batch Gradient Descent
BGD	Batch Gradient Descent
FL	Federated Learning
PPML	Privacy-preserving Machine Learning
PPFL	Privacy-preserving Federated Learning
MPC	Secure Multiparty Computation
2PC	Secure two-party Computation

3PC	Secure three-party Computation
AS	Additive secret sharing
GC	Garbled Circuit
OT	Oblivious Transfer
HE	Homomorphic Encryption
PHE	Partially Homomorphic Encryption
SHE	Somewhat Homomorphic Encryption
AHE	Additive Homomorphic Encryption
FHE	Fully Homomorphic Encryption
CRT	Chinese Remainder Theorem
SIMD	Single Instruction Multiple Data
SISO	Single Input Single Output
MIMO	Multiple Input Multiple Output
ReLU	Rectified Linear Unit
CE	Cross Entropy Loss
MSE	Mean Squared Error
GPU	Graphics Processing Unit
B2A	Bit to Arithmetic
MSB	Most Significant Bit
LSB	Least Significant Bit
LWE	Learning With Errors
RLWE	Ring-Learning With Errors
SEAL	Simple Encrypted Arithmetic Library
EMP	Efficient Multiparty Computation

Chapter 1

Introduction

Machine learning is a cutting-edge technology being used by many large companies to perform tasks such as predictive analysis, voice assistants [14], text recognition, and more. Among many techniques of ML, deep learning is a branch that involves the use of deep neural networks to perform complex tasks as seen in [60]. Large corporations such as Google, Microsoft, Tesla, and Amazon use deep learning in their systems or products, for example, Google's alpha GO [47], Tesla's electric cars [63] and Amazon's voice assistant, Alexa [14]. Some other applications include medical image classification, natural language processing, fake news detection, music composition, and automatic text generation, mentioned in [8], [31]. Neural networks require abundant data to provide reliable results depending on the task. Training a neural network demands accurate data and hence, choosing the type of data involved in the model's training is vital. When a model is trained, evaluated, and tuned, it can be used for inference/prediction services. Due to the availability of enormous data produced every day and machine learning models expanding in size and functionality, it is difficult to maintain the data on individual systems and work with multiple entities. Evolving cloud technology has now made it possible for multiple users to work together. Therefore, machine learning as a service (MLaaS) is be-

ing extensively used in the modern-day world. MLaaS is the collaboration between machine learning and cloud services [72] which makes the data easily accessible for model training and in turn, produces high-quality models for inference. Although healthcare and banking sectors provide MLaaS [83] at a large scale, a major drawback while using these services is maintaining the privacy of the data involved. One of the techniques used to protect the data is federated learning (FL). Federated learning is a cloud-based machine learning technique where multiple users can work together and train a model with their local data without having the need to share their local data on the cloud or with the service provider. Federated learning has an advantage over traditional machine learning as it prevents the user data from direct threat. Some of the other advantages include low latency and low power consumption during computation. The Gboard (keyboard provided by Google) is an example of federated learning on mobile phones [16]. When sensitive and important user data is being used, it needs to be kept private. Similarly, the models used for training and predictions contain important parameters for which confidentiality and privacy also need to be protected. This is where the increasingly sought-after area of study, privacy-preserving machine learning, becomes a valuable component in every researcher's toolkit.

Privacy-preserving machine learning (PPML) is a method that implements cryptographic techniques on machine learning algorithms and operations to protect and preserve the privacy of both the user data and the model involved in either training or inference. Using this method, the confidentiality of both the service provider's and the client's data is protected, thereby providing a safe, secure, and smooth interaction between the parties. Even though this method provides privacy and protects data, it adds overhead in the computation and communication due to which the technique is inefficient. Therefore, there are a lot of publications and research that emphasize on improving the efficiency of the techniques either during private

training, private inference, or both. These efficient techniques can also be applied in the domain of federated learning. Multiple, different cryptographic primitives can be used to implement these techniques that protect the privacy of the user input and models. Some examples of cryptographic primitives are multi-party computation (MPC) techniques, homomorphic encryption, differential privacy and garbled circuit.

1.1 Contributions of This Thesis

This thesis aims at addressing the challenge of ML data privacy leakage during training among one or more clients and a centralized server. Our key contribution is the design of a strongly secure privacy-preserving federated learning training protocol. In order to achieve that, we need a secure gradient computation protocol that is designed to protect the privacy of private input data held by the client, as well as the private model held by the server.

Design of Privacy-preserving Federated Training Protocol. We design our gradient computation protocol that uses homomorphic encryption (HE) and garbled circuit (GC) to protect the privacy of both the input data and the model. In this protocol, the linear layer is optimized using an efficient, secure matrix multiplication technique which reduces the total number of ciphertexts produced and the overhead. For the non-linear layer, we use an optimized garbled circuit to reduce the number of computations of the activation functions. We use our gradient computation protocol and a secure aggregation protocol to securely compute the global model for FL.

Privacy. The secure gradient computation is executed between each client and the server which is secure as it is built on secure cryptographic primitives such as homomorphic encryption, garbled circuits, and secret sharing which are proven

to be secure. Since all the computations are done in shares between the parties, no private unintended data is leaked. In the privacy-preserving federated learning training protocol, we use a secure aggregation protocol to update the global model. Hence we provide a stronger privacy for the computation of the global gradient in the federated training setting.

Experimental Evaluation. We implement and evaluate our secure federated training protocol on two real-world datasets namely MNIST [40] and ESR [3] for two different architectures, namely LeNet [58] and Poseidon [82]. Our experimental results demonstrate that our protocols have faster execution times with a similar or better accuracy. When compared with Poseidon, our protocol is about 3.27x faster for the MNIST dataset and about 8.15x for the ESR dataset. The details are provided in Chapter 7.

1.2 Organization of This Thesis

The thesis is organized as follows:

- *Chapter 2* consists of background on machine learning. This includes 2.1 machine learning, 2.2 federated learning, 2.3 deep neural networks, 2.4 adversarial machine learning followed by a detailed study of existing literature similar to our work, using various cryptographic primitives and how they perform in case of (2.5.1) only private inference, (2.5.2) private inference are training, (2.5.3) private federated training and inference.

- *Chapter 3* provides details on what background information and preliminaries and required before learning how to implement the protocol.3.1.1 building blocks to train NN, 3.2.1 additive secret sharing, 3.2.6 secure aggregation protocol, 3.2.3 garbled circuit, 3.2.2 oblivious transfer and 3.2.4 homomorphic

encryption.

- *Chapter 4* describes the 4.2 system setup and 4.4 threat model we consider for our protocol setting.

- *Chapter 5* introduces the secure gradient computation protocol built on efficient matrix-vector multiplication technique and optimized garbled circuit construction along with the security and computation complexity analysis.

- *Chapter 6* presents the federated training setting in which we implement our protocol and how we improve the efficiency of the training process along with the security and computation complexity analysis.

- *Chapter 7* provides the implementation details and experimental results obtained.

- *Chapter 8* summarizes our work by providing a conclusion and direction for future work.

Chapter 2

Literature Review: PPML

This chapter investigates all the previous and current work in the domain of privacy-preserving machine learning and federated learning. Some work is specific to private inference while others are specific to private training and private federated learning. The scope of this literature review includes the techniques used to optimize and efficiently compute neural networks while preserving the privacy of the data involved. It also covers the basics that help one to understand the protocol that is central to this thesis.

2.1 Machine Learning

Machine learning is a field of study under the umbrella of artificial intelligence where computers are trained to solve and perform complex tasks. Machine learning uses algorithms that take input data and provide an output, relevant to the problem at hand i.e. classification or regression or prediction tasks. Some examples of industries using these algorithms are the healthcare sector for image recognition, the financial sector for loan disbursement, financing credit cards, and loan default prevention. Each of these tasks requires different algorithms. Some of the most commonly used machine learning algorithms are classified under supervised learning, unsupervised

learning or reinforcement learning.

Supervised Learning. Supervised learning algorithms mainly perform classification or regression tasks [85], [81]. Some examples of supervised learning algorithms are:

- **Linear Regression.** Linear regression is an algorithm that is used in classification or prediction tasks when there is one independent variable and one dependent variable. The algorithm plots the value to seek the best fit in the linear equation.

- **Logistic Regression.** Logistic regression is applied when the variables are continuous and the outputs are binary. The goal of logistic regression is to estimate the probability of a binary outcome, which can take values of 0 or 1 such as "true" or "false","yes" or"no" etc. The logistic regression model uses a logistic function to map the input variables to the binary outcome. Logistic regression is commonly used in applications such as medical diagnosis, credit risk analysis, and marketing analytics.

- **Decision Trees.** Decision trees are a popular machine learning algorithm used for both classification and regression analysis. The algorithm creates a tree-like model of decisions and their possible consequences based on input data. At each node of the tree, the algorithm evaluates a specific feature of the input data and makes a decision based on that feature. They are widely used in various applications, such as finance, healthcare, and marketing, due to their ability to handle both categorical and continuous variables and their ability to identify the most important features in a dataset.

- **Support Vector Machine.** Support vector machine (SVM) is a machine learning algorithm that is commonly used for classification and regression anal-

ysis. The primary goal of SVM is to find the best hyperplane that separates two or more classes of data points in a high-dimensional space. SVM uses a kernel function to transform the input data into a higher-dimensional space where it can be more easily separated. SVMs are used when the data points are non-linear. SVM has proven to be a powerful and flexible machine learning algorithm and is widely used in many applications, including image classification, text classification, and bio-informatics.

- **Neural Networks.** Neural networks are classified under deep learning algorithm that is modeled after the human brain. It has an input layer, one or more hidden layers, and an output layer. It is mostly used for prediction and classification tasks but can also be applied for regression tasks. Activation functions and loss functions are integral parts of neural networks.

Unsupervised Learning. Unsupervised learning solves the task of clustering or association [42]. Some examples of unsupervised learning algorithms are:

- **Autoencoders.** Autoencoders are similar to neural networks such that they take in an input and run it through layers to learn about the input. It consists of an input layer, encoder layer, decoder layer, and output layer. The unique aspect of autoencoders is that the output is just a replica of the input and it's modeled to learn that value both ways.

- **K-means Clustering.** The algorithm for K-Means clustering takes in unlabelled input data and places the data points in pre-determined groups which are"K" in number. The number of groups "K" is determined by the user and based on certain criteria. It means if $K = 2$, there are 2 groups into which the data is placed and as the algorithm runs, the data points are placed accordingly until the last data point.

- **Principal Component Analysis.** The principal component analysis (PCA) algorithm is used to reduce the dimensionality of the dataset by eliminating variables with higher co-dependence or interrelation. The process of performing PCA involves computing the eigenvalue decomposition of the covariance matrix of the data along with eigenvectors to project a lower dimension space of the feature space. PCA is used in exploratory data analysis (EDA) and predictive modeling.

- **Apriori Algorithms.** The apriori algorithm is used to create association rules among an itemset or group of data points given their frequency of occurrence. It is a categorization algorithm that works on databases where the database usually holds transactions. A breadth-first search is used to check how two items are correlated. A common application of the algorithm is market basket analysis.

Among these algorithms, most works in the literature survey that are presented involve the use of neural networks. Neural networks with complex structures are called deep artificial neural networks and this falls under the category of deep learning which is a sub-category of machine learning.

2.2 Federated Learning

Since federated learning is the basis for our protocol, we explain how the technique works and how it can be applied along with the advantages and disadvantages. Federated learning is an algorithm proposed by Google. Because of the strict laws and regulations around the world about the protection of data, FL was introduced. The algorithm involves training models using distributed datasets amongst multiple clients. Distributed datasets from multiple sources are used to enhance the performance of the model because diverse data helps the ML model perform better in

various conditions. There are industries that critically need FL because it provides greater security and access privileges to data while generating robust models without sharing local data directly with the service provider. Each client taking part in the FL process receives the model from the server and trains it on their dataset, locally. After training the model locally, users update the model and transfer only the gradients to the server. The server then aggregates these gradients to obtain the global model and this process is repeated iteratively until satisfactory accuracy is achieved.

Advantages. FL has several advantages over traditional centralized machine learning approaches. One of the major advantages is preserving the privacy of client data. This is achieved by allowing the data to remain on the client devices and only model updates are shared, FL protects the privacy of sensitive data that may be stored on these devices. This is particularly important in scenarios where data privacy is a critical concern, such as healthcare or finance sectors. Another advantage of FL is the ability to leverage large amounts of distributed data. In many cases, the data used for training machine learning models are stored in different locations and are difficult to centralize which is possible in FL. Finally, since only small updates are sent to the server, the computation is faster for the server and updates are more frequent.

Disadvantages. Some disadvantages of FL include device specification due to diversity, communication overhead, and depending on the devices, network connectivity. The major disadvantage FL does not address is the protection of the model which is sent to the clients. As the model is being transferred back and forth between clients and servers, it is vulnerable to model extraction, poisoning, or evasion attacks.

2.3 Deep Neural Networks

Deep neural networks are widely used in the PPML. In this section, we elaborate on how the algorithm works and the importance of each phase.

2.3.1 Introducing Neural Networks

Neural networks is a deep learning algorithm modeled after the human brain. The reason they are called neural networks is because they tend to resemble the connections in the neurons and synapses found in the brain. The neural network contains layers of interconnected nodes where each node is known as a perceptron.

A neural network is used to perform inference services before which it needs to be trained. For the training phase, large amounts of data are fed into the neural network and this data is passed through layers in the neural network whereby the neural network learns the behavior of the data or learns the features of the input data using the gradient descent algorithm, the neural network uses information gathered from the training phase and updates the model parameters to improve the accuracy of learning features. The inference phase is where the neural network predicts or infers any particular result from the input data used. In this thesis, the deep learning model is used for image recognition and hence, the input data that will be used is an image.

2.3.2 Implementing Neural Networks

Neural networks are used for tasks such as image classification or recognition which is one of its most widely used applications. During the process of image recognition or classification, the input image is broken down into pixels where each group of pixels represents a feature of the input image. The neural network is trained to learn these features and produces a model which recognizes these features. This model

is then used for image classification. Every neural network is made up of neurons which helps it to learn about the features of an input image. Neurons are the basic building block of a neural network and each of these neurons is assigned a particular weight or value which defines its importance in the structure. All the neurons are then passed through various neural network operations which decide the value of the weight and biases. The neural network then adjusts the weights and biases by comparing them with the pixel values of the image until it produces a model that can recognize similar images with high accuracy.

In the case of any machine learning or deep learning models, there are seven stages of performing the entire process as seen in [13] and displayed in Figure 2.1.

1. **Collection of Data.** This is the first step in applying machine learning in any given situation because it is impossible to apply any algorithms without data. The data has to be collected from trusted and verified sources to obtain the best possible model. Although there are multiple sources with information about the same dataset there is a possibility that it could be tampered with or poisoned. Therefore, one needs to refer to the official sources such as the datasets available in [89] and [53] to get authentic data.

2. **Preparation of Data.** Since the data collected is raw without any pre-processing, the dataset could contain many missing values or unknown values, or even unnecessary rows/columns. In some cases, the dataset contains features that do not contribute to the final model because they do not play a major role in the particular regression or classification task. For example, when the algorithm is predicting the number of customers visiting a restaurant on a particular day, gender might not be relevant but it could be collected and added to the dataset. Hence pre-processing or preparation of the data is an important step that includes cleaning the data, visualizing the data, finding the correlation between features, and many more. 1 and 2 together are labeled

as EDA [17].

3. **Choosing an Appropriate Algorithm.** Machine learning algorithms are used to perform the inference or prediction services and hence finding the right algorithm is an important task. It is quintessential to apply the right machine learning algorithm to the right task since there are many models available for various tasks like speech recognition, image recognition, or weather forecasting. Neural networks would work better for image recognition tasks but are complex whereas simple tasks such as weather forecasting can use linear regression models.

4. **Training the Model.** Training the model results in determining the model weights. It is obtained by running the processed data through the chosen ML algorithm. This technique produces a model that is used for prediction and inference. Training of the model is a computationally intensive as well as memory-intensive execution process when compared to the other processes. Although the model is trained, it cannot be immediately deployed to perform tasks like prediction or inference. The trained model requires verification and validation until it is ready for use.

5. **Evaluating the Model.** After the model is trained, the model needs to be evaluated on data other than what it was trained on. Usually, datasets are split into testing data and training data. The training data is used to train the model and testing data is used to evaluate/test the capability of the trained model. The model usually performs well in most cases. In some cases, the model is either overfit or underfit. Underfit models make accurate incorrect predictions initially whereas overfit models make inaccurate predictions as explained in [71]. We learn this through model evaluation.

6. **Model Parameter Tuning.** This is the final step to produce a model where

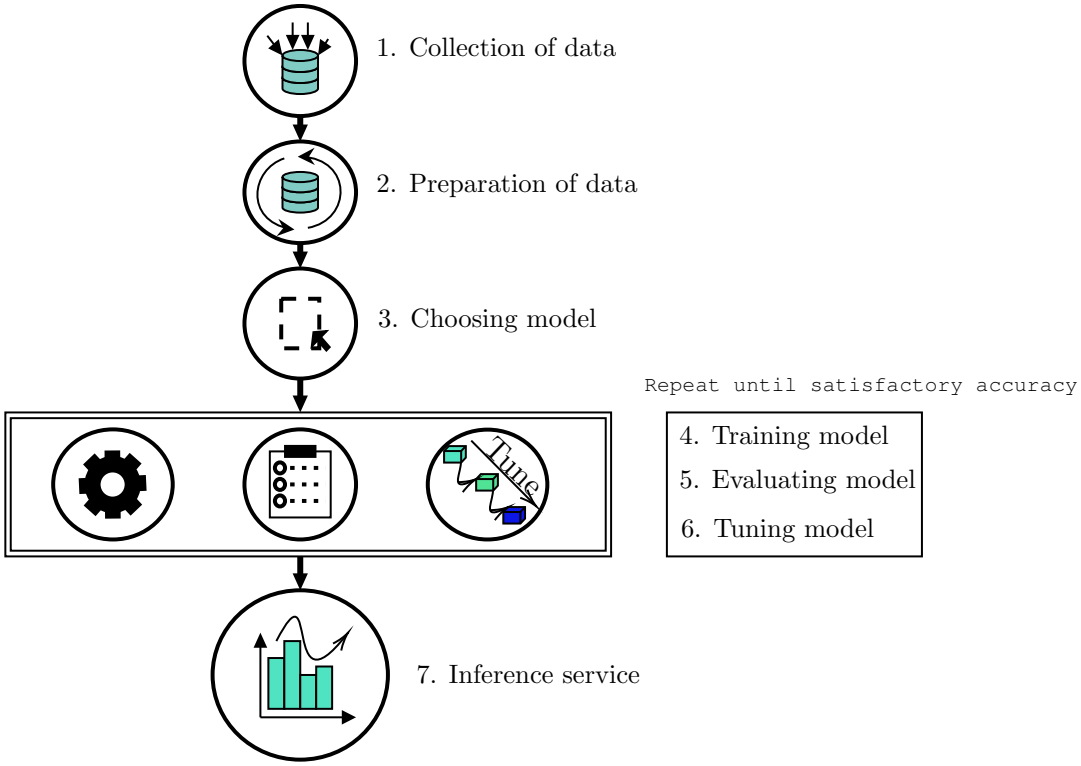

Figure 2.1: Machine Learning Lifecycle

we tune the parameters that need to be changed in accordance with the evaluation of the model. Parameters such as the number of iterations, dimension of the models, learning rate, etc are tuned and changed to meet the required functionality [69]. Once this is complete, the model is ready for deployment.

7. **Inference/Prediction on the Data.** The final model is used in prediction, classification and many more applications.

2.4 Adversarial Machine Learning

This section details the type of attacks that can be launched on the protocols and the capability of the attacks.

Adversarial Machine Learning. Adversarial machine learning is used to launch attacks on machine learning algorithms by identifying vulnerabilities. These attacks

can exploit the data used to train the model or exploit the model by learning about its weights. This is a clear violation of privacy which is considered to be a serious issue given the rapid growth of MLaaS in the modern world as stated in [9] and [21]. Types of adversarial ML attacks are explained below:

1. **Poisoning Attacks.** Usually goes by the name of data poisoning which implies that data is poisoned by changing or adding incorrect samples to it which causes it to under perform after training. This attack can also be launched on data in the re-training process as explained in [21].

2. **Evasion Attacks.** Evasion attack is another common type of attack launched during deployment in a malware or intrusion attack situation. The attackers obfuscate the means of the attacks like spoofing as stated in [21].

3. **Model Extraction Attacks.** Model extraction attacks focus on attacking the model held by a central server. Models are used to perform inference, predictions or classifications. The attack targets the model by extracting values and obtaining information from the models through querying and then, using this information to their malicious benefit while compromising the privacy of the authorized model owner [9].

4. **Model Inversion Attacks.** An adversary can reconstruct training data from model parameters which is a major privacy risk as seen in [20] and [21].

5. **Membership Inference Attacks:** An adversary can infer whether an individual participated in the process of either training or inference, thereby gaining access to sensitive data that can be misused by the adversary as mentioned in [20] and [9].

2.5 Privacy-Preserving Machine Learning

2.5.1 PPML: Private Inference

2.5.1.1 ABY: Mixed Two-Party Computation Framework

In this work [39], the authors use mixed protocols for PPML. It considers a semi-honest secure two-party computation setting. The authors had an idea of improving on the existing mixed-protocol technique which used homomorphic encryption for operations such as addition and multiplication and garbled circuits or other MPC protocols for the activation functions. ABY uses a combination of MPC techniques for both layers and shows that for applications such as private set intersection, biometric matching, and modular exponentiation, it proves to be more efficient than the previous mixed protocol technique. This is due to the conversion cost between the HE scheme and MPC protocol when the parameters for HE are scaled. ABY (arithmetic, boolean, and Yao sharing) has a flexible design process and novel highly efficient framework for PPML. The problem the authors are trying to fix with their approach is the efficiency of using PPML techniques using either a single protocol or using a combination of HE and MPC. By introducing this, they provide a flexible option of using a combination of either arithmetic, boolean or Yao's secret sharing techniques which gives the user more options. As for the technique, it considers arithmetic sharing (uses additive sharing in the ring Z_{2^l}), boolean sharing (uses an XOR-based secret sharing technique to share variable), and Yao's secret sharing (converts a function into a boolean circuit). Each of these types has operations such as shared value, sharing, reconstruction, XOR, AND, and a few others are evaluated and explained. Conversions from arithmetic to boolean to Yao's and vice-versa are compared efficiency-wise and finally, experiments are performed. These techniques have not been tested for neural networks. Although these techniques might be applicable in that domain, they might not be as efficient as using HE and MPC as mixed

protocols.

2.5.1.2 Secure Matrix Computation Framework

This work [51], mainly focuses on applying HE in the domain of PPML. Although many works present a combination of HE and MPC techniques such as garbled circuit protocol or the GMW protocol, HE when used in a certain way for certain applications, can prove to have a certain level of efficiency over other mixed protocols. The authors also consider a setting where both the input provided by the client or user, as well as the model held by the server or service provider, are encrypted. In most cases, the input is usually encrypted since it is being sent over a channel to the server while the server holds the neural network model in plaintext which it uses to perform inference. Various techniques are introduced in this work for HE. The first method is using an encoding technique to convert a matrix into a plaintext vector to operate in SIMD. By using this encoding technique, efficient matrix multiplication algorithms can be modeled. They introduce the algorithm which uses a row ordering encoding map to transform a vector of a particular dimension into a matrix. This encoding forms an isomorphism between additive groups which shows that matrix addition can be computed in a secure manner using homomorphic addition in a SIMD manner. Next, some tweaks are made in permutations on the conventional packed ciphertexts to perform matrix multiplication. The algorithms consist of steps that involve the linear transformation of the first matrix followed by the evaluation of the second matrix with the first after which column and row shifting operations are securely computed and finally, the Hadamard product with ciphertext aggregation gives the result of the entire neural network operation. Some other improvements in the same matrix multiplication mentioned are rectangular matrix multiplication and matrix transposition on packed ciphertexts. They call the model E2DM and results show that these techniques boost the speed of the operations performed.

Amortized communication and run-time costs are lower when compared to previous works. However, when we compare numbers in total communication or run-time costs there are some lacking areas.

2.5.1.3 ABY3: A Mixed Protocol Framework for Machine Learning

The main aim of this work [66], is to show that it is possible to use the basic building blocks that were built in one of the previous works to train the machine and deep learning algorithms. The building blocks were arithmetic, binary, and Yao's secret sharing protocols which can be inter-switched for the purpose. For the threat model, this work considers three parties with malicious intent (malicious adversaries who deviate from the protocol arbitrarily). For cryptographic primitives, the work includes additive, binary and Yao's secret sharing for 2PC and 3PC settings. The work contributes a new approximate fixed-point multiplication set of protocols for the 3PC setting in particular. This protocol is more efficient than the others introduced. Next, they introduce a framework to efficiently switch between the 3 sharing protocols and finally, they introduce a new technique known as delayed re-share which reduces communication complexity by several orders when considering vectorized operations. For the approximate fixed-point multiplication, a 3PC semi-honest setting is considered. Then there is a linear sharing of values followed by offline generation of a truncation pair of values and this is done efficiently. The authors identified that in the ABY switching framework linear regression training required additive secret sharing while, in the case of logistic regression, Yao's sharing protocol was required and to achieve this, efficient conversion techniques were implemented. When it comes to the piece-wise polynomial functions, each of these functions computes varying polynomials at each input interval. A new optimized protocol has been built for both semi-honest and malicious settings. During share conversion, bit manipulation techniques such as bit decomposition, extract, composition, and

injection are used. Conversions can be done from any type to another three. Results when compared to previous work show better performance although only the MNIST dataset was used. Hence, experiments need to be performed on larger datasets and networks to check the variance in outcome.

2.5.1.4 Secure Deep Learning Framework

This work [78], solves the problem of having high communication and computation costs when performing oblivious neural network inference due to the application of cryptographic primitives. The authors propose a scalable and provable secure framework with an approach of using only Yao's garbled circuits to perform the operations and optimize them for the best results. The threat model is considered "honest but curious" or "semi-honest" setting (the adversary follows the rules of the protocol but tries to obtain information passively). The cryptographic protocols considered are oblivious transfer (OT) and garbled circuit (GC). The author's contribution is the first provably-secure framework while also enabling accurate and scalable privacy-preserving deep learning execution, next is custom libraries for "GC-optimized netlists" which are required and these are built on automated design, efficient logic synthesis, and optimization methodologies. Taking a closer look at the techniques first the data and neural network pre-processing is performed. Complex modern data matrices are made into multiple lower-rank subspaces which are highly dimensional and dense. The deep neural network is re-trained using a particular objective function and a streaming-based data sketching methodology is used to solve the optimization objective in this function. Pruning is performed on the neural network to remove unnecessary garbling operations and generate a netlist for optimized GC protocol. An industrial synthesis tool is adopted to optimize the produced netlist for XOR operations. For the GC-optimized circuit components library, using the synthesis tool, all necessary computations are prepared and XOR and non-XOR

circuits are realized. Evaluation of coordinate rotation digital computer circuit is also done (CORDIC). Results show that in some cases the work performs better than its predecessor and in some cases, it does not.

2.5.1.5 GAZELLE: Neural Network Inference Framework

The authors in this work [52], introduced a novel idea to improve the efficiency of performing the operations in the computation of a neural network. The threat model considered by this work is a semi-honest two-party in which the adversary passively tries to obtain information that is not meant to be revealed. The client has an encrypted input and the server holds the plaintext model or weights which are an integral part of neural network computation. To compute these operations and the result, GAZELLE, uses a mixed protocol that includes HE for the linear layer computation and the use of secret sharing along with garbled circuits for the non-linear layer. The linear layer consists of matrix-vector multiplications which were done using a naive approach initially. This work introduced a novel approach. With a combination of ciphertext packing (which some of the HE schemes provide) and multiplication, the authors were able to manipulate the structure of the matrix during the multiplication of packed ciphertexts. This idea was introduced both in the fully connected layer as well as the convolutional layer. Hence this technique can be implemented in any convolution neural network architecture to obtain a faster inference result. The homomorphic encryption scheme used in this work was the paillier scheme because of the homomorphic properties it provides as a packed additively homomorphic encryption scheme. The idea for fully connected layers first experimented with the Naive approach, input and output packing, and diagonal packing after which they proposed a hybrid approach. As for the convolution layer, they improved on the padded SISO by making it the packed SISO. Other techniques included handling strided convolutions, low-noise batched convolutions, arranging

single channels per ciphertext, and channel packing. For the non-linear layer, garbled circuit protocol was used because of the fixed round property. With this, the results were quite superior in comparison to previous works for communication costs and boost in speed. The technique as mentioned has not been tested for very large-sized neural network models which was a task for the future.

2.5.1.6 XONN: Oblivious Deep Neural Network Inference

The authors in this work [77], consider deep neural network inference but due to the heavy computation and communication costs with generic methods used by prior works (HE is used for matrix multiplications and other linear operations), the authors propose an algorithm that replaces neural network operations by XNOR gates which are basically free in garbled circuit protocol (XONN). The work contributes the above-mentioned novel approach in the form of a protocol (XONN), a high-level API to call the XONN functionality in python (Keras), and a compiler that translates model description from high-level python to XONN. The cryptographic primitives considered are secret sharing, oblivious transfer and garbled circuits. The threat model considered is the semi-honest or honest-but-curious (HbC) setting and XONN can be modified to work for malicious security using cut-and-choose techniques. To implement the protocol as explained, the authors had to convert the trained neural network's weights to binary because multiplication in garbled circuits has a quadratic computation and communication cost directly proportional to the input bit length. To get a binarized neural network (BNN), linear scaling and network trimming are performed followed by feature ranking and then iterative pruning. This BNN then performs all the operations of a neural network in the binarized form using garbled circuits. For the first binary linear layer, vector dot product (VDP) operation can be categorized into two classes i.e. integer-VDP and binary-VDP. Next, the activation function, batch normalization, and max-pooling are all performed in the binarized

form. Another protocol that improves the computation is the oblivious conditional addition protocol (OCA). This protocol is applied to the first layer because of the integer-VDP. For larger networks, XONN performs better than previous works but when the network is smaller there is a longer runtime. There are some parameters set and assumptions made with regard to neural network weights but overall has a good approach to private inference.

2.5.1.7 Delphi: An Inference System for Neural Networks

This work [65], is designed to perform private inference in the domain of PPML. The threat model considered here is two-party semi-honest settings where one party is corrupted. The corrupted party does not deviate from the protocol but tries to gain information that the party is not supposed to. The authors here introduce Delphi, a secure prediction system that performs the entire neural network operation in the private domain and produces inference results for the client while preserving the privacy of the model (for the server) and privacy of the input (for the client). The problem in the oblivious domain arises due to inefficiency in the performance when we consider both communication and computation costs. The authors have contributed the following as their work. First, they build protocols on top of GAZELLE [52] and reduce the cost of both layer operations, i.e. linear and non-linear layer. This is done by moving the HE computation to an offline phase instead of performing it online. The online phase will only consist of using secret shared values and obtaining the result. The second contribution comes from using a program that they call the planner to alternate between using an activation function which is a non-linear polynomial and an approximated function of that non-linear polynomial. This makes the trade-off between the accuracy and speed equivalent and hence grants an overall better performance. The cryptographic primitives used include HE (linear layer), GC, and beaver triples (non-linear layer). The technique of doing this is building

a planner which runs an algorithm to choose which layer uses the actual ReLU function or an approximated quadratic polynomial. The experiments implementing these techniques showed results of better performance in the online phase because most of the computationally heavy operations are moved to the offline phase. The computation time would be higher in some cases but communication costs are much lower than its predecessor GAZELLE. This work is possible only if the computation in the offline phase is made possible but this would involve taking an extra step and involving the client prior to beginning the protocol officially. Currently, the results provide insight into the efficiency of DELPHI when compared to its predecessors. Since the techniques apply only to neural network inference, having it work for training would be the next task.

