Make LLMs Private

Sayan Pradhan

INDIAN STATISTICAL INSTITUTE, KOLKATA
R. C. BOSE CENTRE FOR CRYPTOLOGY AND SECURITY

KATHOLIEKE UNIVERSITEIT LEUVEN
COMPUTER SECURITY AND INDUSTRIAL CRYPTOGRAPHY

Master's Thesis

By Sayan Pradhan (CrS2212)

Title : Make LLMs Private

Promotor : Prof. Dr. Bart Preneel
Internal Supervisor : Prof. Dr. Bimal Kumar Roy
Daily Supervisors: Martin Zbudila, Dr. Aysajan Abidin

CERTIFICATE

This is to certify that the dissertation entitled **"Make LLMs Private"** submitted by **Sayan Pradhan** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Cryptology and Security** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Bart Preneel
Professor,
COSIC
Katholieke Universiteit Leuven,
Leuven, BELGIUM.

Bimal Kumar Roy
Professor,
Cryptology and Security Research Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would like to show my highest gratitude to my advisors, Prof. Dr. Ir. Bart Preneel and Prof. Dr. Bimal Kumar Roy for their guidance and continuous support and encouragement.

I would also like to thank my daily supervisors, Martin Zbudila and Dr. Aysajan Abidin, for their precious suggestions and discussions. Thay have taught me how to do research and motivated me with excellent insights and innovative ideas.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

Finally, I am very much thankful to my parents and family for their everlasting supports.

Last but not the least, I would like to thank all of my friends for their help and support. I thank all those, whom I have missed out from the above list.

Sayan Pradhan

Abstract

Large Language Models are Machine Learning models that are trained on large text data, and have the capability to understand and generate human languages. Cryptographic techniques can be able to protect both the input and output privacy of the users. Secure Multi-Party Computation (MPC) is a cryptographic technique that allows multiple parties to compute a function on their secret inputs without revealing any information about their secret inputs. One of the applications of MPC lies in the domain of privacy-preserving machine learning (PPML). In this project "Make LLMs Private", our goal is to choose a large language model and use MPC to make it private. Here we choose Bert model, which is a large language model and using different MPC protocols we construct an efficient combined MPC framework to evaluate Bert. Here our focus is only on the inference of Bert model. To achieve this, we require MPC protocols for multiplication, comparison, truncation, softmax, division, normalization operations. There are different MPC protocols for multiplication, division, comparison operations in ABY^3 [10], Falcon [17] between three parties. Function secret sharing based protocols have smaller communication costs but larger computation costs. In Sigma [6], they introduce different MPC protocols for multiplication, comparison, exponential, truncation, maximum using function secret sharing in two parties. In a 3-party setup, MPC becomes increasingly faster, enabling practical deployment even for larger Machine Learning models. In MPC computing exponential is very expensive. In SecureML [12], the exponential is replaced by relu in an efficient way, and papers Puma [5], Bolt [13], approximate the exponential by polynomials. In Bicoptor [19], they introduce 2 round MPC protocols for multiplication, comparison, maximum with no pre-processing. Here we use those techniques to protect the Bert model. We integrate various basic MPC protocols to construct MPC protocols for complex functions, such as the softmax function, and optimize for both the number of rounds and communication costs. We propose combined MPC frameworks tailored for different scenarios in the Bert model, involving three parties.

Contents

List of Tables

Chapter 1

Introduction

In this project, our goal is to make the Large Language Models private. A Large Language Model is a deep learning model that is trained on large text data so that it can be capable enough to understand and generate human language. The project title is "Make LLMs Private". But the question that comes to mind is why we need to do so. The reason behind this is that using LLMs like ChatGPT, users send their queries to the server, which takes that query as input and executes this to return the answer. However, this process exposes both the query and the answer to the LLM server, so it compromises the privacy of the user. For example, if a user might want to use ChatGPT for spelling correction or grammatical error correction on a research paper that is not yet published, it should not be considered a safe platform to use.

There are already many works on privacy-preserving machine learning and one of the applications of MPC lies in this domain. One way to make the LLM private is using an MPC framework. Suppose the LLM server distributes its trained parameters among n computing parties using some secret sharing scheme and there exists some MPC framework to execute that LLM model. When any user wants to use that LLM, instead of sending its query directly to the server it just breaks its query into n parts using some secret sharing and then sends each part of its query to each of the computing parties. After getting a partial query from the user then the computing parties execute some MPC framework to execute that LLM model and each party finally outputs some value and sends it back to the user. After getting outputs from each of the computing parties then the user just combines them to get the final output. This is a way to make LLMs private. There are many papers on ML models like linear regression, logistic regression, neural networks and using MPC frameworks, they make it possible to train and execute these ML models privately.

In this project, we examine the Bert model, which is a large language model that means it is trained on large text data so that it can be capable enough to understand and generate human language. Bert is a stack of Transformer Encoders. Encoder is a

Deep Learning model and it has four layers. The first layer is a multi-head attention layer, the second layer is add and normalization, the third layer is a two-layer neural network with Relu as an activation function and the fourth layer is another add and normalization layer. The Bert Base model is a stack of 12 encoders and Bert Large model is a stack of 24 encoders. Our goal is to make the Bert model private using MPC protocols.
In these papers SecureML [12], ABY^3 [10], Falcon [17], they introduce MPC protocols for various operations. In these papers Sigma [6], Puma [5], Bolt [13], there are works on making the transformer-based model private. In the Bert model, the softmax function is used, and executing it with MPC is quite challenging because it is significantly more computationally expensive. In SecureML [12], they replace exponential with ReLu and then compute softmax but they use it for classification purposes. There are papers Puma [5], Bolt [13], that approximate the softmax function using polynomials. In Sigma [6], they use function secret sharing to execute different operations in a shared setup, these protocols require less communication in the online phase but have larger computation costs.
We analyze the communication costs and number of rounds required for both the online and offline phases for executing different operations required in the Bert model and then combine those techniques for executing the Bert model using different MPC protocols.

1.1 Motivation

Bert is one of the most widely used LLMs due to its state-of-the-art performance on various natural language processing tasks. Its versatility in handling tasks such as text classification, question answering, and named entity recognition makes it an ideal candidate for demonstrating privacy-preserving techniques. Bert's extensive use in real-world applications, such as search engines, customer service chatbots, and content recommendation systems, underscores the importance of ensuring privacy in its computations. Protecting user data while maintaining Bert's high performance is crucial for practical deployment. There are papers Puma [5], Bolt [13], that provide different MPC protocols for privately executing transformer-based models. In Sigma [6], they introduce protocols for GPT inference models using the function secret sharing scheme. Training large language models requires a vast amount of data, making the process significantly more expensive in a shared setup. Therefore, our focus here is solely on the inference of the Bert model. In this thesis, we analyze the communication costs and the number of rounds required to perform various operations across different frameworks. We then integrate multiple MPC frameworks to run the Bert model, utilizing conversions between 2-party and 3-party protocols within the same framework to achieve optimal runtime in specific scenarios.

1.2 Our Contribution

In this thesis, our contributions are summarized as follows. We analyze various MPC frameworks, determining the communication costs and rounds required for each protocol. Then we combine these protocols to construct an efficient framework for the Bert model, identifying which protocols are best suited for specific scenarios. To optimize the frameworks, we alternate between 2-party and 3-party protocols, utilizing replicated secret sharing to enable conversion from 2-out-of-3 sharing to secret sharing between two parties. FSS-based protocols, which provide lower communication costs but demand substantial computational power, are selectively used to enhance the online phase when adequate computational resources are available. The only drawback is their offline phase, which incurs high communication costs and significant computational power requirements. By using efficient basic protocols, we build complex frameworks and correct some communication costs.

1.3 Thesis Outline

- In Chapter 2, we cover the prerequisites for this thesis. We provide a brief overview of various cryptographic primitives, the architecture of the Bert model, and fixed-point arithmetic.
- In Chapter 3, we explore different MPC protocols from various MPC frameworks that facilitate executing Bert in a shared setup.
- In Chapter 4, we propose four combined MPC frameworks tailored to different scenarios and provide a theoretical cost analysis.

Chapter 2

Preliminaries

2.1 Notations

In this section, we clarify all the notations used in the thesis. Here we use $(i+1)$ to represent the next party and $(i-1)$ to represent the previous party. For example, in a 3-party scenario, $(3+1)$ refers to the 1st party and $(1-1)$ refers to the 3rd party. We use $\langle x \rangle^A = \{x_1, x_2\}$ to denote the 2-out-of-2 arithmetic sharing of x, where $x = x_1 + x_2$, with each of the two parties holding one share. Similarly, $\langle x \rangle^B = \{x_1, x_2\}$ denotes the 2-out-of-2 binary sharing of x, where $x = x_1 \oplus x_2$, with each of the two parties holding one share. We also use $[x]^A = \{x_1, x_2, x_3\}$ to denote the 2-out-of-3 arithmetic sharing of x, where $x = x_1 + x_2 + x_3$, with each of the three parties holding two shares. Similarly, $[x]^B = \{x_1, x_2, x_3\}$ denotes the 2-out-of-3 binary sharing of x, where $x = x_1 \oplus x_2 \oplus x_3$, with each of the three parties holding two shares.

