BumbleBee: Secure Two-party Inference Framework for Large Transformers

Wen-jie Lu[†], Zhicong Huang[†], Zhen Gu[‡], Jingyu Li[§†], Jian Liu[§], Kui Ren[§], Cheng Hong[†], Tao Wei[†], WenGuang Chen[†]
[†]*Ant Group*
[‡]*Alibaba Group*
[§]*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: Firstly, we present optimized homomorphic encryption-based protocols that enable the multiplication of large matrices with $80 - 90\%$ less communication cost than existing methods. Secondly, we offer a general method for designing efficient and accurate protocols for non-linear activation functions in transformers. Our activation protocols have demonstrated speed and reduced the communication overhead by $80 - 95\%$ over two existing methods. Finally, we conducted intensive benchmarks on several large transformer models. Results show that BumbleBee is more than one order of magnitude faster than Iron (NeurIPS22).**

1. Introduction

Large transformer-based models such as BERT [20, 46], GPT [54] and ViT [21] have realized state-of-the-art (SOTA) performance on lots of real-world tasks including person re-identification [43], voice assistant [15] and code auto-completion [68]. 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) [7, 23] to enable private inference on deep learning models such as convolution neural networks (CNN) [2, 29, 35, 56] and transformer-based models [27, 44, 66, 71]. While Secure Two-party (2PC) CNN inference could already be done in a few minutes, the private inference on transformer-based models introduce new efficiency challenges, particularly from the perspective of communication overhead. For example, [27] reports about 90 GB communication for one private inference on a 12-layer BERT model. Moreover [66, Table 1] reports one private inference on a 12-layer Vi might need to exchange about 262 GB messages. Thus, a high-speed bandwidth is indispensable for these communication-intensive approaches. Indeed [66] used a 100Gbps Ethernet cable in their experiments.

We can summarize two major challenges in developing a computational fast and communication-friendly 2PC framework for private inference on 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 Cains. 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) [19, 52, 53] or Homomorphic Encryption (HE) [11, 28, 29, 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, 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 [10, 39, 55, 56], 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 [16] requires evaluating about 3.9×10^6 point-wise GeLUs.
Our second challenge is to design efficient 2PC protocols for these complex activation functions.

1.1. Our Technical Results

Efficient protocols for two types of oblivious linear evaluation (OLE). The multiplication of two private matrices can be achieved using a primitive called Matrix Oblivious Linear Evaluation (mOLE). We propose a computationally fast and communication-friendly mOLE protocol using a lattice-based HE. The coefficient encoding approaches of [27, 29] provide a good starting point. However, the major drawback of these works is the significant communication overhead due to a sparse format of the resulting ciphertexts. To overcome this shortage, we present a local procedure to compress many of these "sparse" ciphertexts into a "dense" form using the property of the underlying HE. Compared to a previous ciphertext compression method [12], our compression procedure is about 20× faster. Overall, we observed 80 – 90% less communication costs over [27, 29] which leads to a shorter end-to-end time on a low bandwidth.

Beside the matrix multiplications, the point-wise multiplication $x_0 \cdot y_0, x_1 \cdot y_1, \cdots, x_n \cdot y_n$ is also an important computation in the transformer inference. A batch of point-wise multiplications can be privately achieved via a batch OLE (bOLE) primitive [18]. The bOLE protocol takes two private vectors as input and produces the point-wise results with a suitable random masking. In this work, we enhance the HE-based protocol [57] by leveraging a plaintext lifting technique. This eliminates the need for statistical random masking, enabling us to instantiate the bOLE protocol with a smaller HE parameter. Our empirical results demonstrate that our approach is $1.3\times$ more efficient than [57] in terms of computation time and communication.

A framework for constructing efficient and accurate protocols for the activation functions in transformers. We first propose a general framework for construct 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 round 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 GeLU with two polynomials $P(x)$ and $Q(x)$ of low degree that minimize the least square error, as follows.

$$\texttt{GeLU}(x) \approx \begin{cases} -10^{-5} & x < -5 \\ P(x) & -5 < x \leq -2 \\ Q(x) & -2 < x \leq 3 \\ x - 10^{-5} & x > 3 \end{cases}$$

Then we propose optimizations based on three key insights. Firstly, we exploit the smoothness of the activation function to enhance efficiency. The smooth function give us some "error" tolerance for selecting a wrong but still nearby segment. For example, the evaluation of an input $x = -1.98$ on the (wrong) 2nd segment can also give a similar result, i.e., $P(-1.98) \approx Q(-1.98)$. This give us some room to design a more efficient method at the cost of introducing a mild error. Secondly, our method reduces the number of segments required for approximation by employing multiple low-degree polynomials. For example, while [45] requires over 12 splines to approximate an activation function, our approach only requires 3 – 4 segments. Finally, we introduce optimizations to improve the amortized efficiency when evaluating multiple polynomials over the same input point. For instance, our optimized 2PC protocol can evaluate one million points of GeLU within 10 seconds, exchanging about 700MB of messages. That is 93% less time and 95% less communication than the existing numerical approach [55].

Our protocols for the activation function may not be as precise as the sophistic approaches such as [37, 55]. However, we want to emphasize that the approximation error in our protocol does not negatively impact transformer accuracy. We have performed private inference on two large transformers using four public datasets. We have run private inference on more than 3000 samples. The results demonstrate that our private inference achieves the same level of accuracy as the inference on cleartext even without model fine-tuning.

1.2. Contributions

To summarize, we make four key contributions:

1) We have designed a matrix multiplication protocol that is optimized for communication efficiency. In fact, our protocol has been shown to require 80 – 99.9% less communication compared to existing methods based on HE and OT. Also, our protocol is speedy. For instance, the private matrix multiplication in the dimensions $128 \times 768 \times 768$ can be done within 1.0 second.
2) We provide methods to design efficient and accurate protocols for the activation functions used in transformers. Unlike prior methods [27, 44, 52] that rely on rough substitutions, our approach does not require changing any components in the transformer model. Therefore, we do not need any additional model fine-tuning when using our private activation protocols.
3) We have implemented all the proposed protocols and developed a new MPC back-end called `BumbleBee` into the `SPU` library [48]. To compare, we also implemented

a baseline using several SOTA 2PC protocols based on the `SPU` library. This baseline is already more efficient than the 2PC approach `Iron` (NeurIPS22) [27], particularly in terms of communication. 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 60% over our baseline, and by 83% over `Iron`.

4) `BumbleBee` enables easy-to-use private transformer inference[1]. Indeed, we have successfully run `BumbleBee` on 5 pre-trained transformer models utilizing the model weights and the Python programs available on HuggingFace's website[2] including BERT-base, BERT-large, GPT2-base, LLaMA-7B, and ViT-base. We also evaluated the accuracy of `BumbleBee` across four datasets, including a 1000-size subset from ImageNet [58]. Importantly, all our experiments were conducted using the proposed protocols, rather than through simulation. Our approach provides a promising evidence that accurate private transformer inference is possible.

