LLMs Can Understand Encrypted Prompt: Towards Privacy-Computing Friendly Transformers

Xuanqi Liu *
Tsinghua University
lxq22@mails.tsinghua.edu.cn

Zhuotao Liu †
Tsinghua University
zhuotaoliu@tsinghua.edu.cn

Abstract

The community explored to build private inference frameworks for transformer-based large language models (LLMs) in a server-client setting, where the server holds the model parameters and the client inputs its private data (or prompt) for inference. However, these frameworks impose significant overhead when the private inputs are forward propagated through the original LLMs. In this paper, we show that substituting the computation- and communication-heavy operators in the transformer architecture with privacy-computing friendly approximations can greatly reduce the private inference costs while incurring very minor impact on model performance. Compared to state-of-the-art Iron (NeurIPS 2022), our privacy-computing friendly model inference pipeline achieves a 5× acceleration in computation and an 80% reduction in communication overhead, while retaining nearly identical accuracy.

1 Introduction

Large language models (LLMs) attracted significant attentions, driven by advances in artificial intelligence and the availability of large amounts of training data [30, 4]. LLMs are trained on massive datasets of text and code, and can be used for a variety of tasks, including generating text, translating languages, writing different kinds of creative content, and answering questions in an informative way.

Nowadays, LLMs are usually provided as online inference services. This, however, raises serious privacy concerns. On the one hand, the client's input (prompt, such as questions and requirements) must be submitted in plaintext to the service provider. In certain use cases, the prompt could contain sensitive information that the client would like to hide from the service provider. The growing concern for privacy, particularly in the age of Web3.0 [26], and the enactment of privacy protection laws such as the GDPR, necessitate that privacy be a top priority when offering online LLM services. On the other hand, the LLMs hosted by the service provider are proprietary, so it is critical to ensure that an adversarial client cannot obtain the model parameters during inference.

The private inference paradigm of neural networks has recently emerged as a solution to the aforementioned problem [8, 17, 24, 28, 14, 11, 32]. In this paradigm, the client submits an encrypted version of its input and works collaboratively with the service provider to obtain an encrypted inference result that can only be recovered by the client itself. The service provider cannot obtain any private information about the input. However, the efficiency of the private inference, especially on large neural networks, is extremely limited by the extensive use of Homomorphic Encryption (HE) and Secure Multiparty Computation (MPC) primitives on various expensive neural network operators.

Two recent art [14] and [11] demonstrate the possibility of private inference on popular neural networks (*e.g.,* convolutional networks and transformers) in computer vision and natural language

*Work partially supported by an internship program funded by Sudo Privacy.
†Corresponding author.

processing, respectively. We observe that the time and communication cost of private inference on LLMs consisting of multiple transformer blocks is much higher than that of the traditional convolutional networks For instance, even on a model as small as BERT-Tiny [3], [11] takes ∼50 seconds and 2GB of communication for a single inference, while Cheetah [14] can scale to ResNet-32 [12] with 15 seconds and 0.11 GB communication for one inference. This difference is because transformers use sophisticated nonlinear functions that are *computationally-unfriendly* to the cryptographic primitives. For instance, convolutional networks use ReLU [10] and batch normalization [15], while transformers prefer GELU [13], meanwhile extensively use softmax function and layer normalization techniques [1][3]. Through experiments, we show that these functions take up to more than 70% of the total cost for private inference on transformers.

This paper explores to improve the efficiency of private inference on transformer-based models. To this end, we first build a private inference system that fully supports the private computations required for transformer-based LLMs. We then conduct extensive experiments to identify the critical performance bottleneck. Based on this, we design various substitutions for these bottlenecked components, and use fine-tuning to retain model performance after replacing these components. Taken together, we build an effective system to provide LLM inference service while fully protecting the privacy of the input data. We perform extensive evaluations and show that applying our privacy-computing friendly operators in LLMs can reduce $\sim 80\%$ of the overall private inference time, while retaining nearly identical model accuracy.

1.1 Related Work

Private inference of neural networks was first proposed by [8]. It demonstrates the feasibility of fully using Homomorphic Encryption (HE) to achieve non-interactive private inference. However, due to the linearity restriction of HE, every nonlinear function such as ReLU and MaxPooling must be replaced by linear or polynomial approximation. Works after [8] primarily sought to use Secure Multiparty Computation (MPC) to deal with the nonlinear functions, and exploit the single instruction multiple data (SIMD) property of HE to accelerate the inference [17, 28, 32]. A recent art Cheetah [14] proposes a special encoding method to encode vectors and matrices into HE polynomials, which achieves state-of-the-art performance in computing matrix-vector multiplication and convolutions. Iron [11] realizes that matrix-matrix multiplication (rather than matrix-vector multiplication) dominates in transformer-based inference, and therefore improves the vanilla polynomial encoding by introducing a blocking method that prioritizes the batch dimension. Despite the optimization, some of the non-linear functions (*e.g.,* GELU, softmax and layer normalization layers) are fundamentally expensive in private inference. For instance, Iron [11] reports that running a single inference on BERT-Tiny [3] requires 50 seconds time and 2GB transmission. Two recent studies explore replacing these fundamentally expensive non-linear functions with operators that are more friendly in private inference. For instance, Chen et al. [5] use ReLU to substitute all non-linearities in a transformer, and relying on HE for linear operations. However, their architecture requires the ReLU functions to be executed *in plaintext by the client*, which may reveal the proprietary model owned by the server. Li et al. [21], on the other hand, use quadratic polynomial approximations for GELU and softmax. Yet, they rely on Trusted Third Party (TTP) to produce correlated randomness for MPC. This is inappropriate in practice because designing and certifying a TTP is an open problem.

2 Preliminaries

2.1 Transformer Architecture

Transformers dominate the model architecture in the area of natural language processing since its birth [37]. The state-of-the-art large language models (LLMs) typically consist of an embedding layer, a transformer encoder stack with n identical encoder layers, and a downstream task sub-model (a classifier model for predicting labels or a generative model for predicting the next token) [7, 25, 4, 31, 30]. In this paper, for simplicity we ignore the embedding layer (*i.e.,* both the server and the client could produce the embeddings with respect to some input sentence), and focus on private inference on the transformer encoder stack and the downstream sub-model.

[3] During inference, batch normalization does not compute data statistics, but layer normalization does.

