SecFormer: Fast and Accurate Privacy-Preserving Inference for Transformer Models via SMPC

Anonymous ACL submission

Abstract

With the growing use of Transformer models hosted on cloud platforms to offer inference services, privacy concerns are escalating, especially concerning sensitive data like investment plans and bank account details. Secure Multi-Party Computing (SMPC) emerges as a promising solution to protect the privacy of inference data and model parameters. However, the application of SMPC in Privacy-Preserving Inference (PPI) for Transformer models, often leads to considerable slowdowns or declines in performance. This is largely due to the multitude of nonlinear operations in the Transformer architecture, which are not well-suited to SMPC and difficult to circumvent or optimize effectively. To address this concern, we introduce a comprehensive PPI framework called *SecFormer* to achieve fast and accurate PPI for Transformer models. We successfully eliminate the high-cost exponential and maximum operations in PPI without sacrificing model performance and developed a suite of efficient SMPC protocols by employing suitable numerical computation methods to boost other complex nonlinear functions in PPI, including GeLU, LayerNorm, and a redesigned Softmax. Our extensive experiments reveal that *SecFormer* outperforms *MPCFormer* in performance, showing improvements of 3.4% and 24.7% for $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, respectively. In terms of efficiency, *SecFormer* is 3.57 and 3.58 times faster than *PUMA* for $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, demonstrating its effectiveness and speed.

1 Introduction

Transformer models (Vaswani et al., 2017; Devlin et al., 2019; Radford et al., 2019; Brown et al., 2020; Raffel et al., 2020; Liu et al., 2019; Lewis et al., 2020; Ouyang et al., 2022; OpenAI, 2023) demonstrate exceptional performance across diverse downstream tasks and are extensively employed in a Model-as-a-Service (MaaS) paradigm to deliver high-quality inference services to clients. However, this MaaS framework poses a significant privacy risk for inference data and model parameters. For instance, both Coplit[1] and ChatGPT[2], which are Transformer-based services, necessitate users to upload plaintext requests. This operational procedure not only poses a threat to users' privacy but also probably contravenes relevant legal regulations such as the EU's General Data Protection Regulation (GDPR)[3].

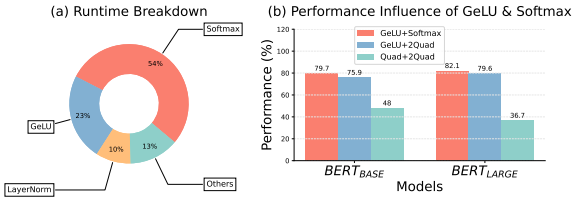

Figure 1: (a) Runtime breakdown of the $\text{BERT}_{\text{BASE}}$ model (12 layers, 512 tokens) based on an SMPC library. The total runtime for an example is 71 seconds. (b) Influence of different activation functions on model performance.

Secure Multi-Party Computation (SMPC) (Shamir, 1979; Yao, 1986; Goldreich et al., 1987), has demonstrated great potential in safeguarding the privacy of both inference data and model weights (Gilad-Bachrach et al., 2016; Liu et al., 2017; Mishra et al., 2020; Rathee et al., 2021; Huang et al., 2022). However, conducting Privacy-Preserving Inference (PPI)[4] for Transformer models using SMPC proves to be notably slow. To illustrate, $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) takes 71 seconds to inference per sample via SMPC, while plain-text inference takes less than 1 second.

This inefficiency stems from the limitations of

[1] https://github.com/features/copilot
[2] https://chat.openai.com
[3] https://gdpr-info.eu/
[4] Without confusion, we refer to SMPC-based PPI as PPI for short in this paper .

current SMPC protocols in executing nonlinear operations in Transformer models. As depicted in Fig. 1(a), we find that Softmax and GeLU, which account for a small share of the plaintext inference overhead, take up 77.03% of the time in PPI. This observation aligns with findings in Wang et al. (2022); Li et al. (2022). In an effort to mitigate this, Li et al. (2022) *redesigned the Transformer model* by substituting Softmax and GeLU with some SMPC friendly quadratics, bypassing the privacy-preserving computations of the non-linear operations (i.e., erf, exponential, and maximum) in Softmax and GeLU. This aggressive substitution significantly enhances PPI efficiency but unfortunately, substantially diminishes the model's performance and is not scalable to larger models (Fig. 1(b)). Some other studies (Dong et al., 2023) tried to boost the PPI by *designing more efficient SMPC protocols*, which can preserve the model performance but still face expensive PPI overhead.

In this study, we present a comprehensive PPI framework named SecFormer, tailed to achieve fast and accurate PPI for Transformer models by exploiting the superiorities of both Transformer model and SMPC protocol designs. *Our investigation reveals that preserving accurate computation of GeLU significantly improves PPI performance (Fig. 1(b))*. Building on this insight, SecFormer implements model design to bypass the expensive nonlinear PPI operations such as exponential and maximum in Softmax (Section 3.1). This adaptation, coupled with the strategic use of knowledge distillation, allows SecFormer to construct a Transformer model that is both high-performing and compatible with SMPC. To further enhance the PPI performance, we turn to protocol design and develop a suite of efficient SMPC protocols that utilize suitable numerical calculation methods to handle other complex nonlinear functions in PPI, such as GeLU, LayerNorm, and the redesigned Softmax (Section 3.2). To be specific, SecFormer introduces a novel SMPC protocol for GeLU based on segmented polynomials and Fourier series, facilitating efficient and accurate computation of GeLU. In addition, SecFormer deploys efficient privacy-preserving square-root inverse and division calculation for LayerNorm and Softmax using the Goldschmidt method (Goldschmidt, 1964; Markstein, 2004), coupled with input deflation techniques to bypass the nonlinear initial-value computation.

We conducted extensive evaluations of SecFormer on various datasets using Transformer models $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. The experimental results reveal that SecFormer achieves an average performance improvement of 3.4% and 24.7% for $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, respectively, compared to the state-of-the-art approach based on pure model design (Section 4.2), while maintaining comparable efficiency. In comparison to the approach that only improves the SMPC protocols, SecFormer exhibits a speedup of 3.57 and 3.58 times in PPI (Section 4.3), while sustaining comparable PPI performance.

2 Background and Related Works

2.1 Workflow of SMPC-based Model Inference

Secure Multi-Party Computation (SMPC) is a cryptographic technique that offers a promising solution for model Privacy-Preserving Inference (PPI) among multiple participants (Gilad-Bachrach et al., 2016; Liu et al., 2017; Mishra et al., 2020; Rathee et al., 2021; Huang et al., 2022). Typically, participants adhere to cryptographic primitives like secret sharing (Shamir, 1979; Goldreich et al., 1987) to safeguard the model weights and inference data. This paper mainly introduces the 2-out-of-2 secret sharing scheme due to its efficiency and representativeness. Specifically, the 2-out-of-2 secret sharing divides a secret x in the ring of integers $\mathcal{Z}_L$ into two random shares $[\![x]\!] = ([x]_0, [x]_1)$ with $x = (([x]_0 + [x]_1) \mod L)$, ensuring that neither share reveals information about x while allowing correct reconstruction of x when the two shares are combined. In constructing the SMPC protocols, the shares are owned by two distinct participants. They communicate the masked intermediate results to each other to accomplish the privacy-preserving computation of different functions and get the shares of the computational results.