1.3. 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. We present examples from two frameworks from the industry, i.e., `CrypTen` [40] and `SPU` [48]. The first example is `SPU`'s JAX front-end for the softmax function.

```
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 [3, 44, 66]. However, we believe that the primary reason for this is the improper calling order, rather than the inherent complexity of the softmax function itself.

1. Our implementation is available in https://github.com/secretflow/spu.
2. https://github.com/huggingface/transformers

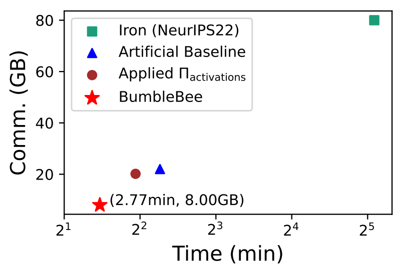


Figure 1: The overall bandwidth improvements of the proposed optimizations on the BERT-base model with 128 input tokens. The baseline consists of many SOTA 2PC protocols that are already communication-friendly.

A second example can be found in the LayerNorm operation in transformers which involves the evaluation of the divide-square-root $x/\sqrt{y}$ formula. One possible approach to evaluating this formula is to sequentially execute the 2PC protocols of square root, reciprocal, and multiplication, as is done in the `CrypTen` [40]. However, in the MPC side, the costs of computing the reciprocal of square root $y^{-1/2}$ is similar to that of the square-root operation using the methods from [33, 39]. Thus the former should be used to save the computation of reciprocal. When implementing `BumbleBee`, we prioritize the use of just-right protocols and ensure that they are employed in a right order to minimize the overhead.

2. Preliminaries

2.1. 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 $\mathbb{A}_{N,q}$ to denote the set of integer polynomials $\mathbb{A}_{N,q} = \mathbb{Z}_q[X]/(X^N + 1)$. 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 $\mathbf{a} \odot \mathbf{b}$.

2.2. Cryptographic Primitives

2.2.1. Additive Secret Sharing. We use 2-out-of-2 additive secret sharing schemes over the ring $\mathbb{Z}_{2^\ell}$ throughout this manuscript. An ℓ-bit ($\ell \geq 2$) value x is additively shared as $\llbracket x \rrbracket_0$ and $\llbracket x \rrbracket_1$ where $\llbracket x \rrbracket_l$ is a random share of x held by P_l. We can write the multiplication of two shared value as $\llbracket x \rrbracket \cdot \llbracket y \rrbracket \equiv_\ell (\llbracket x \rrbracket_0 + \llbracket x \rrbracket_1) \cdot (\llbracket y \rrbracket_0 + \llbracket y \rrbracket_1) \equiv_\ell \llbracket x \rrbracket_0 \llbracket y \rrbracket_0 + \llbracket x \rrbracket_1 \llbracket y \rrbracket_1 + \llbracket x \rrbracket_0 \llbracket y \rrbracket_1 + \llbracket x \rrbracket_1 \llbracket y \rrbracket_0$ where the mixed-terms $\llbracket x \rrbracket_0 \llbracket y \rrbracket_1$ and $\llbracket x \rrbracket_1 \llbracket y \rrbracket_0$ 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 $\llbracket \cdot ; f \rrbracket$ to explicitly denote the shared value is using f-bit fixed-point precision. When $\ell = 1$, we use $\llbracket z \rrbracket^B$ to denote Boolean shares. Also we omit the subscript and only write $\llbracket x \rrbracket$ or $\llbracket z \rrbracket^B$ when the ownership is irrelevant from the context.

2.2.2. Oblivious Transfer. We rely on oblivious transfer (OT) for the non-linear computation. In a general 1-out-of-2 OT $\binom{2}{1}$-$\texttt{OT}_\ell$, 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 [5]. 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 [70] for a lower communication COT.

2.2.3. 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) [47]. 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 $\mathsf{sk} \in \mathbb{A}_{N,q}$ and the public key $\mathsf{pk} \in \mathbb{A}_{N,q}^2$.
- **Encryption.** An RLWE ciphertext is given as a polynomial tuple $(\hat{b}, \hat{a}) \in \mathbb{A}_{N,q}^2$. We write $\mathsf{RLWE}_{\mathsf{pk}}^{N,q,t}(\hat{m})$ to denote the encryption of $\hat{m} \in \mathbb{A}_{N,t}$ 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 $\hat{m}_0, \hat{m}_1 \in \mathbb{A}_{N,t}$ under a same key, the operation $\mathsf{ct}_0 \boxplus \mathsf{ct}_1$ computes the RLWE tuple $(\hat{b}_0 + \hat{b}_1, \hat{a}_0 + \hat{a}_1) \in \mathbb{A}_{N,q}^2$ which can be decrypted to $\hat{m}_0 + \hat{m}_1 \bmod \mathbb{A}_{N,t}$.
- **Multiplication** ($\boxtimes$)**.** Given an RLWE ciphertext $\mathsf{ct} = (\hat{b}, \hat{a})$ that encrypts $\hat{m} \in \mathbb{A}_{N,t}$, and a plain polynomial $\hat{c} \in \mathbb{A}_{N,t}$, the operation $\mathsf{ct} \boxtimes \hat{c}$ computes the tuple $(\hat{b} \cdot \hat{c}, \hat{a} \cdot \hat{c}) \in \mathbb{A}_{N,q}^2$ which can be decrypted to $\hat{m} \cdot \hat{c} \bmod \mathbb{A}_{N,t}$.
- **SIMD Encoding.** By choosing a prime t such that $t \equiv 1 \bmod 2N$, the SIMD technique [60] allows to convert vectors $\mathbf{v}, \mathbf{u} \in \mathbb{Z}_t^N$ of N elements to polynomials $\hat{v}, \hat{u} \in \mathbb{A}_{N,t}$. 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 using RLWE-based HEs, 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 $\mathsf{ct} \in \mathbb{A}_{N,q}^2$ that encrypts a polynomial $\hat{m}(X) \in \mathbb{A}_{N,t}$, and an odd integer $g \in [1, 2N)$, the operation $\mathsf{ct}' := \mathsf{Auto}(\mathsf{ct}, g)$ computes $\mathsf{ct}' \in \mathbb{A}_{N,q}^2$ which decrypts to $\hat{m}'(X) = \hat{m}(X^g) \in \mathbb{A}_{N,t}$. In this work, we use the automorphism to compress the volume of RLWE ciphertexts.

TABLE 1: The proposed protocols in `BumbleBee`. Also we use some existing protocols as the building blocks.

Proposed Protocols	Descriptions
$\llbracket \mathbf{z} \rrbracket \leftarrow \Pi_{\mathrm{bOLEe}}(\mathbf{x}, \mathbf{y})$ such that $\mathbf{z} \equiv_\ell \mathbf{x} \odot \mathbf{y} + \mathbf{e}$ for $\lVert \mathbf{e} \rVert_\infty \leq 1$	Batch OLE with Error (§4)
$\llbracket \mathbf{x} \odot \mathbf{y} \rrbracket \leftarrow \Pi_{\mathrm{mul}}(\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket)$	Point-wise mult.
$\llbracket \mathbf{x}^2 \rrbracket \leftarrow \Pi_{\mathrm{square}}(\llbracket \mathbf{x} \rrbracket)$	Square
$\llbracket \mathbf{Z} \rrbracket \leftarrow \Pi_{\mathrm{mOLE}}(\mathbf{X}, \mathbf{Y})$ such that $\mathbf{Z} \equiv_\ell \mathbf{X} \cdot \mathbf{Y}$	Matrix OLE (§3)
$\llbracket \mathbf{X} \cdot \mathbf{Y} \rrbracket \leftarrow \Pi_{\mathrm{matmul}}(\llbracket \mathbf{X} \rrbracket, \llbracket \mathbf{Y} \rrbracket)$	Matrix mult.
$\llbracket \tilde{y}; f \rrbracket \leftarrow \Pi_{\mathrm{GeLU}}(\llbracket \tilde{x}; f \rrbracket)$	GeLU (§5.1)
$\llbracket \tilde{\mathbf{y}}; f \rrbracket \leftarrow \Pi_{\mathrm{softmax}}(\llbracket \tilde{\mathbf{x}}; f \rrbracket)$	Softmax (§5.2)
Ideal Functionalities	**Descriptions**
$\llbracket \tilde{x}; f \rrbracket \leftarrow \mathcal{F}_{\mathrm{trunc}}^{f}(\llbracket \tilde{x}; 2f \rrbracket)$	Truncation [17, 29]
$\llbracket c?x : y \rrbracket \leftarrow \mathcal{F}_{\mathrm{mux}}(\llbracket c \rrbracket^B, \llbracket x \rrbracket, \llbracket y \rrbracket)$	Multiplexer
$\llbracket 1/\sqrt{\tilde{x}}; f \rrbracket \leftarrow \mathcal{F}_{\mathrm{rsqrt}}(\llbracket \tilde{x}; f \rrbracket; f)$	Reciprocal Sqrt [39]
$\llbracket \mathbf{1}\{x < y\} \rrbracket^B \leftarrow \mathcal{F}_{\mathrm{lt}}(\llbracket x \rrbracket, \llbracket y \rrbracket)$	Less-then [56]
$\llbracket \mathbf{x} \rrbracket \leftarrow \mathcal{F}_{\mathrm{H2A}}(\mathsf{RLWE}(\mathbf{x}))$	HE to share [29, 57]

From HE to Arithmetic Share. The RLWE ciphertexts have a noise associated with them which would leak information about the homomorphic computation. As a result, we need a mechanism to hide the noise in the RLWE ciphertexts particularly when we need to convert an encrypted message to arithmetic shares. We abstract this mechanism as a $\mathcal{F}_{\mathrm{H2A}}$ functionality which takes as input an RLWE ciphertext of $\hat{m}$ and outputs the arithmetic share $\llbracket \hat{m} \rrbracket$ to each party. Concretely, we can adopt the noise flooding [57] and the 1-bit approximated resharing [29] to implement this functionality.

2.3. Transformer-based Models

Many modern language pre-processors such as GPT [16] and BERT [20] are Transformer-based. These models consist of an input embedding layer followed by multiple layers of Transformers [63]. Similarly, Vision Transformers (ViTs) [4, 21] 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 computing $\mathsf{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \texttt{softmax}(\mathbf{Q}\mathbf{K}^\top + \mathbf{M})\mathbf{V}$, 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 vari-

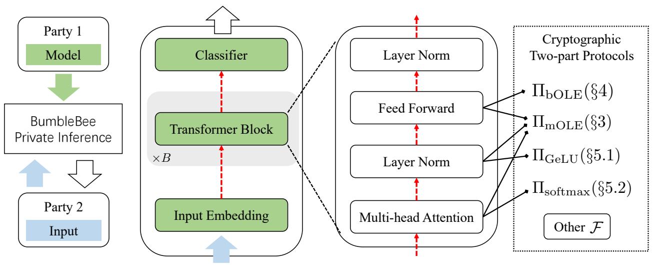

Party 1
Model
BumbleBee Private Inference
Party 2
Input
Classifier
Transformer Block
$\times B$
Input Embedding
Layer Norm
Feed Forward
Layer Norm
Multi-head Attention
Cryptographic Two-part Protocols
$\Pi_{\mathrm{bOLE}}(\S4)$
$\Pi_{\mathrm{mOLE}}(\S3)$
$\Pi_{\mathrm{GeLU}}(\S5.1)$
$\Pi_{\mathrm{softmax}}(\S5.2)$
Other $\mathcal{F}$

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

ant 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 resulting 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.

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

2.4. Threat Model and Private 2PC Inference

Similar to previous work [3, 29, 35, 50, 56], we target privacy against a static and semi-honest probabilistic polynomial time (PPT) adversary following the ideal/real world paradigm [9]. 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 summarized in Table 1 . 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 [22], 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.

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 $\hat{q} := \pi_{\mathrm{lhs}}(\mathbf{Q})$ and $\hat{v} := \pi_{\mathrm{rhs}}(\mathbf{V})$.

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

Figure 3: Example for π_{lhs} and π_{rhs} with $N = 16$ and $\ell = 5$.

3. Fast and Communication-Light 2PC Protocol for Large Matrix Multiplication

Transformer-based models rely heavily on multiplication of high-dimensional matrices. For the private transformer inference, we introduce a primitive called Matrix OLE (mOLE). We describe the mOLE as a 2-party protocol that takes two private (might be random) matrices $\mathbf{X}$ and $\mathbf{Y}$ from the two parties, respectively, and generates the share $[\![\mathbf{XY}]\!]$ between them. We compute the multiplication of two shared matrices $[\![\mathbf{Q}]\!][\![\mathbf{V}]\!]$ by computing the mixed terms $[\![\mathbf{Q}_0\mathbf{V}_1]\!]$ and $[\![\mathbf{Q}_1\mathbf{V}_0]\!]$ which thus needs two mOLE executions.

3.1. Existing HE-based Approaches

Our starting point is the RLWE-based matrix multiplication from [49]. We denote this method as `KRDY`. Specifically, they use two encoding functions π_{lhs} and π_{rhs} to achieve efficient private matrix multiplication protocol $\pi_{\mathrm{lhs}} : \mathbb{Z}_{2^\ell}^{k_{\mathrm{w}} \times m_{\mathrm{w}}} \mapsto \mathbb{A}_{N,2^\ell}$ and $\pi_{\mathrm{rhs}} : \mathbb{Z}_{2^\ell}^{m_{\mathrm{w}} \times n_{\mathrm{w}}} \mapsto \mathbb{A}_{N,2^\ell}$.

$$\hat{q} := \pi_{\mathrm{lhs}}(\mathbf{Q}) \text{ and } \hat{v} := \pi_{\mathrm{rhs}}(\mathbf{V}) \text{ such that} \tag{1}$$
$$\hat{q}[0] = \mathbf{Q}[i,j] \text{ for } i = 0 \land j = 0,$$
$$\hat{q}[N + i \cdot k_{\mathrm{w}} \cdot m_{\mathrm{w}} - j] = -\mathbf{Q}[i,j], \text{ for } i = 0 \land j \in [m_{\mathrm{w}}]/\{0\}$$
$$\hat{q}[i \cdot k_{\mathrm{w}} \cdot m_{\mathrm{w}} - j] = \mathbf{Q}[i,j] \text{ for } i > 0 \land j \in [m_{\mathrm{w}}]$$
$$\hat{v}[k \cdot m_{\mathrm{w}} + j] = \mathbf{V}[j,k] \text{ for } j \in [m_{\mathrm{w}}], k \in [n_{\mathrm{w}}].$$

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.* Figure 3 provides a simple illustration of the ideas behind π_{lhs} and π_{rhs} encodings. Indeed, we made slight modifications to the encodings for our purpose. In particular, the first row of $\mathbf{Q}$ are encoded in different positions. More information is provided in the Appendix.

Proposition 1 (Modified from [49])**.** *Assuming $1 \le k_{\mathrm{w}} \cdot m_{\mathrm{w}} \cdot n_{\mathrm{w}} \le N$. Given two polynomials $\hat{q} = \pi_{\mathrm{lhs}}(\mathbf{Q}), \hat{v} = \pi_{\mathrm{rhs}}(\mathbf{V}) \in \mathbb{A}_{N,2^\ell}$, 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 $\mathbb{A}_{N,2^\ell}$. That is $\mathbf{U}[i,k] = \hat{u}[i \cdot m_{\mathrm{w}} \cdot n_{\mathrm{w}} + k \cdot m_{\mathrm{w}}]$ for all $i \in [k_{\mathrm{w}}], k \in [n_{\mathrm{w}}]$.*

