SIRNN: A Math Library for Secure RNN Inference

0. Introduction

之前的安全多方计算协议多是集中在uniform bitwidth下的，即在整个计算过程中所有的整数bitwdith是一致的。但是有些中间计算过程如果在明文下计算仅仅使用较短的bitwidth即可。而且减少比特位长也可以提升安全计算的性能、减少通信开销。本文基于此，

1. 设计了一系列适用于non-uniform bitwidth计算的两方计算协议；
2. 基于上述协议，进一步优化计算神经网络中的非线性函数，并提升了效率和精度；
3. 并对精度进行了分析和实现验证。

本文的工作是基于之前CrypTFlow2继续展开的，在CrypTFlow2作者优化了两方计算比较函数（$\mathcal{F}_{\text{Mill}}(x, y)$）的计算，使用递归的思想基于OT，节省了大量的开销。本文所有协议的理论开销如下。

Protocol	Comm. (bits)	Rounds
$\Pi^{m,n}_{\text{ZExt}}$ & $\Pi^{m,n}_{\text{SExt}}$	$\lambda(m+1)+13m+n$	$\log m+2$
$^{\star}\Pi^{m,n}_{\text{ZExt}}$ & $^{\star}\Pi^{m,n}_{\text{SExt}}$	$2\lambda-m+n+2$	4
$\Pi^{\ell,s}_{\text{LRS}}$ & $\Pi^{\ell,s}_{\text{ARS}}$	$\lambda(\ell+3)+15\ell+s+20$	$\log \ell+3$
$^{\star}\Pi^{\ell,s}_{\text{LRS}}$ & $^{\star}\Pi^{\ell,s}_{\text{ARS}}$	$\lambda(s+3)+\ell+15s+2$	$\log s+2$
$\Pi^{\ell,s}_{\text{TR}}$	$\lambda(s+1)+\ell+13s$	$\log s+2$
$\Pi^{\ell,s}_{\text{DivPow2}}$	$\lambda(\ell+7s/4+4)+16\ell+23s-5$	$\log \ell+4$
$\Pi^{m,n}_{\text{UMult}}$ & $\Pi^{m,n}_{\text{SMult}}$	$\lambda(3\mu+\nu+4)+2\mu\nu+\mu^2+17\mu+16\nu$	$\log \nu+2$
$^{\star}\Pi^{m,n}_{\text{UMult}}$ & $^{\star}\Pi^{m,n}_{\text{SMult}}$	$\lambda(2\mu+6)+2\mu\nu+\mu^2+3\mu+2\nu+4$	4
$\Pi^{\ell,d}_{\text{DigDec}}$	$(\ell/d-1)(\lambda(d+2)+15d+20)$	$\log d+\ell/d+1$
$\Pi^{\ell,d}_{\text{MSNZB}}$	$(\ell/d-1)(\lambda(d+8)+2^d(\iota+1)+15d+2\iota+60)+6\lambda+2^d(\iota+1)+\ell^2+2\iota$	$\log d+2\ell/d+7$

TABLE V: Exact communication and round expressions for our building blocks, assuming that the cost of Π^{ℓ}_{Mill} and $\Pi^{\ell}_{\text{Mill\&Eq}}$ is $\lambda\ell+14\ell$ bits. $\mu=\min(m,n), \nu=\max(m,n)$, and $\star$ denotes the variant of the protocol in which the MSBs of the inputs are already known in the clear. In case the MSBs are known in the shared form, the additional cost is just $\lambda+2$ bits per input.

本文使用的Notation和基础概念较多，但是大多数已经介绍过，在这里不再展开，请参考原文。

1. Zero Extension & Signed Extension

在基于加法秘密分享的两方计算中，对于环 $\mathbb{Z}_M$ 中的 m-bit 的数 $x \in \mathbb{Z}_M$，如果将其转化到 n-bit 的环 $\mathbb{Z}_N$ 下（其中 $n > m$），则需要x进行拓展（Extension）。对于无符号数（unsigned）和有符号数（signed），需要区别对待。

1.1 Zero Extension (ZExt)

对于无符号数，需要进行 0-拓展（Zero Extension）：$y = \mathrm{ZExt}(x, m, n) \in \mathbb{Z}_N$。即两方（$P_0$和$P_1$）以秘密分享的形式输入$\langle x \rangle^m$，协议输出结果为$\langle y \rangle^n$，满足$\mathrm{uint}(y) = \mathrm{uint}(x)$，其中$\mathrm{uint}(x)$表示$x$在$\mathbb{Z}$下的值。

对于$\langle x \rangle^m \in \mathbb{Z}_M$，在$\mathbb{Z}$下有

$$x = \langle x \rangle_0^m + \langle x \rangle_1^m - w \cdot M \tag{1}$$

其中，$w = (\langle x \rangle_0^m + \langle x \rangle_1^m \geq M)$，而$\langle x \rangle_0^m + \langle x \rangle_1^m \geq M$可表示为$\mathrm{wrap}(\langle x \rangle_0^m, \langle x \rangle_1^m, M)$。

$\langle x \rangle_b^m$都可以在本地从$\mathbb{Z}_M$转化到$\mathbb{Z}_N$，而对于$w \cdot M$，则涉及到w。首先，ZExt调用$\mathcal{F}_{\mathrm{wrap}}^m(\langle x \rangle_0^m, \langle x \rangle_1^m)$返回$w$的Boolean-shares。又因为$N/M = 2^{n-m}$，所以可以调用$\mathbb{F}_{\mathrm{B2A}}^{n-m}(\langle w \rangle^B)$将$\langle w \rangle^B$转化到环$\mathbb{Z}_{2^{n-m}}$上即可。这样处理，在$\mathbb{Z}$上则有

$$w = \langle w \rangle_0^{n-m} + \langle w \rangle_1^{n-m} - 2^{n-m} \cdot \mathrm{wrap}(\langle w \rangle_0^{n-m}, \langle w \rangle_1^{n-m}, 2^{n-m})$$

从而，有

$$M *_n w = M *_n (\langle w \rangle_0^{n-m} + \langle w \rangle_1^{n-m})$$

其中，$x *_n y = x \cdot y \mod N$。

如此，对于

$$y = \sum_{b=0}^{1} (\langle x \rangle_b^m - M \cdot \langle w \rangle_b^{n-m}) \mod N$$

有 $x \mod N = y$。
综上所述，协议$\Pi_{\mathrm{ZExt}}^{m,n}$如下：

Algorithm 1 Zero Extension, $\Pi_{\mathsf{ZExt}}^{m,n}$:

Input: P_0 & P_1 hold $\langle x \rangle^m$.
Output: P_0 & P_1 get $\langle y \rangle^n$ for $y = \mathsf{ZExt}(x, m, n)$.

