BumbleBee: Secure Two-party Inference Framework for Large Transformers

Wen-jie Lu*‡, Zhicong Huang*, Zhen Gu†, Jingyu Li*‡, Jian Liu‡, Cheng Hong✉*, Kui Ren‡, Tao Wei*, WenGuang Chen*

*Ant Group

†Alibaba Group

‡The State Key Laboratory of Blockchain and Data Security, Zhejiang University

***Abstract*—Large transformer-based models have realized state-of-the-art performance on lots of real-world tasks such as natural language processing and computer vision. However, with the increasing sensitivity of the data and tasks they handle, privacy has become a major concern during model deployment. In this work, we focus on private inference in two-party settings, where one party holds private inputs and the other holds the model. We introduce `BumbleBee`, a fast and communication-friendly two-party private transformer inference system. Our contributions are three-fold: First, we propose optimized protocols for matrix multiplication, which significantly reduce communication costs by 80% – 90% compared to previous techniques. Secondly, we develop a methodology for constructing efficient protocols tailored to the non-linear activation functions employed in transformer models. The proposed activation protocols have realized a significant enhancement in processing speed, alongside a remarkable reduction in communication costs by 80% – 95% compared with two prior methods. Lastly, we have performed extensive benchmarks on five transformer models. `BumbleBee` demonstrates its capability by evaluating the LLaMA-7B model, generating one token in approximately 14 minutes using CPUs. Our results further reveal that `BumbleBee` outperforms `Iron` (NeurIPS22) by over an order of magnitude and is three times faster than `BOLT` (Oakland24) with one-tenth communication.**

I. Introduction

Large transformer-based models such as BERT [19], [50], GPT [60] and ViT [22] have realized state-of-the-art (SOTA) performance on lots of real-world tasks including person re-identification [47], voice assistant [14] and code auto-completion [75]. As the transformer models are handling increasingly sensitive data and tasks, privacy has become one of the major concerns during the model deployment.

Private inference aims to protect model weights from users, while guaranteeing that the server learns no information about users' private inputs. Many recent works have introduced cryptographic frameworks based on Secure Multi-party Computation (MPC) [8], [27] to enable private inference on deep learning models such as convolution neural networks (CNN) [36], [2], [41], [62] and transformer-based models [48], [33], [79], [73]. While Secure Two-party (2PC) can efficiently infer CNNs in a matter of minutes, the private inference on transformer-based models introduce new challenges, particularly in terms of communication overhead. For example, studies such as [33], [30] have indicated that a single private inference on a 12-layer BERT model may require up to 90 GB of communication. Furthermore, [73, Table 1] reports one private inference on a 12-layer ViT might need to exchange about 262 GB messages. Thus, a high-speed bandwidth is indispensable for these communication-intensive approaches.

Cheng Hong is the corresponding author. vince.hc@antgroup.com

Network and Distributed System Security (NDSS) Symposium 2025
23 February - 28 February 2025, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2025.23xxx
www.ndss-symposium.org

We summarize two major challenges in developing a computational fast and communication-friendly 2PC framework for the private evaluation of large transformers.

1) **Multiplication of large-scale matrices.** The inference of a transformer-based model can involve hundreds of multiplications of large matrices. For instance, Natural Language Processing (NLP) transformers use embedding tables to convert a query of words into a numerical representation. An embedding table lookup can be defined as a matrix multiplication $\mathbf{EV}$ where each row of $\mathbf{E}$ is a one-hot vector corresponding to the index of each word in the input query. In other words, the dimensions of this multiplication can be parsed as *numbers of words* $\times$ *vocabulary size* $\times$ *embedding size* which is significantly larger than the matrix in CNNs. Vision transformers can also involve large size matrices due to a relatively larger embedding size than the NLP transformers.
   Most of the existing cryptographic protocols for private matrix multiplication rely on either Oblivious Transfer (OT) [18], [57], [59] or Homomorphic Encryption (HE) [41], [12], [36], [35]. However, these two line of approaches have their limitations. The OT-based methods for private matrix multiplication require less computation time but necessitate the transmission of an enormous amount of messages. On the other hand, the HE-based methods demand significantly more computation but are more communication-friendly than the OT approaches.
   The first challenge is to develop a matrix multiplication protocol that is both speedy and communication-friendly.
2) **More complicated activation functions**. In contrast to the straightforward ReLU activation employed in CNNs,

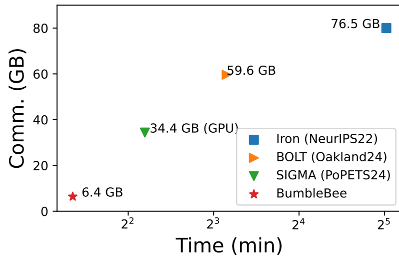

Figure 1: The overall bandwidth improvements of the proposed optimizations on the BERT-base model with 128 input tokens (1Gbps, CPU). Please note that direct comparisons of timing results may not be equitable due to differences in execution environments.

the transformer consists of complex activation functions like softmax, Gaussian Error Linear Unit (GeLU), and Sigmoid Linear Unit (SiLU). The evaluation of these activation functions requires fundamental functions such as the exponentiation, division and hyperbolic. While researchers have developed specific protocols for these fundamental functions [11], [61], [62], [44], however, it is still not practical to use them directly in transformer models. The primary reason is that the number of activations in the transformer models is exceedingly large. For example, a single inference on the GPT2 model [15] requires evaluating about 3.9×10^6 point-wise GeLUs. Our second challenge is to design efficient 2PC protocols for these complex activation functions.

A. Technical Details

Efficient protocols for linear functions. We propose a primitive called Oblivious Linear Transformation (OLT). We describe the OLT as a 2-party protocol that takes two private matrices $\mathbf{Q} \in \mathbb{Z}_{2^\ell}^*$ and $\mathbf{V} \in \mathbb{Z}_{2^\ell}^*$ from the two parties, respectively, and generates the share $[\![\mathbf{QV}]\!]$ between them. Using OLT, we can achieve the multiplication of two additively shared matrices over the ring modulus 2^ℓ. The HE-based approaches [36], [54] provide a good starting point. A significant limitation of these works is the considerable communication overhead, which arises from the "sparse" format of the output ciphertexts. More specifically, *each* of the output ciphertexts in [36], [54] encrypts a long vector. However, only a small subset of the vector entries are needed for the multiplication result. One still needs to transfer the whole encrypted vector for decryption. To overcome this shortage, we present a compression procedure to homomorphically zeroize the unnecessary entries of the encrypted vector so that we can combine many of the "sparse" vectors into a "dense" one. The communication is thus reduced because of a smaller number of ciphertexts to send. Compared to a previous ciphertext compression method [13], our compression procedure is about $50\times$ faster. Overall, we observed 80% – 90% less communication costs over [36], [54].

Besides the matrix multiplications, the point-wise multiplication $x_0 \cdot y_0, x_1 \cdot y_1, \cdots$ is also an important computation in the transformer inference[1]. Many HE-based point-wise multiplication protocols [18], [63] have to set a relatively large plaintext modulus t. For instance, both [18], [63] set the HE parameter $t > 2^{2\ell+40}$ to encrypt a value from $\mathbb{Z}_{2^\ell}$. In this work, we present a modulus lifting function to unify the secret sharing modulus and the plaintext modulus of the underlying HE. The lifting function enables us to perform modulo 2^ℓ arithmetic operations over the HE ciphertexts. This allows us to choose a smaller HE parameter, i.e., $t \approx 2^{2\ell}$. Our empirical results demonstrate $1.3\times$ improvements when applying the lifting function to [63].

A framework for constructing efficient and accurate protocols for the activation functions in transformers. We first propose a general framework for constructing efficient and precise 2PC protocols for the activation functions used by many transformer models. These activation functions share a common property: they are relatively smooth within a short interval around the origin and nearly linear on two sides. With this characteristic in mind, we propose using one or two low-degree polynomials to approximate the activation function within the short interval, and using the identity function on the two sides. For example, we suggest to approximate SiLU with two polynomials $P(x)$ and $Q(x)$ of a low degree that minimize the least square error, as follows.

$$\texttt{SiLU}(x) \approx \begin{cases} -10^{-5} & x < -8 \\ P(x) & -8 < x \le -4 \\ Q(x) & -4 < x \le 4 \\ x - 10^{-5} & x > 4 \end{cases}$$

Then we propose optimizations based on two key insights. Firstly, we exploit the smoothness of the activation function to enhance efficiency. For example, the evaluation of an input $x = -3.98$ on the (wrong) 2nd segment can also give a similar result. That is $P(-3.98) \approx Q(-3.98)$. This smoothness gives us some room to design an efficient evaluation method at the cost of introducing a mild error. Secondly, we introduce optimizations to improve the amortized efficiency when evaluating multiple polynomials over the same input point. Concretely, our optimized activation protocol is 9 – 20 times faster and requiring 80% – 95% less communication compared to the current numerical methods detailed in [61], [53].

B. Contributions

To summarize, we make three key contributions:

1) We have developed a matrix multiplication protocol from HE, specifically tailored for modulo 2^ℓ operations, optimized to enhance communication efficiency. In fact, our protocol has been shown to require 80% less communication compared to existing methods based on HE. Also, our protocol is speedy. For instance, the private matrix

[1]The point-wise multiplications are usually achieved via Batch Oblivious Linear Evaluation [17].

multiplication in the dimensions $128 \times 768 \times 768$ can be done within 1.0 second using a moderate cloud instance.

2) We have implemented all the proposed protocols and developed a new MPC back-end called `BumbleBee` into the `SPU` library [53]. To compare, we also implemented a baseline using several SOTA 2PC protocols based on the `SPU` library. Upon quick examination, Figure 1 illustrates the improvements made by the proposed protocols compared to the baseline. In summary, we have a reduction of communication costs by 84% over `SIGMA` [30], by 90% over `BOLT` [58], and by 92% over `Iron` [33].
3) `BumbleBee` enables easy-to-use private transformer inference. The existing works [48], [33], [58] have only considered the BERT-family models. To compare, we have successfully run `BumbleBee` on 5 pre-trained transformer models utilizing the model weights and the Python programs available on HuggingFace's website, including BERT-base, BERT-large, GPT2-base, LLaMA-7B, and ViT-base. We also evaluated the accuracy of `BumbleBee` across four public datasets. All our experiments were conducted using the proposed protocols, rather than through simulation. We provide a reproducible implementation in https://github.com/AntCPLab/OpenBumbleBee. Our approach provides a promising evidence that accurate and feasible private transformer inference is possible, even without changing the neural network structure.

II. Preliminaries

A. Notations

We write $x = y$ to mean x is equal to y and write $x := y$ to assign the value of y to the variable x. For an interactive protocol Π, we write $[\![x]\!] \leftarrow \Pi$ to denote the execution of the protocol. We denote by $[n]$ the set $\{0, \cdots, n-1\}$ for $n \in \mathbb{N}$. For a set $\mathcal{D}$, $x \in_R \mathcal{D}$ means x is sampled from $\mathcal{D}$ uniformly at random. We use $\lceil \cdot \rceil$, $\lfloor \cdot \rfloor$ and $\lfloor \cdot \rceil$ to denote the ceiling, flooring, and rounding function, respectively. The logical AND and XOR is $\wedge$ and $\oplus$, respectively. Let $\mathbf{1}\{\mathcal{P}\}$ denote the indicator function that is 1 when the predicate $\mathcal{P}$ is true and 0 when $\mathcal{P}$ is false. We use lower-case letters with a "hat" symbol such as $\hat{a}$ to represent a polynomial, and $\hat{a}[j]$ to denote the j-th coefficient of $\hat{a}$. We use the dot symbol $\cdot$ such as $\hat{a} \cdot \hat{b}$ to represent the multiplication of polynomials. We denote $\mathbb{Z}_q = \mathbb{Z} \cap [0, q)$ for $q \geq 2$. The congruence $x \equiv y \bmod 2^\ell$ will be abbreviated as $x \equiv_\ell y$. For a 2-power number N, and $q > 0$, we write R_q to denote the set of integer polynomials $R_q = \mathbb{Z}_q[X]/(X^N + 1)$. We use bold letters such as $\mathbf{a}, \mathbf{M}$ to represent vectors and matrix, and use $\mathbf{a}[j]$ to denote the j-th component of $\mathbf{a}$ and use $\mathbf{M}[j, i]$ to denote the (j, i) entry of $\mathbf{M}$. The Hadamard product is written as $\mathbf{a} \odot \mathbf{b}$.

B. Cryptographic Primitives

1) Additive Secret Sharing: We use 2-out-of-2 additive secret sharing schemes over the ring $\mathbb{Z}_{2^\ell}$ throughout this manuscript. A ℓ-bit ($\ell \geq 2$) value x is additively shared as $[\![x]\!]_0$ and $[\![x]\!]_1$ where $[\![x]\!]_l$ is a random share of x held by P_l. We can write the multiplication of two shared value as $[\![x]\!] \cdot [\![y]\!] \equiv_\ell ([\![x]\!]_0 + [\![x]\!]_1) \cdot ([\![y]\!]_0 + [\![y]\!]_1) \equiv_\ell [\![x]\!]_0[\![y]\!]_0 + [\![x]\!]_1[\![y]\!]_1 + [\![x]\!]_0[\![y]\!]_1 + [\![x]\!]_1[\![y]\!]_0$ where the mixed-terms $[\![x]\!]_0[\![y]\!]_1$ and $[\![x]\!]_1[\![y]\!]_0$ are computed using homomorphic encryption. For a real value $\tilde{x} \in \mathbb{R}$, we first encode it as a fixed-point value $x = \lfloor \tilde{x} 2^f \rceil \in [-2^{\ell-1}, 2^{\ell-1})$ under a specified precision $f > 0$ before secretly sharing it. We write $[\![\cdot; f]\!]$ to explicitly denote the shared value of f-bit fixed-point precision. When $\ell = 1$, we use $[\![z]\!]^B$ to denote Boolean shares. Also we omit the subscript and only write $[\![x]\!]$ or $[\![z]\!]^B$ when the ownership is irrelevant from the context.

2) Oblivious Transfer: We rely on oblivious transfer (OT) for the non-linear computation. In a general 1-out-of-2 OT $\binom{2}{1}$-$\mathtt{OT}_\ell$, a sender inputs two messages m_0 and m_1 of length ℓ bits and a receiver inputs a choice bit $c \in \{0, 1\}$. At the end of the protocol, the receiver learns m_c, whereas the sender learns nothing. When sender messages are correlated, the Correlated OT (COT) is more efficient in communication [6]. In our additive COT, a sender inputs a function $f(x) = x + \Delta$ for some $\Delta \in \mathbb{Z}_{2^\ell}$, and a receiver inputs a choice bit c. At the end of the protocol, the sender learns $x \in \mathbb{Z}_{2^\ell}$ whereas the receiver learns $x + c \cdot \Delta \in \mathbb{Z}_{2^\ell}$. In this work, we use the `Ferret` protocol [77] for a lower communication COT.

3) Lattice-based Additive Homomorphic Encryption: A homomorphic encryption (HE) of x enables computing the encryption of $F(x)$ without the knowledge of the decryption key. In this work, we use an HE scheme that is based on Ring Learning-with-Error (RLWE) [52]. The RLWE scheme is defined by a set of public parameters $\mathsf{HE.pp} = \{N, q, t\}$.

