BumbleBee: Secure Two-party Inference Framework for Large Transformers

今天介绍的是来自蚂蚁密码学实验室的Wen-jie Lu等人的安全两方大模型推理工作BumbleBee，论文链接如下：
https://eprint.iacr.org/2023/1678

1. 背景介绍

针对Transformer模型的安全推理工作，尤其是两方推理还需要大量的通信和计算开销。本文聚焦Transformer模型安全推理中的大模型矩阵乘法和复杂激活函数计算，提出了多个高效两方计算计算协议和优化技术。具体来说，本文的主要贡献如下：

1. 本文提出了基于HE的高效矩阵乘法协议，和已有方案相比减少了$80\%\ 99.\%$的通信量，并且本文提出的矩阵乘法耗时也很少；
2. 本文针对Transformer模型中的复杂激活函数，比如GeLU和SiLU，提出了高效而准确的近似计算算法。和已有的简单方式相比，本文的方法不需要针对已有的模型进行任何的fine-tuning或者post-training；
3. 本文在SPU的基础上进行了实现，和已有的方案相比，本文将通信开销减少了$60\% - 83\%$。且BumbleBee可以直接针对HuggingFace上已有的模型直接加载进行安全预测，架构如下图所示：

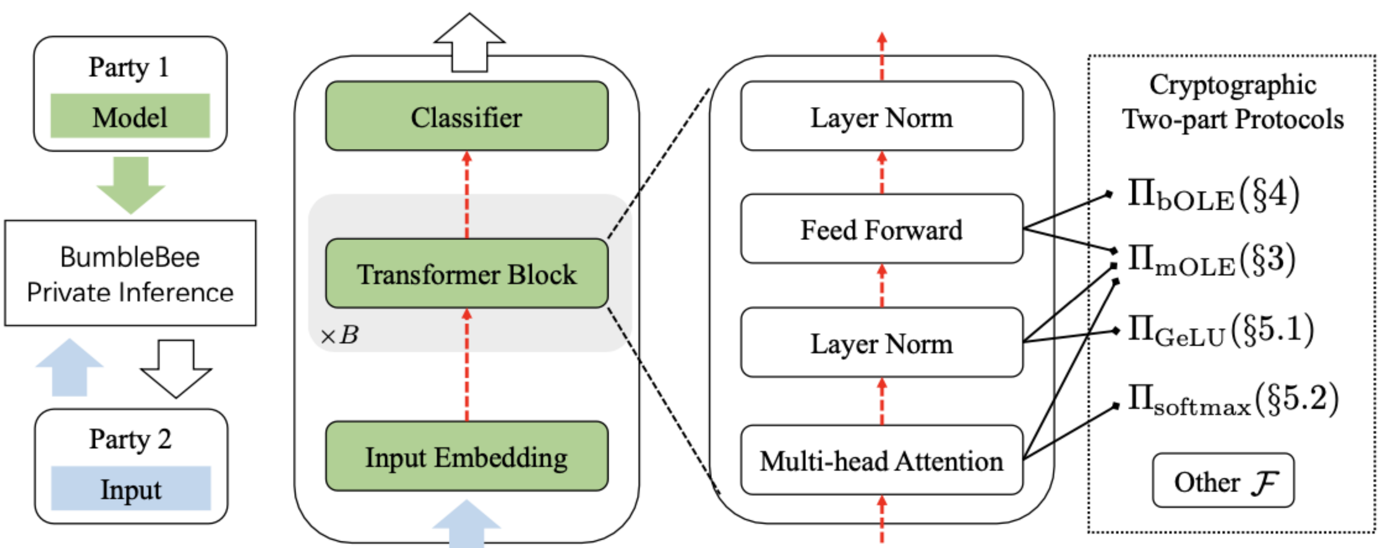

Figure 2: Overview of BumbleBee's private transformer inference. The dash arrows indicate secretly shared messages.