1: P_0 & P_1 invoke $\mathcal{F}_{\mathsf{Wrap}}^m(\langle x \rangle_0^m, \langle x \rangle_1^m)$ and learn $\langle w \rangle^B$.
2: P_0 & P_1 invoke $\mathcal{F}_{\mathsf{B2A}}^{n-m}(\langle w \rangle^B)$ and learn $\langle w \rangle^{n-m}$.
3: For $b \in \{0, 1\}$, P_b outputs $\langle y \rangle_b^n = \langle x \rangle_b^m - M *_n \langle w \rangle_b^{n-m}$.

1.2 Signed Extension (SExt)

对于有符号数，有

$$\text{int}(x) = x' - 2^{m-1}, \text{ for } x' = x + 2^{m-1} \mod M \tag{2}$$

有如下两种情况：

1. $x < 2^{m-1}$：则有$x + 2^{m-1} < M$，从而在$\mathbb{Z}$上有$x' = x + 2^{m-1}$。进一步，则有$\text{int}(x) = x = \text{uint}(x') - 2^{m-1}$；
2. $x \geq 2^{m-1}$：则有$x + 2^{m-1} \geq M$，从而在$\mathbb{Z}$上有 $x' = x + 2^{m-1} - 2^m = x - 2^{m-1}$。进一步，又因为$\text{int}(x) = x - 2^m = (x - 2^{m-1}) - 2^{m-1} = x' - 2^{m-1}$，所以也可得在$\mathbb{Z}$上有 $\text{int}(x) = \text{uint}(x') - 2^{m-1}$。

故，$\text{SExt}(x, m, n) = \text{ZExt}(x', m, n) - 2^{m-1}$。

2. Truncation

对于环$\mathbb{Z}_L$内表示的无符号数和有符号数，首先分别考虑两种右移（Right Shift）：逻辑右移（Logical Right Shift，$\gg_L$）和算术右移（Arithmetic Right Shift，$\gg_A$）。在此基础上，构造截断并缩减（Truncate & Reduce x by s-bits，$\text{TR}(x, s)$），即截断$x \in \mathbb{Z}_L$的末s-bits并使得截断结果在环$\mathbb{Z}_{2^{\ell-s}}$种。最后，构造了除数为2^m的除法。

2.1 Logical Right Shift

对于$x \in \mathbb{Z}_L$，有$x = \langle x \rangle_0^\ell + \langle x \rangle_1^\ell \mod L$。记$\langle x \rangle_b^\ell = u_b || v_b$，其中$u_b \in \{0,1\}^{\ell-s}$，$v_b \in \{0,1\}^s$。那么根据公式(1)有

$$x \gg_L s = u_0 + u_1 - 2^{\ell-s} \cdot \text{wrap}(\langle x \rangle_0^\ell, \langle x \rangle_1^\ell, L) + \text{wrap}(v_0, v_1, 2^s)$$

一种简单的方法是利用分别计算$\text{wrap}(\langle x \rangle_0^\ell, \langle x \rangle_1^\ell, L)$和$\text{wrap}(v_0, v_1, 2^s)$。为了减少开销，本文利用如下引理：

引理 1：令$x \in \mathbb{Z}_L$，$\langle x \rangle$是x的秘密分享，$\langle x \rangle_b^\ell = u_b || v_b$，其中$u_b \in \{0,1\}^{\ell-s}$，$v_b \in \{0,1\}^s$。令$c = \text{wrap}(v_0, v_1, 2^s)$，$d = \text{wrap}(u_0, u_1, 2^{\ell-s})$，$e = 1\{u_0 + u_1 \mod 2^{\ell-s} = 2^{\ell-s} - 1\}$，且$w = \text{wrap}(\langle x \rangle_0^\ell, \langle x \rangle_1^\ell, L)$，则有ParseError: KaTeX parse error: Undefined control sequence: \and at position 12: w=d\oplus(c\and e)。

证明：令$x_b = \langle x \rangle_b^\ell$，在$\mathbb{Z}$上，有$x_b = u_b \cdot 2^s + v_b$和

$$\begin{aligned} x_0 + x_1 &= (v_0 + v_1) + (u_0 + u_1) \cdot 2^s \\ &= (v_0 + v_1 - c \cdot 2^s) + (u_0 + u_1 - d \cdot 2^{\ell-s}) \cdot 2^s + c \cdot 2^s + d \cdot L \\ &= v' + (u' + c) \cdot 2^s + d \cdot L \end{aligned}$$

令$w' = 1\{u' + c > 2^{\ell-s} - 1\}$，则有

$$x_0 + x_1 = v' + (u' + c - w' \cdot 2^{\ell-s}) \cdot 2^s + (d + w') \cdot L \tag{3}$$

1. 如果$d = 1$，那么$u_0 + u_1 \geq 2^{\ell-s}$即$u_0 + u_1 > 2^{\ell-s} - 1$，从而$e = 0$且$u' = u_0 + u_1 - 2^{\ell-s}$。又因为$u_0, u_1 \leq 2^{\ell-s} - 1$，则有$u' \leq 2^{\ell-s} - 2$。因此，$w' = 0$（因为$c \in \{0, 1\}$）；
2. 如果$d = 0$，那么$u' = u_0 + u_1 \leq 2^{\ell-s} - 1$。如此一来，$w' = 1$当且仅当$u' = 2^{\ell-s} - 1$（即$e = 1$）且$c = 1$。否则$w' = 0$。

根据上述两种情况，可以得出d和w'最多有一项为1。所以，我们有
ParseError: KaTeX parse error: Undefined control sequence: \and at position 69: …s + (d\oplus (c\and e)\cdot L
又因为$v' < 2^s$且$u' + c - w' \cdot 2^{\ell-s} < 2^{\ell-s}$，所以$v' + (u' + c - w' \cdot 2^{\ell-s}) \cdot 2^s < L$，故
ParseError: KaTeX parse error: Undefined control sequence: \and at position 13: w=d\oplus (c\and e)。■

基于引理 1，则可以在计算c，e，和d，并得到w。而c，e，和d的计算都只涉及到s和$\ell - s$bits的参数，并且d和e可以同时计算（开销仅比单独计算d增加很少），因此计算量可以大大降低整体开销。具体协议如下：

Algorithm 2 Logical Right Shift, $\Pi^{\ell,s}_{\mathsf{LRS}}$:

Input: P_0 & P_1 hold $\langle x \rangle^\ell$.
Output: P_0 & P_1 get $\langle x \gg_L s \rangle^\ell$.