The PPI workflow leveraging 2-out-of-2 secret sharing is depicted in Fig. 2. It involves three essential stakeholders: a *model inference service provider* that needs to protect model weights, a *client* that needs to protect inference data, and an *SMPC engine* that performs model PPI. The SMPC engine contains three non-colluding servers (i.e., participants): two computing servers S_j for $j \in \{0, 1\}$ for shares computation of PPI and an assistant server T for generating random numbers needed to execute the SMPC protocols. Initially, the service provider and client securely transmit the shares of model weights and inference data to S_0

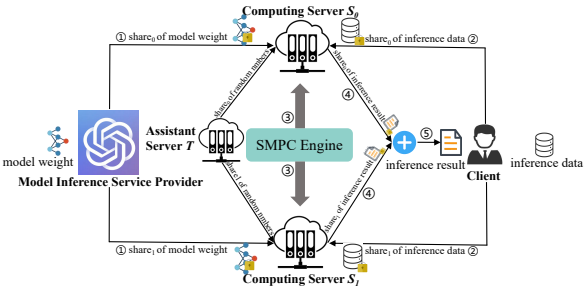

Figure 2: Workflow of PPI based on secret sharing.

and S_1, respectively (① and ②). Subsequently, the computing servers utilize these shares as input and complete PPI by executing the SMPC protocols through interactive computation with the assistance of T, yielding the shares of the inference results. (③). These shares are then relayed to the client (④), facilitating the local reconstruction of the inference result (⑤). Since each participant has only one share of the inputs, outputs, or intermediate results, this PPI workflow can guarantee the privacy of model weights and inference data.

2.2 Main Bottlenecks of SMPC-based Transformer Model Inference

Although the above PPI workflow guarantees the privacy of model weights and inference data, it faces unacceptable communication overhead (Table 1) in implementing some of the nonlinear operations (i.e., Softmax, GeLU, and LayerNorm), which are abundantly present in Transformer models and become a main bottleneck in PPI.

Specifically, for a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, Softmax in Transformer converts it to an n-dimensional probability distribution with

$$\text{Softmax}(\boldsymbol{x})[i] = \frac{e^{x_i - \tau}}{\sum_{h=1}^{n} e^{x_h - \tau}}, \qquad (1)$$

where $\tau = \max_{h=1}^{n} x_h$ is used to ensure stable numerical computations. As indicated in Table 1, there are three obstacles to the SMPC of Softmax: *exponential, division, and maximum*. Note that the calculation of maximum needs to call Π_{LT} operation $\log^n$ times (Knott et al., 2021) and becomes the biggest obstacle.

The function of GeLU is defined as

$$\text{GeLU}(x) = \frac{x}{2}\big(1 + \text{erf}(\frac{x}{\sqrt{2}})\big), \qquad (2)$$

where $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$. The GeLU function's nonlinear component is derived from the erf and *there is currently no SMPC protocol for its privacy-preserving computation*.

Notation	Input	Output	Comm Round	Comm Volume (bit)
Π_{Add}	$([\![x]\!], [\![y]\!])$	$[\![x+y]\!]$	0	0
Π_{Sin}	$[\![x]\!]$	$[\![sin(x)]\!]$	1	42
Π_{Square}	$[\![x]\!]$	$[\![x^2]\!]$	1	128
Π_{Mul}	$([\![x]\!], [\![y]\!])$	$[\![xy]\!]$	1	256
Π_{MatMul}	$([\![X]\!], [\![Y]\!])$	$[\![XY]\!]$	1	$256n^2$
Π_{LT}	$([\![x]\!], c)$	$[\![(x < c)]\!]$	7	3456
Π_{Exp}	$[\![x]\!]$	$[\![e^x]\!]$	8	1024
Π_{rSqrt}	$[\![x]\!]$	$[\![\sqrt{x}]\!]$	$9 + 3t$	6400
Π_{Div}	$[\![x]\!]$	$[\![1/x]\!]$	$16 + 2t$	10368

Table 1: SMPC protocols from Knott et al. (2021); Zheng et al. (2023b). t is the number of Newton iterations for implementing the protocol; n is the dimension of the matrix. These protocols are invoked in a black-box manner in this paper. The details are provided in Appendix E.

Given a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, the LayerNorm function is defined as

$$\text{LayerNorm}(\boldsymbol{x}) = \gamma \cdot \frac{\boldsymbol{x} - \bar{x}}{\sqrt{var(\boldsymbol{x}) + \epsilon}} + \beta, \qquad (3)$$

where $\bar{x} = \sum_{h=1}^{n} x_h/n$, $var(\boldsymbol{x}) = \sum_{h=1}^{n}(x_h - \bar{x})^2$, γ and β are two learnable parameters, and ϵ is a very small decimal used to prevent the denominator from being zero. For SMPC, the main bottleneck in computing LayerNorm comes from the *division and square root operations*.

2.3 Efficient PPI for Transformer Models

To alleviate the aforementioned bottlenecks, existing works on PPI for Transformer models improve the efficiency through either model design or SMPC protocol design. The studies based on model design (Chen et al., 2022; Li et al., 2022; Zeng et al., 2022; Zhang et al., 2023; Liang et al., 2023) bypass the high overhead operations in PPI by replacing the SMPC-unfriendly nonlinear operations in Transformer. These schemes substantially increase efficiency but usually lead to a significant degradation in model performance. The studies that design more efficient SMPC protocols (Hao et al., 2022; Zheng et al., 2023a; Gupta et al., 2023; Dong et al., 2023; Hou et al., 2023; Ding et al., 2023; Pang et al., 2023) improve the efficiency of PPI by customizing efficient SMPC protocols for the nonlinear operators in the Transformer. These schemes preserve the Transformer model's performance but still face expensive computational and communication overheads.

As a representative work based on model design, Li et al. (2022) improves the efficiency of PPI by replacing GeLU and Softmax with $\text{Quad} = 0.125x^2 + 0.25x + 0.5$ and

$$2\text{Quad}(\boldsymbol{x})[i] = \frac{(x_i + c)^2}{\sum_{h=1}^{n}(x_h + c)^2}, \qquad (4)$$

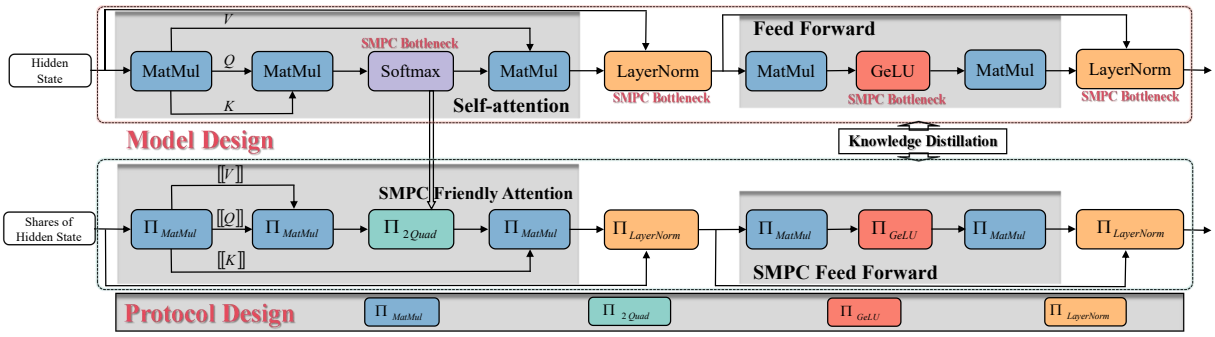

Figure 3: **An illustration of our proposed SecFormer framework.** In the model design phase, SecFormer substitutes Softmax with 2Quad to obtain an SMPC-friendly model while preserving model performance. In the SMPC protocol design stage, SecFormer improves the efficiency of the main bottlenecks in PPI for Transformer models, i.e., GeLU, LayerNorm, and 2Quad.

respectively, such that the privacy-preserving computation of erf, exponential, and maximum is bypassed. Following this, knowledge distillation is employed, with the fine-tuned Transformer model acting as the teacher and the approximate Transformer model as the student. Distillation is carried out on downstream task data, yielding a Transformer model compatible with SMPC. This approach is effective in improving the efficiency of PPI, however, it leads to a significant decrease in model performance. Our investigation reveals that preserving accurate computation of GeLU significantly improves PPI performance. Dong et al. (2023) achieves the first SMPC protocol for GeLU functions by utilizing segmented functions and polynomial fitting. However, the computation of segmented functions and polynomials requires multiple calls of Π_{LT} and Π_{Mul}, making it inefficient.