One transformer encoder layer consists of two main parts: the multihead-attention and the feed-forward layer. A residual structure and layer normalization layer are inserted after both parts. Formally, for the input x to go through one transformer encoder layer:

$$\begin{aligned} x_1 &= \mathsf{LayerNorm}_1(x + \mathsf{MultiheadAttention}(x)) \\ y &= \mathsf{LayerNorm}_2(x_1 + \mathsf{FeedForward}(x_1)) \end{aligned} \tag{1}$$

The multihead-attention consists of an input projection (a fully connected layer), a softmax function and an output projection (also an FC), while the feed-forward layer consists of two fully connected layers and an activation function between them (usually GELU).

Thus, a private inference system on transformer-based models should support forward propagation of fully connected layers, softmax function, GELU function, and layer normalization with private input.

2.2 Cryptography Primitives

To realize a private inference system for LLMs, we mainly use two cryptographic primitives.

Homomorphic Encryption. Homomorphic encryption supports computation (addition and multiplication) over ciphertexts. We use the BFV fully homomorphic encryption cryptosystem based on the RLWE problem with residual number system (RNS) optimization [9, 2]. Specifically, the BFV scheme is constructed with a set of parameters $\{N, t, q\}$ such that the polynomial degree N is a power of two, and t, q represent plaintext and ciphertext modulus, respectively. We let t be chosen as a power of two, 2^ℓ. The plaintext space is the polynomial ring $\mathcal{R}_{t,N} = \mathbb{Z}_t[X]/(X^N + 1)$ and the ciphertext space is $\mathcal{R}^2_{q,N}$. Homomorphism is established on the integer polynomial ring $\mathcal{R}_{t,N}$, supporting addition and multiplication of polynomials in the encrypted domain. We denote the homomorphically encrypted ciphertext of x as $[\![x]\!]$.

Secure Multiparty Computation. We utilize additive secret-sharing scheme upon the field $\mathbb{F} = \mathbb{Z}_t$ (integers modulo t) with $t = 2^\ell$, where an integer $x \in \mathbb{F}$ is shared between a pair of client and server (*i.e.,* $x = \langle x \rangle_0 + \langle x \rangle_1$) [33]. LLMs typically involves decimal numbers rather than integers. To adapt to the BFV scheme and integer-based secret sharing, we use a fixed-point representation of decimal numbers [32, 14]. A decimal $\tilde{x} \in \mathbb{R}$ is represented as an integer $x = \mathsf{Encode}(\tilde{x}) = \lfloor \tilde{x} \cdot 2^f \rfloor \in \mathbb{Z}$, with a precision of f bits. After every multiplication, the precision inflates to $2f$, and a truncation is required to keep the original precision. Since we use $\mathbb{Z}_t$ rather than $\mathbb{Z}$, we require all intermediate results in their decimal form $\tilde{x} \in \mathbb{R}$ not to exceed $\pm t/2^{2f}$, to prevent overflow. In the rest of the paper, unless stated otherwise, all scalars and elements of tensors are in $\mathbb{F} = \mathbb{Z}_t$.

2.3 Threat Model

We consider a semi-honest threat model including two parties: a server holding all the model weights, and a client holding the inference input data. The model architecture is public. The two parties adhere to the protocols but are curious about the private information held by the other party (*i.e.,* the model weights and inference inputs).

3 Approach

We first build a fully functional framework for private inference of transformers and LLMs based on transformers, including all building blocks such as FC layers, ReLU, GELU activation functions, etc. We run real-world models within the framework and measure the inference cost of each kind of operation to determine the bottleneck of the end-to-end inference pipeline. Then, we transform these computationally and communication heavy layers or functions into cryptography-friendly ones, and fine-tune the model to retain the model accuracy during substitution. Finally, we test the post-tuned models with our private inference framework to evaluate their inference performance.

3.1 Private Transformer Inference

We use the secret-sharing form of all intermediate outputs throughout the private inference procedure to protect the privacy of both the inputs and the model weights. Concretely, we treat every neural network operator $y_i = f_i(x_i)$ as a 2-party computation protocol, which takes secret-shares $x_i = \langle x_i \rangle_0 + \langle x_i \rangle_1$ from the two parties as input ant returns the secret-shares of $y_i = \langle y_i \rangle_0 + \langle y_i \rangle_1$ to them.

Algorithm 1: Private matrix multiplication protocol Π_{MatMul}

Input: The server inputs $\mathbf{A} \in \mathbb{R}_{m\times r}$ and the client inputs $\mathbf{B} \in \mathbb{R}_{r\times n}$, $mrn \leq N$, N being the BFV polynomial degree.

Output: The two parties receives the secret shares of $\mathbf{C} = \mathbf{AB}$.

1 Server and client respectively encode $\mathbf{A} = (a_{ij}), \mathbf{B} = (b_{jk})$ into polynomial $a = \pi_A(\mathbf{A}), b = \pi_B(\mathbf{B})$:

$$a = \sum_{i=0}^{m-1}\sum_{j=0}^{r-1} a_{ij}x^{ir+r-1-j}, b = \sum_{j=0}^{r-1}\sum_{k=0}^{n-1} b_{jk}x^{kmr+j}$$

2 Client encrypts the polynomial b and sends $[\![b]\!]$ to server.

3 Server use HE to evaluate $[\![c]\!] = a \cdot [\![b]\!]$, and samples a random polynomial $\langle c\rangle_1 = s$. Server sends $[\![\langle c\rangle_0]\!] = [\![c]\!] - s$ to client for decryption.

4 Server and client respectively output $\langle \mathbf{C}\rangle_i = \pi_C^{-1}(\langle c\rangle_i)$, where the decoding method π_C^{-1} for $c = \sum_{i=0}^{N-1} c_i x^i$ is: (note that only a part of the coefficients in c is used)

$$\mathbf{C} = \pi_C^{-1}(c) = (c_{kmr+ir+r-1})_{ik}.$$

All operators in the transformers could be divided into two categories: (1) linear operators (*e.g.,* fully connected layers); (2) non-linear operators (*e.g.,* GELU, Softmax function). We do not consider the residual structure as a single operator as it is simply an addition of secret shares.

3.1.1 Linear Operators

The core linear protocol for private inference over linear layers is the matrix multiplication protocol. It is realized using homomorphic encryption, with the polynomial encoding primitive first proposed by [14] and extended by [11]. We start with a simple situation where one party holds $\mathbf{A}$ and the other party holds $\mathbf{B}$. The protocol $\Pi_{\mathsf{MatMul}}(\mathbf{A}, \mathbf{B})$ takes the inputs from the two parties and outputs the secret shares $\langle \mathbf{C}\rangle_0, \langle \mathbf{C}\rangle_1$ such that $\mathbf{C} = \mathbf{AB}$. We suppose the input matrices are small enough to be encoded into one plaintext polynomial, as larger matrices could be split into smaller blocks to adapt the protocol. We summarize this protocol Π_{MatMul} in Algorithm 1.