1: For $b \in \{0, 1\}$, P_b parses $\langle x \rangle^\ell_b$ as an ℓ-bit string $u_b || v_b$, where $u_b \in \{0, 1\}^{\ell-s}$ and $v_b \in \{0, 1\}^s$.
2: P_0 & P_1 invoke $\mathcal{F}^s_{\mathsf{Wrap}}(v_0, v_1)$ and learn $\langle c \rangle^B$.
3: P_0 & P_1 invoke $\mathcal{F}^{\ell-s}_{\mathsf{Wrap\&All1s}}(u_0, u_1)$ and learn $\langle d \rangle^B || \langle e \rangle^B$.
4: P_0 & P_1 invoke $\mathcal{F}_{\mathsf{AND}}(\langle c \rangle^B, \langle e \rangle^B)$ and learn $\langle t \rangle^B$.
5: For $b \in \{0, 1\}$, P_b sets $\langle w \rangle^B_b = \langle d \rangle^B_b \oplus \langle t \rangle^B_b$.
6: P_0 & P_1 invoke $\mathcal{F}^\ell_{\mathsf{B2A}}(\langle c \rangle^B)$ and learn $\langle c \rangle^\ell$.
7: P_0 & P_1 invoke $\mathcal{F}^s_{\mathsf{B2A}}(\langle w \rangle^B)$ and learn $\langle w \rangle^s$.
8: For $b \in \{0, 1\}$, P_b outputs $u_b - 2^{\ell-s} *_\ell \langle w \rangle^s_b + \langle c \rangle^\ell_b$.

2.2 Arithmetic Right Shift

对于有符号数，则需要算术右移（$\gg_A$）。从SExt，对于有符号数，在$\mathbb{Z}$上有$x' = x + 2^{\ell-1} \mod L$。因此$x \gg_A s = x' \gg_L s - 2^{\ell-s-1}$。如此一来，可以在$\gg_L$的基础上构造$\gg_A$而不引起额外的开销。

2.3 Truncate and Reduce

基于对$\gg_L$ 和 $\gg_A$的描述，有$\langle \mathrm{TR}(x,s)\rangle^{\ell-s} = u_0 + u_1 + \mathrm{wrap}(v_0, v_1, 2^s)$。这是因为$2^{\ell-s} *_\ell w \mod 2^{\ell-s} = 0$。因此，在计算$\mathrm{TR}(x,s)$的时候可以不用计算算法2中关于只和$w$计算有关的步骤（step3-7）。

2.4 Division by power-of-2

$\mathcal{F}^{\ell,s}_{DivPow2}(x)$对于输入$x$，计算$\langle z\rangle^\ell$满足

1. 如果 $z < 0$，$z = \lceil \mathrm{int}(x)/2^s \rceil \mod L$;
2. 如果 $z \geq 0$，$z = \lfloor \mathrm{int}(x)/2^s \rfloor \mod L$。

对于上述两种情况，如果忽略$\lceil\cdot\rceil$和$\lfloor\cdot\rfloor$则Divsion by poer-of-2可以看作$\gg_A$。而$\lceil\cdot\rceil$和$\lfloor\cdot\rfloor$的影响，则是使得计算结果偏向"0"。令$m_x = 1\{x \geq 2^{\ell-1}\}$，$c = 1\{x \mod 2^s = 0\}$，则有ParseError: KaTeX parse error: Undefined control sequence: \and at position 37: …\gg_A s) + m_x \and c。即当$m_x = 1 \Leftrightarrow z < 0$，$z$取 $\lceil\cdot\rceil$；否则（$z \geq 0$），z取$\lfloor\cdot\rfloor$。

3. Multiplication with non-uniform bidwidths

3.1 Unsigned Multiplication $\mathcal{F}^{m,n}_{\mathrm{UMult}}$

$\mathcal{F}^{m,n}_{\mathrm{UMult}}$输入两个位宽不一致的无符号整数$\langle x\rangle^m$和$\langle y\rangle^n$，计算输出$\langle z\rangle^\ell$满足$z = x *_\ell y$且$\ell = m + n$。之前的工作都集中在$\ell = m = n$的情况。一种简单的处理方法是先对$x$和$y$做拓展（都拓展到$\ell$-bits），然后再做乘法。本文的方法比这种简单的方法高效$1.5\times$。对于x和y，在$\mathbb{Z}$上有

$$\begin{aligned}\mathrm{uint}(x)\cdot\mathrm{uint}(y) &= (x_0 + x_1 - 2^m w_x)\cdot(y_0 + y_1 - 2^n w_y)\\ &= x_0y_0 + x_1y_1 + x_0y_1 + x_1y_0 - 2^m w_x y - 2^n w_y x + 2^\ell w_x w_y\end{aligned}$$

其中，x_by_b可以在P_b本地计算，$2^\ell w_x w_y$在 $\mod L$下自动消除。而$w_x y$和$w_y x$可以通过$\mathcal{F}_{\mathrm{MUX}}$计算，且$w_x$和$w_y$可以通过wrap计算。关键难点在于计算$x_by_{1-b}$。对于$x_by_{1-b}$的计算，本文利用$\mathrm{COT}$构造了类似计算Beaver三元组中交叉项的方法 $\mathcal{F}^{m,n}_{\mathrm{CrossTerm}}: \mathbb{Z}_M \times \mathbb{Z}_N \to \mathbb{Z}_L \times \mathbb{Z}_L$ 。不同的是，在计算x_by_{1-b}的时候，令数据比特位短的一方作为Receiver而数据比特位长的一方作为Sender。不妨令$m < n$，则P_b作为Receiver，其中x_b的每一比特作为每次COT的选择比特，而P_{1-b}输入y_{1-b}的末尾$\ell - i$比特作为第i次$\mathrm{COT}_{\ell-\mathrm{i}}$的输入。实现$\mathcal{F}^{m,n}_{\mathrm{CrossTerm}}$的协议如下：

Algorithm 4 Cross Term Multiplication, $\Pi^{m,n}_{\mathsf{CrossTerm}}$:

Input: P_0 holds $x \in \mathbb{Z}_M$ and P_1 holds $y \in \mathbb{Z}_N$, where $m \leqslant n$.
Output: P_0 & P_1 get $\langle z \rangle^{\ell}_b$, where $z = x *_\ell y$ and $\ell = m + n$.