3 SecFormer Framework

As discussed above, existing efficient PPI studies suffer from either performance degradation or high inference overhead. To resolve this issue, the SecFormer framework is proposed in this section. We begin with an overview of SecFormer in Section 3.1 and introduce the new efficient SMPC protocols for GeLU, LayerNorm, and Softmax in Section 3.2.

3.1 Overview

SecFormer enhances the efficiency of PPI for Transformer models, addressing both model and SMPC protocol design. The overall depiction of SecFormer is presented in Fig. 3.

In the model design phase, SecFormer implements new strategies to bypass the nonlinear operations with the high overhead in PPI, such as exponential and maximum in Softmax, while preserving model performance. Specifically, SecFormer replaces Softmax with 2Quad while retaining the accurate computation of the GeLU. Inspired by (Li et al., 2022), SecFormer further improves the performance of PPI inference by incorporating knowledge distillation techniques.

In the SMPC protocol design phase, SecFormer introduces a suite of efficient SMPC protocols by utilizing appropriate numerical computation methods. Specifically, SecFormer introduces a novel SMPC protocol for GeLU based on segmented polynomials and Fourier series, which facilitates the efficient and accurate computation of GeLU. Subsequently, SecFormer deploys streamlined privacy-preserving calculation for square-root inverse and division using the Goldschmidt method (Goldschmidt, 1964; Markstein, 2004), coupled with input deflation techniques to eliminate the need for nonlinear initial-value computation.

3.2 SMPC Protocols of SecFormer

We next present new efficient SMPC protocols of GeLU, LayerNorm, and the approximated Softmax designed in SecFormer. These algorithms' security proofs and communication complexity analysis are presented in Appendix D.

Protocol for GeLU. To address the efficiency challenges of GeLU private computations (Section 2.2), some studies replaced GeLU in (2) with its SMPC-friendly alternatives such as ReLU (Zeng et al., 2022) or quadratics (Li et al., 2022). Although this approach can enhance PPI efficiency, it may result in irreversible performance losses. Dong et al. (2023) introduces the first SMPC protocol for GeLU using segmented functions and polynomial fitting whose computation, however, entails multi-

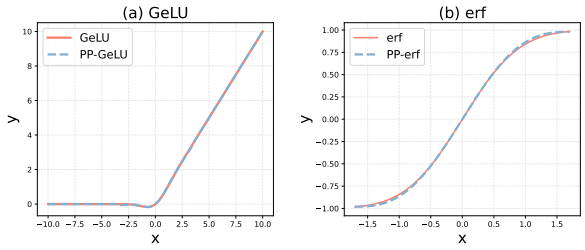

Figure 4: Fitting results of GeLU and erf functions.

ple calls of Π_{LT} and Π_{Mul}, rendering it inefficient.

To solve the above problems, we design an efficient SMPC protocol Π_{GeLU} based on segmented polynomials and *Fourier series*. As shown in Fig. 4, the erf function is an odd function symmetric about the origin with $\lim_{x\to\infty} \text{erf}(x) = 1$ and $\lim_{x\to-\infty} \text{erf}(x) = -1$. Therefore, we can convert it to the following segmented function

$$\text{erf}(x) \approx \begin{cases} -1, & x < -1.7 \\ f(x), & -1.7 \le x \le 1.7, \\ 1, & x > 1.7 \end{cases} \quad (5)$$

where $f(x)$ can be approximated through a Fourier series composed of sine functions with a period[5] of 20. Although a greater number of terms enhances the accuracy of the fitting outcomes, it concurrently leads to increased communication overhead. Here, we employ the following 7-term Fourier series:

$$f(x) = \boldsymbol{\beta} \odot \sin(\boldsymbol{k} \odot \pi x/10), \quad (6)$$

where $\boldsymbol{k} = (1, 2, 3, 4, 5, 6, 7)$, $\boldsymbol{\beta}$ is the Fourier series coefficients and $\odot$ denotes the element-wise multiplication. For $i = 1, 2, \ldots, 7$,

$$\boldsymbol{\beta}_i = \frac{1}{10} \int_{-10}^{10} \text{erf}(x) \sin(\frac{\boldsymbol{k}_i \pi x}{10}) dx \,. \quad (7)$$

According to Eq. (7), we can compute the coefficients $\boldsymbol{\beta}$ = $(1.25772, -0.0299154, 0.382155, -0.0519123, 0.196033, -0.0624557, 0.118029)$.

Based on (5), the computation of the erf function is converted into comparison and sine function. The precise calculation of GeLU can be accomplished by combining Π_{Mul} with the erf function. The specific steps of the SMPC protocol for GeLU are shown in Algorithm 1. Specifically, in steps 1-5 of Algorithm 1, we determine in which interval of the segmented function the input x falls by calling the Π_{LT}. Then, in step 7, the privacy-preserving

[5]The results of the sine function fitting for different periods are shown in Appendix F.

Algorithm 1: SMPC Protocol for Π_{GeLU}

Input: For $j \in \{0, 1\}$, S_j holds the shares $[x]_j$.
Output: For $j \in \{0, 1\}$, S_j returns the shares $[y]_j$ with $y = \text{GeLU}(x)$.

/* Determine the input interval */
1 $[\![c_0]\!] = \Pi_{LT}([\![x]\!], -1.7)$ // $(x < -1.7)$
2 $[\![c_1]\!] = \Pi_{LT}([\![x]\!], 1.7)$ // $(x < 1.7)$
3 $[\![z_0]\!] = [\![c_0]\!]$ // $(x < -1.7)$
4 $[\![z_1]\!] = [\![c_1]\!] - [\![z_0]\!]$ // $(-1.7 \le x \le 1.7)$
5 $[\![z_2]\!] = 1 - [\![c_1]\!]$ // $(x > 1.7)$
/* Compute $f(\frac{x}{\sqrt{2}})$ */
6 $[\![\hat{x}]\!] = \frac{1}{\sqrt{2}}[\![x]\!]$
7 $[\![f(\hat{x})]\!] = \boldsymbol{\beta} \odot \Pi_{sin}(\boldsymbol{k} \odot \pi[\![x]\!]/10)$
/* Compute erf($\hat{x}$)) */
8 $[\![\text{erf}(\hat{x})]\!] = [\![z_0]\!] + \Pi_{Mul}([\![z_1]\!], [\![f(\hat{x})]\!]) + [\![z_2]\!]$
/* Compute GeLU(x) */
9 $[\![y']\!] = 1 + [\![\text{erf}(\hat{x}))]\!]$
10 $[\![y]\!] = \Pi_{Mul}([\![\frac{x}{2}]\!], [\![y']\!])$

computation of $f(x)$ is achieved by utilizing the Π_{sin} presented in (Zheng et al., 2023b). The algorithm requires only 1 round of communication, and the specific steps of it is in Appendix E.2. In steps 8-10, we compute the erf function and execute the GeLU calculation by invoking Π_{Mul}.