4. 如下图所示，本文的方案在通信开销和时间开销上都优于之前的相关工作。

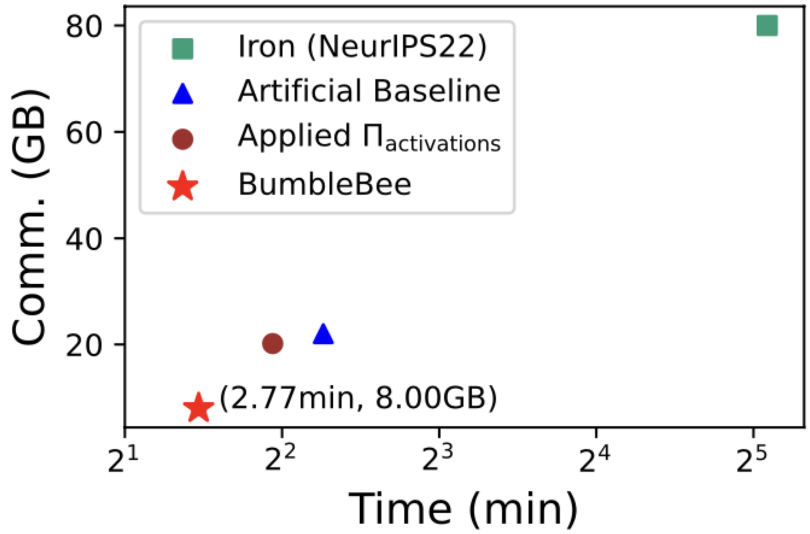

Figure 1: The overall bandwidth improvements of the proposed optimizations on the BERT-base model with 128 input tokens. The baseline consists of many SOTA 2PC protocols that are already communication-friendly.

2. 基础知识

本文使用如下Notations：$\langle x\rangle$表示两方加法秘密分享；$1\{\mathcal{P}\}$表示当条件为$\mathcal{P}$为真时返回1；$\hat{a}$表示多项式，其中$\hat{a}[j]$表示多项式$\hat{a}$的第j个系数，$\hat{a}\cdot\hat{b}$表示多项式乘法；$x\equiv_\ell y$表示$x\equiv y \pmod{2^\ell}$；对于N（N是2的幂次）和$q>0$，$\mathbb{A}_{N,q}$表示整数多项式集合$\mathbb{A}_{N,q}=\mathbb{Z}_q[X]/(X^N+1)$；$\mathbf{a}$和$\mathbf{M}$分别表示向量和矩阵，其中$\mathbf{a}[j]$表示第$j$个向量元素，$\mathbf{M}[j,i]$表示第$(j,i)$个矩阵元素。Hadamard积表示为$\mathbf{a}\odot\mathbf{b}$。

本文主要用到加法秘密分享，茫然传输和基于RLWE的加法同态加密技术构造高效两方计算协议。本文调用了多个已有协议，理想功能如下：

Ideal Functionalities	Descriptions
$[\![\tilde{x}; f]\!] \leftarrow \mathcal{F}^f_{\mathrm{trunc}}([\![\tilde{x}; 2f]\!])$	Truncation [17, 29]
$[\![c?x : y]\!] \leftarrow \mathcal{F}_{\mathrm{mux}}([\![c]\!]^B, [\![x]\!], [\![y]\!])$	Multiplexer
$[\![1/\sqrt{\tilde{x}}; f]\!] \leftarrow \mathcal{F}_{\mathrm{rsqrt}}([\![\tilde{x}; f]\!]; f)$	Reciprocal Sqrt [39]
$[\![\mathbf{1}\{x < y\}]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}([\![x]\!], [\![y]\!])$	Less-then [56]
$[\![\mathbf{x}]\!] \leftarrow \mathcal{F}_{\mathrm{H2A}}(\mathsf{RLWE}(\mathbf{x}))$	HE to share [29, 57]

3. 安全两方矩阵乘法

针对矩阵乘法，本文提出了基于OLE的mOLE（Matrix OLE）来计算两个秘密分享矩阵乘法中涉及到的交叉项相乘。即，给定矩阵$\langle \mathbf{Q} \rangle$和$\langle \mathbf{V} \rangle$，计算$\langle \mathbf{Q}_0 \mathbf{V}_1 \rangle$和$\langle \mathbf{Q}_1 \mathbf{V}_0 \rangle$。

3.1 KRDY方案

KRDY方案提出了两种编码方法（π_{lhs}和π_{rhs}）来实现基于RLWE的高效矩阵乘法：

ParseError: KaTeX parse error: {split} can be used only in display mode.