2.5.1.8 GALA: Linear Algebra in Privacy-preserved Neural Networks

To achieve privacy for machine learning as a service, many methods have been implemented with various cryptographic primitives. The authors of this work [97], aim to improve the efficiency of the techniques that involve the use of Homomorphic encryption and garbled circuits by improving on methods implemented previously. HE is considered to be the linear layer function which includes polynomial computations such as addition, multiplication, and permutations. Since these operations consume the most time, efficiently choosing the right operations helps in the reduction of the total time taken in computing the operations needed for training and inference of neural network model. Using secret shares to share encrypted values also plays an important role and this operation is also modified for better efficiency. The three cryptographic primitives used are packed HE for linear layer computation and a combination of secret sharing along with oblivious transfer for non-linear layer computation. The row-encoding share multiplication technique proposed improves on a previous technique which based its number of permutations to be proportional

to the number of slots. GALA eliminates HstPerm operations on input ciphertext by bringing about disproportionately between the number of permutation operations and a number of slots for the fully connected layer. As for the convolution linear layer, the previous technique uses convolutional results from each block whereas this is redundant. By manipulating the structure of these blocks it is possible to reduce the number of operations for both the input and output ciphertext blocks. GALA's techniques improve the performance of the deep neural networks by a good margin in terms of speed when the number of layers is higher whether that is convolutional or fully connected but when the number of layers is lower the increase in speed is not by a lot compared to previous works. Hence there is still room for improvement.

2.5.1.9 MP2ML: Mixed-Protocol Framework for Private Inference

In this work [18], the authors consider the domain of private inference for an application that falls under PPML. The setting of the adversary model is semi-honest where the adversary follows the protocol but aims to gain information other than what is to be inferred. After reading through previous works, the authors extend on works of nGraphHE [19] by adding some MPC techniques to it since the nGraphHE technique only involved preserving the confidentiality in linear layers, and hence the non-linear layers which consisted of activation functions were left to be computed in cleartext. This leads to the leakage of information. By adding some MPC techniques to this, they are able to secure the computations for activation functions as well. With this as their background, they propose a framework that is a combination of nGraphHE and ABY [39] (MPC-based computation of ReLU activation) to prevent leakage of weights which also provides high accuracy (when considering the ReLU activation function). The authors claim this to be the first private inference scheme to be a combination of the CKKS encryption scheme along with a secret sharing technique and the experiments are performed on neural networks. As for

cryptographic primitives, HE (CKKS scheme), and MPC (GC protocol and GMW protocol are considered in ABY) are considered. The techniques for this framework are a combination of nGraphHE (pure HE), ABY (pure MPC), and TASTY [48] or GAZELLE [52] (a combination of MPC and HE). To invoke the MP2ML protocol, a single line of code needs to be replaced in the execution of TensorFlow. For plaintext packing of values, MP2ML uses batch-axis packing which helps in better efficiency. This work seems to perform better than previous works in terms of speed but accuracy wise there is a slight degradation due to added security and the use of the CKKS scheme which provides approximate results when compared to the BFV scheme.

2.5.1.10 CrypTFlow: Secure TensorFlow Inference

This framework [57], is built by the authors to convert TensorFlow inference code into MPC. The main intention behind producing such a framework is because prior works are not easily applicable by ML practitioners and they experiment with smaller networks which makes their use difficult. It is also possible to achieve PPML application even in the malicious security setting. This is uncommon because most works consider a semi-honest setting. The framework includes three main components i.e. Athos, Porthos and Aramis. Athos, is a setting-dependent compiler used to compile the tensorflow inference code into secure computation protocols. Currently, it considers ABY-based two-party computation (2PC). Porthos is a semi-honest secure 3-party computation (3PC). Finally, Aramis is a malicious secure three-party computation. The efficiency, as well as performance, varies with the settings chosen (this is provided as an option). Computation is performed in the fixed-point arithmetic domain. The code is converted while maintaining the same accuracy as the tensorflow code. Breaking down each individual section, first, there is Porthos, which is built on top of SecureNN [90] and improved. Porthos uses beaver's triples but

computes by matrix reshaping which is better than SecureNN's approach. The non-linear secret shares for ReLU and maxpool are also cut down by 25 percent which allows for better communication overall. Next, Aramis gives significant importance to hardware because it requires the hardware to provide code attestation and secure signing functionality. The bright side of making Aramis work is that it has a strong adversarial threat model and also can be used for semi-honest secure MPC protocol as well. Performance when compared to previous work shows a decrease in time taken and also in communication cost. For now, Cryptflow works for inference only and not training due to the overhead of MPC protocols.

2.5.1.11 CrypTFlow2: Practical Two-party Secure Inference

This work [76], concentrates on oblivious inference on a two-party system considering neural networks. It is efficient and is the first among its peers to benchmark ImagetNet-scale neural network architecture (which is a complex architecture), for example, ResNet50 and DenseNet121. It takes an order of magnitude lesser. Contributions in this work include new protocols for millionaires and DReLU protocols that enable secure and efficient evaluation of the non-linear layer. A new protocol for division and new theorems for fixed point arithmetic and lastly a framework that allows the users to choose between using HE or OT to evaluate the linear layers of the protocol. For cryptographic primitives, the work is built using HE, OT, and secret-sharing schemes. They also use multiplexers and B2A conversions. Taking a deeper look into the techniques, and the millionaires protocol, some changes are made in the algorithm of the original millionaires protocol such as changing the recursion/Tree traversal. Also, combining two OT calls into a single one removes unnecessary equality computations and realizes operation efficiently. As for the DReLU, an optimized algorithm is proposed named optimized integer ring DReLU in some redundant steps are removed from the original. Finally, some optimized protocols are developed for

division and truncation. This work outperforms its predecessors by using these protocols.

2.5.1.12 GForce: GPU-friendly Neural Network Inference

This work [70], is proposed to solve the issues surrounding latency in oblivious inference protocols which occur due to the application of cryptographic primitives on plaintext computation. Prior works have tackled these issues but either has low accuracy or high latency. This work is built in comparison to previous work such as Delphi [65] by introducing new techniques. The contribution includes an offline/online CPU/GPU design called GForce which introduces high-accuracy networks in low-precision settings, and a suite of GPU-friendly protocols, both offline and online, to solve the problem of non-linear operations. The cryptographic primitives used are additive homomorphic encryption (AHE) with the intention of maintaining circuit privacy and additive secret sharing (SS) or secure online/offline share computation (SOS). Going over the techniques used for GForce, the costly AHE operations are performed offline and secure queries are computed online. The GPU plays a huge role in the computation of non-linear operations such as ReLU and max-pool by breaking performing secure computation of Damgård–Geisler–Krøigaard (DGK) protocol which is broken down into linear operations and inexpensive non-linear operations. By using the stochastic weight averaging in low-precision training (SWALP) for training deep neural networks, it is possible to make use of low bit-width integers. To use this technique, the quantization, and dequantization of values need to be handled (without which computation will be the same as working with floating point numbers). This led to the creation of stochastic rounding and truncation (SRT) layers which used the SWALP technique and built an optimized layer for computation. For the linear layer, the authors make use of an AHE-to-SOS transformation which is a GPU-friendly computation of the linear layer (fully connected layer, convolu-

tional layer) which is an upgrade to the HE-only computation on linear layers. As for the non-linear layer, the main idea is to perform a comparison operation. The protocols built on top of DGK are meant for this purpose and are GPU-friendly. In particular, the GForce design considers SWALP-trained DNNs embodied SRT layers because these are able to divide and wrap around SS efficiently. Results show a huge communication and computation cost in the offline phase but a much faster online phase when compared to previous works. The GForce works most efficiently with SWALP-trained DNNs which might not always be the case. These techniques are currently implemented for ReLU and max pool only which the most popularly used functions.

2.5.1.13 CryptGPU: Privacy-preserving Framework on the GPU

The system in this work [88], was developed for PPML but the key difference between other frameworks and CryptGPU is that it runs all the operations (both linear and non-linear) on the GPU of a computer instead of a CPU. For the threat model, the authors considered a semi-honest setting with data shared between three servers by the client. It is also applicable to setting where a group of clients want to train their model aided by a server. The contributions from this paper include an interface to embed secret shared values into floating point values that can be processed by kernels for linear algebra and it identifies a set of GPU-compliant privacy-preserving evaluations. The results of running such protocols on the GPU have proved to be faster than its CPU variant. It is built on top of PyTorch and CrypTen (a framework built on PyTorch to help PPML researchers). It considers a 3PC setting where data is distributed using secret sharing and tolerates one semi-honest corruption. Since the architecture of CPUs and GPUs are not very similar, various techniques were researched to implement cryptography on the GPU. Some of these included leveraging existing CUDA kernels, GPU-friendly cryptography, systematic evaluation of

GPU-based MPA and an ML-friendly approach to handling data. The framework achieved a shorter run time when it considered deeper models with larger datasets but was outperformed by shallow networks with a comparatively smaller dataset. Also, cryptographic protocols are built to run on CPUs and hence are highly optimized in this regard. With the results from CryptGPU, we can see that we require more optimized protocols for GPUs for better performance on larger datasets and deeper models.

2.5.1.14 Hunter: HE-friendly Structured Pruning Framework

The goal of this work [30], was to build an efficient protocol for private inference because previous works that considered HE for linear layer and GC for non-linear layers do not have real-world implementation applications. With the novel approach used by the authors of this paper, it is closer to a possible application in the real world for private inference. The threat model considered is a two-party semi-honest secure multiparty computation setting (2PC). The cryptographic primitives used are packed HE for the linear layer and GC for the non-linear layer. The contributions of this work mainly are techniques to improve the efficiency of linear layer computation. The main idea behind doing this is to implement network pruning to reduce model size. Although this technique has been used previously, it has not been catered to the structure of SIMD HE. The first technique introduced is the HE-friendly dot product computation (fully connected layers). In this technique, the weight matrix is arranged in such a way that when pruning is applied, the number of homomorphic multiplications is reduced. It is a structural improvement in the weight matrix and it reduced the computation complexity by quite a lot. Next, there is the HE-friendly convolutional pruning. Considering the single input single output (SISO) and multiple input multiple outputs (MIMO) structures in convolution layers, Hunter introduced internal and external pruning strategies which when combined can be

used to reduce the parameters in the convolutional layer and hence reduce the entire network size to a fraction of the original. The results, compared to previous works, have performed much better layer-wise, and overall computation and communication cost was reduced. The network was brought down to only two percent of its original size with almost no loss in accuracy but an improvement in efficiency.

2.5.1.15 Cheetah: Secure Two-Party Deep Neural Network Inference

In this work [49], the authors concentrate on secure neural network inference only and not the training aspect. In particular, it considers two-party semi-honest neural network inference setting for the threat model. Since all the work done in this domain does not provide efficient protocols to the extent of using it for a real-world application (due to communication overhead and computation cost), Cheetah aims to improve the situation by making the computation more efficient and reducing the communication overhead. In terms of contribution, there are two main ones. The first one is a structural design protocol for faster homomorphic encryption operation for the linear layer (convolutional, batch-normalization and fully connected) and the second contribution is a group of lean primitives for the non-linear layer functions such as truncation and ReLU which are communication efficient. Since HE-based homomorphic encryption schemes are operated over the field Z_p, the computation is much slower and inefficient. The idea is to build a structure that uses Z_{2^l} ring which makes computation about 40-60 percent faster and modulo reduction is equivalent to free on CPUs. Three different protocols are introduced in the linear layer to evaluate convolutional, batch-normalization, and fully connected layers via polynomial arithmetic circuits. These encoding functions map the values of coefficients in such a way that the number of rotations that need to be performed is reduced which leads to output polynomials being computed using only a single homomorphic multiplication. Next for the non-linear layer, the authors build an efficient trunca-

tion protocol (because it is quite expensive) and a protocol to find the MSB using vector oblivious linear evaluation (VOLE)-style-OT. Leakage of noise is also handled with secret sharing and coefficient packing techniques. Results show that this work outperformed many of the prior work due to a reduction in communication costs.

2.5.2 PPML: Private Inference and Training

2.5.2.1 CryptoNets: High Throughput and Accuracy Privacy-preserving Framework

In this work [41], the authors consider the domain of private inference PPML. This was one of the early works in this domain. It considers the MNIST dataset and applies the concepts of PPML to neural networks in particular. For the polynomial functions, HE can be used because it only supports addition and multiplication. Since HE does not work in the domain of floating points, the weight values of a model need to be converted to a fixed-point domain. As neural networks involve the operation of activation functions which are non-polynomial, an approximation method is used where-in the non-polynomial activation functions are approximated to low-degree non-linear polynomial functions. Pooling (max-pooling) is another non-polynomial function that is replaced by scale-mean pooling because this also requires a lower-degree function. When it comes to the implementation, the work makes use of the CRT to encode large numbers and perform parallel computation in the encrypted domain. Very few schemes allow this type of parallel computation. This parallel computation technique is labeled SIMD. For training, the network consisted of a nine-layer neural network model which consisted of (in order) convolutional, square activation, scaled mean pool, convolutional, scaled-mean pool, fully connected, square activation, fully connected, and sigmoid activation layer. This shows results where the accuracy of the output only for inference seems to be quite high but they also do mention that making use of HE for complicated approximated

non-polynomial functions reduces the efficiency in terms of the time taken to reproduce these results or infer from an input. To improve on this, making use of MPC techniques for the non-polynomial functions would be better, or from a hardware perspective, making use of GPUs and FPGAs can be used to accelerate the process of computing non-linear layer functions. This technique was only tested on a small dataset and neural network architecture hence it cannot be concluded to be one of the best techniques but certainly gives a direction for future improvement.

2.5.2.2 SecureML: A Scalable Privacy-Preserving Framework

In this work [67], the authors build a new and efficient protocol for PPML and they concentrate on algorithms such as linear regression, logistic regression, and neural networks. It considers 2PC setting for the threat model in which there are two servers between which the data is distributed. The authors contribute the first efficient protocols for logistic regression and neural networks. It consists of an offline data-independent phase and an online phase which is much faster. For multiplication operation, two shared decimal values are represented in the finite field, and multiplication is performed using offline generated multiplication triplets. Then a truncation is performed after which during reconstruction there is a one-bit difference in the least significant bit (LSB) position but does not degrade accuracy. The online phase only consists of multiplication, and bit shifting and the offline phase consists of triples generation. For the MPC functions, for example in neural networks, sigmoid and softmax are some functions that need to be approximated. The contribution for MPC is a new activation function that can be viewed as a sum of two ReLU functions and it is computed using GC. Along with this a custom switching algorithm between arithmetic and Yao's sharing techniques. As for the contribution in the vectorized domain, the authors propose two solutions to deal with linear HE and oblivious transfer which involves the generation of multiple triplets. Finally,

another security model with a much faster offline phase is proposed. In this case, the clients are involved (this only works in a case where we assume that parties are non-colluding, i.e. only a single party is corrupted). The cryptographic primitives used for this work are garbled circuits (2PC) with the oblivious transfer, secret sharing, and multiplication triplets. For the threat model, semi-honest secure two-party computation is chosen. The effect of truncation error was analyzed. In the offline phase, there was LHE-based triple generation and OT-based triple generation. For specific algorithms such as logistic regression, the new activation function needs to cover a range of 0 and 1. Hence, substituting the logistic regression with two ReLUs function on subtraction gives the same functionality. Next, to improve efficiency the client-aided offline protocol was implemented. Due to the involvement of cryptographic operation, the efficiency of triple generation is improved. Results show that the implemented techniques for each algorithm produce a better result than its predecessor. Some assumptions for a client-aided protocol have been made for a faster offline phase.

2.5.2.3 MiniONN: Oblivious Neural Network Prediction Framework

The motivation behind this work [59], was to build a framework to transform an existing neural network into an oblivious neural network supporting PPML predictions with good efficiency. The threat model considers a generic setting of semi-honest adversaries for cloud-based services where the server holds the neural network model and the client holds the input. It did not require any special change in how the neural network was trained or change in its structure. The protocols for the linear layer, non-linear layer, and pooling layers were picked and optimized to convert a plaintext neural network to an oblivious neural network. The authors divided the entire process into two phases i.e. offline and online phases. The offline pre-computation phase involves performing request-independent operations using AHE combined with the

SIMD batch technique. The cryptographic primitives used for this work are MPC and HE. The design is divided into multiple parts. The first part is the dot product triplet generation. This step involves generating multiplication triplets. For the next step, oblivious linear transformations are done after homomorphic SIMD operations. This is then followed by an oblivious linear transformation, activations, and pooling operations. For performance, a lot of parameters have to be set and varying these parameters trade-off between security and efficiency. They use an algorithm called limited-memory MFGS optimization for improvement in accuracy when compare to its predecessor, SecureML. It achieves good efficiency in online as well as offline phases. The limitation comes down to the size of the neural network architecture since operations become more complex as the number of layers increases.

2.5.2.4 SecureNN: Three-party Secure Computation for Neural Network Training

This work [90], considers the domain of both oblivious inference and training. The threat model is a semi-honest setting with three parties interacting for computation. It also provides security against malicious adversary settings. The contributions are a scalable framework that provides very high accuracy on the MNIST dataset and it outperforms predecessors of two and three-party settings that consider smaller networks. Since the authors provide a protocol that works in both semi-honest and malicious party settings, it is the first to provide security for malicious adversaries for neural network inference and training. Each of these contributions provides a wide range of individual components or modules that make up the entire neural network operation. The cryptographic primitives used are Beavers triplets, HE, and Yao's GC protocol. For the main protocol, supporting protocols are first built as modules. The following supporting protocols help in neural network operations: matrix multiplication, select share, private compare, share convert, and MSB computation.

The overhead of supporting protocols is important to keep track of. The protocol runs over the ring Z_L but in some cases, they have to be in Z_{L-1}. Beavers triples are used to compute matrix multiplication which is generated using pseudorandom generate functions (PRFs). This is divided into shares and used locally. Hence this reduces multiplication communication by half. As for the main protocols, the linear and convolutional layers can be computed by supporting the matrix multiplication protocol. Then the computation of ReLU and its derivative is important for the non-linear layer. The derivative of ReLU depends on MSB protocol hence ring Z_L has to be converted to Z_{L-1} whereas ReLU is computed over Z_L. For division, the protocol implements long division calculated in a bit-by-bit sequence. It is possible to compute max-pool and its derivative which requires conversion between L and a unit vector that holds the value. Finally, it is all put together and results are gathered. By avoiding the use of GC the communication and runtime costs are greatly reduced with high accuracy.

2.5.2.5 Falcon: Honest-majority Maliciously Secure Framework

The authors in this work [91], build this framework to enable private and efficient training and inference on neural networks. Multiple works have been published in this domain but there are either efficiency issues or security consideration issues. The threat model for this work is malicious security which provides a strong security guarantee in an honest-majority adversarial setting. The authors contribute the following to this work: A protocol that provides security in malicious security settings and a more efficient protocol in the semi-honest setting, new protocols for ReLU (they use these in a smaller ring so communication costs reduce). This also leads to efficiency in the non-linear layer without the need for switching between various types of sharing protocols. This work is also the first of its predecessors to introduce an efficient block for batch normalization. The protocol also has the capability to

handle complex neural networks. The overview of the protocol with some technical details such as the build, techniques used parameters, etc are as follows. The system model is a 3PC machine learning service. The first phase is the training phase where three servers share the data received from the user using secret sharing techniques and the second phase is the inference phase. For cryptographic primitives, the work uses secret sharing MPC protocol for linear and non-linear layer operations (replicated secret sharing). Fixed point encoding is used with 13 bits of precision and multiplication protocol is done over the ring Z_p. As for the protocol and building blocks, the linear layers i.e. fully connected and convolution are taken care of by using correlated randomness with 3-out-of-3 randomness and 2-out-of-3 randomness. After performing the necessary operations reconstruction is done with select shares. Next, private comparison in which two primary parties blind and mask their inputs after which it is sent to a third party and this is possible due to stronger security provided in FALCON. The wrap function along with the specific protocol built for ReLU and its derivative are used in computing DReLU and ReLU. Finally, optimized protocols for max pool, a derivative of max pool, division and batch normalization are also implemented. Results show that this work is faster than prior works and communication efficient as well in the inference and training phase.

2.5.2.6 Trident: Efficient 4PC Privacy-preserving Framework

This framework [32], considers a four-party (4PC) setting for machine learning algorithms, which are, linear regression, logistic regression, and neural networks. The threat model for Trident is a four-party setting with at most one malicious corruption. The framework is built to switch between different operations such as arithmetic, boolean, and garbled operations depending on the need at the time. Data is secretly shared among parties in an outsourced setting for machine learning. The contributions of this framework are fast mixed-world computations, efficient trunca-

tion, secure comparison, and ML building blocks. The protocol requires three ring elements in the online phase instead of four (for each multiplication) and hence the complexity does not increase. Fast mixed-world computation makes use of an additional party to increase the computation speed in the online phase the aim is to achieve 128-bit computational security and for every conversion a switch is proposed between worlds which is more efficient. Efficient truncation uses a reduced number of circuits that are expensive to compute and can be combined with the multiplication protocol. The implementation of this technique does not affect multiplication costs online and removes the need for any circuits in the online phase. The secure comparison uses an optimized parallel prefix adder (PPA) in a 64-bit ring for online rounds. Out of the four parties, only one behaves as an evaluator and the rest are garblers. It also has protocols to switch between sharing protocols. The stated protocols are tested over LAN and WAN. The online communication cost is improved by one ring element compared to previous work It can be observed that, by adding an extra honest party to a 3PC setting, significant improvements can be achieved in the protocol.

2.5.2.7 Blaze: Privacy-preserving Machine Learning Framework

This work [74], considers a three-party (3PC) setting in the domain of PPML accommodating a maximum of one malicious party over a ring and obtains results through secret sharing. The framework consists of a three-layer hierarchy of primitives. The first layer consists of basic operations (multiplication, bit extraction, bit-to-arithmetic conversion), the second layer consists of high operations (dot product, truncation, sigmoid, and ReLU activation functions) and the final layer consists of the algorithms that were tested for the BLAZE framework i.e. logistic regression, linear regression and neural networks (inference only). The framework has an input-independent pre-processing phase and a fast input-dependent online phase.

Expensive operations are performed in the preprocessing phase and the online phase begins when inputs are available. BLAZE only covers private inference but not the private training aspect.

The work of this paper aims to provide an efficient PPML framework that provides a stronger security guarantee of fairness i.e. out of all parties participating, every honest party gets the output whenever the corrupt party does get the same. This work builds upon the basic blocks for cryptographic primitives and improves the efficiency of each of these blocks. The techniques are first broken down into three layers with each layer depending on the previous layer. Algorithms such as Linear regression, logistic regression and neural networks are built using basic operations such as dot product, truncation, activation functions, etc. Blaze has considered such operations and improved the way the technique has been implemented. The work covers multiplication, bit extraction, and a bit to arithmetic for the first layer primitive and each of these primitives has a protocol implemented. For the second layer dot product, truncation, sigmoid, and ReLU activation functions are built with their own protocol. Finally, these are used to perform the efficient computation of results for linear regression, logistic regression, and neural network algorithms. Although this does provide a boost to the previous work in the same domain, this only covers the inference for all three algorithms and training for linear and logistic regression. Training for neural networks requires GC and has been put aside for the future.

2.5.2.8 ABY2: Improved Mixed-protocol Secure Two-party Computation

This work [73], is built in comparison to the previous work ABY [39], which is an efficient mixed protocol for oblivious operations. The applications of the proposed work include AES S-box, circuit-based private set intersection, biometric matching, and privacy-preserving machine learning. The work improves on both training and

inference from previous works. The threat model is a semi-honest secure two-party computation (2PC). The main contribution is an efficient 2PC protocol built over an l-bit ring denoted by Z_{2^l}. k denotes the computational security parameter. Parameters used are $l = 64$ and $k = 128$. It only requires the communication of two ring elements for each multiplication in the online phase. The improvement has been made in online rounds of ABY from two to one for most conversions between arithmetic, boolean and Yao's secret sharing protocols. The building blocks to implement the protocol includes the scalar product, depth-optimized circuits, matrix multiplication, comparison, non-linear activation functions, and maxpool. The overview of the steps involved in performing the functionality are sharing all the inputs (sharing protocols), gate-by-gate evaluation (secret sharing and multiplication protocol) and output reconstruction (reconstruction protocol). Optimizations were made to the standard conversions to build an Efficient conversion protocol between arithmetic, boolean, and Yao's secret shares. This was done by adopting only OT only in the setup phase (unlike ABY which adopted OT during the online phase). When comparing results with works like SecureML, there is an overhead of 1.6 percent for logistic regression and 0.7 percent for neural networks. But there is a runtime improvement of high magnitude.

2.5.2.9 SPINDLE: Scalable Privacy-preserving Distributed Framework

The system developed in this work [45], maintains the privacy of data as well as the model. The threat model it considers a passive-adversary model with up to $(N-1)$ colluding parties. The cryptographic primitives used here are multiparty homomorphic encryption and secret sharing. As for the contributions, SPINDLE is built by taking distributed learning or federated learning as an inspiration and is built to perform the operations of the complete Machine learning flow. It allows cooperative gradient descent (stochastic mini-batch gradient descent) along with evaluation of

the model obtained while preserving the data and the model's confidentiality. For the training part, it uses a technique called MapReduce abstraction which defines the distributed ML process. It has four steps: PREPARE, MAP, COMBINE, and REDUCE. The data holders agree on the parameters of the protocol and initialize the cryptographic keys while checking distribution homogeneity this step (PREPARE), trains the model locally. It relies on HE and least-square polynomial approximation for sigmoid functionality (MAP), then the results are combined homomorphically in a tree structure. The root of the tree has the encrypted combined weights (COMBINE) and finally is used to collectively update the global model by broadcasting it to the rest of the data-providing parties (REDUCE). The system proposed has a trade-off with accuracy but not privacy, unlike other models which cannot provide complete data and model privacy. The system is quantum resistant. It makes use of multiparty cryptographic schemes, SIMD operations, and optimized polynomial approximations for activation functions. Specifically, the CKKS HE scheme is used. By replacing some costly homomorphic operations with other protocols, SPINDLE has achieved better performance than other centralized and distributed systems.

2.5.2.10 Soteria: Privacy-preserving Machine Learning for Apache Spark

The aim of this work [26], is to protect the confidentiality of the data as well as the model so users can take part in the operations of PPML. The threat model considered here is honest-but-curious. The cryptographic primitive which SOTERIA is built around is garbled circuits protocol. Efficient training algorithms for efficient cryptographic primitives that are modified to work in favor of performing operations with less overhead. To evaluate the inference circuits the important factors are model parameters and network structure and hence SOTERIA searches and selects optimal parameter sparsity and network structure. FHE schemes are only restricted to computing linear operations efficiently. To perform non-linear operations, polyno-

mial approximation needs to be used. Even switching protocols are computationally heavy. Due to this, GC was picked. To make these operations possible, the authors make use of binary neural networks. The linear layer computations are calculated using XNOR-pop count. For the search algorithm, NAS is used and it consists of the following components: The search space, search strategy, and performance estimator. The algorithm is regularized and guided to identify a configuration for cells that give optimized model accuracy and performance of private inference. Next moving to a ternary neural network, it is a sparse binary neural network which is a neural network with reduced parameters. Models are trained with ternary parameters and binary activation functions. In some cases, the accuracy was reduced but run time and communication cost were lower.

2.5.2.11 SWIFT: Fast and Robust Privacy-preserving Machine Learning

This work [56], introduces a PPML framework in the secure outsourced computation paradigm. The framework is robust and works on multiple ML algorithms. The framework considers a 3PC setting initially and is then extended to a 4PC in which the threat model chosen is a maliciously secure setting and it assures fairness and guaranteed output delivery (GOD) in honest majority by eliminating a potentially corrupt party while previous works only guaranteed fairness. The system is tested for logistic regression and neural network ML algorithms over the 64-bit ring. The main contribution of this work is the robustness it provides over fairness because fairness alone is not enough to have a worthy service. It provides GOD in an honest-majority setting and is the first known of its type. The cryptographic primitives used are secret sharing protocols for both training and inference (arithmetic and boolean sharing) A new protocol is introduced to identify an honest party (trusted third party) which reduces the need for pivotal private communication with greater cost. This is the joint message passing primitive which has a send and verify phase. Another protocol

is built for multiplication keeping GOD in mind and following the same structure as that of BLAZE [74] It has a preprocessing phase which is an offline and online phase. Some building blocks used to make this 3PC protocol possible are the reconstruction block, MSB extraction, bit-to-arithmetic, bit injection, dot product, truncation and secure comparison. By extending it to 4PC, efficiency increases, and finally, robust protocols for input sharing and output reconstruction phase are used. Similar blocks are used for 4PC but with some changes. The reconstruction protocol is used in the online phase which is much faster than previous work.

2.5.3 Privacy-preserving Federated Training and Inference

2.5.3.1 PRIV-FL: Practical Privacy-preserving Federated Regression

The authors present a protocol in [61], to train a model and perform oblivious prediction in a FL setting with the centralized server using gradient descent to prevent leakage of input or model data. The two contributions are protocols for logistic regression and linear regression which are dropout robust regression training protocols. For cryptographic primitives, the work is built on AHE and secure aggregation protocol. For the threat model, the authors consider a semi-honest adversary setting where the clients or server follow the protocol but try to learn about the model or input data. As for the protocol, first, a multiple two-party shared local gradient protocol is performed in parallel after which a global gradient share reconstruction protocol is performed. The training process begins with the server choosing a set of users to broadcast its model and privately computing the two shares of the local gradient with each party. The server and users then compute an aggregated share after which the server computes the global gradient from the local shares and updates the global model. The datasets chosen were different for both logistic regression as well as linear regression.

2.5.3.2 POSEIDON: Private Federated Neural Network Learning

This work [82], considers a FL setting that is different from the usual data-sharing frameworks which is an extension of SPINDLE. POSEIDON is built to be a quantum-resistant privacy-preserving framework that relies mainly on multiparty HE techniques for N-party settings. The cryptographic primitives used are multiparty homomorphic encryption schemes and secret sharing. Since cryptographic protocols are computationally heavy and complex, optimizing the use of these protocols and the way they are used is important The authors of this work have the following contributions. POSEIDON is a privacy-preserving, federated-based quantum-resistant training and inference system for neural networks with N-party consideration. They propose a new approach for packing which improves the efficiency of the SIMD protocol. An optimization problem that helps choose the correct parameters for training and evaluation. It improves on aspects such as flexibility, security, and scalability. It is flexible because of the federated learning setting where there is no limit to the number of parties who train the model. The data and model confidentiality is secured in a passive adversarial setting whereas MPC-based solutions restrict the number of parties. Scalability is the communication cost that is linearly proportional to the number of parties. The system uses the CKKS HE scheme and implements packing techniques to pack multiple ciphertexts to perform operations simultaneously which helps in optimizations. POSEIDON covers both privacy-preserving inferences as well as privacy-preserving training for neural networks. Compared to its predecessor SPINDLE it has performance improvements as well as quantum resilience.