- **KeyGen.** Generate the RLWE key pair $(\mathsf{sk}, \mathsf{pk})$ where the secret key $\mathsf{sk} \in R_q$ and the public key $\mathsf{pk} \in R_q^2$.
- **Encryption.** An RLWE ciphertext is given as a polynomial tuple $(\hat{b}, \hat{a}) \in R_q^2$. We write $\mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\hat{m})$ to denote the encryption of $\hat{m} \in R_t$ under a key pk.
- **Addition** ($\boxplus$)**.** Given two RLWE ciphertexts $\mathsf{ct}_0 = (\hat{b}_0, \hat{a}_0)$ and $\mathsf{ct}_1 = (\hat{b}_1, \hat{a}_1)$ that respectively encrypts $\hat{m}_0, \hat{m}_1 \in R_t$ under a same key, the operation $\mathsf{ct}_0 \boxplus \mathsf{ct}_1$ computes the RLWE tuple $(\hat{b}_0 + \hat{b}_1, \hat{a}_0 + \hat{a}_1) \in R_q^2$ which can be decrypted to $\hat{m}_0 + \hat{m}_1 \bmod R_t$.
- **Multiplication** ($\boxtimes$)**.** Given an RLWE ciphertext $\mathsf{ct} = (\hat{b}, \hat{a})$ that encrypts $\hat{m} \in R_t$, and a plain polynomial $\hat{c} \in R_t$, the operation $\mathsf{ct} \boxtimes \hat{c}$ computes the tuple $(\hat{b} \cdot \hat{c}, \hat{a} \cdot \hat{c}) \in R_q^2$ which can be decrypted to $\hat{m} \cdot \hat{c} \bmod R_t$.
- **SIMD Encoding.** By choosing a prime t such that $t \equiv 1 \bmod 2N$, the SIMD technique [66] allows to convert vectors $\mathbf{v}, \mathbf{u} \in \mathbb{Z}_t^N$ of N elements to polynomials $\hat{v}, \hat{u} \in R_t$. The product polynomial $\hat{v} \cdot \hat{u}$ can be decoded to the point-wise multiplication $\mathbf{u} \odot \mathbf{v} \bmod t$. In the context of private evaluation, the SIMD technique can amortize the cost of point-wise multiplication by a factor of $1/N$. We denote $\hat{v} := \mathsf{SIMD}(\mathbf{v})$ as the SIMD encoding, and write $\mathsf{SIMD}^{-1}(\cdot)$ as the decoding function.
- **Automorphism.** Given a RLWE ciphertext $\mathsf{ct} \in R_q^2$ that encrypts a polynomial $\hat{m}(X) \in R_t$, and an odd integer $g \in [1, 2N)$, the operation $\mathsf{ct}' := \mathsf{Auto}(\mathsf{ct}, g)$ computes $\mathsf{ct}' \in R_q^2$ which decrypts to $\hat{m}'(X) = \hat{m}(X^g) \in R_t$.

TABLE I: The proposed protocols in `BumbleBee`. Also we use some existing protocols as the building blocks.

Proposed Protocols	Descriptions
$[\![\mathbf{z}]\!] \leftarrow \Pi_{\mathrm{bOLEe}}(\mathbf{x}, \mathbf{y})$ such that $\mathbf{z} \equiv_\ell \mathbf{x} \odot \mathbf{y} + \mathbf{e}$ for $\|\mathbf{e}\|_\infty \le 1$	Batch OLE with Error (§IV)
$[\![\mathbf{x} \odot \mathbf{y}]\!] \leftarrow \Pi_{\mathrm{mul}}([\![\mathbf{x}]\!], [\![\mathbf{y}]\!])$	Point-wise mult.
$[\![\mathbf{x}^2]\!] \leftarrow \Pi_{\mathrm{square}}([\![\mathbf{x}]\!])$	Square
$[\![\mathbf{Z}]\!] \leftarrow \Pi_{\mathrm{OLT}}(\mathbf{X}, \mathbf{Y})$ such that $\mathbf{Z} \equiv_\ell \mathbf{X} \cdot \mathbf{Y}$	Oblivious Linear Transformation (§III)
$[\![\mathbf{X} \cdot \mathbf{Y}]\!] \leftarrow \Pi_{\mathrm{matmul}}([\![\mathbf{X}]\!], [\![\mathbf{Y}]\!])$	Shared matrix mult.
$[\![\tilde{y}; f]\!] \leftarrow \Pi_{\mathrm{GeLU}}([\![\tilde{x}; f]\!])$	GeLU (§V-A)
$[\![\tilde{\mathbf{y}}; f]\!] \leftarrow \Pi_{\mathrm{softmax}}([\![\tilde{\mathbf{x}}; f]\!])$	Softmax (§V-B)

Ideal Functionalities	Descriptions
$[\![\tilde{x}; f]\!] \leftarrow \mathcal{F}^f_{\mathrm{trunc}}([\![\tilde{x}; 2f]\!])$	Truncation [36], [16]
$[\![c ? x : y]\!] \leftarrow \mathcal{F}_{\mathrm{mux}}([\![c]\!]^B, [\![x]\!], [\![y]\!])$	Multiplexer
$[\![1/\sqrt{\tilde{x}}; f]\!] \leftarrow \mathcal{F}_{\mathrm{rsqrt}}([\![\tilde{x}; f]\!])$	Reciprocal Sqrt [44]
$[\![\mathbf{1}\{x < y\}]\!]^B \leftarrow \mathcal{F}_{\mathrm{less}}([\![x]\!], [\![y]\!])$	Less-then [62]
$[\![\mathbf{x}]\!] \leftarrow \mathcal{F}_{\mathrm{H2A}}(\mathsf{RLWE}(\mathbf{x}))$	HE to arithmetic share [36], [63]

In this work, we use the automorphism to compress the volume of RLWE ciphertexts.

Remark 1 (Circuit Privacy)**.** *RLWE ciphertexts have a noise associated with them which would leak information about the homomorphic computation to the decryption key holder. We need a mechanism to hide the noise when sending the computed RLWE ciphertexts for decryption. We abstract this mechanism as a $\mathcal{F}_{\mathrm{H2A}}$ functionality (see Table I) which takes as input an RLWE ciphertext of $\hat{m}$ and outputs the arithmetic share $[\![\hat{m}]\!]$ to each party without revealing the noise to the decryptor. Concretely, we adopt the noise flooding [26] and the approximated resharing [36] to implement this function. The concrete H2A protocols are given in Appendix.*

C. Transformer-based Models

Many modern language pre-processors such as GPT [15] and BERT [19] consist of an input embedding layer followed by multiple layers of Transformers [70]. Similarly, Vision Transformers (ViTs) [22], [5] also employ a similar architecture, except that they do not incorporate an input embedding layer. There are two major computation blocks for Transformer-based models: a multi-head attention mechanism and a feed-forward network. Besides, layer normalization are employed around each of the two consequent layers.

Multi-head Attention.. An attention mechanism $\mathsf{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ computing $\mathtt{softmax}(\mathbf{Q}\mathbf{K}^\top + \mathbf{M})\mathbf{V}$, can be described as mapping a query $\mathbf{Q}$ and a set of key-value pairs $(\mathbf{K}, \mathbf{V})$ to a weighted sum. Note $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are different linear projections of an input matrix. The multi-head attention variant computes a H-parallel attention $\mathsf{Attention}(\mathbf{Q}_j, \mathbf{K}_j, \mathbf{V}_j)$ for $j \in [H]$ and then concatenate these H resultant matrices.

Layer Normalization. For a vector $\mathbf{x} \in \mathbb{R}^d$, let $\mu = (1/d) \cdot \sum_j \mathbf{x}[j] \in \mathbb{R}$ and $\sigma = \sum_{j \in [d]} (\mathbf{x}[j] - \mu)^2 \in \mathbb{R}$. The layer normalization is denoted by $\mathtt{LayerNorm}(\mathbf{x}) = \gamma \cdot (\mathbf{x} - \mu) \cdot \sigma^{-1/2} + \beta$, where $\gamma, \beta \in \mathbb{R}$ are two hyper-parameters.

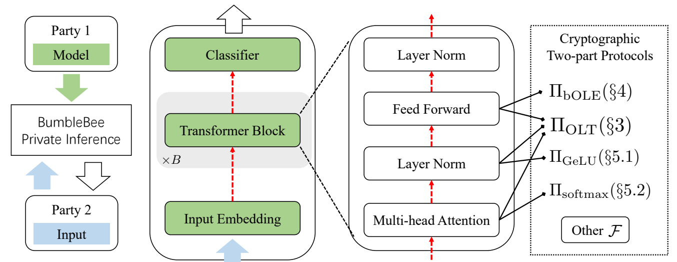

Figure 2: Overview of `BumbleBee`'s private transformer inference. The dash arrows indicate secretly shared messages.

Feed-forward. A feed-forward network typically includes two linear projections, with an activation function applied between them, i.e., $\mathtt{FFN}(\mathbf{X}) = \mathbf{W}_0 \cdot F(\mathbf{W}_1 \cdot \mathbf{X})$, where $F : \mathbb{R}^* \mapsto \mathbb{R}^*$ is an activation function such as GeLU or SiLU.

D. Threat Model and Private Inference

Similar to previous work [41], [55], [62], [36], [3], we target privacy against a static and semi-honest probabilistic polynomial time (PPT) adversary following the ideal/real world paradigm [10]. That is, we consider a computationally bounded adversary that corrupts one of the parties at the beginning of the protocol execution, follows the protocol specification, but tries to learn additional information about the honest party's input.

`BumbleBee` invokes several sub-protocols of smaller private computations that are summarized in Table I. To simplify the protocol description and security proofs, we describe `BumbleBee` using the hybrid model. A protocol invoking a functionality $\mathcal{F}$ is said to be in "$\mathcal{F}$-hybrid model". Also, some functions are computed approximately in `BumbleBee` for the sake of a better efficiency. According to the definition [25], a protocol constitutes a private approximation of F if the approximation reveals no more about the inputs than F itself does. In the context of private 2PC inference (Figure 2), a server S holds a transformer model while a client C queries to the model such as a piece of texts. Assuming semi-honest S and C, `BumbleBee` enables C to learn only two pieces of information: the architecture of the transformer, and the inference result. S is either allowed to learn the result or nothing, depending on the application scenario. All other information about C's private inputs and the model weights of S should be kept secret. Formal definitions of the threat model are provided in the Appendix.

III. Secure Matrix Multiplication

We need two types of matrix multiplication for the private transformer inference:

1) The multiplication of a shared matrix and a plaintext matrix. For instance, for each of the Transformer block (cf. Figure 2), we compute the multiplication between a shared input matrix (that is computed from the previous block) and a weight matrix of the server in plaintext.
2) The multiplication of two secretly shared matrices inside the attention mechanism.

Toy example over $\mathbb{Z}_{2^5}$.

$$\mathbf{Q} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}, \mathbf{V} = \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix} \Rightarrow \mathbf{QV} \equiv \begin{bmatrix} 31 & 2 \\ 7 & 14 \\ 15 & 26 \\ 23 & 6 \end{bmatrix} \bmod 2^5$$

Compute $\mathbf{QV}$ with $\hat{q} := \pi_{\text{lhs}}(\mathbf{Q})$ and $\hat{v} := \pi_{\text{rhs}}(\mathbf{V})$.

$\hat{q} = 1X^0 - 2X^{15} + 3X^4 + 4X^3 + 5X^8 + 6X^7 + 7X^{12} + 8X^{11}$

$\hat{v} = 9X^0 + 11X^1 + 10X^2 + 12X^3$

$\hat{q} \cdot \hat{v} \bmod (X^{16}+1, 2^5) \equiv \mathbf{31}X^0 + 31X^1 + \mathbf{2}X^2 + 16X^3 + \mathbf{7}X^4 + 9X^5 + \mathbf{14}X^6 + 26X^7 + \mathbf{15}X^8 + 19X^9 + \mathbf{26}X^{10} + 4X^{11} + \mathbf{23}X^{12} + 29X^{13} + \mathbf{6}X^{14} + 2X^{15} \bmod (X^{16}+1, 2^5)$

Figure 3: Example of how to compute matrix multiplication using KRDY style with $N = 16$ and $\ell = 5$. Here 8 out of the 16 result coefficients are "useful".

We propose a primitive called Oblivious Linear Transformation (OLT) to achieve these two types of matrix multiplication. We describe the OLT as a 2-party protocol that takes two private matrices $\mathbf{Q}$ and $\mathbf{V}$ from the two parties, respectively, and generates the share $[\![\mathbf{QV}]\!]$ between them. With the OLT primitive, we can compute the multiplication between a shared matrix $[\![\mathbf{Q}]\!]$ and a plaintext matrix $\mathbf{V}$ using one OLT execution. On the other hand, we need two OLTs for multiplying two shared matrices. That is we compute the two mixed terms $[\![\mathbf{Q}]\!]_1 \cdot [\![\mathbf{V}]\!]_0$ and $[\![\mathbf{Q}]\!]_0 \cdot [\![\mathbf{V}]\!]_1$ using two OLTs.

A. Existing HE-based Approaches for OLT

Our starting point is the HE-based OLT used by many works such as [54], [36], [33] (hereafter referred to as KRDY style). KRDY needs two functions $\pi_{\text{lhs}} : \mathbb{Z}_{2^\ell}^{k_w \times m_w} \mapsto R_{2^\ell}$ and $\pi_{\text{rhs}} : \mathbb{Z}_{2^\ell}^{m_w \times n_w} \mapsto R_{2^\ell}$ to encode a matrix into coefficients of a polynomial that can be encrypted using RLWE. Assuming $1 \le k_w \cdot m_w \cdot n_w \le N$, these two functions are defined as follows.

$$\hat{q} := \pi_{\text{lhs}}(\mathbf{Q}) \text{ and } \hat{v} := \pi_{\text{rhs}}(\mathbf{V}) \text{ such that} \tag{1}$$
$$\hat{q}[0] = \mathbf{Q}[i,j] \text{ for } i = 0 \wedge j = 0,$$
$$\hat{q}[N + i \cdot k_w \cdot m_w - j] = -\mathbf{Q}[i,j], \text{ for } i = 0 \wedge j \in [m_w]/\{0\}$$
$$\hat{q}[i \cdot k_w \cdot m_w - j] = \mathbf{Q}[i,j] \text{ for } i > 0 \wedge j \in [m_w]$$
$$\hat{v}[k \cdot m_w + j] = \mathbf{V}[j,k] \text{ for } j \in [m_w], k \in [n_w]$$

All other coefficients of $\hat{q}$ and $\hat{v}$ are set to 0. The product polynomial $\hat{q} \cdot \hat{v}$ directly gives the resultant matrix $\mathbf{QV}$ *in some of its coefficients* as shown in the following proposition.