使得

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

其中，多项式$\hat{q}, \hat{v}$的系数编码如下：

ParseError: KaTeX parse error: {split} can be used only in display mode.

而$\hat{q}$和$\hat{v}$的其他系数设为0。图示如下：

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

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

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

$$\Downarrow \hat{q} \cdot \hat{v} \bmod (X^{16} + 1, 2^5)$$

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

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

根据上述编码，当$1 \leq k_w \cdot m_w \cdot n_w \leq N$的时候，$\mathbf{U} \equiv_\ell \mathbf{Q} \cdot \mathbf{V}$可以通过计算$\hat{u} = \hat{q} \cdot \hat{v}$得到，且$\mathbf{U}[i,k] = \hat{u}[i \cdot m_w \cdot n_w + k \cdot m_w]$

而当$k_w \cdot m_w \cdot n_w > N$时，可以使用分块矩阵技术将矩阵分为不同的小块，分别处理。

3.2 密文打包与交叉

为了进一步优化通信和计算代价，本文用了密文打包（Ciphertext Packing）和密文交叉存储（Ciphertext Interleaving）两种技术来压缩密文。

3.2.1 密文打包

根据前文提到的KRDY方案，结果多项式中的系数有N个但是结果只编码在其中的$k \cdot n$个系数中。本文可以使用PackLWEs技术将多个多项式中的任意系数提取出来，然后编码到一个新的密文中，从而减少密文传输量。不过这样做的代价是增加了同态自同构（homomorphic automorphisms）计算时间。

3.2.2 密文交叉存储

本文提出了比PackLWEs快20倍的的密文交叉存储技术（Ciphertext Interleaving）。该技术基于同态加密下的Auto$(\hat{a}, N+1)$实现，核心思想是计算$\hat{a} +$ Auto$(\hat{a}, N+1)$来实现密文压缩。以$N =$

$8, \hat{a}(X)=\sum_{i=0}^{7} a_i x^i$为例，Auto$(\hat{a}, 9)$得到：
$\sum_{i=0}^{7} \hat{a}_i X^{i \cdot 9}=\sum_{i=0}^{3} a_{2i} X^{2i}-\sum_{i=0}^{3} a_{2i+1} X^{2i+1} \pmod{X^8+1}$
其中奇数位置的系数符号被反转，因此$\hat{a}+$ Auto$(\hat{a}, N+1)$使得奇数位置的系数完全抵消，偶数位置的系数变为原来的2倍。
更一般化，$\hat{a}+$ Auto$(\hat{a}, N/2^j+1)$会使得2^j的奇数倍位置的系数抵消，而偶数倍位置系数变成2倍（而其他$i \nmid 2^j$的位置系数为0）。因此，计算
$\hat{a}+$ Auto$(\hat{a}, N/2^j+1), j=0,1,\ldots,r-1$
最终只会得到位置在2^r倍数位置的系数。同时，上文的两种编码方案π_{lhs}和π_{rhs}可以选择适当的m_w以满足该技术位置间距要求。上述方法形式化为ZeroGap如下：

Input: $\hat{a} \in \mathbb{A}_{N,q}$ for an odd q and $1 \leq 2^r \leq N$.
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 $0 \equiv i \bmod 2^r$.

Compute $\hat{a}_0 := h \cdot \hat{a} \bmod q$ for $h \equiv 2^{-r} \bmod q$.
for $j \in [r]$ **do**
    $\hat{a}_{j+1} := \hat{a}_j + \mathsf{Auto}(\hat{a}_j, \frac{N}{2^j}+1)$.
end for
return $\hat{a}_r$.