2.2 Cryptographic Primitives

2.2.1 Multi-Party Computation (MPC)

Multi-party computation is a cryptographic scheme that enables multiple parties to calculate a public function on their secret inputs without revealing any information about their private input. This means f is a public function of n variables, with n parties, where party-i holds x_i as a secret input, then MPC allows parties to compute $f(x_1, ..., x_n)$ without revealing any information about their secret input.

2.2.2 Secret Sharing

Replicated Secret Sharing

Let x be a secret value in the finite ring $\mathbb{Z}_{2^k}$. Now take x_1, x_2 randomly from $\mathbb{Z}_{2^k}$ and set $x_3 = x - x_1 - x_2$. Let there be 3 parties and the first party gets x_1, x_2; the second

party gets x_2, x_3 and the third party gets x_3, x_1. That means each party has two shares of the secret value x, this is known as *replicated secret sharing* or *2-out-of-3 sharing*.

Function Secret Sharing

Function secret sharing scheme is a pair of algorithms, in which one algorithm is used for key generation and another is used for evaluation. Let f be a secret function, then FSS protocol breaks this secret function into two parts f_1, f_2 and gives f_i to party-i. Now, for any point x, each party can evaluate the partial function on the point x and party-i gets $f_i(x)$ and the relation $f_1(x) + f_2(x) = f(x)$ holds for all x. The functionality of function secret sharing is given in Figure 2.1.

Definition 1 *(Function Secret Sharing (FSS) [6]) A two-party function secret sharing scheme is a pair of algorithms (GEN, EVAL) where*

- *GEN is a probabilistic polynomial time algorithm, it takes the secret function and security parameter as input and gives two keys k_0, k_1.*
- *EVAL is the other probabilistic polynomial time algorithm, it takes key and a point x as input and outputs y_b*

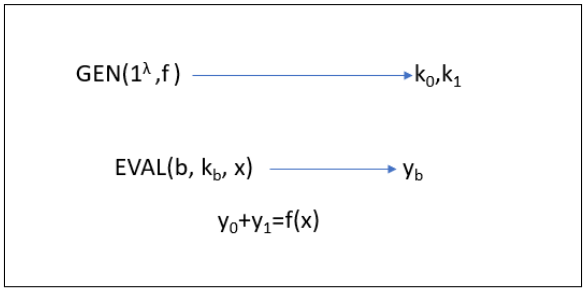

Figure 2.1: Functionality 2-Party FSS

2.2.3 Distributed Point Function

A point function is a function $f_{\alpha,\beta} : X \to Y$ such that

$$f_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases} .$$

Any Function Secret Sharing protocol $(Gen_k^*, Eval_k^*)$ for a point function is known as a Distributed Point Function (DPF). Using PRG a construction of DPF is introduced by Boyle et al. [1].

2.2.4 Distributed Comparison Function

Any comparison function can be written as $f^{<}_{\alpha,\beta} : X \to Y$ such that

$$f^{<}_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x < \alpha \\ 0 & \text{otherwise} \end{cases} .$$

Similar to DPF, any Function Secret Sharing protocol $(Gen^{<}_{k}, Eval^{<}_{k})$ for a comparison function is known as a Distributed Comparison Function (DCF). Using DPF a construction of DCF is introduced by Boyle et al. [1].

2.2.5 Oblivious Transfer

An *oblivious transfer* is a protocol between a sender and a receiver. In this protocol, the sender has two messages m_0, m_1 and the receiver has a selection bit $b \in \{0, 1\}$ and it is such that after execution of this protocol, the receiver gets the message m_b without knowing about any information about the other message and the sender don't get any information about the selection bit b. The functionality of OT is given in Figure 2.2.

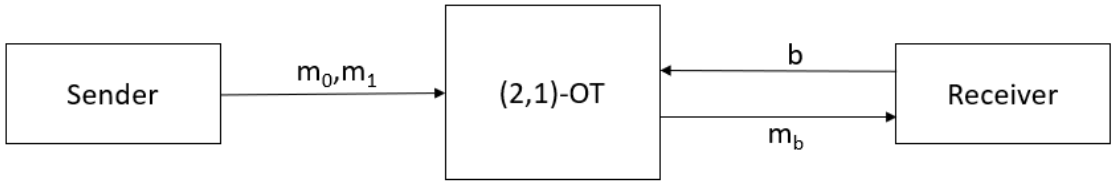

Figure 2.2: Functionality of a (2,1)-OT

2.2.6 Garbled Circuit(GC) 2PC

Garbled Circuits are first introduced by Yao [18]. Using 2PC GC two parties can evaluate a boolean circuit with their private inputs and do not learn any information about others inputs. One party is called the garbler and the other is called the evaluator. Garbler, with input x, and Evaluator with input y, have a circuit C. The protocol is as follows

- The Garbler encrypts the circuit C to $\hat{C}$ and encrypts its private input x to $\hat{x}$
- Then it sends the garbled circuit $\hat{C}$ and encrypted private input $\hat{x}$ to the Evaluator

- To evaluate $\hat{C}$, Evaluator also needs to encrypt its input y. Using OT, the Evaluator gets the corresponding encrypted $\hat{y}$ for its private input y
- Evaluator evaluates the circuit $\hat{C}$ and gets the encrypted output
- Both parties communicate to learn the output

2.2.7 Garbled Circuit 3-PC

In the paper by Mohassel et al. [11], they first extend Yao's Garbled Circuit protocol into a three-party setting with one corrupt party. In this protocol, two parties work as garblers and one party works as an evaluator. The high-level idea of this protocol is basically the two garblers exchange a random seed, which is used to generate all the randomness used in this protocol, and the two garblers separately generate the garbled circuit and then send their copy to the evaluator. Here one party is corrupt, therefore at least one of the garbled circuits is always correct. The Evaluator can also check honest garbling behavior by checking equality of these two garbled circuits.

2.3 Large Language Model

Nowadays LLMs are very popular tools and have a lot of use everywhere, such as creating human-like text for content creation, storytelling, converting text from one language to another, rewriting text in different words while preserving the original meaning, automating responses to customer inquiries, providing instant support and improving user experience, and other creative writing tasks. They are basically deep learning models and these models are trained on large text data, so that have the capability to understand and generate human languages. LLMs belong to the field of Natural Language Processing, an intriguing area that explores how humans and computers can communicate using Artificial Intelligence. LLMs have large number of training parameters and also large training data, so LLMs are able to capture the context from complex sequences of texts and these large scales make LLMs much more accurate. The training process helps LLMs to understand grammar, contextual meaning, some common sense knowledge. LLMs can be used for different natural language processing tasks such as text generation, question-answering, translation, sentiment analysis etc.

2.3.1 Transformer

Transformer is a deep-learning model and is introduced in the paper "Attention Is All You Need" by Vaswani et al. [15]. It consists of two parts, encoder and decoder. Here first each word is converted into a vector, called a token, and then positional embedding is added to make the tokens contextual. Both the encoder and decoder

consist of attention layer and feed-forward network layer. The architecture of the transformer is given in Figure 2.3.

