Scalable Multi-Party Computation Protocols for Machine Learning in the Honest-Majority Setting

Fengrun Liu
University of Science and Technology of China & Shanghai Qi Zhi Institute

Xiang Xie
Shanghai Qi Zhi Institute & PADO Labs

Yu Yu
Shanghai Jiao Tong University &
State Key Laboratory of Cryptology, P. O. Box 5159, Beijing, 100878, China

Fengrun Liu 2024/06/07

Outline

- Background
- Our Results
- Technique 1: 1-round Truncation with 1-bit Gap
  - Efficient and well-defined operations in fields of Mersenne primes
  - A large gap is involved in truncation
  - Able to fix the positive overflow if we only allow positive overflow
  - Seamlessly combined with DN protocol to obtain 1-round fixed-point mult
- Technique 2: Improved Bitwise Primitives
  - Efficient Prefix-OR and Bitwise Comparison

Background

MPC Protocols Tailored for Privacy-preserving Machine Learning (PPML)

Secure Multi-Party Computation (MPC):

enables a group of n parties to collaboratively compute a **function** on their private inputs while preserving the privacy of those inputs

Privacy-preserving Machine Learning (PPML):

allows multiple parties to collaborate on **training or inference** tasks on distributed datasets without exposing the **individual data points** and the **ML model itself**

- Small-scale scenarios for 2 – 4 parties only:
  rely on additive secrete sharing in the ring setting
  used in client-server model

- General-purpose MPC for n parties:
  lack of efficient protocols to realize non-linear functions, such as truncation and comparison.

=> **Motivation:** to develop the **scalable and efficient** MPC protocols tailored for PPML

Starting Point: Damgård-Nielsen [DN07] Protocol based on Shamir's Secret Sharing

Our Results

Scalable and Efficient MPC-baed PPML Framework in the Honest Majority Setting

In application: our protocols facilitate the efficient and scalable online oblivious inference.

We conduct experiments in various settings, ranging from 3PC to 63PC (simulated by 11 servers).

Setting	Online (s)	Offline (s)
LAN	0.1	1.2
WAN	4.6	10.7

Network	3PC	7PC	11PC	21PC	31PC	63PC
3-layer DNN	0.37	0.37	0.38	0.39	0.40	0.47
3-layer CNN	0.39	0.40	0.41	0.44	0.48	0.68
4-layer CNN	1.2	1.3	1.3	1.7	2.0	4.6

Efficient: runtime (s) for oblivious inference of 4-layer CNN with 63 parties

Scalable: online runtime (s) for oblivious inference in the WAN setting from 3-63 PC

In theory: we optimize the following primitives leveraging the unique properties of Mersenne prime fields.

1. truncation (related to fixed-point multiplication) — This: 1-bit gap & 1 round
2. bitwise comparison (related to various non-linear functions) — This: no gap & 1 round

Now we only consider the **integer numbers** $\bar{x}$

Mersenne Primes $p = 2^{\ell} - 1$ for prime ℓ

Efficient and well-defined operations in fields of Mersenne primes

	In the ring setting $\mathbb{Z}_{2^{\ell}}$		In the field setting $\mathbb{F}_p$ ($p = 2^{\ell} - 1$)
Representation of integer $\bar{x}$	$x = \bar{x} \pmod{2^{\ell}}$ $x = \begin{cases} \bar{x}, & \bar{x} \geq 0 \\ 2^{\ell} - \bar{x}, & \bar{x} < 0 \end{cases}$ for $\bar{x} \in [-2^{\ell-1}, 2^{\ell-1})$		$x = \bar{x} \pmod{2^{\ell} - 1}$ $x = \begin{cases} \bar{x}, & \bar{x} \geq 0 \\ 2^{\ell} - 1 - \bar{x}, & \bar{x} < 0 \end{cases}$ for $\bar{x} \in (-2^{\ell-1}, 2^{\ell-1})$
1. MSB of x indicates the sign of $\bar{x}$	Representation of $\bar{x}$ in $\mathbb{Z}_{2^{\ell}}$ [▒ | $\ell - d$ | d] (2's complement)	$\approx$	Representation of $\bar{x}$ in $\mathbb{F}_{2^{\ell}-1}$ [▒ | $\ell - d$ | d]
Efficient mod operation in practice	$x + y = \bar{x} + \bar{y} \pmod{2^{\ell}}$ $x \cdot y = \bar{x} \cdot \bar{y} \pmod{2^{\ell}}$ $a \cdot 2^{\ell} + b = b \pmod{2^{\ell}}$ shift bits		$x + y = \bar{x} + \bar{y} \pmod{2^{\ell} - 1}$ $x \cdot y = \bar{x} \cdot \bar{y} \pmod{2^{\ell} - 1}$ $a \cdot 2^{\ell} + b = a + b \pmod{2^{\ell} - 1}$ shift bits + addition
well-defined truncation: $\bar{x}/2^d$	Representation of $\bar{x}/2^d$ in $\mathbb{Z}_{2^{\ell}}$ [d | ▒ | $\ell - d$] (2's complement)	$=$	Representation of $\bar{x}/2^d$ $\mathbb{F}_{2^{\ell}-1}$ [d | ▒ | $\ell - d$] elaborated in Thm 3.1