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

基于ZeroGap，本文提出了IntrLeave，将2^r个多项式编码为一个多项式，其中输入多项式系数之间间距为2^r。

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: $\quad \hat{b}_j := \texttt{ZeroGap}(\hat{a}_j, 2^r)$ $\triangleright$ zero-out the 2^r gap
3: $\quad \hat{b}_j := \hat{b}_j \cdot X^j \in \mathbb{A}_{N,q}$ $\triangleright$ 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.

基于上述技术，本文提出了矩阵安全OLE协议如下：

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 $\mathtt{IntrLeave}$ 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 $\theta \in [\tilde{m}]$. $\triangleright$ 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

3. S and R jointly call $\hat{c}_{i,0}, \hat{c}_{i,1} \leftarrow \mathcal{F}_{\text{H2A}}(\mathbf{C}[i])$ 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.

3.4 其他优化

基于上述算法，本文基于开销模型选择了合适参数（例如$m_w = \sqrt{N}$），并提出了批量矩阵乘法，动态压缩策略，和对称加密和模数约减等技术进一步优化系统效率。

4. 基于RLWE的批量OLE

批量OLE（Batch OLE，bOLE）适用于两方各自持有一个私有向量，安全计算秘密分享下的矩阵Hamadard乘积。和之前的方法不一样，本文提出了基于更小的RLWE参数的带误差的批量OLE（bOLE with Error，bOLEe）。该方法会引入最低有效位1比特的误差，但是在定点数表示下该误差可以通过后续的截断操作去除。

和基于RLWE的bOLE类似，本文也需要利用SIMD技术来均摊开销。该技术需要使用掩码$\mathbf{r} \in \mathbb{Z}^N$保证安全性。而为了保证$\sigma$统计安全性，需要在$\mathbb{Z}^N_{2^{2\ell+\sigma}}$内采样$\mathbf{r}$，进而在RLWE中需要选择明文模数$t > 2^{2\ell+\sigma+1}$。为了避免这额外的$\sigma$比特开销，本文引入了如下两个函数：

ParseError: KaTeX parse error: {split} can be used only in display mode.

基于上述函数，本文可以选取与σ无关的掩码$\mathbf{r} \in \mathbb{Z}_t^N$。且当$t \nmid 2^\ell$时也可以直接实现模数为$2^\ell$的算术运算。具体协议如下：

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 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 $\|\mathbf{z} - \mathbf{x} \odot \mathbf{y} \bmod 2^\ell\|_\infty \leq 1$.

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 $[\![\hat{u}]\!] \leftarrow \mathcal{F}_{\text{H2A}}(\mathsf{ct})$ 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 $\mathsf{Down}(\mathtt{SIMD}^{-1}([\![\hat{u}]\!]_1))$.

可以证明，上述协议的结果和真实乘积$x \cdot y$之间的误差$e \in \{0, \pm 1\}$。

5. 安全激活函数协议

在Transformer模型中，需要调用大量的复杂激活函数，比如GeLU/SiLU和Softmax等。这些激活函数远远比ReLU等简单的激活函数复杂，因此需要设计高效的安全计算协议。

5.1 GeLU安全计算

GeLU函数的数学表达式如下：
$\mathsf{GeLU}(x) = 0.5x(1 + \tanh(\sqrt{(}2/\pi)(x + 0.044715x^3)))$.
本文提出了如下分度多项式函数Seg4GeLU来近似计算：

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

其中，$\epsilon=10^{-5}$，系数为：

ParseError: KaTeX parse error: {split} can be used only in display mode.