```
Input: $\hat{a} \in \mathbb{A}_{N,q}$ for an odd $q$ and $1 \le 2^r \le N$.
Output: $\hat{b} \in \mathbb{A}_{N,q}$ such that $\hat{b}[i]$ is zero if $0 \not\equiv i \bmod 2^r$.
Also $\hat{b}[i] = \hat{a}[i] \bmod q$ for all $0 \equiv i \bmod 2^r$.
1: Compute $\hat{a}_0 := h \cdot \hat{a} \bmod q$ for $h \equiv 2^{-r} \bmod q$.
2: for $j \in [r]$ do
3:     $\hat{a}_{j+1} := \hat{a}_j + \mathsf{Auto}(\hat{a}_j, \frac{N}{2^j} + 1)$.
4: end for
5: return $\hat{a}_r$.
```

Figure 4: `ZeroGap`: Zero-out a given gap of polynomial.

```
Input: $\{\hat{a}_j \in \mathbb{A}_{N,q}\}_{j\in[2^r]}$ for an odd $q$ and $1 \le 2^r \le N$.
Output: $\hat{c} \in \mathbb{A}_{N,q}$ such that $\hat{c}[i] = \hat{a}_{i \bmod 2^r}[\lfloor i/2^r \rfloor \cdot 2^r]$.
1: for $\forall j \in [2^r]$ in parallel do
2:     $\hat{b}_j := \mathtt{ZeroGap}(\hat{a}_j, 2^r)$   ▷ zero-out the $2^r$ gap
3:     $\hat{b}_j := \hat{b}_j \cdot X^j \in \mathbb{A}_{N,q}$   ▷ right-shift by $j$ unit
4: end for
5: return the sum of polynomials $\sum_{j=0}^{2^r-1} \hat{b}_j$.
```

Figure 5: `IntrLeave`: Coefficients interleaving.

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, [49] needs to exchange $O(\min(\frac{km}{k_w m_w}, \frac{mn}{m_w n_w}) + \frac{kn}{k_w n_w})$ ciphertexts. To provide an example, let consider a set of dimensions commonly used in transformer models such as $k = 16, m = 768$, and $n = 3072$. In this scenario, the Iron protocol requires 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.

3.2. Less Communication via Ciphertext Packing

According to Proposition 1, only $k_w n_w$ coefficients of the resulting polynomial $\hat{u}$ are necessary out of the N coefficients. However, even if $k_w n_w \ll N$, we still need to transmit the entire RLWE ciphertext of $\hat{u}$ for decryption. This leads to communication inefficiency for large matrices. To alleviate this issue, we propose applying a PackLWEs procedure from [12] 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 resulting ciphertexts from $O(kn/(k_w n_w))$ to $O(kn/N)$ ciphertexts at the costs of $O(kn)$ homomorphic automorphism operations. Let take 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. It should be noted that for matrices with large kn, the compression step in KRDY$^+$ can become much more computationally expensive than the homomorphic multiplication step. Overall, KRDY$^+$ successfully minimizes communication overhead but it may also significantly increase the computation time.

3.3. Less Communication and Less Computation via Ciphertext Interleaving

We now present a more efficient method to reduce the number of ciphertexts generated from KRDY's matrix protocol. Our method can run 20× faster than the PackLWEs of [12]. Recall that there are m_w irrelevant coefficients in the resulting polynomial, for each of the two needed values. To clean up the irrelevant coefficients, in Figure 4, we introduce a procedure ZeroGap which has a logarithmic complexity. The key concept is to compute $\hat{a} + \mathsf{Auto}(\hat{a}, N + 1)$. 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. In more general, let us compute $\hat{a} + \mathsf{Auto}(\hat{a}, N/2^j + 1)$ for any power of two factor 2^j of N. Take the $N = 8$ example again, and let us consider $2^j = 2$:

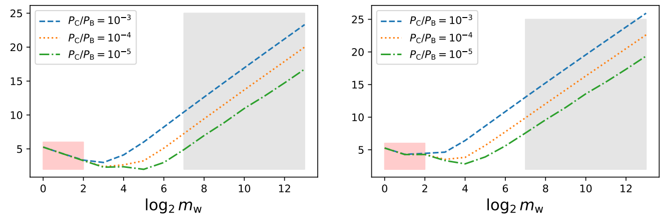
$$\mathsf{Auto}(\hat{a}, 8/2 + 1) = \sum_{i=0}^{i=7} a_i X^{i\cdot 5}$$
$$\equiv a_0 - a_2 X^2 + a_4 X^4 - a_6 X^6 + \sum_{i \nmid 2^j} a_i X^{i\cdot 5} \bmod X^8 + 1.$$

In brief, coefficients in odd multiples of 2^j are sign-flipped. Suppose the coefficients of $\hat{a}$ for all $i \nmid 2^j$ 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. Indeed, by repeatedly applying the formula

$$\hat{a} := \hat{a} + \mathsf{Auto}(\hat{a}, N/2^j + 1) \text{ for } j = 0, 1, \cdots, r - 1$$

we can retain only the (scaled) coefficients that lie at the 2^r-multiple positions. To eliminate the scaling factor, we can first multiply its inverse $2^{-r} \bmod q$ at the beginning of the procedure. This implicitly requires an odd modulus q, which is commonly satisfied in RLWE-based HE that uses modulus $q \equiv 1 \bmod 2N$. Also our ZeroGap procedure requires the gap between two needed coefficients is a two-power number which 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 $1 \le k_w m_w n_w \le N$ is satisfied.

With the ZeroGap procedure in hand, we present the procedure IntrLeave in Figure 5. This procedure allows us to interleave 2^r polynomials into a single polynomial



(a) $k = 128, m = 768, n = 768$ (b) $k = 196, m = 768, n = 3072$

Figure 6: Computation & Bandwidth cost under different partition window m_w and matrix dimensions. Polynomial degree $N = 2^{13}$. The pink (left) part is dominated by sending too much ciphertexts in Step 1 while the gray (right) part is dominated by the homomorphic automorphisms.

with the input coefficients arranged in a stride of 2^r steps. We would like to emphasize that the rotation over the coefficients can be achieved much more efficiently than the SIMD rotation [26] in the ciphertext domain. A toy example of `IntrLeave` is given in Figure **??**. Briefly, `IntrLeave` needs about $O(2^r \cdot r)$ automorphisms to combine N coefficients. For an appropriate choice of r, `IntrLeave` can require significantly fewer automorphisms than the `PackLWEs` procedure. Furthermore, it is noteworthy that `IntrLeave` is multi-core friendly since the 2^r calls to the `ZeroGap` procedure can be executed fully in parallel.

In Algorithm 1, we describe our matrix OLE protocol. The first three steps in Algorithm 1 are similar to those in the protocols [27, 29, 49]. 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 two-power partition window m_w in our approach. The main difference between our protocol and theirs is the use of the `IntrLeave` 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 interleaved polynomials and constructing the resultant matrix is explained in Figure 8 in the Appendix.

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

Parameters Selection via Cost Model. To achieve a 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 `IntrLeave` in Step 4 is $n_2 := k'n'm_\mathrm{w} \log_2 m_\mathrm{w}$. After the interleaving, $n_3 := \lceil \frac{m'n'}{m_\mathrm{w}} \rceil$ ciphertexts are sent in Step 5. To choose the

Algorithm 1 Proposed Matrix OLE Protocol Π_mOLE

Private Inputs: Sender S: $\mathbf{Q} \in \mathbb{Z}_{2^\ell}^{k \times m}$ and secret key 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 $\{\mathsf{ct}'_{\alpha,\beta} := \mathsf{RLWE}_\mathsf{sk}^{N,q,2^\ell}(\hat{q}_{\alpha,\beta})\}$ 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 IntrLeave on subvectors of $\mathbf{c}$. For example
       $$\tilde{\mathbf{c}}[\theta] := \mathtt{IntrLeave}(\underbrace{[\mathbf{c}[\theta \cdot m_\mathrm{w}], \mathbf{c}[\theta \cdot m_\mathrm{w} + 1], \cdots]}_{m_\mathrm{w}}),$$
   for $\theta \in [\tilde{m}]$. ▷ Pad with zero(s) when $k'n' \nmid m_\mathrm{w}$.
5: $S$ and $R$ jointly 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 $\hat{c}_{i,0} \in \mathbb{A}_{N,2^\ell}$ and $R$ obtains $\hat{c}_{i,1} \in \mathbb{A}_{N,2^\ell}$, 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.
```