Protocol for LayerNorm. Previous work (Knott et al., 2021) implements the privacy-preserving computation of LayerNorm in (3) by sequentially invoking Π_{rSqrt} and Π_{Div}, resulting in expensive computational and communication overheads. Goldschmidt's method enables the direct conversion of *square root inverses* (i.e., $\frac{1}{\sqrt{\cdot}}$) directly into multiple iterations of multiplications. However, achieving a broader convergence range often requires complex nonlinear initial value calculations, such as Look-up-table (LUT) (Rathee et al., 2021) or exponentiation (Knott et al., 2021), before the iteration begins. To resolve this issue, we propose to employ the *deflation technique* for bypassing these intricate nonlinear initial value calculations that are incompatible with SMPC. The detailed steps of the SMPC protocol for LayerNorm are in Algorithm 2.

Specifically, in steps 3-8, we use Goldschmid's method to compute $\frac{1}{\sqrt{q}}$ where $q = (var(\boldsymbol{x}) + \epsilon)/\eta$. Through division by a constant η (A hyperparameter whose value is shown in Appendix G.), we initially deflate the denominator into the interval $[0.001, 2.99]$ which ensures fast convergence for linear initial values. Then, we set the initial values $p_0 = 1$, $q_0 = q$, and compute $m_i = (3 - q_{i-1})/2$, $p_i = p_{i-1} m_i$, $q_i = q_{i-1} m_i^2$ at each iteration by calling Π_{Mul} and Π_{Square}. After $t = 11$ iteration, the value of $\frac{1}{\sqrt{q}}$ is calculated.

Algorithm 2: SMPC Protocol for LayerNorm $\Pi_{LayerNorm}$

Input: For $j \in \{0,1\}$, S_j holds the shares $[\boldsymbol{x}]_j$.
Output: For $j \in \{0,1\}$, S_j holds the shares $[\boldsymbol{y}]_j$ with $\boldsymbol{y} = \text{LayerNorm}(\boldsymbol{x})$.

/* Compute the mean and variance */
$[\![\bar{x}]\!] = \frac{1}{n} \cdot \sum_{h=1}^{n} [\![x_h]\!]$
$[\![var(\boldsymbol{x})]\!] = \sum_{h=1}^{n} \Pi_{Square}([\![x_h]\!] - [\![\bar{x}]\!])$
/* Goldschmidt's method */
$p_0 = 1, q_0 = \frac{1}{\eta}([\![var(\boldsymbol{x})]\!] + \epsilon)$
for $i \leftarrow 1$ **to** t **do**
  $[\![m_i]\!] \leftarrow (3 - q_{i-1})/2$
  $[\![q_i]\!] \leftarrow \Pi_{Mul}([\![q_{i-1}]\!], \Pi_{Square}([\![m_i]\!]))$
  $[\![p_i]\!] \leftarrow \Pi_{Mul}([\![p_{i-1}]\!], [\![m_i]\!])$
end
/* Compute LayerNorm(x) */
$[\![\boldsymbol{y}]\!] = \gamma \cdot (\frac{1}{\eta}([\![\boldsymbol{x}]\!] - [\![\bar{x}]\!]) \cdot [\![p_t]\!]) + \beta$

Protocol for Approximated Softmax. As mentioned in Section 3.1, we follow Li et al. (2022) and bypass the privacy-preserving computations of exponential and maximum by substituting Softmax with 2Quad in (4). However, to preserve the normalized nature of Softmax, the division operations cannot be avoided and thus become a new obstacle.

To solve this problem, we again use the Goldschmidt's method to convert the division operation to multiplications. Similar to the LayerNorm protocol, the complex calculation of initial values is avoided by effective deflation. The implementation of the SMPC protocol for the approximated Softmax (i.e., Π_{2Quad}) is shown in Appendix B.

4 Experiments

This section showcases the effectiveness of SecFormer through extensive experiments. We begin with the experiment setup in Section 4.1 and then report the performance assessment results in Section 4.2 and efficiency evaluations in Section 4.3, respectively. We further provide more analysis for SecFormer in Section 4.4, including an efficiency evaluation for Π_{GeLU}, $\Pi_{LayerNorm}$ and Π_{2Quad}.

4.1 Experimental Setup

Implementation. We implemented SecFormer using CrypTen[6], a semi-honest privacy-preserving machine learning framework based on secret sharing. To simulate the experimental environment, we utilized three Tesla V100 servers with a 10GB/s bandwidth. The hyperparameters for model fine-tuning and distillation follow the settings in (Li et al., 2022), see Appendix G for details.

[6]https://github.com/facebookresearch/CrypTen

Baselines. We compare SecFormer with state-of-the-art works based on model design (MPCFormer (Li et al., 2022)) and SMPC protocol design (PUMA (Dong et al., 2023)). Specifically, MPCFormer improves the efficiency of PPI by substituting Softmax and GeLU with some SMPC friendly quadratics. PUMA enhances PPI efficiency by designing more efficient SMPC protocols for GeLU, LayerNorm and Softmax. Following the setting in MPCFormer, we include the result of the fine-tuned redesigned model as the baseline, denoted as MPCFormer$_{w/o}$ and SecFormer$_{w/o}$ (i.e., fine-tuned without distillation). We also re-implement PUMA on CrypTen for consistency.

Models and Datasets. We followed MPCFormer using a representative transformer model BERT, see Appendix G for details. For the reliability of the experimental results, we use datasets with different evaluation metrics and sizes, including RTE, MRPC, CoLA, STS-B, and QNLI. In terms of evaluation metrics, MRPC uses F1 scores, STS-B employs the average of Person and Spearman correlations, CoLA uses Matthews correlations, and RTE and QNLI rely on accuracy.

4.2 Performance Comparison

We validate the performance of Seformer and the main results are shown in Table 2. For the model design framework MPCFormer, SecFormer exhibits a significant performance improvement. Specifically, for $\text{BERT}_{\text{BASE}}$, SecFormer outperforms MPCFormer across all tasks, resulting in a 3.4% average improvement. For $\text{BERT}_{\text{LARGE}}$, MPCFormer faces significant performance degradation, including CoLA task failure. In contrast, even without data distillation, SecFormer outperforms MPCFormer. After distillation, SecFormer demonstrates a substantial 24.7% performance improvement compared to MPCFormer. This is mainly because SecFormer implements an accurate computation of GeLU instead of replacing it aggressively with a quadratic polynomial.

For the protocol design framework PUMA, SecFormer incurs only a marginal 0.9% and 1.3% performance degradation. PUMA does not perform any model design and achieves PPI without performance loss. However, this results in PUMA facing unacceptable communication overheads, as detailed in Section 4.3.

