SHAFT: Secure, Handy, Accurate, and Fast Transformer Inference

Andes Y. L. Kei
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

Sherman S. M. Chow*
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

***Abstract*—Adoption of transformer-based machine learning models is growing, raising concerns about sensitive data exposure. Nonetheless, current secure inference solutions incur substantial overhead due to their extensive reliance on non-linear protocols, such as softmax and Gaussian error linear unit (GELU). Driven by numerical stability needs, softmax approximations (*e.g.*, NeurIPS 2021) typically extract the maximum element of an input vector, incurring logarithmic rounds (in the input length). Existing GELU protocols (*e.g.*, S&P 2024) use piecewise approximations with high-degree polynomials that rely heavily on secure multiplications and comparisons, which are expensive. Such complexities also hinder model owners unfamiliar with cryptography from deploying their custom models easily.**

SHAFT, our proposed system, provides a secure, handy, accurate, and fast transformer inference framework for deployment. Highlights of our contributions include 1) the first constant-round softmax protocol for transformers, uniquely combining the benefits of input clipping and characteristics of ordinary differential equations, and 2) a highly accurate GELU protocol on a novel characterization designed for Fourier series approximation. Extending to broader contexts, our new protocols also apply to general neural networks that use softmax as the final layer and to transformer architectures with different activation functions. Remarkably, SHAFT outperforms state-of-the-art SIGMA (PETS 2024), which uses secret sharing, and BumbleBee (NDSS 2025), which additionally uses RLWE-based homomorphic encryption. More specifically, SHAFT minimizes communication by 25-41% and matches SIGMA's running time while surpassing BumbleBee in running time by 4.6-$5.3\times$ on LANs and 2.9-$4.4\times$ on WANs. Alongside these improvements, SHAFT attains accuracy comparable to plaintext models, confirming its numerical stability. Next in this progression, SHAFT provides an accessible open-source framework for secure and handy deployment by smoothly integrating with the Hugging Face library (EMNLP Demos 2020).

I. Introduction

Attention mechanisms, foundational to the transformer architecture [1], compute relationships between different positions within sequences to generate meaningful representations. Yielding exceptional performance, transformer-based models like BERT [2], GPT [3], and ViT [4] have been widely deployed for many tasks in natural language processing (NLP). Large-scale models have powered many transformative applications in generative artificial intelligence, such as ChatGPT. Key privacy concerns arise as more users rely on these models and submit numerous sensitive inference queries daily, highlighting the need for privacy-preserving inference systems, while service providers must also protect their models.

Secure multi-party computation (MPC) has significantly advanced privacy-preserving machine learning (PPML), enabling secure inference that protects both queries and models. Sophisticated solutions like GForce [5] complete an inference query on the CIFAR-10 image dataset using VGG-16, a 16-layer convolutional neural network (CNN), within 0.3 s [6]. Meeting the demands of private inference for transformer-based models, however, remains challenging due to their reliance on MPC-unfriendly (*i.e.*, inefficient) non-linear functions, notably softmax and Gaussian error linear unit (GELU). Constructing a transformer inference system that addresses these limitations is essential and needs the following features.

Security. Users cannot gain any knowledge about the private inputs of others, namely, the query and the model parameters.
Handiness. The system should be deployable regardless of users' familiarity with cryptographic techniques or libraries.
Efficiency. The cost of private inference should be low even for large models with hundreds of millions of parameters.
Reliability. The system should be reliable, with accuracy comparable to plaintext counterparts.

A. *Manifold Approaches to Non-linearity*

Existing private transformer inference frameworks, such as THE-X [7], Iron [8], MPCFormer [9], Privformer [10], Primer [11], MPCViT [12], BOLT [13], and SecFormer [14] (a comparison with more works is in Section II), can be categorized based on their approach to non-linear functions:

Rough Replacement. Works like MPCFormer [9] replace non-linear functions with simpler, MPC-friendly ones to speed up inference, but this causes notable performance degradation, with accuracy losses $>5\%$ [9, Table 4]. For mitigation, they often employ knowledge distillation, requiring extra training on plaintext data, which is not always feasible or desirable.

*Sherman Chow (corresponding author) is supported by GRF 14210621. Andes Kei is supported by the Hong Kong PhD Fellowship Scheme. The authors are grateful to the reviewers for entrusting them with final revisions. Kei thanks Yu Zheng for her comments on his first-year report (with different softmax and GELU ideas) and for discussing Fourier series during the rebuttal.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.242287
www.ndss-symposium.org

Precise Approximation. Non-linear functions can be more precisely approximated by piecewise (high-order) polynomials [13], iterative methods [14], or table lookups [8]. These approaches introduce substantial overhead: piecewise polynomials require many multiplications and (MPC-unfriendly) comparisons, iterative methods involve numerous communication rounds, and table lookups are computation-intensive, especially for large tables or inputs with a wide bitwidth.[1]

Exact Computation. Some other systems consider exact evaluation by decrypting intermediate values to compute these functions in plaintext [7] or with costly cryptographic tools like garbled circuits [11]. However, the first method could lead to recovery attacks [15], and the second suffers from prohibitive costs (*e.g.*, $\sim$400s running time [11, Table III]).

Alternative Architecture. Frameworks like Privformer [10] and MPCViT [12] (marked as "Restricted" in Table I) adopt simpler attention variants (*e.g.*, [16], [17]) and activations (*e.g.*, rectified linear unit/ReLU) to avoid softmax and GELU, achieving efficiency by targeting only specialized models. However, they lack protocols for softmax and GELU, rendering them incompatible with general transformers like GPT [3].

Most of the above solutions are not open-sourced. To our knowledge, existing works enable private inference only for a predefined set of transformers (often limited to BERT [2] and GPT [3]) using pretrained weights loaded onto local devices. For custom models, ML researchers who are not familiar with cryptographic implementations need to re-implement a private version from scratch, which can be challenging.

Some frameworks (*e.g.*, MP-SPDZ [18] and CrypTen [19]) are known for enabling the conversion of neural networks from Python-like source code to PPML code. However, they are limited to CNNs but not transformers due to the lack of transformer-specific modules like GELU.

B. Our Contributions

Our framework, SHAFT (**S**ecure, **H**andy, **A**ccurate, and **F**ast **T**ransformer inference), overcomes the above limitations. SHAFT is a two-party computation (2PC) framework based on secret sharing (SS). As the cost of running a transformer is dominated by the secure computations of non-linear functions like softmax and GELU, we focus on designing new protocols for these operations. Our key contributions include:

Stable Constant-Round Private Softmax. We introduce the first constant-round softmax protocol that is numerically stable (*i.e.*, the output is unlikely to overflow regardless of input) in private transformer inference. Typical approaches subtract the maximum element from an input vector of length m, requiring $O(\log_2(m))$ communication rounds. Instead, our new method combines *input clipping* (with a refined input range) and an *ordinary differential equation* (ODE) to prevent overflow, reducing rounds to $O(1)$ without notable loss of accuracy.

[1]Reducing bitwidth decreases data precision, generally affecting model accuracy, while excessively high bitwidth imposes substantial computation and communication overheads [6]. Designing efficient protocols with a reasonable default bitwidth is challenging but essential for accuracy, and tailored optimization of bitwidth while maintaining accuracy is highly beneficial.

Efficient and Precise Private GELU. We design a novel *GELU characterization for Fourier series* (FS) approximation with a maximum error of 4.60×10^{-3} and an average error of 7.39×10^{-4}. Our secure GELU protocol surpasses the state-of-the-art (SoTA) of BOLT [13] (Table III) by one round and two secure multiplications, a crucial gain for transformers with hundreds of thousands of GELU, while lowering 51% and 37% of maximum and average error, respectively. Beyond GELU, our formulation can generalize to other ReLU-like activations, *e.g.*, sigmoid linear unit (SiLU) function used in transformer variants like the famous LLaMA model [20] from Meta AI.

Secure Embedding on Index Inputs. We present the first private embedding protocol taking an index (unlike one-hot vector in existing works) as input, aligning to the specification of the standard PyTorch [21] ML library for seamless model conversions. Our protocol uses precomputed randomness [22] to securely convert an index to a one-hot vector in 1 round.

Interoperability with the Hugging Face Library. Building on CrypTen [19] with ML-developer-friendly APIs, SHAFT enables seamless import of pretrained models from the popular Hugging Face transformer library [23] via ONNX, an open standard format for representing neural networks. ML researchers without cryptographic knowledge can thereby easily perform private inference using a wide range of models.

Our experiments on four models (BERT-base, BERT-large [2], GPT-2 [24], and ViT-base [4]) and three datasets (QNLI, CoLA, and SST-2 from the GLUE benchmark [25]) in Section V demonstrate that SHAFT achieves lower private inference costs than SIGMA [26] and BumbleBee [27], while maintaining accuracy comparable to plaintext. SIGMA is the SoTA function-SS (FSS)-based framework that uses smaller bitwidths[2], and the latter is a homomorphic-encryption (HE)-based solution that reduces the communication of BOLT [13]. Section VII elaborates on optimizations for mixed-bitwidth frameworks (*e.g.*, SIRNN [28]) to further reduce costs.

II. Related Works

Literature on private neural network training and inference, particularly CNNs, is extensive. We focus on SoTA techniques for computing non-linear functions. For a comprehensive discussion, see the recent systematization of knowledge [6].

Table I compares SHAFT with existing private transformer inference frameworks, listed in chronological order.

A. MPC-Friendly Approximations of Softmax

$\mathsf{Softmax}(\vec{x})_i = e^{x_i} / \sum_j e^{x_j}$ is composed of two non-linear functions: exponential e^x and reciprocal $1/x$. Since e^x overflows for large x and $1/x$ overflows when $x \approx 0$, a typical approach is to compute softmax on a "normalized" input $(\vec{x} - \max(\vec{x}))$ for numerical stability. As a result, the secure softmax protocol in most existing works involves evaluating a sequence of maximum, exponential, and reciprocal ("M+E+R" in Table I). They are far from competing with us in softmax.

[2]CrypTen [19] does not support mixed-bitwidth operations.

TABLE I
COMPARISON OF PRIVATE TRANSFORMER INFERENCE FRAMEWORKS

Framework	Properties		Cryptographic Tools				Implemented Models				Softmax Methods						GELU Methods				
	Security	Handiness	SS	OT	HE	GC	BERT	GPT	ViT	Restricted	Rough	M+E+R	Iterative	ODE	LUT	Exact	Rough	Piece-Poly	LUT	Fourier	Exact
THE-X [7]	○	○	○	○	●	○	●	○	○	○	●	○	○	○	○	○	●	$(2,1)$	○	○	○
Iron [8]	●	○	●	●	●	○	●	○	○	○	○	●	○	○	●	○	○	(-, -)	●	○	○
MPCFormer [9]	◐	○	●	○	○	○	●	○	○	○	●	○	○	○	○	○	●	(-, -)	○	○	○
Privformer [10]	●	○	●	○	○	○	○	○	○	●	○	○	○	○	○	○	○	(-, -)	○	○	○
Primer [11]	◐	○	●	○	○	●	●	○	○	○	○	○	○	○	○	●	○	(-, -)	○	○	●
PUMA [29]*	◐	○	●	○	○	○	●	●	○	○	○	●	●	○	○	○	○	$(4,6)$	○	○	○
CipherGPT [30]*	◐	○	●	●	●	○	○	●	○	○	○	●	○	○	●	○	○	$(258,1)$	●	○	○
East [31]*	◐	○	●	●	●	○	●	○	○	○	○	●	●	○	○	○	○	$(10,3)$	○	○	○
MPCViT [12]	◐	○	●	○	○	○	○	○	○	●	○	○	○	○	○	○	○	(-, -)	○	○	○
STIP [32]*	○	○	○	○	○	○	●	●	●	○	○	○	○	○	○	●	○	(-, -)	○	○	●
BOLT [13]	◐	○	●	●	●	○	●	○	○	○	○	●	○	○	●	○	○	$(3,4)$	○	○	○
SecFormer [14]	○	◐	●	○	○	○	●	○	○	○	○	●	●	○	○	○	○	$(3,1)$	○	●	○
SIGMA [26]	●	○	●	○	○	○	●	●	○	○	○	●	○	○	●	○	○	$(3,1)$	●	○	○
BumbleBee [27]	●	○	●	●	●	○	●	●	●	○	○	●	●	○	○	○	○	$(4,6)$	○	○	○
NEXUS [33]*	●	○	○	○	●	○	●	●	○	○	○	●	●	○	○	○	○	$(4,6)$	○	○	○
SHAFT (This Work)	●	●	●	○	○	○	●	●	●	○	○	○	○	●	○	○	○	$(3,1)$	○	●	○

SS: Secret sharing, OT: Oblivious transfer, HE: Homomorphic encryption, GC: Garbled circuit, M+E+R: Maximum+Exponential+Reciprocal, ODE: Ordinary differential equations, LUT: Lookup table. For security, ○ means known issues, ◐ means no known issues but no formal proof, and ● signifies formal proofs. For Piece-Poly, (n,d) means an n-piece degree-d polynomial; (-, -) means it is not used. Entries with * are preprints by the submission deadline (July 2024).

