PUMA: SECURE INFERENCE OF LLAMA-7B IN FIVE MINUTES

A PREPRINT

Ye Dong
Ant Group
dongye.dong@antgroup.com

Wen-jie Lu
Ant Group
juhou.lwj@antgroup.com

Yancheng Zheng
Ant Group
zhengyancheng.zyc@antgroup.com

Haoqi Wu
Ant Group
haoqi.whq@antgroup.com

Derun Zhao
Ant Group
zhaoderun.zdr@antgroup.com

Jin Tan
Ant Group
tanjin.tj@antgroup.com

Zhicong Huang
Ant Group
zhicong.hzc@antgroup.com

Cheng Hong
Ant Group
vince.hc@antgroup.com

Tao Wei
Ant Group
lenx.wei@antgroup.com

Wenguang Chen
Ant Group
yuanben.cwg@antgroup.com

July 25, 2023

ABSTRACT

With ChatGPT as a representative, tons of companies have began to provide services based on large Transformers models. However, using such a service inevitably leak users' prompts to the model provider. Previous studies have studied secure inference for Transformer models using secure multiparty computation (MPC), where model parameters and clients' prompts are kept secret. Despite this, these frameworks are still limited in terms of model performance, efficiency, and deployment. To address these limitations, we propose framework PUMA to enable fast and secure Transformer model inference. Our framework designs high quality approximations for expensive functions, such as GeLU and Softmax, which significantly reduce the cost of secure inference while preserving the model performance. Additionally, we design secure Embedding and LayerNorm procedures that faithfully implement the desired functionality without undermining the Transformer architecture. PUMA is about 2× faster than the state-of-the-art MPC framework MPCFORMER(ICLR 2023) and has similar accuracy as plaintext models without fine-tuning (which the previous works failed to achieve). One more thing, PUMA can evaluate LLaMA-7B in around 5 minutes to generate 1 token. To our best knowledge, this is the first time that a model with such a parameter size is able to be evaluated under MPC. PUMA has been open-sourced in the Github repository of SecretFlow-SPU[1].

Keywords Secret Sharing · Privacy · Security · Transformer

1 Introduction

Pre-trained Transformer models [Vaswani et al., 2017] have attracted much attentions for their high performance in practical tasks [Radford and Narasimhan, 2018, Zhuge et al., 2021] and been widely in Deep Learning as a Service (DLaaS) paradigm [Soifer et al., 2019]. However, these services can raise privacy concerns, such as in the case of

[1] https://github.com/secretflow/spu/blob/main/examples/python/ml/flax_llama7b/flax_llama7b.py

ChatGPT [Brown et al., 2020], which requires either users to reveal their private prompts to the service provider or the service provider to release their proprietary trained weights to users.

One solution to address the privacy concerns of Transformer models service is Secure Multi-Party Computation (MPC) [Shamir, 1979, Yao, 1986, Goldreich et al., 1987], which can keep data and model weights securely during inference. However, the vanilla Transformer inference in MPC is too time- and communication-expensive to be practical for real-world applications. To achieve better efficiency, existing work [Hao et al., 2022, Li et al., 2023, Akimoto et al., 2023, Liang et al., 2023, Liu and Liu, 2023] proposed various ways to speed up the secure inference of Transformer models, but these approaches still have one or several of the following drawbacks:

- **Rough Replacements.** Recently, several works [Li et al., 2023, Akimoto et al., 2023, Liu and Liu, 2023] have proposed using fast approximations such as quadratic and ReLU functions to replace expensive functions like GeLU and Softmax to reduce costs. However, simply replacing these functions can result in a significant decrease in Transformer model performance, which may require extra model retraining (fine-tuning) and lead to deployment obstacles.
- **High inference cost.** [Hao et al., 2022] adopted to approximate the expensive non-linear functions by using more accurate polynomials, but their approximation methods do not take the special properties of GeLU and Softmax into account. Therefore, their cost is still high after using approximations.
- **Not-Easy Deployment.** [Li et al., 2023, Liang et al., 2023] proposed to modify the architecture of Transformer models to accelerate secure inference, *e.g.*, decompose the embedding procedure and reorganize the linear layers. Worsely, as framework Crypten [Knott et al., 2021] does not support secure LayerNorm, [Li et al., 2023] only simulated the costs using BatchNorm, resulting in incorrect secure inference results. These modifications are in conflicts with existing plaintext Transformer systems.

To summarize, in the field of MPC Transformer inference, achieving both model performance and efficiency is challenging, and people may ask the following question:

Could pre-trained large transformer models be securely and efficiently evaluated with similar accuracy as in plaintext, without further retraining ?

To address this challenge, we propose the PUMA framework, which is a fast and accurate end-to-end secure Transformer inference framework. Our contributions can be summarized as follows:

- **New Approximations for Non-linear Functions.** We propose more accurate and faster approximations for the expensive non-linear functions (*e.g.*, GeLU and Softmax) in Transformer models. Different from existing works, we design the approximations based on the specialized properties of these non-linear functions to achieve both accuracy and efficiency.
- **Faster and More Accurate Secure Inference.** We make extensive experiments on 6 transformer models and 4 datasets, the results show that PUMA's precision is similar to plaintext ones' and is about 2× faster and more communication-efficient than MPCFORMER (note that MPCFORMER does not achieve similar precision as PUMA). PUMA can even evaluate LLaMA-7B in around 5 minutes to generate one word. To our best knowledge, this is the first time that such a large language model is able to be evaluated under MPC.
- **Open-sourced End-to-End Framework.** We design and implement the secure Embedding and LayerNorm procedures (which are lacked in other related works) faithfully in MPC. As a result, PUMA follows the workflow of plaintext Transformer models and does not change any model architecture, allowing loading and evaluating the pre-trained plaintext Transfomer models (*e.g.* downloaded from Hugging face) easily. To our best knowledge, PUMA is the first open-sourced MPC solution that supports accurate inference of pre-trained Transformer models without further modification efforts such as re-training.

Organization. We summarize the related work in § 2 and present the background in § 3. We give PUMA's high-level view and concrete design in § 4. We analyze the experimental results in § 5 and conclude this work in § 6.

2 Related Work

Transformer models have achieved remarkable success in language understanding [Radford and Narasimhan, 2018, Devlin et al., 2019, Yang et al., 2019, Touvron et al., 2023], vision understanding [Zhuge et al., 2021, Dong et al., 2022, Chen et al., 2021], and etc. Typically, Transformer models employ a two-stage training strategy: i) Transformer models are first pre-trained on a large dataset for general understanding, ii) and then fine-tuned [Sun et al., 2020] on a small downstream dataset to learn task-specific features, resulting in improved model performance. This training strategy has been proven to be effective in various settings and has become the dominant

paradigm [Radford and Narasimhan, 2018, Liu et al., 2019, Devlin et al., 2019]. In this work, we assume that model providers use pre-trained and fine-tuned Transformer models for online services.