如下图所示，本文所提出的方法误差非常小：

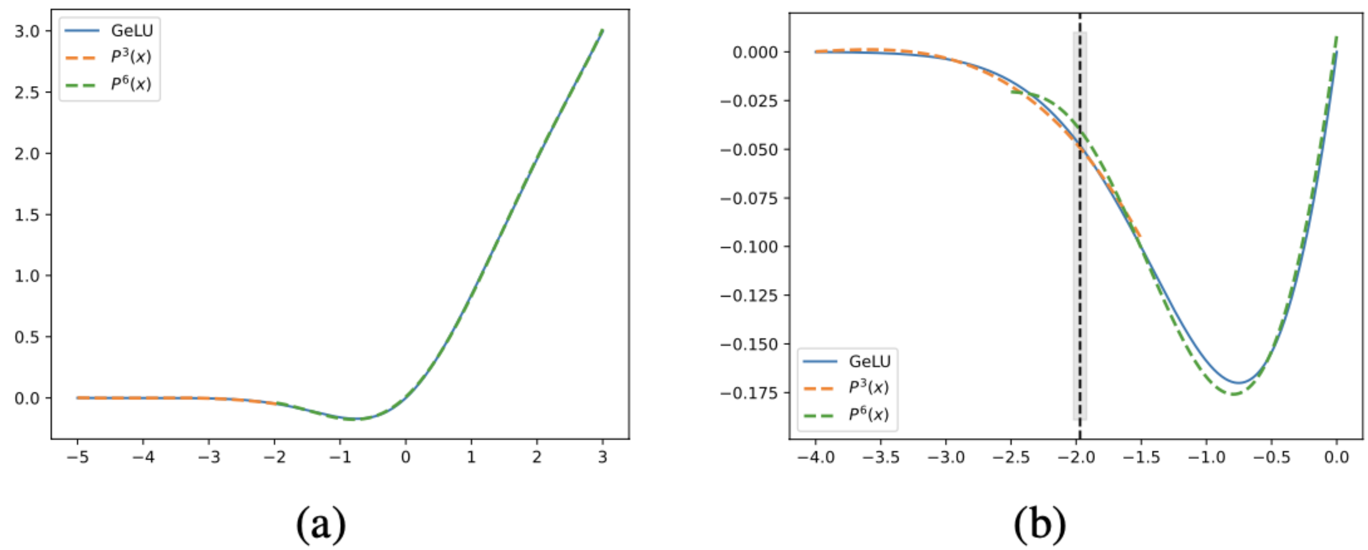

Figure 7: (Left) The maximum absolute error between Seg4GeLU and GeLU within the interval $[-5,3]$ is about 1.5×10^{-2}. (Right) We use a wider range for the polynomial fitting which gives $P^3(x)\approx P^6(x)$ for x around the pivot.

为了进一步优化安全协议的计算效率，本文在设计针对Seg4GeLU的安全协议的时候，进一步提出了如下优化：

1. **近似分段选择：** 为了减少选择分段的比较协议开销，本文舍去f'比特（其中$f'<f$），而只提取$\ell-f'$比特的输入的最高有效位来实现分段选择。基于本文提出的分段近似函数对于分段点具有非常好的容忍（即相邻的两段对于分段点附近的值计算结果非常接近），这样做引入的误差是非常小的。从而减少8%的通信开销。
2. **批量（近似）分段选择：** 在选择分段多项式的时候，两方需要交互计算$1\{2^{\ell-1}-\langle x\rangle_1<\langle x\rangle_0\}$，这需要调用$1$次$\binom{M}{1}-\text{OT}_2$。因此，一共需要调用3次$\binom{M}{1}-\text{OT}_2$。不过，由于这三次调用的选择比特都是一样的，可以将3次调用合并为一次$\binom{M}{1}-\text{OT}_6$，从而减少20%的计算时间。
3. **多项式计算优化：** Seg4GeLU中的多项式$P^3(x)$和$P^6(x)$比较稀疏，同时，本文在计算x的幂次的过程中复用中间结果来节省开销。