2.5.3.3 PROV-FL: Round Optimal Verifiable Private Federated Learning

The authors of the paper [38], concentrated on the domain of FL where a global model is trained by aggregating local models. Privacy concerns arise due to the

server being able to access the local model which leaks data. PROV-FL aims to solve this problem by proposing a single-round secure aggregation protocol and introducing a differentially private solution to provide a balance between efficiency, privacy, and accuracy. The setting considered is semi-honest with three different scenarios of colluding parties. For cryptographic primitives, multi-key linearly homomorphic authenticators, AHE, authenticated encryption, key derivative function, and differential privacy. They introduce two protocols, one which involves a single aggregator scenario and another which is a more general and a multi-aggregator scenario. The single aggregator protocol includes a key setup phase where the server generates the protocol parameters such as session ID, session Key, etc and the users derive these parameters. Next, the server initializes the model followed by the users training the model locally. The aggregator aggregates the model from each user and for the final step, the server computes the updated global model. These steps are repeated until it possesses the final global model. The multi-aggregator protocol is a generalization of the single-aggregator protocol in a real-world scenario and is still a single-round protocol that does not require the server to run multiple iterations. The protocol also supports dynamic joins and dropouts and considers various scenarios for users joining, dropping out, or either of the two. The datasets considered are MNIST, fashion-MNIST, IMDB movie review, and the rice image dataset. This work is the first of its kind to have a single-round protocol with dynamic participation.

2.5.3.4 Eiffel: Ensuring Integrity for Federated Learning

In this work [79], the authors target to secure the integrity of the model updates made during the process of FL. Since there are multiple updates being made by the client, there are chances of malformed updates which might lead to the poisoning of the entire global model. For the threat model, they consider a malicious server that deviates from the protocol to recover updates and malicious clients who either

poison the model or fail the integrity check. The work introduces a secure aggregation with verified inputs (SAVI) protocol that was built to ensure privacy and integrity. For cryptographic primitives, privacy is ensured using Shamir's threshold secret sharing scheme and for integrity, secret shared non-interactive proofs (SNIP) along with verifiable secret shares (VSS) are used. Other primitives such as authenticated encryption and arithmetic circuits are also used. The entire protocol is summarized as a four-round protocol where the first round announces public information to all the parties and the second round involves generating and distributing all the proofs after which in round three these proofs are verified under the supervision of the server and finally the round four which computes the aggregation to produce the final global model. Each of these rounds is extremely detailed in the paper. The technical novelty can be described as an extension to SNIP while considering a fully malicious threat model and for a single server setting. In terms of experimental results, this work outperforms its predecessor in terms of performance and improves with the number of clients. They consider various poisoning attacks some of which are sign flip attacks, scaling attacks, additive noise attacks, and more. For datasets, they consider MNIST, EMNIST, FMNIST, and CIFAR-10 and test them with three different classification models, which are LeNet-5 ResNet-20, and a customized five-layer convolutional neural network similar to LeNet-5.

2.5.3.5 Efficient Differentially Private Secure Aggregation in FL

The work [86], considers the FL setting and looks to improve on differentially private techniques proposed previously by using a variation based on learning with errors. The technique proposed is named federated learning differential privacy (FDFL). Their work has three contributions: (1) A malicious-secure aggregation protocol with improved performance compared to its predecessors; (2) a new protocol that provides differential privacy in the presence of malicious servers; and (3) clients along

with results that show the scalability of the protocol along with good accuracy. The threat model considered is a semi-honest setting where the client and server follow the protocol but try to gain information passively. It also considers malicious settings where clients and servers actively interfere with the data. Multiple protocols are described such as the distributed noisy batch gradient, secure vector aggregation, and Shamir's reconstruction with verification. Finally, the malicious secure FLDP protocol proposed by the authors combines the protocols to produce a noisy gradient output that is secure in a malicious party setting. The model accuracy is trained on MNIST and CIFAR-10 datasets. This was in comparison to the previous work which was built to work with a trusted server whereas FLDP considers an untrusted server. The accuracy shows that FLDP is viable and can be considered a valid alternative in a different setting.

2.6 Conclusion

In the literature review, first, we see the various domains we cover in this thesis and detailed information about the topics to understand how we proceed in the domains. We highlight the importance of ML, FL and why we focus on these domains. Adversarial machine learning causes a violation of privacy and hence, we are motivated to protect data privacy and model privacy in ML and FL. We also find several works in the domain of PPML and PPFL. All the works use HE or MPC or DP or a mix of these techniques to implement their protocol. Some perform optimizations on the operations for the linear layer and some in the non-linear layer. The authors combine various cryptographic primitives to achieve the best performance for the functionality. Although this is the case, there is less work in the FL setting to optimize the protocols in the linear and non-linear layers, which would efficiently execute the operations.

Chapter 3

Background

This chapter provides details on the background which is essential to understand the design of our proposed protocols. The information from these preliminaries is used to lay the groundwork for the working of our protocols and understand their applications.

3.1 Background: Machine Learning

We present the following concepts of machine learning training from [87].

3.1.1 Building Blocks for a Deep Neural Network

We consider a deep neural network (DNN) of ℓ layers and denote the model as $\boldsymbol{\theta} = \{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \cdots (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\}$ where $\mathbf{W}^i$ is the weight matrix and $\mathbf{b}^i$ is the bias vector, $0 \leq i \leq \ell - 1$. The neural network training uses multiple functions and runs the input data through multiple layers made of neurons. The neurons make up the input layer, hidden layers and the output layer of a neural network. All these layers consist of a combination of activation functions and linear transformations (such as vector or matrix multiplication). In the training process, the gradient computation for input has two main phases: the *forward propagation phase* (or forward

pass) and the *backward propagation phase* (or the backward pass). Each phase of the neural network implements a linear layer and a non-linear layer for each hidden layer. Below we describe each layer in detail.

Linear Layer Computation. Given the weight matrix and the bias vector $(\mathbf{W}^i, \mathbf{b}^i)$ for a linear layer and an input $\mathbf{a}_i$ to it, the output of the linear layer is $\mathbf{c}_i = \mathbf{W}^i \cdot \mathbf{a}_i + \mathbf{b}^i$, $0 \leq i \leq \ell - 1$, where $\mathbf{W}^i \cdot \mathbf{a}_i$ is the matrix-vector multiplication.

Activation/Non-linear Functions. Activation functions are used after the execution of each linear layer. These functions are introduced to bring non-linearity to the entire computation. This is an important feature of neural networks that differentiates them from other machine learning algorithms. The non-linearity in the model plays a major role in understanding the data better and allows for an appropriate model weight estimation which consequently leads to better accuracy in predicting the output classes. The forward propagation utilizes the activation functions while the backward propagation utilizes the derivative of the respective activation function. There are many activation functions but the most commonly used ones are summarized below.

1. **Sigmoid.** The sigmoid function (logistic) is a differentiable monotonically increasing function. It takes a real-valued number that can be of any magnitude and maps it to a range between $[0, 1]$. For negative numbers, it maps the value to 0 and for large and positive numbers, it maps the value to 1 as seen in Figure 3.1. The equation for Sigmoid is as written below:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where σ is the activation function, z is the input to the activation function.

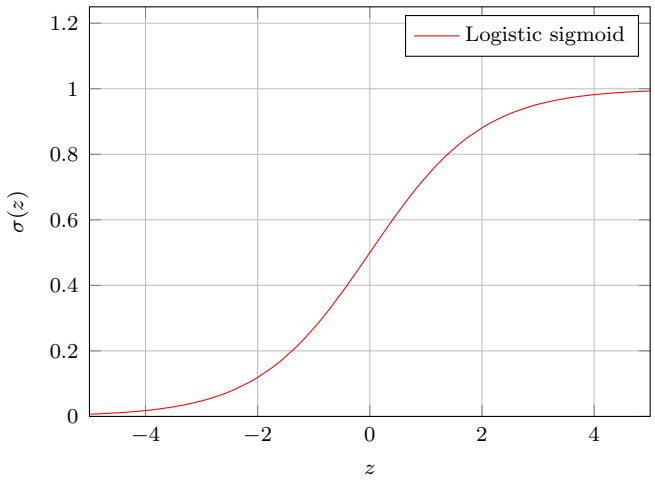

Figure 3.1: Logistic Sigmoid Activation Function

2. **Hyperbolic Tangent.** The hyperbolic tangent (tanh) function is another differentiable monotonically increasing function. It takes in a real-valued input of any magnitude and maps it to a range $[-1, 1]$ seen in Figure 3.2. *tanh* is chosen when small incremental or decremental changes are required in data modeling. The equation is given below:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

where tanh is the activation function, z is the input to the activation function.

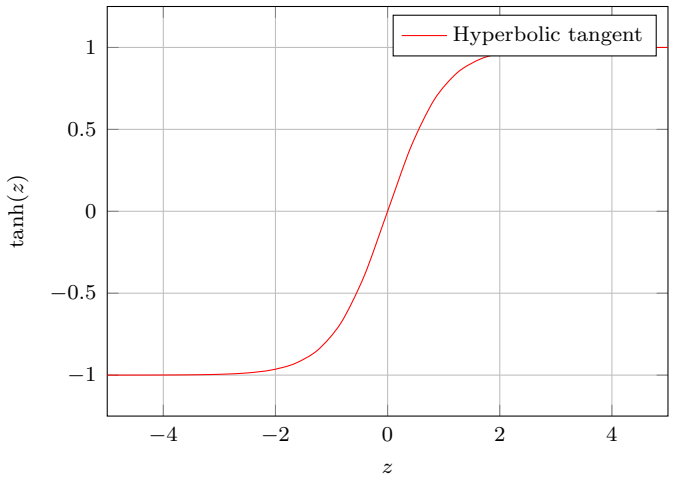

Figure 3.2: Hyperbolic Tangent Activation Function

3. **Rectified Linear Unit.** Rectified linear unit (ReLU) is currently the most popular non-linear activation function being used and it is represented as Figure 3.3. ReLU is preferred because of the improvement in performance as well as simplicity in its usage. It is defined below:

$$ReLU(z) = max(0, z)$$

where $ReLU$ is the activation function, z is the input to the activation function.

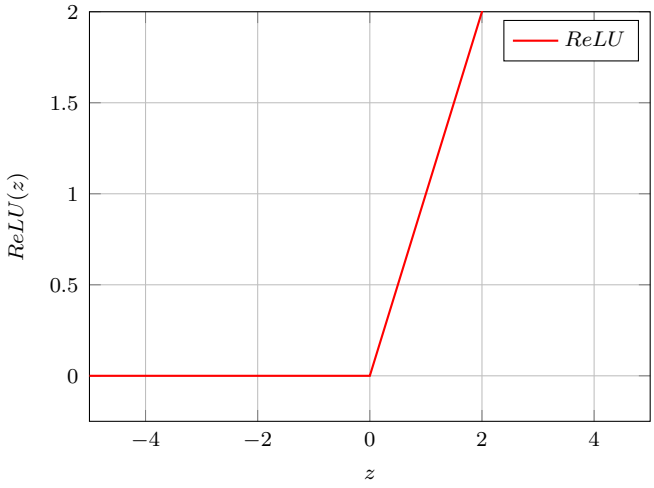

Figure 3.3: Rectified Linear Unit Activation Function

4. **Softmax.** The softmax function is mostly used for categorical probability distribution. This classifies the final layer nodes of the neural network into output classes containing the probability for each class. The function is defined as follows:

$$\sigma_{\mathsf{sm}}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where σ_{sm} is the activation function and z_i is the input where $i \in (1, 2, 3, \ldots, K)$ and K is the total number of data points.

Loss Functions. Loss functions are used in determining the difference between the computed value and the target value. Using the loss function, the model is able to back-propagate the loss and adjust the values such that they are closer to the target value. Common loss functions are the mean squared error and cross-entropy. They are explained below:

1. **Mean Squared Error Loss.** Mean squared error loss (MSE) is calculated by taking the squares of the differences between the actual and computed values. The output will be positive at all times. It is usually used in regression-type algorithms. The equation is given below:

$$MSE = \frac{1}{D}\sum_{i=1}^{D}(y_i - \hat{y}_i)^2$$

where D is the number of output classes, $\hat{y}_i$ is predicted class values and y_i is the actual class values.

2. **Cross-Entropy Loss.** The cross-entropy (CE) loss function (logarithm loss) is mostly used for classification between multiple output classes. It works by penalizing the probability difference between the target and computed values. It

is also called logarithmic loss because the function uses a logarithmic approach to calculate the difference which gives appropriate results for classification.

$$CE = -\sum_{i=1}^{M} y_i \log(\hat{y}_i)$$

where M is the number of classes, y_i is the actual class values, $\hat{y}_i$ is the predicted class value.

3.1.2 Training a DNN

A key task in training a DNN is to compute the gradient on each input in the dataset. We now summarize the main steps to compute the gradient for a data point $(\mathbf{x}, y)$.

Forward Propagation. In the forward propagation phase, the input data is passed through linear and non-linear layers to learn about its features and store these values in its neurons (also known as weight matrices) to use later in the backward propagation phase. In the linear layer, given an input vector $\mathbf{x}$ (with $\mathbf{a}_0 = \mathbf{x}$), the neural network learns the feature of this input data by running it through a linear transformation with the weight matrices $\mathbf{W}^i$ and bias vector $\mathbf{b}^i$ corresponding to that particular hidden layer [87]. For the i-th layer with $0 \leq i \leq \ell - 1$, the linear layer computation is given by:

$$\mathbf{c}_i = \mathbf{W}^i \cdot \mathbf{a}_i + \mathbf{b}^i \text{ with } \mathbf{a}_0 = \mathbf{x},$$

and the non-linear layer computation is:

$$\mathbf{a}_{i+1} = \sigma(\mathbf{c}_i),$$

where σ is the activation function, and in the last layer it is the softmax function. At the final layer, the output is $\hat{y} = \mathbf{a}_\ell$.

Backward Propagation. In the backward propagation phase, the stored weight matrices are updated by calculating the error in the final output layer $\hat{\mathbf{y}}$ using a loss function and back-propagating the loss through the entire network in reverse order. This phase also consists of the linear layer and non-linear layers where the linear layer computes the error $\boldsymbol{\gamma}_i$ using the value computed by the loss function in a linear transformation. The non-linear layer $\boldsymbol{\alpha}_i$, computes the error using the derivative of the respective activation function used in the forward propagation phase [87]. In the backward propagation, for the i-th linear layer computation is:

$$\boldsymbol{\gamma}_{\ell-1-i} = \left(\mathbf{W}^{\ell-1-i}\right)^T \boldsymbol{\alpha}_{\ell-1-i}, 0 \leq i \leq \ell - 2,$$

where $\left(\mathbf{W}^i\right)^T$ is the transpose of the weight matrix $\mathbf{W}^i$ and $\boldsymbol{\alpha}_2 = \mathbf{a}_3 - y$. The non-linear layer computation is:

$$\boldsymbol{\alpha}_{i-1} = \sigma'(\mathbf{c}_{i-1}) \odot \boldsymbol{\gamma}_i,$$

where σ' is the derivative of σ and $\odot$ is the component-wise multiplication or the Hadamard product of two vectors.

Gradient Computation. Using the forward and backward propagation computations, the gradient for the weight matrix and bias vector is computed as $\nabla \mathbf{b}^{\ell-1-i} = \boldsymbol{\alpha}_{\ell-1-i}$ and $\nabla \mathbf{W}^{\ell-1-i} = \boldsymbol{\alpha}_{\ell-1-i} \cdot \mathbf{a}_{\ell-1-i}^T$ for $0 \leq i \leq \ell - 1$. Depending upon different gradient algorithms, a gradient on the dataset is computed to iteratively update the model. Three well-known gradient descent algorithms are mentioned below.

Example 3.1.1. Consider a three-layer DNN where the model parameters are de-

noted as $\boldsymbol{\theta} = \{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), (\mathbf{W}^2, \mathbf{b}^2)\}$. Given an input $\mathbf{x}$, the gradient computation is given in Figure 3.4 where $\nabla\mathbf{W}^i$ and $\nabla\mathbf{b}^i$ denote the gradient for $\mathbf{W}^i$ and $\mathbf{b}^i$, respectively, for $0 \leq i \leq 2$ and $\mathbf{a}_3 = \hat{\mathbf{y}}$. □

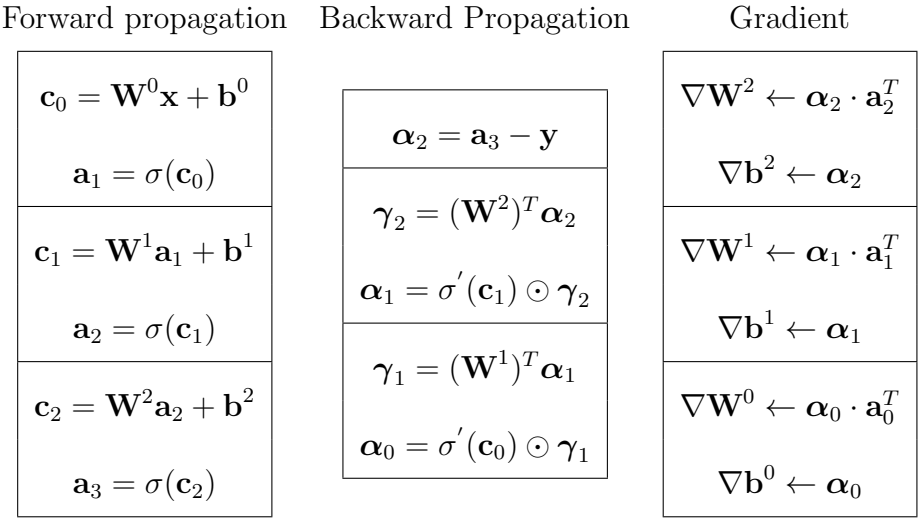

Figure 3.4: Gradient Computation for a Two-layer DNN for an Input $(\mathbf{x}, \mathbf{y})$

Gradient Descent Algorithms. The gradient descent (GD) algorithm is used to update the weights using the error computed by the backpropagation phase. There are different types of algorithms such as batch gradient descent (BGD), stochastic gradient descent (SGD), and mini-batch gradient descent (MBGD). These optimization algorithms are used to find the local minima and minimize loss functions to obtain accurate models, as seen in [7].

1. **Batch Gradient Descent.** Batch gradient considers an entire batch of data samples and updates the model only after it minimizes the loss function for all the samples as elaborated in [80] and [93].

$$\theta = \theta - \alpha\frac{1}{m}\sum_{i=1}^{m}\nabla_\theta J_i(\theta)$$

where α is the learning rate, m is the number of training examples and $J_i(\theta)$

is the loss function for i.

2. **Stochastic Gradient Descent.** Stochastic gradient descent considers random samples from the input and minimizes the loss function. It updates the model for every sample until all samples are covered as in [80] and [93]. It can be represented by:

$$\theta = \theta - \alpha \nabla_\theta J_i(\theta)$$

where α is the learning rate, $J_i(\theta)$ is the loss function for example i.

3. **Mini-batch Gradient Descent.** Mini-batch gradient descent divides the data samples into small batches and minimizes the loss function. It updates the model for each batch until all samples are covered as in [80] and [93]. This is represented by:

$$\theta = \theta - \alpha \nabla_\theta J_b(\theta)$$

where α is the learning rate, θ are the model parameters, $J_b(\theta)$ is the loss function computed on a mini-batch of size b and $\nabla_\theta J_b(\theta)$ is the gradient with respect to θ.

In summary, the batch gradient descent processes all training data at once while stochastic gradient descent processes one example at a time. Mini-batch gradient descent processes small batches of data at once which reduces noise compared to SGD while still being computationally efficient.

This entire process is conducted for a given number of epochs and iterations until satisfactory accuracy is obtained and the model is ready for use.

3.1.3 Approximation functions

Activation functions and loss functions in the non-linear layers of a neural network tend to deal with exponentiation and mathematical operations. The FHE schemes are incapable of computing operations, such as exponentiation functions efficiently. As used in the literature, e.g., [67], we use an approximation function of the activation function to enable an efficient privacy-preserving computing technique such as garbled circuit or FHE. Although this might cause a reduction in the accuracy, the reduction is relatively minor if the function is approximated appropriately. In our proposed private training protocol, we consider approximations of the sigmoid and softmax activation functions.

Sigmoid Approximation. In our proposed private training protocol, we consider the following sigmoid approximation function from [67]:

$$\sigma_{\mathsf{apx}}(x) = f(x) = \begin{cases} 0 & x < -\frac{1}{2} \\ x + \frac{1}{2} & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 1 & x > \frac{1}{2}. \end{cases} \tag{3.1}$$

The derivative of the above approximated sigmoid function is:

$$\sigma'_{\mathsf{apx}}(x) = f'(x) = \begin{cases} 1 & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Softmax Approximation. In our proposed private training protocol, we consider the softmax approximation from [67] given by:

$$\sigma_{\mathsf{s}mapx}(x_i) = f(x_i) = \begin{cases} \frac{\mathsf{ReLU}(x_i)}{\sum_i \mathsf{ReLU}(x_i)} & \text{if } \sum_i \mathsf{ReLU}(x_i) > 0 \\ 1/L & otherwise \end{cases} \tag{3.3}$$

where $L = \dim(x)$ is the number of possible classes and ReLU is defined as follows:

$$\mathsf{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & otherwise \end{cases}$$

And next in [12] the softmax approximation is given by:

$$\sigma_{\mathsf{s}mapx}(x_i) = f(x_i) = \frac{1 + x_i + 0.5x_i^2}{\sum_j^K 1 + x_j + 0.5x_j^2} \tag{3.4}$$

for $i = 1, \ldots, K$ and $x = (x_1, \ldots, x_K) \in \mathbb{R}^K$ and the Taylor series of e^{x_i} is given by:

$$e^{x_i} = 1 + x_i + 0.5x_i^2.$$

3.2 Background: Cryptographic Techniques

In this section, we provide a background on the cryptographic privacy-preserving primitives and protocols that we use to design our protocols. The techniques we employ are additive secret sharing [84], garbled circuit [95], homomorphic encryption scheme [68] and secure aggregation protocols [22].

3.2.1 Additive Secret Sharing

Additive secret sharing protocol is an MPC technique wherein a secret value is split into several shares and given to the parties involved in the MPC computation such that each party learns nothing about the secret. When all shares are combined, the original secret can be reconstructed [94]. Additive secret sharing is defined over a field $\mathbb{F}$. An additive secret-sharing technique can be implemented using only the addition operation. Additions are cheap in comparison to multiplication. An additive secret sharing scheme is defined over a finite field $\mathbb{Z}_2$, which is suitable for boolean

circuits and some are defined over finite ring $\mathbb{Z}_{2^l}$ or finite field $\mathbb{F}$ where operations are performed on arithmetic circuit.

Suppose $x \in \mathbb{Z}_p$ is a secret, where p is prime. In a two-party additive secret sharing, a share for x is created as $[x_1, x_2]$ such that $x = x_1 + x_2 \mod p$. The security of an additive secret-sharing scheme is defined as follows. It is impossible to obtain the secret value x from either x_1 or x_2. If both x_1 and x_2 are known, one can easily construct x as $x = x_1 + x_2 \mod p$.

3.2.2 Oblivious Transfer

Oblivious transfer (OT) plays an important role in secure multiparty computation such as garbled circuit. A 1-out-of-k oblivious transfer protocol is denoted as $\binom{k}{1}-\text{OT}_l$. To execute the protocol, the sender inputs k messages $m_0, m_1, ...m_{k-1} \in \{0,1\}^l$, each of l bits, while the receiver inputs an index $i \in \{0, 1, ..., k-1\}$. By the end of the execution of the protocol, the receiver obtains m_i and the sender receives nothing. From the protocol, the sender learns nothing about the input of the receiver, i.e., i, and the receiver learns only the message m_i, nothing else.

A 1-out-of-2 oblivious transfer, denoted as $\binom{2}{1} - \text{OT}$, is a special case of the above-mentioned protocol, which is run between two parties. Suppose Alice holds 2 messages m_0 and m_1 and Bob holds a bit $b \in \{0, 1\}$. At the end of the protocol $\binom{2}{1}-\text{OT}$, Bob learns the value m_b which corresponds to the bit b, while Alice on the other hand learns nothing. Hence, the oblivious transfer protocol can be used by one party to retrieve a secret value corresponding to its input without learning any of the other secret values as portrayed in [50], [75] and [43].

3.2.3 Garbled Circuit

A secure multiparty computation (MPC) protocol enables computing a function jointly and privately while protecting the input privacy of all parties involved in the

computation and providing the output to a set of prescribed parties only.

Garbled circuit (GC), introduced by Yao in [44], is a privacy-preserving technique to securely compute a function between two parties, called *garbler* and *evaluator*, where the function is represented as a boolean circuit. This can be described as a method of encoding the boolean circuit and the input given to this circuit. The boolean circuit here is the function that the parties compute. Any function can be represented in the form of boolean gates (such as AND, OR etc). On encoding these components by the garbler, an evaluator can evaluate the encoded boolean circuit using a special evaluation technique and obtain the output. The encoding relies on symmetric key cryptographic primitives like the AES encryption algorithm. The procedure guarantees that the evaluator only learns the output and the garbler does not learn anything [15]. The scheme can be defined as a tuple of algorithms given by (Garble, Evaluate) and can be elaborated as follows:

- Garble. Consider boolean circuit C taken as input. Garbler uses the Garble algorithm to produce the garbled circuit, $\hat{C}$. Along with the encoded circuit, the garbler also produces a set of labels $(x_{i,0}, x_{i,1})_{i\in[n]}$. The label, x, represents the value that corresponds to the input bit $b \in (0, 1)$.

- Evaluate. On receiving the input as encoded garbled circuit, $\hat{C}$ and labels, $(x_{i,0}, x_{i,1})_{i\in[n]}$, Evaluator uses the Evaluate algorithm to produce the value corresponding to the bit b for the $x \in (0, 1)^n$.

We explain how the garbled circuit technique works for the example circuit in Figure 3.5. Suppose Alice is the garbler and Bob is the evaluator. Alice and Bob decide on a function f they want to evaluate and a boolean circuit of it is created. Alice computes garbled tables for each logic gate in the circuit by running the garbling algorithm and creates a garbled circuit GC for the entire circuit. Alice then encodes her own input A as k_A^α or k_A^β and sends the encoded value as well as the garbled

circuit GC to Bob. The encoding scheme is only known to Alice. Bob obtains his key k_B that corresponds to his input B. For example: If Bob has input α, it will obtain the key corresponding to α i.e. k_B^α using 1-out-of-2 oblivious transfer protocol. For each key value, Bob performs one oblivious transfer protocol and obtains the corresponding key. Bob then uses this key to decrypt the encrypted values from the garbled table. The output can either be revealed to both parties or only 1 party. This is generally given by: $k_{out}^b \leftarrow \mathsf{GC}\{k_A^b, k_B^b, f\}$ where $b \in \{0,1\}$. In general, AND and OR gates have 6 unique keys. The NOT gate has 3 unique keys.

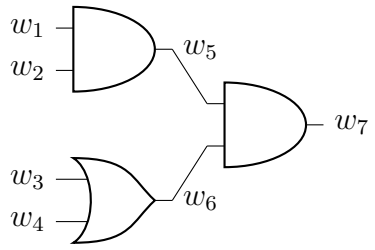

Figure 3.5: Simple Circuit with Two AND Gates and One OR Gate

Table 3.1: Garbled AND Gate 1

input wire (w_1)	input wire (w_2)	output wire (w_5)	Garbled Computation table
k_1^0	k_2^0	k_5^0	$E_{k_1^0}\ (E_{k_2^0}\ (k_5^0))$
k_1^0	k_2^1	k_5^0	$E_{k_1^0}\ (E_{k_2^1}\ (k_5^0))$
k_1^1	k_2^0	k_5^0	$E_{k_1^1}\ (E_{k_2^0}\ (k_5^0))$
k_1^1	k_2^1	k_5^1	$E_{k_1^1}\ (E_{k_2^1}\ (k_5^1))$

Table 3.2: Garbled OR Gate

input wire (w_3)	input wire (w_4)	output wire (w_6)	Garbled Computation table
k_3^0	k_4^0	k_6^0	$E_{k_3^0}\ (E_{k_4^0}\ (k_6^0))$
k_3^0	k_4^1	k_6^1	$E_{k_3^0}\ (E_{k_4^1}\ (k_6^1))$
k_3^1	k_4^0	k_6^1	$E_{k_3^1}\ (E_{k_4^0}\ (k_6^1))$
k_3^1	k_4^1	k_6^1	$E_{k_3^1}\ (E_{k_4^1}\ (k_6^1))$

From the above circuit diagram, we can see a circuit made up of 2 AND gates and 1 OR gate. To evaluate a simple boolean circuit given above Figure 3.5 where Alice has inputs (1,1) and Bob has inputs (1,0), Alice computes the garbled table for the

Table 3.3: Garbled AND Gate 2

input wire (w_5)	input wire (w_6)	output wire (w_7)	Garbled Computation table
k_5^0	k_6^0	k_7^0	$E_{k_5^0}\ (E_{k_6^0}\ (k_7^0))$
k_5^0	k_6^1	k_7^0	$E_{k_5^0}\ (E_{k_6^1}\ (k_7^0))$
k_5^1	k_6^0	k_7^0	$E_{k_5^1}\ (E_{k_6^0}\ (k_7^0))$
k_5^1	k_6^1	k_7^1	$E_{k_5^1}\ (E_{k_6^1}\ (k_7^1))$

boolean circuits (aka the garbled circuit) and holds the keys respective to its input. (In this case k_1^1 and k_3^1). Alice sends Bob the set of obfuscated keys that represent the garbled circuit. Bob obtains corresponding keys (In this case k_2^1 and k_4^0) using oblivious transfer. Bob decrypts all combinations using all the keys and obtains garbage values for all values except $D_{k_1^1}\ (D_{k_2^1}\ (k_5^1))$ (from Table 3.1) and $D_{k_3^1}\ (D_{k_4^0}\ (k_6^1))$ (from Table 3.2) because these values are encrypted using the respective keys. Bob evaluates each gate as so and obtains the final output, $D_{k_6^1}\ (D_{k_5^1}\ (k_7^1)) = k_7^1$ or 1 (from Table 3.3).

3.2.4 Fully Homomorphic Encryption

Before providing the definition of a fully homomorphic encryption (FHE) scheme, for completeness, we briefly mention the learning with errors problem that is the key to the security of an FHE scheme.

Learning with Errors. Learning with errors (LWE) is an important basis for lattice-based cryptography in the mathematical domain. In LWE, a small error is introduced in a system of linear equations consisting of random coefficients. Since the error term is very small, it is difficult to distinguish from the noise and hence solidifies the problem's difficulty in computation. An LWE-based cryptosystem's security relies on the hardness of solving the underlying LWE problem [37].

To define the LWE problem, consider four parameters, n which is a positive integer called the dimension parameter, m is the number of samples, q is a positive integer

referred to as modulus parameter and μ is the probability distribution over rational integers known as the error distribution mentioned in [37] and [11].

Definition 3.2.1. *For a secret $\overrightarrow{s} \in \mathbb{Z}_q^n$ the LWE distribution $A_{\overrightarrow{s},\mu}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\overrightarrow{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \leftarrow \mu$ gives :*

$$(\overrightarrow{a}, b = \langle \overrightarrow{s}, \overrightarrow{a} \rangle + e \mod q)$$

where $\langle \overrightarrow{s}, \overrightarrow{a} \rangle$ is the inner product of vectors $\overrightarrow{s}$ and $\overrightarrow{a}$ over $\mathbb{Z}_q^n$.

Ring Learning with Errors. Ring-learning with errors (RLWE) is a variant of the LWE problem which, instead, works over a polynomial ring and involves finding a small error term. This is also used in lattice-based cryptography because it is computationally hard to solve as explained in [11]. The RLWE variant is defined over the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, where n is a power of 2, q is integer modulus defining the quotient ring R_q, and the error distribution μ over R [37, 11].

Definition 3.2.2. *For a secret $s \in R_q$ the RLWE distribution $A_{s,\mu}$ over $R_q \times R_q$ is sampled by choosing $a \in R_q$ uniformly at random, choosing $e \leftarrow \mu$ gives:*

$$(a, b) = (a, s \cdot a + e)$$

where $b = s \cdot a + e$ is a polynomial, s, a and e are polynomials.