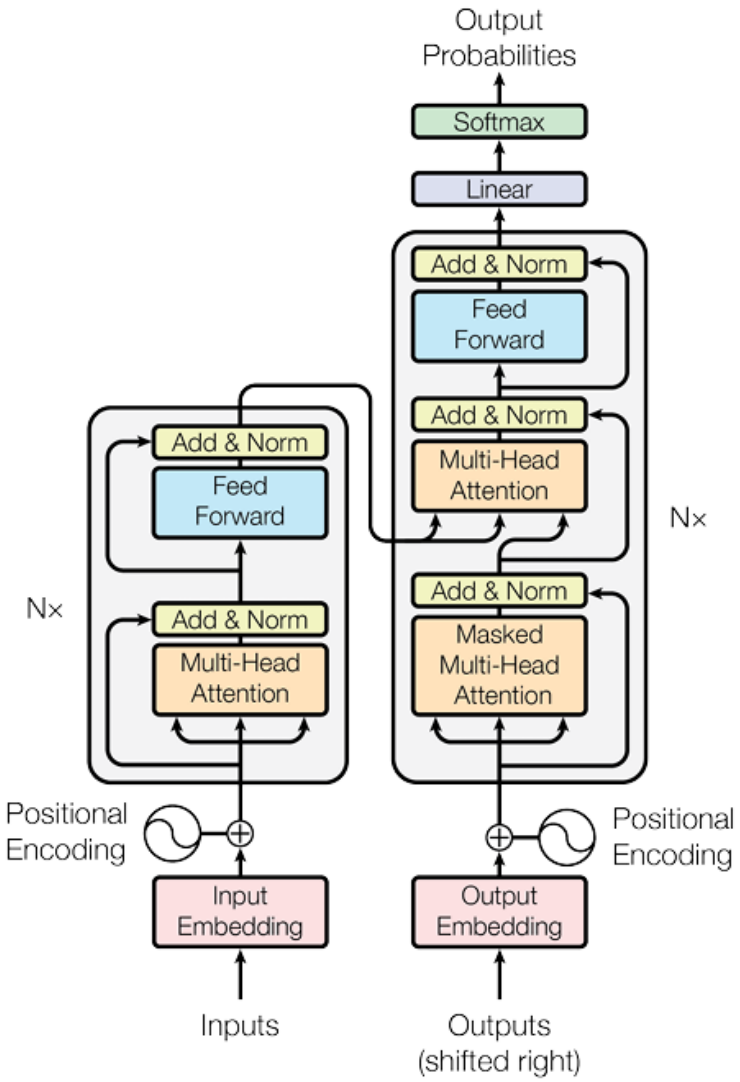

Figure 2.3: Architecture of Transformer (Diagram taken from [15])

The advantage of Transformer is that the full sentence is passed at the same time, therefore it requires less time than recurrent network models. The attention mechanism is the most important thing in this architecture. Here attention implies the importance of each word in the whole sentence. Later we discuss the attention mechanism of the Transformer briefly in Section 2.3.2. Transformer is used as a building block of many LLMs.

Unlike recurrent neural networks, Transformer does not require sequential processing of the input data, allowing for parallelization during the training phase, which speeds

up the training phase. The self-attention mechanism helps this model to capture the correlation between distant words in a sequence of input sentences. The accuracy can be scaled up by increasing the number of layers and attention heads and this improves the performance for various natural language processing tasks.

2.3.2 Attention

An attention is a function that takes query, key, value vectors as inputs and outputs the attention vectors. It enables the model to give more focus on the important part from a sequential data to extract the context accurately.

Scaled Dot-Product Attention

The input sequential data is converted in three vectors query(Q), key(K), value(V), where the dimension of Q, K is d_k and the dimension of V is d_v. Scaled dot-product attention is given in Figure 2.4. First taking the dot-product between the set of queries and keys, it is scaled by $\sqrt{d_k}$, and then again the dot-product with the set of values is taken. The formula for scaled dot-product attention is:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V\ .$$

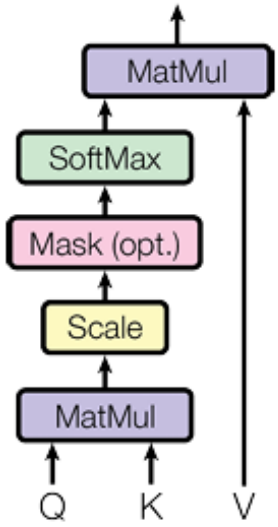

Figure 2.4: Scaled Dot-Product Attention (Diagram taken from [15])

Multi-Head Attention

In the case of Multi-Head Attention (MHA), multiple scaled dot-product attention functions are used and then we concatenate all the outputs to get the final output.

Multi-head attention is given in Figure 2.5. Here linear projections of the query, key, and value are taken, and then with that projected query, key, and value, each of the scaled dot-product attention is performed and all the outputs are concatenated. After that, we take linear projection, resulting in final outputs. Here, dimensions of query, key and value are d_k,d_k,d_v respectively and h is the number of attention heads.

$$\text{MHA}(Q, K, V) = \text{Concat}(head_1, head_2, ..., head_h)W^o,$$

where $head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and the linear projections are the parameter matrices $W_i^Q \in \mathbb{R}^{d_m \times d_k}$, $W_i^K \in \mathbb{R}^{d_m \times d_k}$, $W_i^V \in \mathbb{R}^{d_m \times d_v}$ and $W^o \in \mathbb{R}^{hd_v \times d_m}$.

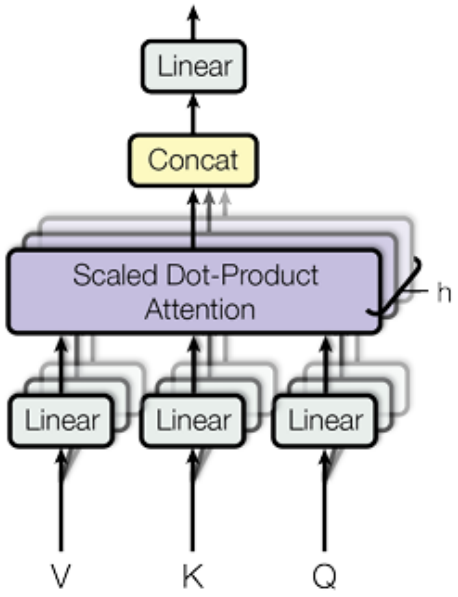

Figure 2.5: Multi-Head Attention (Diagram taken from [15])

2.3.3 Encoder

Encoder is a deep learning model and consists of 4 layers. The first layer is a Multi-head attention layer, the second layer is Add & Norm, the third layer is a feed-forward network and the last layer is Add & Norm. In the third layer, the feed-forward network is a two-layer neural network model in which the activation functions for the first layer is ReLu and the second layer is linear. The architecture of the encoder is given in Figure 2.6.

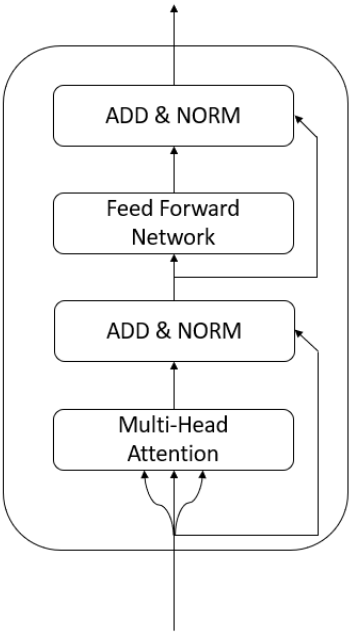

Figure 2.6: Architecture of Encoder

Now we discuss the four layers of an encoder:

First layer

In this layer, Multi-head attention is computed. Here the size of input X is $N \times d_m$, where N is the length of the input sequence and d_m is the dimension of each token. To compute multi-head attention we take Q, K, V all equal to X. The output of this layer is X_1 with size $N \times d_m$. In this layer, there are $(5h + 1)$ matrix multiplications, Nh softmax operations and hN^2 division operations.

Second layer

In this layer, first $X_2^{'} = X + X_1$ is computed and then $X_2 = \text{normalization}(X_2{}^{'})$ is computed. In this layer, there are N normalization operations of size d_m.

Third layer

In this layer, $X_3 = \max(0, b_1 + X_2 W_1) W_2 + b_2$ is computed, where W_1, W_2 are weight matrices of size $d_m \times d_{ff}, d_{ff} \times d_m$ repectively and b_1, b_2 are biases. In this layer, there are 2 matrix multiplications and $N d_{ff}$ comparison operations.

Fourth layer

In this layer, first $X_4^{'} = X_3 + X_2$ is computed and then $X_4 = \text{normalization}(X_4^{'})$ is computed. In this layer, there are N normalization operations of size d_m.

2.3.4 Bert

Bidirectional Encoder Representation from Transformer (Bert) is a natural language processing model and is introduced in the paper Bert [4]. It has significant use in various natural language proceesing tasks and has a robust method for pre-training. Generally, language models read the text data either from left to right or right to left but Bert processes the text in both directions simultaneously. This bidirectional approach helps Bert to understand the context of each word based on its position in a sentence and that gives Bert to get a deep understanding of that language's grammar. Bert is based on Transformer architecture and it is a stack of Transformer's Encoders. The Base model is a stack of 12 encoders and the Large model is a stack of 24 encoders. The architecture of Bert is given in Figure 2.7.

Pre-Training

In the pre-training phase of Bert model two tasks are performed: Masked Language Modeling and Next Sentence Prediction.

- **Masked Language Modeling**:
  In this task random words from a sentence are masked and then this sentence with masked words is given as input to the model and the model is trained to predict the masked words using the context of the neighbor words. This method makes Bert to achieve the bidirectional representation.