Models	Methods	GeLU Approx.	Softmax Approx.	QNLI (108k)	CoLA (8.5k)	STS-B (5.7k)	MRPC (3.5k)	RTE (2.5k)	**Avg.**
$\text{BERT}_{\text{BASE}}$	Plain-text	GeLU	Softmax	91.7	57.8	89.1	90.3	69.7	**79.7**
	PUMA	GeLU	Softmax	91.7	57.8	89.1	90.3	69.7	79.7*
	$\text{MPCFormer}_{w/o}$	Quad	2Quad	69.8	0.0	36.1	81.2	52.7	48.0
	MPCFormer	Quad	2Quad	90.6	52.6	80.3	88.7	64.9	75.4
	$\text{SecFormer}_{w/o}$	GeLU	2Quad	89.3	57.0	86.2	83.8	63.2	75.9
	SecFormer	GeLU	2Quad	91.2	57.1	87.4	89.2	69.0	78.8*
$\text{BERT}_{\text{LARGE}}$	Plain-text	GeLU	Softmax	92.4	61.7	90.2	90.6	75.5	**82.1**
	PUMA	GeLU	Softmax	92.4	61.7	90.2	90.6	75.5	82.1*
	$\text{MPCFormer}_{w/o}$	Quad	2Quad	49.5	0.0	0.0	81.2	52.7	36.7
	MPCFormer	Quad	2Quad	87.8	0.0	52.1	81.4	59.2	56.1
	$\text{SecFormer}_{w/o}$	GeLU	2Quad	90.8	60.8	89.0	87.6	69.7	79.6
	SecFormer	GeLU	2Quad	92.0	61.3	89.2	88.7	72.6	80.8*

Table 2: Performance comparison of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. Bolded numbers indicate best results; numbers marked "*" indicate performance loss within 1.5%. For $\text{BERT}_{\text{BASE}}$, we directly use the results from (Li et al., 2022). For $\text{BERT}_{\text{LARGE}}$, Li et al. (2022) uses the 2ReLU instead of 2Quad for performance reasons, where $\text{2ReLU}(\boldsymbol{x})[i] = \text{ReLU}(\boldsymbol{x})[i] / \sum_{h=1}^{n} \text{ReLU}(\boldsymbol{x})$. Calculating ReLU requires a call to Π_{LT}. This results in more overhead than calculating 2Quad.

4.3 Efficiency Comparison

We evaluate the efficiency by testing the time and communication volume required to perform single-sample inference across different frameworks. The main results are shown in Table 3. We can find that SecFormer is significantly more efficient than PUMA. Specifically, for $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, SecFormer performs 3.57 and 3.58 faster than PUMA on the total inference time. These advantages stem from that SecFormer utilizes model design to achieve efficient computation of Softmax, and design efficient SMPC protocols suitable for the Transformer models for other nonlinear operators (i.e., GeLU, LayerNorm) by using appropriate numerical computation techniques. The efficiency of each SMPC protocol is shown in Table 3 and will be discussed later in Section 4.4.

When considering the framework of model design, SecFormer is only 1.05 and 1.04 times slower than MPCFormer in the scenarios of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, respectively. This result is based on the fact that SecFormer spends 41% of its time performing privacy-preserving calculations for GeLU, while MPCFormer spends only 0.01% of its time to implement the privacy-preserving calculations for Quad. However, the conclusions in Section 4.2 suggest that replacing GeLU with quadratic leads to dramatic degradation of model performance or even failure on some tasks (i.e., performance with 0 in Table 2).

In conclusion, experiments with SecFormer regarding performance and efficiency reveal its dual advantages, combining strengths from both protocol design and model design frameworks.

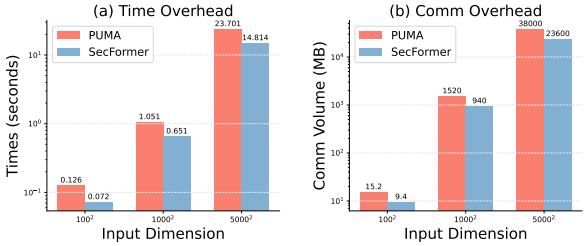

Figure 5: Comparison of Π_{GeLU} Time and Communication Overheads.

4.4 SMPC Protocols Evaluation

We compare Π_{GeLU} with PUMA in terms of time and communication overhead. The comparison results in Fig. 5 show that Π_{GeLU} is about 1.6 times lower than PUMA in time and communication overhead. This is mainly due to the fact that it invokes fewer Π_{LT} relative to PUMA. In terms of accuracy, both Π_{GeLU} and PUMA meet the needs of PPI, while CrypTen can only maintain accuracy over a small range. See Appendix C for details.

We compare $\Pi_{LayerNorm}$ with CrypTen (Knott et al., 2021) in terms of time and communication overhead. Fig. 6 shows that $\Pi_{LayerNorm}$ is up to 4.5 times faster than CrypTen (Knott et al., 2021). This is due to the efficient privacy-preserving square root inverse calculation proposed by SecFormer. As shown in Fig. 7, it is 4.2 times faster than CrypTen and reduces the communication volume by a factor of 2.5.

Models	Methods	GeLU		Softmax		LayerNorm		Others		Total
		Times(s)	Comm(GB)	Times(s)	Comm(GB)	Times(s)	Comm(GB)	Times(s)	Comm(GB)	Times(s)
$\text{BERT}_{\text{BASE}}$	CrypTen	16.46	28.689	37.25	50.285	6.614	0.492	9.365	3.463	71.097
	PUMA	16.343	28.689	42.219	67.837	2.285	0.477	8.781	3.463	69.661
	MPCFormer	0.351	0.604	3.129	1.895	6.522	0.497	8.589	3.463	**18.591**
	SecFormer	8.132	17.817	1.362	1.844	1.523	0.468	8.496	3.463	19.513*
$\text{BERT}_{\text{LARGE}}$	CrypTen	27.881	57.378	83.017	134.093	9.105	1.272	19.945	8.565	140.018
	PUMA	27.357	57.378	89.938	180.898	4.313	1.254	18.278	8.565	139.954
	MPCFormer	0.351	0.604	7.274	5.052	10.864	1.282	19.261	8.565	**37.75**
	SecFormer	14.531	35.635	3.115	4.916	3.122	1.248	18.321	8.565	39.089*

Table 3: Efficiency Comparison of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. Bolded numbers indicate the best results; Numbers marked "*" indicate within 2 seconds slower than the best result. The results are the average of ten runs.

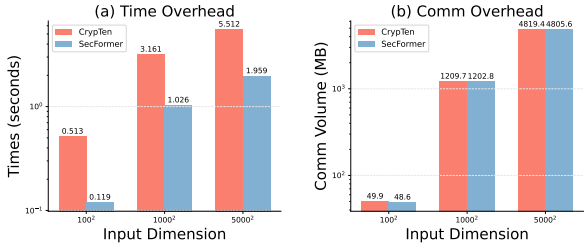

Figure 6: Comparison of $\Pi_{LayerNorm}$ Time and Communication Overheads.

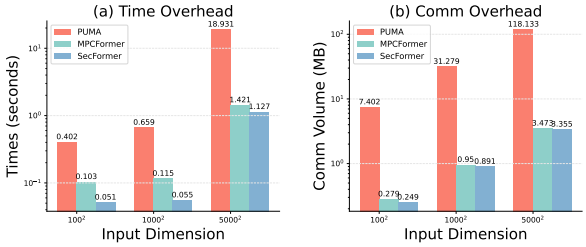

Figure 8: Comparison of Π_{2Quad} Time and Communication Overheads.

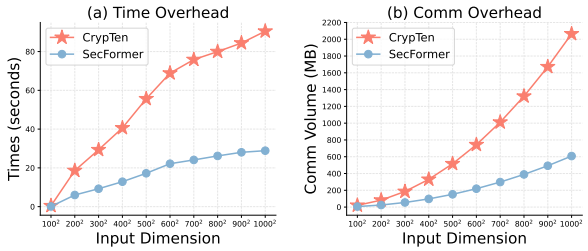

Figure 7: Comparison of Privacy-preserving Calculation for Square-root Inverse Time and Communication Overheads.

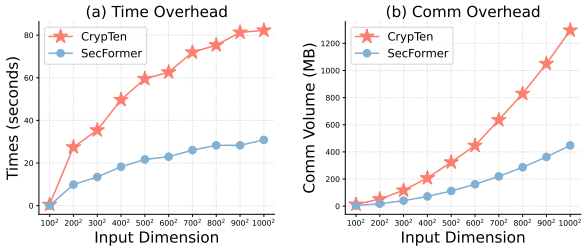

Figure 9: Comparison of Privacy-Preserving Division Calculation Time and Communication Overheads.