1: P_0 parses x as an m-bit string $x = x_{m-1}||\cdots||x_0$, where $x_i \in \{0,1\}$.
2: **for** $i = \{0, \ldots, m-1\}$ **do**
3: P_0 & P_1 invoke $\binom{2}{1}$-$\mathsf{COT}_{\ell-i}$, where P_0 is the sender with input x_i and P_1 is the receiver with input y, and learn $\langle t_i \rangle^{\ell-i}$.
4: **end for**
5: For $b \in \{0,1\}$, P_b sets $\langle z \rangle^{\ell}_b = \sum_{i=0}^{m-1} 2^i \cdot \langle t_i \rangle^{\ell-i}_b$.

上述协议Sender和Receiver角色写反了。实现$\mathcal{F}^{m,n}_{\mathrm{UMult}}$的协议如下：

Algorithm 3 Unsigned Multiplication, $\Pi^{m,n}_{\mathsf{UMult}}$:

Input: P_0 & P_1 hold $\langle x \rangle^m$ and $\langle y \rangle^n$.
Output: P_0 & P_1 get $\langle z \rangle^{\ell}$, where $z = x *_\ell y$ and $\ell = m + n$.

1: For $b \in \{0,1\}$, let $x_b = \langle x \rangle^m_b$ and $y_b = \langle y \rangle^n_b$.
2: P_0 and P_1 invoke the following functionalities.
3: $\mathcal{F}^{m,n}_{\mathsf{CrossTerm}}(x_0, y_1)$ and learn $\langle c \rangle^{\ell}$.
4: $\mathcal{F}^{n,m}_{\mathsf{CrossTerm}}(y_0, x_1)$ and learn $\langle d \rangle^{\ell}$.
5: $\mathcal{F}^{m}_{\mathsf{Wrap}}(x_0, x_1)$ to learn $\langle w_x \rangle^{B}$.
6: $\mathcal{F}^{n}_{\mathsf{Wrap}}(y_0, y_1)$ to learn $\langle w_y \rangle^{B}$.
7: $\mathcal{F}^{m}_{\mathsf{MUX}}(\langle w_y \rangle^{B}, \langle x \rangle^m)$ to learn $\langle g \rangle^m$.
8: $\mathcal{F}^{n}_{\mathsf{MUX}}(\langle w_x \rangle^{B}, \langle y \rangle^n)$ to learn $\langle h \rangle^n$.
9: P_b outputs $x_b *_\ell y_b + \langle c \rangle^{\ell}_b + \langle d \rangle^{\ell}_b - N *_\ell \langle g \rangle^m_b - M *_\ell \langle h \rangle^n_b$ for $b \in \{0,1\}$.

3.2 Signed Multiplication $\mathcal{F}^{m,n}_{\mathrm{SMult}}$

对于有符号数，基于公式（2）得到 $x' = x + 2^{m-1} \mod M$和 $y' = y + 2^{n-1} \mod N$ 满足 $x' = x'_0 + x'_1 \mod M$和 $y' = y'_0 + y'_1 \mod N$。如此，在$\mathbb{Z}$上则有

$$\begin{aligned}\mathrm{int}(x) \cdot \mathrm{int}(y) &= (x' - 2^{m-1}) \cdot (y' - 2^{n-1}) \\ &= x' \cdot y' - 2^{m-1} y' - 2^{n-1} x' + 2^{m+n-2} \\ &= x' \cdot y' - 2^{m-1}(y'_0 + y'_1 - 2^m w_{y'}) - 2^{n-1}(x'_0 + x'_1 - 2^m w_{x'}) + 2^{m+n-2}\end{aligned}$$

其中$w_{x'} = \mathrm{wrap}(x'_0, x'_1, M)$，$w_{y'} = \mathrm{wrap}(y'_0, y'_1, N)$。

对于x'和y'，其秘密分享值可以在本地计算。因此，除了$z_1 = x'y'$和$z_2 = 2^{\ell-1}(w_{x'} + w_{y'})$，其余所有项可以在本地计算。对于$z_1$，可以调用$\mathcal{F}^{m,n}_{\text{UMult}}$计算。对于$z_2$，由于$2^{\ell-1}w_{x'} = 2^{\ell-1}(\langle w_{x'}\rangle^B_0 + \langle w_{x'}\rangle^B_1 - 2\langle w_{x'}\rangle^B_0\langle w_{x'}\rangle^B_1)$中最后一项在 $\bmod L$中可以消除，因此$\langle z_2\rangle = 2^{\ell-1}(\langle w_{x'}\rangle^B + \langle w_{y'}\rangle^B)$。而$w_{x'}$和$w_{y'}$可以在$\mathcal{F}^{m,n}_{\text{UMult}}$计算得到，因此并不会有额外开销。

3.3 Matrix Multiplication and Convolutions

对于矩阵乘法和卷积（卷积可以展开为矩阵乘法）AB，其中$A \in \mathbb{Z}_M^{d_1\times d_2}$，$B \in \mathbb{Z}_N^{d_2\times d_3}$，可以首先调用 $\mathcal{F}^{m,n}_{\text{UMult}}$或者$\mathcal{F}^{m,n}_{\text{SMult}}$，然后进行$d_2$次加法。但是在$\mathbb{Z}_L$ $L = 2^\ell, \ell = m + n$内执行加法会造成溢出。为了防止溢出，需要将数据额外拓展$e = \lceil \log d_2 \rceil$-bits 。简单的直接将所有element-wise相乘的中间结果拓展需要进行$d_1d_2d_3$次拓展。为了减少开销，在进行$\mathcal{F}_{\text{CrossTerm}}$的计算的时候，将比特位长的一方数据拓展$e$-bits，如此则在不增加开销的条件下将中间结果自然拓展到$\mathbb{Z}_{2^{m+n+e}}$上，防止溢出。具体协议如下。

Algorithm 5 Unsigned Matrix Multiplication, $\Pi^{m,n,d_1,d_2,d_3}_{\mathsf{UMatMul}}$:

Input: P_0 & P_1 hold $\langle X\rangle^m$ and $\langle Y\rangle^n$, where $X \in \mathbb{Z}_M^{d_1\times d_2}$, $Y \in \mathbb{Z}_N^{d_2\times d_3}$ and $m \leqslant n$.