Proposition 1. *[54, adapted] Assuming $1 \le k_w \cdot m_w \cdot n_w \le N$. Given two polynomials $\hat{q} = \pi_{\text{lhs}}(\mathbf{Q}), \hat{v} = \pi_{\text{rhs}}(\mathbf{V}) \in R_{2^\ell}$, the multiplication $\mathbf{U} \equiv_\ell \mathbf{Q} \cdot \mathbf{V}$ can be evaluated via the product $\hat{u} = \hat{q} \cdot \hat{v}$ over the ring R_{2^ℓ}. That is $\mathbf{U}[i,k] = \hat{u}[i \cdot m_w \cdot n_w + k \cdot m_w]$ for all $i \in [k_w], k \in [n_w]$.*

Figure 3 provides a simple illustration of the ideas behind π_{lhs} and π_{rhs} encodings. In the context of OLT, we let P_0 to send a ciphertext $\mathsf{RLWE}_{\mathsf{pk}_0}(\pi_{\text{lhs}}(\mathbf{Q}))$ to P_1 using P_0's key. Then P_1 can homomorphically evaluate the matrix multiplication $\mathbf{Q} \cdot \mathbf{V}$ using a single homomorphic multiplication i.e., $\mathsf{RLWE}_{\mathsf{pk}_0}(\pi_{\text{lhs}}(\mathbf{Q})) \boxtimes \pi_{\text{rhs}}(\mathbf{V})$. To convert the encrypted matrix to arithmetic share, P_0 and P_1 then jointly invoke $\mathcal{F}_{\text{H2A}}$ functionality (cf. Table I). When matrix shape $k \cdot m \cdot n > N$, we can split them into smaller sub-matrices and apply KRDY to each pair of the corresponding sub-matrices. We denote the partition windows as (k_w, m_w, n_w). In terms of communication cost, KRDY needs to exchange $O(\min(\frac{km}{k_w m_w}, \frac{mn}{m_w n_w}) + \frac{kn}{k_w n_w})$ ciphertexts.

Input: $\{\hat{a}_j \in R_q\}_{j \in [2^r]}$ for an odd q and $1 \le 2^r \le N$.
Output: $\hat{c} \in R_q$ such that $\hat{c}[i] = \hat{a}_{i \bmod 2^r}[\lfloor i/2^r \rfloor \cdot 2^r]$.

1: **for** $\forall j \in [2^r]$ **do in parallel**
2: Compute $\hat{b}_{j,0} := 2^{-r} \cdot \hat{a}_j \bmod q$.
3: **for** $k = 0, 1, \cdots, r-1$ **do in sequence**
4: $\hat{b}_{j,k+1} := \hat{b}_{j,k} + \mathsf{Auto}(\hat{b}_{j,k}, \frac{N}{2^k} + 1)$.
5: **end for** $\triangleright$ $\hat{b}_{j,r}[i] = 0$ for $\forall i$ $i \equiv 0 \bmod 2^r$
6: $\hat{b}_j := \hat{b}_{j,r} \cdot X^j \in R_q$ $\triangleright$ right-shift by j unit
7: **end for**
8: **return** the sum $\sum_{j=0}^{2^r-1} \hat{b}_j$ as $\hat{c}$.

Figure 4: InterLeave: Coefficients interleaving (Naive).

Compared to other HE-based OLT used in [55], [62], KRDY OLTs do not require any rotations (which are very expensive in HE), thus are quite efficient. However, there are many "useless" coefficients in the result of KRDY OLTs. According to Proposition 1, only $k_w n_w$ coefficients of the result polynomial $\hat{u}$ are "useful" out of the N coefficients, but KRDY still needs to transmit the entire RLWE ciphertext of $\hat{u}$ for decryption, which could be a waste of communication. To provide an example, we consider a set of dimensions commonly used in transformer models such as $k = 16, m = 768$, and $n = 3072$. In this scenario, the KRDY protocol might require exchanging about 25 MB of ciphertexts. This can be a substantial communication overhead since the transformer models often contain hundreds of such large-scale matrices. The following section will elaborate on methods for reducing this communication burden.

B. First attempt via Ciphertext Packing

Our first try is to apply a PackLWEs [13] procedure to KRDY, which we refer as KRDY$^+$ hereafter. The PackLWEs procedure allows us to choose arbitrary coefficients from multiple RLWE ciphertexts and combine them into a single RLWE ciphertext by performing homomorphic automorphisms. In the case of KRDY$^+$, we can reduce the number of ciphertexts from $O(kn/(k_w n_w))$ to $O(kn/N)$ ciphertexts at the costs of $O(kn)$ homomorphic automorphism operations. Consider the matrix shape $(k, m, n) = (16, 768, 3072)$ again. KRDY$^+$ now exchanges about 2.9 MB of ciphertexts which is a significant reduction from the 25 MB of KRDY. However, the automorphism operation in HE is quite slow, in fact for matrices with a large kn, the automorphisms used by KRDY$^+$ could become more computationally expensive than the homomorphic multiplication itself.

C. Ciphertext Interleaving Optimization

We present a specifialized optimization for the PackLWEs procedure [13] in the context of matrix multiplication. The main idea is that, instead of "picking"' useful coefficients one by one follows the PackLWEs procedure, we prefer to "clean up" the useless[2] coefficients in the result ciphertexts, and then combine them into a single ciphertext.

We briefly describe the clean up procedure. Let us use $N = 8$ and $\hat{a}(X) = \sum_{i=0}^{i=7} a_i X^i$ as an example. By definition, the automorphism $\mathsf{Auto}(\hat{a}, 9)$ yields $\sum_{i=0}^{i=7} a_i X^{i \cdot 9}$, which is equivalent to

$$\sum_{i=0}^{i=7} a_i X^{i \cdot 9} \equiv \sum_{i=0}^{i=3} a_{2i} X^{2i} - \sum_{i=0}^{i=3} a_{2i+1} X^{2i+1} \bmod X^8 + 1.$$

Here, the sign of coefficients in odd positions is flipped. Hence, evaluating $\hat{a} + \mathsf{Auto}(\hat{a}, N + 1)$ will cancel out odd-indexed coefficients and double the even-indexed coefficients. More generally, let compute $\hat{a} + \mathsf{Auto}(\hat{a}, N/2^j + 1)$ for any 2-power factor of N. Take $N = 8$ as the example again, and we consider $2^j = 2$. $\mathsf{Auto}(\hat{a}, 8/2 + 1)$ equals to

$$\sum_{i=0}^{i=7} a_i X^{i \cdot 5} \equiv a_0 - a_2 X^2 + a_4 X^4 - a_6 X^6 + \sum_{i \neq 0 \bmod 2} a_i X^{i \cdot 5}.$$

In brief, coefficients in odd multiples of 2^j are sign-flipped. Suppose the coefficients of $\hat{a}[i]$ for all $i \neq 0 \bmod 2^j$ are zeros already. Then $\hat{a} + \mathsf{Auto}(\hat{a}, N/2^j + 1)$ will cancel out the odd multiples of 2^j while doubling the positions of even multiples of 2^j. We define a function $\mathtt{ZeroGap}(\hat{a}, 2^r)$ to repeat the formula r times $\hat{a} := \hat{a} + \mathsf{Auto}(\hat{a}, N/2^j + 1)$ for $j = 0, 1, \cdots, r - 1$. After performing these operations, the result is a polynomial where only coefficients at positions that are multiples of 2^r are non-zero. However, these coefficients are scaled by a factor of 2^r. To correct for this scaling, we initially multiply the polynomial by the inverse scaling factor $2^{-r} \bmod q$. This implicitly requires an odd modulus q, which is commonly satisfied in RLWE-based HE that uses modulus $q \equiv 1 \bmod 2N$. Also the ZeroGap procedure requires the gap between two needed coefficients being a power-of-two number. This equals to the partition window m_{w} in the π_{lhs} and π_{rhs} encodings. Note that this partition window can be chosen freely as long as $1 \leq k_{\mathrm{w}} m_{\mathrm{w}} n_{\mathrm{w}} \leq N$ is satisfied.

With the ZeroGap procedure in mind, we present one of our key contributions, i.e., the InterLeave procedure. The InterLeave procedure interleaves 2^r polynomials into a single polynomial with the input coefficients arranged in a stride of 2^r steps. We first give the easy-to-follow version of InterLeave in Figure 4. In this naive version, we separately apply the ZeroGap procedure on each input polynomials (Step 2 to Step 5) to clean up the irrelevant coefficients. Then we obtain the final result via a simple rotate-then-sum computation (Step 6 to Step 8). We would like to emphasize that the rotation over the coefficients can be achieved much more efficiently than the SIMD rotation [32] in the ciphertext domain. Indeed, Figure 4 computes $O(r \cdot 2^r)$ automorphisms to combine N coefficients into a single polynomial. Figure 5 gives a toy example of InterLeave (Naive).

[2] By "useless", we mean the coefficients are not belong to the matrix multiplication.

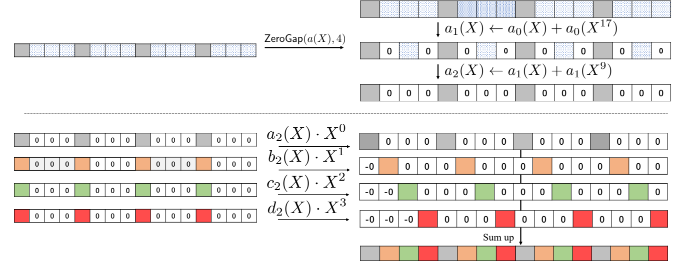

Figure 5: Toy example of InterLeave (Naive) for $N = 16$ and $r = 2$. Upper: 2 automorphisms clean up the irrelevant coefficients, i.e., dashed cells, in one polynomial. Lower: 4 "clean" polynomials merge into one via negacyclic right shift.

Final Version. We now present the optimized version of InterLeave in Figure 6 which improves the complexity of the Figure 4 from $O(r \cdot 2^r)$ automorphisms to $O(2^r)$. We show the intuition behind this optimized version. It is intuitively sensible to exploit the bilinear nature of the automorphism function to combine two automorphisms with identical indices into one automorphism operation. That is the sum of two automorphisms $\mathsf{Auto}(\hat{a}, g) + \mathsf{Auto}(\hat{b}, g)$ equals to the automorphism on the sum polynomial for any valid index g. However, polynomials that apply the same automorphism index are right shifted by distinct units (i.e., Step 6 of Figure 4) before the sum step (i.e., Step 8 of Figure 4). To address this issue, we take advantage the following equation. That is $\mathsf{Auto}(\hat{a}, \frac{N}{2^j} + 1) + \mathsf{Auto}(\hat{b}, \frac{N}{2^j} + 1) \cdot X^{2^j}$ equals to $\mathsf{Auto}(\hat{a} - \hat{b} \cdot X^{2^j}, \frac{N}{2^j} + 1)$ when $\hat{b}[i] = 0$ for all positions i that $i \neq 0 \bmod 2^j$. Recall that the operation $\mathsf{Auto}(\cdot, \frac{N}{2^j} + 1)$ sign-flips the coefficients located at odd multiples of 2^j, while leaving those at even multiples of 2^j unchanged. Also the right-shift to $\hat{b}$ by multiplying X^{2^j} simply changes the parity of $\hat{b}$'s coefficient positions — shifting from an odd multiple to an even multiple of 2^j and vice versa. Thus the pattern of automorphism-then-shift $\mathsf{Auto}(\hat{b}, \frac{N}{2^j} + 1) \cdot X^{2^j}$ is equivalent to the negate-shift-then-automorphism manner $\mathsf{Auto}(-\hat{b} \cdot X^{2^j}, \frac{N}{2^j} + 1)$. Then we can combine the two automorphisms with the same index into one automorphism using the bilinear property.

D. Proposed OLT Protocol

We now describe our OLT protocol in Algorithm 1. The first three steps are similar to those KRDY style protocols [54], [36]. Specifically, we divide a large matrix into sub-blocks and encode each block as a polynomial using π_{lhs} and π_{rhs}. However, we intentionally select a power-of-two partition window m_{w} in our approach. The main difference between our protocol and theirs is the use of the InterLeave procedure in Step 4 to decrease the number of RLWE ciphertexts from $O(kn/(k_{\mathrm{w}} n_{\mathrm{w}}))$ to $O(kn/N)$. The process of parsing the

Input: $\{\hat{a}_j \in R_q\}_{j\in[2^r]}$ for an odd q and $1 \le 2^r \le N$.
Output: $\hat{c} \in R_q$ such that $\hat{c}[i] = \hat{a}_{i \bmod 2^r}[\lfloor i/2^r \rfloor \cdot 2^r]$.

1: Compute $\hat{b}_j := 2^{-r} \cdot \hat{a}_j \bmod q$ for $\forall j \in [2^r]$.
2: **for** $k = 0, 1, \cdots, r-1$ and $h := 2^{r-k}/2$ **do in sequence**
3:     **for** $\forall j \in [h]$ **do in parallel**
4:         Right-shift first $\hat{b}_{j,h} := \hat{b}_{j+h} \cdot X^h \in R_q$.
5:         Bilinear $\hat{b}_j := \hat{b}_j + \hat{b}_{j,h} + \mathsf{Auto}(\hat{b}_j - \hat{b}_{j,h}, \frac{N}{h}+1)$.
6:     **end for**
7: **end for**
8: **return** $\hat{b}_0$ as $\hat{c}$.

Figure 6: Optimized `InterLeave` using a FFT-style algorithm.

interleaved polynomials and constructing the result matrix is explained in Figure 8 in the Appendix.

To achieve a good balance between computation and communication overhead, it is essential to choose an appropriate partition window m_{w}. A smaller value of m_{w} might increase the communication overhead in Step 1 excessively while a larger value of m_{w} results in more homomorphic automorphisms in Step 4. Given the matrix shape (k, m, n) and the partition windows $(k_{\mathrm{w}}, m_{\mathrm{w}}, n_{\mathrm{w}})$, the number of ciphertexts sent in Step 1 is $n_1 := m' \cdot \min(k', n')$. The total number of homomorphic automorphisms to perform `InterLeave` in Step 4 is $n_2 := \lceil k'n'/m_{\mathrm{w}} \rceil m_{\mathrm{w}}$. After the interleaving, $n_3 := \lceil \frac{k'n'}{m_{\mathrm{w}}} \rceil$ ciphertexts are sent in Step 5. To choose the proper partition window, we can minimize $\underset{k_{\mathrm{w}}, m_{\mathrm{w}}, n_{\mathrm{w}}}{\operatorname{argmin}} P_{\mathrm{C}} \cdot n_2 + P_{\mathrm{B}} \cdot (n_1 + n_3)$ under the constrain that m_{w} is a 2-power value and $1 \le k_{\mathrm{w}} m_{\mathrm{w}} n_{\mathrm{w}} \le N$. Here P_{C} models the price for computing one homomorphic automorphism and P_{B} models the price for sending one ciphertext. A smaller ratio $P_{\mathrm{C}}/P_{\mathrm{B}}$ can indicate the scenario of powerful computing or constraint bandwidth condition. For such cases, we prefer to set a larger m_{w}. Conversely, if an ample amount of bandwidth is accessible, a smaller value for m_{w} can be chosen to lighten the time taken for ciphertext interleaving.

Complexity. Based on our empirical results, it appears that selecting $m_{\mathrm{w}} \approx \sqrt{N}$ is a viable choice. Then the protocol in Algorithm 1 requires about $O(kn/\sqrt{N})$ homomorphic automorphisms for the ciphertexts interleaving. To compare, the $\mathtt{KRDY}^+$ baseline needs $O(kn)$ homomorphic automorphisms. As an example, let $m_{\mathrm{w}} = 2^6$ with $N = 2^{13}$. Under this setting, the ciphertext compression time of our protocol is approximately $1/2^6 \approx 1.6\%$ of the ciphertext compression time of $\mathtt{KRDY}^+$. The communication overhead of $\mathtt{KRDY}^+$ and ours are both $O(kn/N)$ ciphertexts. Note that `BOLT`'s prime-modulus OLT might require less homomorphic automorphisms $O(\sqrt{k^2m^2n/N^2})$ according to the matrix dimensions [58].