Fully Homomorphic Encryption. A FHE scheme allows ciphertext operations for both addition and multiplication operations. Well-known examples of FHE schemes are BGV [24], BFV [25], CKKS [33], and TFHE [34] Many of these FHE schemes also support single instruction multiple data (SIMD) operations which allow the encoding of multiple plaintexts into a single ciphertext upon encryption.

Hence, the number of operations is reduced which further leads to a decrease in the magnitude of the computation complexity. An FHE scheme consists of a set of four algorithms, namely key generation, encryption, decryption and evaluation, i.e., FHE = (KeyGen, Encrypt, Decrypt, Evaluate):

- KeyGen: It is used to generate a secret key (sk), a public key (pk) and an evaluation key (evk). The public key is used to encrypt a message whereas the secret key is used to decrypt a ciphertext. Given a security parameter κ, it generates the FHE keys $(sk, pk, ek) \leftarrow \mathsf{KeyGen}(1^{\kappa})$.

- Encrypt: On receiving an input message m and public key pk, the encryption algorithm, denoted by Encrypt, produces a ciphertext, $c = \mathsf{Encrypt}(pk, m)$.

- Decrypt: On receiving a ciphertext c and the secret key sk, the decryption algorithm outputs the original message $m = \mathsf{Decrypt}(sk, c)$.

- Evaluate: On receiving the public key pk, evaluation key evk, a set of ciphertexts $\{c_1, c_2, \cdots c_r\}$ and a function f, to be evaluated over the set of ciphertexts, the evaluation algorithm evaluates f over $\{c_1, c_2, \cdots c_r\}$ and produced a ciphertext $c \leftarrow \mathsf{Evaluate}(pk, evk, f, \{c_1, c_2, \cdots, c_r\})$.

In the following subsection, we give the concrete definition of the BFV scheme that we use in our protocol.

3.2.5 Brakerski-Fan-Vercauteren (BFV) Scheme

The BFV scheme is an FHE scheme that has the capability to perform homomorphic addition, homomorphic subtraction homomorphic multiplication, constant addition, subtraction and multiplication operations. The security of the BFV scheme is based on the RLWE problem [25, 4, 10]. Before providing the definition of the BFV, we need to define some notations:

- $\mathbb{Z}[x]$ contains all polynomials with coefficients in the integer ring.
- $f(x) \in \mathbb{Z}[x]$ is a monic irreducible polynomial of degree d.
- $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ is a quotient ring that contains all polynomials with coefficients in $\mathbb{Z}_q$ of degree up to $(n-1)$.

In the BFV scheme, a plaintext message is converted into a polynomial in plaintext space R_t for some integer $t > 1$ where $R_t = \mathbb{Z}_t[x]/(x^n + 1)$ and R is a quotient ring. Let $\Delta = \lceil q/t \rfloor$ and $r_t(q) = q \pmod t$ where q is the dividend, Δ is the quotient and $r_t(q)$ is the remainder.

- SecretKeyGen(sk): Sample $s \leftarrow \chi$ and output is produced as $sk = s$.
- PublicKeyGen(pk): Set $s = sk$; $a \leftarrow R_q$; $e \leftarrow \chi$ which produces output as $pk = ([-(a.s + e)]_q, a)$.
- Encrypt(pk, m): Set $p_0 = pk[0]$; $p_1 = pk[1]$; $m \in R_t$; $e_1, e_2, u \leftarrow \chi$ and it produces an output as $ct = ([p_0.u + e_1 + \Delta.m]_q, [p_1.u + e_2]_q)$,here ct is a pair of ciphertexts ($ct[0]$, $ct[1]$).
- Decrypt(sk, ct): A pair of ciphertexts $ct_0 = ct[0]$; $ct_1 = ct[1]$; A secret key, $s = sk$, are used to produce the output message $m = \left[\left\lfloor \frac{t.[c_0 + c_1.s]_q}{q} \right\rfloor\right]_t$.

Packing/SIMD Features of BFV. Traditional encryption techniques allow encryption of a single plaintext which is tedious when working with a large number of plaintext values but it is inefficient. The BFV scheme supports the SIMD packing to improve performance by packing multiple plaintext values into a single ciphertext slot efficiently. This improves the speed of computation by a significant amount. To create a ciphertext, multiple plaintext values are encoded into a vector of integers over $\mathbb{Z}_t$ for some t, and after that, the vector is encoded into a polynomial, and then the polynomial resulting from this computation is encrypted into a BFV ciphertext.

Using the SIMD technique, addition, subtraction, and multiplication operations can be performed on multiple plaintext values at one go in parallel.

Homomorphic Operations. We use the Microsoft SEAL for the BFV scheme [5]. In Table 3.4, we project the representations of the homomorphic operations we use in our protocol. Let us consider a BFV ciphertext $\mathsf{Enc}(x)$ for the plaintext x, and a BFV plaintext input y. We use $\mathsf{Eval}_{\otimes}$ to denote the constant multiplication such that $\mathsf{Eval}_{\otimes}(\mathsf{Enc}(x), y)$ produces the ciphertext $\mathsf{Enc}(x \times y)$. Similarly, we denote the constant addition by $\mathsf{Eval}_{\oplus}(\mathsf{Enc}(x), y)$ which produces the ciphertext $\mathsf{Enc}(x+y)$, and the constant subtraction by $\mathsf{Eval}_{\ominus}(\mathsf{Enc}(x), y)$ that produces the ciphertext $\mathsf{Enc}(x-y)$. The Microsoft SEAL API for these operations are given below.

- `evaluator.add_plain`: Performs addition operation between BFV ciphertext and BFV plaintext. This is represented as $\mathsf{Eval}_{\otimes}(\mathsf{Enc}(x), y)$.
- `evaluator.multiply_plain`: Performs multiplication operation between BFV ciphertext and BFV plaintext. This is represented as $\mathsf{Eval}_{\otimes}(\mathsf{Enc}(x), y)$.
- `evaluator.sub_plain`: Performs subtraction operation between BFV ciphertext and BFV plaintext. This is represented as $\mathsf{Eval}_{\ominus}(\mathsf{Enc}(x), y)$.

Table 3.4: Operations using Microsoft SEAL for BFV Scheme

Operation	Operations Representation	Constant Operations Representation	Encrypted Output
Multiplication	$\mathsf{Eval}_{\otimes}(\mathsf{Enc}(x), \mathsf{Enc}(y))$	$\mathsf{Eval}_{\otimes}(\mathsf{Enc}(x), y)$	$\mathsf{Enc}(x \times y)$
Addition	$\mathsf{Eval}_{\oplus}(\mathsf{Enc}(x), \mathsf{Enc}(y))$	$\mathsf{Eval}_{\oplus}(\mathsf{Enc}(x), y)$	$\mathsf{Enc}(x + y)$
Subtraction	$\mathsf{Eval}_{\ominus}(\mathsf{Enc}(x), \mathsf{Enc}(y))$	$\mathsf{Eval}_{\ominus}(\mathsf{Enc}(x), y)$	$\mathsf{Enc}(x - y)$

3.2.6 Secure Aggregation Protocol

Secure aggregation protocol is a useful privacy-preserving protocol that enables a group of mutually distrusting clients, each holding a private value, to collaboratively aggregate their values without revealing their individual private values. Suppose there are m clients where each client holds a secret x_i where $i \in \{1, 2, \ldots, m\}$ and a

server wishes to obtain the aggregate-sum value $\sum_{i=1}^{n} x_i$. A secure aggregation protocol allows the server to obtain the sum $\sum_{i=1}^{n} x_i$ without leaking any information about individual $x_i, 1 \leq i \leq m$. We denote a secure aggregation protocol for m clients $P = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ as $x_{sum} \leftarrow \Pi_{\mathsf{SAP}}(\mathcal{C}, \{x_i\}_{i \in \mathcal{C}})$ where $x_{sum} = \sum_{i=1}^{m} x_i$. In the application of ML, several protocols were developed recently [23], [62]. The desirable properties of a secure aggregation protocol are: (1) operates on high-dimensional vectors; (2) communication efficient; (3) robust to users dropping out; and (4) provides the strongest possible security.

Security. The security of Π_{SAP} requires that the server learns nothing other than what can be inferred from x_{sum}, and each client $\mathcal{C}$ learns nothing [22]. When the value of m is large, x_{sum} reveals nothing about individual x_i's [28].

3.3 Conclusion

To summarize this chapter, we explained the concepts in DNN training and cryptographic primitives and protocols needed to build our protocol. First, we describe the DNN training involving forward propagation, backward propagation and gradient descent. Then we presented cryptographic building blocks such as secret sharing, garbled circuits, homomorphic encryption and secure aggregation protocol, along with their security.

Chapter 4

System Model for Federated Learning

In this chapter, we discuss the system setup, the threat model, and the privacy goals we want to achieve for the federated learning training. We explain the challenges faced in ML in terms of privacy and how our work aims to design a protocol to address these challenges and also outline the major problems that we address in our work.

4.1 Privacy in ML

Privacy in any system is measured by the adversarial challenges designed against it. Various adversarial models attack either the privacy of the model or the privacy of the data involved. Data privacy refers to the protection of confidential, sensitive, and personal information from unauthorized access, use, or disclosure. In the context of machine learning, data privacy is essential to prevent the misuse of sensitive data and protect individuals' privacy rights. Model privacy refers to the protection of the model's parameters or architecture from being exposed or manipulated by attackers. Some examples of these attacks are mentioned in Section 2.4. An ML

model is an intellectual property, usually held by the server, and input data such as medical records is strictly private and should be protected. Since adversarial methods tend to either passively or actively manipulate this information [36], it is of utmost importance to protect the data of either party involved. Some general attack types are "population of members" inference, "training dataset members" inference, "model parameters" inference, etc. Although there is an enormous amount of research done to protect models for inference services such as [49], [52], [65], [30], [97], [82], [79], [56], designing an efficient and scalable training protocol is still a challenging problem. FL tries to provide user data privacy in the training process but fails to provide model privacy which can be misused by an internal user in the system or an external adversary. This strongly motivates us to look into cryptographic primitives to design strongly secure and efficient protocols that provide both data privacy and model privacy.

4.2 System Model

We consider the standard FL setting, as shown in Figure 4.1, which is the same as in [61]. In our system, we have two distinct entities: a set of clients $P = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_m\}$ willing to contribute their private data to train an ML model, and a server, denoted by $\mathcal{S}$, holds the model to be trained on the clients' dataset. We denote the dataset of the client $\mathcal{C}_i$ by $\mathcal{D}_i$, for $1 \leq i \leq m$. Without loss of generality, we assume that the server wishes to train an ℓ-layer DNN $\boldsymbol{\theta} = \{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \ldots, (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\}$. Our goal is to design an efficient and secure protocol to train an ℓ DNN model in the federated training setting while assuring both the model $\boldsymbol{\theta}$ and data $\mathcal{D}_i$ privacy.

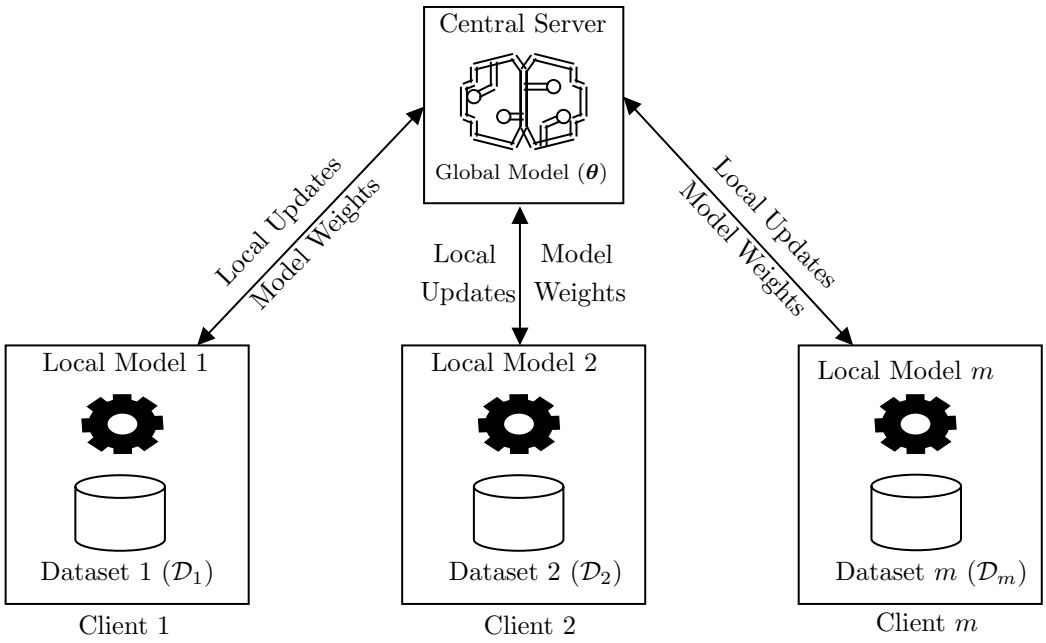

Figure 4.1: A High-level Overview of a Federated Learning Training System

4.3 Problem Statement

We aim to address data privacy and model privacy in the FL setting. Although FL provides privacy in the form of securing the input data of the clients involved, the model is exposed to all participants in the training process, but in our work, the server and the clients train the model in FL, without revealing the model to the clients in cleartext. We consider two fundamental problems: (1) secure local gradient computation by a client and the server; and (2) design an efficient privacy-preserving training in FL.

- **Problem 1: Secure two-party Gradient Computation.** Suppose the server ($\mathcal{S}$) holds a DNN model $\boldsymbol{\theta} = \Big\{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \cdots, (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\Big\}$ and a client ($\mathcal{C}$) holds a data point $(\mathbf{x}, \mathbf{y})$. The task for a secure 2-party protocol between the client and the server is to jointly compute the gradient on $(\mathbf{x}, \mathbf{y})$ for $\boldsymbol{\theta}$, i.e., $\nabla\boldsymbol{\theta}(\mathbf{x}, \mathbf{y}) = \Big\{(\nabla\mathbf{W}^0, \nabla\mathbf{b}^0), (\nabla\mathbf{W}^1, \nabla\mathbf{b}^1), \cdots, (\nabla\mathbf{W}^{\ell-1}, \nabla\mathbf{b}^{\ell-1})\Big\}$ in such a way that the client's data point is not revealed to the server and the server's model is not revealed to the client. At the end of the protocol, the

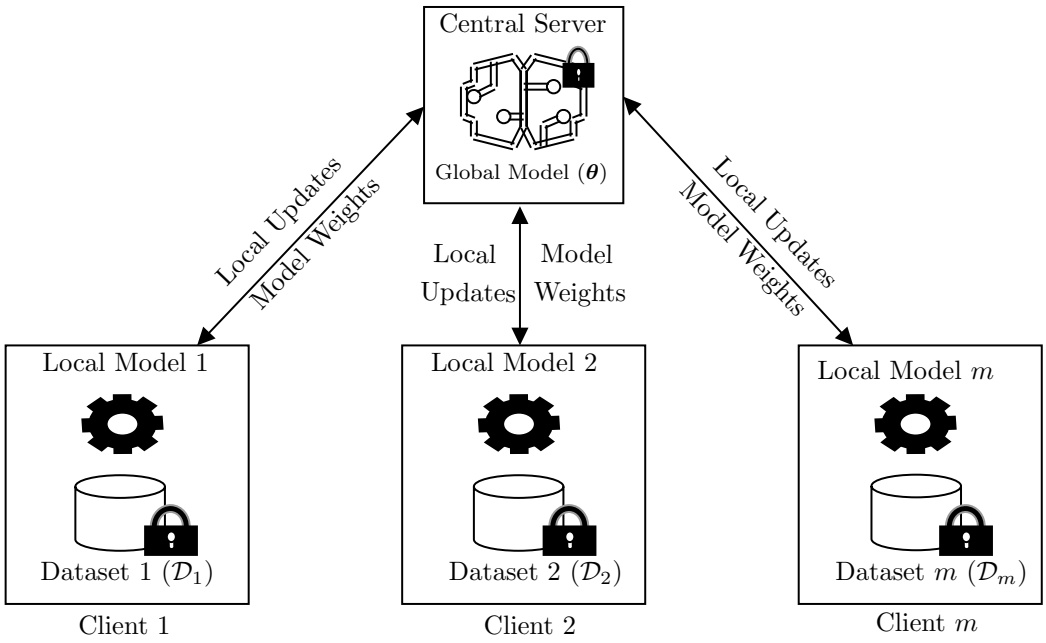

Figure 4.2: Privacy-preserving Federated Learning

server and the client hold an additive share of $\nabla\boldsymbol{\theta}(\mathbf{x}, \mathbf{y})$, i.e., the server holds $\nabla^{\mathcal{S}}$ and the client holds $\nabla^{\mathcal{C}}$ such that $\nabla^{\mathcal{S}} + \nabla^{\mathcal{C}} = \nabla\boldsymbol{\theta}(\mathbf{x}, \mathbf{y})$.

- **Problem 2: Privacy-preserving Federated Learning.** The task is to enable server training a DNN model $\boldsymbol{\theta} = \left\{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \cdots, (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\right\}$ over the combined dataset $\mathcal{D} \leftarrow \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_m$, i.e., $\boldsymbol{\theta} \leftarrow \mathsf{DNNTraining}(\mathcal{D}, m)$ while ensuring clients' dataset privacy and the server's model privacy.

System Goals. We require the system to satisfy the following goals:

- *Correctness:* The correct input from the clients should produce the correct model for the server in the protocol. However, in case the client tampers with or manipulates the input, the protocol will not guarantee the correctness of the model.

- *Privacy:* Our protocol aims to protect the privacy of both the input from the client and the model from the server. During the interaction, the server will not be able to learn about any information from the client's input $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_i$,

and the client will not be able to learn about the model held by the server ($\boldsymbol{\theta} = \Big\{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \ldots, (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\Big\}$, other than what it is expected to learn according to the protocol.

- *Efficiency:* The computation between the clients and the server should be efficient and optimized to minimize the computation and communication cost, without violating privacy goals. As the server holds the model and coordinates the training process, it is expected that the server's computational and communication overhead will be higher than the individual clients.

4.4 Threat Model

In our system, we consider semi-honest adversaries who may try to learn any unintended information about the model $\boldsymbol{\theta} = \Big\{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \ldots, (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\Big\}$ or a client's input $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ while following all the instructions of the protocol. We consider three different adversarial scenarios: (1) only clients being corrupted by the adversary; (2) only the server being corrupted by the adversary; and (3) both the server and clients being corrupted by the adversary.

The intention of the adversary is to learn about the information of the honest users' private data through the locally trained model using adversarial ML techniques. Moreover, the adversary's behavior is modeled as a semi-honest adversary, which is different from an active adversary but is similar to the threat models considered in previous works [52], [65], [97]. If the client tampers with or poisons the data, our protocols do not guarantee any correctness.

4.5 Conclusion

In this chapter, we described the system setting, and discuss the privacy goals and the threat model for securing the federated learning training. Our privacy goals include protecting the privacy of the data held by the clients and the model held by the server. In FL, there is a certain level of privacy for the input data but none for the model. We aim to achieve this in our protocol such that, given the setting in Figure 4.1, the dataset $\mathcal{D}_i$ is only held by the client $\mathcal{C}_i$, and the global model $\boldsymbol{\theta}$ is only accessible to the server. In the training protocol, the computation must be efficient and correct where the final global model adheres to the input data and no leakage of information occurs during the client's and server's interaction such that access to the input data is available only to the client and access to the model is only available to the server as shown in Figure 4.2.

Chapter 5

Efficient Secure Two-party Gradient Computation Protocol

In this chapter, we present our secure gradient computation protocol between a server and a client. The design of our protocol is based on homomorphic encryption, the garbled circuit and additive secret sharing. We introduce a technique to securely compute the matrix-vector multiplication in the linear layer efficiently and build an optimized garbled circuit to compute the non-linear layer. Finally, we put everything together to build an efficient secure two-party gradient computation protocol.

5.1 Revisiting Deep Neural Networks

While revisiting the operations involved in implementing the neural network algorithm, it can be observed that a typical neural network training consists of the forward propagation phase, the backward propagation phase and the gradient computation. Both the forward and backward propagation phases consist of a linear layer and a non-linear layer. Optimizing these fundamental phases makes the computation more efficient.

Description of DNN. Our optimization idea for the DNN is explained with an example. Let us consider a neural network with two hidden layers using the activation functions namely sigmoid and softmax. The DNN model parameters are given by $\boldsymbol{\theta} = \Big\{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), (\mathbf{W}^2, \mathbf{b}^2)\Big\}$. As the sigmoid and softmax computations involve the exponentiation computation that is not FHE and GC friendly, an approximation of these functions (e.g., $\sigma_{\mathsf{apx}}, \sigma'_{\mathsf{apx}}$) is used. Figure 5.1 shows a high-level overview of gradient computation on input $(\mathbf{x}, \mathbf{y})$. Below, the forward and backward propagation computations are described.

Forward Propagation. The output of a linear layer computation is denoted by $\mathbf{c}_i$ and the non-linear layer computation by $\mathbf{a}_i$. The linear layer consists of learning the linear relationship between the input $\mathbf{x}$ (in Figure 5.1 $\mathbf{a}_0 = \mathbf{x}$) and the model parameters, where $\mathbf{W}^i$ is the weight matrix and $\mathbf{b}^i$ is the bias vector. Similarly, the non-linear brings non-linearity by applying an activation function (such as ReLU or sigmoid) on the output of the previous layer. The linear layer computation is given by:

$$\mathbf{c}_i = \mathbf{W}^i \cdot \mathbf{a}_i + \mathbf{b}^i, 0 \leq i \leq 2$$

and the non-linear layer is computed as:

$$\mathbf{a}_{i+1} = \sigma_{\mathsf{apx}}(\mathbf{c}_i), 0 \leq i \leq 2$$

where σ_{apx} is the approximated activation function which is explained in Section 3.1.3, and in the last layer is the softmax function. At the final layer, the output is $\hat{\mathbf{y}} = \mathbf{a}_3$.

Backward Propagation. For the backward propagation, the linear layer and non-linear layer results are denoted as $\boldsymbol{\gamma}_i$ and $\boldsymbol{\alpha}_i$, respectively. The linear layer $\boldsymbol{\gamma}_i$ in the

backward pass is computed by performing the matrix-vector multiplication between the transpose of the weight matrix $(\mathbf{W}^i)^T$ and the loss $\boldsymbol{\alpha}_i$. The non-linear layer is computed by first computing the derivative of the activation function, σ', on the linear layer output $\mathbf{c}_i$ from the forward propagation and then computing the Hadamard product with the output $\boldsymbol{\gamma}_i$ of the linear layer in the backward propagation. The linear and non-linear layers computation is summarized as follows.

$$\boldsymbol{\gamma}_i = (\mathbf{W}^i)^T \boldsymbol{\alpha}_i, 0 \leq i \leq 2,$$

where $(\mathbf{W}^i)^T$ is the transpose of the weight matrix $\mathbf{W}^i$ and loss $\boldsymbol{\alpha}_l = \hat{\mathbf{y}} - \mathbf{y}$. The non-linear layer computation is:

$$\boldsymbol{\alpha}_{i-1} = \sigma'_{\mathsf{apx}}(\mathbf{c}_{i-1}) \odot \boldsymbol{\gamma}_i$$

where σ'_{apx} is the derivative of σ and $\odot$ is the component-wise multiplication known as Hadamard product of 2 vectors.

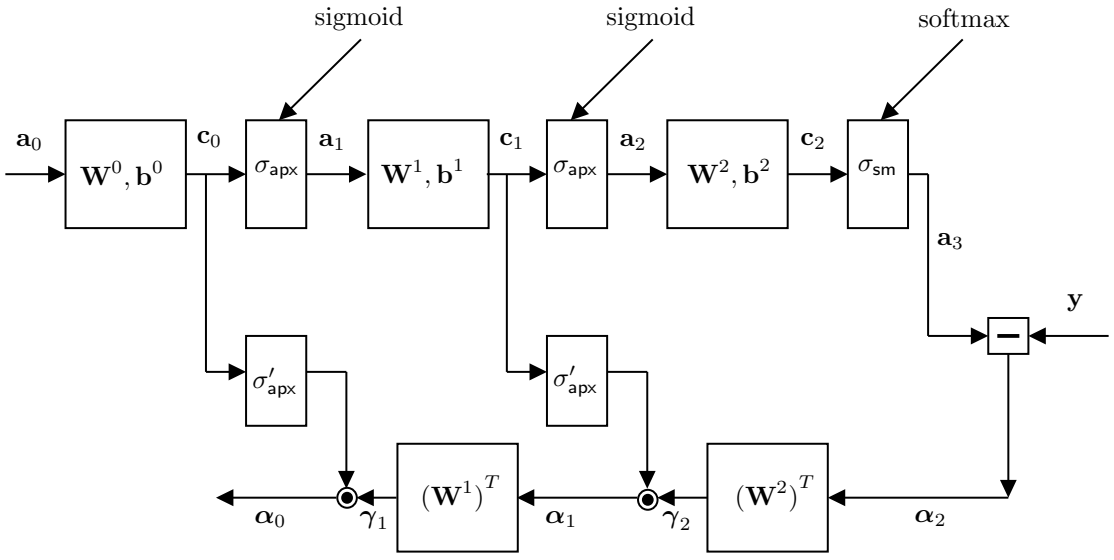

Figure 5.1: Example of Two-hidden Layer Neural Network without Privacy

Compute Gradient. The gradient for a weight matrix $\nabla \mathbf{W}^i$ and bias vector $\nabla \mathbf{b}^i$, $0 \leq i \leq 2$, is computed using the output of the non-linear layer from the forward propagation, $\mathbf{a}$, and backward propagation, $\boldsymbol{\alpha}$. The gradient for the weight matrix and bias vector is computed as:

$$\nabla \mathbf{W}^i = \boldsymbol{\alpha}_i \cdot \mathbf{a}_i^T \tag{5.1}$$

$$\nabla \mathbf{b}^i = \boldsymbol{\alpha}_i, 0 \leq i \leq 2. \tag{5.2}$$

Figure 5.2 summarizes the gradient computation of $\boldsymbol{\theta}$ on $(\mathbf{x}, \mathbf{y})$.

Example 5.1.1. Let $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$ be a vector of length 3 and $\mathbf{a} = (\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$ be another vector of length 4, the resultant matrix $\boldsymbol{\alpha} \cdot \mathbf{a}^T$ is as follows:

$$\boldsymbol{\alpha} \cdot \mathbf{a}^T = \begin{bmatrix} \boldsymbol{\alpha}_0 \\ \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{\alpha}_0 \cdot \mathbf{a}_0 & \boldsymbol{\alpha}_0 \cdot \mathbf{a}_1 & \boldsymbol{\alpha}_0 \cdot \mathbf{a}_2 & \boldsymbol{\alpha}_0 \cdot \mathbf{a}_3 \\ \boldsymbol{\alpha}_1 \cdot \mathbf{a}_0 & \boldsymbol{\alpha}_1 \cdot \mathbf{a}_1 & \boldsymbol{\alpha}_1 \cdot \mathbf{a}_2 & \boldsymbol{\alpha}_1 \cdot \mathbf{a}_3 \\ \boldsymbol{\alpha}_2 \cdot \mathbf{a}_0 & \boldsymbol{\alpha}_2 \cdot \mathbf{a}_1 & \boldsymbol{\alpha}_2 \cdot \mathbf{a}_2 & \boldsymbol{\alpha}_2 \cdot \mathbf{a}_3 \end{bmatrix}.$$

Our Observation. Note that in the approximated sigmoid function computation the key operations are comparisons and additions where the comparison operation is an expensive operation for FHE and GC. We optimize the operation for the approximated sigmoid and its derivative $(\sigma_{\mathsf{apx}}, \sigma'_{\mathsf{apx}})$ computation. We make the following two observations.

Observation 1. *We compute the approximated sigmoid and its derivative $(\sigma_{\mathsf{apx}}, \sigma'_{\mathsf{apx}})$ efficiently by reducing one greater than and one less than operation for each activation*

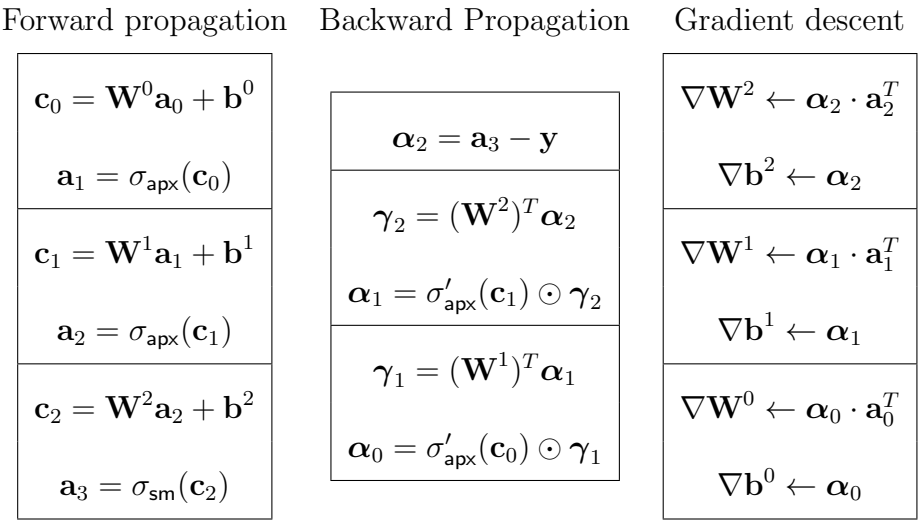

Figure 5.2: Summary of Two-hidden Layer Neural Network Training

layer except the last activation (softmax) layer. We use this functionality as a single circuit.

Observation 2. *We notice that packing the input vector using one extra space with the first element of the input vector allows us to compute the inner product efficiently with fewer ciphertexts, as explained in Section 5.2.1.*

5.2 Design Rationale and Techniques Behind Our Protocol

Training a neural network is a computationally heavy task, additionally securing this operation adds overhead to it. The gradient computation is a sub-task. Neural network computations in the encrypted domain involve constant multiplication, addition, and permutation operations in the linear layer of which the task of constant multiplication creates the largest overhead. Hence, to efficiently perform these operations, our first task is to design a technique to perform the linear layer computation securely and efficiently, explained in Section 5.2.1. Next, non-linear layer

computation is performed efficiently using the observations in the previous section. In previous work such as [52, 65], the authors considered the problem of secure inference services which involves only the forward propagation and built a protocol using the GC and an FHE scheme, but since we consider the task of neural network training, we need to design a secure gradient computation protocol involving both the forward and backward propagation phases. This is done by using FHE and GC as used in [52, 65], but we design an optimized protocol for the gradient computation. The BFV FHE scheme is chosen over other HE schemes because the BFV scheme allows efficient arithmetic operations by applying the SIMD packing technique (explained in Section 3.2.5). The work in [82] provides privacy to both the model and the input in the federated learning setting using only an FHE scheme but we plan to use a hybrid technique of FHE and GC for better efficiency and accuracy.

5.2.1 Our Secure Matrix-Vector Computation Technique

A major problem considered here is securely computing the matrix-vector multiplication ($\mathbf{A} \cdot \mathbf{x}$) where the server holds a private matrix ($\mathbf{A}$) and the client holds a private vector ($\mathbf{x}$) and they wish to jointly perform the matrix-vector multiplication ($\mathbf{y} = \mathbf{A} \cdot \mathbf{x}$) without revealing their private inputs to each other and obtain an additive share of the result ($\mathbf{y}$) of the matrix-vector multiplication. To efficiently compute the linear layers, we combine the ideas from work in [51, 97] and build a new technique with the reduced computational overhead of constant multiplications and permutations in the encrypted domain. In our technique, the client encrypts her private input vector, $\mathsf{Enc}(x)$ using an FHE scheme that supports packing (e.g., BFV) and sends it to the server, and the server applies an efficient packing technique and computes constant multiplication operation using FHE (i.e., $\mathsf{Eval}_{\otimes}(\mathsf{Enc}(\mathbf{x}), \mathbf{A})$). Our new technique allows us to cut down on the number of constant multiplications and rotations. The idea and the details behind this computation are explained below.