Output: P_0 & P_1 get $\langle Z\rangle^\ell$, where $Z = X \boxtimes_\ell Y$, $\ell = m + n + e$ and $e = \lceil \log d_2 \rceil$.

1: P_0 & P_1 invoke $\mathcal{F}^{n,n+e}_{\mathsf{ZExt}}(\langle Y\rangle^n)$ and learn $\langle Y'\rangle^{n'}$.
2: For $b \in \{0, 1\}$, let $X_b = \langle X\rangle^m_b$ and $Y'_b = \langle Y'\rangle^{n'}_b$.
3: P_0 and P_1 invoke the following functionalities.
4: $\quad\mathcal{F}^{m,n',d_1,d_2,d_3}_{\mathsf{MatCrossTerm}}(X_0, Y'_1)$ and learn $\langle C\rangle^\ell$.
5: $\quad\mathcal{F}^{n',m,d_3,d_2,d_1}_{\mathsf{MatCrossTerm}}({Y'}^T_0, X^T_1)$ and learn $\langle D\rangle^\ell$.
6: $\quad\mathcal{F}^{m}_{\mathsf{Wrap}}(X_0, X_1)$ to learn $\langle W_X\rangle^B$.
7: $\quad\mathcal{F}^{n'}_{\mathsf{Wrap}}(Y'_0, Y'_1)$ to learn $\langle W_{Y'}\rangle^B$.
8: $\quad\mathcal{F}^{m,d_3,d_2,d_1}_{\mathsf{BitMatMul}}(\langle W^T_{Y'}\rangle^B, \langle X^T\rangle^m)$ to learn $\langle G\rangle^m$.
9: $\quad\mathcal{F}^{n',d_1,d_2,d_3}_{\mathsf{BitMatMul}}(\langle W_X\rangle^B, \langle Y'\rangle^{n'})$ to learn $\langle H\rangle^{n'}$.
10: P_b outputs $X_b \boxtimes_\ell Y'_b + \langle C\rangle^\ell_b + \langle D^T\rangle^\ell_b - 2^{n'} *_\ell \langle G^T\rangle^m_b - 2^m *_\ell \langle H\rangle^{n'}_b$ for $b \in \{0, 1\}$.

3.4 Multiply and Truncate

$\mathcal{F}^{m,n,\ell,s}_{\text{SMultTR}}$首先调用$\mathcal{F}^{m,n}_{\text{SMult}}$得到$z = \text{int}(x) *_\ell \text{int}(y)$，进一步调用$\text{TR}$得到$z' = \text{TR}(z, s)$。

4. Digit Decomposition and MSNZB

4.1 Digit Decomposition

$\mathcal{F}_{\text{DigDec}}^{\ell,\{d_i\}_{i\in[c]}}$将$\ell$-bit的$\langle x\rangle^{\ell}$分解为$c$个子串$\langle z_{c-1}\rangle^{d_{c-1}},...,\langle d_0\rangle^{d_0}$，其中$z_i$的长度为$d_i$，满足$x = z_{c-1}||...||z_0$。首先考虑$d \mid \ell$ 的情况，此时分解结果为$c = \ell/d$个长度均为d的子串。对于每一个z_i，需要额外计算其后低位的串对其的进位（carry）。具体来说，对$\langle x\rangle_b^{\ell} = y_{b,c-1}||...||y_{b,0}$，其中$y_{b,i} \in \{0,1\}^d$，$i \in [c]$。令$Y_{b,i} = y_{b,i}||...||y_{b,0}$。那么$z_i = y_{0,i} + y_{1,i} + carry_i \mod 2^d$，其中$carry_i = Y_{0,i-1} + Y_{1,i-1} \geq 2^{id}$。利用wrap计算$carry_i$，并递归利用引理 1，可以计算得到所有的$z_i$。

Algorithm 6 Digit Decomposition, $\Pi_{\text{DigDec}}^{\ell,d}$:

Input: P_0 & P_1 hold $\langle x\rangle^{\ell}$ s.t. $c = \ell/d$.
Output: P_0 & P_1 get $\{\langle z_i\rangle^d\}_{i\in[c]}$ s.t. $x = z_{c-1}||\ldots||z_0$.

1: For $b \in \{0,1\}$, P_b parses $\langle x\rangle_b^{\ell}$ as an ℓ-bit string $y_{b,c-1}||\ldots||y_{b,0}$ s.t. $y_{b,i} \in \{0,1\}^d$ for all $i \in [c]$.
2: For all $i \in \{0,\ldots,c-2\}$, P_0 & P_1 invoke $\mathcal{F}_{\text{Wrap\&All1s}}^{d}(y_{b,i}, y_{b,1})$ and learn $\langle w_i\rangle^B||\langle e_i\rangle^B$.
3: For $b \in \{0,1\}$, P_b sets $\langle u_0\rangle_b^B = 0$ and $\langle z_0\rangle_b^d = y_{b,0}$.
4: **for** $i \in \{1,\ldots,c-1\}$ **do**
5: P_0 & P_1 invoke $\mathcal{F}_{\text{AND}}(\langle u_{i-1}\rangle^B, \langle e_{i-1}\rangle^B)$ to learn $\langle v_{i-1}\rangle^B$.
6: For $b \in \{0,1\}$, P_b sets $\langle u_i\rangle_b^B = \langle v_{i-1}\rangle_b^B \oplus \langle w_{i-1}\rangle_b^B$.
7: P_0 & P_1 invoke $\mathcal{F}_{\text{B2A}}^{d}(\langle u_i\rangle^B)$ and learn $\langle u_i\rangle^d$.
8: For $b \in \{0,1\}$, P_b sets $\langle z_i\rangle_b^d = y_{b,i} + \langle u_i\rangle_b^B$.
9: **end for**