Security. We only add a ciphertext compression step to reduce the communication overhead, the information we send are a subset of `KRDY` protocols, so we have the same security level. The proof of Theoreom 1 follows [54], [36].

Algorithm 1 Proposed Oblivious Linear Transform Π_{OLT}

Inputs: Sender S: $\mathbf{Q} \in \mathbb{Z}_{2^\ell}^{k\times m}$ and sk. Receiver R: $\mathbf{V} \in \mathbb{Z}_{2^\ell}^{m\times n}$.
Output: $[\![\mathbf{U}]\!]$ such that $\mathbf{U} \equiv_\ell \mathbf{Q} \cdot \mathbf{V}$.
Public Params: $\mathsf{pp} = (\mathsf{HE.pp}, \mathsf{pk}, (k_{\mathrm{w}}, m_{\mathrm{w}}, n_{\mathrm{w}}))$

- The size m_{w} is a 2-power value, and $1 \le k_{\mathrm{w}} m_{\mathrm{w}} n_{\mathrm{w}} \le N$.
- $k' = \lceil \frac{k}{k_{\mathrm{w}}} \rceil$, $m' = \lceil \frac{m}{m_{\mathrm{w}}} \rceil$, $n' = \lceil \frac{n}{n_{\mathrm{w}}} \rceil$, and $\tilde{m} = \lceil \frac{k'n'}{m_{\mathrm{w}}} \rceil$.
- **Note:** If $k' > n'$ then flip the role of sender and receiver.

1: S first partitions the matrix $\mathbf{Q}$ into block matrices $\mathbf{Q}_{\alpha,\beta} \in \mathbb{Z}_\ell^{k_{\mathrm{w}} \times m_{\mathrm{w}}}$. Then S encodes each block matrices as a polynomial $\hat{q}_{\alpha,\beta} := \pi_{\mathrm{lhs}}(\mathbf{Q}_{\alpha,\beta})$ for $\alpha \in [k']$ and $\beta \in [m']$. After that S sends $\{\mathsf{ct}'_{\alpha,\beta} := \mathsf{RLWE}_{\mathsf{pk}}^{q,2^\ell}(\hat{q}_{\alpha,\beta})\}$ to R.
2: R first partitions the matrix $\mathbf{V}$ into block matrices $\mathbf{V}_{\beta,\gamma} \in \mathbb{Z}_\ell^{m_{\mathrm{w}} \times n_{\mathrm{w}}}$. Then R encodes each block matrices as a polynomial $\hat{v}_{\beta,\gamma} := \pi_{\mathrm{rhs}}(\mathbf{V}_{\beta,\gamma})$ for $\beta \in [m']$ and $\gamma \in [n']$.
3: On receiving $\{\mathsf{ct}'_{\alpha,\beta}\}$ from S, R computes a vector of RLWE ciphertexts, denoted as $\mathbf{c}$, where $\mathbf{c}[\alpha n' + \gamma] := \boxplus_{\beta\in[m']} \left(\mathsf{ct}'_{\alpha,\beta} \boxtimes \hat{v}_{\beta,\alpha}\right)$ for $\alpha \in [k'], \gamma \in [n']$,
4: To compress the the vector $\mathbf{c}$ of $k'n'$ ciphertexts into $\tilde{m}$ ciphertexts without touching the needed coefficients, R runs `InterLeave` on subvectors of $\mathbf{c}$. For example
$$\tilde{\mathbf{c}}[\theta] := \mathtt{InterLeave}([\underbrace{\mathbf{c}[\theta \cdot m_{\mathrm{w}}], \mathbf{c}[\theta \cdot m_{\mathrm{w}} + 1], \cdots}_{m_{\mathrm{w}}}]),$$
for $\theta \in [\tilde{m}]$. ▷ Pad with zero(s) when $k'n' \nmid m_{\mathrm{w}}$.
5: Call $\hat{c}_{i,0}, \hat{c}_{i,1} \leftarrow \mathcal{F}_{\mathrm{H2A}}(\tilde{\mathbf{c}}[i])$ on each ciphertext in $\tilde{\mathbf{c}}$, where S obtains $\hat{c}_{i,0} \in R_{2^\ell}$ and R obtains $\hat{c}_{i,1} \in R_{2^\ell}$, respectively. After that both S and R can derive their share using a local procedure $[\![\mathbf{U}]\!]_l := \mathsf{ParseMat}(\hat{c}_{0,l}, \hat{c}_{1,l}, \cdots)$ in Appendix.

Theorem 1. *The protocol Π_{OLT} in Algorithm 1 privately realizes the OLT functionality in presence of a semi-honest adversary under the $\mathcal{F}_{\mathrm{H2A}}$ hybrid.*

E. Further Optimizations

1) Batched Matrix Multiplications: Recall that each multi-head attention in a transformer model involves $H > 1$ parallel matrix multiplications, e.g., $\mathbf{Q}_j \cdot \mathbf{K}_j$ for $j \in [H]$. When each of these resultant matrix is small, i.e., $|\mathbf{Q}_j \cdot \mathbf{K}_j| \le N/2$, we prefer to apply `InterLeave` to compress the ciphertexts from the batch multiplications. Briefly, the $O(H)$ RLWE ciphertexts of the batch multiplications are further packed into $O((H \cdot kn)/N)$ ciphertexts.

2) Dynamic Compression Strategy: We adopt a dynamic strategy for ciphertext compression that allows us to strike a balance between reducing communication costs and incurring additional computation costs. Specifically, we *do not* use any ciphertext compression when there is only one RLWE ciphertext to send, i.e., $\lceil k/k_{\mathrm{w}} \rceil \cdot \lceil n/n_{\mathrm{w}} \rceil = 1$. Also when

$kn \ll N$, we use PackLWEs instead of InterLeave where the former can run faster for small cases.

IV. Faster Batch OLE from RLWE

Besides of matrix multiplications, scalar multiplications (or Hadamard product, in a batched way) are also required in transformer models. We use the notation Batch OLE (bOLE) [17] to describe a 2-party computation protocol that takes a vector $\mathbf{x}$ from a sender S, and a vector $\mathbf{y}$ from a receiver R, and generates the secret share $[\![\mathbf{x} \odot \mathbf{y}]\!]$ of their Hadamard product. In the context of private inference, we demonstrate that a variant of bOLE with Error (bOLEe) is sufficient. We can build more efficient bOLEe protocol using smaller RLWE parameters. Our bOLEe may introduce Least-Significant-Bit (LSB) errors in the final output. Nonetheless, due to the use of fixed-point representation in MPC, the LSB errors would be removed by subsequent truncations, so our approach does not affect the overall accuracy.

Similarly to the RLWE-based bOLE from [63], we also apply the SIMD technique [66] to implement bOLE but with a better amortized efficiency. In [63], a sender S pre-processes its private input $\mathbf{x} \in \mathbb{Z}_{2^\ell}^N$ as $\mathsf{SIMD}(\mathbf{x})$, and sends the ciphertext $\mathsf{RLWE}_S^{q,t}(\mathtt{SIMD}(\mathbf{x}))$ to the receiver R. Then R responses the ciphertext $\mathsf{RLWE}_S^{q,t}(\mathtt{SIMD}(\mathbf{x})) \boxtimes \mathtt{SIMD}(\mathbf{y}) \boxplus \mathtt{SIMD}(\mathbf{r})$ to S with a masking vector $\mathbf{r} \in \mathbb{Z}^N$. Indeed, the arithmetic $\mathbf{x} \odot \mathbf{y} + \mathbf{r}$ here is carried out modulo t, even though the values in $\mathbf{x}$ and $\mathbf{y}$ are from the ring $\mathbb{Z}_{2^\ell}$. Since the modulus t is indivisible by 2^ℓ, the masking vector $\mathbf{r}$ should be sampled from a larger ring to provide statistical security. Specifically, [63] samples $\mathbf{r}$ from $\mathbb{Z}_{2^{2\ell+\sigma}}^N$ to provide statistical security of σ-bit. In addition, to prevent the overflow from addition, they needs to set a large plaintext modulus $t > 2^{2\ell+\sigma+1}$. We present how to avoid this extra σ-bit overhead, at the cost of 1-bit LSB error. Basically, we use a lifting function to introduce an intermediate layer of a modulus of 2^ℓ upon the prime plaintext modulus t.

$$\mathsf{Lift}(\mathbf{x}) : \mathbb{Z}_{2^\ell}^N \mapsto \mathbb{Z}_t^N \text{ via } \lfloor \frac{t}{2^\ell} \cdot \mathbf{x} \rceil \bmod t,$$
$$\mathsf{Down}(\mathbf{y}) : \mathbb{Z}_t^N \mapsto \mathbb{Z}_{2^\ell}^N \text{ via } \lfloor \frac{2^\ell}{t} \cdot \mathbf{y} \rceil \bmod 2^\ell.$$

Proposition 2. *If $t > 2^{2\ell}$ then $\mathsf{Down}(\mathsf{Lift}(x) \cdot y \bmod t) \equiv_\ell x \cdot y$ given any $x, y \in \mathbb{Z}_{2^\ell}$. That is, we can operate modulo 2^ℓ ontop of the prime modulus t.*

Proof. It suffices to show the error term can be rounded to zero. $\mathsf{Lift}(x) \cdot y \bmod t$ can be written as $\frac{t}{2^\ell} \cdot ((x \cdot y \bmod 2^\ell) + r_e \cdot y$ for a round error $|r_e| \le 1/2$. Then the error term is rounded as $\lfloor \frac{2^\ell}{t} \cdot (r_e \cdot y) \rceil = 0$ given $t > 2^{2\ell}$ and $y < 2^\ell$. □

We are now able to use an informatics random masking $\mathbf{r} \in \mathbb{Z}_t^N$ instead of the statistical one. There might be a chance of introducing 1-bit error in the result.

Proposition 3. *Let $u' := \mathsf{Down}(\mathsf{Lift}(x) \cdot y - r \bmod t)$ and $v' := \mathsf{Down}(r)$ given $x, y \in \mathbb{Z}_{2^\ell}$ and $r \in \mathbb{Z}_t$. If $t > 2^{2\ell}$ and r distributes uniformly over $\mathbb{Z}_t$, then $u' + v' \equiv_\ell x \cdot y + e$ for $e \in \{0, \pm 1\}$.*

Algorithm 2 bOLE with Error Protocol Π_{bOLEe}

Input: Sender S: $\mathbf{x} \in \mathbb{Z}_{2^\ell}^N$, secret key sk. Receiver R: $\mathbf{y} \in \mathbb{Z}_{2^\ell}^N$. Public parameters $\mathsf{pp} = \{N, t\}$ such that $t = 1 \bmod 2N$ is a prime and $t > 2^{2\ell}$ and the public key pk.
Output: $[\![\mathbf{z}]\!] \in \mathbb{Z}_{2^\ell}^N$ such that $\|\mathbf{z} - \mathbf{x} \odot \mathbf{y} \bmod 2^\ell\|_\infty \le 1$.

1: S sends $\mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\hat{x})$ to S, where $\hat{x} := \mathtt{SIMD}(\mathsf{Lift}(\mathbf{x}))$.
2: R computes $\hat{y} := \mathtt{SIMD}(\mathbf{y})$.
3: On receiving the ciphertexts $\mathsf{RLWE}_{\mathsf{sk}}^{q,t}(\hat{x})$, R computes $\mathsf{ct} := \mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\hat{x}) \boxtimes \hat{y}$.
4: $[\![\hat{u}]\!] \leftarrow \mathcal{F}_{\mathrm{H2A}}(\mathsf{ct})$ to convert to arithmetic share. Suppose S's share is $[\![\hat{u}]\!]_0 \in R_t$ and R's share is $[\![\hat{u}]\!]_1 \in R_t$. Remind that we let the $\mathcal{F}_{\mathrm{H2A}}$ function to capture the circuit privacy (c.f. Remark 1).
5: S outputs $\mathsf{Down}(\mathtt{SIMD}^{-1}([\![\hat{u}]\!]_0))$.
6: R outputs $\mathsf{Down}(\mathtt{SIMD}^{-1}([\![\hat{u}]\!]_1))$.

The proof basically follows the similar arguments in [36, full version, Appendix C].

Theorem 2. *The protocol Π_{bOLEe} in Algorithm 2 privately realizes the bOLEe functionality (cf. Table I) in presence of a semi-honest adversary under the $\mathcal{F}_{\mathrm{H2A}}$ hybrid model.*

The correctness of Theorem 2 simply follows the SIMD packing and Proposition 3. For the concrete improvements, our bOLEe protocol requires $t \approx 2^{128}$ for $\ell = 64$, whereas the approach of [63] requires a larger value of $t \approx 2^{168}$. Our empirical results show that our protocol is about $1.3\times$ faster than their protocol.

V. Protocols for Activation Functions

We first describe our protocol for the GeLU function. The method and optimizations described in this section can also be used for other functions such as SiLU and ELU (see Appendix D).

A. Gaussian Error Linear Unit (GeLU)

A common definition of the GeLU function is

$$\mathtt{GeLU}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3))).$$

Following the existing works [58], [49], [24], [21], we approximate GeLU function using piecewise functions such as

$$\mathtt{Seg4GeLU}(x) = \begin{cases} -\epsilon & x < -5 \\ P^3(x) & -5 < x \le -1.97 \\ P^6(x) & -1.97 < x \le 3 \\ x - \epsilon & x > 3 \end{cases} \quad (2)$$

where $P^b(x)$ a degree-b polynomial that approximates the GeLU function in a short interval. For instance, setting $\epsilon = 10^{-5}$ we plot Seg4GeLU in Figure 7a. We can see that Seg4GeLU approximates the GeLU function very well. Our GeLU protocol in Algorithm 3 basically follows (2) using $\mathcal{F}_{\mathrm{less}}$ and $\mathcal{F}_{\mathrm{mux}}$ for the branch selection. To further enhance the efficiency, we discuss and introduce three independent optimizations which are not mentioned in the previous works [58], [49], [24], [21].

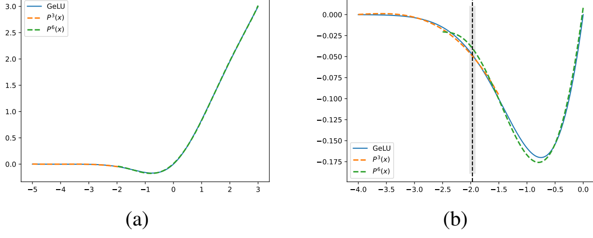

Figure 7: (Left) The maximum absolute error between `Seg4GeLU` and `GeLU` within the interval $[-5, 3]$ is about 1.5×10^{-2}. (Right) We use a wider range for the polynomial fitting which gives $P^3(x) \approx P^6(x)$ for x around the pivot.