Fully connected layer. In fully connected layers ($\mathbf{y} = f(\mathbf{x}) = \mathbf{Wx} + \mathbf{b}$), we can directly use Π_{MatMul} in the forward propagation:

- Suppose the client holds $\langle \mathbf{x}\rangle_0$, while the server holds $\langle \mathbf{x}\rangle_1$, the weights $\mathbf{W}$ and bias $\mathbf{b}$.
- The two parties invoke $\Pi_{\mathsf{MatMul}}(\mathbf{W}, \langle \mathbf{x}\rangle_0)$ to produce $\langle \mathbf{W}\langle \mathbf{x}\rangle_0\rangle_0$ and $\langle \mathbf{W}\langle \mathbf{x}\rangle_0\rangle_1$.
- The client outputs $\langle \mathbf{y}\rangle_0 = \langle \mathbf{W}\langle \mathbf{x}\rangle_0\rangle_0$, and the server outputs $\langle \mathbf{y}\rangle_1 = \langle \mathbf{W}\langle \mathbf{x}\rangle_0\rangle_1 + \mathbf{W}\langle \mathbf{x}\rangle_1 + \mathbf{b}$.

Attention. In multihead attention, however, we need to calculate the multiplication of two secret-shared matrices (*i.e.,* $\mathbf{QK}^T$ and $\mathbf{AV}$, for $\mathbf{A} = \mathsf{Softmax}(\mathbf{QK}^T/\sqrt{E})$). The key observation is that we need only to calculate the secret-shares of the "cross terms", by invoking the Π_{MatMul} protocol twice. Suppose we need to compute $\mathbf{Z} = \mathbf{XY}$, with both the input matrices secret-shared. The two parties perform the following procedure:

- The two parties invoke $\Pi_{\mathsf{MatMul}}(\langle \mathbf{X}\rangle_1, \langle \mathbf{Y}\rangle_0)$ and $\Pi_{\mathsf{MatMul}}(\langle \mathbf{X}\rangle_0, \langle \mathbf{Y}\rangle_1)$[4] and add their results, so that they obtain the secret shares of the cross terms:
$$\langle \mathbf{Z}_{\mathsf{cross}}\rangle_0 + \langle \mathbf{Z}_{\mathsf{cross}}\rangle_1 = \mathbf{Z}_{\mathsf{cross}} = \langle \mathbf{X}\rangle_1\langle \mathbf{Y}\rangle_0 + \langle \mathbf{X}\rangle_0\langle \mathbf{Y}\rangle_1 \quad (2)$$
- The client outputs $\langle \mathbf{Z}\rangle_0 = \langle \mathbf{Z}_{\mathsf{cross}}\rangle_0 + \langle \mathbf{X}\rangle_0\langle \mathbf{Y}\rangle_0$; The server outputs $\langle \mathbf{Z}\rangle_1 = \langle \mathbf{Z}_{\mathsf{cross}}\rangle_1 + \langle \mathbf{X}\rangle_1\langle \mathbf{Y}\rangle_1$.

3.1.2 Non-linear Operators

For the non-linear operators, we mainly use several primitives provided by [14, 32] libraries, which rely on the oblivious transfer cryptographic primitive. Recall that each operator takes as input secret-shares, and output secret-shares to the two parties. We use these primitives as black boxes in our system:

[4]For clarity, we semantically abuse the notation: to make sure that the client encrypts/decrypts and the server performs HE operations, the correct form should be $\Pi_{\mathsf{MatMul}}(\langle \mathbf{Y}^T\rangle_1, \langle \mathbf{X}^T\rangle_0)$, and the two parties transposes the result after the invocation.

- Π_{ReLU}: $\mathsf{ReLU}(x) = \max\{x, 0\}$.
- Π_{EleMul}: field element multiplication $x \cdot y$.
- Π_{max}: $\max(\mathbf{x}) = \max\{x_i | \mathbf{x} = (x_i)\}$.
- Π_{exp}: $\mathsf{exp}(x) = e^x$, for $x \leq 0$.
- Π_{recip}: $\mathsf{recip}(x) = 1/x$.
- Π_{rSqrt}: $\mathsf{rSqrt}(x) = 1/\sqrt{x}$, for $x > 0$.
- Π_{tanh}: $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$.

We now discuss the private inference procedure for each kind of non-linearity in the transformer architecture.

GELU. GELU is an activation function commonly used in transformers:

$$\mathsf{GELU}(x) = 0.5x \cdot \left[1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right]$$

To produce $\mathsf{GELU}(x)$ for secret shared $\langle x \rangle$, the two parties invoke $\Pi_{\mathsf{EleMul}}(\langle x \rangle, \langle x \rangle)$ to produce $\langle x^2 \rangle$, and again invoke $\Pi_{\mathsf{EleMul}}(\langle x \rangle, \langle x^2 \rangle)$ to produce $\langle x^3 \rangle$. Addition and multiplication by scalar are performed subsequently before invoking Π_{tanh} on $\langle \sqrt{2/\pi}(x + 0.044715x^3) \rangle$. Finally, they again invoke once Π_{EleMul} to obtain the final result $\langle \mathsf{GELU}(x) \rangle$.

Softmax. Softmax is a key operator in scaled-dot attention construction. It is applied to the attention scores as a non-linearity and normalization to put more verbosity into the model. For a vector $\mathbf{x} = (x_0, \cdots, x_{n-1})$,

$$\mathsf{Softmax}(\mathbf{x}) = \left(e^{x_i} / \textstyle\sum_{j=0}^{n-1} e^{x_j}\right)_{i \in [0,n)}$$

To compute the softmax function, the two parties first invoke $\Pi_{\mathsf{max}}(\langle \mathbf{x} \rangle)$ to obtain $\langle x_{\max} \rangle$. They substract the original vector by the maximum value to ensure the inputs to $\Pi_{\mathsf{exp}}(\langle \mathbf{x} - x_{\max} \rangle)$ is negative. The exponentiated results are summed and used in Π_{recip} to produce the denominator in softmax, and finally they call Π_{EleMul} to obtain $\langle \mathsf{Softmax}(\mathbf{x}) \rangle$.