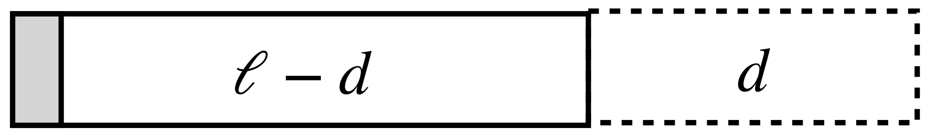
2. Truncation on x: shift the bits down by d positions and fill the top d bits with the MSB of x

We can view the field element in $\mathbb{F}_{2^\ell - 1}$ as the almost 2's complementation of some integer.

Fixed-point numbers represented in $\mathbb{F}_{2^\ell - 1}$

Truncation is required when performing fixed-point multiplication

For a fixed-point number x



fractional part

$$x = \sum_{i=0}^{\ell-1} 2^{i-d} \cdot x_i$$

scale x by multiplying 2^d to obtain an integer $x' \in \mathbb{F}_{2^\ell - 1}$

$$x' = x \cdot 2^d = \sum_{i=0}^{\ell-1} 2^i \cdot x_i$$

For two fixed-point numbers x and y, we can perform fixed-point operations with x′ and y′ $\in \mathbb{F}_{2^\ell - 1}$

addition: $x' + y' = (x + y) \cdot 2^d$

multiplication: $x' \cdot y' = (x \cdot y) \cdot 2^{2d}$

multiplication with truncation: $\dfrac{(x' \cdot y')}{2^d} = (x \cdot y) \cdot 2^d$

Tech1: Truncation in $\mathbb{F}_{2^\ell-1}$

A large gap is involved when performing truncation on secret values

truncation on secret x

In the ring setting $\mathbb{Z}_{2^\ell}$

1. MSB of x indicates the sign of $\bar{x}$

Representation of $\bar{x}$ in $\mathbb{Z}_{2^\ell}$

	$\ell - d$	d

(2's complement)

2. Trunc(x): Truncation on x, performing $\bar{x}/2^d$

Representation of $\bar{x}/2^d$ in $\mathbb{Z}_{2^\ell}$

d		$\ell - d$

(2's complement)

Given (r, Trunc(r)), we can perform truncation on x in $\mathbb{Z}_{2^k}$ as follows:

1. $a = x + r$ (mod 2^ℓ) where $x \in \mathbb{Z}_{2^\ell}$ and $r \leftarrow \mathbb{Z}_{2^\ell}$
2. Reveal a
3. Trunc(x) = Trunc(a) - Trunc(r) (mod 2^ℓ)
   only holds iff $\bar{a} = \bar{x} + \bar{r}$ (in the view of integers) where $\bar{x} \in [-2^{\ell-1}, 2^{\ell-1})$ and $\bar{r} \leftarrow [-2^{\ell-1}, 2^{\ell-1})$

When $x + r$ is performed in Step 1, the **OVERFLOW** might happen, meaning $\bar{x} + \bar{r} \notin [-2^{\ell-1}, 2^{\ell-1})$.

Tech1: Truncation in $\mathbb{F}_{2^\ell-1}$

A large gap is involved when performing truncation on secret values

truncation on secret x

In the ring setting $\mathbb{Z}_{2^\ell}$

Given (r, Trunc(r)), we could perform truncation on x in $\mathbb{Z}_{2^\ell}$ as follows:

1. $a = x + r$ (mod 2^ℓ) where $x \in \mathbb{Z}_{2^\ell}$ and $r \leftarrow \mathbb{Z}_{2^\ell}$
2. Reveal a

?? 3. Trunc(x) = Trunc(a) - Trunc(r) (mod 2^ℓ)

only holds iff $\bar{a} = \bar{x} + \bar{r}$ (in the view of integers) where $\bar{x} \in [-2^{\ell-1}, 2^{\ell-1})$ and $\bar{r} \leftarrow [-2^{\ell-1}, 2^{\ell-1})$

When $x + r$ is performed in Step 1, the **OVERFLOW** might happen, meaning $\bar{x} + \bar{r} \notin [-2^{\ell-1}, 2^{\ell-1})$.