*1) **Approximated Branch Selection**:* The first optimization leverages the smoothness of the activation function. Specifically, we first find the approximation polynomials $P^3(x)$ and $P^6(x)$ such that $P^3(x) \approx P^6(x)$ for x around the pivot point, e.g., $x = -1.97$ in our case. The selection in (2) can be privately achieved via a less-than protocol $\mathbf{1}\{x < y\}$ for $x, y \in \mathbb{Z}_{2^\ell}$. Given these observations, we suggest computing the less-than bit with some low-end f' bits ignored to reduce communication overhead. This is a common folkloric optimization for performing comparisons over secret shares of fixed-point values. However, it is crucial to ensure proper approximation polynomials, as fluctuations exceeding the pivot point might ruin the approximation. Empirically, the approximated segment selection helps reduce the communication overhead of the GeLU protocol in Algorithm 3 by 5%.

*2) **Batching (Approximated) Branch Selection**:* Our second optimization is specified for the concrete OT-based MSB protocol of [46], [36]. These protocol compute the MSB of an arithmetic share $[\![x]\!]$ using the formula $\mathrm{MSB}([\![x]\!]_0) \oplus \mathrm{MSB}([\![x]\!]_1) \oplus \mathbf{1}\{([\![x]\!]_0 \bmod 2^{\ell-1}) + ([\![x]\!]_1 \bmod 2^{\ell-1}) \geq 2^{\ell-1}\}$, where the computation of the last bit needs one $\binom{M}{1}$-$\mathtt{OT}_2$. Thus we need 3 calls to $\binom{M}{1}$-$\mathtt{OT}_2$ for the branch selection in (2). Note that these "batched" comparisons are conducted using one secret share and multiple plaintext thresholds. This arrangement allows us to consolidate three calls to $\binom{M}{1}$-$\mathtt{OT}_2$ into a single $\binom{M}{1}$-$\mathtt{OT}_6$, that is 1-of-M OT on 6-bit messages. While this OT combination may not reduce communication overhead by too much, it can reduce the computation time of GeLU by 35% according to our experiments. This is because the running time of one call to $\binom{M}{1}$-$\mathtt{OT}_2$ is similar with one call to $\binom{M}{1}$-$\mathtt{OT}_6$ when using `Ferret` OT.

*3) **Optimizing Polynomial Evaluation**:* For the evaluation of $P^3(x)$ and $P^6(x)$ in (2), we suggest to use the faster square protocol Π_{square} to compute all even-power terms, such as x^2, x^4 and x^6. Indeed, in 2PC, the cost of performing a square is half of a standard multiplication operation. Additionally, we present to reduce the communication cost for the odd-power terms by half. As an example, let us consider the computation of $[\![x^3]\!]$ given $[\![x]\!]$ and $[\![x^2]\!]$. We can employ two calls of bOLEs to compute $[\![x^3]\!]$, i.e., $\mathcal{F}_{\mathrm{bOLE}}([\![x]\!]_0, [\![x^2]\!]_1)$ and $\mathcal{F}_{\mathrm{bOLE}}([\![x^2]\!]_0, [\![x]\!]_1)$. For our bOLE construction in Algorithm 2, P_0 sends the ciphertexts of $\mathsf{SIMD}(\mathsf{Lift}([\![x]\!]_0))$ and $\mathsf{SIMD}(\mathsf{Lift}([\![x^2]\!]_0))$ to P_1. We note that these two ciphertexts are already sent to P_1 when computing $[\![x^2]\!]$ and $[\![x^4]\!]$, respectively. Therefore, to compute the cubic term, the players can skip Step 1 in Algorithm 2, and follow the remaining steps identically.

Algorithm 3 Private GeLU protocol Π_{GeLU}

Input: $[\![\tilde{x}; f]\!]$ with f-bit fixed-point precision. The polynomial coefficients $\{a_0, a_1, a_2, a_3\}$ in $P^3(x)$ and the coefficients $\{b_0, b_1, b_2, b_4, b_6\}$ in $P^6(x)$.
Output: $[\![\mathtt{Seg4GeLU}(\tilde{x}); f]\!]$. See (2) for definition.

1: Compute the powers $[\![\tilde{x}^2]\!] \leftarrow \Pi_{\mathrm{square}}([\![\tilde{x}]\!])$, $[\![\tilde{x}^4]\!] \leftarrow \Pi_{\mathrm{square}}([\![\tilde{x}^2]\!])$, $[\![\tilde{x}^3]\!] \leftarrow \Pi_{\mathrm{mul}}([\![\tilde{x}^2]\!], [\![\tilde{x}]\!])$, and $[\![\tilde{x}^6]\!] \leftarrow \Pi_{\mathrm{square}}([\![\tilde{x}^3]\!])$. The truncations are implicitly called.
2: Evaluate two polynomials $[\![P^3(\tilde{x}) + \epsilon; f]\!] \leftarrow \mathcal{F}^f_{\mathrm{trunc}}(\lfloor(\epsilon + a_0) \cdot 2^{2f}\rfloor + \sum_{k\in\{1,2,3\}} [\![\tilde{x}^k]\!] \cdot \lfloor a_k \cdot 2^f\rfloor)$, and $[\![P^6(\tilde{x}) + \epsilon; f]\!] \leftarrow \mathcal{F}^f_{\mathrm{trunc}}(\lfloor(\epsilon + b_0) \cdot 2^{2f}\rfloor + \sum_{k\in\{1,2,4,6\}} [\![\tilde{x}^k; f]\!] \cdot \lfloor b_k \cdot 2^f\rfloor)$.
3: Compute the comparisons for segement selection

$$[\![b_0]\!]^B \leftarrow \mathcal{F}_{\mathrm{less}}([\![\tilde{x}]\!], -5) \qquad \triangleright\, b_0 = \mathbf{1}\{\tilde{x} < -5\}$$
$$[\![b_1]\!]^B \leftarrow \mathcal{F}_{\mathrm{less}}([\![\tilde{x}]\!], -1.97) \qquad \triangleright\, b_1 = \mathbf{1}\{\tilde{x} < -1.97\}$$
$$[\![b_2]\!]^B \leftarrow \mathcal{F}_{\mathrm{less}}(3, [\![\tilde{x}]\!]) \qquad \triangleright\, b_2 = \mathbf{1}\{3 < \tilde{x}\}$$

Locally sets $[\![z_0]\!]^B_l := [\![b_0]\!]^B_l \oplus [\![b_1]\!]^B_l$, $[\![z_1]\!]^B_l := [\![b_1]\!]^B_l \oplus [\![b_2]\!]^B_l \oplus l$ and $[\![z_2]\!]^B_l := [\![b_2]\!]^B_l$. Note $z_0 = \mathbf{1}\{-5 < \tilde{x} \leq -1.97\}$, $z_1 = \mathbf{1}\{-1.97 < \tilde{x} \leq 3\}$, and $z_2 = \mathbf{1}\{3 < \tilde{x}\}$.
4: Compute the multiplexers $[\![z_0 \cdot (P^3(\tilde{x}) + \epsilon)]\!]$, $[\![z_1 \cdot (P^6(\tilde{x}) + \epsilon)]\!]$, and $[\![z_2 \cdot \tilde{x}]\!]$ using the $\mathcal{F}_{\mathrm{mux}}$ functionality. Then P_l locally aggregates them and outputs as the share of $[\![\mathtt{Seg4GeLU}(\tilde{x}); f]\!]_l$.

Tradeoffs. In our evaluations, we demonstrate that these three optimizations can result in a 35% reduction in time and a 7% reduction in communication. It is important to note that these improvements come with an increase in approximation error. Specifically, the average ULP error for our GeLU evaluation is approximately 11, compared to an average ULP error of 4 for [58]. Additionally, we tested the proposed GeLU protocols in four transformers, and the results indicate a minimal inference precision downgrade of less than 1%. We consider this a reasonable tradeoff between accuracy and efficiency.

B. Softmax

For the sake of numerical stability [29, Chapter 4], the softmax function is commonly computed as

$$\mathtt{softmax}(\mathbf{x})[i] = \frac{\exp(\mathbf{x}[i] - \bar{x})}{\sum_j \exp(\mathbf{x}[j] - \bar{x})}, \tag{3}$$

where $\bar{x}$ is the maximum element of the input vector $\mathbf{x}$. For a two-dimension matrix, we apply (3) to each of its row vector. It is worth noting that all inputs to the exponentiation operation in (3) are negative. We leverage the negative operands to accelerate the private softmax. Particularly, we approximate the exponentiation with a simple clipping branch. That is $\exp(x) \approx \left(1 + \frac{x}{2^n}\right)^{2^n}$ for $x \in [T_{\exp}, 0]$ otherwise $\exp(x) \approx 0$ for $x < T_{\exp}$. For the clipping range $T_{\exp}$, we simply set $T_{\exp}$ such that $\exp(T_{\exp}) \approx 2^{-f}$ where f is the

fixed-point precision. Suppose we set $f = 18$. Then we set $T_{\exp} = -13$ since $\exp(-13) < 2^{-18}$. When $T_{\exp}$ is fixed, we can empirically set the Taylor expansion degree n. For instance, in our experiments, we set $n = 6$ for $T_{\exp} = -13$ to achieve an average error within 2^{-10}. Also we apply the approximated less-than proposed in the previous section for the branch selection since the exponentiation on negative inputs is smooth too. The division by 2^n can be achieved using a call to $\mathcal{F}^n_{\text{trunc}}$, and the power-of-$2^n$ is computed via a sequences of Π_{square}.

C. Mixed-Bitwidth Evaluation

Before running our activation protocols, we can first switch the shares to a smaller ring $\ell' < \ell$ to save communication overhead. The share conversion from a large ring $\mathbb{Z}_{2^\ell}$ to a smaller ring $\mathbb{Z}_{2^{\ell'}}$ can be done locally via setting $[\![x]\!]_l \bmod 2^{\ell'}$. For the opposite direction, we combine the ring extension protocol from [61] with a heuristic optimization [16]. For our implementation, each conversion from $\mathbb{Z}_{2^{\ell'}}$ to $\mathbb{Z}_{2^\ell}$ will exchange about $O(2(\ell - \ell'))$ bits in two rounds. The major issue in this mixed-bitwidth evaluation is the overflow from multiplication. For instance, the double fixed-point precision $2f$ might be already larger than ℓ'. To address this issue, we have to make two compromises. We must 1) reduce the fixed-point precision **within** the activation function using one extra call to $\mathcal{F}_{\text{trunc}}$, 2) and shrink the approximation interval or increase the number of branches so that we can use lower degree polynomials. The recent work [58] approximates the GeLU within a narrow interval using one low-degree polynomial, i.e., $\texttt{GeLU}(x) \approx P^4(x) = 0.5x + \sum_{i=0}^{i=4} c_i|x|^i$ for $x \in [-2.7, 2.7]$. In some of our experiments, we apply this $P^4(x)$ for GeLU but we still need two polynomials for the SiLU function.

VI. Related Work

Low Communication OLT. Many works that leverages the homomorphic SIMD [66] can be directly used for the OLT functionality with a relatively small communication overhead, such as [32], [12], [39], [35]. The SIMD technique in turn demands a prime plaintext modulus t rather than the 2^ℓ modulus used by secret sharing. One can use the Chinese Remainder Theorem to accept secret shares from $\mathbb{Z}_{2^\ell}$ at the cost of increasing the computation and communication overheads on the HE-side by many times. An alternative strategy for constructing low-communication OLT is to use Vector Oblivious Linear Evaluation (VOLE) [9], [77], [7], as proposed in `CipherGPT` [34]. However, this VOLE-based approach requires a significantly large matrix dimension of $k \approx 10^7$ to achieve low (amortized) communication, making it suitable only for auto-regression transformers.

Non-linear Functions. For the softmax function, [45], [68] also use the Taylor series $(1 + x/2^n)^{2^n}$ to approximate the exponentiation except they do not apply the range clipping. For example, `CryptGPU` empirically sets $n = 9$ according to their examination on some datasets. `SiRNN` [61] uses a Look-up-Table for an initial guess with a few Newton iterations. These approaches do provide a precise exponentiation but are also expensive to evaluate in 2PC. [80] approximates the softmax function using an alternative numerical approach. However, it may require more than 64 rounds of multiplications, making it communication-intensive. Kelkar et al. [42] propose a novel 2PC exponentiation protocol but with limitations of using a large ring $\ell \approx 128$ or to strictly constrain the input domain (e.g., $|x| \le 5$) due to a failure probability. [49] uses 12 polynomials of degree-1 to approximate the activation functions via Garbled Circuit [78]. The recent approaches also [20], [21], [24], [58] approximate the GeLU function using multiple low-degree polynomials. However, none of these approaches consider the smoothness of the GeLU function and batched comparisons to reduce the overhead of the branch selection.

Private Transformer Inference. (2PC). `Iron` [33], `BOLT` [58] and `CipherGPT` [34] are the 2PC inference frameworks designed for transformers. Both these three frameworks heavily re-use the OT-based protocols from the `SiRNN` framework [61], [23] for evaluating the activation functions. `BOLT` [58] is considered state-of-the-art for secure two-party transformer inference. `BumbleBee` shares some design elements with `BOLT`, such as piecewise approximation for nonlinear functions and private matrix multiplication using HE. However, `BumbleBee` outperforms `BOLT`, particularly in terms of communication efficiency (c.f. Figure 1). The primary reasons for `BumbleBee`'s advantages over `BOLT` are summarized as follows:

- **Share-type Consistent Multiplications.** `BOLT`'s matrix multiplication protocol works over a prime modulus p while their non-linear protocols works over a ring modulus 2^ℓ. They have to switch back and forth between different share types during the private inference. To integrate a ring-to-prime conversion (e.g., [42]), `BOLT` requires a special share multiplication where the bitwidth of the outputs can be larger than the bitwidth of the inputs. Specifically, `BOLT` reuses the OT-based protocol from the EzPC framework [23], [61] to achieve this non-uniform share multiplication.
  In contrast, in `BumbleBee`, both the matrix multiplication protocol and the share multiplication protocol work over the ring modulus 2^ℓ. These share-type consistent multiplications enable us to build a more efficient share multiplication protocol from HE. As already shown by existing works such as [63], the HE-based share multiplication protocol can be $5\times$–$6\times$ more efficient than the OT-based counterpart in terms of communication. Note that we further optimize the communication costs of [63] by $1.3\times$.
- **Square is Half the Cost of Standard Multiplication.** To privately evaluate the piecewise function, we need to evaluate a low-degree polynomial on a secretly shared input $[\![x]\!]$. `BOLT` leverages Horner's method for polynomial evaluation. For example, `BOLT` evaluates a degree-4 polynomial as follows:
  $$P^4([\![x]\!]) = (((((a_4 \cdot [\![x]\!] + a_3) \cdot [\![x]\!]) + a_2) \cdot [\![x]\!]) + a_1) \cdot [\![x]\!] + a_0,$$
  which requires 3 standard multiplications of secret shares.

BOLT applies Motzkin's polynomial preprocessing to reduce the number of standard multiplications down to 2. In contrast, in BumbleBee, we prefer to leverage the square because the cost of performing a square is half that of a standard multiplication operation. For example, we need two squares (for the quadratic and quartic terms) and "half" standard multiplication (for the cubic term) to evaluate the degree-4 polynomial. Let alone the more efficient share multiplication protocol. The way we evaluate a given low-degree polynomial is also efficient than the Horner method used by BOLT, particularly in the context of private computation.

- **Lower Communication OT.** The final advantage comes from the specific choice of OT. Specifically, BOLT chooses to use the IKNP OT [38] as its underlying OT protocol while we leverage the Ferret OT [77]. The communication overhead of Ferret OT is smaller than that of IKNP OT, at the cost of more local computation. We have made engineering efforts to integrate Ferret OT into a multi-core CPU to minimize its running time, including the syncrhonization across multiple threads.