- **Next Sentence Prediction**:
  In this task the model is given two sentences at a time and is trained to predict whether one sentence follows the other. This task makes Bert to understand the correlation and association between sentences.

Fine Tuning

The pre-training phase makes Bert to understand the language. After that, Bert can be fine-tuned on specific tasks with relatively smaller data than the pre-training

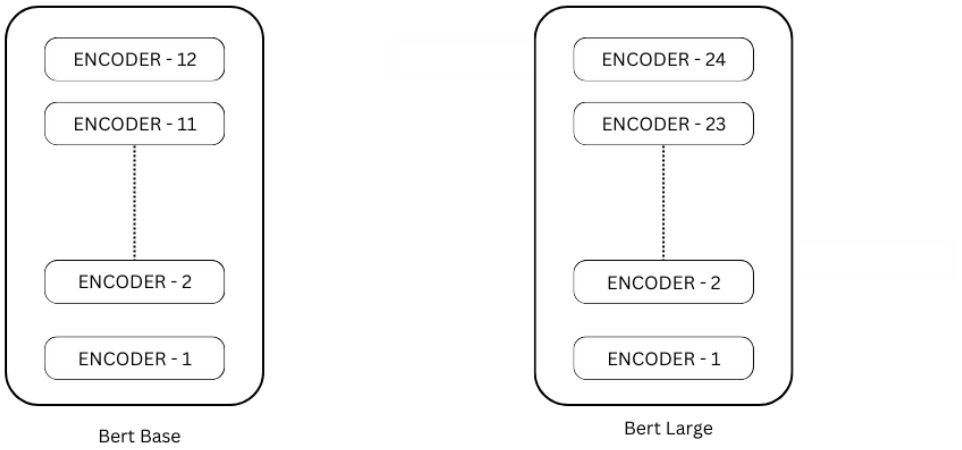

Figure 2.7: Bert Architecture

phase's data. This makes the model more accurate on specific tasks like question-answering, sentiment analysis, etc.

2.4 Mathematical Operations

In this section, we discuss the different types of mathematical operations required to execute a Bert model. Our goal is to execute all the operations in a shared setup so that we can execute Bert in a shared setup and that makes Bert private. We divide all the operations into two classes: linear operations and non-linear operations.

2.4.1 Linear Operations

- Addition
- Matrix Multiplication

2.4.2 Non-Linear Operations

- Division
- Rectified Linear Unit
- Exponential
- Square-root

The dataset of the LLMs consists of real numbers, but MPC protocols are compatible with ring structures. To use MPC protocols for executing the above-mentioned operations where the input consists of real numbers we use fixed-point representation. In the next section, we discuss that.

2.5 Fixed-Point Arithmetic

Fixed-point representation is a method to represent a real number by storing a fixed number of digits for the fractional part. In MPC we transform the fixed-point number to an integer by scaling it according to its precision.
For example, let's consider the number 1.234, therefore, the fixed-point representation of 1.234 with 2-digit precision is 1.23, now multiplying by 100 we get an integer 123. In our case we consider x to be a fixed-point binary number with f bits fractional part, therefore multiplying by 2^f with x, we get $\hat{x} = 2^f x$, which is an integer. Let $\hat{x}$ is a k bit number, so now $\hat{x} \in \mathbb{Z}_{2^k}$.
For our case, we take integers in the domain $(-2^{l_k}, 2^{l_k})$, where $l_k < k - 1$ and we represent it in an element of $\mathbb{Z}_{2^k}$ as follows:

$$x = \begin{cases} x & \text{if } x \in [0, 2^{l_k}) \\ 2^k - |x| & \text{if } x \in (-2^{l_k}, 0) \end{cases} .$$

2.6 Pseudo-Random Function (PRF)

Given a function $F : K \times X \to Y$, where X, Y, K are input space, output space and key space respectively, F is a Pseudo-Random function if the following conditions hold :

- $\forall k \in K$, $F_k(.)$ is a deterministic function
- It is computationally infeasible for any efficient algorithm to distinguish between $F_k(.)$ and a truly random function without knowing k

We use $F_k(.)$ to represent a PRF F with key k.

2.7 Security Model

2.7.1 Semi-Honest Security

Let's consider a cryptographic protocol among n parties and in this model assume that all parties strictly follow the protocol honestly without deviating from the prescribed steps of that protocol. However, parties may try to learn additional information about the other parties' private input from the transcript of the execution of this protocol. If the transcript is such that it is computationally indistinguishable from uniformly random transcript then no party learns any additional information, such notion of security is known as semi-honest security.

2.7.2 Honest-Majority Security

Honest Majority security for a MPC protocol refers to the security assumption that if a majority of the participating parties in this protocol are honest and correctly follow the protocol, then this protocol is secure.

Chapter 3

MPC For Different Operations

In this chapter, we explore various MPC protocols used to perform addition, multiplication, matrix multiplication, division, ReLu, softmax and normalization operations, all of which are essential for running the Bert model in a shared setup.

3.1 2-Party share to 3-Party share conversion

This is a 3-party protocol for converting a 2-party share into a 3-party share. Here P_i has input $\langle x \rangle^A = x_i$ for $i \in \{1, 2\}$ and P_1, P_2 have shared key for a public PRF and using that they generate two random values r_1, r_2. The complete protocol is given below:

- Setup : 3-Party
- Input : Arithmetic sharing of x in 2 parties
- Output : Replicated secret sharing of x

Algorithm 1 Π_{2-to-3}

1. P_1 sets $z_1 := x_1 - r_1$ and sends this to P_3
2. P_2 sets $z_3 := x_2 - r_2$ and sends this to P_3
3. P_1, P_2 set $z_2 := r_1 + r_2$

This protocol takes $2k$ bits of communication in 1 round. Here, we consider a semi-honest setup with an honest majority, meaning at most one party can be corrupted, and that party may attempt to learn additional information. In this protocol, there are no incoming messages for P_1 and P_2, so they do not obtain any information. If P_3

is corrupted, it tries to extract additional information from the protocol's execution transcript. However, P_3 receives the transcript $T := \{x_1 - r_1, x_2 - r_2\}$, which is computationally indistinguishable from a uniformly random distribution. Therefore, P_3 learns nothing, ensuring the protocol's security in this setup.

3.2 Addition

In this section, we discuss 2-party and 3-party protocols for addition operation. Our goal is to combine different frameworks and make an efficient framework, therefore we need both 2-party and 3-party protocols for addition and that helps us to execute addition in both 2-party and 3-party setups.

3.2.1 3-Party case

The 3-party addition protocol is given below:

- Setup : 3-Party
- Input : Replicated secret sharing of x, y
- Output : Replicated secret sharing of $x + y$

Algorithm 2 Addition : $\Pi^3{}_{add}$

---

1. P_i locally computes $z_i = x_i + y_i$ and $z_{i+1} = x_{i+1} + y_{i+1}$
2. Each P_i has 2-out-of-3 sharing of $x + y$

So, this 3-party addition can be done locally, with no need for communication.

3.2.2 2-Party case

The 2-party addition protocol is given below:

- Setup : 2-Party
- Input : Arithmetic secret sharing of x, y
- Output : Arithmetic secret sharing of $x + y$

Here P_i has x_i, y_i as inputs for $i = 1, 2$ such that $x = x_1 + x_2$ and $y = y_1 + y_2$.

Algorithm 3 Addition : $\Pi^2{}_{add}$

1. P_i locally computes $z_i = x_i + y_i$
2. Each P_i has arithmetic sharing of $x + y$

So, this 2-party addition can be done locally, with no need for communication.

3.3 Multiplication

In this section, we discuss MPC protocols for multiplication in two cases, in the first case both inputs are in arithmetic sharing and in the second case one input is in arithmetic sharing and the other is in binary sharing.

3.3.1 Case 1 : When both inputs are in Arithmetic Sharing

This MPC protocol is based on 3-party setup and inputs are replicated secret sharing of x and y and the output is replicated secret sharing of xy.
This multiplication with truncation protocol is taken from ABY^3 [10], which is an efficient multiplication with truncation. Parties P_1, P_2, P_3 have 2-out-of-3 shares of inputs x, y, where $x = x_1 + x_2 + x_3$ and $y = y_1 + y_2 + y_3$. Each party P_i has keys k_i, k_{i+1} and a public PRF F for the parties to generate zero sharing at each time. Suppose till $J-1$-th zero sharing is generated. The complete protocol is given below:

- Setup : 3-Party
- Input : Replicated secret sharing of x, y
- Output : Replicated secret sharing of xy

Algorithm 4 Multiplication with Truncation : Π_{Mult}

1. Each P_i locally computes $\alpha_i = F_{k_i}(J) - F_{k_{i+1}}(J)$
2. P_i locally calculates $z_i = x_i y_i + x_i y_{i+1} + x_{i+1} y_i + \alpha_i$
3. Each P_i sends z_i to P_{i-1}
4. P_2, P_3 locally calculate a random element $r \in \mathbb{Z}_{2^k}$ by using PRF $F_{k_3}()$ and set $m_3 := r$
5. P_1, P_3 locally compute $m_1 :=$ rightshift(z_1, f) and P_2 computes $m_2 := 2^k -$ rightshift$(2^k - (z_2 + z_3), f) - r$
6. P_2 sends m_2 to P_1
7. Parties output $[xy]^A := \{m_1, m_2, m_3\}$

So here we need 2 rounds and a total $4k/3$ bits of communication per party in the online phase and nothing is required in the offline phase. This protocol is secure in a semi-honest setup with an honest majority.