CrypTen [19] approximates e^x by its limit characterization $(1+x/2^t)^{2^t}$. It requires t sequential multiplications to bound the maximum error by $e^{O(-x^2/2^t)}$. CryptGPU [34] approximates $1/x$ by Newton's method, computing $y_i = 2y_{i-1} - xy_{i-1}^2$ from an initial guess y_0. It takes $2t$ multiplications for a maximum error of $e^{O(-2^t)}$ over t iterations. SIRNN [28] uses two lookup tables (LUTs) to map the upper and lower bits of x to e^x. While secure table lookup might only take a constant number of rounds [35] or consume bandwidth logarithmic in input bitwidth [36], the computation cost remains exponential in input bitwidth. This demands highly optimized LUTs with carefully chosen sizes for practical application.

Despite recent advances in e^x and $1/x$ approximations, secure maximum computation remains a bottleneck. A common approach, known as the tree-reduction algorithm [19], involves dividing the input into two halves and recursively comparing each half's elements. This is still dominantly costly, requiring $O(\log_2(m))$ secure comparisons for input length m.

Zheng *et al.* [37] propose directly using an ODE to approximate softmax. This method requires $2t$ multiplications for accurate results, provided that $\max(\vec{x}) - \min(\vec{x}) \leq t$. It avoids evaluating e^x and $1/x$, thus bypassing any maximum computation. The ODE-based approach demonstrated SoTA performance in terms of running time for private CNN training by setting $t = 16$ or 32. However, when it comes to transformers with unbounded inputs and tens of thousands of softmax, the above correctness condition can be *easily violated*, and the errors accumulate over multiple softmax calls. In practice, to provide reasonable accuracy in private transformer inference, t needs to be as large as 128 or 256, diminishing the benefits.

For private transformer inference frameworks, PUMA [29], East [31], SecFormer [14], BumbleBee [27], and NEXUS [33] adopt iterative approximations[3] from CrypTen [19] and CryptGPU [34]. Iron [8], CipherGPT [30], BOLT [13], and SIGMA [26] rely on previous LUT protocols [28], [38], [39] to compute e^x and $1/x$. Interestingly, computing private maximum is *the* standard method in all these works (apart from weaker rough/revealing/inefficient/restricted approaches in Section II-C). Improving secure softmax computations for transformers addresses a critical bottleneck in private inference frameworks. This focus drives our pursuit of new techniques for secure softmax in transformers and motivates benchmarking against specialized secure softmax protocols [19], [37].

B. MPC-Friendly Approximations of GELU

GELU is a common activation function in transformers. It is defined as $\mathsf{GELU}(x) = 0.5x\left(1 + \mathsf{Erf}\left(x/\sqrt{2}\right)\right)$, where $\mathsf{Erf}(x) = (2/\sqrt{\pi})\int_0^x e^{-u^2}du$ is the Gaussian error function.

Iron [8] considers an approximation of GELU based on tanh: $\mathsf{GELU}(x) \approx 0.5x(1+\tanh(\sqrt{2/\pi}(x+0.044715x^3)))$. It follows the idea of SIRNN [28] to evaluate the $\tanh$ function (also a combination of e^x and $1/x$ like softmax).

As $\mathsf{GELU}(x)$ is close to $\mathsf{ReLU}(x) = \max(0,x)$ for larger $|x|$, most existing private transformer inference frameworks approximate GELU within a small range near 0 and set it as ReLU outside this range. This method requires evaluating a 3-piece function with two comparisons and one multiplication.

PUMA [29], BumbleBee [27], and NEXUS [33] (all ○ under GELU in Table I) use 2-piece, degree-6 polynomials, requiring one comparison and three multiplications for evaluation. CipherGPT [30] opts for a 256-piece, degree-1 polynomial, leveraging an LUT [38] to fetch coefficients and one multiplication for evaluating degree-1 polynomials.

[3]ODE is also an iterative approximation. Table I singles it out as a distinctive approach from those "Iterative" ones for more meaningful classifications.

East [31] (also all ○) adopts an oblivious-transfer (OT)-based approach to evaluate an 8-piece, degree-3 polynomial with one comparison, one multiplication, and two OTs.

SIGMA [26] observes that $\mathsf{ReLU}(x) - \mathsf{GELU}(x)$ is even. It uses an LUT to store the output of this function for the input range $[0, 4]$. For $x \in [-4, 4]$, it evaluates the LUT on $|x| = 2\mathsf{ReLU}(x) - x$. Computing $\mathsf{ReLU}(x)$ needs one comparison.

A concurrent work, SecFormer [14], uses an FS to approximate $\mathsf{Erf}(x)$ around $[-1.7, 1.7]$. Securely evaluating an FS takes one round [37]. While Table I marks both SecFormer and SHAFT with "Fourier" and $(3, 1)$ for 3-piece degree-1 polynomial approximation, GELU of SecFormer takes four secure multiplications, while ours takes only one. Ours also saves two rounds and, more importantly, enjoys *order-of-magnitude* improvements in both maximum and average errors (Table III). We note that the experimental results of SecFormer rely on the 3PC truncation protocol from CrypTen, which has been found to be insecure [40] (○ for security in Table I).

BOLT [13] approximates an even function $0.5x\mathsf{Erf}\left(x/\sqrt{2}\right)$ using a degree-4 polynomial for $x \in [0, 2.7]$ and evaluates it (with two multiplications) on $|x|$ for $x \in [-2.7, 2.7]$. Despite using piecewise polynomials, its number of pieces and degree are the smallest (apart from rough approximation). BOLT and SecFormer are thus our main competitors in secure GELU (Tables III and VI), but not others marked with "Rough," "LUT," "Exact," or all ○ under GELU (*e.g.*, [27]) in Table I.

C. Other Private Transformer Inference Frameworks

MPCFormer [9] uses rough approximations for non-linear functions (*e.g.*, $\mathsf{Softmax}(\vec{x}) \approx (\vec{x}+c)^2/\sum_j(x_j+c)^2$, where c is a constant), leading to an accuracy loss of $>5\%$.

THE-X [7] reveals intermediate outputs and computes non-linear functions on plaintexts, which is generally insecure [15]. STIP [32] transforms model weights and data with random permutation matrices for inference on permuted plaintext, violating standard MPC security guarantees.

Primer [11] uses computationally expensive garbled circuits for exact non-linear function computations, requiring $\sim$400s to run BERT-base [2] on a 30-token input query.

Privformer [10] focuses on an alternative transformer architecture using a piecewise linear attention variant [17] and ReLU activations (unlike GELU in typical transformers). MPCViT [12] works on transformers with linear attention [16] and replaces GELU with a linear function. These solutions do not handle private softmax and GELU computations, making them inapplicable to general transformers like GPT [3].

D. Conversions from Plaintext NNs to Secure NNs

Several PPML frameworks, such as MP-SPDZ [18], CrypTFlow [41], and CrypTen [19], enable the conversion of plaintext NNs (usually in Python-like source code) to secure NNs. This allows developers with limited cryptography knowledge to deploy their models. Yet, most of them require familiarity with the specific library designs, except for CrypTen, which is built on the standard PyTorch [21] ML library. Moreover, they all (including CrypTen) lack support for transformer-specific functionalities (*e.g.*, embedding layers and GELU). In contrast, SHAFT enables seamless conversion from plaintext transformers with just a few lines of code (Figure 2).

Regarding handiness, although SecFormer [14] *in principle* supports converting PyTorch models to PPML code as it is built on CrypTen, it is uncertain whether the conversion works for transformers. Additionally, its code is not open-sourced. This explains the ◐ indication in Table I.

III. Preliminaries

A. Notations

Set $\{0, \ldots, n-1\}$ is denoted by $[n]$, $n \in \mathbb{N}$. Matrices are represented by bold capital letters like $\mathbf{M}$. Column/row vectors use lowercase letters, with x_i referring to the i-th element of a vector $\vec{x}$. $\vec{1}$ denotes an all-one vector, $\vec{0}$ is a zero vector, $\vec{e_i}$ is a one-hot vector where the i-th element is 1 and all others are 0. $[\![y]\!] \leftarrow \Pi$ means the execution of an interactive protocol Π.

Scalar multiplication, element-wise product, inner product, right cyclic rotation, and concatenation are denoted by $\cdot$, $*$, $\langle\cdot,\cdot\rangle$, $\ggg$, and $||$, respectively. The indicator function $\mathbb{1}_{b(x)}(x)$ returns 1 if predicate $b(x)$ is true; 0 otherwise. The sign function $\mathsf{sgn}(x)$ returns 1 if and only if $x \geq 0$; -1 otherwise.

We use fixed-point encoding with a uniform bitwidth of $\ell = 64$ and precision $t = 16$. A floating-point value x_R is encoded as an integer x by scaling with 2^f and rounding down: $x = \lfloor x_R \cdot 2^f \rfloor$. Decoding recovers x_R approximately as $x/2^f$.

B. Transformer Architecture

A transformer comprises encoder and decoder blocks that extract feature vectors from input sequences (words in NLP or image patches in image processing) and generate outputs, respectively. These blocks have similar structures, consisting of many linear and non-linear layers, typically the following:

Embedding Layer is the first layer in transformers for NLP tasks. It maps each word (encoded as an index $i \in [d_{\mathsf{Emb}}]$) in the input to a vector $\vec{x} \in \mathbb{R}^{d_{\text{model}}}$ based on an LUT T_{Emb}, where d_{Emb} is the size of T_{Emb} and d_{model} is the dimension of intermediate layers in transformers. If the input is in the form of a one-hot vector $\vec{e_i}$, then the embedding layer is a matrix multiplication. In particular, this can be defined as follows:

$$\vec{x} = T_{\mathsf{Emb}}[i] = \vec{e_i}\mathbf{W}^E,$$

where the j-th row of $\mathbf{W}^E$ equals to $T_{\mathsf{Emb}}[j]$.