LayerNorm. Layer normalization is an operator used to limit the bound of the layer outputs of self attention and feed forward sub-networks. It first calculate the mean and variance (standard deviation) along the embedding dimension and normalizes the input with these statistics, and then perform a learnable affine projection (with parameters γ, β) to produce the output (ϵ is a small term to prevent division by zero):

$$\mathsf{LayerNorm}(\mathbf{x}) = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\mathrm{Var}(\mathbf{x}) + \epsilon}} \cdot \gamma + \beta$$

For calculating the normalized values, the two parties invoke the Π_{recip} and Π_{rSqrt} protocols, and for the affine projection, they invoke Π_{EleMul} to produce the outputs.

3.1.3 Other Optimizations

Reducing the communication cost of matrix multiplication. We observe that in the BFV homomorphic encryption system, the core of decryption could be rendered as computing $m = s \cdot c_1 + c_0$, where c_0, c_1, s are all polynomials, and $c = (c_0, c_1) \in \mathcal{R}_{q,N}^2$ is the HE ciphertext, s the secret key. This form of decryption indicates that, to obtain each coefficient m_i in m there is only one coefficient $(c_0)_i$ required in b but all coefficients in c_1 are needed. In Step 4 of Algorithm 1, to decipher $\mathbf{C}$, only part of the cofficients of c are required. Therefore, we could omit transmitting the other coefficients in the c_0 part of the ciphertexts. Since the required coefficients are very sparse in matrix multiplication, this roughly reduces half of the server-to-client communication for linear operators.

Hardware parallelization of HE operations. In RLWE-based HE cryptosystem, the operations are essentially done on the polynomial ring. We observe that the addition and multiplication of polynomials could be effectively parallelized.[5] Therefore, we implement a GPU-version of the BFV cryptosystem, including homomorphic addition and cipher-plain multiplication, to further accelerate linear evaluations.

3.2 Identifying Performance Bottleneck

Given the inference framework, we run private inference on the transformer-based language models to identify the performance bottleneck. As an example, we experiment with the BERT-Tiny [3, 36] model with embedding dimension $E = 128$, consisting of $n = 2$ transformer encoder blocks. The maximum sequence length is set to 128, with shorter sentences padded. As a common practice, the

[5]Multiplication of polynomials is done by number theory transform (NTT) and elementwise multiplication.

Operator	Time	Comm.
MatMul	3.75s	111MB
Softmax	5.95s	518MB
GELU	3.67s	1020MB
LayerNorm	0.62s	165MB
Total	13.99s	1814MB

Table 1: Inference cost of various operators used in BERT-Tiny

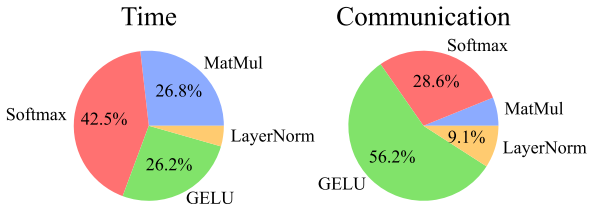

Figure 1: Ratio of various operators' cost

Algorithm 2: Substitution workflow

Input: An original transformer model $\mathcal{M}$ with n encoder blocks $\{m_i\}_{i\in[n]}$; target original operator type t, and the replaced operator type t'; acceptable accuracy drop $\Delta\alpha$

Output: The replaced model $\mathcal{M}'$

Evaluate the original model: $\alpha = \mathsf{Eval}(\mathcal{M})$ and set $\mathcal{M}_{\mathsf{accept}} \leftarrow \mathcal{M}$
for $i = n-1, n-2, \cdots 0$ **do**
  $\mathcal{M}_{\mathsf{temp}} \leftarrow$ substitute t_i in the i-th block of $\mathcal{M}_{\mathsf{accept}}$ with t'_i.
  Finetune $\mathcal{M}_{\mathsf{temp}}$ with the parameters of block 0 to $i-1$ fixed.
  Evaluate $\mathcal{M}_{\mathsf{temp}}$: $\alpha_i = \mathsf{Eval}(\mathcal{M}_{\mathsf{temp}})$.
  if $\alpha_i > \alpha - \Delta\alpha$ **then** $\mathcal{M}_{\mathsf{accept}} \leftarrow \mathcal{M}_{\mathsf{temp}}$.
end
Output $\mathcal{M}_{\mathsf{accept}}$.

embedding dimension of each attention head is 64 (2 heads for $E = 128$), and the hidden dimension of the feed forward block is $4E$. The time and communication costs are summarized in Table 1 and Figure 1.

These results indicate that the non-linear functions (GeLU, Softmax and LayerNorm) consume a significant portion of time and communication cost in the privacy inference pipeline, indicating that these non-linearities are *not privacy-computing friendly*. For example, compared with ReLU activation function, GELU requires four evaluation of element-wise multiplication, and a computation-heavy $\tanh$ function based on look-up tables. Thus, it is critical to substitute these operators for privacy-computing friendly ones. Yet, retaining the model accuracy after applying alternative operators is non-trivial. In the following subsection, we elaborate on an automatic substitution workflow.

3.3 Exploring Privacy-computing Friendly Transformers

In order to accelerate the private inference of transformers, the server substitutes the operators in its model with privacy-computing friendly alternatives and fine-tune the model to adapt to the replacement. Specifically, it replaces the GELU, Softmax and LayerNorm operators layer by layer, and test the model accuracy of each replacement after fine-tuning. The modification is accepted if the accuracy drop is within a predetermined threshold, or reverted otherwise.

3.3.1 Substitution Workflow

We introduce a workflow to substitute the privacy-computing unfriendly operators in the transformer architecture. As introduced in preliminaries, the transformer-based language models typically include an encoder stack consisting of n blocks of *transformer encoder block* (in some literature called *layers*), and the construction of each block is the same. Directly replacing all undesired operators with alternatives would greatly harm the model accuracy performance. Thus, we design a workflow to gradually substitute these operators layer by layer, from the last block to the first one. The model is fine-tuned between every substitution to make sure the model can adapt to the change. Denote the evaluate function for model $\mathcal{M}$ as $\mathsf{Eval}(\mathcal{M})$ (*i.e.,* testing the model on the validation data set) The workflow is summarized in Algorithm 2.