proper partition window, we can minimize

$$\operatorname*{argmin}_{k_\mathrm{w}, m_\mathrm{w}, n_\mathrm{w}} P_\mathrm{C} \cdot n_2 + P_\mathrm{B} \cdot (n_1 + n_3) \tag{2}$$

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. Indeed, only the ratio $P_\mathrm{C}/P_\mathrm{B}$ is relevant for minimizing (2). 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. Based on our empirical results, it appears that selecting $m_\mathrm{w} \approx \sqrt{N}$ is a viable choice. Then the protocol in Algorithm 3 requires about $O(kn \log_2 N/(2\sqrt{N}))$ homomorphic automorphisms for the ciphertexts interleaving. To compare, the `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 $2^6 \cdot 6/2^{13} \approx 5\%$ of the ciphertext compression time of `PackLWEs`.

3.4. Further Optimizations

3.4.1. 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 resulting matrix is small, i.e., $|\mathbf{Q}_j \cdot \mathbf{K}_j| \le N/2$, we prefer to apply `IntrLeave` 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.

3.4.2. Dynamic Compression Strategy. We adopt a dynamic strategy for ciphertext compression that allows us to strike a balance between reducing communication 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 `IntrLeave` where the former can run faster for such small cases.

3.4.3. Using Symmetric Encryption & Modulus Reduction. We mention two common optimizations used by many HE-based applications [13, 14, 26]. One is to use the symmetric version of the RLWE encryption for the sender S. In the symmetric ciphertext, one of the ciphertext component is sampled uniformly at random. Thus we can just send a random seed (e.g., 256 bits) and saving about half of the communication size. Secondly, whenever a receiver responses a RLWE ciphertext $\mathsf{ct} \in \mathbb{A}_{N,q}^2$ to S, it can first apply the modulus reduction technique [8] to convert the ciphertext to a smaller modulus $\mathsf{ct}' \in \mathbb{A}_{N,q'}^2$ for $q' < q$ without affecting the decryption result. We note that both of these optimizations are compatible with the bOLE protocol in the next section.

4. Faster Batch OLE from RLWE

Batch OLE (bOLE) can be described as a 2-party computation protocol that takes a vector $\mathbf{x}$ from a sender S, and vectors $\mathbf{y}$ from a receiver R, and generates the share $[\![\mathbf{x} \odot \mathbf{y}]\!]$ between them [18]. 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 a smaller RLWE parameter. This approach may introduce Least-Significant-Bit (LSB) errors in the final output. Nonetheless, due to the use of fixed-point representation, the LSB errors can be safely removed through a subsequent truncation protocol, without affecting the overall accuracy of the computation.

Similarly to the RLWE-based bOLE protocol from [57], we also apply the SIMD technique [60] to implement the bOLE functionality with a better amortized efficiency. In [57], 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

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 $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,t}(\hat{x})$ 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 $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,t}(\hat{x})$, $R$ computes $\mathsf{ct} := \mathsf{RLWE}_{\mathsf{sk}}^{N,q,t}(\hat{x}) \boxtimes \hat{y}$.
4: Call $[\![\hat{u}]\!] \leftarrow \mathcal{F}_{\mathrm{H2A}}(\mathsf{ct})$ to convert to arithmetic share where $R$ inputs ct and $S$ inputs its secret key sk. Suppose $S$'s share is $[\![\hat{u}]\!]_0 \in \mathbb{A}_{N,t}$ and $R$'s share is $[\![\hat{u}]\!]_1 \in \mathbb{A}_{N,t}$.
5: $S$ outputs $\mathsf{Down}(\mathtt{SIMD}^{-1}([\![\hat{u}]\!]_0))$.
6: $R$ outputs $\mathsf{Down}(\mathtt{SIMD}^{-1}([\![\hat{u}]\!]_1))$.
```

$\mathsf{RLWE}_S^{N,q,t}(\mathtt{SIMD}(\mathbf{x}))$ the receiver R. Then R responses the ciphertext $\mathsf{RLWE}_S^{N,q,t}(\mathtt{SIMD}(\mathbf{x})) \boxtimes \mathtt{SIMD}(\mathbf{y}) \boxplus \mathtt{SIMD}(\mathbf{r})$ to S with a masking vector $\mathbf{r} \in \mathbb{Z}^N$. To provide a statistical security of σ-bit (e.g., $\sigma = 40$), R samples the vector $\mathbf{r}$ from $\mathbb{Z}_{2^{2\ell+\sigma}}^N$. To prevent the overflow from addition, [57] 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. We first introduce two helper functions:

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

The major differences with [57] are two-fold. In Figure 2, S sends $\mathsf{RLWE}_S^{N,q,t}(\mathtt{SIMD}(\mathsf{Lift}(\mathbf{x})))$ in the first step. Also R samples an informatics masking $\mathbf{r} \in_R \mathbb{Z}_t^N$. With the lifting function, we can achieve modulo-2^ℓ arithmetic even $t \nmid 2^\ell$.

Proposition 2. *If $t > 2^{2\ell}$ then $\mathsf{Down}(\mathsf{Lift}(x) \cdot y \bmod t) \equiv_\ell x \cdot y$ given any $x, y \in \mathbb{Z}_{2^\ell}$.*

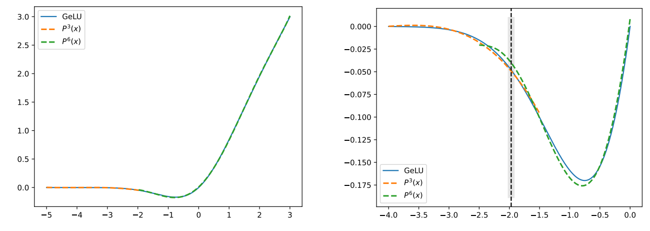
Proof. It suffices to show the error term can be rounded to zero. $\mathsf{Lift}(x) \cdot y \bmod t$ can be written as $\frac{t}{2^\ell} \cdot ((x \cdot y \bmod 2^\ell) + r_e \cdot y$ for a round error $|r_e| \le 1/2$. Then the term is rounded as $\lfloor \frac{2^\ell}{t} \cdot (r_e \cdot y) \rceil =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

We now able to use informatics random masking which allows us to avoid the extra σ-bit overhead. There might be a chance of introducing 1-bit error in the result.

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

The proof basically follows the similar argument in [29, full version, Appendix C].

Theorem 2. *The protocol Π_{bOLEe} in Algorithm 2 privately realizes the bOLEe functionality in presence of a semi-honest admissible adversary.*



(a) (b)

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.

For the concrete improvements, our bOLEe protocol requires $t \approx 2^{128}$ for $\ell = 64$, whereas the approach of [57] 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.

Beaver's Triples on-the-fly. The point-wise multiplication protocol $\Pi_{\mathrm{mul}}([\![\mathbf{x}]\!], [\![\mathbf{y}]\!])$ is commonly implemented using Beaver's triple [6]. A batch of Beaver's triples $([\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\![\mathbf{c}]\!])$ such that $\mathbf{c} \equiv_\ell \mathbf{a} \odot \mathbf{b}$ are usually pre-generated in an input-independent phase using two calls to the bOLE protocol [19, 57]. However, in `BumbleBee`, we employ a slightly different approach. Specifically, we compute the cross terms $[\![\mathbf{x}]\!]_0 \odot [\![\mathbf{y}]\!]_1$ and $[\![\mathbf{x}]\!]_1 \odot [\![\mathbf{y}]\!]_0$ directly using two bOLEs when the input size is large enough, i.e., $|\mathbf{x}| \ge N/2$. On the other hand, when $|\mathbf{x}| < N/2$, we utilize Beaver's triples to take full advantage of the SIMD batching used by our Π_{bOLEe} protocol. More specifically, we first generate $N/2$ Beaver's triples, and then consume parts of them to perform $\Pi_{\mathrm{mul}}([\![\mathbf{x}]\!], [\![\mathbf{y}]\!])$. The remaining triples are stored locally by each player, and will be used for subsequent multiplications.

We generate a batch of $N/2$ Beaver's triples on-the-fly using a single bOLE as follows. S and R samples $\mathbf{x} \in_R \mathbb{Z}_{2^\ell}^N$ and $\mathbf{y} \in_R \mathbb{Z}_{2^\ell}^N$, respectively and then feed it to Π_{bOLEe}. By parsing S's input as $\mathbf{x} = \mathbf{a}_0 \| \mathbf{b}_0$ for $|\mathbf{a}_0| = |\mathbf{b}_0| = N/2$, and parsing R's input as $\mathbf{y} = \mathbf{b}_1 \| \mathbf{a}_1$ for $|\mathbf{a}_1| = |\mathbf{b}_1| = N/2$, they obtain the share $[\![\mathbf{a}_0 \odot \mathbf{b}_1 \| \mathbf{b}_0 \odot \mathbf{a}_1]\!]$ at the end of the Π_{bOLEe} execution. Then each player can locally compute the triple $(\mathbf{a}_l, \mathbf{b}_l, \mathbf{c}_l := \mathbf{a}_l \odot \mathbf{b}_l + [\![\mathbf{a}_0 \odot \mathbf{b}_1]\!]_l + [\![\mathbf{a}_1 \odot \mathbf{b}_0]\!]_l)$ such that $(\mathbf{a}_0 + \mathbf{a}_1) \odot (\mathbf{b}_0 + \mathbf{b}_1) \equiv_\ell \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{e}$ for small errors $\|\mathbf{e}\|_\infty \le 2$. This idea can be extended for the square function as well.

The on-demand approach, which needs a small local cache, is relatively easy to integrate into existing the private frameworks such as `SPU` and `SiRNN`. This is because these frameworks were originally designed without considering the input-independent stage.

5. Optimized 2PC Protocols for the Activation Functions in Transformers

We first describe our protocol for the GeLU function. Note that the method and optimizations described in this section can also be used for other activation such as SiLU and ELU. See Appendix B for the details.

5.1. Gaussian Error Linear Unit (GeLU)

A common definition of the GeLU function is

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

We approximate the GeLU function using 4 segments

$$\mathtt{Seg4GeLU}(x) = \begin{cases} -\epsilon & x < -5 \\ P^3(x) = \sum_{i=0}^{i=3} a_i x^i & -5 < x \le -1.97 \\ P^6(x) = \sum_{i=0}^{i=6} b_i x^i & -1.97 < x \le 3 \\ x - \epsilon & x > 3 \end{cases} \tag{3}$$

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. To find the approximation polynomials $P^3(x)$ and $P^6(x)$, we can use the `numpy.polyfit` API. To ensure better accuracy and error tolerance, we recommend fitting these polynomials using a wider approximation range. For example, the coefficients of $P^3(x)$ can be fitted from the range of $[-6, -1]$. This wider range helps to avoid large fluctuations on the margins and provides more room for error tolerance in subsequent optimizations. Our goal is to obtain polynomials that $P^3(x) \approx P^6(x)$ for x around the pivot point, i.e., -1.97 in our case. Also, we suggest to fit the higher degree polynomial within a symmetric interval such as $[-4, 4]$. This will give us a "sparse" approximation polynomial which are cheaper to evaluate. For instance, the coefficients b_3 and b_5 in (3) are actually 0.

We note that in our GeLU protocol, we respond with a small value ϵ instead of 0 on the branch $x < -5$. This is to avoid the gradient vanishing issue if one apply our GeLU protocol for a private training task.

Theorem 3. *The protocol* Π_{GeLU} *in Algorithm 3 privately realizes* (3) *in presence of a semi-honest admissible adversary under the* $\mathcal{F}_{\mathrm{trunc}}$-, $\mathcal{F}_{\mathrm{lt}}$-, *and* $\mathcal{F}_{\mathrm{mux}}$- *hybrid.*

The number of activation can be massive in a transformer inference. We present three optimizations to further improve the concrete efficiency of Algorithm 3.

5.1.1. Approximated Segment Selection. The segment selection in (3) can be privately achieved using a private less-than $\mathbf{1}\{x < y\}$ for $x, y \in \mathbb{Z}_{2^\ell}$. The private less-than is commonly obtained via computing the most significant bit (MSB) of $x - y \bmod 2^\ell$ [29, 51, 56]. Our second optimization is to take advantage the smoothness of the activation function. For example, the evaluation of an input point $\tilde{x} = -1.95$ should be assigned to the 3rd segment and results at $P^6(-1.95) \approx -0.049938$ while the evaluation on the 2nd segment can also give a similar result $P^3(-1.95) \approx -0.050821$. Given these observations, we suggest to compute the less-than bit with some low-end f'

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: $[\![\texttt{Seg4GeLU}(\tilde{x}); f]\!]$. See (3) for definition.