Secure Multiparty Computation (MPC) [Shamir, 1979, Yao, 1986, Goldreich et al., 1987] enables distrusted parties to jointly compute a function with keeping their private inputs securely, and secure NN inference using MPC has gained much attention due its high privacy protection. These works operate in a variety of different models and architectures, including two-party setting [Mohassel and Zhang, 2017, Liu et al., 2017, Mishra et al., 2020, Huang et al., 2022, Patra et al., 2021, Rathee et al., 2020], three-party setting [Wagh et al., 2019, Mohassel and Rindal, 2018, Wagh et al., 2020, Kumar et al., 2019, Patra and Suresh, 2020, Tan et al., 2021, Dong et al., 2023], four-party setting [Byali et al., 2020, Dalskov et al., 2021], and etc [Braun et al., 2022]. Among these works, the three-party based approaches resisting semi-honest adversaries in honest majority has the highest concrete efficiency, and have gained much attention. However, most of these approaches only consider secure inference of convolutional/deep neural networks, and cannot be directly extended to support fast secure Transformer models inference. Recently, several research works [Hao et al., 2022, Li et al., 2023, Akimoto et al., 2023, Liang et al., 2023, Liu and Liu, 2023] have proposed MPC-based secure inference solutions for Transformer models. However, these approaches still have limitations in terms of model performance, efficiency, and deployment. Among these works, MPCFORMER [Li et al., 2023] is the only one that have been open-sourced, it is based on CrypTen [Knott et al., 2021] which is a three-party framework that uses a non-colluding third party to produce correlated randomness for the client and server. In this work, we mainly compare our proposed framework PUMA with MPCFORMER under the same three-party threat model.

3 Background

We first show the notations used in this paper as § 3.1. Then, we present the key building blocks of Transformer models in § 3.2. Finally, we give a brief introduction to 2-out-of-3 replicated secret sharing in § 3.3.

3.1 Notations

The main used notations are as follows: P_i represents the i-th computing party in 3PC, $i \in \{0, 1, 2\}$. The uppercase bold letter $\mathbf{X}$ is used for matrices, and the lowercase bold letter $\mathbf{x}$ denotes vectors. $\mathbf{x}[i]$ denotes the i-th element of vector $\mathbf{x}$, while lowercase letter x is used for scalar values. $\mathbb{Z}_{2^\ell}$ denotes the discrete ring modulo 2^ℓ, $\mathbb{R}$ denotes real numbers. $[\![\cdot]\!]$ is used for 2-out-of-3 replicated secret sharing [Araki et al., 2016, Mohassel and Rindal, 2018].

3.2 Transformer Model

A Transformer model [Vaswani et al., 2017] mainly consists of an **Embedding** layer and multiple **Transformer** layers. Given a token (*e.g.*, a word) id, the Embedding layer maps it to a hidden vector representation. And one Transformer layer includes **Attention**, **Feed-Forward Network**, and **LayerNorm** sub-layers:

Attention. Given inputs $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, the $\mathsf{Attention}$ function is computed as $\mathsf{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathsf{Softmax}(\mathbf{Q} \cdot \mathbf{K}^\mathsf{T} + \mathbf{M}_{\{0,-\infty\}}) \cdot \mathbf{V}$, where $\mathbf{M}_{\{0,-\infty\}}$, which is composed of $\{0, -\infty\}$, is used to perform *masking* $\mathsf{Attention}$ in Decoder, and it can be viewed as a bias matrix. Besides, [Vaswani et al., 2017] proposed Multi-Head $\mathsf{Attention}$ to jointly attend to information from different representation subspaces at different positions.

Feed-Forward Network (FFN). FFN is applied to each position separately and identically. This consists of two linear transformations with a activation in between, and the most common used activation function is GeLU. Given input $\mathbf{x}$ and parameters $\{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2\}$, FFN can be formalized as $\mathsf{FFN}(\mathbf{x}) = \mathbf{W}_2\mathsf{GeLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$. Note that the parameters of linear transformations are different from layer to layer.

LayerNorm. Given vector $\mathbf{x} \in \mathbb{R}^n$, $\mathsf{LayerNorm}$ is defined as: $\mathsf{LayerNorm}(\mathbf{x})[i] = \gamma \cdot \frac{\mathbf{x}[i]-\mu}{\sqrt{\sigma}} + \beta$, where (γ, β) are trained parameters, $\mu = \frac{\sum_{i=1}^n \mathbf{x}[i]}{n}$, and $\sigma = \sum_{i=1}^n (\mathbf{x}[i] - \mu)^2$.

Indeed, the Transformer model has evolved into various variants, each designed for specific tasks and domains. Two popular variants are Bert (Bidirectional Encoder Representations from Transformers) [Devlin et al., 2019] and GPT (Generative Pre-Trained models) [Radford and Narasimhan, 2018]. In these variants, the last layer of the Transformer model is often followed by a **Prediction** layer, also known as the task-specific layer. This additional layer is appended to the Transformer to generate the desired output for the specific task at hand.

3.3 2-out-of-3 Replicated Secret Sharing

Secret value $x \in \mathbb{Z}_{2^\ell}$ is shared by three random values $x_0, x_1, x_2 \in \mathbb{Z}_{2^\ell}$ with $x = x_0 + x_1 + x_2 \pmod{2^\ell}$. In 2-out-of-3 replicated secret sharing (denoted as $[\![\cdot]\!]$-sharing), party P_i gets $[\![x]\!]_i = (x_i, x_{i+1})$. Without special declaration, we compute in $\mathbb{Z}_{2^\ell}$ and omit $\pmod{2^\ell}$ for brevity. In the case of $\ell > 1$ (*e.g.*, $\ell = 64$) which support arithmetic operations (*e.g.*, $+$, $-$, and $\cdot$), we refer to this type as *Arithmetic Sharing* and use notation $[\![\cdot]\!]$. *Boolean Sharing* ($[\![\cdot]\!]^{\mathsf{B}}$) refers to $\ell = 1$ where $(+, -)$ and $\cdot$ are respectively replaced by bit-wise $\oplus$ and $\wedge$.