Idea. From the work of [52], [97], arranging the matrices in a particular order allows for a simpler computation of the matrix-vector multiplication in the encrypted domain. This idea reduces the number of constant multiplications, permutations and the number of ciphertexts needed to compute the result. The better the arrangement of the matrix, the lesser computation is required to compute the result. We propose a new packing technique to perform $\mathsf{Eval}_{\otimes}(\mathsf{Enc}(\mathbf{x}), \mathbf{A})$ operation efficiently. It is explained in the following example.

Example 5.2.1. Suppose the server holds the following matrix $\mathbf{A}$ and the client holds an input $\mathbf{x}$. Assume in an FHE ciphertext there are 12 slots (the BFV has this feature). The client generates the keys (sk, pk, evk) for the FHE scheme using the key generation algorithm and gives pk and evk to the server. The task of securely computing the matrix-vector multiplication is to compute $\mathbf{y}^{\mathcal{S}} + \mathbf{y}^{\mathcal{C}} = \mathbf{y} = \mathbf{A} \cdot \mathbf{x}$.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}.$$

1. **Client:** The vector $\mathbf{x}$ is encrypted so that the SIMD operations can be leveraged. The client creates a plaintext for $\mathbf{x}$ as follows.

$$\mathsf{pt}_x = \left| \begin{array}{cccccccccccc} x_1 & x_2 & x_3 & x_4 & x_1 & x_1 & x_2 & x_3 & x_4 & x_1 & 0 & 0 \end{array} \right|$$

   Then the client creates a ciphertext $\mathsf{ct}_x = \mathsf{Enc}(\mathsf{pt}_x)$ and sends ct_x to the server.

2. **Server:** First, a hybrid transformation is applied on $\mathbf{A}$, as used in [52]

$$\mathbf{A}_1 = \begin{bmatrix} a_{1,1} & a_{2,2} & a_{1,3} & a_{2,4} \\ a_{2,1} & a_{1,2} & a_{2,3} & a_{1,4} \\ a_{3,1} & a_{4,2} & a_{3,3} & a_{4,4} \\ a_{4,1} & a_{3,2} & a_{4,3} & a_{3,4} \end{bmatrix}.$$

Our new packing technique for the matrix $\mathbf{A}_1$ is performed as follows:

pt_1	$a_{1,1}$	$a_{2,2}$	$a_{1,3}$	$a_{2,4}$	$a_{1,1}$	$a_{3,1}$	$a_{4,2}$	$a_{3,3}$	$a_{4,4}$	$a_{3,1}$	0	0
pt_2	$a_{2,1}$	$a_{1,2}$	$a_{2,3}$	$a_{1,4}$	$a_{2,1}$	$a_{4,1}$	$a_{3,2}$	$a_{4,3}$	$a_{3,4}$	$a_{4,1}$	0	0

Note that the even indexed rows are placed in one plaintext and odd indexed ciphertexts are placed in a different ciphertext. To encode the entire matrix, two plaintexts are required. The server performs the encrypted constant multiplication as follows:

$\mathsf{ct}_{x1} = \mathsf{Eval}_{\otimes}(\mathsf{ct}_x, \mathsf{pt}_1) =$

$$\begin{bmatrix} a_{1,1} \cdot x_1 & a_{2,2} \cdot x_2 & a_{1,3} \cdot x_3 & a_{2,4} \cdot x_4 & a_{1,1} \cdot x_1 & a_{3,1} \cdot x_1 & a_{3,2} \cdot x_2 & a_{3,3} \cdot x_3 \\ a_{4,4} \cdot x_4 & a_{3,1} \cdot x_1 & 0 & 0 & & & & \end{bmatrix}$$

$\mathsf{ct}_{x2} = \mathsf{Eval}_{\otimes}(\mathsf{ct}_x, \mathsf{pt}_2) =$

$$\begin{bmatrix} a_{2,1} \cdot x_1 & a_{1,2} \cdot x_2 & a_{2,3} \cdot x_3 & a_{1,4} \cdot x_4 & a_{2,1} \cdot x_1 & a_{4,1} \cdot x_1 & a_{3,2} \cdot x_2 & a_{4,3} \cdot x_3 \\ a_{3,4} \cdot x_4 & a_{4,1} \cdot x_1 & 0 & 0 & & & & \end{bmatrix}$$

The server randomly generates a plaintext

$$\mathsf{pt}_r = \begin{bmatrix} r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 & r_8 & r_9 & r_{10} & 0 & 0 \end{bmatrix}$$

The server then performs the following homomorphic operations:

$\mathsf{ct} = \mathsf{Eval}_{\ominus}(\mathsf{Eval}_{\oplus}(\mathsf{ct}_{x1}, \mathsf{ROT}(\mathsf{ct}_{x2}, 1)), \mathsf{pt}_r) =$

$$\begin{bmatrix} a_{1,1}x_1 + a_{1,2}x_2 - r_1 & a_{2,2}x_2 + a_{2,3}x_3 - r_2 & a_{1,3}x_3 + a_{1,4}x_4 - r_3 & a_{2,4}x_4 + a_{2,1}x_1 - r_4 \\ a_{1,1}x_1 + a_{4,1}x_2 - r_5 & a_{3,1}x_1 + a_{3,2}x_2 - r_6 & a_{4,2}x_2 + a_{4,3}x_3 - r_7 & a_{3,3}x_3 + a_{3,4}x_4 - r_8 \\ a_{4,4}x_4 + a_{3,4}x_4 - r_9 & a_{3,1}x_1 + a_{4,1}x_1 - r_{10} & 0 & 0 \end{bmatrix}$$

where $\mathsf{ROT}(\cdot, \cdot)$ is the homomorphic rotation operation. The server sends ct to the client.

3. **Client:** The client decrypts ct using the secret key sk and computes:

$$b_1 = a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 - r_1 - r_3$$

$$b_2 = a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 + a_{2,1}x_1 - r_2 - r_4$$

$$b_3 = a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 - r_6 - r_8$$

$$b_4 = a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 + a_{3,4}x_4 - r_7 - r_9$$

The share for the server is given by $\mathbf{y}^{\mathcal{S}} = (R_1, R_2, R_3, R_4) = (r_1 + r_3, r_2 + r_4, r_6 + r_8, r_7 + r_9)$. The share for the client is given by $\mathbf{y}^{\mathcal{C}} = (b_1, b_2, b_3, b_4)$.

The above technique for securely computing a matrix-vector multiplication can be applied to any dimension matrix and vector. We emphasize that in our secure gradient computation protocol the results of the linear layer computations are split between the client and the server, as shown in the above example. The secure matrix-vector multiplication in the backward propagation is also computed in the same way.

5.2.2 Our Optimized Non-linear Layer Computation

The idea for an efficient and secure evaluation of a non-linear layer from two shares of a linear layer is to apply the garbled circuit technique [96] and the result of the non-linear layer is again split into two shares, one is for the server and another is for

Input : Client inputs $(\mathbf{c}^{\mathcal{C}}, \mathbf{a}^{\mathcal{C}}, \boldsymbol{\beta}^{\mathcal{C}})$. Server inputs $(\mathbf{c}^{\mathcal{S}})$
Output: Server receives $(\mathbf{a}^{\mathcal{S}}, \boldsymbol{\beta}^{\mathcal{S}})$

1. Compute $\mathbf{c} = \mathbf{c}^{\mathcal{S}} + \mathbf{c}^{\mathcal{C}}$
2. Compute $\mathbf{a}^{\mathcal{S}} = \sigma_{\mathsf{apx}}(\mathbf{c}) - \mathbf{a}^{\mathcal{C}}$ and $\boldsymbol{\beta}^{\mathcal{S}} = \sigma'_{\mathsf{apx}}(\mathbf{c}) - \boldsymbol{\beta}^{\mathcal{C}}$.
3. Denote the circuit for the above two steps by $\mathcal{SCKT}$.
4. Client and server run the garbled circuit on $\mathcal{SCKT}$ with client's input $(\mathbf{c}^{\mathcal{C}}, \mathbf{a}^{\mathcal{C}}, \boldsymbol{\beta}^{\mathcal{C}})$ and server's input $(\mathbf{c}^{\mathcal{S}})$ and the server gets output $(\mathbf{a}^{\mathcal{S}}, \boldsymbol{\beta}^{\mathcal{S}})$, i.e., $(\mathbf{a}^{\mathcal{S}}, \boldsymbol{\beta}^{\mathcal{S}}) \leftarrow \mathsf{GC}(\mathcal{SCKT}, (\mathbf{c}^{\mathcal{C}}, \mathbf{a}^{\mathcal{C}}, \boldsymbol{\beta}^{\mathcal{C}}), (\mathbf{c}^{\mathcal{S}}))$.

Figure 5.3: Secure Non-linear Layer Computation using The Garbled Circuit

the client. In Figure 5.1, we observe that both the sigmoid and sigmoid derivative activation functions can be combined into a single functionality, instead of creating two different functions (**Observation 1**). Hence, a single function is created for the approximated sigmoid and its derivative in Eqs. (3.1) and (3.2) used in the forward and backward propagation, respectively, where the greater than and less than operations for a value are performed only once, instead of twice. This improves the efficiency of the non-linear computation by having a reduced-size circuit for the non-linear layer computation.

Non-linear Activation Layer Functionality Details for Garbled Circuit Evaluation. Note that the input to the non-linear layers is from the output of the linear layer, denoted by $\mathbf{c}$. Suppose after the secure linear layer computation the server ($\mathcal{S}$) holds the share $\mathbf{c}^{\mathcal{S}}$ and the client ($\mathcal{C}$) holds the share $\mathbf{c}^{\mathcal{C}}$ such that $\mathbf{c}^{\mathcal{S}} + \mathbf{c}^{\mathcal{C}} = \mathbf{c}$. To evaluate the sigmoid computation and its derivative on $\mathbf{c}$, i.e., $\sigma_{\mathsf{apx}}(\mathbf{c})$ and $\sigma'_{\mathsf{apx}}(\mathbf{c})$, $\mathbf{c} = \mathbf{c}^{\mathcal{S}} + \mathbf{c}^{\mathcal{C}}$ and then $\mathbf{a} = \sigma_{\mathsf{apx}}(\mathbf{c}) = \sigma_{\mathsf{apx}}(\mathbf{c}^{\mathcal{S}} + \mathbf{c}^{\mathcal{C}})$ and $\boldsymbol{\beta} = \sigma'_{\mathsf{apx}}(\mathbf{c}) = \sigma'_{\mathsf{apx}}(\mathbf{c}^{\mathcal{S}} + \mathbf{c}^{\mathcal{C}})$ need to be computed first. After that, the results of $\mathbf{a} = \sigma_{\mathsf{apx}}(\mathbf{c})$ for the forward propagation and $\boldsymbol{\beta} = \sigma'_{\mathsf{apx}}(\mathbf{c})$ for the backward propagation need to be split into two parts such that $\mathbf{a}^{\mathcal{S}} + \mathbf{a}^{\mathcal{C}} = \sigma_{\mathsf{apx}}(\mathbf{c}^{\mathcal{S}} + \mathbf{c}^{\mathcal{C}})$ and $\boldsymbol{\beta}^{\mathcal{S}} + \boldsymbol{\beta}^{\mathcal{C}} = \sigma'_{\mathsf{apx}}(\mathbf{c}^{\mathcal{S}} + \mathbf{c}^{\mathcal{C}})$. A high-level description of the process is summarized in Figure 5.3 and Figure 5.4.

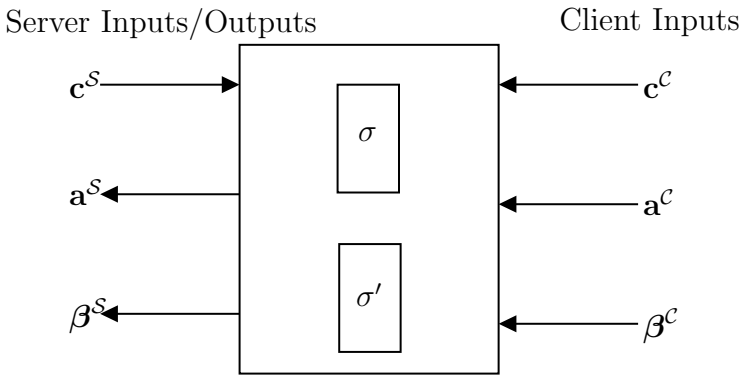

Figure 5.4: Optimized Sigmoid and Sigmoid Derivative

Input: Client inputs $(\boldsymbol{\beta}^{\mathcal{C}}, \boldsymbol{\gamma}^{\mathcal{C}}, \boldsymbol{\alpha}^{\mathcal{C}})$ and Server inputs $(\boldsymbol{\beta}^{\mathcal{S}}, \boldsymbol{\gamma}^{\mathcal{S}})$
Output: Server receives output $\boldsymbol{\alpha}^{\mathcal{S}}$

1. Compute $\boldsymbol{\beta} = \boldsymbol{\beta}^{\mathcal{C}} + \boldsymbol{\beta}^{\mathcal{S}}$ and $\boldsymbol{\gamma} = \boldsymbol{\gamma}^{\mathcal{C}} + \boldsymbol{\gamma}^{\mathcal{S}}$.
2. Compute $\boldsymbol{\alpha} = (\boldsymbol{\beta}^{\mathcal{C}} + \boldsymbol{\beta}^{\mathcal{S}}) \odot (\boldsymbol{\gamma}^{\mathcal{C}} + \boldsymbol{\gamma}^{\mathcal{S}})$
3. Compute $\boldsymbol{\alpha}^{\mathcal{S}} = \boldsymbol{\alpha} - \boldsymbol{\alpha}^{\mathcal{C}}$
4. Denote the circuit for the above three steps by $\mathcal{HCKT}$.
5. Client and server run the garbled circuit on $\mathcal{HCKT}$ with client's input $(\boldsymbol{\beta}^{\mathcal{C}}, \boldsymbol{\gamma}^{\mathcal{C}}, \boldsymbol{\alpha}^{\mathcal{C}})$ and server's input $(\boldsymbol{\beta}^{\mathcal{S}}, \boldsymbol{\gamma}^{\mathcal{S}})$ and the server gets output $\boldsymbol{\alpha}^{\mathcal{S}}$, i.e., $\boldsymbol{\alpha}^{\mathcal{S}} \leftarrow \mathsf{GC}(\mathcal{HCKT}, (\boldsymbol{\beta}^{\mathcal{C}}, \boldsymbol{\gamma}^{\mathcal{C}}, \boldsymbol{\alpha}^{\mathcal{C}}), (\boldsymbol{\beta}^{\mathcal{S}}, \boldsymbol{\gamma}^{\mathcal{S}}))$.

Figure 5.5: Hadamard Product Circuit

Hadamard Product Garbled Circuit Computation. In addition to the activation function computation (e.g., σ_{apx}, σ'_{apx}), the Hadamard product on $\boldsymbol{\beta}$ and the output $\boldsymbol{\alpha}$ of the linear layer in the backward propagation needs to be computed. This operation is perfomed securely on two shares of $\boldsymbol{\beta}$ and the linear layer output $\boldsymbol{\gamma}$ using the garbled circuit. Suppose $\boldsymbol{\beta}^{\mathcal{C}}$ and $\boldsymbol{\beta}^{\mathcal{S}}$ are the shares of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}^{\mathcal{C}}$ and $\boldsymbol{\gamma}^{\mathcal{S}}$ are the shares of $\boldsymbol{\gamma}$, respectively. Then, the Hadamard product on $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ is computed as $\boldsymbol{\alpha} = \boldsymbol{\beta} \odot \boldsymbol{\gamma} = (\boldsymbol{\beta}^{\mathcal{C}} + \boldsymbol{\beta}^{\mathcal{S}}) \odot (\boldsymbol{\gamma}^{\mathcal{C}} + \boldsymbol{\gamma}^{\mathcal{S}})$, and then create two shares of $\boldsymbol{\alpha}$ as $\boldsymbol{\alpha}^{\mathcal{S}} = \boldsymbol{\alpha} - \boldsymbol{\alpha}^{\mathcal{C}}$. A high-level description of this process is summarized in Figure 5.5 and Figure 5.6.

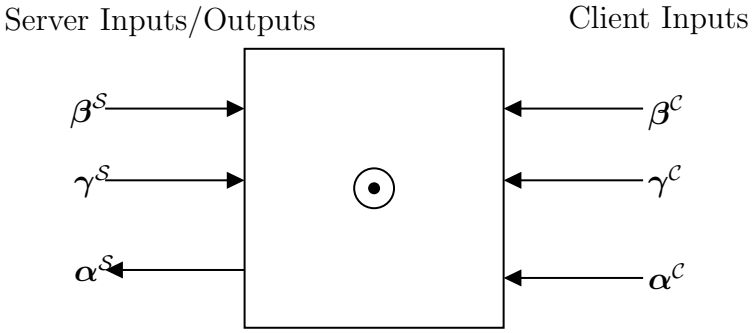

Figure 5.6: Optimized Hadamard Computation

In the above, we have explained all elementary components needed to securely compute the linear layer and the non-linear layer. These secure building blocks are used for each linear and non-linear layer computation, except for the softmax computation. In the next section, a description of the entire protocol for securely computing the gradient for input is provided.

5.3 Our Proposed Two-party Secure Local Gradient Computation

High-level Overview. The gradient computation protocol is executed between the server ($\mathcal{S}$) and a client ($\mathcal{C}$) where the client holds an input $\mathbf{x} \in \mathcal{D}$ and the server holds a model $\boldsymbol{\theta} = \left\{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \ldots, (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\right\}$ for an ℓ layered neural network. For the sake of simplicity, we assume that the DNN is of two layers, i.e., $\ell = w$. To securely compute the gradient, we apply the techniques described in Section 5.2.1: the secure matrix-vector multiplication is performed using the FHE and the additive secret sharing and the non-linear layer computation is performed using the GC and the additive secret sharing techniques. Figure 5.8 provides a summary of it. Figure 5.7 shows how the additive secret sharing is performed so that no leakage at an intermediate stage can happen.

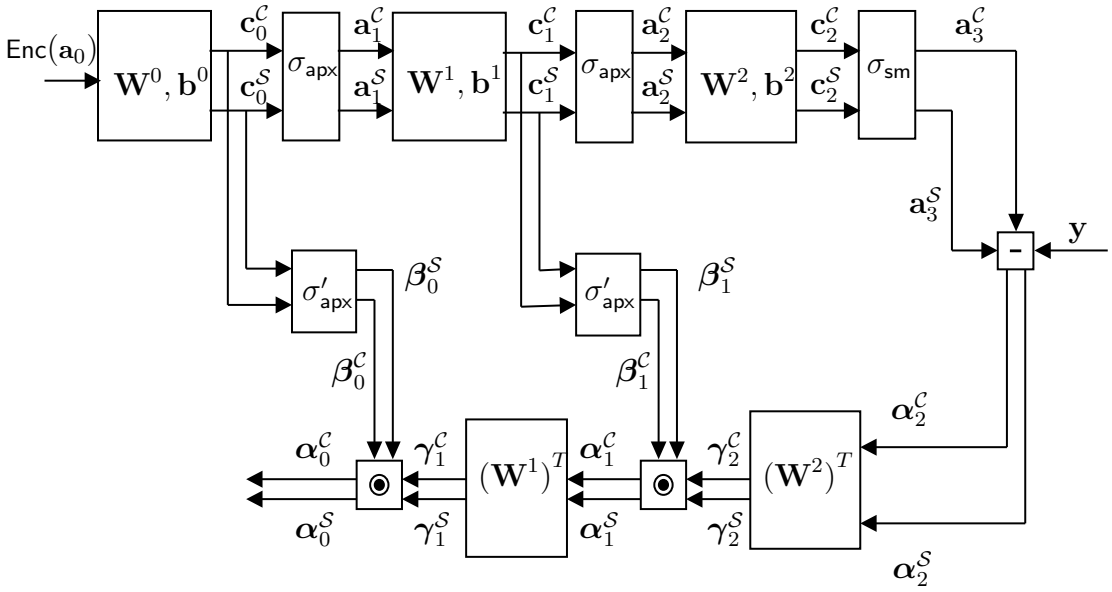

Figure 5.7: Neural Network Training Computation using FHE and Garbled Circuit Techniques

5.3.1 Secure Forward Propagation

The steps between the client and server in forward propagation are explained in detail. A summary of one complete linear and nonlinear layer computation is provided in Figure 5.9.

- **Joint Linear Layer Computation:** The client encrypts its input $\mathbf{a}_0 = (\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ using the BFV scheme as $\mathsf{Enc}(\mathbf{a}_0)$ where $\mathcal{D}$ is the dataset. The server holds $(\mathbf{W}^i, \mathbf{b}^i)$. On receiving the encrypted input from the client, the server begins the matrix-vector multiplication and creates additive shares of the result, as explained in Section 5.2.1. The computation for the i-th linear layer is given by:

$$\mathsf{Enc}(\mathbf{c}_i) = \mathsf{Eval}_{\oplus}(\mathsf{Eval}_{\otimes}(\mathsf{Enc}(\mathbf{a}_i), \mathbf{W}^i), \mathbf{b}^i)$$
$$\mathsf{Enc}(\mathbf{c}_i^{\mathcal{C}}) = \mathsf{Eval}_{\ominus}(\mathsf{Enc}(\mathbf{c}_i), \mathbf{c}_i^{\mathcal{S}})$$

  where the server randomly generates $\mathbf{c}_i^{\mathcal{S}}$ and holds it, and the client holds $\mathbf{c}_i^{\mathcal{C}}$.

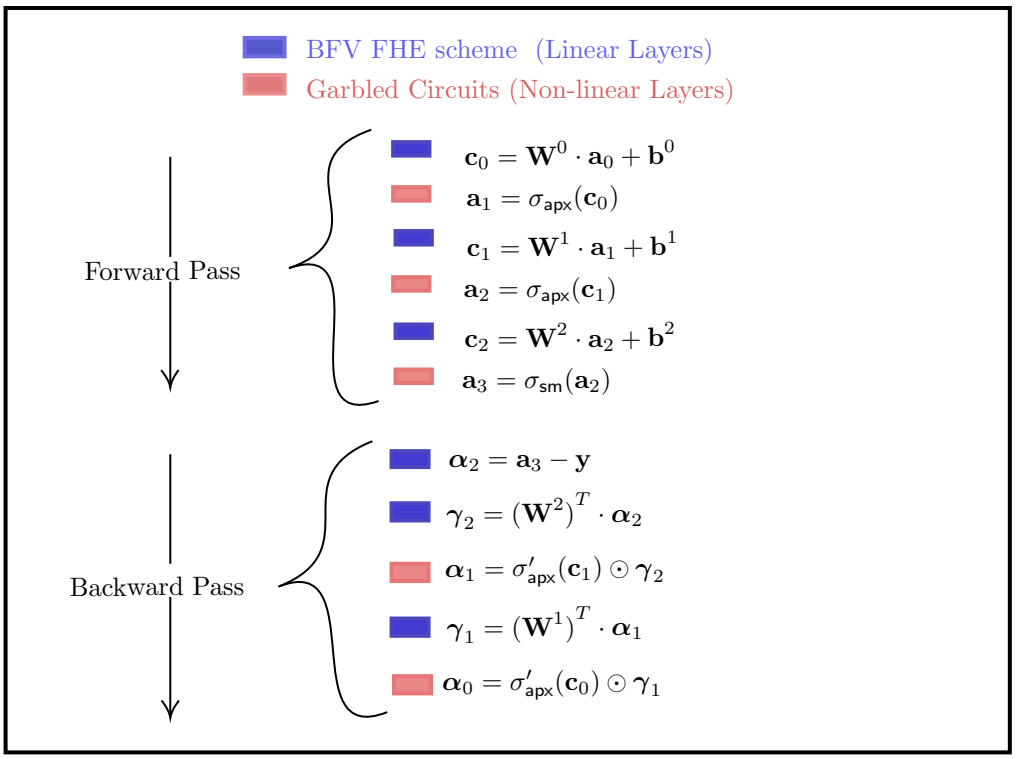

Figure 5.8: The Use of FHE and Garbled Circuit in Our Gradient Computation

- **Joint Non-linear Layer Computation:** Next the server and the client jointly compute the activation function and its derivative on $\mathbf{c}_i^{\mathcal{S}}$ and $\mathbf{c}_i^{\mathcal{C}}$ simultaneously as shown in Figure 5.3 in Section 5.2.2 where the client randomly picks $\mathbf{a}_i^{\mathcal{C}}$ and $\boldsymbol{\beta}_i^{\mathcal{C}}$. The server receives $(\mathbf{a}_i^{\mathcal{S}}, \boldsymbol{\beta}_i^{\mathcal{S}})$ as the shares of the outputs of the activation function and its derivative.

$$(\mathbf{a}_i^{\mathcal{S}}, \boldsymbol{\beta}_i^{\mathcal{S}}) \leftarrow \mathsf{GC}(\mathcal{SCKT}, (\mathbf{c}_i^{\mathcal{C}}, \mathbf{a}_i^{\mathcal{C}}, \boldsymbol{\beta}_i^{\mathcal{C}}), (\mathbf{c}_i^{\mathcal{S}})), 0 \leq i \leq 1,$$

where $\mathcal{SCKT}$ is the optimized circuit for computing the approximated sigmoid and its derivative in Eqs. (3.1) and (3.2).

5.3.2 Secure Backward Propagation

By the end of the forward propagation, the client holds a share $\mathbf{a}_3^{\mathcal{C}}$ and the server holds another share $\mathbf{a}_3^{\mathcal{S}}$ of the softmax layer output ($\mathbf{a}_3 = \mathbf{a}_3^{\mathcal{C}} + \mathbf{a}_3^{\mathcal{S}}$). In the backward

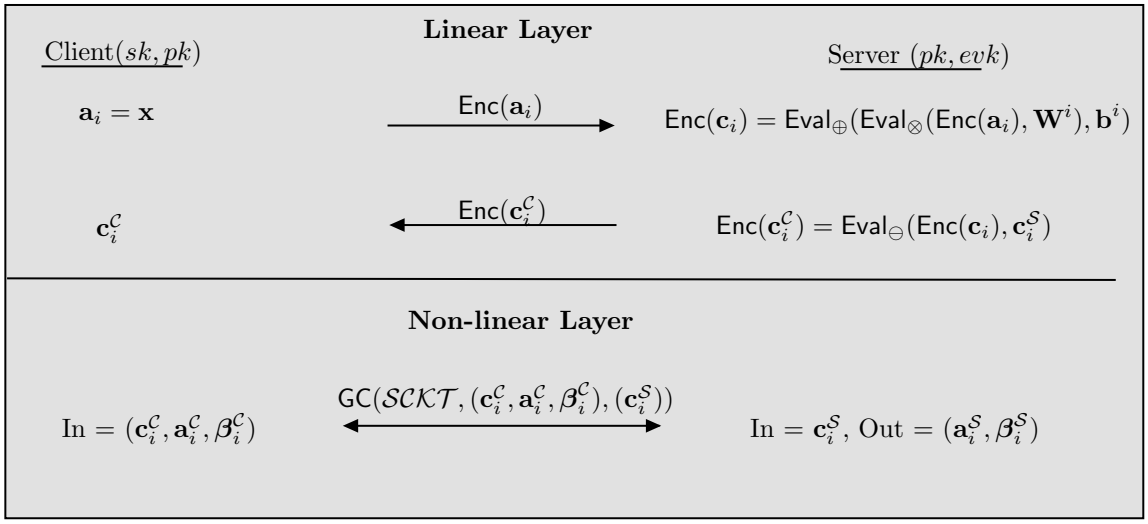

Figure 5.9: Single Iteration of Forward Propagation

propagation, the first task is to compute the shares of $\boldsymbol{\alpha}_2 = \mathbf{a}_3 - \mathbf{y}$, which can be easily computed as $\boldsymbol{\alpha}_2 = (\mathbf{a}_3^{\mathcal{S}} + \mathbf{a}_3^{\mathcal{C}}) - \mathbf{y} = \mathbf{a}_3^{\mathcal{S}} + (\mathbf{a}_3^{\mathcal{C}} - \mathbf{y})$ where for the server the share is $\boldsymbol{\alpha}_2^{\mathcal{C}} = \mathbf{a}_3^{\mathcal{S}}$ and the share for the client is $\boldsymbol{\alpha}_2^{\mathcal{C}} = (\mathbf{a}_3^{\mathcal{C}} - \mathbf{y})$ as the data point $(\mathbf{x}, \mathbf{y})$ belongs to the client who can compute $\boldsymbol{\alpha}_2^{\mathcal{C}}$ without any interaction with the server. Figure 5.10 shows one round of the linear and non-linear computation of the backward propagation computation, except the last layer.

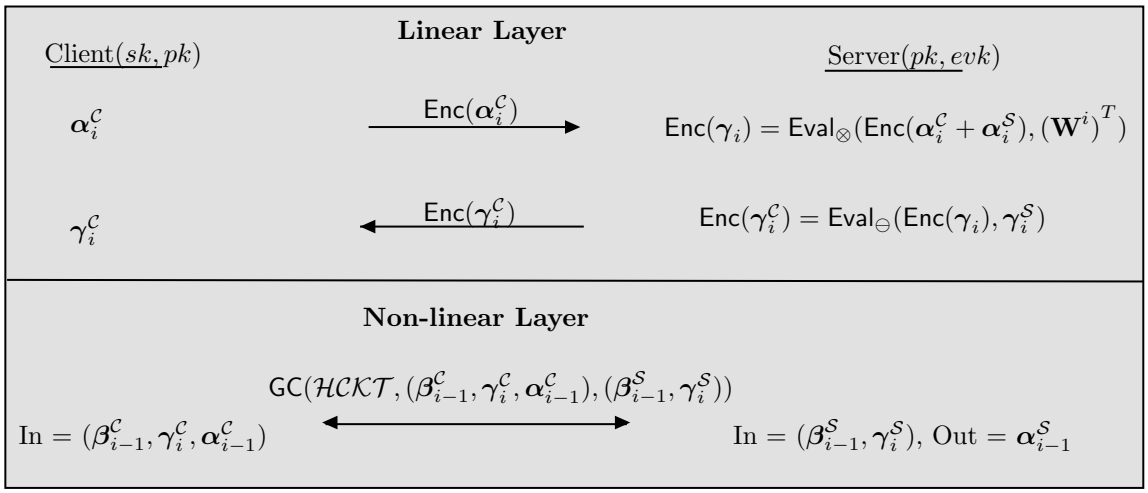

Figure 5.10: Single Iteration of Backward Propagation

- **Joint Linear Layer Computation.** For this layer, the client and server securely compute the propagation of loss $\boldsymbol{\gamma}_i$ through the model $\boldsymbol{\theta}$ using the

shares $\boldsymbol{\alpha}_i^{\mathcal{C}}$ and $\boldsymbol{\alpha}_i^{\mathcal{S}}$ and both receive an additive share of $\boldsymbol{\gamma}$. For the i-th layer, the server holds the weight matrix $(\mathbf{W}^i)^T$ in cleartext and a share $\boldsymbol{\alpha}_i^{\mathcal{S}}$ of $\boldsymbol{\alpha}_i$ and the client holds $\boldsymbol{\alpha}_i^{\mathcal{C}}$. The homomorphic matrix multiplication from shares and creating two additive shares are done as follows.

$$\begin{aligned}\mathsf{Enc}(\boldsymbol{\gamma}_i) &= \mathsf{Eval}_{\otimes}(\mathsf{Eval}_{\oplus}(\mathsf{Enc}(\boldsymbol{\alpha}_i^{\mathcal{C}}), \boldsymbol{\alpha}_i^{\mathcal{S}}), (\mathbf{W}^i)^T) \\ &= \mathsf{Eval}_{\otimes}(\mathsf{Eval}_{\oplus}(\mathsf{Enc}(\boldsymbol{\alpha}_i^{\mathcal{C}} + \boldsymbol{\alpha}_i^{\mathcal{S}}), (\mathbf{W}^i)^T) \\ \mathsf{Enc}(\boldsymbol{\gamma}_i^{\mathcal{C}}) &= \mathsf{Eval}_{\ominus}(\mathsf{Enc}(\boldsymbol{\gamma}_i), \boldsymbol{\gamma}_i^{\mathcal{S}}), 2 \geq i \geq 1\end{aligned}$$

where the server randomly generates $\boldsymbol{\gamma}_i^{\mathcal{S}}$ and holds it and the client holds $\boldsymbol{\gamma}_i^{\mathcal{C}}$.