3.3.2 Case 2 : When one input is in Arithmetic Sharing and the other is in Binary Sharing

This protocol is taken from ABY^3 [10]. In semi-honest setting, to compute $[a]^A[b]^B \rightarrow [ab]^A$, it is same to do $x[b]^B \rightarrow [xb]^A$ two times with taking values of x as a_1 and $a_2 + a_3$, where $a = a_1 + a_2 + a_3$.

- Setup : 3-Party
- Input : Replicated arithmetic sharing of a and replicated binary sharing of bit b
- Output : Replicated arithmetic sharing of ab

Therefore first we go through the protocol for computing $x[b]^B \rightarrow [xb]^A$.
Suppose here party 3 knows the value x. The protocol is given below:

Algorithm 5 : $x[b]^B \rightarrow [xb]^A$

1. Parties 1 and 3 locally sample $c_1 \leftarrow \mathbb{Z}_{2^k}$ and parties 2 and 3 locally sample $c_3 \leftarrow \mathbb{Z}_{2^k}$
2. Party 3 defines two messages $m_i := (i \oplus b_1 \oplus b_3)x - c_1 - c_3$
3. Party 3 uses these messages as inputs of three-party OT and party 3 works as sender of this OT and parties 1 & 2 work as receiver and helper of this OT respectively with input b_2 and party 1 gets message $c_2 = m_{b_2} = xb - c_1 - c_3$ and sends it to party 2

This protocol takes a total of $4k$ bits of communication in 2 rounds due to the three-party OT from ABY^3 [10].

Since $a[b]^B = a_1[b]^B + (a_2 + a_3)[b]^B$, therefore using the above protocol twice, $[a]^A[b]^B \rightarrow [ab]^A$ is computed and for $a_1[b]^B$ part party 1 can play the role of sender and for $(a_2 + a_3)[b]^B$ party 2 can play the role of sender. We denote this protocol by Π_{MultAB}. Hence total of $8k$ bits of communication are required over 2 rounds in Π_{MultAB}, which implies $8k/3$ bits of communication per party in 2 rounds. This protocol is secure in a semi-honest setup with an honest majority.

3.4 ReLu

Rectified Linear unit (ReLu) is a popular activation function used in neural networks and deep learning. It is defined as follows : $\text{ReLu}(x) = \max(0, x)$. The derivative of the ReLu, commonly referred to as DReLu, is used in the backpropagation process during neural network training. It is defined as follows:

$$\text{DReLu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} .$$

Here we discuss some MPC protocols for computing ReLu in a shared setup.

3.4.1 Using ABY^3 Framework

- Setup : 3-Party
- Input : Replicated secret sharing of x
- Output : Replicated secret sharing of $\text{ReLu}(x)$

This protocol uses ABY^3 framework and is one of the efficient MPC protocol for computing ReLu in a shared setup. The high-level idea of this protocol is first the parties convert the arithmetic sharing into binary sharing, so they get the binary sharing of the sign of x and after that, they execute multiplication protocol for doing multiplication where one input is in arithmetic sharing and the other is in binary sharing.
In ABY^3 [10], they mention the Bit Decomposition protocol that converts $[x]^A$ into $[x]^B$ and this protocol takes total $(k + klogk)$ bits of communication for each party in $(1 + logk)$ rounds. Using this protocol the parties get binary sharing of the sign bit b of x.
After that, they use Π_{MultAB} to do $[x]^A[b]^B \longrightarrow [xb]^A$ and this takes $8k/3$ bits of communication per party in 2 rounds. This gives the replicated secret sharing of ReLu(x).
Therefore to compute ReLu, this protocol takes $(\frac{11k}{3} + klogk)$ bits of communication per party in $(3 + logk)$ rounds. This protocol is secure in a semi-honest setup with an honest majority.

3.4.2 Using Bicoptor Framework

This MPC protocol is introduced in the Bicoptor framework [19] and the key aspect of this protocol is that it requires a constant number of rounds to compute the ReLu in the online phase and no offline phase is required. Therefore, when greater efficiency in terms of the number of rounds is required, this protocol is superior to the previously mentioned protocol using ABY^3 framework.

This protocol uses truncation protocol between two parties from SecureML [12] and we use the notation TRC(x, f) to represent the truncation of the last f bits from x. Consider two parties, P_1 and P_2, each holding the arithmetic sharing of a secret value x which means P_i has x_i such that $x_1 + x_2 = x$. This truncation is done as follows:

- P_1 computes $y_1 = \text{TRC}(x, f)_1 = \text{rightshift}(x_1, f)$
- P_2 computes $y_2 = \text{TRC}(x, f)_2 = 2^k - \text{rightshift}(2^k - x_2, f)$

This truncation is used in their protocol. Bicoptor modifies the Strawman DReLu protocol using masking by random value and then extends that protocol for computing ReLu. This protocol is in an unbalanced 3-party setup which means two of the three parties hold the share of the input and one party holds nothing. The output of this protocol is the secret sharing of ReLu in 2 parties. The complete setup of this protocol is given below:

- Setup : 3-Party Unbalanced model P_1, P_2, P_3
- Input : Arithmetic sharing of x between 2 parties

- Output : Arithmetic sharing of ReLu(x) between 2 parties

For computing ReLu, Bicoptor needs 2 rounds and a total of $(2l_k + 9)k$ bits of communication in the online phase, and no preprocessing is required.
Since this protocol gives output as secret sharing of ReLu(x) in 2 parties but we need replicated secret sharing at each level of operation so we modify this protocol a little bit by adding an extra step in the last to make the output 2-out-of-3 sharing and for that, we add the conversion protocol Π_{2-to-3}, which needs $2k$ bits of communication in 1 round. So in total, the modified protocol for ReLu takes $(2l_k + 11)k$ bits of communication in 3 rounds in the online phase without preprocessing. Here we take input as replicated secret sharing between 3-party and any 2 parties can directly convert this share into 2-party sharing. This protocol is secure in a semi-honest setup with an honest majority. The modified protocol is given below:

- Setup : 3-Party model P_1, P_2, P_3
- Input : Arithmetic sharing of x between 2 parties
- Output : Replicated secret sharing of ReLu(x)

Algorithm 6 ReLu : $\Pi_{ReLu}^{mBicoptor}$

1. P_1, P_3 generate a_1, b_1, c_1 from seed_{13} and P_2, P_3 generate a_2, b_2 from seed_{23}
2. Parties execute the DReLu protocol [19] and P_1, P_2 input $\langle x \rangle^A$ to this protocol
3. P_1 sends $d_1 := x_1 - a_1$ to P_2 and P_2 sends $d_2 := x_2 - a_2$ to P_1
4. P_3 gets DeReLu$(x)'$ from execution of DReLu protocol and computes $c_2 := (a_1 + a_2)(b_1 + b_2) - c_1$ and $e := \text{DeReLu}(x)' - (b_1 + b_2)$ and sends e to P_1, and e, c_2 to P_2
5. P_1, P_2 set $d = d_1 + d_2$ and compute the shares $z_i = \langle \text{ ReLu}(x) \rangle^A = (1 - 2t)(de + d\langle b \rangle^A + e\langle a \rangle^A + \langle c \rangle^A) + t\langle x \rangle^A$
6. All parties execute Π_{2-to-3} with secret inputs z_1, z_2 to get 2-out-of-3 sharing

3.4.3 Using Sigma Framework

In Sigma framework they introduce an FSS-based 2-party DReLu protocol and this protocol gives the share of DReLu. For a k-bit value $x \in \mathbb{Z}_{2^k}$ the DReLu of x is written as :

$$\text{DReLu}(x) = 1\{x < 2^{k-1}\} = 1 \oplus \text{MSB}(x) \ ,$$

where $\{x < 2^{k-1}\}$ is an indicator function, equals to 1 if $x < 2^{k-1}$, otherwise 0. Since in FSS-based protocol, the input is masked by random value r_{in} and output is masked by random value r_{out}, so the offset function can be written as:

$$\begin{aligned}\text{DReLu}^{[r_{in},r_{out}]}(\hat{x}) &= \text{DReLu}(\hat{x} - r_{in}) \oplus r_{out} \\ &= 1 \oplus \text{MSB}(\hat{x} - r_{in}) \oplus r_{out}\,.\end{aligned}$$