Bound Controlling. In the private inference framework, we adapt fixed-point decimals, where each real number is encoded into a integer with a scale of 2^f. If the total bit-length is ℓ, we have a plaintext space of $\ell - 2f$ bits, because after each multiplication the scale will grow to 2^{2f}. Our preliminary

experiments show that when the encoder stack consists of many transformer encoder blocks, the absolute bound of the intermediate hidden states becomes larger after each block. As a result, the private inference procedure will encounter overflow in the secret-shares, producing meaningless prediction results.

To address this issue, rather than directly using division to control the bounds, we modify the fine-tuning process to be aware of bound controlling. Specifically, we set an acceptable bound B, and add a loss term to penalize the hidden states with absolute values greater than B. Suppose for some sample $\mathbf{x}$, the hidden states of the transformer blocks are $\mathbf{h}_i, i = 0, \cdots, n-1$. We design the loss function with three terms to be minimized during fine-tuning:

$$\mathcal{L} = (1 - \alpha_1 - \alpha_2)\mathcal{L}_{\text{task}} + \alpha_1 \mathcal{L}_{\text{decay}} + \alpha_2 \mathcal{L}_{\text{bound}} \tag{3}$$

where $\mathcal{L}_{\text{task}}$ is the loss (*e.g.,* cross entropy) function for the downstream task, $\mathcal{L}_{\text{decay}}$ is the weight decay term against overfitting, and

$$\mathcal{L}_{\text{bound}} = \sum_{i=0}^{n-1} ||\max\{|\mathbf{h}_i| - B, 0\}||_2$$

is a bounding term that penalizes the values too great. $|\mathbf{h}_i|$ is taking the absolute value, and $||\cdot||_2$ is the L2-norm. Note that we do not directly set $\mathcal{L}_{\text{bound}}$ as the L2-norm of the hidden state, because we do not wish the values to be *as small as possible*, but only require them to be *lower than some bound* B. If the hidden states' values were too small, the relative error due to fixed-point approximation would become too large.

3.3.2 Substitution Strategy

As linear projection and ReLU activation function are more friendly to privacy computing, we mainly use these two components and their combination as our substitution candidates.

LayerNorm. The expensive part of the layer normalization operator is division operation of the standard deviation. Intuitively, the mean value is substracted to keep the intermediate activations centralized, and the deviation division is to keep them bounded. The average value may vary greatly across different samples, but the standard deviation (or the bound) can be captured by the affine transformation $\hat{\mathbf{x}} \cdot \gamma + \beta$. Based on this insight, we remove the standard-deviation calculation part and only keep the centralization and affine transformation to be fine-tuned:

$$\mathsf{LayerNorm}'(\mathbf{x}) = (\mathbf{x} - \bar{\mathbf{x}}) \cdot \gamma + \beta$$

Softmax. Replacing softmax function is more challenging. As the attention mask is added to the attention scores before the softmax, and the attention mask may contain $-\infty$ values, we cannot directly use a linear transformation (∞ values would contaminate the intermediate results). Therefore, we combine the relatively cheap function ReLU to eliminate them. Furthermore, we notice that softmax function ensures that the output summed along the last dimmension is one. To simulate this feature, we divide the ReLU'ed result by their sum to obtain the output. Formally, for input $\mathbf{x} = (x_0, \cdots, x_{n-1})$ (ϵ is a small constant to prevent division by zero):

$$\mathsf{Softmax}'(\mathbf{x}) = (\mathsf{ReLU}(x_i)/\textstyle\sum_{j=0}^{n-1} \mathsf{ReLU}(x_j) + \epsilon)_{i \in [0,n)}$$

Since softmax function itself is not trainable, whenever we substitute softmax function with the simplified version, we treat the input and output projections of the related multi-head attention as the trainable parameters in Algorithm 2. With this approach, the whole model can adapt to the softmax replacements faster.

GELU. GELU is the most expensive operator in the private inference pipeline. It is surprising that they could be simply replaced with ReLU, with nearly no accuracy drop in model performance.

4 Evaluation

We first evaluate the accuracies of our substituted models. Then we execute the entire private inference pipeline to measure the end-to-end privacy inference performance.

4.1 Experimental Setup

Models and datasets. We test our substitution strategies and the private inference framework with three models: BERT-Tiny, BERT-Medium [3, 36] and RoBERTa-Base [25] (referred to as

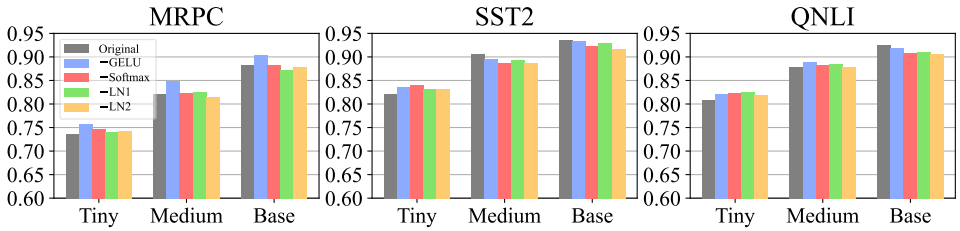

Figure 2: Accuracies of the model before and after each substitution. We use − mark to denote which operators have been replaced. The changes are done incrementally. For example, "−Softmax" means both GELU and Softmax are replaced with privacy-computing friendly alternatives.

Tiny, Medium and Base hereafter). These three models have similar architectures, only differing in hyperparameters (number of transformer encoder blocks $n = 2, 8, 12$ and embedding dimensions $E = 128, 512, 768$ respectively). During training, we limit the sequence length to 512 tokens, while in inference we use a shorter 128 length for efficiency. We use the MPRC, SST-2 and QNLI subsets of the GLUE benchmark [38] to evaluate the accuracy performance of fine-tuned models.

Implementation. For plaintext fine-tuning, we implemented the substitution and training procedure with the Huggingface's `transformers` and the widely-applied `pytorch` libraries. For private inference, our 2-party interactive framework is build upon HE and MPC primitives. For the HE part we implement a GPU version of the BFV scheme [9, 2]. We set polynomial degree $N = 8192$, with q set to ~ 180 bits. For the MPC primitives, we directly use the funtionalities provided by two open-sourced library: OpenCheetah [14] and SCI [32]. The plaintext and secret-sharing modulus $t = 2^{41}$, and bit precision $f = 13$. We combine the HE and MPC cryptographic primitives using a high-level framework written in `python`, where we build different types of neural network layers as independent modules and provide end-to-end private inference interfaces. We evaluate our framework on a physical machine with Intel Xeon Gold 6230R CPU and NVIDIA RTX A6000 GPU (CUDA version 11.7).