We compare Π_{2Quad} with MPCFormer and PUMA in terms of time and communication overhead. Fig. 8 shows that Π_{2Quad} is $1.26 \sim 2.09$ times faster than MPCFormer and the communication overhead is reduced by $1.04 \sim 1.12$ times. These enhancements come from the efficient privacy-preserving division calculation proposed by SecFormer. As shown in Fig. 9, it is 3.2 times faster than CrypTen, and the communication overhead is reduced by 1.6 times.

Compared to PUMA, which achieves precise privacy-preserving Softmax, Π_{2Quad} gets a drastic improvement in efficiency, i.e., $8.24 \sim 16.8$ times faster and $30.53 \sim 36.2$ times less communication. This is due to the fact that the model design performed by SecFormer avoids the calculation of exponential and maximum.

5 Conclusion

We present SecFormer, a synergistic PPI framework that strategically combines the strengths of both SMPC protocol design and Transformer model design. Extensive experiments reveal that SecFormer surpasses existing PPI methods, achieving fast and accurate PPI for Transformer models. It not only matches the performance of approaches that focus exclusively on SMPC protocols but also competes with the efficiency of methods dedicated solely to model design. SecFormer holds significant potential for enhancing large language models, offering an effective solution that promises to maintain high performance while ensuring stringent privacy and efficiency standards in increasingly complex and expansive linguistic landscapes.

6 Limitations

We summarize the limitations of SecFormer as follows: (1) SecFormer focuses on implementing PPI for the encoder-only Transformer model, such as BERT, without extending to other Transformer model families like the GPT series. We concentrate on the encoder-only Transformer model because of its continued prominence in real-world natural language understanding tasks, particularly within resource-constrained environments like edge computing. Prior efforts to implement the encoder-only Transformer model for PPI have encountered obstacles, including slow inference speeds and substantial performance degradation. Our work addresses these challenges and offers insights to guide future optimization efforts concerning PPI across diverse Transformer model families. The proposed protocols can be applied to implement PPI of other transformer-based models straightforwardly and we will consider PPI for decoder only Transformer models like GPT in the future. (2) Regarding SMPC protocols, SecFormer executes only on CrypTen and does not invoke the cutting-edge underlying SMPC protocols. We will try to exploit other privacy-preserving frameworks with more advanced SMPC protocols to further improve the inference efficiency of SecFomer in future work. (3) SecFormer only performs model design by replacing Softmax with 2Quad and does not incorporate other model lighting techniques. Other model lightweight techniques such as model quantization and pruning are compatible with the proposed SMPC protocols and can be combined into SecFormer to further improve the PPI efficiency in the future.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186.

Yuanchao Ding, Hua Guo, Yewei Guan, Weixin Liu, Jiarong Huo, Zhenyu Guan, and Xiyong Zhang. 2023. East: Efficient and accurate secure transformer framework for inference. *arXiv preprint arXiv:2308.09923*.

Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Cheng. 2023. PUMA: Secure inference of LLaMA-7B in five minutes. *arXiv preprint arXiv:2307.12533*.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 201–210.

Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM.

Robert E Goldschmidt. 1964. Applications of division by convergence. In *M.Sc dissertation, Massachusetts Institute of Technology*.

Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. 2023. SIGMA: Secure GPT inference with function secret sharing. *Cryptology ePrint Archive, Paper 2023/1269*.

Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private inference on transformers. *Advances in Neural Information Processing Systems*, 35:15718–15731.

Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen jie Lu, Cheng Hong, and Kui Ren. 2023. CipherGPT: Secure two-party GPT inference. *Cryptology ePrint Archive, Paper 2023/1147*.

Zhicong Huang, Wenjie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and fast secure two-party deep neural network inference. In *Proceedings of 31st USENIX Security Symposium*, pages 809–826.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. CrypTen: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. 2022. MPCFormer: Fast, performant and private transformer inference with MPC. *arXiv preprint arXiv:2211.01452*.

Zi Liang, Pinghui Wang, Ruofei Zhang, Nuo Xu, and Shuo Zhang. 2023. MERGE: Fast private text generation. *arXiv preprint arXiv:2305.15769*.

Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 619–631.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Peter W. Markstein. 2004. Software division and square root using Goldschmidt's algorithms. In *6th Conference on Real Numbers and Computers*, pages 146–157.

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

OpenAI. 2023. GPT-4 technical report. *ArXiv*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2023. BOLT: Privacy-preserving, accurate and efficient inference for transformers. *Cryptology ePrint Archive, Paper 2023/1893*.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. 2021. SIRNN: A math library for secure RNN inference. In *Proceedings of 2021 IEEE Symposium on Security and Privacy*, pages 1003–1020.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Yongqin Wang, G Edward Suh, Wenjie Xiong, Benjamin Lefaudeux, Brian Knott, Murali Annavaram, and Hsien-Hsin S Lee. 2022. Characterization of MPC-based private inference for transformer-based models. In *Proceedings of 2022 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 187–197.

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

Wenxuan Zeng, Meng Li, Wenjie Xiong, Wenjie Lu, Jin Tan, Runsheng Wang, and Ru Huang. 2022. MPCViT: Searching for MPC-friendly vision transformer with heterogeneous attention. *arXiv preprint arXiv:2211.13955*.

Yuke Zhang, Dake Chen, Souvik Kundu, Chenghao Li, and Peter A Beerel. 2023. SAL-ViT: Towards latency efficient private inference on ViT using selective attention search with a learnable softmax approximation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5116–5125.

Mengxin Zheng, Qian Lou, and Lei Jiang. 2023a. Primer: Fast private transformer inference on encrypted data. *arXiv preprint arXiv:2303.13679*.

Yu Zheng, Qizhi Zhang, Sherman SM Chow, Yuxiang Peng, Sijun Tan, Lichun Li, and Shan Yin. 2023b. Secure softmax/sigmoid for machine-learning computation. In *Proceedings of the 39th Annual Computer Security Applications Conference*, pages 463–476.

A 2-out-of-2 Secret Sharing

The 2-out-of-2 secret sharing includes arithmetic secret sharing and Boolean secret sharing. The 2-out-of-2 arithmetic secret sharing contains two algorithms:

- $Shr(x) \rightarrow ([x]_0, [x]_1)$ is used to generate the shares by randomly selecting a number r from $\mathcal{Z}_L$, letting $[x]_0 = r$, and computing $[x]_1 = (x - r) \mod L$;
- $Rec([x]_0, [x]_1) \rightarrow x$ is used to reconstruct the original value from the shares, which can be done by simply calculating $([x]_0 + [x]_1) \mod L$.

Note that due to the randomness of r, neither a single $[x]_0$ nor $[x]_1$ can be used to infer the original value of x. The arithmetic secret sharing technique has been widely used to construct SMPC protocols for ML operations (e.g., $+$, $-$ and $\cdot$, etc.) such that both the inputs and outputs of the protocol are the arithmetic shares of the original inputs and outputs:

$$\Pi_f([inputs]_0, [inputs]_1) \rightarrow ([f]_0, [f]_1), \quad (8)$$

where Π_f denotes an SMPC protocol of the operation f. The shares in $\mathcal{Z}_2$ is called *Boolean* shares, and the operations of $+$, $-$ and $\cdot$ are replaced by bit-wise operations $\oplus$ and $\wedge$. We use $[\![x]\!]$, $\langle\!\langle x \rangle\!\rangle$ denotes the arithmetic and boolean shares of x, i.e., $[\![x]\!] = ([x]_0, [x]_1), \langle\!\langle x \rangle\!\rangle = (\langle x \rangle_0, \langle x \rangle_1)$.