```
1: Jointly 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: Jointly evaluate $[\![P^3(\tilde{x})+\epsilon; f]\!] \leftarrow \mathcal{F}^f_{\mathrm{trunc}}(\lfloor(\epsilon+a_0)\cdot 2^{2f}\rfloor + \sum_{k=1}^3 [\![\tilde{x}^k]\!]\cdot\lfloor a_k \cdot 2^f\rfloor)$, and $[\![P^6(\tilde{x})+\epsilon; f]\!] \leftarrow \mathcal{F}^f_{\mathrm{trunc}}(\lfloor(\epsilon+b_0)\cdot 2^{2f}\rfloor + \sum_k [\![\tilde{x}^k; f]\!]\cdot\lfloor b_k \cdot 2^f\rfloor)$.
3: Jointly compute the comparisons for segement selection
       $[\![b_0]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}([\![\tilde{x}]\!], -5)$      $\triangleright\ b_0 = \mathbf{1}\{\tilde{x} < -5\}$
       $[\![b_1]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}([\![\tilde{x}]\!], T)$       $\triangleright\ b_1 = \mathbf{1}\{\tilde{x} < -1.97\}$
       $[\![b_2]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}(3, [\![\tilde{x}]\!])$       $\triangleright\ b_2 = \mathbf{1}\{3 < \tilde{x}\}$
   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 $z_0 = \mathbf{1}\{-5 < \tilde{x} \le -1.97\}$, $z_1 = \mathbf{1}\{-1.97 < \tilde{x} \le 3\}$, and $z_2 = \mathbf{1}\{3 < \tilde{x}\}$.
4: Jointly 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 $[\![\texttt{Seg4GeLU}(\tilde{x}); f]\!]_l$ after subtracting $\lfloor \epsilon \cdot 2^f \rfloor$.
```

bits ignored to reduce the communication overhead. That is $\mathbf{1}\{x < y\} \approx \mathrm{MSB}((x-y)/2^{f'})$ and setting $f' < f$. We now compute the MSB over shares of $(\ell - f')$ bits length. Empirically, the approximated segment selection helps reducing 8% communication overhead of the GeLU protocol in Algorithm 3.

5.1.2. Batching (Approximated) Segment Selection. Our second optimization is specified for the concrete OT-based MSB protocol of [29, 42]. These protocol compute the MSB of an arithmetic share $[\![x]\!]$ using the formula

$$\mathrm{MSB}([\![x]\!]_0) \oplus \mathrm{MSB}([\![x]\!]_1) \oplus \mathbf{1}\{2^{\ell-1} - [\![x]\!]_1 < [\![x]\!]_0\},$$

where the computation of the last bit needs one $\binom{M}{1}$-$\texttt{OT}_2$. We thus need 3 calls of $\binom{M}{1}$-$\texttt{OT}_2$ for the segment selection in (3) while the choice bits to these $\binom{M}{1}$-$\texttt{OT}_2$ are identical. We suggest to combine the 3 calls of $\binom{M}{1}$-$\texttt{OT}_2$ as one $\binom{M}{1}$-$\texttt{OT}_6$, i.e., 1-of-M OT on 6-bit messages. While this OT combination may not reduce communication overhead by too much, it can help reduce computation time of GeLU by 20% according to our experiments. This is because the running time of one $\binom{M}{1}$-$\texttt{OT}_2$ execution is almost the same as that of one $\binom{M}{1}$-$\texttt{OT}_6$ execution when using the `Ferret` OT protocol [70].

5.1.3. Optimizing Polynomial Evaluation. For the evaluation of $P^3(x)$ and $P^6(x)$ in (3), 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. Additionally, we present a method 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 need to send 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 4 Private Softmax Protocol Π_{softmax}

Input: $[\![\tilde{\mathbf{x}}; 2f]\!] \in \mathbb{Z}^d_{2^\ell}$ with double-precision.
Output: $[\![\mathsf{softmax}(\tilde{\mathbf{x}}); f]\!] \in \mathbb{Z}^d_{2^\ell}$. See (4) for definition.