Addition. Let (c_1, c_2, c_3) be public constants, and $([\![x]\!], [\![y]\!])$ be two secret-shared values. Then, $[\![c_1 x + c_2 y + c_3]\!]$ can be computed as $(c_1 x_0 + c_2 y_0 + c_3, c_1 x_1 + c_2 y_1, c_1 x_2 + c_2 y_2)$ where P_i can compute its share locally. When $(c_1 = 1, c_2 = 1, c_3 = 0)$, we get $[\![x + y]\!]$.

Multiplication. In secure multiplication protocol Π_{Mul}, given two shared values $[\![x]\!]$ and $[\![y]\!]$, parties follows steps: i) First, P_i computes $z_i = x_i y_i + x_{i+1} y_i + x_i y_{i+1}$ locally, ii) Parties then perform *re-sharing* by letting P_i sends $z'_i = \alpha_i + z_i$ to P_{i-1}, where $\alpha_0 + \alpha_1 + \alpha_2 = 0$ (P_i can generate α_i in the setup phase as Mohassel and Rindal [2018]). iii) Finally, $\{(z'_0, z'_1), (z'_1, z'_2), (z'_2, z'_0)\}$ form $[\![x \cdot y]\!]$.

Underlying Protocols. In addition to addition and multiplication, PUMA relies on several other underlying protocols: boolean-arithmetic multiplication ($\Pi_{\mathsf{Mul_{BA}}}$), square Π_{Square}, equality test (Π_{Eq}), less than (Π_{LT}), reciprocal (Π_{Recip}), maximum (Π_{Max}), and reciprocal of square root (Π_{rSqrt}), from the state-of-the-art works. We employ them in a black-box manner, thus we only enumerate the inputs and outputs of these protocols as follows:

- $[\![z]\!] = \Pi_{\mathsf{Mul_{BA}}}([\![b]\!]^{\mathsf{B}}, [\![x]\!])$, s.t. $z = b \cdot x$
- $[\![z]\!] = \Pi_{\mathsf{Square}}([\![x]\!])$, s.t. $z = x^2$
- $[\![z]\!]^{\mathsf{B}} = \Pi_{\mathsf{Eq}}([\![x]\!], [\![y]\!])$, s.t. $z = 1\{x = y\}$
- $[\![z]\!]^{\mathsf{B}} = \Pi_{\mathsf{LT}}([\![x]\!], [\![y]\!])$, s.t. $z = 1\{x < y\}$
- $[\![z]\!] = \Pi_{\mathsf{Recip}}([\![x]\!])$, s.t. $z = 1/x$
- $[\![z]\!] = \Pi_{\mathsf{rSqrt}}([\![x]\!])$, s.t. $z = 1/\sqrt{x}$
- $[\![z]\!] = \Pi_{\mathsf{Max}}([\![\mathbf{x}]\!])$, s.t. $z = \mathsf{maximum}(\mathbf{x})$

$1\{e\}$ returns 1 that when condition e is true, and 0 otherwise. For detailed protocol constructions, please refer to [Mohassel and Rindal, 2018, Lu et al., 2020, Keller, 2020].

Fixed-Point Representation & Truncation. Real tasks (*e.g.*, Transformer models) usually use floating-point values, we need to encode the floating-point value as integers in finite rings/fields to support secret sharing-based MPC protocols [Mohassel and Rindal, 2018]. To avoid overflow in several sequential secure multiplications, [Mohassel and Rindal, 2018] proposed protocol Π^f_{Trunc} to truncate the least f bits securely. For ease of use, we include Π^f_{Trunc} in Π_{Mul} and Π_{Square} by default and and do not explicitly mention it in our protocol designs.

The above operations can be easily extended to vectors and matrices, and we use the same notation for vector and matrix operations for simplicity. For more details, please refer to [Mohassel and Rindal, 2018, Wagh et al., 2020].

Threat Model. Following previous works [Mohassel and Rindal, 2018, Li et al., 2023], PUMA resists a semi-honest (a.k.a., honest-but-curious) adversary in honest-majority [Lindell and Pinkas, 2009], where the adversary passively corrupts no more than one computing party. Such an adversary follows the protocol specification exactly, but may try to learn more information than permitted. Please note that PUMA cannot protect against the extraction of information from the inference results, and the examination of mitigating solutions (*e.g.*, differential privacy [Abadi et al., 2016]) falls outside the scope of this study.

4 Secure Design of PUMA

In this section, we first present an overview of PUMA in § 4.1. Then, we present the protocol for secure embedding in § 4.2. Next, we give our accurate approximation protocols for function GeLU and Softmax in § 4.3 and § 4.4, respectively. Finally, we show the secure design of LayerNorm in § 4.5 to reach framework PUMA.

4.1 Overview of PUMA

In PUMA, we aim to enable secure computation of Transformer-based models. To achieve this, the system defines three entities: model owner, client, and computing parties. The model owner provides the trained Transformer models, client is responsible for providing data to the system and receiving the inference results, while the computing parties (i.e., P_0, P_1, and P_2) execute the secure computation protocols. Note that the model owner and client can also be the computing parties, we describe them separately for ease of illustration.

During the secure inference process, a key invariant is maintained: the computing parties always start with 2-out-of-3 replicated secret shares of the clients' input and model owner's weights of the layer, and end with 2-out-of-3 replicated secret shares of layer's output. As the shares do not leak any information to each party, this ensures that the protocol modules can be sequentially combined for arbitrary depths to obtain a secure computation scheme for any Transformer-based model. The main focus of PUMA is to reduce the runtime and communication costs between the computing parties while maintaining the desired level of security. By leveraging replicated secret sharing and our 3PC protocols, PUMA enables fast secure inference of Transformer-based models in 3-party setting.

4.2 Protocol for Secure Embedding

The current secure embedding procedure [Li et al., 2023] requires the client to create a one-hot vector using the token id, which deviates from the plaintext workflow and undermines the Transformer structure. Therefore, this method is not easy to deploy in real Transformer models services applications.