Attention Layer is a mapping from a query $\vec{q}$, a key-value pair $(\vec{k}, \vec{v})$, to a weighted sum of values, where the weights depend on $\vec{q}$ and $\vec{k}$. All the $\vec{q}, \vec{k}, \vec{v}$ are linearly projected from the same input $\vec{x}$, *i.e.*, $\vec{q} = \vec{x}\mathbf{W}^Q, \vec{k} = \vec{x}\mathbf{W}^K, \vec{v} = \vec{x}\mathbf{W}^V$ for some trained weights $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$. There are many attention variants (*e.g.*, scaled dot-product attention [1], strided attention [42], FAVOR+ [17]). Here, we focus on the scaled dot-product attention used in typical transformers. Let d_k be the dimension of $\vec{k}$. The attention layer can be computed by

$$\mathsf{Attention}(\vec{q}, \vec{k}, \vec{v}) = \mathsf{Softmax}\left(\frac{\vec{q}\vec{k}^T}{\sqrt{d_k}}\right)\vec{v}.$$

Multi-Head Attention (MHA) Layer extends the attention layer by projecting $\vec{q}, \vec{k}, \vec{v}$ into h different linear subspaces, where h is the number of heads, and performing the attention function on them in parallel. Let $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V, \mathbf{W}^O$ be trainable weights. The MHA layer can be described as

$$\mathsf{MHA}(\vec{x}) = (\mathsf{head}_0 || \cdots || \mathsf{head}_{h-1})\mathbf{W}^O,$$

$$\mathsf{head}_i = \mathsf{Attention}(\vec{x}\mathbf{W}_i^Q, \vec{x}\mathbf{W}_i^K, \vec{x}\mathbf{W}_i^V) \text{ for } i \in [h].$$

Feed-Forward Network (FFN) consists of two linear layers, with a GELU function in between. It can be represented as

$$\mathsf{FFN}(\vec{x}) = \mathsf{GELU}(\vec{x}\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2,$$

where $\mathbf{W}_i, b_i$ are the weight and bias of the i-th linear layer.

Layer Normalization (LayerNorm) normalizes the distribution of the output of a layer. Given the layer output $\vec{a}$, this operation can be formalized as follows:

$$\mathsf{LN}(\vec{a}) = \frac{g}{\sigma}(\vec{a} - \mu) + b,$$

$$\mu = \frac{1}{H}\sum_{i=1}^{H} a_i, \ \sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i - \mu)^2},$$

where H is the dimension of $\vec{a}$, and g, b are trainable parameters that control the mean and variance of the output.

A residual connection [43], followed by LayerNorm, is applied around each MHA and FFN layer. Altogether, the output of a transformer block can be defined as

$$\mathsf{Block}(\vec{x}) = \mathsf{LN}(\vec{y} + \mathsf{FFN}(\vec{y})), \vec{y} = \mathsf{LN}(\vec{x} + \mathsf{MHA}(\vec{x})).$$

C. Security Model

SHAFT supports secure outsourced inference [6], where the model and data owners distribute private inputs via SS to two untrusted servers, P_0 and P_1, typically large technology firms with substantial computing power. Inputs can be reconstructed only by combining shares from both. While it is possible to distribute the inputs to more servers using replicated SS, this raises the risk of secret recovery [6]. We consider static semi-honest probabilistic polynomial time (PPT) adversaries corrupting either P_0 or P_1, a standard assumption in private (transformer) inference [13], [26], [44], suitable for use cases where non-compliance risks reputational damage if caught.

Some protocols require precomputations, *e.g.*, Beaver's triples [45] for multiplication. These can be prepared in a data-independent offline phase by a semi-honest cryptographic service provider (SP) offering triples-as-a-service [46], or through 2PC protocols using HE [47] or OT [48]. We adopt an SP for simplicity, as it does not participate in the online phase and learns nothing about the inputs of P_0 and P_1.

Model extraction attacks [49] on plaintext model outputs lie outside the scope of cryptographic inference. Concerns like this can be mitigated by combining SHAFT with orthogonal techniques like differential privacy [50], [51].

TABLE II
EXISTING BUILDING BLOCKS

Protocol	Input/Output Description	Rounds
Less-than, $\Pi_{<}$	$[\![x]\!]$, $[\![y]\!]$ or $y \to [\![\mathbb{1}_{x<y}(x, y)]\!]$	$\log_2(\ell) + 2$
ReLU, Π_{ReLU}	$[\![x]\!] \to [\![\mathsf{ReLU}(x)]\!]$	$\log_2(\ell) + 3$
Inv. sq. root, Π_{rSqrt}	$[\![x]\!] \to [\![1/\sqrt{x}]\!]$	$t_{\mathsf{exp}} + 2t_{\mathsf{rSqrt}}$
Sine, $\Pi_{\sin}$	$[\![x]\!] \to [\![\sin(x)]\!]$	1

Protocol Π_{rSqrt} requires computing e^x to obtain a good initial guess. $t_{\mathsf{exp}}, t_{\mathsf{rSqrt}}$ are iteration counts for approximating e^x and $1/\sqrt{x}$, respectively. Specifically, we set $t_{\mathsf{exp}} = 8$ and $t_{\mathsf{rsqrt}} = 5$ in this work.

D. Cryptographic Primitives

Arithmetic Secret Sharing shares a scalar $x \in \mathbb{Z}/Q\mathbb{Z}$ among a set of parties $\mathcal{P}$, where $\mathbb{Z}/Q\mathbb{Z}$ is a quotient ring with Q elements. Here, we use 2-out-of-2 arithmetic SS with $Q = 2^\ell$. We denote the arithmetic SS of an ℓ-bit value x by $[\![x]\!] = \{[\![x]\!]_i\}_{P_i \in \mathcal{P}}$, where $[\![x]\!]_i$ is the share of x held by party P_i. Linear operations on (secret-)shared data, *i.e.*, $[\![z]\!] = [\![\alpha x + \beta y + \gamma]\!]$ for shared values $[\![x]\!], [\![y]\!]$ and public constants α, β, γ, can be done non-interactively by having P_0 and P_1 compute the linear operation on their shares of $[\![x]\!]$ and $[\![y]\!]$.

Private Multiplication ($\Pi_\times$) evaluates $[\![xy]\!]$ on shared values $[\![x]\!]$ and $[\![y]\!]$. We use the one-round Beaver-triple-based protocol [45]. Multiplying two floating-point values (of precision t) is followed by a truncation that is non-interactive in 2PC.

We also use secure protocols for some non-linear functions [19], [37] (*e.g.*, inverse square root), listed in Table II.

IV. Secure Transformer Inference

A. Overview of SHAFT

Figure 1 provides an overview of SHAFT's design. Model owners are machine-learning-as-a-service (MLaaS) providers who offer inference services on their private models. Data owners are MLaaS users who want to query the model with their sensitive data. They are both users of SHAFT. Once they outsourced the secure computations to two non-colluding servers P_0 and P_1 by secret-sharing their private inputs, they can go offline. Private inference between P_0 and P_1 is implemented on SS-based protocols, some needing precomputed triples obtained from MPC protocols or a semi-honest SP.

Transformers, as in Section III-B, consist of an embedding layer, multiple transformer blocks, and a final single-layer NN classifier. Each block includes MHA, FFN, and LayerNorm layers. These layers are supported by our secure protocols for softmax (Π_{Softmax}), GELU (Π_{GELU}), and embedding (Π_{Emb}).

Like CrypTen [19], we implement the tensor-computation APIs of the popular PyTorch library [21]. Our secure computations utilize highly optimized PyTorch operations, wrapped within `CrypTensor` objects. While CrypTen allows model owners to convert code from PyTorch to PPML via ONNX, it lacks support for transformer-specific layers (*e.g.*, embedding and GELU) and private transformer inference. We bridge this gap by implementing conversions from PyTorch to ONNX and ONNX to PPML for these layers, enabling full interoperability with PyTorch and compatible libraries like Hugging Face [23].

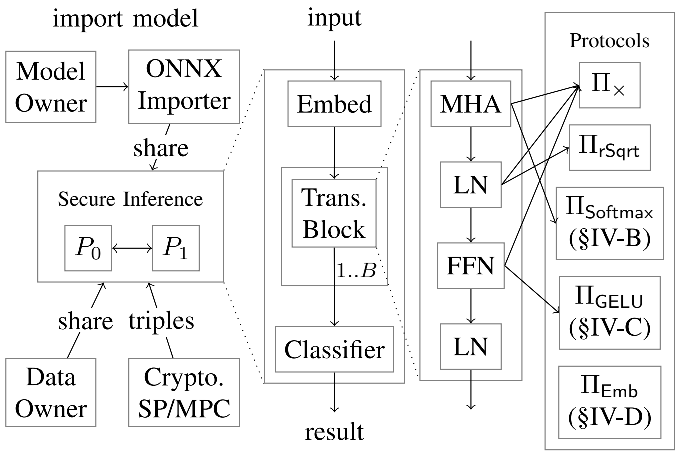

Fig. 1. SHAFT's high-level design over transformer architecture with embedding layer, B transformer blocks (LN denotes LayerNorm), and classifier

```
from transformers import (
        AutoModelForSequenceClassification )
import crypten as ct
# (standard) data loading and preprocessing omitted
# load pretrained model from Hugging Face
model = AutoModelForSequenceClassification
       .from_pretrained("user/bert-base-cased-qnli")
ct.init() # establish communication channels
# load secret-shared model and data (on GPU)
model_ss = ct.nn.from_pytorch(model, dummy_data)
           .encrypt().cuda()
data_ss = ct.cryptensor(data).cuda()
output_ss = model_ss(data_ss) # private inference
output = output_ss.get_plain_text() # recover result
```

Fig. 2. Private inference using a pretrained model from Hugging Face

Figure 2 shows how a pretrained transformer from Hugging Face can be seamlessly imported and deployed for private inference. Data owners invoke the standard Hugging Face APIs to load and preprocess their datasets.[4] Model owners load their pretrained models from the Hugging Face repository with a single `from_pretrained()` call. The users then initialize communication channels with the servers by calling `init()`. The model is first converted from PyTorch to ONNX with `from_pytorch()`, then to PPML code and shared with the servers via `encrypt()`. The dataset is shared to the servers through `CrypTensor` object creation. Private inference is done by supplying the shared input `data_ss` into the shared model `model_ss`. Finally, the data owner can recover the shared output `output_ss` using `get_plain_text()`.

In addition to importing models from Hugging Face, users of SHAFT, such as ML researchers, can develop and train their transformers with a customized architecture (*e.g.*, alternative attention variants or transformer blocks) in PyTorch and similarly import them into PPML code. This design enables flexible deployment of diverse transformers.

[4]Examples can be found in the Hugging Face's Github repository.

B. Private Softmax

For secure computation of the MHA layer, designing an efficient private softmax protocol is essential. Recall the definition of softmax on an input $\vec{x} \in \mathbb{R}^m$:

$$\mathsf{Softmax}(\vec{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{x_i - \max(\vec{x})}}{\sum_j e^{x_j - \max(\vec{x})}}.$$

As e^x and $1/x$ overflow easily, a common way is to evaluate the second expression so that exponentials are computed only on non-positive values and the reciprocal input is within the range $[1, m]$. Computing this expression requires an expensive $O(\log_2(m))$-round secure maximum computation.

Avoiding Private Maximum. To bypass the logarithmic round complexity, we begin by directly approximating the first expression (without the $\max(\vec{x})$ terms) using an ODE. This method starts with a constant initialization $\vec{y}_0 = \vec{1}/m$ and iteratively updates over t iterations as follows:

$$\vec{y}_i = \vec{y}_{i-1} + \frac{1}{t} \cdot \left(\vec{x} - \langle \vec{x}, \vec{y}_{i-1} \rangle \cdot \vec{1}\right) * \vec{y}_{i-1}.$$

Theorem 1 shows the correctness of the ODE approximation.

Theorem 1 ([37]). *Suppose* $\max(\vec{x}) - \min(\vec{x}) \le t$*:*

(i) *For all* $i \in [t+1]$, $\vec{y}_i$ *is a probability distribution, i.e.,* $(y_i)_j \in [0,1]$ *for all* $j \in [m]$ *and* $\vec{1}^T \vec{y}_i = 1$.
(ii) $(y_i)_j \le (y_i)_k \iff x_j \le x_k \ \forall i \in [t+1],\ j,k \in [m]$.
(iii) $\lim_{i \to +\infty} \vec{y}_i = \mathsf{Softmax}(\vec{x})$.