They use the logic from CrypTFlow2 [14] to compute the MSB over secret shared value. Let $x = x_0 + x_1 \mod 2^k$ and $z_0 = x_0 \mod 2^{k-1}$, $z_1 = x_1 \mod 2^{k-1}$, then

$$\text{MSB}(x) = \text{MSB}(x_0) \oplus \text{MSB}(x_1) \oplus 1\{z_0 + z_1 \geq 2^{k-1}\}\,.$$

Using this above form they rewrite DReLu as follows:

$$\begin{aligned}\text{DReLu}^{[r_{in},r_{out}]}(\hat{x}) &= \text{MSB}(\hat{x} - r_{in}) \oplus r_{out} \oplus 1 \\ &= \text{MSB}(\hat{x}) \oplus \text{MSB}(2^k - r_{in}) \oplus 1\{2^{k-1} - z_0 - 1 < z_1\} \oplus r_{out} \oplus 1\,,\end{aligned}$$

where $z_0 = \hat{x} \mod 2^{k-1}$ and $z_1 = 2^{k-1} - r_{in} \mod 2^{k-1}$.

In their protocol they compute this expression to get the DReLu. In the key generation algorithm, they use the key generation function of DPF and in the evaluation algorithm, they use the evaluation algorithm of DCF. This DReLu protocol is non-interactive and gives the masked output of DReLu.
To construct the FSS protocol for ReLu, we use the select protocol from the Orca paper [8] with the above protocol for DReLu. The select protocol's functionality is to take two inputs, a selector bit s and a number x, and output x if $s = 1$, otherwise, it outputs 0.

To compute ReLu using the function secret sharing, we execute the following protocol:

- Setup : 2-Party
- Input : Arithmetic secret sharing of x
- Output : Arithmetic secret sharing of $\text{ReLu}(x)$

The cost analysis of this protocol is given in Table 3.2.

Algorithm 7 ReLu : Π_{ReLu}^{FSS}

1. Two Parties execute the DReLu protocol from Sigma and get the shares of the masked output of DReLu(x) and reconstruct the masked output of DReLu(x) using interaction
2. Two Parties use select protocol from Orca [8] to get the masked output of ReLu(x)

3.5 Division

This is a 3-party MPC protocol and it is introduced in Falcon [17]. The high-level idea of this protocol is first the parties find out the bounding power α for b such that $2^{\alpha} \leq b < 2^{\alpha+1}$ using Bounding Power protocol Π_{Pow} from Falcon [17] and then transform b by taking $\alpha + 1$ bits as precision of the fixed-point representation so that $b \in [0.5, 1)$ and after that they use approximation of $\frac{1}{b} = w_0(1 + \epsilon_0)(1 + \epsilon_1)$ from [3] and then the parties execute multiplication to compute $a\frac{1}{b}$. This protocol is secure in a semi-honest setup with an honest majority. The complete protocol is given below:

- Setup : 3-Party
- Input : Replicated secret sharing of a, b
- Output : Replicated secret sharing of $\frac{a}{b}$

Algorithm 8 Division : Π_{Div}

1. Parties run the Π_{Pow} protocol to find α such that $2^{\alpha} \leq b < 2^{\alpha+1}$ and transform $b \rightarrow x$ such that $x \in [0.5, 1)$ by taking $\alpha + 1$ bits of precision in the fixed-point representation
2. Parties compute $w_0 := 2.9142 - 2x$
3. Compute $\epsilon_0 := 1 - xw_0$ and $\epsilon_1 := \epsilon_0^2$
4. Outputs $aw_0(1 + \epsilon_0)(1 + \epsilon_1)$

This division protocol takes $(5k^2 + k + klogk + 7k)$ bits of communication in $(5k + klogk + 7)$ rounds in the online phase and $(5k^2 + k^2logk)$ bits of communication in $(2+logk)$ rounds in the offline phase. In this Falcon paper there we find some mistakes in the calculation of communication costs in some protocols and we update that and use the modified communication costs in this division protocol.

3.6 Softmax

The softmax function $f : \mathbb{R}^n \to \mathbb{R}^n$ takes an n-dimensional vector $\vec{x} = (x_1, x_2, ...x_n)$ as input and outputs another n-dimensional vector $\vec{y} = (y_1, y_2, ..., y_n)$.

$$f(x_1, ..., x_n) = (y_1, ..., y_n)\,,$$

$$\text{where } y_i = \frac{e^{x_i}}{\Sigma_{j=1}^{n} e^{x_j}} \text{ , for } i \in \{1, 2, ..., n\}.$$

Since $\Sigma_{i=1}^{n} y_i = 1$, the outputs can be considered a probability distribution. This function is used as an activation function in the neural network models. In Bert, this function is used for calculating the attention vectors for each token. In this project to make Bert private we need MPC protocol for calculating softmax in a shared setup and here we discuss MPC protocols for softmax using different frameworks that we use later for different requirements.

3.6.1 Using Puma Framework

In Puma [5], they introduce MPC protocol for calculating softmax using basic MPC protocols for comparison, multiplication, truncation, square and use the underlying protocols as black-box manner.
The high-level idea of this protocol is that first they represent the softmax function in such a way that all power of the exponentials are negative and then they approximate exponentials using the Taylor series with a simple trick given below:

$$\text{negExp}(x) = \begin{cases} 0 & \text{if } x < T_{exp} \\ (1 + \frac{x}{2^t})^{2^t} & \text{if } x \in [T_{exp}, 0] \end{cases} .$$

To compute the negExp, parties use comparison protocol to compare $x < T_{exp}$ and get binary shares of that comparison. After that using truncation protocol $\frac{x}{2^t}$ can be computed. Then parties use square protocol t times to get $(1 + \frac{x}{2^t})^{2^t}$. After that, parties use multiplication protocol between arithmetic and binary sharing to get the exponentiation values and sum all the exponential. Then using reciprocal protocol parties calculate the inverse of that sum and finally multiply this value by all the exponentiation to get the softmax. This protocol is secure in a semi-honest setup with an honest majority. The complete protocol is given below:

- Setup : 3-Party
- Input : Replicated secret sharing of $\vec{x}$
- Output : Replicated secret sharing of Softmax($\vec{x}$)

Algorithm 9 Softmax : $\Pi^{puma}_{softmax}$

1. Parties jointly compute $[b]^B$ which is the sign of $x - T_{exp}$ using comparison protocol and the maximum component $[x_{max}]^A$ in $\vec{x}$
2. Parties locally compute $[\vec{y}]^A = [\vec{x}]^A - [x_{max}]^A - \epsilon$
3. Parties jointly truncate t bits from $\vec{y}$ using truncation protocol and then increase it's value by 1 and set that value in $[z_0]^A$
4. for $j = 1, 2, ..., t$ do

   set $[z_j]^A = [z_{j-1}.z_{j-1}]^A$

   end for
5. Parties locally compute $[Z]^A = \Sigma_{i=1}^{n}[z[i]]^A$
6. Parties jointly compute the inverse $[Y]^A := \frac{1}{[Z]^A}$ using reciprocal protocol
7. Parties jointly compute $[z.Y]^A$ using multiplication protocol
8. Using arithmetic and binary sharing multiplication parties compute $[y]^A := [b]^B.[zY]^A$ as output

The cost analysis of the above protocol is given in Table 3.4.

3.6.2 Using Sigma Framework

In Sigma framework MPC protocols are based on FSS. To compute softmax of $\vec{x} = (x_1, ...x_n)$ this framework uses three operations, which are maximum, exponentiation of negative values and inverse. Softmax function can be written as the following way:

$$y_i = \frac{e^{x_i - x_{max}}}{\Sigma_{j=1}^{n} e^{x_j - x_{max}}} ,$$
$$\text{where } y = (y_1, ..., y_n) = \text{softmax}(\vec{x}) .$$

The high-level idea is here first they use maximum protocol to find out x_{max} in $\vec{x}$. Then the parties locally subtract x_{max} from each component of $\vec{x}$. After that, they use exponentiation protocol to get the exponentiation of negative values and get z_i and then compute locally $z = \Sigma_{j=1}^{n} z_j$. After that, they invoke inverse protocol to obtain z^{-1} and then compute $y_i := z_i.z^{-1}$ with truncation to get the final result y_i. The overall protocol is a combination of multiple FSS-based protocols. They use

their own DReLu protocol [6] for DReLu operation and Truncation protocol [6] for truncation. They also use multiplication protocol from [2], Select protocol from Orca [8], Look-up-Table protocol from Pika [16]. This Multiplication is a non-interactive protocol with keysize $3k$. The Select protocol is a non-interactive FSS-based protocol and the keysize is $4k$. The look-up-table protocol takes $2k$ bits of communication in 1 round. Combining those protocols they provide an overall FSS-based framework for softmax.