To address this issue, we propose a secure embedding design as follows. Assuming that the token $\mathsf{id} \in [n]$ and all embedding vectors are denoted by $\mathbf{E} = (\mathbf{e}_1^T, \mathbf{e}_2^T, \ldots, \mathbf{e}_n^T)$, the embedding can be formulated as $\mathbf{e}_{\mathsf{id}} = \mathbf{E}[\mathsf{id}]$. Given $(\mathsf{id}, \mathbf{E})$ are in secret-shared fashion, our secure embedding protocol Π_{Embed} works as follows:

- The computing parties securely compute the one-hot vector $[\![\mathbf{o}]\!]^{\mathsf{B}}$ after receiving $[\![\mathsf{id}]\!]$ from the client. Specifically, $[\![\mathbf{o}[i]]\!]^{\mathsf{B}} = \Pi_{\mathsf{Eq}}(i, [\![\mathsf{id}]\!])$ for $i \in [n]$.
- The parties can compute the embedded vector via $[\![\mathbf{e}_{\mathsf{id}}]\!] = \Pi_{\mathsf{Mul_{BA}}}([\![\mathbf{E}]\!], [\![\mathbf{o}]\!]^{\mathsf{B}})$, where $\Pi_{\mathsf{Mul_{BA}}}$ does not require secure truncation.

In this way, our Π_{Embed} does not require explicit modification of the workflow of Transformer models.

4.3 Protocol for Secure GeLU

Most of the current approaches view the GeLU function as a composition of smaller functions and try to optimize each piece of them, making them to miss the chance of optimizing the private GeLU as a whole. Given the GeLU function:

$$\begin{aligned} \mathsf{GeLU}(x) &= \frac{x}{2} \cdot \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}} \cdot \left(x + 0.044715 \cdot x^3\right)\right)\right), \\ &\approx x \cdot \mathsf{sigmoid}(0.071355 \cdot x^3 + 1.595769 \cdot x) \end{aligned} \tag{1}$$

these approaches [Hao et al., 2022, Wang et al., 2022] focus either on designing efficient protocols for function $\tanh$ or using the existing MPC protocols of exponentiation and reciprocal for sigmoid.

However, none of current approaches have utilized the fact that GeLU function is almost linear on the two sides (*i.e.*, $\mathsf{GeLU}(x) \approx 0$ for $x < -4$ and $\mathsf{GeLU}(x) \approx x$ for $x > 3$). Within the short interval $[-4, 3]$ of GeLU, we suggest a piece-wise approximation of low-degree polynomials is a more efficient and easy-to-implement choice for its secure protocol. Concretely, our piece-wise low-degree polynomials are shown as equation (2):

$$\mathsf{GeLU}(x) = \begin{cases} 0, & x < -4 \\ F_0(x), & -4 \le x < -1.95 \\ F_1(x), & -1.95 \le x \le 3 \\ x, & x > 3 \end{cases}, \tag{2}$$

where polynomials $F_0()$ and $F_1()$ are computed by library numpy.ploytit[2] as equation (3). Surprsingly, the above simple poly fit works very well and our max error < 0.01403, median error $< 4.41e-05$, and mean error < 0.00168.

$$\begin{cases} F_0(x) & = -0.011034134030615728x^3 - 0.11807612951181953x^2 \\ & \quad -0.42226581151983866x - 0.5054031199708174 \\ F_1(x) & = 0.0018067462606141187x^6 - 0.037688200365904236x^4 \\ & \quad +0.3603292692789629x^2 + 0.5x + 0.008526321541038084 \end{cases} \tag{3}$$

Formally, given secret input $[\![x]\!]$, our secure GeLU protocol Π_{GeLU} is constructed as algorithm 1.

[2] https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html

Algorithm 1 Secure GeLU Protocol Π_{GeLU}

Input: P_i holds the 2-out-of-3 replicate secret share $[\![x]\!]_i$ for $i \in \{0,1,2\}$
Output: P_i gets the 2-out-of-3 replicate secret share $[\![y]\!]_i$ for $i \in \{0,1,2\}$, where $y = \mathsf{GeLU}(x)$.

1: P_0, P_1, and P_2 jointly compute

$$[\![b_0]\!]^{\mathsf{B}} = \Pi_{\mathsf{LT}}([\![x]\!], -4), \quad \triangleright\ b_0 = 1\{x < -4\}$$
$$[\![b_1]\!]^{\mathsf{B}} = \Pi_{\mathsf{LT}}([\![x]\!], -1.95), \quad \triangleright\ b_1 = 1\{x < -1.95\}$$
$$[\![b_2]\!]^{\mathsf{B}} = \Pi_{\mathsf{LT}}(3, [\![x]\!]), \quad \triangleright\ b_2 = 1\{3 < x\}$$

and compute $[\![z_0]\!]^{\mathsf{B}} = [\![b_0]\!]^{\mathsf{B}} \oplus [\![b_1]\!]^{\mathsf{B}}$, $[\![z_1]\!]^{\mathsf{B}} = [\![b_1]\!]^{\mathsf{B}} \oplus [\![b_2]\!]^{\mathsf{B}} \oplus 1$, and $[\![z_2]\!]^{\mathsf{B}} = [\![b_2]\!]^{\mathsf{B}}$. Note that $z_0 = 1\{-4 \le x < -1.95\}$, $z_1 = 1\{-1.95 \le x \le 3\}$, and $z_2 = 1\{x > 3\}$.

2: Jointly compute $[\![x^2]\!] = \Pi_{\mathsf{Square}}([\![x]\!])$, $[\![x^3]\!] = \Pi_{\mathsf{Mul}}([\![x]\!], [\![x^2]\!])$, $[\![x^4]\!] = \Pi_{\mathsf{Square}}([\![x^2]\!])$, and $[\![x^6]\!] = \Pi_{\mathsf{Square}}([\![x^3]\!])$.
3: Computing polynomials $[\![F_0(x)]\!]$ and $[\![F_1(x)]\!]$ based on $\{[\![x]\!], [\![x^2]\!], [\![x^3]\!], [\![x^4]\!], [\![x^6]\!]\}$ as equation (2) securely.
4: **return** $[\![y]\!] = \Pi_{\mathsf{Mul_{BA}}}([\![z_0]\!]^{\mathsf{B}}, [\![F_0(x)]\!]) + \Pi_{\mathsf{Mul_{BA}}}([\![z_1]\!]^{\mathsf{B}}, [\![F_1(x)]\!]) + \Pi_{\mathsf{Mul_{BA}}}([\![z_2]\!]^{\mathsf{B}}, [\![x]\!])$.

4.4 Protocol for Secure Softmax

In the function $\mathsf{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathsf{Softmax}(\mathbf{Q} \cdot \mathbf{K}^{\mathsf{T}} + \mathbf{M}) \cdot \mathbf{V}$, where $\mathbf{M}$ can be viewed as a bias matrix, the key challenge is computing function $\mathsf{Softmax}$. For the sake of numerical stability, the $\mathsf{Softmax}$ function is computed as

$$\mathsf{Softmax}(\mathbf{x})[i] = \frac{\exp(\mathbf{x}[i] - \bar{x} - \epsilon)}{\sum_i \exp(\mathbf{x}[i] - \bar{x} - \epsilon)}, \tag{4}$$

where $\bar{x}$ is the maximum element of the input vector $\mathbf{x}$. For the normal plaintext softmax, $\epsilon = 0$. For a two-dimension matrix, we apply equation (4) to each of its row vector.