While this idea seems correct, it is far from enough to yield accurate results. Observe that the ODE approximation may not converge if the input assumption $\max(\vec{x}) - \min(\vec{x}) \le t$ is not met. In transformers, softmax inputs are unbounded, implying this assumption can be *violated easily* if t is set to a small value like 16. Determining a suitable t without prior knowledge of the model and data is challenging. In practice, a conservative choice is to set a large t for reasonable accuracy, such as 128 or 256. However, this is inefficient as the costs of privately running the ODE approximation scale linearly with t.

Ensuring the Correctness of ODE. We propose clipping the inputs to a predefined range $[a, b]$. Setting $t = b - a$ can then satisfy the input assumption, even when t is small. Clipping may cause errors when inputs fall outside the range, but selecting $[a, b]$ properly maintains the precision. Thus, we focus on choosing a suitable range for transformer inference.

Softmax in the attention layer converts weights that represent the relevance into a probability distribution, assigning probabilities in proportion to their relatedness. Taking NLP as an example, typically, a word is closely related to only a few other words in a sentence.[5] This observation suggests that most transformer inputs are small, but clipping the few large inputs causes larger errors as their exponents "dominate" the softmax denominator. So, we set a larger positive b to minimize errors from large inputs and a slightly negative a to include most small inputs. Specifically, we set $t = 16$, $a = -4$, $b = 12$.

[5]For instance, "it" in "The animal didn't cross the street because it was too tired" refers to "animal" rather than any other word (from Google Research).

Protocol 1: Softmax, $\Pi_{\mathsf{Softmax}}([\![\vec{x}]\!])$

```
Parameters: $P_0$ and $P_1$ know public values $t, a, b$.
Data: $P_0$ and $P_1$ hold shares of $\vec{x}$.
Result: $P_0$ and $P_1$ get shares of $\mathsf{Softmax}(\vec{x})$.
1  $[\![\vec{\delta}_a]\!] \,||\, [\![\vec{\delta}_b]\!] \leftarrow \Pi_{\mathsf{ReLU}}(a - [\![\vec{x}]\!] \,||\, [\![\vec{x}]\!] - b)$;
2  $[\![\vec{x}]\!] \leftarrow [\![\vec{x}]\!] + [\![\vec{\delta}_a]\!] - [\![\vec{\delta}_b]\!]$;        // clip $\vec{x}$ into $[a, b]$
3  $[\![\vec{x}]\!] \leftarrow (1/t)[\![\vec{x}]\!]$;        // multiply $1/t$ at first
4  $m \leftarrow \mathrm{len}([\![\vec{x}]\!])$;        // get length of $\vec{x}$
5  $P_0$: $[\![\vec{y}_0]\!]_0 \leftarrow \vec{1}/m$; $P_1$: $[\![\vec{y}_0]\!]_1 \leftarrow \vec{0}$;        // initialize $\vec{y}_0$
6  for $k = 1, \ldots, t$ do
7      $[\![\vec{a}]\!] \leftarrow \Pi_\times([\![\vec{x}]\!], [\![\vec{y}_{k-1}]\!])$;        // $\vec{a} \leftarrow \vec{x} * \vec{y}_{k-1}$
8      $[\![\vec{q}]\!] \leftarrow (\sum_i [\![a_i]\!])\vec{1}$;        // $\vec{q} \leftarrow \langle \vec{x}, \vec{y}_{k-1} \rangle \cdot \vec{1}$
9      $[\![\vec{s}]\!] \leftarrow \Pi_\times([\![\vec{q}]\!], [\![\vec{y}_{k-1}]\!])$;        // $\vec{s} \leftarrow \vec{q} * \vec{y}_{k-1}$
10     $[\![\vec{y}_k]\!] \leftarrow [\![\vec{y}_{k-1}]\!] + [\![\vec{a}]\!] - [\![\vec{s}]\!]$;
11 end
12 return $[\![\vec{y}_t]\!]$;
```

Secure Protocol for Softmax. Protocol 1 describes our softmax protocol. The inputs are first clipped into $[a, b]$ in Lines 1-2. Instead of multiplying $1/t$ in each iteration, we can do it once at the beginning (Line 3) without affecting the correctness. We get the input length from its shares locally (Line 4) for initializing $\vec{y}_0$ (Line 5) and perform iterative updates in Lines 6-11. Finally, we output $\vec{y}_t$ in Line 12.

Cost Analysis. Π_{Softmax} requires $\log_2(\ell)+2t+3 = 41$ rounds, including 1 Π_{ReLU} call and $2t$ calls to $\Pi_\times$.

Comparison to Existing Works. Many frameworks (*e.g.*, [19], [14], [27]) compute a sequence of maximum, exponential, and reciprocal. For $m = 128$ (common in NLP), CrypTen requires 116 rounds ($O(\log_2(m))$ comparisons for maximum, 8 rounds for exponential, and 28 rounds for reciprocal). Also, adopting naïve ODE approximation [37] with a larger $t = 128$ (and without clipping) takes $2t = 256$ rounds.

C. Private GELU

Most private transformer inference frameworks (*e.g.*, [13], [27]) utilize piecewise (high-order) polynomials to approximate GELU. We explore a fundamentally different approach, outperforming them in both accuracy and communication cost.

The starting point of our GELU protocol is the FS approximation and an efficient sine protocol from Zheng *et al.* [37], which provides a precise approximation (within a bounded input range) of non-linear functions with a sinusoidal shape (in the approximation range). Unfortunately, the GELU function is *not* sinusoidal, and its inputs are unbounded.

Designing Suitable Function. A natural idea in a concurrent work [14] is to approximate the $\mathsf{Erf}(x)$ function. However, recall that $\mathsf{GELU}(x) = 0.5x\left(1 + \mathsf{Erf}\left(x/\sqrt{2}\right)\right)$, computing $\mathsf{GELU}(x)$ from $\mathsf{Erf}(x)$ needs an extra multiplication by $0.5x$, increasing communication costs and amplifying errors for $x > 2$. We formulate $\delta(x) = \mathsf{sgn}(x)\left(\mathsf{GELU}(x) - \mathsf{ReLU}(x)\right)$,

Protocol 2: GELU, $\Pi_{\mathsf{GELU}}([\![x]\!])$

```
Parameters: $P_0$ and $P_1$ know public vectors $\vec{\beta}, \vec{k}$.
Data: $P_0$ and $P_1$ hold shares of $x$.
Result: $P_0$ and $P_1$ get shares of $\mathsf{GELU}(x)$.
1  $[\![x_{\mathsf{ReLU}}]\!] \leftarrow \Pi_{\mathsf{ReLU}}([\![x]\!])$;
2  $[\![x_{\mathsf{abs}}]\!] \leftarrow 2[\![x_{\mathsf{ReLU}}]\!] - [\![x]\!]$;
3  $[\![b]\!] \leftarrow \Pi_<([\![x_{\mathsf{abs}}]\!], 4)$;        // $b \leftarrow \mathbb{1}_{|x|<4}(x)$
4  $[\![\vec{x}_{\sin}]\!] \leftarrow \Pi_{\sin}(\vec{k}[\![x_{\mathsf{abs}}]\!])$;        // $\vec{x}_{\sin} \leftarrow \sin(\vec{k}|x|)$
5  $[\![\vec{\delta}]\!] \leftarrow \vec{\beta}[\![\vec{x}_{\sin}]\!]$;
6  $[\![x_\delta]\!] \leftarrow \sum_i [\![\delta_i]\!]$;        // $x_\delta \leftarrow \vec{1}^T\left(\vec{\beta}\sin(\vec{k}|x|)\right)$
7  return $[\![x_{\mathsf{ReLU}}]\!] + \Pi_\times([\![b]\!], [\![x_\delta]\!])$;
```

a function for a GELU characterization that eliminates $\mathsf{Erf}(x)$ and the extra multiplication to avoid these issues:

$$\begin{aligned}\mathsf{GELU}(x) &= \mathsf{ReLU}(x) + (\mathsf{GELU}(x) - \mathsf{ReLU}(x)) \\ &= \mathsf{ReLU}(x) + \mathsf{sgn}(|x|)\left(\mathsf{GELU}(|x|) - \mathsf{ReLU}(|x|)\right) \\ &= \mathsf{ReLU}(x) + \delta(|x|).\end{aligned}$$

The $\delta(x)$ function is sinusoidal near $x = 0$ and is close to 0 for larger $|x|$, making it ideal for accurate FS approximation. While $\delta(x)$ is similar to $\mathsf{GELU}(x) - \mathsf{ReLU}(x)$ in SIGMA [26], the latter is *not* sinusoidal. We adopt an FS-based approach, unlike the computation-intensive LUT protocol in SIGMA.

Our FS Approximation. To approximate $\delta(x)$, we use the following K-term FS[6] for the input range $(-4, 4)$:

$$\delta(x) \approx \mathbb{1}_{|x|<4}(x) \sum_{n=1}^{K} \beta_n \sin\left(\frac{n\pi x}{4}\right)$$

where $\vec{\beta} = [\beta_1, \ldots, \beta_K]$ is a vector of predefined constants:

$$\beta_n = \frac{1}{4}\int_{-4}^{4} \delta(u) \sin\left(\frac{n\pi u}{4}\right) du.$$

We express $\sum_{n=1}^{K} \beta_n \sin(n\pi x/4)$ as vector $\vec{1}^T\left(\vec{\beta}\sin(\vec{k}x)\right)$, where $\vec{k} = [\pi/4, \ldots, K\pi/4]$, for simplicity. In this work, we fix $K = 8$ and have $\vec{\beta} = [-0.0818, -0.0809, -0.0424, -0.0176, -0.0079, -0.0043, -0.0026, -0.0017]$.

Secure Protocol for GELU. Our protocol is described in Protocol 2. We compute $\mathsf{ReLU}(x)$ in Line 1, evaluate $|x| = 2\mathsf{ReLU}(x) - x$ locally in Line 2, and check if $x \in (-4, 4)$ by comparing $|x|$ with 4 in Line 3. We approximate $\delta(|x|)$ with FS in Lines 4-6 and output the result in Line 7.

Cost Analysis. Π_{GELU} requires $2\log_2(\ell) + 7 = 19$ rounds (1 Π_{ReLU} call, 1 $\Pi_<$ call, 1 $\Pi_{\sin}$ call, and 1 $\Pi_\times$ call).

Accuracy. Figure 3 compares our approximation to the actual GELU. The maximum and average (absolute) errors are 4.60×10^{-3} and 7.39×10^{-4} for the input range $[-5, 5]$, respectively.[7]

[6]We omit the constant and cosine terms (all equal 0 because $\delta(0) = 0$ and $\delta(x)$ is odd) used in general FS approximations for simplicity.

[7]Another common error measure for numerical approximations is units in last place (ULPs). For a numerical precision of 12 bits, our results show 19 maximum and 3 average ULPs, *cf.*, BOLT's 39 maximum and 5 average ULPs.