From the summary above, we emphasize the importance of our matrix multiplication protocol that supports a ring modulus. It is the share-type consistency of our multiplication protocols that allows BumbleBee to significantly outperform BOLT.

Private Transformer Inference ((2+1)-PC). Recent studies such as [48], [31], [67], [30] have examined scenarios in which both parties (e.g., S and C) can access to a trusted third party (TTP) for help. These approaches typically divide the private computation process into two distinct phases: an initial preprocessing, followed by an evaluation phase. Notably, during the preprocessing phase, the TTP is responsible for generating and distributing all correlated randomness that the two parties will subsequently utilize during the evaluation stage. Following the name of [67], we denote such a TTP-assisted computation as a (2+1)-PC to clearly distinguish it from our 2PC setting.

Private Transformer Inference. (3PC). PrivFormer [3] and PUMA [21] two private inference frameworks designed for transformers based on the three-party setting [4], [56]. Specifically, PrivFormer replaces the softmax attention by an MPC-friendly alternative (i.e., ReLU attention) which requires model fine-tune. PUMA shares some design elements with BumbleBee, such as the piecewise approximation for GeLU, and Taylor approximation for the exponentiation function. In brief, these 3PC approaches are OT-free and HE-free but at the costs of a larger communication than ours.

Other related works [79], [48], [3] consider alternative models with different approximated structures that are easier to compute in MPC. However it has been demonstrated that rough approximations can significantly compromise model accuracy [43], so model fine-tune would be necessary in their works. One of BumbleBee's advantage is that it **does not require any model fine-tune**; instead, our focus is on the private inference of a given pre-trained transformer model. Note that our techniques could also be adapted for use in their models to yield even better performance.

VII. EVALUATIONS

Models & Datasets. We evaluate BumbleBee on 5 Transformer models, including four NLP models, i.e., BERT-base, BERT-large [19], GPT2-base [15], and LLaMA-7B [69], and a computer vision model ViT-base [74], [22]. These models are parameterized by three hyper-parameters: the number of blocks B, the dimension of representations D and the number of heads H. We directly reuse the trained models from publicly available sources.

To demonstrate the effectiveness of BumbleBee, we conducted private inference on 4 datasets, which includes CoLA, RTE, and QNLI from the GLUE benchmarks [71] for NLP tasks, and ImageNet-1k [64] for image classification. In more details, the ImageNet-1k dataset is a classification task of 1000 different classes, while the three from the GLUE benchmarks are binary classification tasks.

Metrics. Given the 2PC setting, we do not distinguish between the "offline" and "online" costs as in some prior (2+1)-PC works [55], [30]. We report the end-to-end running time including the time of transferring ciphertexts through the network. However, we have not included the time taken for loading models from a hard disk. We measure the total communication including all the messages sent by the two parties. We write 1GB = 2^{10}MB = 2^{30} bytes.

Testbed Environment. The experiments described in this paper were majorly conducted on two Alibaba Cloud instances (ecs.g7.16xlarge), equipped with 64 vCPUs, operating at 2.70GHz, and 256 GB of RAM. We utilize multi-threading as much as possible. To simulate different network conditions, we manipulated the bandwidth between the cloud instances using the traffic control command in Linux. Specifically, we conducted our benchmarks in two network settings: a Local Area Network (LAN) with of 1 gigabit per second (one-way 1 Gbps) bandwidth, and 0.5ms of ping time, and a Wide Area Network (WAN) with 400 Mbps bandwidth and 4ms ping time.

Concrete Parameters. We set $\ell = 64$ for the secret sharing and set the fixed-point precision $f = 18$. When doing the mixed ring optimization on GeLU and exponentiation, we set $\ell' = 32$ and $f' = 12$. Specifically, we apply the mixed ring optimization for GeLU and exponentiation on the BERT-base, GPT2-base and ViT-base models, while we only apply to the GeLU/SiLU activation for the other larger models. We extend the Ferret implementation in the Yacl library [1] to support various application-level OT types, such as $\binom{M}{1}$-OT_6. For RLWE, we use the SEAL library [65] with [37] for acceleration on Intel CPUs. For the approximated less-than, we evaluate an MSB protocol on 50-bit inputs for the segment selection in (2). For approximating the exponentiation, we set $T_{\text{exp}} = -14$ and use $n = 6$. More details on the implementation parameters refer to Appendix.

Availability. We provide a reproducible implementation in https://github.com/AntCPLab/OpenBumbleBee.

A. Microbenchmarks

In Table II, we compare the performance of the proposed protocols with many SOTA approaches.

TABLE II: Comparison of proposed protocols with SOTA in terms of running time and communication costs. The time for HE key generation and base OTs are **excluded**. Each machine was tested with 25 threads. For BOLT's implementation [58], we set $\ell = 37$ to follow their settings. We evaluated GeLU using the mixed-bitwidth approach in §V-C.

(k, m, n)		$\Pi_{\text{OLT}}(\mathbf{X}, \mathbf{Y})$ Comm.	LAN	WAN
$(1, 50257, 768)$	[23]	9.41GB	96.22s	217.03s
	KRDY	20.84MB	0.45s	0.73s
	KRDY$^+$	1.38MB	0.46s	0.46s
	Ours*	1.38MB	0.46s	0.46s
$(128, 768, 768)$	[23]	18.41GB	159.98s	397.83s
	KRDY	30.16MB	0.66s	1.06s
	KRDY$^+$	5.02MB	7.07s	7.85s
	Ours	5.02MB	0.51s	0.52s
		$\Pi_{\text{bOLE}}(\mathbf{x}, \mathbf{y})$ Comm.	LAN	WAN
$\lvert\mathbf{x}\rvert = \lvert\mathbf{y}\rvert = 2^{15}$	[63]	4.67MB	0.07s	0.17s
	Ours	3.61MB	0.05s	0.14s
$\lvert\mathbf{x}\rvert = \lvert\mathbf{y}\rvert = 2^{20}$	[63]	139.94MB	2.39s	5.33s
	Ours	105.93MB	1.71s	4.02s
		$\Pi_{\text{GeLU}}(\mathbf{x})$ Comm.	LAN	WAN
$\lvert\mathbf{x}\rvert = 2^{20}$	[61]	16.06GB	141.52s	353.76s
	[53]	3.54GB	66.50s	103.68s
	[58]	3.85GB	33.41s	83.47s
	Ours†	0.43GB	6.47s	11.37s
	Ours‡	0.42GB	4.16s	9.42s
	Ours	0.40GB	3.95s	8.82s
$\lvert\mathbf{W}\rvert$		$\Pi_{\text{softmax}}(\mathbf{W})$ Comm.	LAN	WAN
$(960, 180)$	[61]	1.66GB	16.39s	40.84s
	[44], [53]	0.66GB	9.28s	14.53s
	[58]	1.25GB	11.00s	27.12s
	Ours	0.27GB	3.03s	5.79s

* Identical to KRDY$^+$ in this case due to the strategy in §III-E2

† We call 3 $\binom{M}{1}$-OT_2 without approximated less-than in this run.

‡ We call 1 $\binom{M}{1}$-OT_6 without approximated less-than in this run.

Linear Operations. To demonstrate the performance of our OLT protocol, we have implemented the approach from KRDY and our baseline KRDY$^+$ which adapts the PackLWEs [13] to reduce the communication overhead of KRDY. We also compared with the COT-based approach from the SCI_{OT} library [23], which leverages the IKNP for COT [38]. We can see that KRDY is already more efficient than the COT-based method. KRDY$^+$ could bring down 80% – 90% of the communication compared with KRDY, but the overall time cost is higher than KRDY on large matrices, this is because of the expensive homomorphism in PackLWEs. Our OLT protocol was shown to be rapid and light. With our ciphertext interleaving technique, our OLT protocol has the same communication complexity as KRDY$^+$, while $14\times$ faster than KRDY$^+$.

Non-linear Activation Functions. We compare our non-linear protocols with the default implementations in the SPU library [53] by changing their underlying OT to Ferret [72].

TABLE III: Prediction accuracy on the GLUE benchmarks using BERT-base, and classification accuracy on the ImageNet-1k dataset using ViT-base. We report Matthews correlation (higher is better) for CoLA and Top-1 (Top-3) accuracy for the ImageNet-1k dataset.

Dataset	Size	Class Distribution	Plaintext	BumbleBee
RTE	277	131/146	0.7004	0.7004
QNLI	1000	519/481	0.9030	0.9020
CoLA	1043	721/322	0.6157	0.6082
ImageNet	1000	one img per class	0.7317 (0.894)	0.7257 (0.891)

Also, we compare with SiRNN [61] and BOLT [58] by re-running their codes. The results in Table II demonstrate that our protocols yielded a significant decrease in communication costs, resulting in savings of about 89% communication for GeLU and 80% for softmax. Additionally, we run our GeLU protocol without the two optimizations, namely the approximated less-than (§V-A1) and batched MSBs (§V-A2). Our findings indicate that these two optimizations can reduce the computation time of GeLU by about 3% and 35% respectively.

B. Evaluation on Large Transformers

We have run BumbleBee on five transformer models, including four NLP models (BERT-base, BERT-large, GPT2-base, and LLaMA-7B) and one vision transformer (ViT-base). Note we select top-1 token in the GPT2 and LLaMA models.

Accuracy. To demonstrate the effectiveness of BumbleBee, we performed private inference on the BERT-base and ViT-base models on four datasets. The results are given in Table III. As shown in the table, BumbleBee attains comparable levels of accuracy when compared to the cleartext prediction. It is important to highlight that all our experiments were conducted using the proposed 2PC protocols rather than through cleartext simulation. Also, we **do not** perform any model fine-tunning.

Break-down. Table IV breaks down the BumbleBee inference time and communication for GPT2-base (left) and LLaMA-7B (right). The inputs to these two models consist of 128 and 8 tokens, respectively. The first column of Table IV indicates a specific operation in the SPU framework. Operations beginning with the "i_" prefix take integer inputs, while those beginning with the "f_" prefix take fixed-point values as inputs. The truncation protocol [16], [36] may be implicitly invoked within these fixed-point operations. In terms of communication, the activations make up about 50% of the communication costs. In contrast, matrix multiplication is the most expensive operation which takes 70% – 95% of the total inference time. We provide the performance breakdown of ViT-base in Appendix.

From the results, we could observe the striving directions: even after our optimizations, matrix multiplication (mmul)/scalar multiplication (mul)/GeLU/softmax still constitute the major cost in secure transformer inferences.

Discussions. Indeed, the cost of i_equal can be saved by allowing the client to query an one-hot vector instead of the token-id. Nonetheless, there are scenarios where a private conversion from token-id to one-hot vector is still required,

TABLE IV: Performance breakdown of BumbleBee on two transformers. The input to the GPT2 model and LLaMA-7B model consist of 128 and 8 tokens, respectively. Both models generate 1 token. The LAN setting was used.

Operation	Used by	GPT2-base ($B=12$, $D=768$, $H=12$) #Calls	Time (sec)	Sent (MB)	Recv (MB)	LLaMA-7B ($B=32$, $D=4096$, $H=32$) #Calls	Time (sec)	Sent (MB)	Recv (MB)
i_equal	token-id to one-hot	128	6.67	70.83	34.05	8	3.76	11.10	9.64
mixed_mmul	embedding lookup	128	32.59	265.47	41.88	8	30.89	18.31	16.39
f_mmul	linear projections	49	58.35	230.82	230.82	225	747.25	900.47	457.92
f_batch_mmul	multi-head attention	24	12.17	165.72	156.02	64	10.94	403.26	400.21
f_less	max / argmax	131	6.59	159.90	32.79	117	0.73	6.08	1.31
multiplixer	max / argmax	386	0.99	24.65	23.42	155	0.30	1.20	1.20
f_exp	softmax	12	15.30	779.69	636.92	32	2.27	39.33	36.08
f_reciprocal	softmax	12	4.18	28.18	29.08	32	1.36	12.87	7.95
f_mul	layer norm, softmax	174	17.91	880.51	878.63	356	10.53	758.30	730.10
f_rsqrt	layer norm	25	0.93	3.13	2.62	65	1.14	0.81	0.58
f_gelu	GeLU / SiLU	12	20.55	1088.35	724.91	32	17.99	1175.45	745.04
		Total	3.06min	Sent + Recv 6.61 GB		Total	13.87min	Sent + Recv 5.64 GB	

such as when outsourcing private inference to two collusion-free servers, where the generated token (in the middle of the computation) is also unknown by both servers.

C. Comparison with Existing Frameworks

We mainly compare with two existing 2PC frameworks: Iron [33] and BOLT [58]. Note that both Iron and BOLT have only considered the BERT-base transformer model. For the other frameworks, i.e., MPCFormer [48], PUMA [21], and SIGMA [30], they have a different threat model as the 2PC ones, but we list their performances for completeness. Also we note the communication costs associated with the GPT2-base model of CipherGPT [34], which amount to 14GB for 256 input tokens. The results of BOLT, MPCFormer, and PUMA are obtained by re-running their codes[3] under ours environment while that of SIGMA are estimated using the reported numbers from their paper. In SIGMA, the TTP is required to distribute a FFS key with a size of 45.06 GB to each party for the BERT-large model. Assuming our LAN setting, this transfer would take approximately $45.06 \cdot 2$ GB $/1$ Gbps $=12$ minutes.

We present the comparisons in Table V. In summary, we have achieved up to $3\times$ (resp. $13\times$) improvements in inference time while and a reduction of communication costs by 90% (resp. 92%) compared to the BOLT (resp. Iron) framework. BumbleBee is about $1.3\times-1.5\times$ slower than MPCFormer and PUMA in LAN, which is expected since these approaches are OT-free and HE-free. However, BumbleBee can perform better than MPCFormer and PUMA in the WAN setting, as BumbleBee requires 50% – 60% less communication compared to them. This translates similarly when considering SIGMA, due to their necessity of transferring a substantially large FSS key.

TABLE V: End-to-end comparisons with the existing private inference frameworks. The numbers of Iron and CipherGPT are taken from their papers. The timings of SIGMA include both the key-transmission and online inference, which are estimated based on our bandwidth. GPT2 models generated 1 token. Frameworks marked with "*" **are not** 2PC framework.

Model	Framework	Total Time (min) LAN	WAN	Comm. (GB)
BERT-base 128 input tokens	Iron	≈ 34	–	76.50
	BOLT	8.89	16.90	59.61
	SIGMA*	≈ 4	≈ 12	34.37
	MPCFormer*	2.79	5.09	12.08
	PUMA*	2.19	4.55	10.77
	BumbleBee	2.55	4.86	6.40
BERT-large 128 input tokens	Iron	≈ 92	–	≈ 220
	SIGMA*	≈ 12	≈ 31	92.75
	MPCFormer*	4.52	9.81	32.58
	PUMA*	4.02	9.06	27.25
	BumbleBee	6.19	9.81	16.37
GPT2-base 64 input tokens	SIGMA*	≈ 4	≈ 10	28.71
	MPCFormer*	1.10	2.85	7.32
	PUMA*	1.20	2.42	7.82
	BumbleBee	1.48	2.05	2.77

VIII. Conclusion

Private and accurate two-party inference on large transformer is possible. We present a highly optimized 2PC framework BumbleBee that can run large transformers with a significantly less overhead than the previous arts. Our approach offers a promising way forward for advancing the use of privacy-enhancing techniques. Considering future development of network and hardware, using privacy-preserving transformer models in a variety of applications would be possible. In the future, we would like to apply specialized hardware to accelerate the ciphertext interleaving procedure. A number of studies have demonstrated that the use of GPUs can improve the speed of homomorphic automorphisms by up to two orders of magnitude [40], [76].