需要注意的是，算法中的$u_i = carry_i$，且step-8中 $\langle z_i\rangle_b^d = y_{b,i} + \langle u_i\rangle_b^d$。对于 $d \nmid \ell$的情况，可以类似处理。

4.2 MSNZB

对于ℓ-bit 整数x，$\text{MSNZB}(x)$返回二进制下x最高非0位的索引，即如果$\text{MSNZB}(x) = k \in [\ell]$当且仅当$x_k = 1$且$x_j = 0, j > k$（$2^k \leq x < 2^{k+1}$）。对于0，$\text{MSNZB}(x) = 0$。$\mathcal{F}_{\text{MSNZB}}^{\ell}$对$\langle x\rangle^{\ell}$输出$\{\langle z_i\rangle^B\}_{i\in[\ell]}$满足$z_i = 1$ 其中 $\text{MSNZB}(x) = 1$，而 $z_j = 0, j \neq i$。$\mathcal{F}_{\text{MSNZB}}^{\ell}$首先调用$\mathcal{F}_{\text{DigDec}}^{\ell,d}$将$\ell$-bit整数分解为$c$个$d$-bit的小整数$\{y_i\}_{i\in[c]}$。然后对每一个$y_i$求MSNZB。最终$\text{MSNZB}(x) = \text{MSNZB}(y_i) + i \cdot d$，其中$y_i \neq 0$ 且 $y_j = 0, j > i$。因此，还需要判断$y_i = 0, i \in [c]$。

具体来说，令$\iota = \log \ell$，$\mathcal{F}_{\text{MSNZB-P}}^{d,\ell,i}$对于输入$\langle y_i\rangle^d$输出$\langle u\rangle^{\iota}$满足$2^{u-id} \leq y_i < 2^{u-id+1}$。令$\mathcal{F}_{\text{Zeros}}^{d}$判断$\langle y_i\rangle^d$是否为0：$v_i = 1\{y_i = 0\}$。如此，计算$z_i' = u_i \cdot (1 \oplus v_i) \cdot \Pi_{j>i} v_j$。如此，$z_i' = u_i$当且仅当$y_i \neq 0$ 且$y_j = 0, j > i$。其余$z_j' = 0, j \neq i$。最终，$\text{MSNZB}(x) = \tilde{z} = \sum_i z_i'$。最终调用$\mathcal{F}_{\text{One-Hot}}^{\ell}(\langle\tilde{z}\rangle^{\iota})$得到$\{\langle z\rangle_i^B\}_{i\in[\ell]}$使得$z_i = 1, i = \tilde{z}$且$z_i = 0, i \neq \tilde{z}$。协议如下：

Algorithm 7 Most Significant Non-Zero Bit, $\Pi^{\ell,d}_{\mathsf{MSNZB}}$:

Input: For $b \in \{0,1\}$, P_b holds $\langle x \rangle^{\ell}_b$, $c = \ell/d, \iota = \log \ell$.
Output: For $b \in \{0,1\}$, P_b learns $\{\langle z_i \rangle^B_b\}_{i \in [\ell]}$ s.t. $z_i = 1$ if $2^i \leqslant x < 2^{i+1}$ and 0 otherwise.

P_0 & P_1 invoke $\mathcal{F}^{\ell,d}_{\mathsf{DigDec}}(\langle x \rangle^{\ell})$ and learn $\{\langle y_i \rangle^d\}_{i \in [c]}$.
for $i \in \{0, \ldots, c-1\}$ **do**
    P_0 & P_1 invoke $\mathcal{F}^{d,\ell,i}_{\mathsf{MSNZB\text{-}P}}(\langle y_i \rangle^d)$ and learn $\langle u_i \rangle^{\iota}$.
    P_0 & P_1 invoke $\mathcal{F}^{d}_{\mathsf{Zeros}}(\langle y_i \rangle^d)$ and learn $\langle v_i \rangle^B$.
    For $b \in \{0,1\}$, P_b sets $\langle v'_i \rangle^B_b = (b \oplus \langle v_i \rangle^B_b)$.
end for
P_0 & P_1 invoke $\mathcal{F}^{\iota}_{\mathsf{MUX}}(\langle v'_{c-1} \rangle^B, \langle u_{c-1} \rangle^{\iota})$ and learn $\langle z'_{c-1} \rangle^{\iota}$.
For $b \in \{0,1\}$, P_b sets $\langle w_{c-1} \rangle^B_b = b$.
for $i \in \{c-2, \ldots, 0\}$ **do**
    P_0 & P_1 invoke $\mathcal{F}_{\mathsf{AND}}(\langle w_{i+1} \rangle^B, \langle v_{i+1} \rangle^B)$ and learn $\langle w_i \rangle^B$.
    P_0 & P_1 invoke $\mathcal{F}_{\mathsf{AND}}(\langle w_i \rangle^B, \langle v'_i \rangle^B)$ and learn $\langle w'_i \rangle^B$.
    P_0 & P_1 invoke $\mathcal{F}^{\iota}_{\mathsf{MUX}}(\langle w'_i \rangle^B, \langle u_i \rangle^{\iota})$ and learn $\langle z'_i \rangle^{\iota}$.
end for
For $b \in \{0,1\}$, P_b sets $\langle \tilde{z} \rangle^{\iota}_b = \sum_{i=0}^{c-1} \langle z'_i \rangle^{\iota}_b$.
P_0 & P_1 invoke $\mathcal{F}^{\ell}_{\mathsf{One\text{-}Hot}}(\langle \tilde{z} \rangle^{\iota})$ and learn $\{\langle z_i \rangle^B\}_{i \in [\ell]}$.