B Protocol for the Approximated Privacy-Preserving Softmax

In this section, we give the specific implementation of the SMPC Protocol for the approximated Softmax (i.e., 2Quad) as mentioned in Section 3.1. In steps 3-8 of Algorithm 3, we first deflate the denominator $q = \sum_{h=1}^{n} (\boldsymbol{x} + c)^2$ into the interval $[0.001, 1.999]$, which ensures fast convergence for linear initial values, through division by a constant η. Subsequently, we set the initial values $q_0 = q, p_0 = (\boldsymbol{x} + c)^2$, and compute $m_i = 2 - q_{i-1}, p_i = p_{i-1} m_i, q_i = q_{i-1} m_i$ at each iteration by calling Π_{Mul}. After $t = 13$ iterations, $\frac{p}{q}$ is computed.

C Accuracy Comparison of Privacy-Preserving GeLU Algorithms

In this section we compare the performance of privacy-preserving GeLU with Puma and CrypTen.

Algorithm 3: SMPC Protocol for Softmax Π_{2Quad}

Input: For $j \in \{0, 1\}$, S_j holds the shares $[\boldsymbol{x}]_j$.
Output: For $j \in \{0, 1\}$, S_j holds the shares $[\boldsymbol{y}]_j$, where $\boldsymbol{y} = 2quad(\boldsymbol{x})$.

/* Compute the numerator */
1 $[\![\boldsymbol{p}]\!] = \Pi_{Square}([\![\boldsymbol{x} + c]\!])$
/* Compute the denominator */
2 $[\![q]\!] = \sum_{h=1}^{n} [\![\boldsymbol{p}[h]]\!]$
/* Goldschmidt's method */
3 $q_0 = \frac{1}{\eta}[\![q]\!], [\![\boldsymbol{p}_0]\!] = \frac{1}{\eta}[\![\boldsymbol{p}]\!]$
4 **for** $i \leftarrow 1$ **to** t **do**
5 $\quad [\![m_i]\!] \leftarrow 2 - [\![q_{i-1}]\!]$
6 $\quad [\![\boldsymbol{p_i}]\!] \leftarrow \Pi_{Mul}([\![\boldsymbol{p}_{i-1}]\!], [\![\boldsymbol{m}_i]\!])$
7 $\quad [\![q_i]\!] \leftarrow \Pi_{Mul}([\![q_{i-1}]\!], [\![m_i]\!])$
8 **end**
9 $[\![\boldsymbol{y}]\!] = [\![\boldsymbol{p_t}]\!]$

The specific comparison results are shown in Table 4. Both SecFormer and Puma achieve privacy-preserving computation within the entire interval of the GeLU function by using segmented polynomials. CrypTen, on the other hand, locally fits the erf function using a low-order Taylor expansion and thus can only achieve privacy-preserving computation of the GeLU function in a smaller interval.

D Security Proof and Communication Complexity Analysis

D.1 Security Proof

SecFormer adheres to a semi-honest (also known as honest-but-curious) assumption similar to the works of Li et al. (2022) and Dong et al. (2023), where honest participants constitute the majority. Under this assumption, the security of SecFormer can be formally proved within the simulation paradigm, specifically against static semi-honest adversaries denoted as $\mathcal{A}$, which can potentially corrupt one of the servers. The simulation paradigm delineates two distinct worlds: the real world and the ideal world. In the real world, the servers execute the protocols in the presence of semi-honest adversaries $\mathcal{A}$. In contrast, the ideal world involves the servers transmitting inputs to a trusted dealer capable of correctly executing the protocol. The security of SecFormer necessitates that, for any semi-honest adversary $\mathcal{A}$, the distribution of the real world remains indistinguishable from that of the ideal world. The definition of privacy-preserving inference protocols (Mishra et al., 2020; Huang et al., 2022; Hao et al., 2022) is as follows:

Definition 1 *A protocol Π_P between the servers*

Input Interval	$[-1, 1]$			$[-5, 5]$			$[-10, 10]$		
Methods	CrypTen	Puma	SecFormer	CrypTen	Puma	SecFormer	CrypTen	Puma	SecFormer
Error Mean	0.001	0.005	0.001	30437.717	0.003	0.005	7480209.5	0.002	0.003
Error Var	$\pm 8.37 \times 10^{-6}$	$\pm 6.85 \times 10^{-6}$	$\pm 2.03 \times 10^{-6}$	$\pm 3.28 \times 10^{9}$	$\pm 1.01 \times 10^{-5}$	$\pm 3.82 \times 10^{-5}$	$\pm 1.68 \times 10^{14}$	$\pm 7.06 \times 10^{-6}$	$\pm 2.54 \times 10^{-5}$

Table 4: Accuracy Comparison of Privacy-Preserving GeLU Algorithms.

who have the shares of the model weights and the inference data is a privacy-preserving protocol if it complies with the following criteria: (1) Correctness: For a model M with weights w and input samples x, the client's output at the end of the protocol is the correct inference M (w, x); and (2) Security: For a computational server $S_j, j \in \{0, 1\}$ that is corrupted by adversary $\mathcal{A}$, there exists a probabilistic polynomial time simulator Sim_{S_j} such that adversary $\mathcal{A}$ cannot distinguish $View_{S_j}^{\Pi_P}$ (i.e., the view of S_j during the implementation of Π_P) from Sim_{S_j}. Similarly, for a corrupted server T, there exists an efficient simulator Sim_T such that $View_T^{\Pi_P}$ is indistinguishable from Sim_T.

SecFormer is constructed from the sub-protocols outlined in the works of Knott et al. (2021) and Zheng et al. (2023b). Leveraging the concept of universally composable security established by Canetti (2001), we can prove that SecFormer satisfies Definition 1 directly.

D.2 Communication Complexity Analysis

The execution of Π_{GeLU} invokes two Π_{LT}, one Π_{Sin} and one Π_{Mul}. Thus the execution of the privacy-preserving GeLU algorithm takes a total of $2\log^L + 4$ rounds of online communication and transmit 7210 bits.

The implementation of privacy-preserving LayerNorm requires calls to Π_{Mul}, Π_{Square} and privacy-preserving inverse of the square root. The inverse of the square root requires one call to Π_{Square} and two calls to Π_{Mul} in parallel per iteration, costing 2 rounds of communication and transferring 640 bits. Thus performing the square root inverse takes a total of 22 rounds of communication and transfers 7040 bits and the implementation of privacy-preserving LayerNorm takes a total of 24 rounds of communication and transfers 7424 bits.

The implementation of approximate privacy-preserving Softmax requires calls to Π_{Mul} and Π_{Div}. The Π_{Div} requires two call to Π_{Mul} in parallel per iteration, costing 1 rounds of communication and transferring 512 bits. Thus performing the Π_{Div} takes a total of 13 rounds of communication and transfers 6,656 bits and the implementation of approximate privacy-preserving Softmax takes a total of 23 rounds of communication and transfers 6,784 bits.

E Underlying SMPC Protocols

In this section, we provide a brief overview of the underlying protocols used and refer to the works of Knott et al. (2021) and Zheng et al. (2023b) for details. Let S_j with $j \in \{0, 1\}$ be two parties that are used to execute the SMPC protocol. Each party S_j will be given one additive share $([u]_j, [v]_j) \in \mathcal{Z}_L$ of the operation inputs u and v for $j \in \{0, 1\}$.

E.1 Privacy-Preserving Linear Protocols

Privacy-preserving addition is implemented with $[u + v]_j = [u]_j + [v]_j$ for $j \in \{0, 1\}$.