[3] BOLT https://github.com/Clive2312/EzPC/tree/bert/SCI, MPCFormer https://github.com/DachengLi1/MPCFormer and PUMA https://github.com/secretflow/spu/tree/main/examples/python/ml/flax_llama7b.

Acknowledgment

The authors would like to express their gratitude to the anonymous reviewers and Mingbo Li from Xiamen University for their insightful comments. The authors would also like to thank Qi Pang from CMU for helpful discussions on BOLT. They also extend their thanks to the SPU team from Ant Group

for their constructive support in developing the BumbleBee framework.

REFERENCES

[1] (2023, Sep.) YACL (Yet Another Common crypto Library). https://github.com/secretflow/yacl.

[2] N. Agrawal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón, "QUOTIENT: two-party secure neural network training and prediction," in *CCS*, 2019, pp. 1231–1247.

[3] Y. Akimoto, K. Fukuchi, Y. Akimoto, and J. Sakuma, "Privformer: Privacy-Preserving Transformer with MPC," in *EuroSP*, 2023, pp. 392–410.

[4] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority," in *CCS*, 2016, pp. 805–817.

[5] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lucic, and C. Schmid, "ViViT: A Video Vision Transformer," in *ICCV*, 2021, pp. 6816–6826.

[6] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More Efficient Oblivious Transfer and Extensions for Faster Secure Computation," in *CCS*, 2013, pp. 535–548.

[7] C. Baum, L. Braun, A. Munch-Hansen, and P. Scholl, "Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs over $\mathbb{Z}_{2^k}$," in *CRYPTO*, vol. 13510, 2022, pp. 329–358.

[8] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC*, 1988, pp. 1–10.

[9] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round OT extension and silent non-interactive secure computation," in *CCS*, 2019, pp. 291–308.

[10] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.

[11] O. Catrina and A. Saxena, "Secure Computation with Fixed-Point Numbers," in *FC*, vol. 6052, 2010, pp. 35–50.

[12] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *CCS*, 2019, pp. 395–412.

[13] ——, "Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts," in *ACNS*, 2021, pp. 460–479.

[14] P. Cheng and U. Roedig, "Personal Voice Assistant Security and Privacy - A Survey," *Proc. IEEE*, vol. 110, no. 4, pp. 476–507, 2022.

[15] V. Cohen and A. Gokaslan, "OpenGPT-2: Open language models and implications of generated text," *XRDS*, vol. 27, no. 1, 2020.

[16] A. P. K. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 4, pp. 355–375, 2020.

[17] L. de Castro, C. Juvekar, and V. Vaikuntanathan, "Fast vector oblivious linear evaluation from ring learning with errors," in *WAHC*, 2021, pp. 29–41.

[18] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *NDSS*, 2015.

[19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019, pp. 4171–4186.

[20] Y. Ding, H. Guo, Y. Guan, W. Liu, J. Huo, Z. Guan, and X. Zhang, "East: Efficient and Accurate Secure Transformer Framework for Inference," vol. abs/2308.09923, 2023.

[21] 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," *CoRR*, vol. abs/2307.12533, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.12533

[22] 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.

[23] (2021, Jun.) EzPC - a language for secure machine learning.

[24] X. Fan, K. Chen, G. Wang, M. Zhuang, Y. Li, and W. Xu, "Nfgen: Automatic non-linear function evaluation code generator for general-purpose MPC platforms," in *CCS 2022*, pp. 995–1008.

[25] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright, "Secure multiparty computation of approximations," *ACM Trans. Algorithms*, pp. 435–472, 2006.

[26] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC 2009*, 2009, pp. 169–178.

[27] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *TOC*, 1987, p. 218–229.

[28] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[30] 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.*, 2024. [Online]. Available: https://eprint.iacr.org/2023/1269

[31] K. Gupta, D. Kumaraswamy, N. Chandran, and D. Gupta, "LLAMA: A Low Latency Math Library for Secure Inference," *Proc. Priv. Enhancing Technol.*, vol. 2022, pp. 274–294.

[32] S. Halevi and V. Shoup, "Algorithms in HElib," in *CRYPTO*, 2014, pp. 554–571.

[33] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private Inference on Transformers," in *NeurIPS*, 2022.

[34] X. Hou, J. Liu, J. Li, Y. Li, W. Lu, C. Hong, and K. Ren, "CipherGPT: Secure Two-Party GPT Inference," *IACR Cryptol. ePrint Arch.*, p. 1147, 2023.

[35] Z. Huang, C. Hong, C. Weng, W. Lu, and H. Qu, "More Efficient Secure Matrix Multiplication for Unbalanced Recommender Systems," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 1, pp. 551–562, 2023.

[36] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference," *USENIX Security*, 2022.

[37] (2021, Oct.) Intel HEXL (release 1.2.2).

[38] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending Oblivious Transfers Efficiently," in *CRYPTO*, 2003, pp. 145–161.

[39] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *CCS*, 2018, pp. 1209–1222.

[40] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 114–148, 2021.

[41] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *USENIX Security*, 2018, pp. 1651–1669.

[42] M. Kelkar, P. H. Le, M. Raykova, and K. Seth, "Secure Poisson Regression," in *USENIX Security*, 2022, pp. 791–808.

[43] M. Keller and K. Sun, "Effectiveness of MPC-friendly Softmax Replacement," *CoRR*, 2020.

[44] ——, "Secure quantized training for deep learning," in *ICML*, vol. 162, 2022, pp. 10 912–10 938.

[45] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," in *arXiv 2109.00984*, 2021.

[46] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure tensorflow inference," in *SP*, 2020, pp. 336–353.

[47] K. W. Lee, B. Jawade, D. D. Mohan, S. Setlur, and V. Govindaraju, "Attribute De-biased Vision Transformer (AD-ViT) for Long-Term Person Re-identification," in *AVSS*, 2022.

[48] D. Li, R. Shao, H. Wang, H. Guo, E. P. Xing, and H. Zhang, "Mpcformer: fast, performant and private transformer inference with MPC," *ICLR*, 2023.

[49] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious Neural Network Predictions via MiniONN Transformations," in *CCS*, 2017, pp. 619–631.

[50] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," *CoRR*, 2019.

[51] W. Lu, S. Kawasaki, and J. Sakuma, "Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data," in *NDSS*, 2017.

[52] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *EUROCRYPT*, vol. 6110, 2010, pp. 1–23.

[53] J. Ma, Y. Zheng, J. Feng, D. Zhao, H. Wu, W. Fang, J. Tan, C. Yu, B. Zhang, and L. Wang, "SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning," in *USENIX ATC*, 2023.

[54] P. K. Mishra, D. Rathee, D. H. Duong, and M. Yasuda, "Fast secure matrix multiplications over ring-based homomorphic encryption," *Inf. Secur. J. A Glob. Perspect.*, vol. 30, no. 4, pp. 219–234, 2021.
[55] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *USENIX Security*, 2020, pp. 2505–2522.
[56] P. Mohassel and P. Rindal, "ABY3: A Mixed Protocol Framework for Machine Learning," in *CCS*, 2018, pp. 35–52.
[57] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *SP*, 2017, pp. 19–38.
[58] Q. Pang, J. Zhu, H. Mollering, W. Zheng, and T. Schneider, "BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers," in *SP*, 2024, p. To appear.
[59] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: improved mixed-protocol secure two-party computation," in *USENIX Security*, 2021, pp. 2165–2182.
[60] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," 2019.
[61] 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 *SP*, 2022, pp. 1003–1020.
[62] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *CCS*, 2020, pp. 325–342.
[63] D. Rathee, T. Schneider, and K. K. Shukla, "Improved Multiplication Triple Generation over Rings via RLWE-Based AHE," in *CANS 2019*, 2019, pp. 347–359.
[64] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[65] (2021, Sep.) Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA.
[66] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.
[67] K. Storrier, A. Vadapalli, A. Lyons, and R. Henry, "Grotto: Screaming fast (2+1)-PC or $\mathbb{Z}_{2^n}$ via $(2,2)$-DPFs," in *CCS 2023*, pp. 2143–2157.
[68] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU," in *SP*, 2021, pp. 1021–1038.
[69] 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," *CoRR*, vol. abs/2302.13971, 2023.
[70] 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.
[71] 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*, 2019.
[72] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2022.
[73] Y. Wang, G. E. Suh, W. Xiong, B. Lefaudeux, B. Knott, M. Annavaram, and H. S. Lee, "Characterization of MPC-based Private Inference for Transformer-based Models," in *ISPASS*, 2022, pp. 187–197.
[74] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual Transformers: Token-based Image Representation and Processing for Computer Vision," 2020.
[75] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *SIGPLAN*. ACM, 2022, pp. 1–10.
[76] H. Yang, S. Shen, Z. Liu, and Y. Zhao, "cuXCMP: CUDA-Accelerated Private Comparison Based on Homomorphic Encryption," *IEEE Transactions on Information Forensics and Security*, 2023.
[77] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast Extension for Correlated OT with Small Communication," in *CCS*, 2020, pp. 1607–1626.
[78] A. C.-C. Yao, "How to Generate and Exchange Secrets," in *FOCS*, 1986, pp. 162–167.
[79] W. Zeng, M. Li, W. Xiong, W. jie Lu, J. Tan, R. Wang, and R. Huang, "MPCViT: Searching for MPC-friendly Vision Transformer with Heterogeneous Attention," *ICCV*, 2023.
[80] 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, pp. 463–476.

APPENDIX

A. Some Observations and Practices

Many private machine learning frameworks utilize a high-level front-end to translate a deep learning program (e.g,. written in PyTorch or JAX) to an Intermediate Representation (IR). By running a MPC back-end engine on each operation defined by the IR, we can obtain the private evaluation of the deep learning program. However, the development of a deep learning program without a thorough understanding of the behavior and properties of the underlying MPC primitives can significantly diminish the performance of the private evaluation, which unfortunately is a common occurrence. For instance, SPU's JAX front-end for the softmax is as belows.

```
1. def _softmax(x, axis = -1):
2.   x_max = jnp.max(x, axis,
 ↪   keepdims=True)
3.   unnormalized = jnp.exp(x - x_max)
4.   sum_exp = jnp.sum(unnormalized, axis,
 ↪   keepdims=True)
5.   result = unnormalized / sum_exp
6.   return result
```

The issue lies at Line 4 and Line 5 where perform a keep-dimension-then-division workflow. This will significantly increases the costs for the division part. The proper approach is to perform the reciprocal operation first and then broadcast the shape for multiplication. Indeed, several of recent papers on private transformer inference have claimed that the softmax function is the bottleneck [48], [73], [3]. We believe that the primary reason for this is the improper calling order, rather than the inherent complexity of the softmax function itself.

B. Threat Model and Security

We provide security against a static semi-honest probabilistic polynomial time adversary $\mathcal{A}$ following the simulation paradigm [28]. That is, a computationally bounded adversary $\mathcal{A}$ corrupts either the server S or the client C at the beginning of the protocol $\Pi_{\mathcal{F}}$ and follows the protocol specification honestly. Security is modeled by defining two interactions: a real interaction where S and C execute the protocol $\Pi_{\mathcal{F}}$ in the presence of $\mathcal{A}$ and the environment $\mathcal{E}$ and an ideal interaction where the parties send their inputs to a trusted party that computes the functionality $\mathcal{F}$ faithfully. Security requires that for every adversary $\mathcal{A}$ in the real interaction, there is an adversary Sim (called the simulator) in the ideal interaction, such that no environment $\mathcal{E}$ can distinguish between real and ideal interactions.

We recap the definition of a cryptographic inference protocol in [55]. S holds a model $\mathcal{W}$ consisting of d layers $\mathbf{W}_1, \cdots, \mathbf{W}_d$, and C holds an input vector $\mathbf{x}$.

Definition 1. *A protocol Π between S having as input model parameters $\mathcal{W} = (\mathbf{W}_1, \cdots, \mathbf{W}_d)$ and C having as an input vector $\mathbf{x}$ is a cryptographic inference protocol if it satisfies the following guarantees.*

- ***Correctness.*** *On every set of model parameters $\mathcal{W}$ that the server holds and every input vector $\mathbf{x}$ of the client,*

the output of the client at the end of the protocol is the correct prediction $\mathcal{W}(\mathbf{x})$.

- ***Privacy.*** *We require that a corrupted, semi-honest client does not learn anything about the server's network parameters $\mathcal{W}$. Formally, we require the existence of an efficient simulator algorithm $\mathcal{S}_C$ to generate $\mathcal{S}_C(\mathtt{meta}, \mathtt{out}) \approx^c \mathsf{View}_C^{\Pi}$. Here View_C^{Π} is the view of C in the execution of Π, $\mathtt{meta}$ includes the meta information (i.e., the public parameters $\mathsf{HE.pp}$, the public key pk, the number of layers, the size and type of each layer, and the activation) and $\mathtt{out}$ denotes the output of the inference. We also require that a corrupted, semi-honest S does not learn anything about the private input $\mathbf{x}$ of the client. Formally, we require the existence of an efficient simulator algorithm $\mathcal{S}_S$ to generate $\mathcal{S}_C(\mathtt{meta}) \approx^c \mathsf{View}_S^{\Pi}$ where View_S^{Π} is the view of the server in the execution of Π.*

C. From HE to Arithmetic Share $\mathcal{F}_{\mathrm{H2A}}$

We mention two different methods to convert RLWE ciphertext to arithmetic secret sharing. The first one is used for ciphertexts of SIMD-packed messages, e.g., $\mathsf{ct} = \mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\mathsf{SIMD}(\mathbf{m}))$ for $\mathbf{m} \in \mathbb{Z}_t^N$ for a prime modulus t. As mentioned in [63], we need a noise-flooding step by using a large enough random e' to statistically hide the noises in ct. Also one need to choose a proper ciphertext modulus q so that the ciphertext ct' can be correctly decrypted even applying the noise-flooding.

The second conversion is used for RLWE ciphertext of a normal polynomial, e.g., $\mathsf{ct} = \mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\hat{m})$. Let parse ct as a tuple $(\hat{b}, \hat{a}) \in R_q^2$. To properly masking ct, the sender S computes $\mathsf{ct}' := (\hat{b} + \hat{r}, \hat{a}) \boxplus \mathsf{RLWE}_{\mathsf{pk}}^{q,t}(0)$ where the random polynomial $\hat{r} \in_R R_q$. Let the decryption of ct' be $\hat{u} \in R_t$, which can be viewed as one share of $\hat{m}$. On the other hand, we have $\hat{v} := -\lfloor \frac{t \cdot \hat{r}}{q} \rceil \bmod 2^\ell$ as the other share of $\hat{m}$. However, the uniform random $\hat{r}$ may lead to an incorrect decryption result. According to the analysis in [36], it will introduce at most a 1-bit error into the arithmetic share, i.e., $\|\hat{m} - (\hat{u} + \hat{v} \bmod 2^\ell)\|_\infty \leq 1$. As mentioned in [36], 1-bit error in polynomial coefficients will lead to a significantly large error during SIMD decoding. This is the reason why we require two distinct constructions for the $\mathcal{F}_{\mathrm{H2A}}$ functionality.