其中 $\mathcal{F}^{d,\ell,i}_{\mathrm{MSNZB-P}}$和$\mathcal{F}^{d}_{\mathrm{Zeros}}$都是用LUTs（$d$-bits输入）实现的。又因为这两部分输入相同（均为$\langle y_i \rangle^d$），可以构造同一个LUT，其输入每一项为$(u_i || v_i)$。而$\mathcal{F}^{\ell}_{\mathrm{One-Hot}}(\langle \tilde{z} \rangle^{\iota})$也可以使用LUT实现，其中每一项为$\ell$-bit，输入为$\iota$-bits。

5. MSB-to-Wrap Optimization

本文的协议都依赖于$w = \mathrm{wrap}(\langle x \rangle^{\ell}_0, \langle x \rangle^{\ell}_1, L)$。而在某些应用场景下，可以确定$m_x = \mathrm{MSB}(x)$或者能得到$\langle m_x \rangle^B$。基于此，可得ParseError: KaTeX parse error: Undefined control sequence: \and at position 18: …((1\oplus m_x) \and (m_0 \oplus m_1…，其中$m_b = \mathrm{MSN}(\langle x \rangle^{\ell}_b))$。当$m_x$处于$\langle \cdot \rangle$时，可以使用一次$\binom{4}{1} - \mathrm{OT}_1$计算；当$m_x$时明文的时候，则可以通过一次$\binom{2}{1} - \mathrm{OT}_1$计算。

6. Math Library Functions

6.1 Exponential

$\mathcal{F}^{m,s,n,s'}_{\mathrm{rExp}}$ 计算$\mathrm{rExp}(z) = e^{-z}, z \in \mathbb{R}^+$。对于该函数，本文还是使用分段逼近的思想，不同点在于对于本文每一段穷举在预设精度下的所有$\mathrm{rExp}(z)$的值，然后使用LUT选择出所需要的一项。具体来说，首先调用$\mathcal{F}^{\ell,\{d_i\}_{i \in [c]}}_{\mathrm{DigDec}}$对$x$做分解，得到$\mathrm{rExp}(srt_{m,s}(x)) = \mathrm{rExp}(2^{d(k-1)-s} x_{k-1}) \cdot \ldots \cdot$

$\mathrm{rExp}(2^{-s}x_0)$，进一步对每一项穷举所有值做成一个LUT：$L_i(j) = \mathrm{Fix}(\mathrm{rExp}(2^{di-s}j, s'+2, s))$；然后选取$L_i[x_i]$，最后计算相乘计算结果。具体如下：

Functionality $\mathcal{F}_{\mathsf{rExp}}^{m,s,n,s'}(\langle x\rangle^m)$

1) Let $x = x_{k-1}||\ldots||x_0$, $x_i \in \{0,1\}^d$, $i \in [k]$, $dk = m$.
2) For $i \in [k]$, let $L_i : \{0,1\}^d \to \mathbb{Z}_{2^{s'+2}}$ s.t. $L_i(j) = \mathsf{Fix}\left(\mathsf{rExp}(2^{di-s}j), s'+2, s'\right)$.
3) Compute $g = L_{k-1}[x_{k-1}] * \ldots * L_0[x_0]$, g has bitwidth $s'+2$ and scale s'.
4) Return $\langle y\rangle^n$ for $y = \mathsf{SExt}(g, s'+2, n)$.

6.2 Sigmoid and Tanh

对Sigmoid的近似如下

$$\mathrm{Sigmoid}(z) = \begin{cases} 0.5, & z = 0 \\ \frac{1}{1+\mathrm{rExp}(z)}, & z > 0 \\ \mathrm{rExp}(-z)\frac{1}{1+\mathrm{rExp}(-z)}, & z < 0 \end{cases}$$

因此，除了需要计算rExp，还需要计算倒数。本文利用Goldschmidt's 算法实现了在non-bitwidth下的倒数计算。

Functionality $\mathcal{F}_{\mathsf{Rec}}^{\ell,s}(\langle v\rangle^\ell)$

Computes the initial approximation w as follows [63]:

1) $v = d||e||f$, $d \in \{0,1\}^{\ell-s}$, $e \in \{0,1\}^g$, $f \in \{0,1\}^{s-g}$.
2) $c_0||c_1 = L_{\mathsf{rec}}(e)$, $c_0 \in \{0,1\}^{g+4}$ and $c_1 \in \{0,1\}^{2g+3}$.
3) $c_2 = \mathsf{SExt}((c_0 *_{s+4} f), s+4, s+g+4)$.
4) $w' = 2^{s-g+1} *_{s+g+4} c_1 - c_2$, $w = \mathsf{TR}(w', g+3)$.

Goldschmidt's method for t iterations.

1) $p_1 = 2^s - \mathsf{TR}(v *_{2s+2} w, s)$.
2) $q_1 = 2^s + p_1, a_1 = q_1$.
3) For $i \in \{2,\ldots,t\}$ do
   a) $a_i = \mathsf{TR}(a_{i-1} *_{2s+2} q_{i-1}, s)$.
   b) $p_i = \mathsf{TR}(p_{i-1} *_{2s+2} p_{i-1}, s)$.
   c) $q_i = 2^s + p_i$.
4) Return $\mathsf{SExt}(a_t, s+2, \ell)$.

其中LUTL_{rec}的构造见参考文献 [1]。关于Goldschmidt's 算法的更多介绍，请参考文献 [2]。基于此，Sigmod计算如下：

Functionality $\mathcal{F}_h^{m,s,n,s'}(\langle x\rangle^m)$

1) $\langle u\rangle^{s'+2} \leftarrow \mathcal{F}_{\mathsf{rExp}}^{m,s,s'+2,s'}(\langle x\rangle^m)$.
2) $\langle w\rangle^{s'+2} \leftarrow \mathcal{F}_{\mathsf{Rec}}^{s'+2,s'}(\langle 2^{s'}+u\rangle^{s'+2})$.
3) Return $\mathsf{SExt}(w, s'+2, n)$.

而$\mathrm{Tanh}(z) = 2\cdot \mathrm{Sigmoid}(2z) - 1$。

6.3 Reciprocal of Sqaure Root

$\mathcal{F}_{\mathrm{rsqrt}}(\ell, s, \ell, s')$计算 $\mathrm{rsqrt}(x) = \frac{1}{\sqrt{x}}$，其中$x \geq \epsilon$。首先对$\sqrt{x}$做近似计算，然后再执行Goldschmidt's算法迭代近似。具体如下：

Functionality $\mathcal{F}_{\mathsf{rsqrt}}^{\ell,s,\ell,s'}(\langle x\rangle^\ell)$

Normalizes x to x' as follows:

1) $k = \mathsf{MSNZB}(x) \in [\ell]$.
2) $A = 2^{\ell-2-k}$, $B = (s-k) \bmod 2$.
3) $C = 2^{\lceil \frac{s-k}{2} \rceil + \lfloor \frac{\ell-s-1}{2} \rfloor}$.
4) $x' = x *_\ell A$.