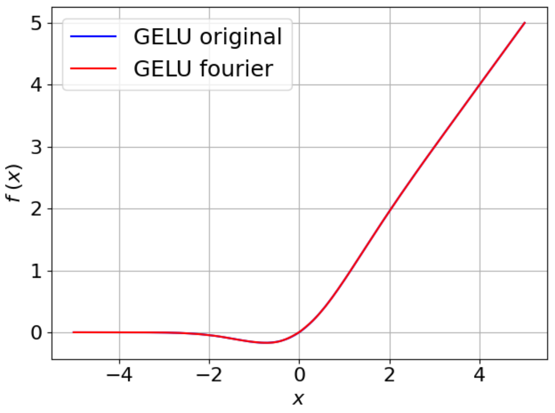

Fig. 3. Comparison of the original and FS-approximated GELU

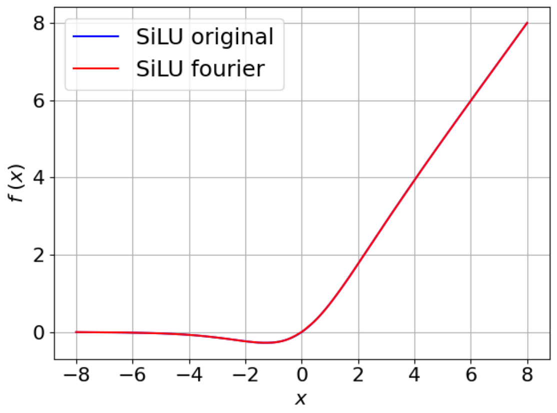

Fig. 4. Comparison of the original and FS-approximated SiLU

TABLE III
PERFORMANCE COMPARISON OF GELU PROTOCOLS

	SecFormer [14]	BOLT [13]	SHAFT
Maximum error	1.94×10^{-2}	9.36×10^{-3}	4.60×10^{-3}
Average error	5.00×10^{-3}	1.18×10^{-3}	7.39×10^{-4}
$\Pi_<$, Π_{ReLU} calls	2, 0	1, 1	1, 1
$\Pi_\times$, $\Pi_{\sin}$ calls	4, 1	3, 0	1, 1
Rounds*	$2\log_2(\ell)+9$	$2\log_2(\ell)+8$	$2\log_2(\ell)+7$

*For a fair comparison, we assume the same $\Pi_{\mathsf{ReLU}}, \Pi_<, \Pi_\times$ protocols from CrypTen [19] and the same $\Pi_{\sin}$ protocol [37] are used.

Table III compares the performance of our approach to the SoTA and related methods. Compared to the piecewise polynomial in BOLT [13], our protocol saves one round while reducing the maximum and average error by 51% and 37%, respectively. Compared to the FS approximation in SecFormer [14], our protocol lowers the maximum error by 76%, the average error by 85%, and reduces two rounds.

Extension to Other Activations. Our characterization for FS approximations is not only useful for GELU but also for other ReLU-like activations, notably $\mathsf{SiLU}(x) = x * \sigma(x)$, where $\sigma(x) = 1/(1+e^{-x})$. When using a 12-term FS to approximate $\mathsf{sgn}(x)(\mathsf{SiLU}(x) - \mathsf{ReLU}(x))$ within the input range $(-8, 8)$, the maximum and average errors are 5.40×10^{-3} and 8.89×10^{-4}, respectively (see Figure 4 for a comparison).

D. Private Embedding

The embedding layer in PyTorch is designed to accept inputs encoded as integer indices. Unfortunately, existing private transformer inference frameworks often assume inputs are one-hot vectors. Implementing a converted embedding layer to accommodate this assumption impairs code readability and burdens data owners with an extra one-hot vector conversion.

We address this limitation by introducing the first private embedding protocol that directly accepts indices as inputs, along with an index-to-one-hot protocol using precomputed pair $([\![r]\!], [\![\vec{e}_r]\!])$ to translate arithmetic operations on scalars to cyclic rotations on one-hot vectors [22].

Protocol 3: Index to One-Hot, $\Pi_{\mathsf{Idx2v}}([\![i]\!])$

```
Data: $P_0$ and $P_1$ hold shares of $i$.
Result: $P_0$ and $P_1$ get shares of $\vec{e}_i$.
Common Randomness: $P_0$ and $P_1$ hold shares of $r, \vec{e}_r$.
1 $[\![\delta]\!] \leftarrow [\![i]\!] - [\![r]\!]$;
2 Reconstruct $\delta$;
3 $[\![\vec{e}_i]\!] \leftarrow [\![\vec{e}_r]\!] \ggg \delta$;
4 return $[\![\vec{e}_i]\!]$;
```

Protocol 4: Embedding Layer, $\Pi_{\mathsf{Emb}}([\![i]\!], [\![\mathbf{W}^E]\!])$

```
Data: $P_0$ and $P_1$ hold shares of $i, \mathbf{W}^E$ (corr. to $T_{\mathsf{Emb}}$).
Result: $P_0$ and $P_1$ get shares of $T_{\mathsf{Emb}}[i] = \vec{e}_i \mathbf{W}^E$.
1 $[\![\vec{e}_i]\!] \leftarrow \Pi_{\mathsf{Idx2v}}([\![i]\!])$;        // get one-hot vector corr. to $i$
2 return $\Pi_\times([\![\vec{e}_i]\!], [\![\mathbf{W}^E]\!])$;        // "table lookup" by mult.
```

Converting Index to One-Hot Vector. Protocol 3 converts a shared index $[\![i]\!]$ to one-hot vector $[\![\vec{e}_i]\!]$. P_0 and P_1 first hide i by $[\![\delta]\!] = [\![i]\!] - [\![r]\!]$ (Line 1) and reconstruct δ (Line 2). They then apply right cyclic rotations to $[\![\vec{e}_r]\!]$ (Line 3).

Correctness. To see the correctness of the protocol, consider

$$\begin{aligned}[\![\vec{e}_i]\!]_0 + [\![\vec{e}_i]\!]_1 &= ([\![\vec{e}_r]\!]_0 \ggg \delta) + ([\![\vec{e}_r]\!]_1 \ggg \delta) \\ &= ([\![\vec{e}_r]\!]_0 + [\![\vec{e}_r]\!]_1) \ggg \delta \\ &= \vec{e}_r \ggg (i - r) = \vec{e}_i.\end{aligned}$$

Secure Protocol for Embedding. Our embedding protocol (Protocol 4) invokes Protocol 3 to obtain a one-hot vector $\vec{e}_i$ from an index i and then multiplies it by $\mathbf{W}^E$.

Cost Analysis. Π_{Idx2v} requires 1 round for reconstructing the blinded index, while Π_{Emb} takes 2 (1 for Π_{Idx2v} and 1 for $\Pi_\times$).

V. EVALUATION

A. Experiment Setup

We perform experiments on a server with Intel Xeon Gold 5318Y CPUs at 2.10 GHz, two NVIDIA A40 GPUs, and 256 GB of RAM. Like CrypTen [19], each party is assigned a GPU, and all experiments run in separate processes on the same machine. We implement our secure protocols on CrypTen [19].

TABLE IV
ARCHITECTURE HYPERPARAMETERS OF MODELS

Model	# of Parameters	B	h	d_{model}	d_{Emb}
BERT-base	110M	12	12	768	30522
BERT-large	340M	24	16	1024	30522
GPT-2	117M	12	12	768	50257
ViT-base	86.6M	12	12	768	-

B: Number (#) of transformer blocks. h: # of attention heads. d_{model}: Dimension of intermediate layers. d_{Emb}: Size of T_{Emb}.

Models and Datasets. Using pretrained weights from Hugging Face [23], we evaluate the cost of private transformer inference for four models (BERT-base, BERT-large [2], GPT-2 [24], and ViT-base [4]). Table IV details the models. Like prior works [13], [26], [27], we run BERT and GPT-2 on length-128 text sequences and ViT-base on 224×224 RGB images. Moreover, we evaluate the private inference accuracy of BERT-base (fine-tuned for 3 epochs with the AdamW optimizer and a learning rate of 2×10^{-5}) on the QNLI, CoLA, and SST-2 classification tasks from the GLUE benchmark [25].

Metrics. We evaluate the efficiency of SHAFT by running time and communication cost. Like most private frameworks (recent ones included [13], [27]), we do not separate offline and online costs (CrypTen comes with no easy phase separation). Each query takes offline and online computation, but our online cost is much lower than our offline cost, like other frameworks. The accuracy metric is dataset-specific and will be specified when results are reported. The results are averaged over 10 runs.

B. Comparison of Non-linear Protocols

Secure Softmax. We compare our softmax protocol with the SoTA approach from CrypTen [19] adopted in many private transformer inference frameworks (*e.g.*, [14], [27]) and the naïve ODE approximation by Zheng *et al.* [37]. Table V shows the costs of evaluating an ℓ-input softmax, including time, communicated bytes (Comm.), and rounds (Rd.). For ODE-based approaches, the running time and rounds remain constant, and the communication increases linearly with ℓ. While having the same theoretical complexity [37], our protocol is $6\times$ faster, saving 76% in communication and 84% in rounds. Compared with CrypTen, where running time and rounds grow logarithmically with ℓ, and communication cost increases linearly with ℓ, our protocol is 2.2-2.5$\times$ faster, saves 61-67% in rounds, and incurs only 1.3-1.4$\times$ more communication.

Secure GELU. We evaluate our GELU protocol on two input sizes, $(128, 3072)$ (used in BERT-base, GPT-2, and ViT-base) and $(128, 4096)$ (used in BERT-large), comparing it to BOLT [13], the SoTA with a 3-piece degree-4 polynomial, and SecFormer [14], using an FS approximation like SHAFT. From Table VI, our protocol is 1.05-1.11$\times$ faster and saves 5-6% communication over these two works. It also requires one fewer round than BOLT and two fewer than SecFormer. Moreover, Table III shows that our protocol reduces maximum and average errors by 51-76% and 37-85%, respectively. These results confirm that our secure GELU protocol outperforms existing works in running time, communication, and accuracy.

TABLE V
COST OF SECURE SOFTMAX (TIME IN S, COMM. IN MB)

Framework	$\ell = 32$			$\ell = 64$		
	Time	**Comm.**	**Rd.**	**Time**	**Comm.**	**Rd.**
CrypTen [19]	0.3230	0.0453	105	0.3470	0.0843	115
Zheng *et al.* [37]	0.9398	0.2500	256	0.9322	0.5000	256
SHAFT	0.1464	0.0596	41	0.1450	0.1191	41
Framework	$\ell = 128$			$\ell = 256$		
	Time	**Comm.**	**Rd.**	**Time**	**Comm.**	**Rd.**
CrypTen [19]	0.3662	0.1817	116	0.3835	0.3379	126
Zheng *et al.* [37]	0.9306	1.0000	256	0.9533	2.0000	256
SHAFT	0.1420	0.2383	41	0.1505	0.4766	41

TABLE VI
COST OF SECURE GELU (TIME IN S, COMM. IN MB)

Framework	$(128, 3072)$			$(128, 4096)$		
	Time	**Comm.**	**Rd.**	**Time**	**Comm.**	**Rd.**
BOLT [13]	0.2363	372.0	20	0.2970	496.0	20
SecFormer [14]	0.2448	378.0	21	0.3057	504.0	21
SHAFT	0.2203	354.0	19	0.2818	472.0	19

C. Comparison of Private Transformer Inference

Table VII compares SHAFT to SIGMA [26], BumbleBee [27], and BOLT [13]. FSS-based SIGMA uses highly optimized LUTs and smaller bitwidths. BOLT is the SoTA HE-based work, while BumbleBee claims performance improvement over BOLT. As the code of SIGMA and BumbleBee were not available[8] and we could not compile the code of BOLT, we quote their results and detail how we ensure fair comparisons.[9]

Accurately measuring the empirical running time requires isolating the uncontrollable fluctuations introduced by inter-process communication. The running time of SHAFT is calculated as: computation time$+ 2 \times$communication$\div$bandwidth$+$ rounds $\times$ latency. The "2" refers to the 2 parties in the protocols. Unless stated otherwise, our reported results are based on the same network settings as BumbleBee [27]: a local-area network (LAN) with 1 GBps bandwidth and 0.5 ms latency and a wide-area network (WAN) with (400 MBps, 4 ms).