Formally, our detailed secure protocol Π_{Softmax} is illustrated in algorithm 2, where we propose two optimizations:

- For the first optimization, we set ϵ in equation 4 to a tiny and positive value, e.g., $\epsilon = 10^{-6}$, so that the inputs to exponentiation in equation 4 are all negative. We exploit the negative operands for acceleration. Particularly, we compute the exponentiation using the Taylor series [Tan et al., 2021] with a simple clipping

$$\mathsf{negExp}(x) = \begin{cases} 0, & x < T_{\exp} \\ (1 + \frac{x}{2^t})^{2^t}, & x \in [T_{\exp}, 0]. \end{cases} \tag{5}$$

  Indeed, we apply the less-than for the branch $x < T_{\exp}$ The division by 2^t can be achieved using Π^t_{Trunc} since the input is already negative. Also, we can compute the power-of-2^t using t-step sequences of square function Π_{square} and Π^f_{Trunc}. Suppose our MPC program uses 18-bit fixed-point precision. Then we set $T_{\exp} = -14$ given $\exp(-14) < 2^{-18}$, and empirically set $t = 5$.
- Our second optimization is to reduce the number of divisions, which ultimately saves computation and communication costs. To achieve this, for a vector $\mathbf{x}$ of size n, we have replaced the operation $\mathsf{Div}(\mathbf{x}, \mathsf{Broadcast}(y))$ with $\mathbf{x} \cdot \mathsf{Broadcast}(\frac{1}{y})$, where $y = \sum_{i=1}^{n} \mathbf{x}[i]$. By making this replacement, we effectively reduce n divisions to just one reciprocal operation and n multiplications. This optimization is particularly beneficial in the case of the $\mathsf{Softmax}$ operation. The $\frac{1}{y}$ in the $\mathsf{Softmax}$ operation is still large enough to maintain sufficient accuracy under fixed-point values. As a result, this optimization can significantly reduce the computational and communication costs while still providing accurate results.

4.5 Protocol for Secure LayerNorm

Recall that given a vector $\mathbf{x}$ of size n, $\mathsf{LayerNorm}(\mathbf{x})[i] = \gamma \cdot \frac{\mathbf{x}[i]-\mu}{\sqrt{\sigma}} + \beta$, where (γ, β) are trained parameters, $\mu = \frac{\sum_{i=1}^{n} \mathbf{x}[i]}{n}$, and $\sigma = \sum_{i=1}^{n} (\mathbf{x}[i] - \mu)^2$. In MPC, the key challenge is the evaluation of the divide-square-root $\frac{\mathbf{x}[i]-\mu}{\sqrt{\sigma}}$ formula. To securely evaluate this formula, CrypTen sequentially executes the MPC protocols of square-root, reciprocal, and multiplication. However, we observe that $\frac{\mathbf{x}[i]-\mu}{\sqrt{\sigma}}$ is equal to $(\mathbf{x}[i] - \mu) \cdot \sigma^{-1/2}$. And in the MPC side, the costs of computing the inverse-square-root $\sigma^{-1/2}$ is similar to that of the square-root operation [Lu et al., 2020]. Besides, inspired by the second optimization of § 4.4, we can first compute $\sigma^{-1/2}$ and then $\mathsf{Broadcast}(\sigma^{-1/2})$ to support fast and secure $\mathsf{LayerNorm}(\mathbf{x})$. And our formal protocol $\Pi_{\mathsf{LayerNorm}}$ is shown in algorithm 3.

Algorithm 2 Secure Softmax Protocol Π_{Softmax}

Input: P_i holds the 2-out-of-3 replicate secret share $[\![\mathbf{x}]\!]_i$ for $i \in \{0,1,2\}$, and $\mathbf{x}$ is a vector of size n.
Output: P_i gets the 2-out-of-3 replicate secret share $[\![\mathbf{y}]\!]_i$ for $i \in \{0,1,2\}$, where $\mathbf{y} = \mathsf{Softmax}(\mathbf{x})$.

P_0, P_1, and P_2 jointly compute $[\![\mathbf{b}]\!]^{\mathsf{B}} = \Pi_{\mathsf{LT}}(T_{\exp}, [\![\mathbf{x}]\!])$ and the maximum $[\![\bar{x}]\!] = \Pi_{\mathsf{Max}}([\![\mathbf{x}]\!])$.
Parties locally computes $[\![\hat{\mathbf{x}}]\!] = [\![\mathbf{x}]\!] - [\![\bar{x}]\!] - \epsilon$, and jointly compute $[\![\mathbf{z}_0]\!] = 1 + \Pi^t_{\mathsf{Trunc}}([\![\hat{\mathbf{x}}]\!])$.
for $j = 1, 2, \ldots, t$ **do**
    $[\![\mathbf{z}_j]\!] = \Pi_{\mathsf{Square}}([\![\mathbf{z}_{j-1}]\!])$.
end for
Parties locally compute $[\![z]\!] = \sum_{i=1}^{n} [\![\mathbf{z}[i]]\!]$ and jointly compute $[\![1/z]\!] = \Pi_{\mathsf{Recip}}([\![z]\!])$.
Parties jointly compute $[\![\mathbf{z}/z]\!] = \Pi_{\mathsf{Mul}}([\![\mathbf{z}]\!], [\![1/z]\!])$
return $[\![\mathbf{y}]\!] = \Pi_{\mathsf{Mul_{BA}}}([\![\mathbf{b}]\!]^{\mathsf{B}}, [\![\mathbf{z}/z]\!])$.

Algorithm 3 Secure LayerNorm Protocol $\Pi_{\mathsf{LayerNorm}}$

Input: P_i holds the 2-out-of-3 replicate secret share $[\![\mathbf{x}]\!]_i$ for $i \in \{0,1,2\}$, and $\mathbf{x}$ is a vector of size n.
Output: P_i gets the 2-out-of-3 replicate secret share $[\![\mathbf{y}]\!]_i$ for $i \in \{0,1,2\}$, where $\mathbf{y} = \mathsf{LayerNorm}(\mathbf{x})$.