- **Joint Non-linear Layer Computation.** The server and the client jointly compute the Hadamard product of two vectors $\boldsymbol{\beta}_{i-1}$ and $\boldsymbol{\gamma}_i$ as $\boldsymbol{\alpha}_{i-1} = \boldsymbol{\beta}_{i-1} \odot \boldsymbol{\gamma}_i = (\boldsymbol{\beta}_{i-1}^{\mathcal{C}} + \boldsymbol{\beta}_{i-1}^{\mathcal{S}}) \odot (\boldsymbol{\gamma}_i^{\mathcal{C}} + \boldsymbol{\gamma}_i^{\mathcal{S}})$ as shown in Figure 5.5 in Section 5.2.2 where the client randomly picks $\boldsymbol{\alpha}_{i-1}^{\mathcal{C}}$. The server receives $\boldsymbol{\alpha}_{i-1}^{\mathcal{S}}$ as the share of the outputs of the non-linear layer.

$$\boldsymbol{\alpha}_{i-1}^{\mathcal{S}} \leftarrow \mathsf{GC}(\mathcal{HCKT}, (\boldsymbol{\beta}_{i-1}^{\mathcal{C}}, \boldsymbol{\gamma}_i^{\mathcal{C}}, \boldsymbol{\alpha}_{i-1}^{\mathcal{C}}), (\boldsymbol{\beta}_{i-1}^{\mathcal{S}}, \boldsymbol{\gamma}_i^{\mathcal{S}})), 2 \geq i \geq 1,$$

where $\mathcal{HCKT}$ is the circuit for computing the Hadamard product.

Gradient Computation. After securely computing the forward and backward propagations, the client holds the additive shares $\{\mathbf{a}_2^{\mathcal{C}}, \mathbf{a}_1^{\mathcal{C}}, \mathbf{a}_0^{\mathcal{C}}, \boldsymbol{\alpha}_2^{\mathcal{C}}, \boldsymbol{\alpha}_1^{\mathcal{C}}, \boldsymbol{\alpha}_0^{\mathcal{C}}\}$ of the intermediate computations that are required to compute the gradient $(\nabla\mathbf{W}^i, \nabla\mathbf{b}^i)$. Similarly the server holds $\{\mathbf{a}_2^{\mathcal{S}}, \mathbf{a}_1^{\mathcal{S}}, \mathbf{a}_0^{\mathcal{S}}, \boldsymbol{\alpha}_2^{\mathcal{S}}, \boldsymbol{\alpha}_1^{\mathcal{S}}, \boldsymbol{\alpha}_0^{\mathcal{S}}\}$. The bias gradient $\nabla\mathbf{b}^i$ is computed using the server's share $\boldsymbol{\alpha}_i^{\mathcal{S}}$ and client's share $\boldsymbol{\alpha}_i^{\mathcal{C}}$. Table 5.1 summarizes the shares for $\nabla\mathbf{b}^i$. The gradient $\nabla\mathbf{W}^i$ is also computed using the $\boldsymbol{\alpha}_i$ and $\mathbf{a}_i$ shares of the client and server. The model $\nabla\mathbf{W}^i$ on the other hand is computed using the

Table 5.1: Share Distribution for The Bias Vector

Gradient	**Client's share**	**Server's share**
$\nabla \mathbf{b}_0 = \boldsymbol{\alpha}_0 = \boldsymbol{\alpha}_0^{\mathcal{C}} + \boldsymbol{\alpha}_0^{\mathcal{S}}$	$\boldsymbol{\alpha}_0^{\mathcal{C}}$	$\boldsymbol{\alpha}_0^{\mathcal{S}}$
$\nabla \mathbf{b}_1 = \boldsymbol{\alpha}_1 = \boldsymbol{\alpha}_1^{\mathcal{C}} + \boldsymbol{\alpha}_1^{\mathcal{S}}$	$\boldsymbol{\alpha}_1^{\mathcal{C}}$	$\boldsymbol{\alpha}_1^{\mathcal{S}}$
$\nabla \mathbf{b}_2 = \boldsymbol{\alpha}_2 = \boldsymbol{\alpha}_2^{\mathcal{C}} + \boldsymbol{\alpha}_2^{\mathcal{S}}$	$\boldsymbol{\alpha}_2^{\mathcal{C}}$	$\boldsymbol{\alpha}_2^{\mathcal{S}}$

shares of the server $(\mathbf{a}_i^{\mathcal{S}}, \boldsymbol{\alpha}_i^{\mathcal{S}})$ and shares of the client $(\mathbf{a}_i^{\mathcal{C}}, \boldsymbol{\alpha}_i^{\mathcal{C}})$ as

$$\nabla \mathbf{W}^i = (\boldsymbol{\alpha}_i^{\mathcal{C}} + \boldsymbol{\alpha}_i^{\mathcal{S}}) \cdot ((\mathbf{a}_i^{\mathcal{C}})^T + (\mathbf{a}_i^{\mathcal{S}})^T) \tag{5.3}$$

An additive share of $\nabla \mathbf{W}^i$ for the client and the server can be computed using the HE scheme. Figure 5.12 shows the steps of this computation. Finally, Figure 5.11 shows all the steps for our secure gradient computation protocol for the two-layer DNN. Using the above sub-protocols for the forward and backward propagation a secure gradient computation protocol for an ℓ-layer DNN is constructed. The GC operations are represented in the forward propagation as $\mathsf{GC}(\mathcal{SCKT}1)$ and $\mathsf{GC}(\mathcal{SCKT}2)$ for the first layer and second layer sigmoid computation and this is given by $\mathsf{GC}(\mathcal{SCKT}, (\mathbf{c}_0^{\mathcal{C}}, \mathbf{a}_1^{\mathcal{C}}, \boldsymbol{\beta}_0^{\mathcal{C}}), (\mathbf{c}_0^{\mathcal{S}}))$ and $\mathsf{GC}(\mathcal{SCKT}, (\mathbf{c}_1^{\mathcal{C}}, \mathbf{a}_2^{\mathcal{C}}, \boldsymbol{\beta}_1^{\mathcal{C}}), (\mathbf{c}_1^{\mathcal{S}}))$ respectively. Similarly, for the backward propagation, $\mathsf{GC}(\mathcal{HCKT}2)$ and $\mathsf{GC}(\mathcal{HCKT}1)$ for the second layer and the first layer Hadamard product computation and this is given by $\mathsf{GC}(\mathcal{HCKT}, (\boldsymbol{\beta}_1^{\mathcal{C}}, \boldsymbol{\gamma}_2^{\mathcal{C}}, \boldsymbol{\alpha}_1^{\mathcal{C}}), (\boldsymbol{\beta}_1^{\mathcal{S}}, \boldsymbol{\gamma}_2^{\mathcal{S}}))$ and $\mathsf{GC}(\mathcal{HCKT}, (\boldsymbol{\beta}_0^{\mathcal{C}}, \boldsymbol{\gamma}_1^{\mathcal{C}}, \boldsymbol{\alpha}_0^{\mathcal{C}}), (\boldsymbol{\beta}_0^{\mathcal{S}}, \boldsymbol{\gamma}_1^{\mathcal{S}}))$, respectively.

5.3.3 Security Analysis

In our gradient computation protocol, the three fundamental cryptographic schemes or protocols used are an FHE scheme (i.e., the BFV scheme), Yao's GC and a two-party additive secret sharing scheme. It has been proven in the literature that these individual protocols are secure, which is summarized below.

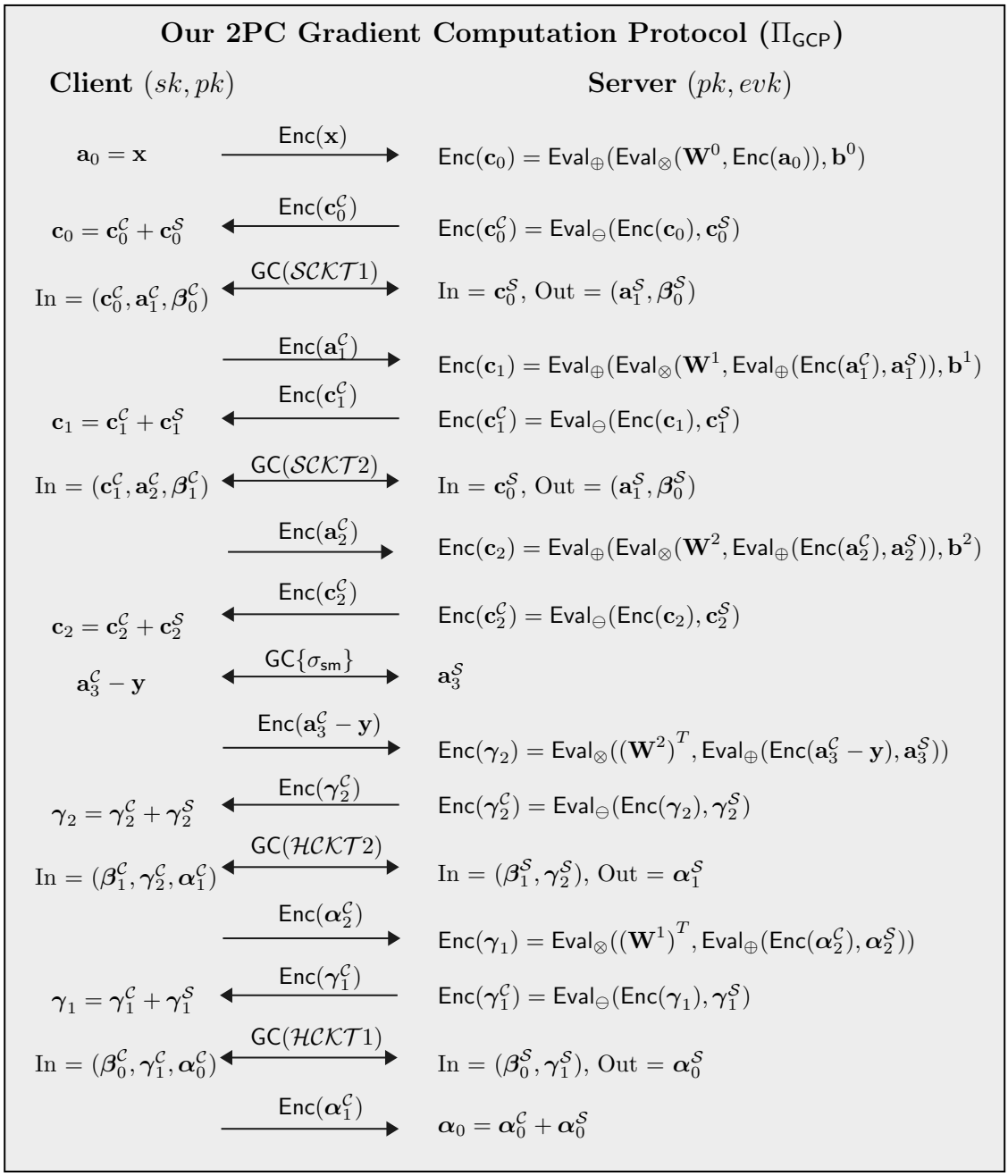

Figure 5.11: Our Proposed Two-party Gradient Computation Protocol

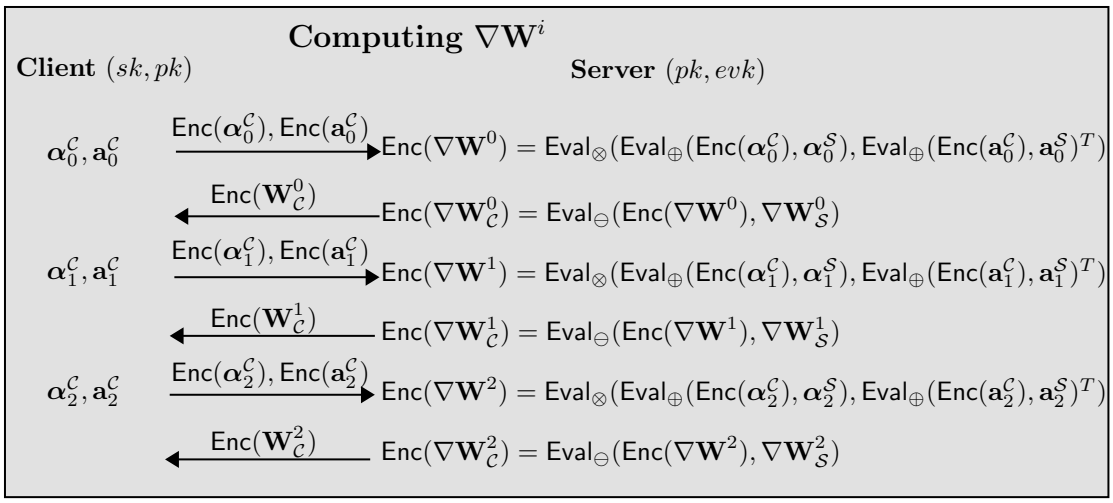

Figure 5.12: Computation of $\nabla \mathbf{W}^i$

Semantic Security or IND-CPA Security. A HE scheme is said to be secure if no adversary has an advantage in guessing (with better than a 50% chance) whether a given ciphertext is encryption of either one of the two equally distinct messages. Moreover, the encryption scheme can be defined as indistinguishable, and the encryption does not disclose any information regarding plaintexts.

Theorem 5.3.1 (Semantic Security or IND-CPA Security)**.** *[46] For a public key pk and any two messages $\{m_0, m_1\}$, we require $\{pk,$ BFV.Encrypt$(pk, m_0)\} \approx_c \{pk,$ BFV.Encrypt$(pk, m_1)\}$, where the two distributions are over the random choice of pk and the randomness of the encryption algorithm.*

Security of Additive Secret Sharing. An additive secret-sharing scheme is unconditionally secure when the computation is performed with shares of each party and no one party can obtain the actual value without involving every party's share. Hence, it does not leak any information.

Theorem 5.3.2. *[54] Let $\mathbb{F}$ be a finite field and $x, y \in \mathbb{F}, n \in \mathbb{N}$ Let $\langle x_{\mathcal{C}} \rangle =$ Share$(\mathbb{F}, x, n, n)$ be the share of the client and $\langle x_{\mathcal{S}} \rangle) =$ Share$(\mathbb{F}, y, n, n)$ be the share of the client using Shamir's secret sharing. Then the secret z is obtained as $z =$ Reconstruct$(\mathbb{F}, \langle z \rangle) = x_{\mathcal{C}} + y_{\mathcal{S}}$*

Security of Garbled Circuits. The privacy of the GC arises from the fact that every output label is encrypted using the secure encryption scheme and only the output label corresponding to the input can be decrypted at a time by using decryption keys. Since the wire labels are random strings the garbler can encode its own private inputs into the circuit. The evaluator uses its keys to iteratively decrypt the output and only the output corresponding to the input labels is legit. This is defined in [55].

Theorem 5.3.3 (Non-adaptive input privacy.)**.** *[46] Garbled circuit applications rely on the fact that the security notion of the circuit GC along with encoded input x and decoding keys dk reveal the function $f(x)$. Through simulation-based definition, intuitively a set of inputs x, an efficient adversary that holds (GC, x, dk) will not learn anything beyond $f(x)$.*

Security of Our Protocol. The security of our gradient computation protocol is summarized in Theorem 5.3.4.

Theorem 5.3.4. *Assume that the FHE scheme is semantically secure, the garbled circuit and additive secret-sharing protocols are secure. Our gradient computation protocol* (Π_{LGCP}) *between the client and the server is secure, meaning it does not leak any information about the client's private input* $(\boldsymbol{x}, \boldsymbol{y})$ *and the server's model* $\boldsymbol{\theta}$*.*

Proof sketch. Our gradient computation protocol is executed under the privacy preservation assumption of a semi-honest adversarial model. We assume the semi-honest adversary is non-adaptive and computationally bounded. In this security model, the client $\mathcal{C}$ and the server $\mathcal{S}$ should not be able to retrieve any information other than inputs, outputs, and intermediate messages. Since our protocol is based on the BFV FHE scheme, Yao's GC and additive secret sharing which are proven to be secure, our protocol is also secure. For the linear layer, we use the BFV scheme which has proven to be semantically secure against chosen plaintext attacks as mentioned in Section 5.3.3. Hence, no information is revealed while computing the

linear layers because the input $\mathbf{x}$ is encrypted by $\mathcal{C}$ with its secret key which meets the requirement of 128-bit security. For the activation functions or the non-linear layers, we use 2PC Yao's garbled circuit which is proven to be secure as explained in Section 5.3.3. To put the two primitives together, we use additive secret sharing as the building block which is unconditionally secure when the computation is performed with the client's $\langle x \rangle_{\mathcal{C}}$ and server's share, $\langle x \rangle_{\mathcal{S}}$ seen in Section 5.3.3. Hence, by building our protocol on these secure primitives, we claim that our protocol is also secure.

5.4 Computational Complexity Analysis

5.4.1 Complexity Analysis for Gradient Computation

The computational complexity for any protocol is analyzed by estimating the time and space complexity of operations performed by the client and the server to execute the protocol. We estimate the complexity by calculating the number of ciphertexts that need to be exchanged, the number of homomorphic operations, and the number of gates computed for each layer of execution of the protocol between the server and the client. Let us consider n to be the number of slots in the FHE scheme, i.e., for the BFV scheme, $n = 8192$, n_i is the number of input neurons and n_o is the number of output neurons. The number of plaintext/ciphertexts required to encode the weight matrix is calculated as $\lceil \frac{n_i \times (n_o+1)}{n} \rceil$ for each layer ℓ. The number of constant multiplications can be calculated as $\lceil \frac{n_i \times (n_o+1)}{n} \rceil$, the number of rotations as $(\lceil \frac{n_i \times (n_o+1)}{n} \rceil - 1)$ and constant additions/subtractions as $\lceil \frac{n_i \times (n_o+1)}{n} \rceil$. For example, if $n = 8192$, the dimensions of the weight matrices for layers 1, 2 and 3 of the LeNet-300-100 architecture are ($n_i^1 = 300$, and $n_o^1 = 784$), ($n_i^2 = 100$, and $n_o^2 = 300$) and ($n_i^3 = 10$, and $n_o^3 = 100$) the number of plaintext/ciphertexts for the MNIST dataset is computed as $\lceil \frac{300 \times (784+1)}{8192} \rceil \approx 30$ for the first layer, $\lceil \frac{100 \times (300+1)}{8192} \rceil \approx 4$

for the second and $\lceil \frac{10\times(100+1)}{8192} \rceil \approx 1$ for the last layer which is a total of ≈ 35 ciphertexts. The summary of complexity analysis for a single data point $\mathbf{x}$ is given in Table 5.2. The server tends to perform all the homomorphic evaluation operations including constant multiplications, additions and subtractions. The client only has the role of encryption and decryption of the sent and received ciphertexts. The complexity of garbled circuits is shown in Table 5.2. We can Similarly, it is possible to estimate the complexity of a given garbled circuit computation by accessing the number of unit operations during each circuit computation. Since a gabled circuit is used to perform operations of sigmoid, addition, sigmoid derivative (sigmoid'), and Hadamard product, the number of unit operations for each of them is computed. Let n_o^j be the number of output neurons for the j-th layer in an ℓ layer neural network. The computation estimates are done for gradient computation in Table 5.2.

Table 5.2: Complexity of Gradient Computation for ℓ Layers

HE				
-	**Ciphertexts**	$\mathsf{Eval}_{\otimes}$	$\mathsf{Eval}_{\oplus/\ominus}$	**Rotations**
Complexity	-	$\mathcal{O}(n\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Calculation	$\sum_{j=0}^{\ell}\lceil \frac{n_i^j\times(n_o^j+1)}{n} \rceil$	$\sum_{j=0}^{\ell}\lceil \frac{n_i^j\times(n_o^j+1)}{n} \rceil$	$\sum_{j=0}^{\ell}\lceil \frac{n_i^j\times(n_o^j+1)}{n} \rceil$	$\frac{1}{2}\sum_{j=0}^{\ell}\lceil \frac{n_i^j\times(n_o^j+1)}{n} \rceil - 1$
GC				
-	**Sigmoid**	**Addition**	**Sigmoid'**	**Hadamard**
GC operations	$\sum_{j=0}^{\ell}(n_o^j)$	$\sum_{j=0}^{\ell}(n_o^j)$	$2\sum_{j=0}^{\ell}(n_o^j)$	$\sum_{j=0}^{\ell} j$

5.5 Conclusion

In this chapter, we propose a secure gradient computation protocol. The construction of our protocol is based on a secure matrix-vector multiplication technique and an optimized GC computation for the non-linear computation of the forward and backward propagation. We proposed an efficient secure matrix-vector multiplication technique and created an optimized non-linear activation computation. We put the

techniques together to propose a secure gradient computation protocol that protects the privacy of the model held by the server and the input held by the client. Finally, we analyze the security of the individual components making up our protocol and hence achieving the client's and the server's input privacy.

Chapter 6

Efficient Privacy-Preserving Deep Neural Network Training Protocol in Federated Learning

In this chapter, we present our secure DNN training protocol in the federated learning setting. The construction of our protocol is based on our new gradient computation protocol in Chapter 5 and an aggregation protocol.

6.1 Secure DNN Training Building Blocks

6.1.1 Federated Private DNN Training

Privacy in Federated Learning. As mentioned in Chapter 4, we consider the FL setting, shown in Figure 6.1, as used in [61, 22, 79, 82, 64], where there are m clients, denoted by $\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_m\}$, and a centralised server denoted by $\mathcal{S}$ that coordinates the training process. Each client $\mathcal{C}_i$ has a private dataset $\mathcal{D}_i, 1 \leq i \leq m$, and the server holds the model $\boldsymbol{\theta} = \left\{(\mathbf{W}^0, \mathbf{b}^0), \ldots, (\mathbf{W}^{\ell-1}, \mathbf{b}^{\ell-1})\right\}$ that will be trained on the combined dataset $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_m$. In a traditional FL [16], at the

j-th iteration the server, broadcasts the current model $\boldsymbol{\theta}^j$ to the participating clients in the training process, and each client $\mathcal{C}_i$ computes a local gradient of $\boldsymbol{\theta}^j$, denoted by $\nabla\boldsymbol{\theta}^j(\mathcal{D}_i)$, on the dataset $\mathcal{D}_i, 1 \leq i \leq m$, and sends its local gradient to the server. After receiving the local gradients from the clients $\{\nabla\boldsymbol{\theta}^j(\mathcal{D}_i)\}_{i=1}^m$, the server updates the model as $\boldsymbol{\theta}^{j+1} \leftarrow \boldsymbol{\theta}^j - \frac{1}{m}\sum_{k=1}^m \nabla\boldsymbol{\theta}^j(\mathcal{D}_i)$. This process is continued until a desirable accuracy is achieved. Note that in the traditional FL, only user data privacy is provided when the dataset size is more than one. It does not provide the model $\boldsymbol{\theta}^j$ privacy as a malicious participant can abuse the model.

Strongly Secure DNN Training in FL. Our privacy goal for training a DNN in FL is to provide both data privacy and model privacy, meaning in the FL training process the model will not be given to the clients in cleartext. We use the PrivFL framework (see Figure 6.2), proposed by Mandal and Gong [61], to design our privacy-preserving DNN training protocol. There are two key phases in the PrivFL protocol framework: (1) computing the local gradient between the server and a client on the client's dataset; and (2) executing a secure aggregation protocol among the server and the set of clients. Our techniques for privacy-preserving DNN training in PrivFL are as follows:

- *Local DNN gradient computation.* The server and the client compute the local gradient $\nabla\boldsymbol{\theta}^j(\mathcal{D}_i)$ privately using our protocol in Chapter 5, where the server holds $\boldsymbol{\theta}^j$ and the client holds $\mathcal{D}_i$. By the end of the computation, the server and the client hold an additive share of $\nabla_{ji} = \nabla\boldsymbol{\theta}^j(\mathcal{D}_i)$, i.e., $(\nabla_{ji}^{\mathcal{S}}, \nabla_{ji}^{\mathcal{C}_i}) \leftarrow \nabla_{ji}^{\mathcal{S}} + \nabla_{ji}^{\mathcal{C}_i} = \nabla\boldsymbol{\theta}^j(\mathcal{D}_i), 1 \leq i \leq m$. In this protocol, the model is not given to the clients in cleartext and the clients do not give their dataset to the server in cleartext. Note that the server runs the local gradient computation steps individually with each client. After running the local gradient computation protocol with each client, the server holds m shares $\{\nabla_{j1}^{\mathcal{S}}, \nabla_{j2}^{\mathcal{S}}, \cdots, \nabla_{jm}^{\mathcal{S}}\}$.

- *Aggregating the gradients.* After finishing the local gradient computation with each client, the server and all the clients run a secure aggregation protocol to compute $\sum_{i=1}^{m} \nabla_{ji}^{\mathcal{C}_i}$. Then the server can compute $\sum_{k=1}^{m} \nabla \boldsymbol{\theta}^j(\mathcal{D}_i) = \sum_{k=1}^{m} \nabla_{jk}^{\mathcal{S}} + \sum_{k=1}^{m} \nabla_{jk}^{\mathcal{C}_k}$, followed by updating the model as $\boldsymbol{\theta}^{j+1} \leftarrow \boldsymbol{\theta}^j - \eta \frac{1}{m} \sum_{k=1}^{m} \nabla \boldsymbol{\theta}^j(\mathcal{D}_i)$. These two steps are iteratively executed until the expected model accuracy is achieved.

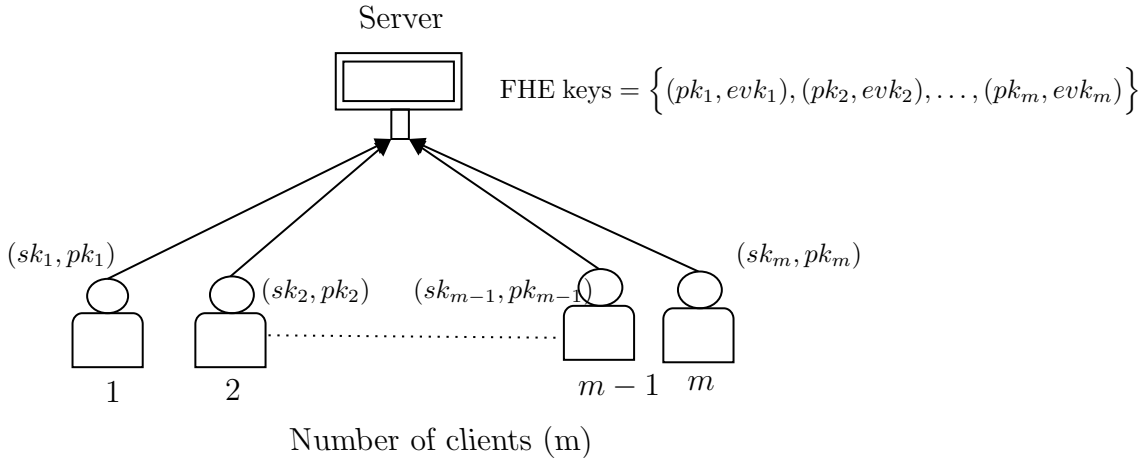

Figure 6.1: FHE Keys Setup for the Participants in the DNN Training Process

6.1.2 Secure Local DNN Gradient Computation

We now explain how to securely compute the local gradient $\nabla \boldsymbol{\theta}^j(\mathcal{D}_i)$ on dataset $\mathcal{D}_i$ between the server ($\mathcal{S}$) and the client $\mathcal{C}_i$. The idea for the local gradient computation protocol is that for each data point in $\mathcal{D}_i$ the server and the client $\mathcal{C}_i$ jointly execute our gradient computation protocol Π_{GCP} and then compute the average on all gradient values. Algorithm 1 provides all steps of securely computing $\nabla \boldsymbol{\theta}^j(\mathcal{D}_i)$. After a successful completion of the protocol Π_{LGCP}, the server receives a share $\nabla_{ji}^{\mathcal{S}}$ and the client $\mathcal{C}_i$ receives another $\nabla_{ji}^{\mathcal{C}_i}$ of $\nabla \boldsymbol{\theta}^j(\mathcal{D}_i)$. Splitting the local gradient value $\nabla \boldsymbol{\theta}^j(\mathcal{D}_i)$ between the client and the server does not reveal any information about the actual local gradient value.

Algorithm 1 Local Gradient Computation Protocol (Π_{LGCP}) on a Dataset

Input: Server's input $\boldsymbol{\theta}^j$ and client $\mathcal{C}_i$'s input $\mathcal{D}_i$
Output: Server's receives $\nabla_{ji}^{\mathcal{S}}$ and $\mathcal{C}_i$ receives $\nabla_{ji}^{\mathcal{C}_i}$ s.t. $\nabla_{ji}^{\mathcal{S}} + \nabla_{ji}^{\mathcal{C}_i} = \nabla\boldsymbol{\theta}^j(\mathcal{D}_i)$.
procedure PROTOCOL($(\nabla_{ji}^{\mathcal{S}}, \nabla_{ji}^{\mathcal{C}_i}) \leftarrow \Pi_{\mathsf{LGCP}}(\boldsymbol{\theta}^j, \mathcal{D}_i)$)
    Set $\nabla_{ji}^{\mathcal{S}} \leftarrow \mathbf{0}$ and $\nabla_{ji}^{\mathcal{C}_i} \leftarrow \mathbf{0}$;
    for each $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_i$ **do**
        The server ($\mathcal{S}$) and the client ($\mathcal{C}$) jointly run Π_{GCP} with $\mathcal{S}$'s input $\boldsymbol{\theta}^j$ and $\mathcal{C}_i$'s input $(\mathbf{x}, \mathbf{y})$ and both receive an additive share of the gradient on $(\mathbf{x}, \mathbf{y})$, i.e., $(\nabla\boldsymbol{\theta}^j(\mathbf{x}, \mathbf{y})^{\mathcal{S}}, \nabla\boldsymbol{\theta}^j(\mathbf{x}, \mathbf{y})^{\mathcal{C}_i}) \leftarrow \Pi_{\mathsf{GCP}}(\boldsymbol{\theta}^j, (\mathbf{x}, \mathbf{y}))$.
        Server computes $\nabla_{ji}^{\mathcal{S}} \leftarrow \nabla_{ji}^{\mathcal{S}} + \nabla\boldsymbol{\theta}^j(\mathbf{x}, \mathbf{y})^{\mathcal{S}}$
        Client $\mathcal{C}_i$ computes $\nabla_{ji}^{\mathcal{C}_i} \leftarrow \nabla_{ji}^{\mathcal{C}_i} + \nabla\boldsymbol{\theta}^j(\mathbf{x}, \mathbf{y})^{\mathcal{C}_i}$
    end for
    Server computes the average gradient on the dataset, i.e., $\nabla_{ji}^{\mathcal{S}} \leftarrow \frac{1}{|\mathcal{D}_i|}\nabla_{ji}^{\mathcal{S}}$
    Client $\mathcal{C}_i$ computes the average gradient on the dataset, i.e., $\nabla_{ji}^{\mathcal{C}_i} \leftarrow \frac{1}{|\mathcal{D}_i|}\nabla_{ji}^{\mathcal{C}_i}$
    return $(\nabla_{ji}^{\mathcal{S}}, \nabla_{ji}^{\mathcal{C}_i})$
end procedure

6.1.3 Secure Aggregation Protocol for Local Gradient Shares

After executing the local gradient protocol Π_{LGCP} between each client and the server, each client holds a share $\nabla_{ji}^{\mathcal{C}_i}$ of the local gradient on its dataset. However, to update the model $\boldsymbol{\theta}^j$, the server needs the sum of all local gradients. We use a secure aggregation protocol to aggregate the shares of the local gradient for all clients in the set $\mathcal{C}$ and after that, the server adds it with its own shares of all local gradients. In the aggregation protocol, at the j-th round of the training, each client has an input $\nabla_{ji}^{\mathcal{C}_i}$ that a share of the local gradient, and the server wishes to get $\nabla_j^{\mathcal{C}} = \sum_{k=1}^{m} \nabla_{jk}^{\mathcal{C}_k}$ as an output from the protocol. We denote this protocol Π_{AggShare} by:

$$\nabla_j^{\mathcal{C}} \leftarrow \Pi_{\mathsf{AggShare}}\Big(\mathcal{C}, \big\{\nabla_{jk}^{\mathcal{C}_k}, 1 \leq k \leq m\big\}\Big).$$

In our DNN training protocol, we use the secure aggregation protocol from [61]. For completeness, the protocol is explained with an example in Example 6.1.1, but the details of the protocol are not provided. See [61] for the details.