FSS-based frameworks like SIGMA need large offline key transfers. SIGMA runs on a very high-performant LAN with (9.4 Gbps, 0.05 ms). For 1 Gbps LAN, the estimated running-time lower bound is: computation time$+$key generation time$+$ $2 \times$ (communication $+$ key size) $\div$ bandwidth.

[8]The acceptance of these two preprints became public only shortly before our submission deadline, with the latter appearing just two days prior.

[9]Fair experimental comparisons across different prototypes are challenging [6], exacerbated by the cost and time-consuming nature of cryptographic ML computation. Compared to the HE-based BumbleBee, SS-based solutions (counting both offline and online costs) we adopted are known to be much faster [6]. Communication comparisons are more straightforward, as all parameters are known, and their determination is simple and deterministic.

TABLE VII
COST OF PRIVATE INFERENCE (TIME IN S, COMM. IN GB)

Model	Framework	Time		Comm.
		LAN	WAN	
BERT-base	**SIGMA** [26]	†38.42	-	17.68
	BumbleBee [27]	153.00	291.60	6.40
	BOLT [13]	‡190.80	‡1563.00	59.61
	SHAFT	28.60	66.46	10.46
BERT-large	**SIGMA** [26]	†103.30	-	47.69
	BumbleBee [27]	371.40	588.60	16.37
	SHAFT	77.13	176.21	28.46
GPT-2 ($\ell = 64$)	**BumbleBee** [27]	88.80	123.00	2.77
	SHAFT	19.32	42.37	5.76
GPT-2 ($\ell = 128$)	**SIGMA** [26]	†32.68	-	14.99
	SHAFT	†32.59	73.47	11.27
ViT-base	**BumbleBee** [27]	239.40	-	11.56
	SHAFT	45.66	108.24	18.41

LAN: (1 Gbps, 0.5 ms), WAN: (400 Mbps, 4 ms), unless otherwise specified.
†: Here, we estimate running time by assuming (1 Gbps, 0.05 ms) for LAN.
‡: LAN uses (3 Gbps, 0.8 ms) and WAN uses (200 Mbps, 40 ms).

Comparison with SIGMA. Despite using a larger bitwidth (64 vs. 50), SHAFT is 1.3× faster for BERT models and has comparable running time for GPT-2 on LAN, while reducing communication by 25-41%. SIGMA lacks WAN results, but we anticipate greater running-time improvements, as the key transfer time increases substantially on slower WANs.

Comparison with BumbleBee. SS-based frameworks generally compute faster but require more communication than HE-based ones [6].[10] SHAFT, as an SS-based framework, is 4.6-5.3× faster on LAN and 2.9-4.4× faster on WAN while increasing communication by 59-108%. The running time improvements on slower WAN demonstrate that SHAFT achieves better computation-communication tradeoff.[11]

Comparison with BOLT. SHAFT is 6.7× faster than BOLT on LAN (with a smaller bandwidth). On WAN with (200 Mbps, 40 ms), the running time of SHAFT is 173.83 s, 9.0× faster than BOLT. Moreover, it saves 82% of communication.

D. Performance Breakdown

Figure 5 shows SHAFT's breakdown of computation time, communication, and rounds. Private transformer inference mainly uses five modules – embedding, matrix multiplication (MatMul), softmax, GELU, and LayerNorm. The "Other" category includes most non-interactive operations, the tanh function in BERT's classifier (using the default approximation and parameters from CrypTen [19]), and a (linear) convolution performed at the beginning of ViT-base.

TABLE VIII
ACCURACY ON THE GLUE BENCHMARK

Dataset	Size	Metric	Plaintext	SHAFT
QNLI	5463	Accuracy	90.77%	90.38%
SST-2	872	Accuracy	92.66%	92.20%
CoLA	1043	Matthews correlation	0.5778	0.5685

Computation Time Breakdown. MatMul dominates, taking 58-77% of computation time. CrypTen [19] utilizes GPU for efficient MatMul but avoids overflow by decomposing 64-bit floating-point values into 4 partitions, requiring numerous cross-term multiplications [6]. We identify these multiplications as the major bottleneck. Embedding (in BERT/GPT)[12] accounts for 7-19% of the time due to large MatMuls. In contrast, softmax and GELU only take 8-11% and 1-2% of the time, respectively. These findings highlight the effectiveness of our non-linear protocols in reducing computation.

Communication Breakdown. Softmax and GELU dominate communication, accounting for 38-55% and 35-40%, respectively [13], [27]. Our protocols can be further adapted to smaller bitwidths (fixed to 64 for now) to significantly reduce communication, to be elaborated in Section VII.

Rounds Breakdown. With our constant-round protocol, the softmax module, while still requiring a moderate number of rounds (33-34% of the total), is no longer dominating. Instead, the LayerNorm module using iterative methods to approximate $1/\sqrt{x}$ accounts for the largest portion (43-45%) of rounds.

Our breakdowns reveal promising directions. Designing a more efficient secure MatMul protocol on GPU could reduce computation time by >50%. Also, existing protocols for $1/\sqrt{x}$ (*e.g.*, [19]) target general NNs. We could develop transformer-specific protocols, aiming to reduce round complexity.

E. Accuracy

We perform private inference on the full validation set (*cf.*, a subset [27]) of three GLUE classification tasks (QNLI, CoLA, SST-2) using BERT-base. Table VIII confirms that SHAFT achieves accuracy comparable to the plaintext setting.

VI. SECURITY ANALYSIS

We prove the security of SHAFT[13] following the ideal/real-world paradigm [54]. Table IX defines the ideal functionalities for our protocols[14] and building blocks[15] [19], [37].

[10]For SS-based frameworks, communication dominates running time since SS computations are almost as efficient as plaintext. In SHAFT, communication accounts for 70-80% of the time on LAN and 85-90% on WAN.

[11]Our new method relies on iterative approximations requiring multiple rounds. A hybrid approach [6] combining existing HE-based optimizations (*e.g.*, [27]) with our SS-based secure protocols for non-linear layers could negate our unique advantages, as such integration risks inheriting the drawbacks of both paradigms due to the high computational cost of HE operations.

[12]Recall that ViT for computer vision does not have an embedding layer.

[13]The security of our private inference should follow straightforwardly [26] from our protocols, as transformers involve sequential compositions and concurrent self-compositions of these protocols preserve security [52], [53].

[14]Since softmax and GELU are not computed exactly, their ideal (approximated) functionalities match their real protocols, with the underlying approximation protocols replaced by the corresponding ideal functionalities, rather than by the protocols for exact computation.

[15]CrypTen has two truncation protocols: a non-interactive one for 2PC and an interactive one using precomputed randomness for MPC with ≥3 servers. Li *et al.* [40] show that the latter is insecure due to the misuse of precomputed randomness. SHAFT uses the first one, which remains secure.

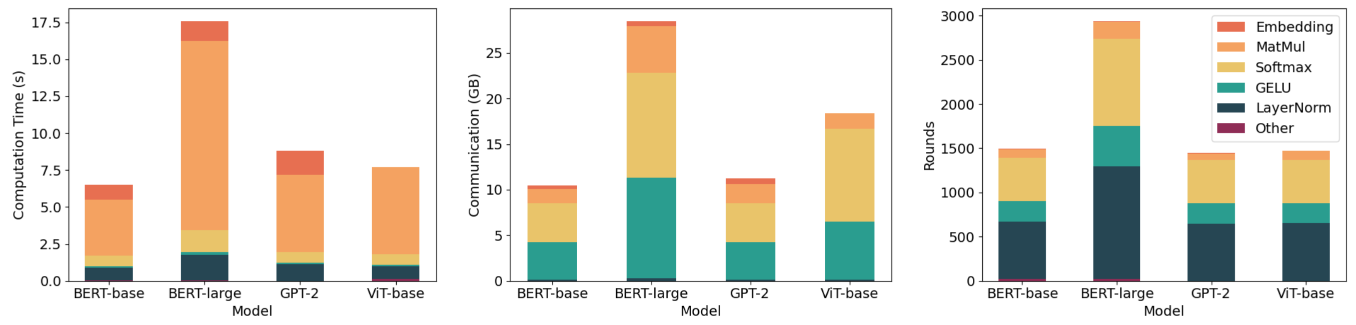

Fig. 5. Breakdown of computation time (left), communication (middle), and rounds (right) for SHAFT on four transformers

TABLE IX
DEFINITIONS OF IDEAL FUNCTIONALITIES

Ideal Functionality	Input/Output Description	Proof
Multiplication $\mathcal{F}_\times$	$[\![x]\!], [\![y]\!] \to [\![xy]\!]$	[55, §4.1]
Less-than $\mathcal{F}_<$	$[\![x]\!], [\![y]\!]$ or $y \to [\![\mathbb{1}_{x<y}(x, y)]\!]$	[19, App. B]
ReLU $\mathcal{F}_{\mathsf{ReLU}}$	$[\![x]\!] \to [\![\mathsf{ReLU}(x)]\!]$	[19, App. B]
Sine $\mathcal{F}_{\sin}$	$[\![x]\!] \to [\![\sin(x)]\!]$	[37, App. C]
Softmax $\mathcal{F}_{\mathsf{Softmax}}$	$[\![\vec{x}]\!] \to [\![\mathsf{Softmax}(\vec{x})]\!]$	§VI-A
GELU $\mathcal{F}_{\mathsf{GELU}}$	$[\![x]\!] \to [\![\mathsf{GELU}(x)]\!]$	§VI-B
Index to one-hot $\mathcal{F}_{\mathsf{Idx2v}}$	$[\![i]\!] \to [\![\vec{e}_i]\!]$	§VI-C
Embedding $\mathcal{F}_{\mathsf{Emb}}$	$[\![i]\!], [\![\mathbf{W}^E]\!] \to [\![\vec{e}_i \mathbf{W}^E]\!]$	§VI-C

Definition 1 ([54])**.** *Let $\mathcal{F} = (\mathcal{F}_0, \mathcal{F}_1)$ be a functionality, Π be a protocol, view_i^Π be the view of P_i in the execution of Π, out^Π be the output of Π, and $\mathsf{in} = (\mathsf{in}_0, \mathsf{in}_1)$ be the input of $\mathcal{F}$ and Π. Π securely realizes $\mathcal{F}$ in the presence of a static semi-honest adversary if there exist PPT simulators $\mathcal{S}_0, \mathcal{S}_1$ s.t.*

- $\{\mathcal{S}_0(\mathsf{in}_0, \mathcal{F}_0(\mathsf{in})), \mathcal{F}(\mathsf{in})\} \overset{c}{\equiv} \{\mathsf{view}_0^\Pi(\mathsf{in}), \mathsf{out}^\Pi(\mathsf{in})\}$,
- $\{\mathcal{S}_1(\mathsf{in}_1, \mathcal{F}_1(\mathsf{in})), \mathcal{F}(\mathsf{in})\} \overset{c}{\equiv} \{\mathsf{view}_1^\Pi(\mathsf{in}), \mathsf{out}^\Pi(\mathsf{in})\}$,

for any in*, where $\overset{c}{\equiv}$ denotes computational indistinguishability.*

A. Security of Private Softmax

Theorem 2. *Suppose Π_{ReLU}, $\Pi_\times$ securely realizes $\mathcal{F}_{\mathsf{ReLU}}$, $\mathcal{F}_\times$, respectively. Π_{Softmax} (Protocol 1) securely realizes the functionality $\mathcal{F}_{\mathsf{Softmax}}$ against static semi-honest adversaries.*