P_0, P_1, and P_2 compute $[\![\mu]\!] = \frac{1}{n} \cdot \sum_{i=1}^{n} [\![\mathbf{x}[i]]\!]$ and $[\![\sigma]\!] = \sum_{i=1}^{n} \Pi_{\mathsf{Square}}([\![\mathbf{x}]\!] - [\![\mu]\!])[i]$.
Parties jointly compute $[\![\sigma^{-1/2}]\!] = \Pi_{\mathsf{rSqrt}}([\![\sigma]\!])$.
Parties jointly compute $[\![\mathbf{c}]\!] = \Pi_{\mathsf{Mul}}(([\![\mathbf{x}]\!] - [\![\mu]\!]), [\![\sigma^{-1/2}]\!])$
return $[\![\mathbf{y}]\!] = \Pi_{\mathsf{Mul}}([\![\gamma]\!], [\![\mathbf{c}]\!]) + [\![\beta]\!]$.

5 Experimental Evaluations

Implementation. We implement PUMA on top of SecretFlow [Ma et al., 2023] in C++ and Python. SecretFlow compiles a high-level Flax code to secure computation protocols, which are then executed by our designed cryptographic backends, and we encode the floating-ponit values as 64-bit integers in ring $\mathbb{Z}_{2^{64}}$ with 18-bit fractional part. Our experiments are run on 3 Alibaba Cloud ecs.g7.8xlarge servers with 32 vCPU and 128GB RAM each. The CPU model is Intel Xeon(Ice Lake) Platinum 8369B CPU @ 2.70GHz. We evaluate PUMA on Ubuntu 20.04.6 LTS with Linux kernel 5.4.0-144-generic. Our bandwidth is about 5GB/s and round trip time is about 1ms.

Models & Datasets. We evaluate PUMA on seven NLP models: Bert-Base, Roberta-Base, and Bert-Large [Devlin et al., 2019]; GPT2-Base, GPT2-Medium, and GPT2-Large [Radford and Narasimhan, 2018]; and LLaMA-7B [Touvron et al., 2023]. We measure the Bert performance for three NLP tasks over the datasets of Corpus of Linguistic Acceptability (CoLA), Recognizing Textual Entailment (RTE), Stanford Question Answering Dataset (QNLI) from GLUE benchmarks [Wang et al., 2019], and GPT2 performance on Wikitext-103 V1 [Merity et al., 2016].

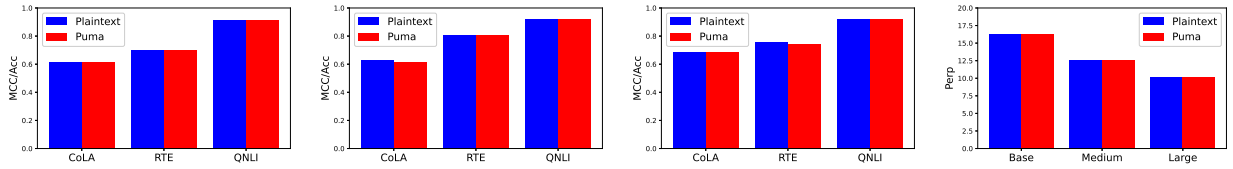
Baseline. We compare PUMA to the most similar prior work MPCFORMER [Li et al., 2023]. But for fair comparison, we have the following considerations: i) As MPCFORMER neither supports loading pretrained transformer models nor implements LayerNorm faithfully[3], we cannot achieve meaningful secure inference results using their framework. Therefore, we compare our secure Transformer models inference performance to that of plaintext (floating-point) to show our precision guarantee. ii) MPCFORMER with *Quad* approximations (for both GeLU and Softmax) requires retraining the modified models. As PUMA does not require retraining, we compare our cost to that of MPCFORMER without *Quad* approximations. Also, we re-run MPCFORMER in our environment.

5.1 Precision

We compare our secure model inference performance to that of plaintext (floating-point) in Figure 1 to show our precision guarantee.

In Figure 1(a)-1(c), we show the Matthews correlation/accuracy of plaintext and PUMA on the Bert-Base, Roberta-base, and Bert-Large. We observe that the accuracy achieved by PUMA matches the accuracy of the plaintext Flax code. Specifically, the accuracy difference does not exceed 0.011 over all datasets.

[3] As MPCFORMER does not support loading pre-trained Transformer models, we did an experiment in plaintext Bert-Base that replaced LayerNorm with BatchNorm as MPCFORMER did. This resulted in a significant drop in the MCC score for CoLA task from 0.616 to −0.020. On the contrary, PUMA achieves an MCC score of 0.613.



(a) (b) (c) (d)

Figure 1: Performance on GLUE benchmark (1(a) is for Bert-Base, 1(b) is for Roberta-Base, and 1(c) is for Bert-Large) and Wikitext-103 V1 (1(d), including GPT2-Base, GPT2-Medium, and GPT2-Large). We select CoLA, RTE, and QNLI of GLUE, Matthews correlation (MCC) is reported for CoLA and Accuracy (Acc) is reported for RTE and and QNLI. Perplexity (Prep) is reported for Wikitext-103 V1.

Table 1: Costs of Bert-Base, Roberta-Base, and Bert-Large for one input sentence with a length of 128. Time is in seconds and Communication (Comm. for short) is in GB, which is the same for the following tables.

Model	Bert-Base		Roberta-Base		Bert-Large	
Costs	Time	Comm.	Time	Comm.	Time	Comm.
MPCFORMER	85.887	9.673	86.798	9.957	227.076	26.380
PUMA	42.525	3.591	55.046	3.821	100.424	9.082
Improv.	2.020×	2.694×	1.577×	2.606×	2.261×	2.905×

Moreover, in Figure 1(d), we also compare our perplexity on dataset Wikitext-103 V1 with the plaintext baseline on models GPT2-Base, GPT2-Medium, and GPT2-Large. The results are similar and the perplexity differences do not exceed 0.02 over all models.

The above accuracy and perplexity advantages experimentally validate that our protocols are numerically precise.

5.2 Inference cost

In this subsection, we compare PUMA's inference cost to that of MPCFORMER. We evaluate three Bert models (Bert-Base, Roberta-Base, and Bert-Large) and three GPT2 models (GPT2-Base, GPT2-Medium, and GPT2-Large). The costs are for processing one input sentence: i) For Bert models the input sentence is of length 128. ii) GPT2 models input one length-32 sentence and generate 1 new word.

On the 3 Bert models in Table 1, PUMA is $1.577 \sim 2.261\times$ faster than MPCFORMER, and is $2.606 \sim 2.905\times$ more communication-efficient. For the GPT2 models in Table 2, PUMA is $1.457 \sim 2.244\times$ faster than MPCFORMER, and is $3.590 \sim 5.082\times$ more communication-efficient.