4.2 Operator Substitution

We suspect that the more layers we substitute, the worse the model accuracy. Therefore, to largely retain the model performance, we substitute the layers starting from the most expensive ones to the least expensive ones. Specifically, denote the two layer normalizations in each block as LN1 and LN2, we first substitute all GELUs, then softmaxs, then LN1s and finally LN2s.

To prevent a too strict bound from impacting the fine-tuning accuracy, we introduce gradually decreasing controlling bound (Section 3.3.1) in these four substitutions: we set the acceptable bound $B = +\infty$ (*i.e.,* no bound controlling) when replacing GELUs; $B = 32$ replacing softmaxes; $B = 24$ replacing LN1s, and finally $B = 16$ replacing LN2s. This setting allows the model to gradually adapt to smaller values of intermediate hidden states in the fine-tuning procedure. We set the maximum acceptable accuracy drop to $\Delta\alpha = 2\%$. In the loss function (Eq. 3), we set $\alpha_1 = 0.1$ and $\alpha_2 = 0.2$.

After each substitution, we test the accuracy of the model. The results are shown in Figure 2. Overall, we observe that our substitution strategies can successfully replace all the GELU and softmax functions in every model, with accuracy drop of $\Delta\alpha < 2\%$. But only part of the LN layers in larger models can be replaced without significant accuracy drop. We notice that earlier LN layers are more important. For example, in the MRPC task, substituting the later 10 LN2s of BERT-Medium only accuracy by $< 1\%$. Yet once the first 2 layers of BERT-Medium is changed, the model cannot converge . Interestingly, we observe that using ReLU instead of GELU sometimes results in better accuracy than the original models. This might be because that GELU is more helpful when training (possibly unsupervised) from scratch due to its non-zero differentials in the negative domain, yet the complexity of GELU may not be necessary when fine-tuning for a downstream task.

4.3 End-to-end Performance

We measured the runtime and communication cost of the end-to-end private inference on our privacy-computing friendly models. We first report the cost of *one single* encoder layer. The results are shown

Model	Cost	Iron [11]	Orig.	−GE.	−Sm.	−LN1	Ours −LN2	Improvement
BERT -Tiny	Time	26.24	13.89	8.75	3.35	3.10	2.86	9.2×
	Comm.	1.07	1.77	0.78	0.33	0.27	0.22	4.9×
BERT -Medium	Time	108.53	60.99	42.16	19.84	19.35	19.08	5.7×
	Comm.	4.23	7.03	3.06	1.25	1.05	0.85	5.0×
RoBERTa -Base	Time	168.43	84.50	59.19	35.54	34.72	35.01	4.8×
	Comm.	6.38	9.51	3.55	1.74	1.44	1.14	5.6×

Table 2: Private inference costs on our privacy-computing friendly models. Time costs are in seconds, and communication costs are in GB. "Orig.", "GE." and "Sm." stands for Original, GELU and Softmax.

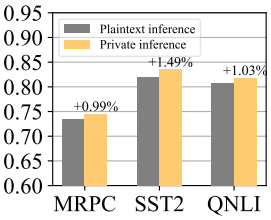

Figure 3: End-to-end model accuracy of private inference.

in Table 2. Since the GELU and softmax are the most expensive operators in the privacy inference, replacing these two operators with alternative operators results in 3× speedup and reduction of 80% communication cost. Furthermore, when the LN layers are replaced with affine transformations, the communication costs are further reduced to 13% of the original model. We also compared the efficiency of our implementation to Iron [11], where our approach showed state-of-the-art runtime and communication cost in private inference of LLMs, outperforming Iron by five times in runtime and communication efficiency. Note that when using the original model, our communication costs are slightly greater than Iron's because we use larger parameters in HE and MPC to retain high precision in large models.

Further, we report the end-to-end accuracy of private inference on BERT-Tiny for the three datasets. As shown in Figure 3, the private inference on our private-computing friendly model achieves slightly better accuracies compared to plaintext inference on the original model. This result is consistent with Figure 2, where our modified BERT-Tiny performs slightly better than the original model on plaintext inference.

5 Discussion

Model Pruning and Knowledge Distillation. The overhead of private inference is still much higher than plaintext inference, and the overhead is proportional to the model size (number of transformer encoder blocks and embedding dimension). Thus, it might be beneficial to consider model pruning and knowledge distillation [27, 20, 34] to reduce the scale of the model while retaining comparable model accuracy.

Hardware Assist in MPC. Hardware acceleration is widely adopted in the field of machine learning. In this work, we have explored the feasibility of using parallel hardware to accelerate HE operations. Prior works proposed to utilize similar parallelization techniques to accelerate MPC primitives [19, 35]. Incorporating these techniques might further reduce the inference costs.

Trusted Hardware. To further accelerate cryptography operations, it is possible to rely on trusted hardware. Prior works proposed to generate related randomness (*e.g.,* distributing beaver triples, random oblivious transfers) [19, 21] using application-specific trusted hardware or generic Trusted Execution Environment (*e.g.,* Intel SGX) [41, 40]. These trusted hardware essentially serves as a Trusted Third Party (TTP). In future work, we will explore how TTP may further accelerate private inference in the case where the introduction of a TTP is acceptable by all stakeholders.

Malicious Security. We currently build our private inference framework with passively secure protocols that protect privacy only from semi-honest adversaries. To strengthen our framework to malicious security, one could apply extra consistency checking protocols (*e.g.,* [18, 39] uses extra checks to ensure the fundamental oblivious transfers are generated correctly). Additionally, the advances in zero-knowledge proof (ZKP) technology, especially the development of zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK), sparked significant research on how to force desired participant behavior by requiring the participants to prove their behavior in a zero-knowledge and efficient manner using zero-knowledge proving systems. For instance, [6] and [23] apply ZKP over extension fields and large prime fields respectively to guarantee the faithful execution of the multiplication protocol. martFL [22] ensures that the training process of Federated Learning is fair to data trading.

6 Conclusion

We propose an efficient framework for private inference on large language models (LLMs). Homomorphic encryption (HE) and secure multi-party computation (MPC) are used respectively for linear and non-linear operators. We observe that the privacy-computing unfriendly operators are the performance bottleneck, and substituting them with privacy-computing friendly alternatives brings 5x acceleration and 80% reduction of communication costs while retaining model accuracies. We hope this work will shed light on a practical way of adapting LLMs to offer privacy-preserving inference service.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Jean Claude Bajard, Julien Eynard, M. Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. pages 423–442, 10 2017.