Proof. Let in be arbitrary and $\{\Pi_{\mathsf{ReLU}}, \Pi_\times\}$ securely realizes $\{\mathcal{F}_{\mathsf{ReLU}}, \mathcal{F}_\times\}$. By definition, $\mathcal{F}_{\mathsf{Softmax}}(\mathsf{in}) \overset{c}{\equiv} \mathsf{out}^{\Pi_{\mathsf{Softmax}}}(\mathsf{in})$.[16] Observe that Π_{Softmax} sequentially calls Π_{ReLU} and $\Pi_\times$. By the sequential composition theorem [52, Corollary 7], $\mathcal{S}_i(\mathsf{in}_i, \mathcal{F}_i(\mathsf{in})) \overset{c}{\equiv} \mathsf{view}_i^{\Pi_{\mathsf{Softmax}}}(\mathsf{in})$ for $i \in \{0, 1\}$. □

B. Security of Private GELU

Theorem 3. *Suppose $\{\Pi_{\mathsf{ReLU}}, \Pi_<, \Pi_{\sin}, \Pi_\times\}$ securely realizes $\{\mathcal{F}_{\mathsf{ReLU}}, \mathcal{F}_<, \mathcal{F}_{\sin}, \mathcal{F}_\times\}$. Π_{GELU} (Protocol 2) securely realizes $\mathcal{F}_{\mathsf{GELU}}$ in the presence of static semi-honest adversaries.*

Proof. Let in be arbitrary and $\{\Pi_{\mathsf{ReLU}}, \Pi_<, \Pi_{\sin}, \Pi_\times\}$ securely realizes $\{\mathcal{F}_{\mathsf{ReLU}}, \mathcal{F}_<, \mathcal{F}_{\sin}, \mathcal{F}_\times\}$. By definition, $\mathcal{F}_{\mathsf{GELU}}(\mathsf{in}) \overset{c}{\equiv} \mathsf{out}^{\Pi_{\mathsf{GELU}}}(\mathsf{in})$. Since Π_{GELU} sequentially invokes Π_{ReLU}, $\Pi_<$, $\Pi_{\sin}$, and $\Pi_\times$, by the sequential composition theorem [52], $\mathcal{S}_i(\mathsf{in}_i, \mathcal{F}_i(\mathsf{in})) \overset{c}{\equiv} \mathsf{view}_i^{\Pi_{\mathsf{GELU}}}(\mathsf{in})$ for $i \in \{0, 1\}$. □

[16]While Definition 1 is defined in a joint-distribution manner, we can analyze the former and latter distributions separately without affecting the security [54], as all our functionalities are deterministic.

C. Security of Private Embedding

Theorem 4. *Π_{Idx2v} (Protocol 3) securely realizes $\mathcal{F}_{\mathsf{Idx2v}}$ in the presence of any static semi-honest PPT adversaries.*

Proof. Let $\mathsf{in} = ([\![i]\!], [\![r]\!], [\![\vec{e}_r]\!])$ be arbitrary. The correctness $\mathcal{F}_{\mathsf{Idx2v}}(\mathsf{in}) \overset{c}{\equiv} \mathsf{out}^{\Pi_{\mathsf{Idx2v}}}(\mathsf{in})$ is explained in Section IV-D. We first consider P_0 is corrupted. We have $\mathsf{view}_0^{\Pi_{\mathsf{Idx2v}}}(\mathsf{in}) = ([\![i]\!]_0, [\![\delta]\!]_1)$. With $\mathsf{in}_0 = ([\![i]\!]_0, [\![r]\!]_0, [\![\vec{e}_r]\!]_0)$ and $\mathcal{F}_0(\mathsf{in}) = [\![\vec{e}_i]\!]_0$, $\mathcal{S}_0$ can enumerate all $\mathsf{len}(\vec{e}_i)$ possible values of δ and find the one that satisfies $[\![\vec{e}_i]\!]_0 = [\![\vec{e}_r]\!]_0 \ggg \delta$ (Line 3) and evaluate $[\![\delta]\!]_1 = \delta - [\![\delta]\!]_0$. Thus, $\mathcal{S}_0(\mathsf{in}_0, \mathcal{F}_0(\mathsf{in})) \overset{c}{\equiv} \mathsf{view}_0^{\Pi_{\mathsf{Idx2v}}}(\mathsf{in})$. By symmetry, $\mathcal{S}_1(\mathsf{in}_1, \mathcal{F}_1(\mathsf{in})) \overset{c}{\equiv} \mathsf{view}_1^{\Pi_{\mathsf{Idx2v}}}(\mathsf{in})$ holds as well. □

Theorem 5. *Suppose Π_{Idx2v}, $\Pi_\times$ securely realizes $\mathcal{F}_{\mathsf{Idx2v}}$, $\mathcal{F}_\times$, respectively. Π_{Emb} (Protocol 4) securely realizes $\mathcal{F}_{\mathsf{Emb}}$ in the presence of static semi-honest adversaries.*

Proof. Let in be arbitrary and $\{\Pi_{\mathsf{Idx2v}}, \Pi_\times\}$ securely realizes $\{\mathcal{F}_{\mathsf{Idx2v}}, \mathcal{F}_\times\}$. The correctness $\mathcal{F}_{\mathsf{Emb}}(\mathsf{in}) \overset{c}{\equiv} \mathsf{out}^{\Pi_{\mathsf{Emb}}}(\mathsf{in})$ holds by definition. Observe that Π_{Emb} sequentially invokes Π_{Idx2v} and $\Pi_\times$. By the sequential composition theorem [52], we have $\mathcal{S}_i(\mathsf{in}_i, \mathcal{F}_i(\mathsf{in})) \overset{c}{\equiv} \mathsf{view}_i^{\Pi_{\mathsf{Emb}}}(\mathsf{in})$ for $i \in \{0, 1\}$. □

VII. Optimizations for Mixed-Bitwidth Settings

We instantiate SHAFT on fixed-bitwidth CrypTen [19]. When instantiating SHAFT using mixed-bitwidth frameworks like SIRNN [28], communication costs can be further reduced. Below, we discuss how our protocols can use a shorter bitwidth with other optimizations enabled by our design.

A. Optimizations for Softmax

Secure ReLU. Recall that in transformers, softmax is always preceded by a MatMul, and followed by a truncation of $t = 16$ bits. Thus, the (effective) bitwidth for the input of ReLU is $n - t = 48$ (*i.e.*, the high-end 16 bits are ignored). By the definition of ReLU, the output bitwidth is 48 as well.

Secure Multiplication. By Theorem 1, we have $(y_k)_i \in [0, 1]$ and $\vec{1}^T \vec{y}_i = 1$. For $(a, b) = (-4, 12)$, after input clipping, we have $|x_i| < 12$, implying $|\langle \vec{x}, \vec{y}_{i-1} \rangle| < 12$. So, the bitwidth of $\vec{y}, \vec{x}, \langle \vec{x}, \vec{y}_{i-1} \rangle$, calculated by 1 (sign bit) + 16 (precision) + $\lceil \log_2(\text{upper bound of value}) \rceil$, are $18, 21, 21$, respectively.

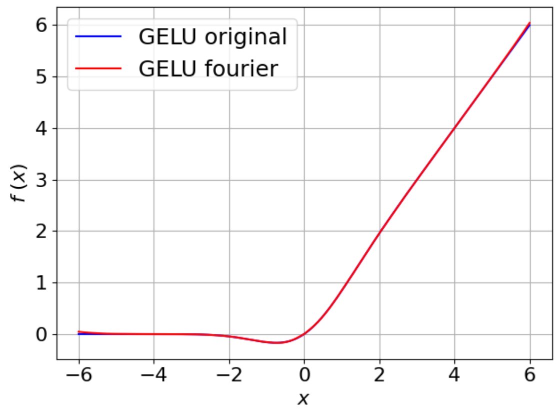

Fig. 6. The original and FS-approximated GELU with comparison errors

B. Optimizations for GELU

Secure ReLU. Similar to softmax, the bitwidth of ReLU is 48 since GELU is also preceded by a MatMul and a truncation.

Secure Comparison. Besides ignoring the high-end 16 bits, our GELU formulation allows skipping some low-end bits, despite potential comparison errors. For example, ignoring the last 16 bits could round both 2.01 and 2.99 to 2, leading to an incorrect comparison of $[\![4.98]\!] = ([\![4.98]\!]_0 = 2.99, [\![4.98]\!]_1 = 1.99)$ with 4. Fortunately, our FS approximation remains accurate not only within $[-4, 4]$ but also over a broader range (Figure 6 shows a plot by assuming $[\![1]\!] \leftarrow \Pi_<([\![x]\!], 4)$ for $x \in [-6, 6]$), tolerating such errors. With optimizations at both ends, the input bitwidth of comparison can be reduced to 32.

Secure Sine. As 2^{64} is divisible by the period of our FS (*i.e.*, 8), both parties can take a modulo 8 (floating-point) operation on their input shares without affecting the result. This reduces the sine input bitwidth to 19. Note that the sign bit is not needed because the shares are in $[0.0, 8.0)$.

Secure Multiplication. The outputs of $\Pi_<([\![x]\!], 4)$ and our FS lie in $\{0, 1\}$ and $[-0.17, 0.17]$, respectively. So, the bitwidths of b and x_δ are 17 and 15, respectively.

VIII. Conclusion and Future Works

We introduce SHAFT, a secret-sharing-based framework for secure, handy, accurate, and fast transformer inference.[17] It is powered by the first constant-round softmax protocol for transformers using a new input-clipped-ODE approach, and an efficient GELU protocol derived from a characterization tailored for FS approximations, which is also applicable to SiLU. We also present the first protocol that directly handles secret-shared indices for inputs of embedding layers.

SHAFT outperforms state-of-the-art private transformer inference frameworks in computation and communication, while maintaining plaintext-comparable accuracy verified through empirical analyses, and providing formal security guarantees.

[17]We hope SHAFT embodies the literary essence of a “shaft of light”—delivering sharp and illuminating responses with the “lightning” speed and precision of a well-trained transformer model, all while safeguarding privacy.

As the first private transformer framework with a constant-round softmax protocol and an accurate GELU protocol that surpasses prior arts in computation and communication costs, SHAFT offers a strong foundation for further advancements.

We have open-sourced our code and implemented the conversion from PyTorch to PPML for transformer-specific layers, enabling seamless import of pretrained transformers from the popular Hugging Face library. This bridges our scientific work with real-world applications, addressing the pressing need for privacy-preserving solutions in widespread transformer inference deployments and paving the way for future research and development by machine learning practitioners to design and deploy cryptographically secure transformer models.

We conclude by sharing two directions for future work:

Private Transformer Fine-Tuning is an important future direction. Fine-tuning pretrained models like BERT [2] on small, task-specific datasets is often necessary, but such datasets may contain sensitive data. It is crucial to support fine-tuning without revealing the model or data [56]. Unlike inference, fine-tuning requires computing the derivatives of non-linear functions. Our non-linear protocols perform linear operations and comparisons, with secure derivative computations implemented on CrypTen [19]. The only missing piece is an efficient protocol for computing $\cos(x)$ (*i.e.*, the derivative of $\sin(x)$), which can be computed via $\cos(x) = \sin(\pi/2 - x)$. While extending SHAFT to private fine-tuning is technically feasible, we leave its implementation and evaluation to future work.

Security against Malicious Adversaries is desirable in some use cases. Existing works provide malicious security for private CNN training and inference using replicated SS [57] or customized SS [58]. Potential extensions of SHAFT include instantiating our protocol within these frameworks or using authenticated SS [48] to defend against malicious adversaries.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *NeurIPS*, 2017.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *NAACL-HLT (1)*, 2019.

[3] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018, OpenAI blog.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *ICLR*, 2021.

[5] L. K. L. Ng and S. S. M. Chow, “GForce: GPU-friendly oblivious and rapid neural network inference,” in *USENIX Security Symposium*, 2021.

[6] ——, “SoK: Cryptographic neural-network computation,” in *S&P*, 2023.