We observe that PUMA's improvements increase as the model size grows, particularly for the GPT2 models. This trend is because our specialized optimizations are more effective when processing large-scale evaluations.

5.3 Scalability

In this subsection, we measure the costs of evaluating PUMA on Bert-Base and GPT2-Base models for batched inputs, varying-length inputs, and varying-length outputs (only for GPT2-Base). We also compare our costs to those of MPCFORMER to demonstrate our improvements.

Batch Inputs Evaluation. Table 3 presents our costs on batched inputs. For Bert-Base, PUMA is $1.871 \sim 2.037\times$ faster and achieves $2.276 \sim 2.499\times$ reduction in communication costs. For GPT2-Base, our improvements in runtime and communication costs are in the range of $1.176 \sim 1.499\times$ and $1.377 \sim 2.406\times$ respectively. Unlike the observa-

Table 2: Costs of GPT2-Base, GPT2-Medium, and GPT2-Large. The input sentence is of length 32, all of the costs are for generating 1 token.

Model	GPT2-Base		GPT2-Medium		GPT2-Large	
Costs	Time	Comm.	Time	Comm.	Time	Comm.
MPCFORMER	42.615	4.516	102.022	10.633	193.544	20.245
PUMA	29.248	1.258	56.322	2.353	86.263	3.984
Improv.	1.457×	3.590×	1.811×	4.519×	2.244×	5.082×

Table 3: Costs of Bert-Base and GPT2-Base for a batch of $\{2, 4, 8, 16\}$ sentences. The input lengths for Bert-Base and GPT2-Base are respectively set as 128 and 32, and the costs of GPT2 are for generating 1 token.

	#Batch	2		4		8		16	
	Costs	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.
Bert	MPCFORMER	153.921	17.747	312.025	33.897	616.090	66.196	1245.631	130.793
	PUMA	78.547	7.102	153.180	14.363	307.757	28.586	665.875	57.457
	Improv.	1.960×	2.499×	2.037×	2.360×	2.002×	2.316×	1.871×	2.276×
GPT2	MPCFORMER	57.312	6.040	87.818	9.089	149.617	15.186	292.452	27.380
	PUMA	38.234	2.510	65.444	5.038	122.505	10.000	248.789	19.882
	Improv.	1.499×	2.406×	1.342×	1.804×	1.221×	1.519×	1.176×	1.377×

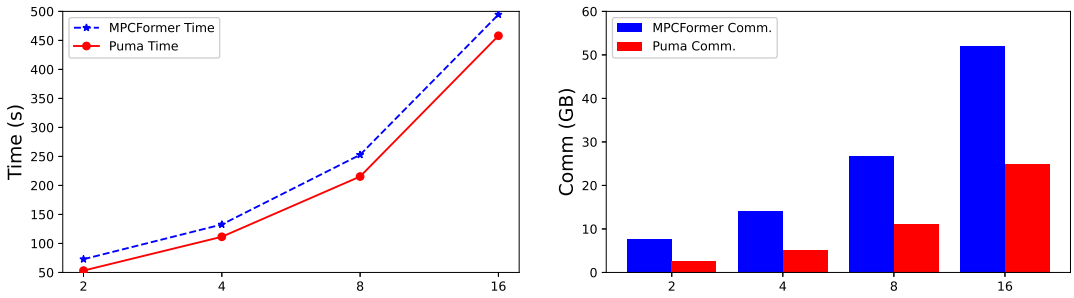
Table 4: Costs of Bert-Base and GPT2-Base for different input length (denoted as #Input). The input lengths for Bert-Base and GPT2-Base are respective $\{64, 128, 256, 512\}$ and $\{16, 32, 64, 128\}$. GPT2-Base generates 1 token.

	#Input	64/16		128/32		256/64		512/128	
	Costs	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.
Bert	MPCFORMER	46.428	4.750	85.887	9.673	196.372	23.443	582.787	68.069
	PUMA	24.345	1.627	42.525	3.591	87.561	8.668	212.600	23.439
	Improv.	1.907×	2.919×	2.020×	2.694×	2.243×	2.705×	2.741×	2.904
GPT2	MPCFORMER	34.522	3.767	42.615	4.516	60.451	6.281	105.028	11.225
	PUMA	20.692	0.625	29.248	1.258	40.968	2.607	74.529	5.611
	Improv.	1.668×	6.027×	1.457×	3.590×	1.476×	2.409×	1.409×	2.001×

tions in Tables 1 and 2, our efficiency gains decrease with increasing batch sizes. This phenomenon is attributed to the interesting fact: To directly support pre-trained plaintext models, PUMA strictly follows the plaintext model format that only accept token ids as input, so PUMA has to compute the one-hot vectors from token ids in an MPC way. On the other hand, MPCFORMER uses modified models that accept one-hot vectors as input, so the one-hot function could be computed at the client side in plaintext. Nevertheless, PUMA remains more efficient than MPCFORMER.

Input Length Evaluation. Table 4 shows our costs on varying-length inputs, we evaluate Bert-Base on the inputs of length $\{64, 128, 256, 512\}$, and GPT2-Base on the inputs of length $\{16, 32, 64, 128\}$. For Bert-Base, PUMA is $1.907 \sim 2.741\times$ faster and achieves $2.694 \sim 2.919\times$ reduction in communication costs. For GPT2-Base, our improvements in runtime and communication costs range from $1.409 \sim 1.668\times$ and $2.001 \sim 6.027\times$ respectively.

Output Length Evaluation. Fig 2 presents our costs on varying-length outputs for GPT2-Base, and compares our costs to those of MPCFORMER. Our improvements in runtime and communication costs range from $1.080 \sim 1.369\times$ and $2.086 \sim 3.011\times$ respectively. As we generate more output tokens, our efficiency gains decrease, this is because each output token must be input back into the model to generate the next token, increasing the required one-hot embedding costs. We should emphasize again that although the time costs might be close for long outputs, PUMA could achieve a similar accuracy as plaintext models while MPCFORMER could not.



(a) (b)

Figure 2: Costs of GPT2-Base for generating different output tokens, the input length is of length 32. Fig 2(a) is for runtime and 2(b) is for communication costs.

Table 5: Costs of the secure inference of LLaMA-7B, #Input denotes the length of input sentence and #Output denotes the number of generated tokens.