Example 6.1.1. Suppose there are five clients $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_5\}$ and each

client $\mathcal{C}_i$ has a vector $\mathbf{v}_i$ as input. The server wishes to learn an aggregate-sum of $\mathbf{v}_i$ from the clients, i.e., $\mathbf{v} = \sum_{i=1}^{5} \mathbf{v}_i$. The secure aggregation protocol $\mathbf{v} \leftarrow \Pi_{\mathsf{AggShare}}\Big(\mathcal{C}, \big\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5\big\}\Big)$ works as follows:

1. **Client establishing pairwise keys.** Each client $\mathcal{C}_i$ establishes keys with other clients using the Diffie-Hellman (DH) key exchange protocol as

   - Client $\mathcal{C}_i$ generates a DH key k_i and computes g^{k_i} and gives it to the server, $1 \leq i \leq 6$.
   - The server broadcasts all DH keys $\{g^{k_1}, g^{k_2}, \cdots, g^{k_5}\}$ to all clients.
   - $\mathcal{C}_i$ computes a key k_{ij} for other clients $\mathcal{C}_j, j \neq i$ as $k_{ij} = g^{k_i k_j} = (g^{k_j})^{k_i}$ and generates keystreams $\mathbf{k}_{ij} \leftarrow \mathsf{PRG}(k_{ij})$ where $\mathsf{PRG}(\cdot)$ is a pseudorandom bit generator that generates keystream same as the vector length in bits.
   - Each client $\mathcal{C}_i$ has 4 pairwise keys $\{\mathbf{k}_{ij}, i \leq j\}$.

2. **Client encrypting its private input.** Each client $\mathcal{C}_i$ encrypts its input vector $\mathbf{v}_i$ as:

$$\mathcal{C}_1 : \ \mathbf{c}_1 = \mathbf{v}_1 + \mathbf{k}_{12} + \mathbf{k}_{13} + \mathbf{k}_{14} + \mathbf{k}_{15}$$

$$\mathcal{C}_2 : \ \mathbf{c}_2 = \mathbf{v}_2 - \mathbf{k}_{21} + \mathbf{k}_{23} + \mathbf{k}_{24} + \mathbf{k}_{25}$$

$$\mathcal{C}_3 : \ \mathbf{c}_3 = \mathbf{v}_3 - \mathbf{k}_{31} - \mathbf{k}_{32} + \mathbf{k}_{34} + \mathbf{k}_{35}$$

$$\mathcal{C}_4 : \ \mathbf{c}_4 = \mathbf{v}_4 - \mathbf{k}_{41} - \mathbf{k}_{42} - \mathbf{k}_{43} + \mathbf{k}_{45}$$

$$\mathcal{C}_5 : \ \mathbf{c}_5 = \mathbf{v}_5 - \mathbf{k}_{51} - \mathbf{k}_{52} - \mathbf{k}_{53} - \mathbf{k}_{54}$$

   Client $\mathcal{C}_i$ sends ciphertext $\mathbf{c}_i$ to the server.

3. **Server computing aggregate-sum.** After receiving these ciphertexts i.e. ciphertexts $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5\}$, the server computes $\mathbf{v} = \sum_{i=1}^{5} \mathbf{c}_i = \sum_{i=1}^{5} \mathbf{v}_i$ as $\mathbf{k}_{ij} = \mathbf{k}_{ji} = \mathsf{PRG}(g^{k_i k_j})$.

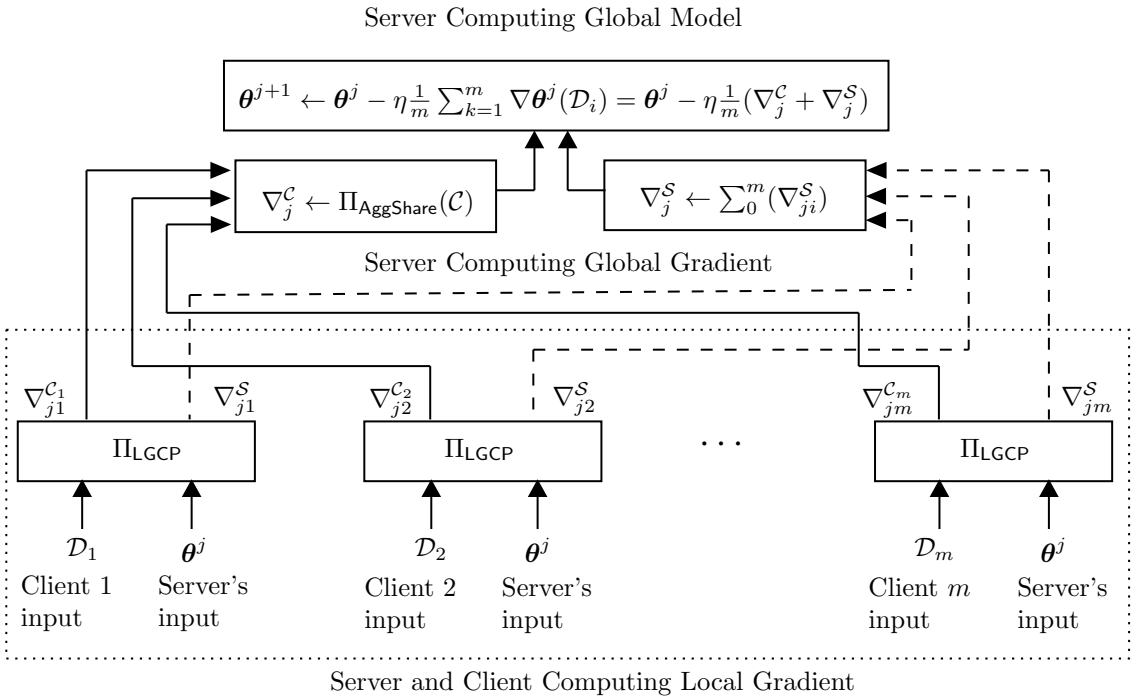

Figure 6.2: Protocol Flow for Privacy-preserving Federated Learning

6.2 Our Proposed Protocol: Putting it Together

To explain our privacy-preserving DNN training protocol, we put together all the protocols and phases explained in Section 6.1. Figure 6.2 shows a high-level overview of our DNN protocol in the PrivFL framework. Suppose there are m clients $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ participating in the DNN training process where each client has a private dataset $\mathcal{D}_i$. The server $\mathcal{S}$ holds the private model $\boldsymbol{\theta}$ to be trained where $\boldsymbol{\theta} = \Big\{(\mathbf{W}^0, \mathbf{b}^0), (\mathbf{W}^1, \mathbf{b}^1), \ldots, (\mathbf{W}^{\ell-1}, \mathbf{b}_{\ell-1})\Big\}$, ℓ is the number of layers. The construction of our privacy-preserving DNN training is based on the local gradient computation protocol in Section 6.1.2 and the secure aggregation protocol in Section 6.1.3. Our protocol has two main phases: 1) the *key setup phase*; and 2) the *DNN training phase*. In the key setup phase, each client establishes a pairwise-key with other clients using the DH key exchange protocol and also generates an FHE key and shares the public-key and the evaluation key with the server. As training a deep neural network is an iterative process, in the DNN training phase, the server and the clients jointly

compute the global gradient using the secure local gradient computation protocol Π_{LGCP} and the secure aggregation protocol (Π_{AggShare}). Algorithm 2 summarizes the steps of the protocol.

Algorithm 2 Privacy-preserving DNN Training Protocol in Federated Learning (Π_{PrivDNN})

Input: $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ is the set of clients participating in the DNN training process with their datasets $\mathcal{D}_i, 1 \leq i \leq m$. Server holds the initial model $\boldsymbol{\theta}^0$ to be trained on $\mathcal{D} = \cup_{i=1}^{n} \mathcal{D}_i$. Security level parameter κ.
Output: Server receives the trained model $\boldsymbol{\theta} = \{(\mathbf{W}^i, \mathbf{b}^i)\}_{i=0}^{\ell-1}$ of ℓ-layer.

▷ Server-assisted pairwise key establishment & FHE key setup

Clients establish a pairwise key with each other using the DH protocol where the server receives all DH public keys and broadcasts all DH keys to all the clients.
for each client $\mathcal{C}_i \in \mathcal{C}$ **do**
    Run the FHE key generation algorithm $(sk_i, pk_i, evk_i) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$.
    Send the public-key and evaluation key (pk_i, evk_i) to the server.
end for

▷ Iteratively training the DNN model $\boldsymbol{\theta}$

Set $j \leftarrow 0$.
repeat

▷ Server and each client jointly compute local gradient

    for each client $\mathcal{C}_i \in \mathcal{C}$ **do**
        Server and client $\mathcal{C}_i$ run the local gradient computation protocol and compute two shares of the local gradient $\nabla\boldsymbol{\theta}^j(\mathcal{D}_i)$, i.e., $(\nabla_{ji}^{\mathcal{S}}, \nabla_{ji}^{\mathcal{C}_i}) \leftarrow \Pi_{\mathsf{LGCP}}(\boldsymbol{\theta}^j, \mathcal{D}_i)$.
    end for

▷ Server and all clients aggregate the client shares of local gradients

    Server and all clients in $\mathcal{C}$ run the secure aggregation protocol to compute the clients' share of the global gradient $\nabla_j^{\mathcal{C}}$ as $\nabla_j^{\mathcal{C}} \leftarrow \mathsf{AggShare}\Big(\mathcal{C}, \{\nabla_{j1}^{\mathcal{C}_1}, \nabla_{j2}^{\mathcal{C}_1}, \cdots, \nabla_{jm}^{\mathcal{C}_m}\}\Big)$.

▷ Server computes global gradient and updates the model

    Server first computes its global gradient share $\nabla_j^{\mathcal{S}} = \sum_{i=1}^{m} \nabla_{ji}^{\mathcal{S}}$ and then updates the model as

$$\boldsymbol{\theta}^{j+1} \leftarrow \boldsymbol{\theta}^j - \eta \frac{1}{m} \sum_{k=1}^{m} \nabla\boldsymbol{\theta}^j(\mathcal{D}_i) = \boldsymbol{\theta}^j - \eta \frac{1}{m} (\nabla_j^{\mathcal{C}} + \nabla_j^{\mathcal{S}})$$

    .
    $j \leftarrow j + 1$.
until desired accuracy achieved
return $\boldsymbol{\theta}$

6.3 Security Analysis

In this section, we analyze the security of our proposed DNN training protocol (Π_{PrivDNN}) in Algorithm 2. The design of Π_{PrivDNN} is based on the local gradient protocol and a share aggregation protocol. The security of our DNN training protocol is summarized in Theorem 6.3.1.

Theorem 6.3.1. *Assume that the share aggregation protocol (Π_{AggShare}) is secure and the local gradient computation protocol (Π_{LGCP}) is secure. Then our DNN training protocol (Π_{PrivDNN}) is secure, meaning it does not leak any information about the client's private dataset $\mathcal{D}_i$ and the server's model $\boldsymbol{\theta}$.*

Proof sketch. We prove that our local gradient computation protocol is secure from Section 5.3.3 and hence leaks no information about the shares of the clients $\{\nabla_{j1}^{\mathcal{C}_1}, \nabla_{j2}^{\mathcal{C}_2}, \ldots, \nabla_{jm}^{\mathcal{C}_m}\}$ or that of the server's $\{\nabla_{j1}^{\mathcal{S}}, \nabla_{j2}^{\mathcal{S}}, \ldots, \nabla_{jm}^{\mathcal{S}}\}$. These shares and then aggregated using a secure aggregated protocol explained in Section 6.1.3 which is also proved to be secure. The shares of the client are aggregated as $\nabla_j^{\mathcal{C}} \leftarrow \Pi_{\mathsf{PrivDNN}}\left(\mathcal{C}, \nabla_{j1}^{\mathcal{C}_1}, \nabla_{j2}^{\mathcal{C}_2}, \ldots, \nabla_{jm}^{\mathcal{C}_m}\right)$ and the server's share is aggregated as follows $\nabla_j^{\mathcal{S}} \leftarrow \sum_0^m(\{\nabla_{j1}^{\mathcal{S}}, \nabla_{j2}^{\mathcal{S}}, \ldots, \nabla_{jm}^{\mathcal{S}})$ and both shares $\nabla_j^{\mathcal{C}}$ of the client, $\nabla_j^{\mathcal{S}}$ of the server does not give any information about the dataset $\mathcal{D}_i$ held by the client $\mathcal{C}_i$ or $\boldsymbol{\theta}^j$ held by the server and hence is secure. Finally, the global updated model is calculated using these shares as $\boldsymbol{\theta}^{j+1} \leftarrow \boldsymbol{\theta}^j - \eta\frac{1}{m}\sum_{k=1}^m \nabla\boldsymbol{\theta}^j(\mathcal{D}_i) = \boldsymbol{\theta}^j - \eta\frac{1}{m}(\nabla_j^{\mathcal{C}} + \nabla_j^{\mathcal{S}})$ which is securely computed and hence by building on Π_{LGCP} and Π_{AggShare}, our protocol is secure.

6.4 Computation Complexity Analysis

The computational complexity for our protocol is based on the computational complexity explained in Section 5.4. In the FL setting, we have multiple clients interacting with a single centralized server. The increase in the magnitude of the complexity

depends on two major aspects: (1) The number of data points in the dataset held by the client; and (2) The number of clients taking part in the protocol. Let us consider the number of clients to be m and the number of data points held by each client to be d. The clients train a neural network model with a n_i input nodes and n_0 output nodes for a layer l where inputs $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{d-1}\}$ held by a client are each packed in n slots of a BFV plaintext. We detail the number of operations on the client and server sides along with the number of ciphertexts exchanged in Table 6.1. Several closely related frameworks have been compared in Table 6.2.

Table 6.1: Client and Server Computation Complexity

Operations	Client	Server
$\mathsf{Eval}_{\otimes}$	-	$m \times d \times \sum_{j=0}^{\ell} \lceil \frac{n_i^j \times (n_o^j+1)}{n} \rceil$
$\mathsf{Eval}_{\oplus}$	-	$m \times d \times \sum_{j=0}^{\ell} \lceil \frac{n_i^j \times (n_o^j+1)}{n} \rceil$
$\mathsf{Eval}_{\ominus}$	-	$m \times d \times \sum_{j=0}^{\ell} \lceil \frac{n_i^j \times (n_o^j+1)}{n} \rceil$
Rotations	-	$m \times d \times \frac{1}{2} \sum_{j=0}^{\ell} (\lceil \frac{n_i^j \times (n_o^j+1)}{n} \rceil - 1)$
Ciphertexts exchanged for layer ℓ	$2(\ell+1)$	$m \times d \times 2 \sum_{j=0}^{\ell} (\lceil \frac{n_i^j \times (n_o^j+1)}{n} \rceil$

6.5 Conclusion

To conclude this chapter, we describe our protocol and how it works in a federated learning setting. We discuss how our gradient computation protocol combined with the secure aggregation protocol can compute the global gradient from the local gradients while securing the values of both the input and the model. Using the local gradient computation protocol, introduced in the previous chapter, it is possible to

efficiently compute the local gradients. The global model is constructed from the local gradient using the secure aggregation protocol which is run by the server to reconstruct the shares it holds along with the clients' shares. The analysis of the security of our protocol shows it is secure and protects both data privacy and model privacy while providing a stronger security guarantee.

Table 6.2: Complexity Comparison Between Frameworks

Operations	Naive	GAZELLE [52]	GALA [97]	Our Work
$\mathsf{Eval}_{\otimes}$	n_o	$\frac{n_i \times (n_o)}{n}$	$\frac{n_i \times (n_o)}{n}$	$\frac{n_i \times (n_o+1)}{n}$
$\mathsf{Eval}_{\oplus}$	$n_o \log_2 n_i$	$\log_2 \frac{n}{n_o} + \frac{n_i \times (n_o)}{n} - 1$	$\frac{n_i \times (n_o)}{n} - 1$	$\frac{n_i \times (n_o+1)}{n}$
Rotations	$n_o \log_2 n_i$	$\log_2 \frac{n}{n_o}$	$\frac{n_i \times (n_o)}{n} - 1$	$\frac{1}{2}(\frac{n_i \times (n_o+1)}{n} - 1)$

Chapter 7

Implementation and Evaluation of Our Protocols

In this chapter, we present the performance result of our protocols on different datasets along with the details on the experimental setup, implementation details, the libraries used to implement our protocols and the datasets used in the experiment.

7.1 Building Blocks for Implementation

In this section, we provide our systems specification and other implementation-related concepts.

7.1.1 System Specifications

For our implementation, we use the Microsoft SEAL library for HE in the linear layer and the EMP-toolkit for GC in the non-linear layer. Both libraries are written in C++. The implementation of our DNN training protocols is developed in C++ using SEAL and EMP-toolkit. The experimental setup is as follows:

- **OS.** Ubuntu 20.04.5 LTS
- **Architecture.** x86_64
- **Processor.** Intel(R) Core(TM) i7-10700 CPU @ 2.90 GHz
- **RAM.** 32GB
- **Platform.** Visual Studio 1.74.3
  - **C++ compiler.** g++
  - **C++ libraries.** Microsoft SEAL [5], EMP-toolkit (emp-sh_2pc)[2], EMP-toolkit (emp-ot)[1]

7.1.2 Processing Data for Neural Network Computation

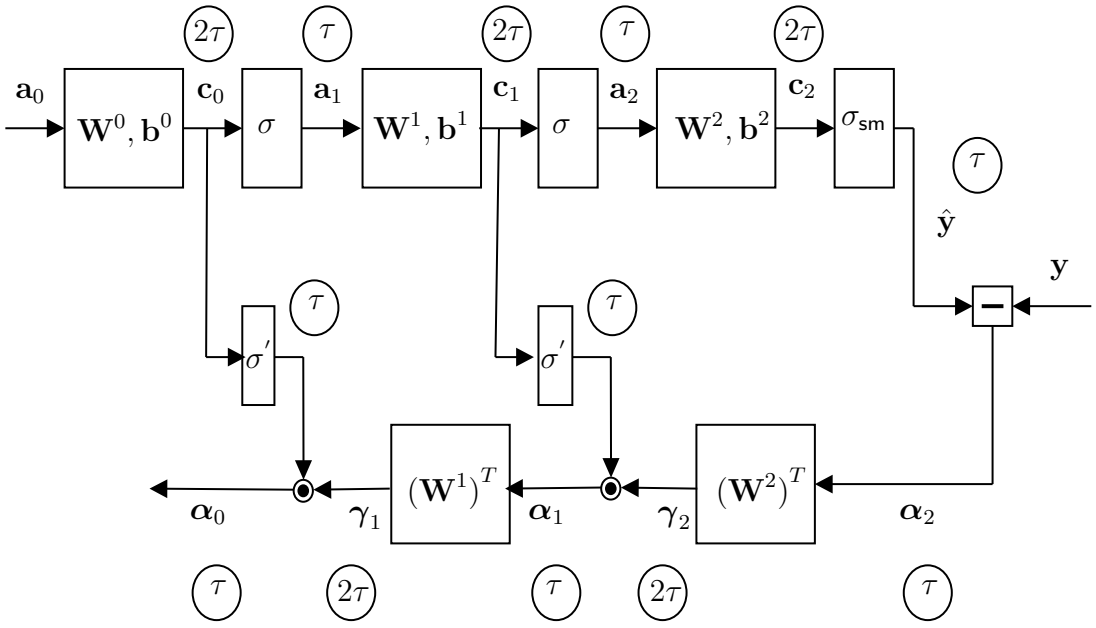

Figure 7.1: Propagation of Scaling over Neural Network Computation

7.1.2.1 Handling Floating-Point Numbers

Machine learning algorithms work over floating point numbers, whether it is input data or a model. These values can be either positive or negative. Since most cryp-

tographic schemes compute in the domain of finite rings, The values need to be converted to/from floating-point numbers. The fixed-point technique is used to convert a floating-point number to an integer. These conversion techniques (encoding and decoding) allow interoperability. The BFV scheme has a message space $\in \mathbb{Z}_p$ to and/or from which floating point values need to be converted where p is a prime number.

Encoding and Decoding. Similar to the technique in [61], the values are floating point numbers initially and require encoding before performing cryptographic operations on them. We follow the approach of dividing the message space into two halves where the positive half is represented in $[0, \frac{p}{2} - 1]$ and the negative numbers are represented in $[\frac{p}{2}, p - 1]$. This technique is used to convert every floating point value while preserving the precision. An overview is presented in Figure 7.1 Given an absolute floating point number x, the ring element corresponding to a ring element, represented by $\tilde{x} \in \mathbb{Z}_p$ is computed as $\tilde{x} = \mathsf{FE}(x, \tau) = \mathsf{round}(x \cdot 2^{\tau})$. If x is negative, the corresponding ring element as $\tilde{x} = p - \mathsf{FE}(x, \tau)$. Given $\tilde{x} \in \mathbb{Z}_p$, the decoding of $\tilde{x}$ is performed as $\frac{\tilde{x}}{2^{\tau}}$ or $-\frac{p-\tilde{x}}{2^{\tau}}$.

7.1.2.2 Normalization and Initialization

Normalization. Normalization [35] is a data pre-processing technique that is performed before feeding data into any machine learning algorithm. The process of normalizing data is scaling each value of a feature over all the other values in that feature within an admissible range to ensure every feature and data point has equal importance. Normalizing values in the dataset allows for more accurate models which further lead to better services. The 2 most commonly used normalization techniques are min-max normalization and z-score normalization.

- **Min-max Normalization.** The min-max normalization is used when the

data does not contain many outliers. If the data contains outliers, this would result in disrupted maximum or minimum values which would in turn lead to wrong calculations. The formula for min-max normalization where we have a data point x_i and in a set of data points containing x_{max} and x_{min} is given by:

$$x_{min-max} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

- **Z-score Normalization.** The Z-score normalization is most commonly used because it handles outliers well. The formula for z-score normalization given a standard deviation over the feature σ_{std}, and given the mean μ for a data point x is given by:

$$x_{z-score} = \frac{x_i - \mu}{\sigma_{std}}$$

Initialization. Initialization [27] is used to initialize the weight matrix or the model. Depending on the type of initialization, the model can have better or worse accuracy. The more tailored the heuristic, the more effective the model is. Random initialization initializes the weights to random floating point values in a given range and zero initialization initializes weights to zero both of which do not provide good results. Hence, the following ones are used and are explained below:

1. **Xavier Initialization.** The Xavier initialization was meant to keep the variances of the activations and gradients approximately constant during training. The formula for weight W_{ij}, a normal distribution $\mathcal{N}$ and number of inputs from the previous layer n_{in} is given by:

$$W_{ij} \sim \mathcal{N}\left(-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}\right)$$

2. **He Initialization.** He initialization is used for the ReLU activation function

and avoids magnifying magnitudes of inputs. The formula for weight W_{ij}, a normal distribution $\mathcal{N}$ and number of inputs from the previous layer n_{in} is given by:

$$W_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$$

7.1.3 Datasets

Datasets are of utmost importance to test or train ML algorithms. Datasets are a collection of multiple data points of similar features depending on which domain the data has been collected from. ML algorithms learn from these features and produce models that can be used to provide services related to the data the model was trained on.

To evaluate our protocol, we use real-world datasets which are popularly used in different works and are publicly available in [89] and [53]. The number of data points are represented by n, the number of features by d, and the number of output neurons by h_l. The two datasets we use are: (a) epileptic seizure recognition dataset (ESR) [3] with $n = 115000$, $d = 179$, $h_l = 2$ and (b) the hand-written dataset (MNIST) [40] with $n = 70000$, $d = 28 \times 28$ (784), $h_l = 10$.

7.1.4 Neural Network Architectures

We use various neural network architectures and fine-tune parameters depending on the architecture. For the ESR dataset, a two-layer fully connected NN with various numbers of neurons is used. We represent the number of neurons in the two-hidden layer neural network as (h_1, h_2) such as the (64, 64) from Poseidon's architecture [82] and (300,100) from LeNet's architecture [58]. In the case of a three-hidden layer neural network, the (64, 64, 64) from Poseidon's architecture is used on the MNIST dataset.

Key aspects such as SGD or MBGD, learning rate, batch size, number of iterations,

normalization and initialization techniques determine the accuracy of the models produced.

7.1.5 Cryptographic Libraries

7.1.5.1 Microsoft SEAL

Microsoft SEAL (Simple Encrypted Arithmetic Library) is a widely used open-source software library developed by Microsoft Research that enables secure homomorphic encryption. One of the most significant features of the library is its support for the BFV scheme, which is a type of FHE scheme. The BFV scheme allows arithmetic operations on encrypted data, such as addition and multiplication, which makes it ideal for various applications such as data analysis and machine learning. BFV scheme operations in Microsoft SEAL are designed to be efficient and scalable, which allows complex computations to be performed on large amounts of encrypted data. This is achieved through several techniques such as modulus switching and batching that optimize the computations and reduce the computational overhead. The efficiency of BFV scheme operations has contributed to the widespread use of homomorphic encryption for various privacy-sensitive applications which is available in [5] and [6]. The polynomial modulus is represented by N, the plaintext modulus is represented by T and a number of slots are represented by n which is the same as N. Using these representations, we denote the values used for our experimentation in Table 7.1.

Table 7.1: BFV Parameters for Our Protocol Experimental Setup

BFV params	Prime (bits)	N	T
Values	30	8192	1073692673

7.1.5.2 EMP-Toolkit

The EMP-toolkit is an open-source software library designed for implementing secure two-party (2PC) computations based on cryptographic techniques such as GC and OT. The toolkit is designed to provide developers with a platform to create secure applications that can perform computations on data owned by multiple parties without revealing the data to each other [92]. In our implementation, we use GC for the semi-honest case, i.e., emp-sh2pc package that implements the half-gate technique.

7.2 Experimental Results

We perform experiments on standard neural network architectures and use real-world datasets to obtain the accuracy of our trained models. We compare our results with that of Poseidon [82] which is the closest to our system model. We calculate the performance of our gradient computation protocol (Π_{GCP}) and then our efficient neural network training in federated learning protocol (Π_{PrivDNN}).

Datasets and NN Architecture. The two real-world datasets we consider are the MNIST dataset [40] and the ESR dataset [3]. For both datasets, we use Poseidon's neural network architecture ($h_1 = 64, h_2 = 64, h_3 = 64$ for MNIST dataset and $h_1 = 64, h_2 = 64$ for ESR dataset) and the standard LeNet-300-100 ($h_1 = 300, h_2 = 100$) fully connected neural network architecture [58], where the number of hidden nodes in each layer i is given by h_i.

Initialization and Normalization. For initialization of the weight matrices, Xavier's initialization technique is used (explained in Section 1) and for normalization of the input, the z-score normalization is used (explained in Section 7.1.2.2).

Activation Functions. For the sigmoid function, the approximation given in Eq. (3.1) is used while for its derivative Eq. (3.2) is used. For the softmax function, the approximation mentioned in [67] is explained in Section 3.3. Another alternative is the taylor series approximation from Eq. (3.4). The results in cleartext can be seen in Table 7.2 where we represent the original sigmoid, sigmoid derivative and softmax functions by $(\sigma, \sigma', \sigma_{\mathsf{sm}})$ (these accuracies are similar to that of vanilla models), the approximation functions are represented by $(\sigma_{\mathsf{apx}}, \sigma'_{\mathsf{apx}}, \sigma_{\mathsf{smapx}})$, and fixed-point approximation refers to computation using fixed-point values with approximated activation functions. For all the experiments, we choose the sigmoid activation function, a learning rate 0.01, use Xavier initialization and Z-score normalization and perform the training for one epoch. The experiments show the accuracy degradation from the original activation functions to the approximated activation functions and then fixed point encoded values for the approximation functions. Since the accuracy degradation is not drastic, the secure gradient computation protocol is a viable and secure solution.

Table 7.2: Experiments for Gradient Computation in Cleartext

Datasets	NN Architecture	Accuracies (%)		
	(Poseidon , LeNet)	$(\sigma, \sigma', \sigma_{\mathsf{sm}})$	$(\sigma_{\mathsf{apx}}, \sigma'_{\mathsf{apx}}, \sigma_{\mathsf{smapx}})$	Fixed-point Approximation Computation
MNIST, MBGD	$h_1 = 64$, $h_2 = 64$, $h_3 = 64$	92.00	91.20	90.52
	$h_1 = 300$, $h_2 = 100$	97.02	94.58	93.20
ESR, MBGD	$h_1 = 64$, $h_2 = 64$	92.73	91.30	90.65

7.3 The Experimental Evaluation of Our Secure Gradient Computation Protocol

In this section, our gradient computation protocol, Chapter 5, for DNN is presented. We consider the architectures of Poseidon-64-64-64 and LeNet-300-100 for MNIST dataset and Poseidon-64-64 for ESR dataset mentioned previously. We compute the

results for the execution time of a single epoch of the secure gradient computation protocol between a single client and the centralized server and present them.

Table 7.3: Execution Time for Secure Gradient Computation Protocol for a Single Client

Architectures	MNIST (in ms)		ESR (in ms)	
	Gradient Computation	Secure Aggregation	Gradient Computation	Secure Aggregation
Poseidon-64-64-64	265.00	4.20	-	-
Poseidon 64-64	225.50	4.00	112.00	1.50
LeNet-300-100	740.00	19.00	-	-

7.3.1 Protocol Execution Time

We present the execution time of the secure gradient computation protocol between client and server in Table 7.3. For example, for the LeNet-300-100 architecture, the gradient computation time between one client and the server would be 740 ms and the server has an added overhead of computing the secure aggregation for a single user, which would be 19 ms and this overhead on the server also includes other tasks such as conversion of floating point values to fixed-point values and vice-versa which is all included in the secure aggregation execution time. The execution time remains the same irrespective of the input provided since execution time is dependent on the architecture and not the input provided. In Figure 7.2, we show the time taken by the client and server to jointly compute the gradient for MNIST and ESR datasets for different neural network architectures.