详细的计算协议如下所示：

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

$$
\begin{aligned}
&[\![b_0]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}([\![\tilde{x}]\!], -5) && \triangleright\ b_0 = \mathbf{1}\{\tilde{x} < -5\}\\
&[\![b_1]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}([\![\tilde{x}]\!], T) && \triangleright\ b_1 = \mathbf{1}\{\tilde{x} < -1.97\}\\
&[\![b_2]\!]^B \leftarrow \mathcal{F}_{\mathrm{lt}}(3, [\![\tilde{x}]\!]) && \triangleright\ b_2 = \mathbf{1}\{3 < \tilde{x}\}
\end{aligned}
$$

Locally sets $[\![z_0]\!]^B_l := [\![b_0]\!]^B_l \oplus [\![b_1]\!]^B_l$, $[\![z_1]\!]^B_l := [\![b_1]\!]^B_l \oplus [\![b_2]\!]^B_l \oplus l$ and $[\![z_2]\!]^B_l := [\![b_2]\!]^B_l$. Note $z_0 = \mathbf{1}\{-5 < \tilde{x} \leq -1.97\}$, $z_1 = \mathbf{1}\{-1.97 < \tilde{x} \leq 3\}$, and $z_2 = \mathbf{1}\{3 < \tilde{x}\}$.

4: 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 $\lfloor \epsilon \cdot 2^f \rfloor$.

5.2 Softmax安全计算

给定向量$\mathbf{x}$，Softmax的安全计算中需要首先计算最大值$\bar{x} = \mathrm{Max}(\mathbf{x})$，进而计算 $\mathrm{Softmax}(\mathbf{x})[i] = \frac{\exp(\mathbf{x}[i]-\bar{x})}{\sum_i \exp(\mathbf{x}[i]-x)}$

其中难点在于高效计算exp函数。本文提出了如下近似计算方法：

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

由于本文$f = 18$，因此满足$\exp(T_{\exp}) < 2^{-18}$即可。故令$T_{\exp} = -13$。

本文实验中，$(n = 6, T_{\exp} = -13)$可以令误差小于2^{-10}。在协议实现中$\frac{x}{2^n}$可以调用截断协议截断最后n比特，而幂次计算可以多次调用安全平方协议。另外，由于Softmax计算紧跟在矩阵乘法之后，所以截断n比特操作和乘法中的截断f比特操作可以合并为截断$n + f$比特操作节省开销。具体协议如下所示：

Algorithm 4 Private Softmax Protocol Π_{softmax}

Input: $[\![\tilde{\mathbf{x}}; 2f]\!] \in \mathbb{Z}_{2^\ell}^d$ with double-precision.

Output: $[\![\text{softmax}(\tilde{\mathbf{x}}); f]\!] \in \mathbb{Z}_{2^\ell}^d$. See (4) for definition.

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

6. 实验评估

本文对模型性能、各个协议模块、和end-to-end安全预测的开销都进行了详细的开销测试。部分实验结果如下：

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

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)	$\Pi_{\mathrm{mOLE}}(\mathbf{X}, \mathbf{Y})$	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

	$\Pi_{\mathrm{bOLE}}(\mathbf{x}, \mathbf{y})$	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

	$\Pi_{\mathrm{GeLU}}(\mathbf{x})$	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

$\lvert\mathbf{W}\rvert$	$\Pi_{\mathrm{softmax}}(\mathbf{W})$	Comm.	LAN	WAN
	[55]	1697.86MB	16.39s	40.84s

$(960, 180)$	[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 $\binom{M}{1}$-$\mathtt{OT}_2$ without approximated less-than in this run.
‡ We call 1 $\binom{M}{1}$-$\mathtt{OT}_6$ without approximated less-than in this run.

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 ($B = 12$, $D = 768$, $H = 12$)				LLaMA-7B ($B = 32$, $D = 4096$, $H = 32$)			
		#Calls	Time (sec)	Sent (MB)	Recv (MB)	#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

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)		Comm.
		LAN	WAN	(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

可以看到，BumbleBee取得了和明文相当的准确率，且在通信和计算时间等方面比现有工作取得了长足的提升。