(#Input, #Output)	(4, 1)		(8, 1)		(8, 2)	
Costs	Time	Comm.	Time	Comm.	Time	Comm.
PUMA	263.453	0.930	346.126	1.865	660.881	3.927

Scale to LLaMA-7B in Five Minutes. We evaluated the large language model LLaMA-7B using PUMA under 3 Alibaba Cloud ecs.r7.32xlarge servers, each has 128 threads and 1TB RAM, with 20GB bandwidth, 0.06ms round-trip-time. As shown in Table 5, PUMA can support the secure inference of large language model LLaMA-7B with reasonable costs. For example, given an input sentence of 8 tokens, PUMA can output one token in around 346.126 seconds with communication costs of 1.865 GB. To our knowledge, this is the first time that LLaMA-7B has been evaluated using MPC.

6 Conclusion

We propose an efficient MPC framework PUMA for secure inference on Transformer models based on replicated secret sharing. To reduce the costs of secure inference, we approximate expensive functions with accurate polynomials and propose secure Embedding and LayerNorm protocols to support end-to-end secure inference. Although the inference cost is still quite high, we successfully make it one step closer to solving users' privacy concerns in Transformer-based DLaaS. We believe that by combining PUMA with quantization methods and hardware accelerations in the future, secure inference of large Transformer models in seconds is no longer impossible.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

Mingchen Zhuge, Dehong Gao, Deng-Ping Fan, Linbo Jin, Ben Chen, Haoming Zhou, Minghui Qiu, and Ling Shao. Kaleido-bert: Vision-language pre-training on fashion domain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12647–12657, 2021.

Jonathan Soifer, Jason Li, Mingqin Li, Jeffrey Zhu, Yingnan Li, Yuxiong He, Elton Zheng, Adi Oltean, Maya Mosyak, Chris Barnes, et al. Deep learning inference service at microsoft. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, pages 15–17, 2019.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

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

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

O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912217.

Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=deyqjpcTfsG`.

Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. MPCFORMER: FAST, PERFORMANT AND PRIVATE TRANSFORMER INFERENCE WITH MPC. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=CWmvjOEhgH-`.

Y. Akimoto, K. Fukuchi, Y. Akimoto, and J. Sakuma. Privformer: Privacy-preserving transformer with mpc. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroSP)*, pages 392–410, Los Alamitos, CA, USA, 2023. IEEE Computer Society. doi:10.1109/EuroSP57164.2023.00031. URL https://doi.ieeecomputersociety.org/10.1109/EuroSP57164.2023.00031.

Zi Liang, Pinghui Wang, Ruofei Zhang, Lifeng Xing, Nuo Xu, and Shuo Zhang. Merge: Fast private text generation, 2023.

Xuanqi Liu and Zhuotao Liu. Llms can understand encrypted prompt: Towards privacy-computing friendly transformers, 2023.

B. Knott, S. Venkataraman, A.Y. Hannun, S. Sengupta, M. Ibrahim, and L.J.P. van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *arXiv 2109.00984*, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Xiaoyi Dong, Jianmin Bao, Ting Zhang, Dongdong Chen, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, and Nenghai Yu. Bootstrapped masked autoencoders for vision bert pretraining. In *European Conference on Computer Vision*, pages 247–264. Springer, 2022.

Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12299–12310, 2021.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification?, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2505–2522, 2020.

Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure Two-Party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1.

Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. {ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2165–2182, 2021.

Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370899. URL https://doi.org/10.1145/3372297.3417274.

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

Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, page 35–52, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356930. doi:10.1145/3243734.3243760. URL https://doi.org/10.1145/3243734.3243760.

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.

Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. *arXiv preprint arXiv:1909.07814*, 2019.

Arpita Patra and Ajith Suresh. Blaze: blazing fast privacy-preserving machine learning. *arXiv preprint arXiv:2005.09042*, 2020.

Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. *arXiv preprint arXiv:2104.10949*, 2021.

Ye Dong, Chen Xiaojun, Weizhan Jing, Li Kaiyun, and Weiping Wang. Meteor: Improved secure 3-party neural network inference with reducing online communication costs. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 2087–2098, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394161.

Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. Flash: Fast and robust framework for privacy-preserving machine learning. *Proc. Priv. Enhancing Technol.*, 2020(2):459–480, 2020.

Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. Motion–a framework for mixed-protocol multi-party computation. *ACM Transactions on Privacy and Security*, 25(2):1–35, 2022.

Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817, 2016.

Wen-jie Lu, Yixuan Fang, Zhicong Huang, Cheng Hong, Chaochao Chen, Hunter Qu, Yajin Zhou, and Kui Ren. Faster secure multiparty computation of adaptive gradient descent. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, PPMLP'20, page 47–49, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380881.

Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1575–1590, 2020.

Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of cryptology*, 22(2):161–188, 2009.

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

Yongqin Wang, G. Edward Suh, Wenjie Xiong, Benjamin Lefaudeux, Brian Knott, Murali Annavaram, and Hsien-Hsin S. Lee. Characterization of mpc-based private inference for transformer-based models. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 187–197, 2022. doi:10.1109/ISPASS55109.2022.00025.

Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 17–33, Boston, MA, July 2023. USENIX Association. ISBN 978-1-939133-35-9. URL `https://www.usenix.org/conference/atc23/presentation/ma`.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJ4km2R5t7`.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

A Details of Experimental Models

In this section, we present the architecture of the experimental models in brief. For more details, please refer to HuggingFace Transformers library [Wolf et al., 2020].

- Bert-Base: Bert-Base is the base version of the Bert model and consists of 12 Transformer encoder layers, 768 hidden size, and 12 heads. It has 110 million parameters and is trained on a large corpus of unlabeled text data.
- Roberta-Base: Similar to Bert-base, Roberta-base is a base version of the Roberta model. It comprises 12 Transformer layers, 768 hidden size, and 12 heads. It has around 125 million parameters.
- Bert-Large: Bert-Large is an extended version of Bert-base with 24 Transformer encoder layers, 1024 hidden size, and 16 heads. It has approximately 340 million parameters, making it more powerful and capable of capturing complex language patterns.
- GPT2-Base: GPT2-Base is the base version of the Gpt2 model and consists of 12 Transformer decoder layers, 768 hidden size, and 12 heads. It has 117 million parameters and is trained on a large corpus of text data. GPT2-Base is mainly used for tasks involving text generation and language understanding.
- GPT2-Medium: GPT2-Medium comprises 24 Transformer decoder layers, 1024 hidden size, and 16 heads. And it has approximately 345 million parameters.
- GPT2-Large: GPT2-Large is the largest variant of the GPT2 model, featuring 36 Transformer decoder layers, 1280 hidden size, and 16 heads. It approximately 774 million parameters.