Maximum

To calculate the maximum x_{max} from $\vec{x}$ they use their comparison protocol [6] $(n-1)$ times and Select protocol $2\lceil logn \rceil$ times.

Negative Exponential

To get the exponential they introduce an FSS-based negative exponential protocol [6], this protocol takes input as $\hat{x} = x + r_{in}$, which is masked by random value r_{in} and gives output, which is also masked by random value r_{out}. This protocol is a combination of multiple FSS-based protocols: DReLu, Select, Truncate-Reduce [6], look-up-table, Multiplication, GapARS [6].

Inverse

To obtain z^{-1} they use look-up-table of the chosen size. Since the dimension of $\vec{x}$ is n and each $z_i \in [0, 1]$ and at least one of z_i is 1, that implies $z \in [1, n]$. They use look-up-table protocol from Pika [16] in an efficient way to get the inverse with appropriate bitwidth.

The cost analysis of this protocol is given in Table 3.4.

3.7 Normalization

In Bert model, we need normalization operation in the second and fourth layers of each encoder block. The normalization of $\vec{x} = (x_1, ..., x_n)$ is done as follows:

$$\text{Normlization}(\vec{x}) = \gamma.\frac{x_i - \mu}{\sqrt{\sigma}} + \beta \ ,$$

where μ is the mean and σ^2 is the variance of $\vec{x}$.
In Puma [5], they introduce a protocol using basic MPC protocols to compute normalization operation.

The high-level idea of this protocol is initially parties hold 2-out-of-3 sharing of $\vec{x}$ and then they locally compute share of mean $\mu := \frac{\Sigma_{i=1}^{n} x_i}{n}$ and jointly compute the

variance $\sigma^2 = \frac{\Sigma_{i=1}^{n}(x_i-\mu)^2}{n}$. Then parties use InvertSqrt protocol from [9] to get $\frac{1}{\sigma}$ from variance σ^2. After that parties jointly compute $(x_i - \mu).\frac{1}{\sigma}$ using multiplication protocol from ABY^3 [10] and then multiply this value with γ using multiplication protocol from ABY^3 and add β to get the final output.
This protocol is secure in a semi-honest setup with an honest majority. Here P_1, P_2, P_3 are the 3 parties holding shares of input $\vec{x}$. The complete protocol is given below:

- Setup : 3-Party
- Input : Replicated secret sharing of $\vec{x}$
- Output : Replicated secret sharing of Normalization($\vec{x}$)

Algorithm 10 Normalization : Π_{norm}^{puma}

1. P_1, P_2, P_3 locally compute $[\mu]^A = \frac{\Sigma_{i=1}^{n}[x_i]^A}{n}$
2. Parties jointly compute $[\sigma^2]^A = \left[\frac{\Sigma_{i=1}^{n}(x_i-\mu)^2}{n}\right]^A$
3. Parties jointly compute $[\sigma^{-1}]^A$ using InvertSqrt protocol from [9]
4. Parties jointly compute $[z_i]^A = [(x_i-\mu).\sigma^{-1}]^A$ for all $i \in \{1, 2, ..., n\}$ using Π_{Mult} (Algorithm 4)
5. Parties jointly compute $[s_i]^A = [z_i.\gamma]^A$ for all $i \in \{1, 2, ..., n\}$ using Π_{Mult} (Algorithm 4)
6. Parties locally compute $[y_i]^A = [s_i]^A + [\beta]^A$ for all $i \in \{1, 2, ..., n\}$ as output

The cost analysis of this protocol is given in Table 3.4.

3.8 Cost Analysis

In this section, we describe the online and offline cost analysis of different operations using various MPC protocols in the form of tables. In Table 3.1, we mention the cost of *Multiplication with Truncation*, *Matrix Multiplication with Truncation* and *Division* operations using different frameworks. In Table 3.2, we mention the costs of *ReLu* and *Maximum* operations using various frameworks. In Table 3.3, we mention the number of AES calls required to evaluate *ReLu*, *Softmax*, *Maximum* using Sigma framework. In Table 3.4, we mention the cost of *Softmax*, *Normalization* operations using different frameworks.

Frameworks	Phase	Mult. With Truncation		Matrix Mult. with Truncation $[(x \times y), (y \times z)]$		Division	
		Rnd	Com.	Rnd	Com.	Rnd	Com.
SecureML (2-party) [12]	Online	1	$2k$	1	$(x+z)yk$	–	–
	Offline	1	$k(k+\lambda)/2$	1	$xyzk(k+\lambda)/2$	–	–
ABY^3 (3-party) [10]	Online	2	$\frac{4k}{3}$	2	$\frac{4kxz}{3}$	–	–
	Offline	0	0	0	0	–	–
Falcon (3-party) [17]	Online	2	$\frac{4k}{3}$	2	$\frac{4kxz}{3}$	$5k + klogk + 7$	$5k^2+k+klogk+7k$
	Offline	0	0	0	0	$2+logk$	$5k^2 + k^2logk$
Bicoptor (3-party) [19]	Online	2	$\frac{5k}{3}$	2	$\frac{xz+2xy+2yz}{3}$	–	–
	Offline	0	0	0	0	–	–
Cheetah (2-party) [7]	Online	–	–	1	$kz\lceil\frac{x(N_p+1)}{2} + \frac{N_p y}{n_{iw}}\rceil$	–	–
	Offline	–	–	0	0	–	–
Boyle (2-party) [2]	Online	0	0	–	–	–	–
	Offline	1	$3k$	–	–	–	–

Table 3.1: Cost analysis of MPC protocols for *Multiplication*, *Matrix Multiplication*, *Division* operations

Here "Mult." stands for multiplication, "Rnd" stands for the number of rounds, and "Com." stands for average communication cost per party in bits. Here k is the bit-size of the ring element, n is the size of vector input, n_{iw} is input-window size.

Frameworks	Phase	ReLu		Maximum (n)	
		Rnd	Com.	Rnd	Com.
Sigma (2-party) [6]	Online	1	1	$2\lceil logn \rceil$	$n-1$
	Offline	1	$1+4k+2\lambda+(\lambda+2)[k-1-log(\lambda+1)]$	1	$(n-1)[1+2\lambda+(\lambda+2)(k-1-log(\lambda+1))]+8k\lceil logn \rceil$
ABY^3 (3-party) [10]	Online	$3+logk$	$\frac{11k}{3}+klogk$	–	–
	Offline	0	0	–	–
Falcon (3-party) [17]	Online	$5+logk$	$5k+1+logk$	–	–
	Offline	$2+logk$	$5k+klogk$	–	–
Bicoptor (3-party) [19]	Online	2	$\frac{(2l_k+9)k}{3}$	2	$\frac{k[n^2(2+l_k)+n(3-l_k)]}{3}$
	Offline	0	0	0	0

Table 3.2: Cost analysis of MPC protocols for *ReLu* and *Maximum* operations

Here "Rnd" stands for the number of rounds, and "Com." stands for average communication cost per party in bits. Here k is the bit-size of the ring element, n is the size of vector input, and λ is security parameter.

Relu	Softmax (n)	Maximum (n)
$[k-1-log(\lambda+1)]$	$(n-1)[k-1-\nu]+n[k-1+8+f+(2^{8-\nu}-1)(k-\nu)-3\nu]+[f-6-\nu]+(2^{8-\nu}-1)(k-\nu)+n[k-2\nu]$	$(n-1)[k-1-log(\lambda+1)]$

Table 3.3: Number of AES calls required to evaluate different operations using FSS protocols in Sigma.

Here λ is security parameter and $\nu = log(\lambda+1)$, k is the bit-size of the ring element, n is the size of vector input.