```
 1: Jointly compute $[\![\mathbf{b}]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}([\![\tilde{\mathbf{x}}; 2f]\!], \lfloor T_{\exp} \cdot 2^f \rfloor)$.
 2: Jointly compute the maximum $[\![\bar{x}; 2f]\!] \leftarrow \mathcal{F}_{\max}([\![\tilde{\mathbf{x}}; 2f]\!])$.
 3: $P_l$ locally computes $[\![\tilde{\mathbf{y}} = \tilde{\mathbf{x}} - \bar{x}; 2f]\!]$.
 4: Jointly compute $[\![\tilde{\mathbf{z}}_0; f]\!] \leftarrow 1 \cdot 2^f + \mathcal{F}^{n+f}_{\mathrm{trunc}}([\![\tilde{\mathbf{y}}; 2f]\!])$.
 5: for $i = 1, 2, \cdots, n$ sequentially do
 6:     $[\![\tilde{\mathbf{z}}_i; f]\!] \leftarrow \Pi_{\mathrm{square}}([\![\tilde{\mathbf{z}}_{i-1}; f]\!])$ $\triangleright\ \tilde{\mathbf{z}}_i = (\tilde{\mathbf{z}}_{i-1})^2$
 7: end for
 8: $P_l$ locally aggregates $[\![\tilde{z}; f]\!]_l \in \mathbb{Z}_{2^\ell} \leftarrow \sum_{i\in[d]} [\![\tilde{\mathbf{z}}_n[i]]\!]_l$.
 9: Jointly compute $[\![1/\tilde{z}; f]\!] \in \mathbb{Z}_{2^\ell} \leftarrow \mathcal{F}_{\mathrm{recip}}([\![\tilde{z}; f]\!])$.
10: Joint compute $[\![\tilde{\mathbf{z}}_n/\tilde{z}; f]\!] \leftarrow \Pi_{\mathrm{mul}}([\![\tilde{\mathbf{z}}_n]\!], [\![1/\tilde{z}]\!])$.
11: Output $[\![\mathbf{b} \odot (\tilde{\mathbf{z}}_n/\tilde{z}); f]\!]$ using the $\mathcal{F}_{\mathrm{mux}}$ functionality.
```

5.2. Attention and Softmax

For the sake of numerical stability [25, Chapter 4], the softmax function is commonly computed as

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

where $\bar{x}$ is the maximum element of the input vector $\mathbf{x}$. For a two-dimension matrix, we apply (4) to each of its row vector. It is worth noting that all inputs to the exponentiation operation in (4) are negative. We leverage the negative operands to accelerate the private softmax. Particularly, we compute the exponentiation using the Taylor series with a simple clipping branch on a hyper-parameter $T_{\exp} < 0$

$$\exp(x) \approx \begin{cases} 0 & x < T_{\exp} \\ \left(1 + \dfrac{x}{2^n}\right)^{2^n} & x \in [T_{\exp}, 0]. \end{cases}$$

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 our MPC program uses $f = 18$. 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 a targeting precision. 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 are smooth too. The division by 2^n can be achieved using a call to $\mathcal{F}^n_{\mathrm{trunc}}$, and the power-to-$2^n$ is computed via a sequences of Π_{square}.

Our second optimization involves fusing a portion of the linear projection in the Attention mechanism with the softmax function. More specifically, we propose combining the truncation $\mathcal{F}_{\mathrm{trunc}}^{n}$ used for division by 2^n with the truncation $\mathcal{F}_{\mathrm{trunc}}^{f}$ used in the matrix multiplication operation $\mathbf{Q} \cdot \mathbf{K}^{\top}$, into a single truncation operation $\mathcal{F}_{\mathrm{trunc}}^{f+n}$. This modification does not affect correctness, as the max protocol lacks awareness of fixed-point precision.

Theorem 4. Π_{softmax} *in Algorithm 4 privately realizes* (4) *in presence of a semi-honest admissible adversary under the* $\mathcal{F}_{\mathrm{lt}}$-*,* $\mathcal{F}_{\mathrm{mux}}$-*,* $\mathcal{F}_{\mathrm{trunc}}$-*, and* $\mathcal{F}_{\mathrm{recip}}$-*hybrid.*

6. Related Work

Private Transformer Inference. Several frameworks for privacy-preserving machine learning (PPML) are capable of conducting private inference on transformers, including `CrypTen` [40], `SiRNN` [55], `MP-SPDZ` [39] and `SPU` [1, 48]. `CrypTen` and `SiRNN` support 2PC, with `CrypTen` demanding an additional trusted dealer for the optimal efficiency. On the other hand, `MP-SPDZ` and `SPU` can support both 2PC and three-party computation (3PC). Furthermore, researchers have improved the existing PPML frameworks and developed systems for private transformer inference. For instance, [27] is a 2PC inference system that integrates `SiRNN`'s OT-based protocols for the non-linear functions. [44] is built on-top of `CrypTen` but heavily relies on a trusted dealer for performance. Both of [27, 44] require extensive communication.

Low Communication mOLE. Many works that leverages the homomorphic SIMD [60] can be directly used for the mOLE functionality with a relatively small communication, such as [11, 26, 28, 32]. The SIMD technique in turn demands a prime plaintext modulus t rather than 2^{ℓ}. 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.

Non-linear Functions. To evaluate the softmax function, `CrypTen` and `CryptGPU` [61] also use the Taylor series $(1 + x/2^n)^{2^n}$ to approximate the exponentiation. But they do not apply the range clipping. For example, `CryptGPU` empirically sets $n = 9$ according to their examination on some datasets. On the other hand, `MP-SPDZ` and `SPU` [48] uses the formula $\exp(x) = 2^{x \cdot \log_2(e)}$. `SiRNN` uses a private Look-up-Table for an initial guess with a few Newton iterations. These approaches do provide a more stable and precise exponentiation but also expensive for 2PC. `SPU` also approximates $\tanh(x) \approx (x + x^3/9.0 + x^5/945.0)/(1 + 4x^2/9.0 + x^4/63.0)$. `MiniONN` [45] also uses splines to approximate activation functions while they use Garble Circuit for the branch selection. Kelkar et al. [36] 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.

Instead of calculating the complicated softmax and activations, many frameworks turn to alternatives that are easy to compute privately. For instance [52] replaces the softmax function as $\frac{\mathsf{ReLU}(\mathbf{x}[i])}{\sum_i \mathsf{ReLU}(\mathbf{x}[i])}$. Also [44] adapts $\frac{(\mathbf{x}[i]+c)^2}{\sum_i (\mathbf{x}[i]+c)^2}$ as the alternative for softmax and replaces $\mathtt{GeLU}(x) \approx x^2/8 + x/4 + 1/2$. However, some works have shown that such MPC-friendly replacements might deteriorate the accuracy [38].

7. Evaluations

Models & Datasets. We evaluate `BumbleBee` on 5 Transformer models, including four NLP models, i.e., BERT-base, BERT-large [20], GPT2-base [16], and LLaMA-7B [62], and a computer vision model ViT-base [21, 67]. All these pre-trained models are downloaded from HuggingFace. These models are parameterized by three hyper-parameters: the number of blocks B, the dimension of representations D and the number of heads H. A detailed description of these transformer models can be found in Table 6 in the Appendix.

To demonstrate the effectiveness of `BumbleBee`, we conducted private inference on 4 datasets, which includes CoLA, RTE, and QNLI from the GLUE benchmarks [64] for NLP tasks, and ImageNet-1k [58] 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.

Testbed Environment. All of the experiments described in this paper were conducted on two cloud instances, equipped with 104 processors (on two physical sockets), operating at 2.50GHz, and 384 GB of RAM. We utilize the 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 a bandwidth of 1 gigabit per second (1 Gbps), and a ping time of 0.5ms, and a Wide Area Network (WAN) with a bandwidth of 400Mbps and a ping time of 4ms.

Metrics. We do not distinguish between the "offline" and "online" costs as in some prior works, such as [50, 56]. 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 the transformer model from the 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.

Concrete Parameters. We set $\ell = 64$ for the secret sharing and set the fixed-point precision $f = 18$. $\mathsf{HE.pp}_{\mathrm{bOLE}} = \{N = 4096, q \approx 2^{60+52}, t \approx 2^{45\times3}\}$ for the bOLEe protocol, and $\mathsf{HE.pp}_{\mathrm{mOLE}} = \{N = 8192, q \approx 2^{59+55}, t = 2^{64}, q' = 2^{49}\}$ for the mOLE protocol, where q' is needed for the homomorphic automorphism. We apply [30] to accelerate SEAL on Intel CPUs. We extend the `Ferret` implementation in the EMP library [65] to support various application-level OT types, such as $\binom{M}{1}$-$\mathtt{OT}_6$. The protocols Π_{trunc} [17], $\Pi_{\max}$ [41], Π_{recip} [10] and Π_{rsqrt} [33] protocol are already implemented in the SPU framework [1].

For the approximated less-than, we set $f_{\mathrm{lt}} = 16$. In other words, we evaluate an MSB protocol on 48-bit inputs for the segment selection in (3). Also, for the concrete MSB protocol [56], we use the $M = 16$ hyper-parameter. For

TABLE 2: Comparison of proposed protocols with SOTA in terms of running time and communication costs. Each machine was tested with 25 threads. Timing results are averaged from 20 runs.

(k, m, n)	$\Pi_{\text{mOLE}}(\mathbf{X}, \mathbf{Y})$	Comm.	LAN	WAN
$(1, 50257, 768)$	[59]	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
$(128, 768, 768)$	[59]	18.41GB	159.98s	397.83s
	`KRDY`	30.16MB	0.66s	1.06s
	`KRDY`$^+$	5.02MB	7.07s	7.85s
	Ours	5.02MB	0.82s	0.84s
	$\Pi_{\text{bOLE}}(\mathbf{x}, \mathbf{y})$	Comm.	LAN	WAN
$\lvert\mathbf{x}\rvert = \lvert\mathbf{y}\rvert = 2^{15}$	[57]	7.54MB	0.07s	0.17s
	Ours	5.78MB	0.05s	0.14s
$\lvert\mathbf{x}\rvert = \lvert\mathbf{y}\rvert = 2^{20}$	[57]	241.26MB	2.39s	5.33s
	Ours	184.46MB	1.71s	4.02s
	$\Pi_{\text{GeLU}}(\mathbf{x})$	Comm.	LAN	WAN
$\lvert\mathbf{x}\rvert = 2^{20}$	[55]	16.06GB	141.52s	353.76s
	[48]	3.54GB	66.50s	103.68s
	Ours†	0.77GB	10.73s	17.84s
	Ours‡	0.75GB	8.21s	15.71s
	Ours	0.69GB	6.89s	13.77s
$\lvert\mathbf{W}\rvert$	$\Pi_{\text{softmax}}(\mathbf{W})$	Comm.	LAN	WAN
$(960, 180)$	[55]	1697.86MB	16.39s	40.84s
	[39, 48]	435.14MB	9.28s	14.53s
	Ours	162.24MB	2.11s	5.79s

* Ours is identical to `KRDY`$^+$ in this case due to the dynamic strategy.
† We call 3 $\binom{M}{1}$-$\texttt{OT}_2$ without approximated less-than in this run.
‡ We call 1 $\binom{M}{1}$-$\texttt{OT}_6$ without approximated less-than in this run.

approximating the exponentiation, we set $T_{\text{exp}} = -14$ and use $n = 6$. We fixed the ratio $P_C/P_B = 10^{-3}$ in (2).

7.1. Microbenchmarks

In Table 2, we compare the performance of the proposed protocols with many SOTA approaches.

Linear Operations. To demonstrate the performance of the proposed mOLE protocol, we have implemented the RLWE-based approach from `Iron` [27] and our baseline `KRDY`$^+$ which adapts the `PackLWEs` [12] procedure to reduce the communication overhead of `Iron`. We also compared our protocols with the OT-based approach from the $\texttt{SCI}_{\texttt{OT}}$ library [59], which leverages the IKNP OT [31]. The three RLWE-based approaches were found to be more communication-friendly than the OT-based method. Our mOLE protocol was shown to be rapid and light With our ciphertext interleaving technique, our mOLE protocol is 80 – 90% less communication intensive than `Iron`. In terms of computation time, our ciphertext interleaving approach was found to be 7 – 14× faster than the `PackLWEs`-based `KRDY`$^+$. Although we need to perform some homomorphic automorphisms to reduce the number of RLWE ciphertexts, this helps reduce the computation of decryption. Consequently, the overall running time of our mOLE protocol is only slightly larger than that of `Iron` in LAN.

Non-linear Activation Functions. We compare our non-linear protocols with the default implementations in the `SPU` library [1] by changing their underlying OT to `Ferret` [65]. Also, we compare with `SiRNN` using their codes from [59]. The results in Table 2 demonstrate that our protocols yielded a significant decrease in communication costs, resulting in savings of 95% communication for GeLU and 90% for softmax. Additionally, we run our GeLU protocol without the two optimizations, namely the approximated less-than (§5.1.1) and batched MSBs (§5.1.2). Our findings indicate that these two optimizations can reduce the computation time of GeLU by about 12% and 23% respectively.