[7] T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei, “THE-X: Privacy-preserving transformer inference with homomorphic encryption,” in *ACL (Findings)*, 2022.

[8] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, “Iron: Private inference on transformers,” in *NeurIPS*, 2022.

[9] D. Li, R. Shao, H. Wang, H. Guo, E. P. Xing, and H. Zhang, “MPC-Former: Fast, performant and private transformer inference with MPC,” in *ICLR*, 2023.

[10] Y. Akimoto, K. Fukuchi, Y. Akimoto, and J. Sakuma, “Privformer: Privacy-preserving transformer with MPC,” in *EuroS&P*, 2023.

[11] M. Zheng, Q. Lou, and L. Jiang, “Primer: Fast private transformer inference on encrypted data,” in *DAC*, 2023.

[12] W. Zeng, M. Li, W. Xiong, T. Tong, W. Lu, J. Tan, R. Wang, and R. Huang, "MPCViT: Searching for accurate and efficient MPC-friendly vision transformer with heterogeneous attention," in *ICCV*, 2023.

[13] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," in *S&P*, 2024.

[14] J. Luo, Y. Zhang, Z. Zhang, J. Zhang, X. Mu, H. Wang, Y. Yu, and Z. Xu, "SecFormer: Fast and accurate privacy-preserving inference for transformer models via SMPC," in *ACL (Findings)*, 2024.

[15] H. W. H. Wong, J. P. K. Ma, D. P. H. Wong, L. K. L. Ng, and S. S. M. Chow, "Learning DNN model with error - Exposing the hidden model of BAYHENN," in *IJCAI*, 2020.

[16] X. Wang, R. B. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *CVPR*, 2018.

[17] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *ICLR*, 2021.

[18] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *CCS*, 2020.

[19] B. Knott, S. Venkataraman, A. Y. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "CrypTen: Secure multi-party computation meets machine learning," in *NeurIPS*, 2021.

[20] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," arXiv 2302.13971, 2023.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[22] K. Storrier, A. Vadapalli, A. Lyons, and R. Henry, "Grotto: Screaming fast (2 + 1)-PC for $\mathbb{Z}_{2^n}$ via (2, 2)-DPFs," in *CCS*, 2023.

[23] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *EMNLP (Demos)*, 2020.

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019, OpenAI blog.

[25] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *ICLR (Poster)*, 2019.

[26] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "SIGMA: Secure GPT inference with function secret sharing," *Proc. Priv. Enhancing Technol. (PETS)*, no. 4, 2024.

[27] W. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "BumbleBee: Secure two-party inference framework for large transformers," in *NDSS*, 2025.

[28] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "SiRnn: A math library for secure RNN inference," in *S&P*, 2021.

[29] Y. Dong, W. Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W. Chen, "PUMA: Secure inference of LLaMA-7B in five minutes," arXiv 2307.12533, 2023.

[30] X. Hou, J. Liu, J. Li, Y. Li, W. Lu, C. Hong, and K. Ren, "CipherGPT: Secure two-party GPT inference," Cryptology ePrint 2023/1147, 2023.

[31] Y. Ding, H. Guo, Y. Guan, W. Liu, J. Huo, Z. Guan, and X. Zhang, "East: Efficient and accurate secure transformer framework for inference," arXiv 2308.09923, 2023.

[32] M. Yuan, L. Zhang, and X. Li, "Secure transformer inference protocol," Cryptology ePrint 2023/1763, 2023, version 20240508:021841.

[33] J. Zhang, X. Yang, L. He, K. Chen, W. Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," in *NDSS*, 2025.

[34] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *S&P*, 2021.

[35] D. Heath, V. Kolesnikov, and L. K. L. Ng, "Garbled circuit lookup tables with logarithmic number of ciphertexts," in *EUROCRYPT Part V*, 2024.

[36] J. Doerner and a. shelat, "Scaling ORAM for secure computation," in *CCS*, 2017.

[37] Y. Zheng, Q. Zhang, S. S. M. Chow, Y. Peng, S. Tan, L. Li, and S. Yin, "Secure softmax/sigmoid for machine-learning computation," in *ACSAC*, 2023.

[38] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, and H. Yalame, "FLUTE: Fast and secure lookup table evaluations," in *S&P*, 2023.

[39] S. Wagh, "Pika: Secure computation using function secret sharing over rings," *Proc. Priv. Enhancing Technol. (PETS)*, no. 4, 2022.

[40] Y. Li, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Efficient 3PC for binary circuits with application to maliciously-secure DNN inference," in *USENIX Security Symposium*, 2023.

[41] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure tensorflow inference," in *S&P*, 2020.

[42] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," arXiv 1904.10509, 2019.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[44] J. P. K. Ma, R. K. H. Tai, Y. Zhao, and S. S. M. Chow, "Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation," in *NDSS*, 2021.

[45] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991.

[46] N. P. Smart and T. Tanguy, "TaaS: Commodity MPC via triples-as-a-service," in *CCSW@CCS*, 2019.

[47] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

[48] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *CCS*, 2016.

[49] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, "Extracting training data from large language models," in *USENIX Security Symposium*, 2021.

[50] M. Du, X. Yue, S. S. M. Chow, and H. Sun, "Sanitizing sentence embeddings (and labels) for local differential privacy," in *The Web*, 2023.

[51] M. Du, X. Yue, S. S. M. Chow, T. Wang, C. Huang, and H. Sun, "DP-Forward: Fine-tuning and inference on language models with differential privacy in forward pass," in *CCS*, 2023.

[52] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptol.*, vol. 13, no. 1, 2000.

[53] E. Kushilevitz, Y. Lindell, and T. Rabin, "Information-theoretically secure protocols and security under composition," in *STOC*, 2006.

[54] O. Goldreich, *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001.

[55] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *ESORICS*, 2008.

[56] L. K. L. Ng, S. S. M. Chow, A. P. Y. Woo, D. P. H. Wong, and Y. Zhao, "Goten: GPU-outsourcing trusted execution of neural network training," in *AAAI*, 2021.

[57] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proc. Priv. Enhancing Technol. (PETS)*, no. 1, 2021.

[58] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: Super-fast and robust privacy-preserving machine learning," in *USENIX Security Symposium*, 2021.

Artifact Appendix

A. Description & Requirements

1) How to access: SHAFT's source code is available on GitHub.[18] The `main` branch contains the latest version, while the `ndss-25-ae` tag corresponds to the exact code submitted for artifact evaluation. This tagged version is also on Zenodo.[19]

2) Hardware dependencies: We conducted experiments on a machine with Intel Xeon Gold 5318Y CPUs at 2.10 GHz, two NVIDIA A40 GPUs, and 256 GB of RAM.

3) Software dependencies: We implemented SHAFT using Python 3.10.12. The dependencies for SHAFT (and the following experiments) are detailed in `requirements.txt`.

[18]https://github.com/andeskyl/SHAFT.

[19]https://doi.org/10.5281/zenodo.14253770

4) Benchmarks: (1) Datasets: QNLI, SST-2, CoLA. (2) Models: BERT-base[20], BERT-large, GPT-2, ViT-base.

B. Artifact Installation & Configuration

Download the SHAFT artifact, *e.g.*, via the "`git clone https://github.com/andeskyl/SHAFT`" command. Our repository includes `README.md` files with step-by-step installation instructions for SHAFT and its dependencies.

C. Major Claims

- (C1): Our secure softmax protocol outperforms the SoTA approaches [19], [37] in both running time and communication rounds. This is proven by the experiment (E1), with results reported in Table V.
- (C2): Our private GELU protocol surpasses the SoTA method [13] in accuracy, running time, and communication. This is demonstrated by the experiments (E2), with results reported in Tables III and VI.
- (C3): For private transformer inference, SHAFT outperforms SIGMA [26] in communication and BumbleBee [27] in running time. This is proven by the experiment (E3), with results reported in Table VII.
- (C4): SHAFT achieves comparable accuracy to plaintext transformer inference. This is shown by the experiment (E4), with results reported in Table VIII.

D. Evaluation

1) Experiment (E1): [Softmax performance] [5 human-minutes + 1 compute-minute]: This experiment benchmarks the performance of our secure softmax protocol.

[Preparation] Go to the `examples/unit-test` folder.

[Execution] Run the command below.

```
$ python run_test_softmax.py
```

[Results] A similar output[21] as follows should be shown:

```
l=32 time: 0.1464s, bytes: 0.0596 MB, rounds: 41
l=64 time: 0.1450s, bytes: 0.1191 MB, rounds: 41
l=128 time: 0.1420s, bytes: 0.2383 MB, rounds: 41
l=256 time: 0.1505s, bytes: 0.4766 MB, rounds: 41
```

2) Experiment (E2): [GELU performance] [1 human-minute + 1 compute-minute]: This experiment examines the performance of our secure GELU protocol.

[Preparation] This can be done right after (E1).

[Execution] Run the command below.

```
$ python run_test_gelu.py
```

[Results] A similar output as follows should be shown:

```
max error: 0.0046, avg error: 0.000739
(128,3072) time: 0.2203s, bytes: 354 MB, rounds: 19
(128,4096) time: 0.2818s, bytes: 472 MB, rounds: 19
```

3) Experiment (E3): [Private transformer inference cost] [5 human-minutes + 20 compute-minutes]: This experiment verifies SHAFT's private inference cost over different models.

[Preparation] To test the private inference cost of {BERT, GPT, ViT} model, go to the {`text-classification`, `text-generation`, `image-classification`} folder under the `examples` directory. Then, run the following command to install the additional dependencies:

```
$ pip install -r requirements.txt
```

[Execution] The running time of private inference is reported using formula: computation time$+2\times$communication$\div$ bandwidth$+$rounds$\times$latency. We provide scripts (in the form of `test_<model>_<in_size>_<comp/comm>.sh`) to evaluate private BERT/GPT/ViT inference costs. Please refer to `README.md` for detailed steps for evaluating each model.

As an example, consider evaluating private BERT-base inference costs on length-128 input. The following two commands evaluate the computation and communication costs.

```
$ bash test_bert_base_128_comp.sh
$ bash test_bert_base_128_comm.sh
```

[Results] An example output of the above scripts:

```
comp time: 6.93s
comm byte: 10.46 GB, round: 1495
```

With the results, we can calculate the running time for different network settings using the above formula, *e.g.*, under the LAN setting with 1 Gbps bandwidth and 0.5 ms latency, the running time is $6.93 + 2 \times 10.46 \div 1 + 1495 \times (0.5 \div 1000) = 28.60$ s.

4) Experiment (E4): [Private transformer inference accuracy] [3 human-minutes + 1 compute-hour]: This experiment validates our accuracy. Given its time-consuming nature, we suggest evaluating only the accuracy of the SST-2 dataset, the smallest one among the three reported in Table VIII.

[Preparation] Go to `text-classification`.

[Execution] Assume that the dependencies have already been installed in experiment (E3). Run the command below.

```
$ bash test_bert_base_acc.sh
```

By default, the above script evaluates the SST-2 dataset. To evaluate QNLI and CoLA datasets, replace the "`sst2`" in the first line of the script with "`qnli`" and "`cola`", respectively.

[Results] A similar output as follows should be shown:

```
running private inference, samples_seen=1
...
running private inference, samples_seen=872
metric: {'accuracy': 0.9220183486238532}
running time: 3804.113387823105s
```

[20]BERT models need fine-tuning on task-specific datasets for inference. To ensure the reproducibility of accuracy results, we have uploaded our fine-tuned BERT-base models for SST-2 (bert-base-cased-sst2), CoLA (bert-base-cased-cola), and QNLI (bert-base-cased-qnli) to Hugging Face. With SHAFT's seamless import from Hugging Face, users need not download them manually.

[21]Transferred bytes should match the expected results, but the running time may vary (*e.g.*, due to system load or inter-process communication latency).