Frameworks	Phase	Softmax (n)		Normalization (n)	
		Rnd	Com.	Rnd	Com.
Sigma (2-party) [6]	Online	$17 + 2\lceil logn \rceil$	$30n + 8nk + 6 + 2k - nf + log(n+1)$	–	–
	Offline	1	$n[16\lambda + 115 + (\lambda + 2)(2k + m + f + 22 - 8\nu) + 17k - 2f] + 2\lambda + 2k + 3log(n+1) + 17 + (\lambda+2)[k + f + log(n+1) - 1 - 3\nu] + 8k\lceil logn \rceil$	–	–
Puma (3-party) [5]	Online	$14+(1+k)logk+2t+5k$	$5k^2 + 12k + \frac{(8+4t)nk}{3} + (n+1)klogk + \frac{k[n^2(2+l_k)+n(3-l_k)]}{3}$	$20 + 5k+(3+k)logk$	$6k^2 + 11k + 4k(n + logk)$
	Offline	$2 + logk$	$5k^2 + k^2logk$	$2 + logk$	$5k^2 + k^2logk$

Table 3.4: Cost analysis of MPC protocols for *Softmax* and *Normalization* operations

Here "Rnd" stands for the number of rounds, and "Com." stands for average communication cost per party in bits. Here k is the bit-size of the ring element, f is the number of precision in fix point representation, n is the size of vector input, λ is security parameter, $\nu = log(\lambda + 1)$, $m = k - f + 1$ captures effective bit-width in Sigma [6] and here we use Bicoptor framework for computing maximum in Puma's Softmax protocol.

Chapter 4

Efficient Frameworks For Different Scenario

In this chapter, we propose different combined MPC frameworks for Bert corresponding to different requirements.

4.1 Proposed 1st Framework

In an environment, where there is sufficient time for the offline pre-processing phase and network bandwidth is limited, then reducing the amount of communication costs in the online phase can significantly improve the performance and reliability. In this framework, our primary focus is on minimizing the communication costs in the online phase. We mainly select the function secret sharing scheme based on protocols from the Sigma framework to achieve this. This is a 3-party semi-honest setup with an honest majority.

Below, we outline the protocols chosen to execute Bert in a 3-party setup to reduce online communication costs:

- Multiplication: ABY^3 [4]
- Division: Falcon [8]
- ReLu: Sigma [7]
- Softmax: Sigma [3.6.2]
- Normalization: Puma [10]

4.2 Proposed 2nd Framework

Consider an environment where we have high network bandwidth and sufficient time for the offline phase, then reducing the number of rounds in the online phase can significantly improve the performance and reliability and also decrease the overall latency. This setup leverages the offline phase for huge computation and ensures fast output in the online phase. For example, a University conducting online exams can use this framework to execute a Bert model for plagiarism checking in University exams. This efficient framework can process the pre-processing data in advance before the exam, ensuring that during the exam, plagiarism checking can be done securely without compromising the answer sheets of the students. That's why in this framework, we focus on reducing the number of rounds in the online phase. This is a 3-party semi-honest setup with an honest majority. This framework sets itself apart from the first proposed framework by parallelizing the three-party OT procedure, which necessitates multiplication between arithmetic and binary shared values.

Below, we outline the protocols chosen to execute Bert in a 3-party setup to reduce the number of rounds in the online phase:

- Multiplication: ABY^3 [4]
- Division: Falcon [8]
- ReLu: Sigma [7]
- Softmax: Sigma [3.6.2]
- Normalization: Puma [10]

4.3 Proposed 3rd Framework

In environments where bandwidth is limited or expensive (e.g., satellite communications, and remote areas with poor internet infrastructure), minimizing the amount of data exchanged is crucial. Here, reducing communication costs can significantly improve the feasibility and performance of MPC protocols. This is important for applications running on devices with limited data plans or where data transfer costs need to be minimized. In this proposed framework, we aim to reduce the overall communication cost, including both online and offline phases. To accomplish this, we utilize MPC protocols from various frameworks and integrate them into a comprehensive solution. We select protocols from different frameworks that minimize the overall communication cost for each operation required to execute the Bert model. Our approach considers a 3-party semi-honest setup with an honest majority, where all parties are involved in both the online and offline phases.

Below, we outline the recommended protocols in this framework for each operation required in Bert:

- Multiplication: ABY^3 [4]
- Division: Falcon [8]
- ReLu: ABY^3 [3.4.1]
- Softmax: Sigma [3.6.2]
- Normalization: Puma [10]

4.4 Proposed 4th Framework

In scenarios where we have high network bandwidth, reducing the number of rounds is crucial. Each round involves synchronization among all participating parties, and fewer rounds translate directly into lower latency. To achieve this, in this proposed framework we focus on reducing the number of rounds in both online and offline phases. We select protocols from different frameworks that minimize the number of overall rounds for each operation required to execute the Bert model. Our approach considers a 3-party semi-honest setup with an honest majority, where all parties are involved in both the online and offline phases.

Below we outline the recommended protocols in this framework for each operation required in Bert:

- Multiplication: ABY^3 [4]
- Division: Falcon [8]
- ReLu: Bicoptor [6]
- Softmax: Sigma [3.6.2]
- Normalization: Puma [10]

4.5 Theoretical Results

In this section, we mention the theoretical costs of executing Bert using these proposed frameworks. For computing theoretical costs we take the values $k = 64, l_k = 32, t = 5, N = 100, f = 16, \lambda = 128, d_m = 512, d_{ff} = 2048, h = 8, d_k = d_v = 64$. We compute

this on a stack of 12 encoders. Since we do not implement any of these proposed frameworks, therefore if these frameworks are implemented then it may be possible to do more optimizations.

- In the first framework, we get that each party communicates 213.34 MB in the online phase in 18408 many rounds and this framework takes 5 GB of communication per party in the offline phase in 8 rounds.
- In the second framework, we get that each party communicates 213.36 MB in the online phase in 18384 many rounds and this framework takes 5 GB of communication per party in the offline phase in 8 rounds.
- In the third framework, we get that each party communicates 369.34 MB in the online phase in 18480 many rounds and it takes 2.7 GB of communication per party in the offline phase in 8 rounds.
- In the fourth framework, we get that each party communicates 656.88 MB in the online phase in 18384 many rounds and this framework takes 2.7 GB communication per party in the offline phase in 8 rounds.

Chapter 5

Conclusion

In this thesis, we explore how to enhance the privacy of the Bert model using various MPC protocols. Our approach involves a comprehensive investigation into different MPC frameworks, aiming to identify and analyze the communication costs and the number of computational rounds necessary during both the online and offline phases of the protocols. Ultimately, we propose four MPC frameworks by integrating different protocols to meet specific requirements and theoretically compute the overheads associated with these frameworks. In summary, this thesis contributes to the field by proposing innovative MPC frameworks that enhance the privacy of the Bert model. Through a meticulous investigation of existing protocols and a theoretical evaluation of our proposed solutions, we lay the groundwork for future research and practical applications in privacy-preserving machine learning.

Bibliography

[1] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303 (2016)

[2] Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part I 17. pp. 341–371. Springer (2019)

[3] Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14. pp. 35–50. Springer (2010)

[4] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[5] Dong, Y., Lu, W.j., Zheng, Y., Wu, H., Zhao, D., Tan, J., Huang, Z., Hong, C., Wei, T., Cheng, W.: Puma: Secure inference of llama-7b in five minutes. arXiv preprint arXiv:2307.12533 (2023)

[6] Gupta, K., Jawalkar, N., Mukherjee, A., Chandran, N., Gupta, D., Panwar, A., Sharma, R.: Sigma: secure gpt inference with function secret sharing. Cryptology ePrint Archive (2023)

[7] Huang, Z., Lu, W.j., Hong, C., Ding, J.: Cheetah: Lean and fast secure {Two-Party} deep neural network inference. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 809–826 (2022)

[8] Jawalkar, N., Gupta, K., Basu, A., Chandran, N., Gupta, D., Sharma, R.: Orca: Fss-based secure training with gpus. Cryptology ePrint Archive (2023)

[9] jie Lu, W., Fang, Y., Huang, Z., Hong, C., Chen, C., Qu, H., Zhou, Y., Ren, K.: Faster secure multiparty computation of adaptive gradient descent. Proceed-

ings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice (2020), https://api.semanticscholar.org/CorpusID:226238776

[10] Mohassel, P., Rindal, P.: Aby3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 35–52 (2018)

[11] Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 591–602 (2015)

[12] Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE symposium on security and privacy (SP). pp. 19–38. IEEE (2017)

[13] Pang, Q., Zhu, J., Möllering, H., Zheng, W., Schneider, T.: Bolt: Privacy-preserving, accurate and efficient inference for transformers. Cryptology ePrint Archive (2023)

[14] Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 325–342 (2020)

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems 30 (2017)

[16] Wagh, S.: Pika: Secure computation using function secret sharing over rings. Proceedings on Privacy Enhancing Technologies (2022)

[17] Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. arXiv preprint arXiv:2004.02229 (2020)

[18] Yao, A.C.: Protocols for secure computations. In: 23rd annual symposium on foundations of computer science (sfcs 1982). pp. 160–164. IEEE (1982)

[19] Zhou, L., Wang, Z., Cui, H., Song, Q., Yu, Y.: Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 534–551. IEEE (2023)