Overall Improvements. To show the overall improvements by our protocols, we compare with a baseline that is implemented on top of the `SPU` library [1]. We defer the details of this artificial baseline to Appendix. We summarize the results in Figure 1, which are produced by replacing the protocols in the baseline with our proposed protocols. From the results, we observe that our softmax and GeLU protocols make significant contributions towards reducing the running time while our mOLE protocol results in a substantial decrease in the overall communication.

TABLE 3: 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 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-1k	985	one img one class	0.8944	0.8913

7.2. Evaluation on Large Transformers

We have successfully 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 int 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. Specifically, we randomly selected 1000 texts from the QNLI dataset, we also removed a total of 15 grayscale images from the ImageNet-1k dataset. We note that all our experiments were conducted using the proposed 2PC protocols, rather than through fixed-point simulation. The results are given in Table 3. As shown in the table, `BumbleBee` attains comparable levels of accuracy when compared to the cleartext prediction.

TABLE 4: 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 model 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	9.08	98.24	61.44	8	3.76	11.10	9.64
`mixed_mmul`	embedding lookup	128	38.06	229.01	180.71	8	30.89	18.31	16.39
`f_mmul`	linear projections	49	75.95	740.10	693.35	225	747.25	1303.14	1272
`f_batch_mmul`	multi-head attention	24	15.05	165.72	156.02	64	10.94	403.26	400.21
`f_less`	max / argmax	131	3.17	157.88	30.01	117	0.73	6.08	1.31
`multiplixer`	max / argmax	411	0.55	19.71	19.71	155	0.30	1.20	1.20
`f_exp`	softmax	12	12.14	805.49	663.88	32	2.27	39.33	36.08
`f_reciprocal`	softmax	12	2.54	29.55	18.45	32	1.36	12.87	7.95
`f_mul`	layer norm, softmax	174	8.68	779.62	749.68	356	10.53	758.30	730.10
`f_rsqrt`	layer norm	25	1.88	5.31	2.90	65	1.14	0.81	0.58
`f_seg4_act`	GeLU / SiLU	12	30.79	1951.34	1226.35	32	17.99	1175.45	745.04
		Total	3.41min	4.87GB	3.71GB	Total	13.87min	3.66GB	3.16GB

Break-down. Table 4 breaks down the BumbleBee inference runtime and communication for GPT2-base (left) and LLaMA-7B (right). The input to these two models consist of 128 and 8 tokens, respectively. The first column of Table 4 indicate 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. The truncation protocol [17, 29] may be implicitly invoked within these fixed-point operations. In terms of communication, the activations make up about 60% of the communication costs. In contrast, matrix multiplication is the most expensive operation which takes 70 – 95% of the total inference time. Due to the space limit, we provide the performance breakdown of ViT-base in Appendix.

We note that the `i_equal` operation can be saved by allowing the client to directly query the one-hot vector. Nonetheless, there are scenarios where a private conversion from token-id to one-hot vector is still required, 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.

7.3. Comparison with Existing Methods

In Table 5, we compare the performance of BumbleBee with three private transformer solutions. In summary, we have achieved a 10-fold improvement in inference time while and a reduction of communication costs by 90% compared to the Iron framework, even they have used a smaller ring $\ell = 37$, compared to ours $\ell = 64$. As Iron's implementation is not publicly available, we have also compared our approach with our artificial baseline on two additional transformers. Our results indicate that our approach has achieved about 1.7 – 2$\times$ improvement in inference time and reduced communication costs by 60 – 80% when compared to the artificial baseline.

We also provide the performance of MPCFormer [44], an efficient 3-party (or dealer-aided) private inference framework for transformer. These results are obtained by re-running their implementation under ours environment. As shown in the table, BumbleBee is about 1.5$\times$ slower than MPCFormer in LAN, which is expected since their solution is cryptographic-free, and its performance heavily relies on the trusted dealer. However, BumbleBee is faster than MPCFormer on WAN since BumbleBee introduces 35 – 60% less communication than theirs.

TABLE 5: End-to-end comparisons with two existing private inference frameworks and a baseline built from SPU. The numbers of Iron are estimated from their paper. GPT2 models generated 1 token.

Model	Framework	Time (min) LAN	WAN	Comm. (GB)
BERT-large 128 input tokens	MPCFormer	4.52	9.81	32.58
	Iron	≈ 100	–	≈ 200
	Artificial basline	11.88	18.37	52.14
	BumbleBee	6.74	9.88	20.85
GPT2-base 32 input tokens	MPCFormer	0.72	1.96	4.98
	Artificial basline	1.52	2.64	6.36
	BumbleBee	0.92	1.32	1.94
GPT2-base 64 input tokens	MPCFormer	1.10	2.85	7.32
	Artificial basline	2.74	4.45	11.55
	BumbleBee	1.55	2.53	3.90

Conclusion and Future Work

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-preserving deep learning techniques in a variety of applications.

In the furture, 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 [34, 69].

References

[1] (2022, Sep.) SPU: Secure Processing Unit. https://github.com/secretflow/spu.

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

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

[6] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, vol. 576, 1991, pp. 420–432.

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

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Innovations in Theoretical Computer Science 2012, Cambridge, USA, January 8-10, 2012*, pp. 309–325.

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

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

[11] 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.

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

[13] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled PSI from Fully Homomorphic Encryption with Malicious Security," in *CCS*, 2018, pp. 1223–1237.

[14] H. Chen, K. Laine, and P. Rindal, "Fast Private Set Intersection from Homomorphic Encryption," in *CCS*. ACM, 2017, pp. 1243–1255.

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

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

[17] 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.

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

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

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

[21] 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.

[22] 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.

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

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

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

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

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

[28] 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.

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

[30] (2021, Oct.) Intel HEXL (release 1.2.2). https://arxiv.org/abs/2103.16400.

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

[32] 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.

[33] W. jie Lu, Y. Fang, Z. Huang, C. Hong, C. Chen, H. Qu, Y. Zhou, and K. Ren, "Faster Secure Multiparty Computation of Adaptive Gradient Descent," in *PPMLP*, 2020, pp. 47–49.

[34] 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.

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

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

[37] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *CCS 2020*, pp. 1575–1590.

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

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

[40] 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.

[41] V. Kolesnikov, A. Sadeghi, and T. Schneider, "Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima," in *CANS 2009*, vol. 5888. Springer, 2009, pp. 1–20.

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

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

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

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

[46] 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.

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

[48] 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.

[49] 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.

[50] 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.

[51] P. Mohassel and P. Rindal, "ABY3: A Mixed Protocol Framework for Machine Learning," in *CCS*, 2018, pp. 35–52.

[52] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *SP*. IEEE Computer Society, 2017, pp. 19–38.

[53] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: improved mixed-protocol secure two-party computation," in *USENIX Security*, M. Bailey and R. Greenstadt, Eds., 2021, pp. 2165–2182.

[54] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," 2019.

[55] 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.

[56] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *CCS*. ACM, 2020, pp. 325–342.

[57] 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.

[58] 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.

[59] (2021, Jun.) EzPC - a language for secure machine learning. https://github.com/mpc-msri/EzPC/tree/master/EzPC.

[60] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

[61] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU," in *SP*. IEEE, 2021, pp. 1021–1038.

[62] 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.

[63] 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.

[64] 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.

[65] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2022.

[66] 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.

[67] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan,

TABLE 6: Transfomer models are parameterized by three hyper-parameters: the number of blocks B, the dimension of representations D and the number of heads H. MLP means multi-layer perceptron.

Model	Hyper parameters	MLP size	#Parm
BERT-base	$B = 12, D = 768, H = 12$	3072	110M
GPT2-base	$B = 12, D = 768, H = 12$	3072	117M
ViT-base	$B = 12, D = 768, H = 12$	3072	86M
BERT-large	$B = 24, D = 1024, H = 16$	4096	340M
LLaMA-7B	$B = 32, D = 4096, H = 32$	11008	7B

M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual Transformers: Token-based Image Representation and Processing for Computer Vision," 2020.
[68] 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.
[69] 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.
[70] 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.
[71] 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.

Appendix

1. An Artificial Baseline

We present a baseline for the 2PC private inference on transformer through the combination of existing methods. For the matrix multiplication protocol, we suggest using RLWE-based methods [49] as they are more communication-friendly compared to OT-based approaches. For the truncation protocol, we propose combining the approximated truncation [29] with the positive heuristic from [17]. As for the softmax protocol, we suggest using the Taylor expansion from [61] and computing $\exp(x) \approx (1 + x/2^n)^{2^n}$ for $n = 9$ *without* a range clipping. For the activation functions, we use the spline-based approach from [45], which approximates them using 12 splines. All of these protocols can be implemented using the HE/OT primitives from the SPU library.