Privacy-preserving multiplication is implemented with Beaver-triples: (a, b, c) where $a, b \in \mathcal{Z}_L$ are randomly sampled from $\mathcal{Z}_L$ and $c = a \cdot b \mod L$. Specifically, for each $j \in \{0, 1\}$, S_j first calculates $[d]_j = [u]_j - [a]_j$ and $[e]_j = [v]_j - [b]_j$. Then, they send the $[d]_j$ and $[e]_j$ to each other and reconstruct $d = Rec([d]_0, [d]_1)$ and $e = Rec([e]_0, [e]_1)$. Finally, the additive share of $u \cdot v$ can be computed using $[u \cdot v]_j = -jd \cdot e + [u]_j \cdot e + d \cdot [v]_j + [c]_j$. To complete the SS-based multiplication, both parties need to spend 1 round of two-way communication and transmit 256 bits.

E.2 Privacy-Preserving Non-Linear Protocols

Privacy-preserving comparison is implemented by the conversion between the additive shares and the binary shares. Specially, $[\![z]\!] = [\![x - y]\!]$ is converted to the binary shares $\langle\langle z \rangle\rangle$ through additive circuit with $\log^L$ round of communication. Subsequently, the binary shares of z's sign bit can be determined by $\langle\langle b \rangle\rangle = \langle\langle z \rangle\rangle >> (l-1)$[7]. Finally, the additive shares of $x < y$ can be derived by converting $\langle\langle b \rangle\rangle$ to $[\![b]\!]$ with one round of communication. Thus, the implementation of privacy-preserving compare algorithm cost $\log^L + 1$ round of communication and transmit 3456 bits.

[7] $>> l$ denote shift l bit to the right.

Privacy-preserving maximum of the n-element vector $\boldsymbol{x}$ is implemented by calling $\log^n$ privacy-preserving comparisons using the tree reduction algorithm (Knott et al., 2021).

Privacy-preserving exponential is implemented using the repeated-squaring method

$$e^x = \lim_{x\to\infty}\left(1+\frac{x}{2^n}\right)^{2^n}, \tag{9}$$

which converts exponential calculations into addition and square operations. The number of iterations n is set to 8 in (Knott et al., 2021) by default.

Privacy-preserving reciprocal is implemented by Newton-Raphson method, which converts reciprocal calculations into addition and multiplication operations. The iterative formula is

$$y_{n+1} = y_n(2 - xy_n). \tag{10}$$

The initial value of the iteration is

$$y_0 = 3e^{\frac{1}{2}-x} + 0.003. \tag{11}$$

The number of iterations is set to 10 in (Knott et al., 2021) by default.

Privacy-preserving square root is implemented by Newton-Raphson method, which converts exponential calculations into addition and multiplication operations. The iterative formula is

$$y_{n+1} = \frac{1}{2}y_n(3 - xy_n^2). \tag{12}$$

The initial value of the iteration is

$$y_0 = e^{-2.2(\frac{x}{2}+0.2)} + 0.198046875. \tag{13}$$

The number of iterations is set to 3 in (Knott et al., 2021) by default.

Privacy-preserving sine is implemented on trigonometric identities. Specifically, $\sin(x) = \sin(\delta)\cos(t) + \cos(\delta)\sin(t)$, where $\delta = x - t$. With the assistance of the server T, the random numbers $t, \sin(t), \cos(t)$ are generated in the offline phase, and the share of $\sin(x)$ is computed in the online phase with only one round of communication and transmits 42 bit.[8] See Algorithm 4 for an implementation of the privacy-preserving sine.

[8] CrypTen uses 16-bit computational precision.

Algorithm 4: Privacy-preserving sine

Input: For $j \in \{0,1\}$, S_j holds the shares $[x]_j$; Same Pseudo-Random Function (PRF) and key k_j.

Output: For $j \in \{0,1\}$, S_j returns the shares $[y]_j$, where $y = sin(x)$.

`/* Offline Phase */`

$S_0, T : [t]_0, [u]_0, [v]_0 \leftarrow PRF(k_0)$

$S_1, T : [t]_1 \leftarrow PRF(k_1)$

$T : t = [t]_0 + [t]_1, [u]_1 = sin(t) - [u]_0, [v]_1 = cos(t) - [v]_0$

`/* Online Phase */`

$[\delta]_j = ([x]_j - [t]_j) \mod 20$

$\delta = [\delta]_0 + [\delta]_1$ `// reconstruct` δ `by 1 round of communication`

$p = sin(\delta), q = cos(\delta)$

$[y]_j = p[v]_j + q[u]_j$

F Fourier Series Fitting Results

In this section, we give the results of fitting $\text{erf}(x)$ using Fourier series composed of different periodic sin functions. Specifically, we fit $\text{erf}(x)$ using the 7-th order Fourier series composed of sin functions with periods of 10, 20, 30, and 40, respectively, and the specific fitting results are shown in Fig. 10.

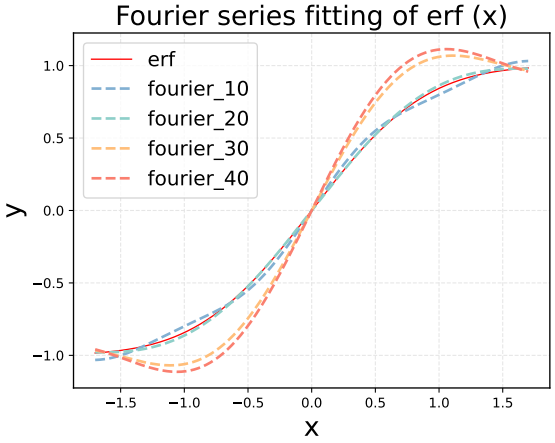

Figure 10: Fourier series fitting results for different periods. "fourier_{10}" denotes that the period of the sin function in the Fourier series is 10.

G Models and Hyper-parameter

Models. In this section, we provide a concise overview of the architecture of the experimental models. For more detailed information, we refer the readers to the HuggingFace Transformers library.

- $\text{BERT}_{\text{BASE}}$: $\text{BERT}_{\text{BASE}}$ represents the foundational version of the Bert model, comprising 12 Transformer encoder layers, a hidden size

of 768, and 12 heads. With 110 million parameters, it undergoes training on a substantial corpus of unlabeled text data.

- $\text{BERT}_{\text{LARGE}}$: $\text{BERT}_{\text{LARGE}}$ serves as an expanded iteration of $\text{BERT}_{\text{BASE}}$, featuring 24 Transformer encoder layers, a hidden size of 1024, and 16 heads. Boasting approximately 340 million parameters, this version exhibits increased potency, enabling it to capture intricate language patterns.

Hyper-parameter. For LayerNorm and Softmax, we set the constants η as 2000 and 5000, respectively, to ensure that the value of the denominator can be deflated to a reasonable range of convergence. We follow the choice of hyperparameters for fine-tuning and distillation in MPCFormer (Li et al., 2022). Specifically, in the fine-tuning phase, we use a learning rate of $[1e-6, 5e-6, 1e-5, 1e-4]$, a batch size of $[64, 256]$, and epochs of $[10, 30, 100]$. We fine-tuned each model with a combination of hyperparameters and selected the best performing model as teacher. In the distillation phase, we decide the number of epochs based on the MSE loss of the embedding and transformation layer distillations. For small datasets (CoLA, MRPC, RTE), the batch size is 8; for large datasets (QNLI, STS-B), the batch size is 32. Specifically, in the embedding and transform layer distillation phases, 10 epochs for QNLI, 20 epochs for MRPC, 50 epochs for STS-B, 50 epochs for CoLA, and 50 epochs for RTE.