[3] Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. Generalization in nli: Ways (not) to go beyond simple heuristics, 2021.

[4] 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.

[5] Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. THE-X: Privacy-preserving transformer inference with homomorphic encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3510–3520, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[6] Ronald Cramer, Ivan Damgård, Daniel E. Escudero, Peter Scholl, and Chaoping Xing. Spdz2k: Efficient mpc mod 2 for dishonest majority. 2022.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

[9] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

[10] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975.

[11] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 15718–15731. Curran Associates, Inc., 2022.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[14] 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.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[16] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 145–161, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

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

[18] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure ot extension with optimal overhead. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 724–741, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[19] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021.

[20] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[21] Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. Mpcformer: fast, performant and private transformer inference with mpc. *arXiv preprint arXiv:2211.01452*, 2022.

[22] Qi Li, Zhuotao Liu, Qi Li, and Ke Xu. martFL: Enabling Utility-Driven Data Marketplace with a Robust and Verifiable Federated Learning Architecture. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[23] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5377–5394, Anaheim, CA, August 2023. USENIX Association.

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

[25] 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. *arXiv preprint arXiv:1907.11692*, 2019.

[26] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, Qi Li, and Yih-Chun Hu. Make Web3. 0 Connected. *IEEE transactions on dependable and secure computing*, 2022.

[27] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

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

[29] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[30] OpenAI. Gpt-4 technical report, 2023.

[31] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

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

[34] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China, November 2019. Association for Computational Linguistics.

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

[36] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962, 2019.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[38] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[39] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated ot with small communication. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1607–1626, New York, NY, USA, 2020. Association for Computing Machinery.

[40] Wei Zheng, Ying Wu, Xiaoxue Wu, Chen Feng, Yulei Sui, Xiapu Luo, and Yajin Zhou. A survey of intel sgx and its applications. *Frontiers of Computer Science*, 15(3):153808, Dec 2020.

[41] Xing Zhou, Zhilei Xu, Cong Wang, and Mingyu Gao. Ppmlac: High performance chipset architecture for secure multi-party computation. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 87–101, New York, NY, USA, 2022. Association for Computing Machinery.

7 Supplementary Material

7.1 Formal Description of Threat Model

We provide a formal description of the threat model of two semi-honest parties in the private inference framework.

Definition 1. *A protocol Π between a server holding model weights $\mathbf{W}$ (the model architecture $\mathcal{M}$ is public) and a client holding inference data $\mathbf{x}$ is a **private inference protocol** if it satisfies the following guarantees.*

- ***Correctness.*** *On every set of model weights $\mathbf{W}$ and every input data $\mathbf{x}$, the output of the client is a prediction result $\mathbf{y}$ produced by correctly doing neural network inference on $\mathbf{x}$,* i.e., *$\mathbf{y} = \mathcal{M}(\mathbf{x}; \mathbf{W})$, and the output of server is $\perp$.*
- ***Security.***
  - *(**Data privacy**) A corrupted, semi-honest server does not learn anything useful about the client's inference data $\mathbf{x}$. Formally, we require the existence of an efficient simulator Sim_S such that $\mathsf{View}_S^{\Pi} \approx_c \mathsf{Sim}_S(\mathbf{W}, \perp)$, where View_S^{Π} denotes the view of server in the execution of Π.*

Algorithm 3: GELU evaluation on secret shares

Input: The client inputs $\langle x \rangle_0$ and the server inputs $\langle x \rangle_1$, where $x = \langle x \rangle_0 + \langle x \rangle_1$.
Output: The two parties receives the secret shares of $y = \mathsf{ReLU}(x)$.

1 The two parties invoke $\Pi_{\mathsf{EleMul}}(\langle x \rangle, \langle x \rangle)$, followed by a truncation (see Section 2.2), to produce shares $\langle x^2 \rangle$.
2 The two parties invoke $\Pi_{\mathsf{EleMul}}(\langle x \rangle, \langle x^2 \rangle)$, followed by a truncation to produce shares $\langle x^3 \rangle$.
3 The two parties multiply shares $\langle x^3 \rangle$ locally with $\lfloor 0.044715 \cdot 2^f \rceil$, and invoke a truncation to obtain $\langle 0.044715 x^3 \rangle$.
4 The two parties add locally the shares $\langle x \rangle$ and $\langle 0.044715 x^3 \rangle$, and multiply the sum with $\lfloor \sqrt{2/\pi} \cdot 2^f \rceil$, and truncates the product to obtain $\langle x' \rangle = \langle \sqrt{2/\pi}(x + 0.044715 x^3) \rangle$.
5 The two parties invoke Π_{tanh} to obtain $\langle \tanh(x') \rangle$.
6 The server adds 2^f to its share $\langle \tanh(x') \rangle_1$. This step semantically produces the shares of $\langle 1 + \tanh(x') \rangle$.
7 The two parties invoke $\Pi_{\mathsf{EleMul}}(\langle x \rangle, \langle 1 + \tanh(x') \rangle)$, and truncates by $f + 1$ bits (because of the 0.5 term of ReLU), producing $\langle \mathsf{ReLU}(x) \rangle$.

Algorithm 4: Softmax evaluation on secret shares

Input: The client inputs $\langle \mathbf{x} \rangle_0$ and the server inputs $\langle \mathbf{x} \rangle_1$, where $\mathbf{x} = \langle \mathbf{x} \rangle_0 + \langle \mathbf{x} \rangle_1$.
Output: The two parties receives the secret shares of $\mathbf{y} = \mathsf{Softmax}(\mathbf{x})$.

1 The two parties invoke $\Pi_{\mathsf{max}}(\langle \mathbf{x} \rangle)$, obtaining $\langle x_{\max} \rangle$.
2 The two parties substract every element of $\langle \mathbf{x} \rangle$ with $\langle x_{\max} \rangle$ locally, and invoke $\Pi_{\mathsf{exp}}(\langle \mathbf{x} - x_{\max} \rangle)$, obtaining $\langle \mathbf{x}' \rangle = \langle \exp(\mathbf{x} - x_{\max}) \rangle$. Denote the elements of $\mathbf{x}'$ as $(x'_i)_i$.
3 The two parties take the sum along the secret-shared vector $\langle \mathbf{x}' \rangle$ locally, producing shares $\langle s \rangle = \langle \sum_i x'_i \rangle$. They invoke $\Pi_{\mathsf{recip}}(\langle s \rangle)$ to produce $\langle 1/s \rangle$.
4 The two parties invoke $\Pi_{\mathsf{EleMul}}(\langle x' \rangle_i, \langle 1/s \rangle)$ for each element in $\mathbf{x}'$, followed by a truncation. This step produces the elements of $\langle \mathbf{y} \rangle = \langle \mathsf{Softmax}(\mathbf{x}) \rangle$.