7.4 Results on Secure Federated Training Protocol

In this section, we present the time taken by our secure FL training protocol in Chapter 6. The architectures of Poseidon and LeNet-300-100 for the MNIST dataset

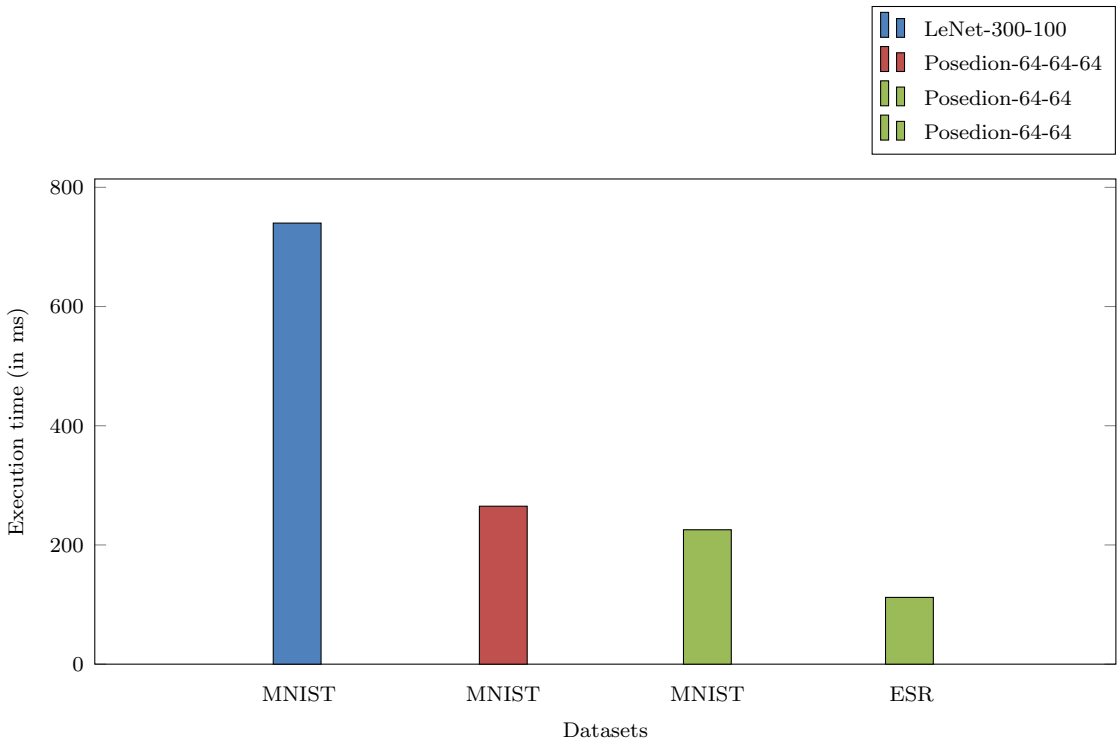

Figure 7.2: Execution Time of Secure Gradient Computation Protocol

and Poseidon-64-64 for the ESR dataset mentioned previously are considered. We compute the timings and present our results

7.4.1 Protocol Execution Time

Here, we present the time taken by our protocol to execute our neural network training operation along with the accuracy obtained. Table 7.4 shows the results we obtain compared to that of Poseidon [82]. This is the total time taken by the server and the clients to compute the global gradient. We consider 10 clients each holding an equal number of data points. When the experiment is performed on our desktop, the time taken to complete the execution of the protocol sequentially is computed. For example, the time taken by private federated training protocol is 4.48 hours for MNIST dataset where server and client computation are performed sequentially. However, when the server runs our two-party gradient computation

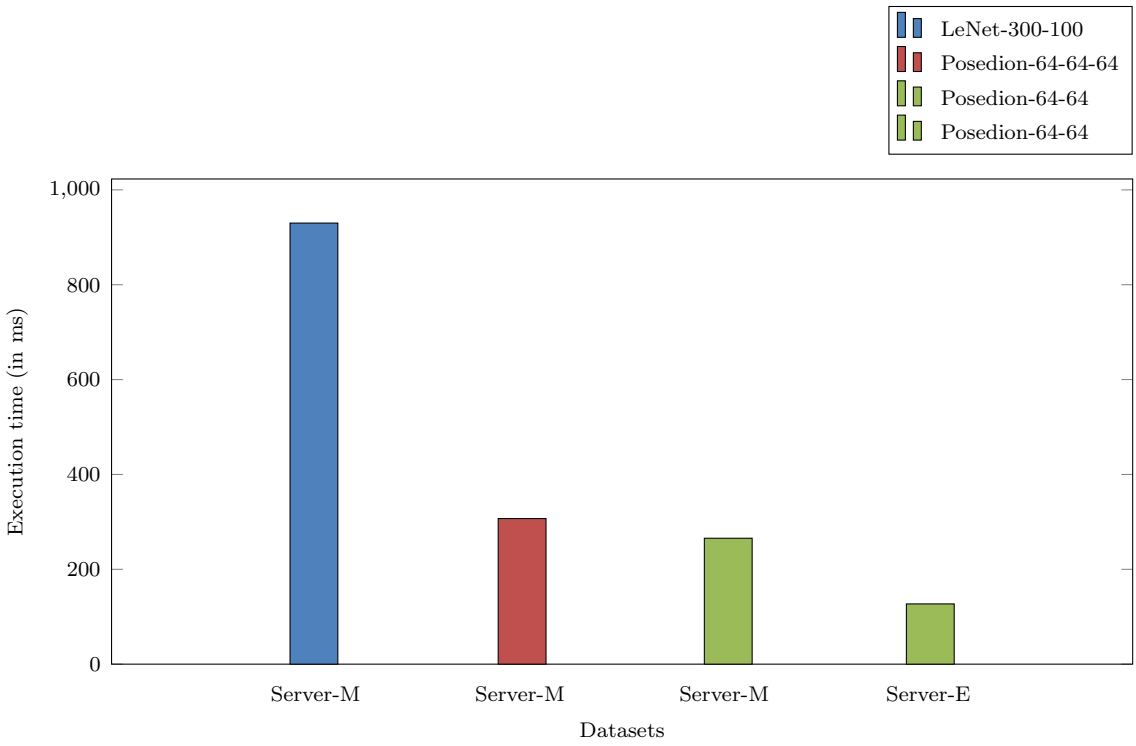

Figure 7.3: Execution Time of Secure Federated Training Protocol for 10 users for a Single Data Point

protocol parallelly, the time to execute our private federated training protocol is 44.8 minutes which is observably better.

Since the server performs the extra secure aggregation with other tasks in between the Figure 7.3 shows the time taken to complete the gradient computation combined with secure aggregation protocol for 10 users (Server-M in the figure represents the server for the MNIST dataset and Server-E represents the server for ESR dataset). The overhead on average is about 1.18x - 1.25x when compared to just the gradient computation.

Table 7.4: Experiments for Secure Federated Training

Datasets	NN Architecture	Poseidon [82]		Our Work	
	(From Poseidon)	Execution Time (in sec)	Accuracy(%)	Execution Time (in sec)	Accuracy(%)
MNIST	$h_1 = 64, h_2 = 64, h_3 = 64$	5283.10	89.90	1612.80	91.20
	$h_1 = 64, h_2 = 64$	-	-	1377.00	89.20
ESR	$h_1 = 64, h_2 = 64$	851.84	90.40	104.40	90.65

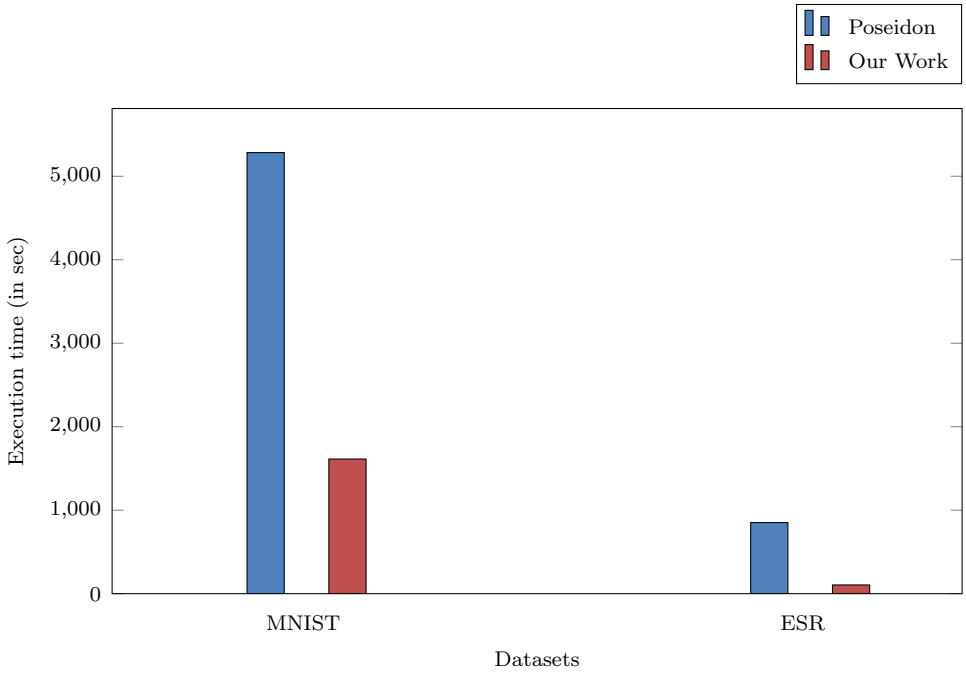

Figure 7.4: Comparison between Poseidon [82] and Our Work in Terms of Execution Time

Communication Cost. We compute the number of ciphertexts exchanged between the client and the server during the execution of our protocol. For example, if the tuple under client is (a, b, c, d), for each weight matrix $(\mathbf{W}^0, \mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3)$, the number of ciphertexts transferred between client and server is represented as, a ciphertexts for $\mathbf{W}^0$, b ciphertexts for $\mathbf{W}^1$, c ciphertexts, for $\mathbf{W}^2$ and d ciphertexts for $\mathbf{W}^3$ to the server. The same representation is followed for the server. The values a, b, c, d are the total number of ciphertexts in the forward propagation, backward propagation and gradient computation. The garbled circuit is also computed between the client and the server using shares. Each layer executes one round of garbled circuit for each layer in the forward and backward propagation. The communication cost is given in Table 7.5 for MNIST dataset and Table 7.6 for the ESR dataset.

Comparison. We provide a comparison in terms of the execution time of our protocol and Poseidon [82] and a system-level comparison is provided in Table 7.7,

Table 7.5: Ciphertexts Transferred for MNIST Dataset for Various NN Architectures

MNIST			No. of Ciphertexts	
NN Architecture	Hidden Neurons	$(\mathbf{W}^0, \mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3)$	Client	Server
Poseidon	$h_1 = 64$, $h_2 = 64$, $h_2 = 64$	(64,784), (64,64), (64,64), (10,64)	(4,4,4,4)	(4,2,2,2)
LeNet	h_1= 300, $h_2 = 100$	(300,784), (100,300), (10,100)	(4,4,4)	(30,4,4)

Table 7.6: Ciphertexts Transferred for ESR Dataset for Various NN Architectures

ESR			No. of Ciphertexts	
NN Architecture	Hidden Neurons	$(\mathbf{W}^0, \mathbf{W}^1, \mathbf{W}^2)$	Client	Server
Poseidon	$h_1 = 64$, $h_2 = 64$	(64,179), (64,64) (10,64)	(4,4,4)	(4,2,2)

where Priv. Infr.refers to private inference, (DC) refers to data confidentiality, A and P are active and passive adversarial capabilities. Finally, P' is our assumption of the adversarial models that are not mentioned. The training process for the MNIST dataset takes 1.46 hours whereas our protocol takes about 44.8 minutes and for the ESR dataset, it takes 14.40 minutes whereas our protocol takes 1.75 minutes (or 104.76 seconds).

7.5 Conclusion

In this chapter, we implement our protocol using C/C++ and Microsoft SEAL and EMP-toolkit library. We present our experimental results for the execution time of the federated training and the gradient computation protocol. We also provide the amount of data exchanged for this protocol and compare the results with Poseidon. Our experimental result demonstrates that our protocol is faster than the state-of-the-art protocol. When we compare communication overhead, our protocol has a greater overhead than Poseidon as our protocol offloads some work to the client and Poseidon outsources its computation and hence has less overhead.

Table 7.7: Comparision of Various Private Deep Learning Frameworks

	MiniONN [59]	SecureML [67]	Gazelle [52]	ABY3 [66]	SecureML [74]	Flash [29]	Trident [32]	CryptoNets [41]	Poseidon [82]	Eiffel [79]	Ours
MPC Setup	2PC	2PC	2PC	3PC	3PC	4PC	4PC	4PC	N-party	N-party	N-party
Priv. Infr.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
Priv. Train	✗	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓
Adversarial Model (DC)	1P	1A	1P	1A/P	1P	1A	1A/P'	1P'	1P'	N-1 A/P	N-1 P
Techniques	HE,GC,SS	GC,SS	HE,GC,SS	GS,SS	HE,GC,SS	SS	GC,SS	HE	HE,SS	AE, SS, SNIP	HE,GC,SS

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Although FL was introduced to secure the interaction between clients and a server, there are still some loopholes through which an adversary can learn about the data. FL has its advantages but due to the disadvantage where the model held by the server can be learned and used unlawfully, we build a protocol on top of this technique to secure both the input and the model held by the parties. We look at various related works and introduce important preliminaries needed to understand the protocol we propose. The BFV FHE scheme is used to securely compute the linear layer and the garbled circuit is used to securely compute the non-linear layer. Additive secret sharing is used to share values between parties so no secret value is revealed. Finally, our gradient computation and efficient federated training protocols are introduced along with their design, security, and computational complexity. We compare our results with prior works and highlight the differences and improvements in our work.

Contributions. In this thesis, to summarize our work, we propose an efficient privacy-preserving neural network federated training protocol through which the security and privacy of both the clients' input data and the server's model are pre-

served. The protocol is built on efficient gradient computation protocol and secure aggregation protocol. The protocol was based on the foundation of PrivFL [61] and our technique is based on the idea that the arrangement of the matrix can improve the efficiency of matrix-vector multiplication in the encrypted domain. Then, we construct an optimized garbled circuit to reduce the number of operations (sigmoid, sigmoid derivative) to reduce the complexity even further. The protocol security is analyzed along with its computation efficiency. Our protocol provides stronger security and performs computations efficiently to obtain a faster execution time. Finally, the efficient privacy-preserving neural network federated training protocol is built on top of the gradient computation protocol hence, making it an efficient and secure protocol. We introduce these protocols to protect the privacy of the input data as well as the model held by the server. By building this protocol on efficient computation building blocks we are also able to improve the speed of computation.

8.2 Future Work

PPML is a popular area of research and has multiple directions as well as various applications. Since the technique can be used for various purposes depending on the need and application, the amount of work done in this domain only seems to be increasing. In our current work, we only considered neural network architectures with fully-connected layers and run the experiments using CPU-efficient operations. Going further:

- These optimizations can be applied to convolution neural networks as well as other deep learning algorithms.
- It is also possible to integrate these protocols with a GPU since a GPU's computation capacity is much greater than a CPU as they did in [88].

By improving the security and efficiency of computation, real-world applications of

privacy-preserving machine learning can be implemented which makes it harder for adversaries to access data from sensitive interactions such as in the healthcare or finance domains and more.

Bibliography

[1] *Emp-toolkit, emp-ot*, https://github.com/emp-toolkit/emp-ot.

[2] *Emp-toolkit, emp-tool*, https://github.com/emp-toolkit/emp-sh2pc.

[3] *Epileptic seizure recognition dataset*, https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition.

[4] *Introduction to bfv encryption scheme*, https://inferati.com/blog/fhe-schemes-bfv.

[5] *Microsoft seal*, https://github.com/microsoft/SEAL.

[6] *Microsoft seal, bfv examples*, https://github.com/microsoft/SEAL/blob/main/native/examples/1_bfv_basics.cpp.

[7] *What is gradient descent?*, https://www.ibm.com/topics/gradient-descent.

[8] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad, *State-of-the-art in artificial neural network applications: A survey*, Heliyon **4** (2018), no. 11, e00938.

[9] Mohammad Al-Rubaie and J. Morris Chang, *Privacy-preserving machine learning: Threats and solutions*, IEEE Security & Privacy **17** (2019), no. 2, 49–58.

[10] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan, *Homomorphic encryption security standard*, Tech. report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[11] David Balbás, *The hardness of lwe and ring-lwe: A survey*, Cryptology ePrint Archive, Paper 2021/1358, 2021, `https://eprint.iacr.org/2021/1358`.

[12] Kunal Banerjee, Vishak Prasad C., Rishi Raj Gupta, Karthik Vyas, Anushree H., and Biswajit Mishra, *Exploring alternatives to softmax function*, CoRR **abs/2011.11538** (2020).

[13] M Banoula, *Machine Learning Steps: A Complete Guide!*, `https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-steps`, 2023.

[14] B Barrett, *This year alexa grew up*, `https://www.wired.com/story/amazon-alexa-2018-machine-learning/`, 2018.

[15] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway, *Foundations of garbled circuits*, Proceedings of the 2012 ACM Conference on Computer and Communications Security (New York, NY, USA), CCS '12, Association for Computing Machinery, 2012, p. 784–796.

[16] Subrato Bharati, M. Rubaiyat Mondal, Prajoy Podder, and Surya Prasath, *Federated learning: Applications, challenges and future directions*, International Journal of Hybrid Intelligent Systems **18** (2017), 19–35.

[17] A Biswal, *What is Exploratory Data Analysis? Steps and Market Analysis*, `https://www.simplilearn.com/tutorials/data-analytics-tutorial/exploratory-data-analysis`, 2023.

[18] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame, *Mp2ml: A mixed-protocol machine learning framework for private inference*, Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice (New York, NY, USA), PPMLP'20, Association for Computing Machinery, 2020, p. 43–45.

[19] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski, *ngraph-he: a graph compiler for deep learning on homomorphically encrypted data*, Proceedings of the 16th ACM International Conference on Computing Frontiers, 2019, pp. 3–13.

[20] Gaudenz Boesch, *Privacy-preserving deep learning for private computer vision (2023)*, `https://viso.ai/deep-learning/privacy-preserving-deep-learning-for-computer-vision/`.

[21] ———, *What is adversarial machine learning? attack methods in 2023)*, `https://viso.ai/deep-learning/adversarial-machine-learning/`.

[22] K. A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth, *Practical secure aggregation for federated learning on user-held data*, NIPS Workshop on Private Multi-Party Machine Learning, 2016.

[23] 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*, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '17, Association for Computing Machinery, 2017, p. 1175–1191.

[24] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, *(leveled) fully homomorphic encryption without bootstrapping*, Electronic Colloquium on Computational Complexity (ECCC) **18** (2011), 111.

[25] Zvika Brakerski and Vinod Vaikuntanathan, *Fully homomorphic encryption from ring-lwe and security for key dependent messages*, Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Lecture Notes in Computer Science, vol. 6841, Springer, 2011, p. 501.

[26] Cláudia Brito, Pedro Ferreira, Bernardo Portela, Rui Oliveira, and João Paulo, *Soteria: Privacy-preserving machine learning for apache spark*, Cryptology ePrint Archive, Paper 2021/966, 2021.

[27] Jason Brownlee, *Weight initialization for deep learning neural networks)*, `https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/`.

[28] Niklas Büscher, Spyros Boukoros, Stefan Bauregger, and Stefan Katzenbeisser, *Two is not enough: Privacy assessment of aggregation schemes in smart metering*, Proc. Priv. Enhancing Technol. **2017** (2017), no. 4, 198–214.

[29] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh, *Flash: Fast and robust framework for privacy-preserving machine learning*, Proceedings on Privacy Enhancing Technologies **2020** (2020), 459–480.

[30] Yifei Cai, Qiao Zhang, Rui Ning, Chunsheng Xin, and Hongyi Wu, *Hunter: He-friendly structured pruning for efficient privacy-preserving deep learning*, Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (New York, NY, USA), ASIA CCS '22, Association for Computing Machinery, 2022, p. 931–945.

[31] M Chatterjee, *Top 20 Applications of Deep Learning in 2022 Across Industries*, https://www.mygreatlearning.com/blog/deep-learning-applications/, 2022.

[32] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh, *Trident: Efficient 4pc framework for privacy preserving machine learning*, 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020, The Internet Society, 2020.

[33] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song, *Homomorphic encryption for arithmetic of approximate numbers*, Advances in Cryptology – ASIACRYPT 2017 (Cham) (Tsuyoshi Takagi and Thomas Peyrin, eds.), Springer International Publishing, 2017, pp. 409–437.

[34] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, *Tfhe: Fast fully homomorphic encryption over the torus*, Cryptology ePrint Archive, Paper 2018/421, 2018, https://eprint.iacr.org/2018/421.

[35] CodeAcademyTeam, *Normalization*, https://codecademy.com/article/normalization.

[36] Emiliano De Cristofaro, *An overview of privacy in machine learning*, ArXiv **abs/2005.08679** (2020).

[37] Vitor Falcão da Rocha and Julio López, *An overview on homomorphic encryption algorithms*, 2019.

[38] Vishnu Asutosh Dasu, Sumanta Sarkar, and Kalikinkar Mandal, *Prov-fl : privacy-preserving round optimal verifiable federated learning*, The 15th ACM Workshop on Artificial Intelligence and Security (AISec 2022), ACM, 2022, © ACM, YYYY. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The

definitive version was published in PUBLICATION, {VOL#, ISS#, (DATE)} http://doi.acm.org/10.1145/nnnnnn.nnnnnn.

[39] Daniel Demmler, Thomas Schneider, and Michael Zohner, *Aby - a framework for efficient mixed-protocol secure two-party computation*, 01 2015.

[40] Li Deng, *The mnist database of handwritten digit images for machine learning research*, IEEE Signal Processing Magazine **29** (2012), no. 6, 141–142.

[41] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing, *Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy*, Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, JMLR.org, 2016, p. 201–210.

[42] Salim Dridi, *Unsupervised learning - a systematic literature review*, 12 2021.

[43] Shimon Even, Oded Goldreich, and Abraham Lempel, *A randomized protocol for signing contracts*, Commun. ACM **28** (1985), no. 6, 637–647.

[44] Matthew K. Franklin and Michael K. Reiter, *Fair exchange with a semi-trusted third party (extended abstract)*, Proceedings of the 4th ACM Conference on Computer and Communications Security (New York, NY, USA), CCS '97, Association for Computing Machinery, 1997, p. 1–5.

[45] David Froelicher, Juan R Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat, and Jean-Pierre Hubaux, *Scalable privacy-preserving distributed learning*, Proceedings on Privacy Enhancing Technologies **2021** (2021), no. 2, 323–347.

[46] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich, *Reusable garbled circuits and succinct functional encryption*,

Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '13, Association for Computing Machinery, 2013, p. 555–564.

[47] D Hassabis, *AlphaGo: using machine learning to master the ancient game of Go*, (2016).

[48] Wilko Henecka, Stefan K ögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg, *Tasty: tool for automating secure two-party computations*, Proceedings of the 17th ACM conference on Computer and communications security, 2010, pp. 451–462.

[49] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding, *Cheetah: Lean and fast secure Two-Party deep neural network inference*, 31st USENIX Security Symposium (USENIX Security 22) (Boston, MA), USENIX Association, August 2022, pp. 809–826.

[50] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank, *Extending oblivious transfers efficiently*, Advances in Cryptology - CRYPTO 2003 (Berlin, Heidelberg) (Dan Boneh, ed.), Springer Berlin Heidelberg, 2003, pp. 145–161.

[51] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song, *Secure outsourced matrix computation and application to neural networks*, Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '18, Association for Computing Machinery, 2018, p. 1209–1222.

[52] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan, *GAZELLE: A low latency framework for secure neural network inference*, 27th USENIX Security Symposium (USENIX Security 18) (Baltimore, MD), USENIX Association, August 2018, pp. 1651–1669.

[53] Kaggle, *Kaggle Datasets*, https://www.kaggle.com/datasets, 2010.

[54] Daniel Kales, *Secret sharing*, https://www.iaik.tugraz.at/wp-content/uploads/teaching/mfc/secret_sharing.pdf.

[55] Seny Kamara and Lei Wei, *Garbled circuits via structured encryption*, Financial Cryptography and Data Security (Berlin, Heidelberg) (Andrew A. Adams, Michael Brenner, and Matthew Smith, eds.), Springer Berlin Heidelberg, 2013, pp. 177–188.

[56] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh, *SWIFT: superfast and robust privacy-preserving machine learning*, IACR Cryptol. ePrint Arch. (2020), 592.

[57] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma, *Cryptflow: Secure tensorflow inference*, IEEE Symposium on Security and Privacy, IEEE, May 2020.

[58] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.

[59] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan, *Oblivious neural network predictions via minionn transformations*, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '17, Association for Computing Machinery, 2017, p. 619–631.

[60] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi, *A survey of deep neural network architectures and their applications*, Neurocomputing **234** (2017), 11–26.

[61] Kalikinkar Mandal and Guang Gong, *Privfl: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks*, Cryptology ePrint Archive, Paper 2019/979, 2019, https://eprint.iacr.org/2019/979.

[62] Kalikinkar Mandal, Guang Gong, and Chuyi Liu, *Nike-based fast privacy-preserving highdimensional data aggregation for mobile devices*, IEEE T Depend Secure (2018), 142–149.

[63] B Marr, *How Tesla Is Using Artificial Intelligence to Create The Autonomous Cars Of The Future.*

[64] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas, *Federated learning of deep networks using model averaging*, CoRR **abs/1602.05629** (2016).

[65] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa, *Delphi: A cryptographic inference system for neural networks*, Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice (New York, NY, USA), PPMLP'20, Association for Computing Machinery, 2020, p. 27–30.

[66] Payman Mohassel and Peter Rindal, *Aby3: A mixed protocol framework for machine learning*, Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '18, Association for Computing Machinery, 2018, p. 35–52.

[67] Payman Mohassel and Yupeng Zhang, *Secureml: A system for scalable privacy-preserving machine learning*, 05 2017, pp. 19–38.

[68] Kundan Munjal and Rekha Bhatia, *A systematic review of homomorphic encryption and its contributions in healthcare industry*, Complex & Intelligent Systems (2022).

[69] J Navas, *What is hyperparameter tuning?*, `https://www.anyscale.com/blog/what-is-hyperparameter-tuning`, 2022.

[70] Lucien K. L. Ng and Sherman S. M. Chow, *GForce: GPU-Friendly oblivious and rapid neural network inference*, 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, August 2021, pp. 2147–2164.

[71] D Nikolaiev, *Overfitting and Underfitting Principles*, `https://towardsdatascience.com`, 2021.

[72] E Onose, *Machine Learning as a Service: What It Is, When to Use It and What Are the Best Tools Out There*, `https://neptune.ai/blog`, 2023.

[73] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame, *ABY2.0: Improved Mixed-Protocol secure Two-Party computation*, 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, August 2021, pp. 2165–2182.

[74] Arpita Patra and Ajith Suresh, *BLAZE: blazing fast privacy-preserving machine learning*, 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020, The Internet Society, 2020.

[75] Michael O. Rabin, *How to exchange secrets with oblivious transfer*, Cryptology ePrint Archive, Paper 2005/187, 2005, `https://eprint.iacr.org/2005/187`.

[76] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma, *Cryptflow2: Practical 2-party secure inference*, Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '20, Association for Computing Machinery, 2020, p. 325–342.

[77] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar, *Xonn: Xnor-based oblivious deep neural network in-*

ference, Proceedings of the 28th USENIX Conference on Security Symposium (USA), SEC'19, USENIX Association, 2019, p. 1501–1518.

[78] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar, *Deepsecure: Scalable provably-secure deep learning*, Proceedings of the 55th Annual Design Automation Conference (New York, NY, USA), DAC '18, Association for Computing Machinery, 2018.

[79] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten, *Eiffel: Ensuring integrity for federated learning*, Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '22, Association for Computing Machinery, 2022, p. 2535–2549.

[80] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, CoRR **abs/1609.04747** (2016).

[81] Iqbal H" "Sarker, *"machine learning: Algorithms, real-world applications and research directions"*, "SN Comput. Sci." **2** (2021), no. 3, 160 (en).

[82] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux, *POSEIDON: privacy-preserving federated neural network learning*, 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021, The Internet Society, 2021.

[83] Nida Shahid, Tim Rappon, and Whitney Berta, *Applications of artificial neural networks in health care organizational decision-making: A scoping review*, PLoS One **14** (2019), no. 2, e0212356 (en).

[84] Adi Shamir, *How to share a secret*, Commun. ACM **22** (1979), no. 11, 612–613.

[85] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma, *A review of supervised machine learning algorithms*, 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 1310–1315.

[86] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near, *Efficient differentially private secure aggregation for federated learning via hardness of learning with errors*, 31st USENIX Security Symposium (USENIX Security 22) (Boston, MA), USENIX Association, August 2022, pp. 1379–1395.

[87] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal, *Introduction to multi-layer feed-forward neural networks*, Chemometrics and Intelligent Laboratory Systems **39** (1997), no. 1, 43–62.

[88] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu, *Cryptgpu: Fast privacy-preserving machine learning on the gpu*, 2021 IEEE Symposium on Security and Privacy (SP) (2021), 1021–1038.

[89] UCI, *UCI Machine Learning Repository*, `https://archive.ics.uci.edu/ml/index.php`, 1987.

[90] Sameer Wagh, Divya Gupta, and Nishanth Chandran, *Securenn: 3-party secure computation for neural network training*, Proceedings on Privacy Enhancing Technologies **2019** (2019), 26 – 49.

[91] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin, *Falcon: Honest-majority maliciously secure framework for private deep learning*, arXiv preprint arXiv:2004.02229 (2020).

[92] Xiao Shaun Wang, *A new paradigm for practical maliciously secure multi-party computation*, 2018.

[93] Xin Wang, Liting Yan, and Qizhi Zhang, *Research on the application of gradient descent algorithm in machine learning*, 2021 International Conference on Computer Network, Electronic and Automation (ICCNEA), 2021, pp. 11–15.

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

[95] Sophia Yakoubov, *A gentle introduction to yao ' s garbled circuits*, 2017.

[96] Andrew Chi-Chih Yao, *How to generate and exchange secrets*, 27th Annual Symposium on Foundations of Computer Science (sfcs 1986) (1986), 162–167.

[97] Qiao Zhang, Chunsheng Xin, and Hongyi Wu, *Gala: Greedy computation for linear algebra in privacy-preserved neural networks*, 01 2021.

Vita

Candidate's full name: Gaurav Vinay Uttarkar
University attended (with dates and degrees obtained):

- Bachelor of Engineering in Computer Science, Vidyavardhaka College of Engineering, 2016-2020

Publications:

- Shabnam Saderi, Oyonika Samazder, Gaurav Uttarkar, and Kalikinkar Mandal. "Efficient Security Protocols for Scalable Data Collection in the Smart Grid". To be submitted soon.
- Gaurav Uttarkar and Kalikinkar Mandal. "PrivDNN: Efficient Privacy preserving Deep Neural Network Training Protocol in Federated Learning". In preparation and to be submitted soon.

Poster Presentations:

- Gaurav Uttarkar, Oyonika Samazder, Shabnam Saderi, Kalikinkar Mandal. "ZTSGrid: A Zero-trust Approach for the SCADA System in the Smart Grid". 19th Annual International Conference on Privacy, Security and Trust (PST), Fredericton, New Brunswick, Canada. August 22-24, 2022.