```
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

2. Activation Functions

We use the following coefficients of (3) for the GeLU approximation.

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

Using the method described in §5.1, we can approximate the SiLU function $\mathsf{SiLU}(x) = \dfrac{x}{1+\exp(-x)}$ as

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

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.

TABLE 7: Zoom-in the matrix multiplications involved in GPT2-base in Table 4.

$k \times m \times n$	$k_\mathrm{w} \times m_\mathrm{w} \times n_\mathrm{w}$	Operation	Sent	Recv	`IntrLeave` / `PackLWEs`	#Calls
$1 \times 50257 \times 768$	$1 \times 8192 \times 1$	`mixed_mmul`	1.24MB	0.13MB	90.63ms	128
$128 \times 768 \times 2304$	$128 \times 64 \times 1$	`f_mmul`	2.12MB	8.73MB	740.69ms	12
$H@128 \times 64 \times 128$	$128 \times 64 \times 1$	`f_batch_mmul`	2.12MB	5.82MB	490.20ms	12
$H@64 \times 128 \times 128$	$64 \times 128 \times 1$	`f_batch_mmul`	2.12MB	2.91MB	531.93ms	12
$128 \times 768 \times 768$	$128 \times 64 \times 1$	`f_mmul`	2.12MB	2.91MB	245.01ms	12
$128 \times 768 \times 3072$	$128 \times 64 \times 1$	`f_mmul`	2.12MB	11.64MB	999.85ms	12
$128 \times 3072 \times 768$	$128 \times 64 \times 1$	`f_mmul`	8.47MB	2.91MB	238.33ms	12
$128 \times 768 \times 50257$	$128 \times 16 \times 4$	`f_mmul`	8.47MB	190.66MB	7257.34ms	1

TABLE 8: Performance breakdown of `BumbleBee` on ViT-base. The input is one 224×224 RGB image. The LAN setting was used.

Operation	ViT-base ($B = 12$, $D = 768$, $H = 12$)			
	#Calls	Time (sec)	Sent (MB)	Recv (MB)
`f_mmul`	74	85.59	1144.05	1098.31
`f_batch_mmul`	24	28.32	393.50	373.20
`multiplixer`	145	1.34	43.13	43.65
`f_less`	145	20.34	392.31	73.48
`f_exp`	12	28.35	1910.65	1579.13
`f_reciprocal`	12	5.73	46.12	28.49
`f_mul`	174	15.03	1385.10	1333.12
`f_rsqrt`	25	0.67	6.21	4.13
`f_seg4_gelu`	12	48.10	3000.59	1898.47
	Total	3.90min	8.14GB	6.30GB

TABLE 9: Compared with `PackLWEs`. Fixing $m_\mathrm{w} = 64$ for `IntrLeave`. 4C means 4 threads were used.

Matrix Shape	(16, 768, 768)	(256, 768, 768)	(128, 768, 3072)
`PackLWEs` (4C)	4901.00ms	74.85s	150.35s
`IntrLeave` (4C)	206.17ms	3.43s	6.55s
Speedup	23.77×	21.82×	22.95×
`PackLWEs` (16C)	1714.42ms	21.01s	38.87s
`IntrLeave` (16C)	82.28ms	1.16s	2.03s
Speedup	20.83×	18.11×	19.15×
`PackLWEs` (26C)	1204.57ms	17.39s	35.12s
`IntrLeave` (26C)	52.22ms	0.81s	1.58s
Speedup	23.07×	21.47×	22.23×

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

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

using the proposed exponential protocol on the first branch.

3. More Experiment Results

In Table 7, we present more details on each matrix multiplications in the GPT2 inference. We know that the final matrix multiplication for token generation is very large, which needs more optimizations. In Table 8, we present the running details of `BumbleBee` on a Vision Transformer. In Table 9, we compare the performance of `IntrLeave` with `PackLWEs`. We can see that `IntrLeave` is about $20\times$ faster than `PackLWEs`.

4. 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., $\mathsf{ct} = \mathsf{RLWE}_{\mathsf{pk}}^{N,q,t}(\mathsf{SIMD}(\mathbf{m}))$ for $\mathbf{m} \in \mathbb{Z}_t^N$ for a prime modulus t. The sender S needs some random masking before sending ct for decryption. Specifically, S computes $\mathsf{ct}' := \mathsf{ct} \boxplus \mathsf{RLWE}_{\mathsf{pk}}^{N,q,t}(\mathsf{SIMD}(\mathbf{r}); e')$ where the random vector $\mathbf{r} \in_R \mathbb{Z}_t^N$. As mentioned in [57], 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. Suppose the decryption is correctly done. Then we can have $-\mathbf{r} \bmod t$ as one of the arithmetic share of $\mathbf{m}$, and having the decryption of ct' as the other share.

The second conversion is used for RLWE ciphertext of a normal polynomial, e.g., $\mathsf{ct} = \mathsf{RLWE}_{\mathsf{pk}}^{N,q,2^\ell}(\hat{m})$. Let parse ct as a tuple $(\hat{b}, \hat{a}) \in \mathbb{A}_{N,q}^2$. To properly masking ct, the sender S computes $\mathsf{ct}' := (\hat{b} + \hat{r}, \hat{a}) \boxplus \mathsf{RLWE}_{\mathsf{pk}}^{N,q,2^\ell}(0)$ where the random polynomial $\hat{r} \in_R \mathbb{A}_{N,q}$. Let the decryption of ct' be $\hat{u} \in \mathbb{A}_{N,2^\ell}$, which can be viewed as one share of $\hat{m}$. On the other hand, we have $\hat{v} := -\lfloor \frac{2^\ell \cdot \hat{r}}{q} \rceil \bmod 2^\ell$ as the other share of $\hat{m}$. However, generating a uniform random $\hat{r}$ may lead to decryption failure. According to the analysis in [29], it will introduce at most a 1-bit error into the arithmetic share, i.e., $\|\hat{m} - (\hat{u} + \hat{v} \bmod 2^\ell)\|_\infty \leq 1$.

5. Threat Model and Security

We provide security against a static semi-honest probabilistic polynomial time adversary $\mathcal{A}$ following the simulation paradigm [24]. 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 [50]. 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, `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 `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.*

6. Proofs

Proof of Theorem 1. The correctness of Theorem 1 is directly derived from Proposition 1. We now show the the privacy part.

(**Corrupted Receiver.**) Receiver's view of $\mathsf{View}_R^{\Pi_{\mathrm{mOLE}}}$. consists of RLWE ciphertexts $\{\mathsf{ct}'_{\alpha,\beta}\}$ for $\alpha \in [k']$ and $\beta \in [m']$. The simulator $\mathcal{S}_R^{\Pi_{\mathrm{mOLE}}}$ for this view can be constructed as follows.

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

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

(**Corrupted Sender.**) Sender's view of $\mathsf{View}_S^{\Pi_{\mathrm{mOLE}}}$ 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 $\mathcal{F}_{\mathrm{H2A}}$. The simulator $\mathcal{S}_S^{\Pi_{\mathrm{mOLE}}}$ for this view can be constructed as follows.

1) On receiving the RLWE ciphertexts $\{\mathsf{ct}'\}$ from R and given the access to meta, $\mathcal{S}_S^{\Pi_{\mathrm{mOLE}}}$ samples uniform random polynomial $\hat{r}_i \in_R \mathbb{A}_{N,q}$ and computes $\mathsf{ct}_i := \mathsf{RLWE}_{\mathsf{pk}}^{N,q,2^\ell}(0) \boxplus (\hat{r}_i, 0)$ for $i \in [\tilde{m}]$.
2) $\mathcal{S}_S^{\Pi_{\mathrm{mOLE}}}$ computes the rounding $\hat{r}'_i := \lfloor 2^\ell \cdot \hat{r}_i / q \rceil \bmod 2^\ell$ for $i \in [\tilde{m}]$, then 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.

Similarly, the RLWE ciphertexts $\mathbf{c}[i] \approx^c \mathsf{ct}_i$ due to the informatics masking $\hat{r}_i$ and the semantic security. Also, the values in the output matrix $[\![\mathbf{U}]\!]_l$ of S in Π_{mOLE} distribute uniformly in $\mathbb{Z}_{2^\ell}$ which is exact the same distribution $\mathcal{S}_S^{\Pi_{\mathrm{mOLE}}}$ creates $\tilde{\mathbf{U}}$. Thus we have $\mathsf{View}_S^{\Pi_{\mathrm{mOLE}}} \approx^c \mathcal{S}_S^{\Pi_{\mathrm{mOLE}}}(\mathsf{meta}, \mathsf{out})$. □

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 $\mathcal{F}_{\mathrm{H2A}}$ function for SIMD-packed messages. Also, we argue that the security proofs for Theorem 3 and Theorem 4 simply follow in the hybrid model since only OT messages and properly masked HE ciphertexts are exchange.