– *(**Model privacy**) A corrupted, semi-honest client does not learn anything useful about the server's model weights* $\mathbf{W}$. *Formally, we require the existence of an efficient simulator* Sim_C *such that* $\mathsf{View}_C^{\Pi} \approx_{\mathrm{c}} \mathsf{Sim}_C(\mathbf{x}, \mathbf{y})$, *where* View_C^{Π} *denotes the view of client in the execution of* Π *and* $\mathbf{y}$ *denotes the output (inference result) of the protocol* Π *to the client.*

The security proof of our private inference framework follows by the combination of each sub-protocol and the sequential composibility of each operators to a full transformer architecture. The security proof of matrix multiplication follows from the security of the RLWE-based BFV HE scheme [9] and the proofs in [14, 11] for the protocol itself. For the security proof of each non-linearity protocol, we refer the readers to [32] for $\Pi_{\mathsf{EleMul}}, \Pi_{\mathsf{exp}}, \Pi_{\mathsf{tanh}}$ and [14] for $\Pi_{\mathsf{ReLU}}, \Pi_{\mathsf{rSqrt}}, \Pi_{\mathsf{max}}, \Pi_{\mathsf{recip}}$. These protocols mainly relies on the security of oblivious transfer [29, 16] and subfield vector oblivious evaluation [39] as basic cryptographic primitives.

7.2 Detail of the Non-linear Protocols

We provide detailed description of the three main non-linear protocols, GELU (Algorithm 3), Softmax (Algorithm 4) and LayerNorm[6] (Algorith 5), in the private inference framework.

7.3 Additional Evaluation Details

Codes. We make our codes publicly available at `https://github.com/privateLLM001/Private-LLM-Inference`.

Dataset details. We list details of the datasets used in our evaluation in Table 3.

[6]For simplicity, we assume the tensor is 2-dimensional. In transformer, usually the input is 3-dimensional with batch size, sequence and embedding dimensions. This could be coerced as 2-dimensional by squeezing all the dimensions except the last.

Algorithm 5: Layer normalization evaluation on secret shares

Input: The client inputs 2d-tensor share $\langle \mathbf{x} \rangle_0$ and the server inputs $\langle \mathbf{x} \rangle_1$, where $\mathbf{x} = \langle \mathbf{x} \rangle_0 + \langle \mathbf{x} \rangle_1$ is of shape (N, E).

Output: The two parties receives the secret shares of $\mathbf{y} = \mathsf{LayerNorm}(\mathbf{x})$.

1 The two parties locally takes the sum along the N dimension, multiply the sum by $\lfloor 2^f / E \rceil$ and truncates it, obtaining the mean shares $\langle \bar{\mathbf{x}} \rangle$.

2 The two parties invoke $\Pi_{\mathsf{EleMul}}(\langle \mathbf{x} - \bar{\mathbf{x}} \rangle, \langle \mathbf{x} - \bar{\mathbf{x}} \rangle)$, followed by a truncation. Then they repeat a similar process as in Step 1 to produce the variance shares $\langle \mathrm{Var}(\mathbf{x}) \rangle$.

3 The server adds $\epsilon \cdot 2^f$ to his share $\langle \mathrm{Var}(\mathbf{x}) \rangle_1$. This step semantically produces the shares of $\langle \mathrm{Var}(\mathbf{x}) + \epsilon \rangle$.

4 The two parties invoke $\Pi_{\mathsf{rSqrt}}(\langle \mathrm{Var}(\mathbf{X}) + \epsilon \rangle)$ to produce $\langle \mathbf{v} \rangle = \langle 1/\sqrt{\mathrm{Var}(\mathbf{x}) + \epsilon} \rangle$, and then they invoke $\Pi_{\mathsf{EleMul}}(\langle \mathbf{x} - \bar{\mathbf{x}} \rangle, \langle \mathbf{v} \rangle)$ and a truncation, where $\langle \mathbf{v} \rangle$ is repeated to fit the original shape (N, E). This step produces the normalized values $\tilde{\mathbf{x}}$ shares, where

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\mathrm{Var}(\mathbf{x}) + \epsilon}}.$$

5 For the affine transform, the two parties invoke $\Pi_{\mathsf{EleMul}}(\langle \tilde{\mathbf{x}} \rangle, \langle \gamma \rangle)$, where server provides $\langle \gamma \rangle_1 = \gamma$, and client provides $\langle \gamma \rangle_0 = 0$. The two parties then invoke a truncation, and the server adds $\beta \cdot 2^f$ to its share. This step produces the final result $\langle \mathbf{y} \rangle = \langle \mathsf{LayerNorm}(\mathbf{x}) \rangle$.

Name	Task	Domain	#Train	#Test
MRPC	2-class paraphrase	News	3.7k	408
SST-2	2-class sentiment	Movie reviews	67k	872
QNLI	2-class question answering	Wikipedia	105k	2k

Table 3: Dataset details

Accepted replacements. We list in Table 4 the how many operators are successfully replaced in the evaluation for the three models and three datasets with our substution strategy, with the allowed accuracy drop set to $\Delta\alpha = 2\%$. We observe that the replacement for all GELUs and Softmaxes are successful, while in large models, a very small portion of the LN2s of the first few blocks could not be replaced. We conjecture that the first few layers are essential to capture the overall features of the input sentences, and thus play a vital role for high accuracies. We leave the exploration further into this phenomenon as a future work.

Model	#Blocks	Task	GELU	Softmax	LN1	LN2
BERT-Tiny	2	MRPC	2	2	2	2
		SST2	2	2	2	2
		QNLI	2	2	2	2
BERT-Medium	8	MRPC	8	8	8	8
		SST2	8	8	8	8
		QNLI	8	8	8	8
RoBERTa-Base	12	MRPC	12	12	12	10
		SST2	12	12	12	12
		QNLI	12	12	12	11

Table 4: Successful model operator replacements for the three models and three datasets