Computes the initial approximation w as follows:

1) $x' = d||e||f, d \in \{0,1\}^2, e \in \{0,1\}^g, f \in \{0,1\}^{\ell-2-g}$.
2) $w = L_{\mathsf{rsqrt}}(e||B)$, $w \in \{0,1\}^{g+4}$.

Goldschmidt's method for t iterations:

1) $x'' = \mathsf{TR}(x', \ell-3-s')$, $q_0 = B \ ? \ x'' : \mathsf{TR}(x', 1)$.
2) $a_0 = 2^{s'-g-2} *_{s'+2} w$, $p_0 = a_0$.
3) For $i \in \{1, \ldots, t\}$ do
   a) $Y_i = \mathsf{TR}(p_{i-1} *_{2s'+2} p_{i-1}, s')$.
   b) $q_i = \mathsf{TR}(q_{i-1} *_{2s'+2} Y_i, s')$.
   c) $p_i = 3\cdot 2^{s'-1} - (q_i \gg_A 1)$.
   d) $a_i = \mathsf{TR}(a_{i-1} *_{2s'+2} p_i, s')$.

Uses reciprocal square root of x' to compute the same for x:

1) Return $\mathsf{TR}(a_t *_{\ell/2+s'+3} C, \lfloor \frac{\ell-s-1}{2} \rfloor) \bmod L$.

其中，$L_{\mathrm{rsqrt}}(e||B) = \mathrm{Fix}(\frac{1}{\sqrt{(B+1)(1+urt_{g,g}(e))}}, g+4, g+2)$。

关于倒数和平方根的倒数的计算，在安全计算中本身就是很难的问题。更多的相关资料可以查看参考文献 [2]。

7. Evaluation

首先是关于Sigmoid函数计算的开销比较，可以看到SIRNN比MP-SPDZ和MiniONN都开销少很多。

Technique	Total Time for #Instances (in sec)				Comm./ Instance (in KB)	Max ULP Err.
	10^2	10^3	10^4	10^5		
Our Work	0.08	0.10	0.25	1.58	4.88	3
MiniONN 48-piece	0.20 (2.5x)	1.94 (19.4x)	18.85 (75x)	182.2 (115x)	341.03 (70x)	4
MiniONN 12-piece	0.06 (0.8x)	0.54 (5.4x)	5.24 (21x)	53.84 (34x)	93.36 (19.1x)	104
Deep-Secure	0.16 (2x)	0.84 (8.4x)	8.1 (32x)	141.3 (89x)	124.65 (25x)	NA
MP-SPDZ Ring Poly	0.75 (9.4x)	1.72 (17.2x)	14.88 (59.5x)	140.6 (89x)	981.11 (201x)	2
MP-SPDZ Ring PL	0.27 (3.4x)	0.28 (2.8x)	1.32 (5.3x)	12.34 (7.8x)	76.42 (15.7x)	266
MP-SPDZ Field Poly	0.91 (11.4x)	1.91 (19.1x)	16.51 (66x)	127 (80x)	228.63 (46.9x)	2
MP-SPDZ Field PL	0.52 (6.5x)	0.47 (4.7x)	1.79 (7.2x)	14.23 (9x)	27.52 (5.6x)	266

TABLE I: Comparison with prior works on sigmoid with varying number of instances.

其次是Division-2的开销比较：

Technique	Total Time for #Instances (in sec)				Comm./ Instance (in KB)	Max ULP Error
	10^2	10^3	10^4	10^5		
Exponentiation						
Our Work	0.03	0.04	0.15	1.00	2.12	3
MP-SPDZ	0.34 (11.3x)	0.56 (14x)	3.90 (26x)	35.95 (35.9x)	254.95 (120x)	2
Reciprocal Square Root						
Our Work	0.13	0.13	0.30	1.84	6	4
MP-SPDZ	0.94 (7.2x)	3.90 (30x)	35.87 (120x)	338.9 (184x)	2535 (423x)	8

TABLE II: Comparison with (power-of-2) ring-based MP-SPDZ protocols with varying number of instances.

进一步是和MiniONN和DeepSecure关于安全预测的比较：

Inference Benchmark	Runtime (in sec)		Comm.	
	Prior	Our Work	Prior	Our Work
MiniONN LSTM	1.1 (2.2x)	0.48	182 MB (19.5x)	9.32 MB
DeepSecure B4	465 (87x)	5.3	83.7 GB (43x)	1.94 GB

TABLE III: Comparison with benchmarks from MiniONN [83] and DeepSecure [102].

最后是和ABY在大型神经网络上的安全预测比较：

Benchmark	Batch	Runtime (sec)		Comm.	
		[41]	SIRNN	[41]	SIRNN
Industrial-72	1	68.33 (18x)	3.7	11.84 GB (510x)	23.8 MB
	128	8746* (661x)	13.2	1.47 TB* (1451x)	1.04 GB
Google-30	1	3337 (67x)	49.6	259 GB (574x)	0.45 GB
	128	$4.3x10^{5}$* (3050x)	140	32.38 TB* (1316x)	25.2 GB
Heads	1	NA	409.7	NA	85.5 GB

*extrapolated, the run could not be completed due to TB comm.

TABLE IV: Secure inference on DNNs using SIRNN and [41].

8. 结论

本文提出了在non-uniform bitwidth下的而一种安全两方计算库，这种计算能进一步节省开销。但是许多uniform bitwidth下的常用技术也变得不能直接使用，比如Beaver Triples。但目前来看整体性能还是得到了很大提升。

参考文献

[1] M. Ito, N. Takagi, and S. Yajima, “Efficient Initial Approximation for Multiplicative Division and Square Root by a Multiplication with Operand Modification,” IEEE Transactions on Computers, 1997.
[2] Keller M, Sun K. Secure Quantized Training for Deep Learning[J]. arXiv preprint arXiv:2107.00501, 2021.