We unify these two kinds of H2A in Figure 4.

Algorithm 4 From HE to Arithmetic Share $\Pi_{\mathrm{H2A}}^{\mathsf{tag}}$

NOTE: This protocol is parameterized by a tag $\mathsf{tag} \in \{\mathsf{SIMD}, \mathsf{Coeff}\}$.
Input: Sender S: $\mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\hat{m}; \hat{e}), B, B' > 0$ such that $\|\hat{e}\|_\infty < B$ and $B' \gg B$. Receiver R: the corresponding decryption key sk.
Output: $\hat{z}_0$ to S and $\hat{z}_1$ to R such that $\hat{z}_0 + \hat{z}_1 = \hat{m} \bmod t$.

1: **if** $\mathsf{tag} = \mathsf{SIMD}$ **then**
2: S samples his share $\hat{z}_0 \in_R R_t$ from the plaintext ring.
3: S constructs a public encryption of $\hat{z}_0$ but sampling the noise $\hat{e}'$ from a wider noise range $\mathbb{Z}_{B'}$.
4: S sends $\mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\hat{m}; \hat{e}) \boxplus \mathsf{RLWE}_{\mathsf{pk}}^{q,t}(-\hat{z}_0; \hat{e}')$ to R.
5: **else if** If $\mathsf{tag} = \mathsf{Coeff}$ **then**
6: S samples $\hat{r} \in_R R_q$ from the ciphertext ring R_q.
7: S constructs a fresh encryption of 0, i.e., $\mathsf{RLWE}_{\mathsf{pk}}^{q,t}(0)$.
8: S sends $\mathsf{RLWE}_{\mathsf{pk}}^{q,t}(\hat{m}; \hat{e}) \boxplus \mathsf{RLWE}_{\mathsf{pk}}^{q,t}(0) \boxplus (\hat{r}, 0)$ to R.
9: S outputs $-\lfloor \frac{t \cdot \hat{r}}{q} \rceil \bmod t$ as $\hat{z}_0$.
10: **end if**
11: R decrypts the received ciphertext and outputs the result as $\hat{z}_1$.

Proof of Theorem 1. The correctness of Theorem 1 is directly derived from Proposition 1. We now show the the privacy part. Particularly we instantiate $\mathcal{F}_{\mathrm{H2A}}$ as $\Pi_{\mathrm{H2A}}^{\mathsf{Coeff}}$.
(**Corrupted Receiver.**) Receiver's view of $\mathsf{View}_R^{\Pi_{\mathrm{OLT}}}$ consists of RLWE ciphertexts $\{\mathsf{ct}'_{\alpha,\beta}\}$ for $\alpha \in [k']$ and $\beta \in [m']$. The simulator $\mathcal{S}_R^{\Pi_{\mathrm{OLT}}}$ for this view can be constructed as follows.

1) Given the access to meta, $\mathcal{S}_R^{\Pi_{\mathrm{OLT}}}$ outputs the ciphertexts $\{\tilde{\mathsf{ct}}_{\alpha,\beta} := \mathsf{RLWE}_{\mathsf{pk}}^{q,2^\ell}(0)\}$ to R.

The security against a corrupted R is directly reduced to the semantic security of the underlying RLWE encryption. Thus we have $\mathsf{View}_R^{\Pi_{\mathrm{OLT}}} \approx^c \mathcal{S}_R^{\Pi_{\mathrm{OLT}}}(\mathsf{meta})$.

(**Corrupted Sender.**) Sender's view of $\mathsf{View}_S^{\Pi_{\mathrm{OLT}}}$ consists of an array of RLWE ciphertexts $\mathbf{c}$ (that encrypted under S's key), and the decryption of these ciphertexts in the execution in the H2A protocol $\Pi_{\mathrm{H2A}}^{\mathsf{Coeff}}$. The important part is to show that the Sender (i.e., decrypt key holder) can not gain extra information from the ciphertext noise. The simulator $\mathcal{S}_S^{\Pi_{\mathrm{OLT}}}$ for this view can be constructed as follows. Also, instead of calling a simulator for $\mathcal{F}_{\mathrm{H2A}}$, we directly simulate the corresponding view in the $\Pi_{\mathrm{H2A}}^{\mathsf{Coeff}}$ protocol.

1) On receiving the RLWE ciphertexts $\{\mathsf{ct}'\}$ from S and given the access to meta, $\mathcal{S}_S^{\Pi_{\mathrm{OLT}}}$ samples uniform random polynomial $\hat{r}_i \in_R R_q$ and computes $\mathsf{ct}_i := \mathsf{RLWE}_{\mathsf{pk}}^{q,2^\ell}(0) \boxplus (\hat{r}_i, 0)$ for $i \in [\tilde{m}]$.
2) $\mathcal{S}_S^{\Pi_{\mathrm{OLT}}}$ computes the rounding $\hat{r}'_i := \lfloor 2^\ell \cdot \hat{r}_i / q \rceil \bmod 2^\ell$ for $i \in [\tilde{m}]$.
3) $\mathcal{S}_S^{\Pi_{\mathrm{OLT}}}$ outputs a matrix $\tilde{\mathbf{U}} := \mathsf{ParseMat}(\hat{r}'_0, \hat{r}'_1, \cdots, \hat{r}'_{\tilde{m}-1}, \mathsf{meta})$ using the parsing procedure in Figure 8.

Indeed, Step (1) and Step (2) basically simulate the Step 6 – Step 9 of $\Pi_{\mathrm{H2A}}^{\mathsf{Coeff}}$. The RLWE ciphertexts $\mathbf{c}[i] \approx^c \mathsf{ct}_i$ due to the informatics masking $\hat{r}_i$ and the semantic security. That is, the ciphertext noise in the real execution is informatically masked by the random $\hat{r}$ (Step 6 of Figure 4) which is indistinguishable from the uniform random $\hat{r}_i$ sampled by the simulator. Also, the values in the output matrix $[\![\mathbf{U}]\!]_l$ of S in Π_{OLT} distribute uniformly in $\mathbb{Z}_{2^\ell}$ which is exact the same distribution $\mathcal{S}_S^{\Pi_{\mathrm{OLT}}}$ creates $\tilde{\mathbf{U}}$. Thus we have $\mathsf{View}_S^{\Pi_{\mathrm{OLT}}} \approx^c \mathcal{S}_S^{\Pi_{\mathrm{OLT}}}(\mathsf{meta}, \mathsf{out})$. □

The security proofs for Π_{bOLEe} (Theorem 2) can be given in a similar manner while the simulator $\mathcal{S}_S^{\Pi_{\mathrm{bOLEe}}}$ invoke a different $\Pi_{\mathrm{H2A}}^{\mathsf{SIMD}}$ function for SIMD-packed messages. Also, we argue that the security proofs for the GeLU protocol and the softmax protocol simply follow in the hybrid model since only OT messages and properly masked HE ciphertexts are exchanged.

D. Activation Functions

We use the following coefficients of (2) for the GeLU approximation.

$$a_0 = -0.5054031199708174, \quad a_1 = -0.4222658115198386$$
$$a_2 = -0.1180761295118195, \quad a_3 = -0.0110341340306157$$
$$b_1 = 0.5$$
$$b_0 = 0.0085263215410380, \quad b_2 = 0.3603292692789629$$
$$b_4 = -0.037688200365904, \quad b_6 = 0.0018067462606141.$$

We recap the polynomial $\texttt{GeLU}(x) \approx P^4(x) = 0.5x + \sum_{i=0}^{i=4} c_i|x|^i$ used in [58] for the GeLU approximation in the narrow range $x \in [-2.7, 2.7]$. That is

$$c_0 = 0.001620808531841547, c_1 = -0.03798164612714154$$
$$c_2 = 0.5410550166368381, c_3 = -0.18352506127082727$$
$$c_4 = 0.020848611754127593.$$

This polynomial is constructed using a symmetric property of the GeLU function.

Using the method described in §V-A, we can approximate the SiLU function $\mathsf{SiLU}(x) = \dfrac{x}{1+\exp(-x)}$ as using the proposed exponential protocol on the first branch.

$$\mathsf{SiLU}(x) \approx \begin{cases} -\epsilon & x < -8 \\ a_2x^2 + a_1x + a_0 & -8 < x \le -4 \\ b_6x^6 + b_4x^4 + b_2x^2 + b_1x + b_0 & -4 < x \le 4 \\ x - \epsilon & x > 4 \end{cases}$$

where the coefficients

$$a_0 = -0.3067541139982155 \quad a_1 = -0.0819767021525476$$
$$a_2 = -0.0055465625580307 \quad b_0 = 0.0085064025895951$$
$$b_1 = 0.5 \quad b_2 = 0.2281430841728270$$
$$b_4 = -0.011113046708173 \quad b_6 = 0.0002743776353465$$

are fitted using `numpy.polyfit` API.

Also we can approximate the ELU function for $\alpha > 0$

$$\mathsf{ELU}(x) = \begin{cases} \alpha \cdot (\exp(x) - 1) & x < 0 \\ x & x \ge 0 \end{cases}$$

E. More Details on Implementations

1) Parsing Interleaved Polynomials: The procedure in Figure 6 merges multiple (encrypted) polynomials into one polynomial in an interleaving manner. Indeed, we need a way to parse back the packed polynomial to obtain the final matrix multiplication. We assume the (encrypted) polynomials are arranged in a row-major manner, i.e., the ciphertext array $\mathbf{c}$ in Step 3 of Figure 6. In Figure 8, we present the pseudocode that parses an array of polynomials into a matrix, given the meta information.

```
def parse_mat(polys: Array<Poly>, dim3, win3):
  # matrix shape
  (k, m, n) = dim3
  # partition windows
  (k_w, m_w, n_w) = win3
  # number of blocks along each axis
  k_prime = ceil(k / k_w)
  m_prime = ceil(m / m_w)
  n_prime = ceil(n / n_w)
  # expected polys to parse
  m_tilde = ceil(k_prime * n_prime / m_w)
  assert(len(polys) == m_tilde)
  out = Matrix(k, n) # initialize
  # each poly are packed from `m_w` polys
  for i in range(k_prime * n_prime, step=m_w):
    pidx = i // m_w
    for j in range(m_w):
      # we assume row-major packing
      row_blk, col_blk = j // mprime, j % mprime
      rbgn = row_blk * k_w
      rend = min(rbgn + k_w, k) # min on margin cases
      cbgn = col_blk * n_w
      cend = min(cbgn + n_w, n) # min on margin cases
      for r in range(rend - rbgn):
        for c in range(cend - cbgn):
          # flatten index of this (r, c) entry
          fidx = r * n_w + c
          # `+ pidx` due to some packing
          # might across two mult.
          cidx = fidx * m_w + (j + pidx) % m_w
          out[rbgn + r, cbgn + c] = polys[pidx][cidx]
  return out
```

Figure 8: ParseMat Parse the packed polynomials as matrix

TABLE VI: Performance breakdown on ViT-base. The input is one 224×224 RGB image. The LAN setting was used. The mixed ring evaluating was used for GeLU with a smaller ring $\ell' = 32$, while Exp was computed over modulus 2^{64}.

Operation	ViT-base ($B = 12$, $D = 768$, $H = 12$) #Calls	Time (sec)	Sent (MB)	Recv (MB)
`f_mmul`	73	90.91	440.83	584.38
`f_tensordot`	1	6.57	38.68	30.52
`f_batch_mmul`	24	23.81	393.50	373.20
`multiplixer`	120	1.15	43.10	43.09
`f_less`	145	16.18	375.33	79.15
`f_exp`	12	36.18	1870.28	1537.38
`f_reciprocal`	12	3.71	38.98	40.07
`f_mul`	174	29.73	1574.58	1571.98
`f_rsqrt`	25	0.77	5.28	5.13
`f_gelu`	12	29.85	1676.25	1117.00
	Total	3.99min	6.30GB	5.26GB

2) More Concrete Parameters: For the matrix multiplication, we use the SEAL parameters $\mathsf{HE.pp}_{\mathrm{OLT}} = \{N = 8192, q \approx 2^{147}, t = 2^{64}, q' = 2^{49}\}$ where q' is needed for the homomorphic automorphism. For the point-wise multiplication, we use a footnotesizeer SEAL parameters $\mathsf{HE.pp}_{\mathrm{bOLE}} = \{N = 8192, q \approx 2^{148}, t \approx 2^{45\times 3}\}$. Indeed, we decomposite the plaintext modulus t into 3 co-primes where each of them is about 45 bits. This is a commonly used technique when doing the SIMD encoding with a large plaintext modulus [51], [63]. Also, we sample $B' = 2^{98}$ bits randomness for the noise flooding used in $\Pi^{\mathsf{SIMD}}_{\mathrm{H2A}}$. For the concrete MSB protocol [62],

TABLE VII: Compared with PackLWEs. Fixing $m_{\mathrm{w}} = 64$ for InterLeave. 4T means 4 threads were used.

Matrix Shape	(16, 768, 768)	(256, 768, 768)	(128, 768, 3072)
PackLWEs	4901.00ms	74.85s	150.35s
InterLeave	32.61ms	483.75ms	851.13ms
(4T) Speedup	150×	154×	176×
PackLWEs	1714.42ms	21.01s	38.87s
InterLeave	33.51ms	392.72ms	808.68ms
(16T) Speedup	51×	53×	48×

Algorithm 5 Ring Extension Protocol [61] with the positive heuristic [16]

Input: $x'_0, x'_1 \in \mathbb{Z}_{2^{\ell'}}$ where $x'_0 + x'_1 \equiv_{\ell'} \lfloor \tilde{x} \cdot 2^f \rfloor$ for some $\tilde{x} \in \mathbb{R}$.

Output: $x_0, x_1 \in \mathbb{Z}_{2^\ell}$ such that $x_0 + x_1 \equiv_\ell \lfloor \tilde{x} \cdot 2^f \rfloor$.

1: Update $x'_1 \leftarrow x'_1 + M$ for some heuristic $M > 0$.
2: Compute $[\![w := \mathbf{1}\{x'_0 + x'_1 > 2^{\ell'-1}\}]\!] \in \mathbb{Z}_{2^\ell}$.
3: Output $x_l \equiv_\ell x'_l - 2^{\ell'} \cdot ([\![w]\!]_l \bmod 2^{\ell-\ell'}) - l \cdot M$ for $l \in \{0, 1\}$.

we set $M = 16$ for the $\binom{M}{1}$-$\mathtt{OT}_2$ and $\binom{M}{1}$-$\mathtt{OT}_6$.

3) More Experiment Results: In Table VI, we present the running details of BumbleBee on a Vision Transformer. We can see that the activation part in the ViT model need more optimizations.

In Table VII, we compare the performance of InterLeave with PackLWEs. We observe that the PackLWEs procedure gains a more significant performance improvement from multi-threading compared to the InterLeave procedure. In practice, we can further enhance parallelism at the InterLeave level. This means that for large-scale matrix multiplication tasks, we may require multiple InterLeave calls. For instance, to fully leverage the capabilities of a 32-core CPU, we could launch 8 threads for each InterLeave instance and execute 4 InterLeave calls simultaneously.