For example, we have $a = x + r$ in $\mathbb{Z}_{2^8}$.

x=162	0 1 0 1 0 0 0 1 0	$\bar{x}$ is positive		Trunc(x)=20	0 0 0 0 1 0 1 0 0
		+	truncation	+	
r=161	0 1 0 1 0 0 0 0 1	$\bar{r}$ is positive	d=3	Trunc(r)=20	0 0 0 0 1 0 1 0 0
+		≠		≠	
a=323	1 0 1 0 0 0 0 1 1	$\bar{a}$ is negative		Trunc(a)=488	1 1 1 1 0 1 0 0 0

positive overflow happens, i.e. $\bar{x} + \bar{r} = 323 \notin [-2^7, 2^7)$

∴ Trunc(a) ≠ Trunc(x) + Trunc(r)

Tech1: Truncation in $\mathbb{F}_{2^\ell - 1}$

A large gap is involved when performing truncation on secret values

truncation on secret x

In the ring setting $\mathbb{Z}_{2^\ell}$

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

1. MSB of x indicates the sign of $\bar{x}$

2. Trunc(x): Truncation on x, performing $\bar{x}/2^d$

Given (r, Trunc(r)) :

In the ring setting $\mathbb{Z}_{2^\ell}$

Representation of $\bar{x}$ in $\mathbb{Z}_{2^\ell}$

(2's complement)

Representation of $\bar{x}/2^d$ in $\mathbb{Z}_{2^\ell}$

(2's complement)

Truncation on x in $\mathbb{Z}_{2^\ell}$ as follows:

1. $a = x + r$ $\pmod{2^\ell}$
2. Reveal a
3. Trunc(x) = Trunc(a) - Trunc(r) $\pmod{2^\ell}$

only holds iff $\bar{a} = \bar{x} + \bar{r} \in [-2^{\ell-1}, 2^{\ell-1})$

In the field setting $\mathbb{F}_p$

Representation of $\bar{x}$ in $\mathbb{F}_{2^\ell - 1}$

Representation of $\bar{x}/2^d$ $\mathbb{F}_{2^\ell - 1}$

Truncation on x in $\mathbb{F}_{2^\ell - 1}$ as follows:

1. $a = x + r$ $\pmod{2^\ell - 1}$
2. Reveal a
3. Trunc(x) = Trunc(a) - Trunc(r) $\pmod{2^\ell - 1}$

only holds iff $\bar{a} = \bar{x} + \bar{r} \in (-2^{\ell-1}, 2^{\ell-1})$

Trunc(a) = Trunc(x) + Trunc(r) only holds w.h.p. for small x.

The above methods introduce A LARGE GAP between secrets and modulus.

Tech1: Truncation in $\mathbb{F}_{2^\ell - 1}$

We are able to fix the positive overflow if we only allow positive overflow

truncation on secret x

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

Take the same example, we have $a = x + r$ in $\mathbb{Z}_{2^8-1}$.

x=162	0 1 0 1 0 0 0 1 0	truncation	Trunc(x)=20	0 0 0 0 1 0 1 0 0
r=161	0 1 0 1 0 0 0 0 1	d=3	+ Trunc(r)=20	0 0 0 0 1 0 1 0 0
+ a=323	1 0 1 0 0 0 0 1 1		$\neq$ Trunc(a)=488	1 1 1 1 0 1 0 0 0

Tech1: Truncation in $\mathbb{F}_{2^\ell - 1}$

We are able to fix the positive overflow if we only allow positive overflow

truncation on secret x

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

Take the same example, we have $a = x + r$ in $\mathbb{Z}_{2^8-1}$.

x=162 | 0 1 0 1 0 0 0 1 0

r=161 | 0 1 0 1 0 0 0 0 1

+

a=323 | 0 | 1 0 1 0 0 0 0 1 1

truncation

d=3

Trunc(x)=20 | 0 0 0 0 1 0 1 0 0

+

Trunc(r)=20 | 0 0 0 0 1 0 1 0 0

=

Trunc(a)=40 | 0 | 0 0 0 1 0 1 0 0 0

Expected Truncation

- Δ

just remove the misfilled top d bits

Actual Result | 1 1 1 1 0 1 0 0 0

Q1: How to ensure only the positive overflow is allowed?

Q2: How to detect and correct the error introduced by positive overflow?

Tech1: Truncation in $\mathbb{F}_{2^\ell-1}$

We are able to fix the positive overflow if we only allow positive overflow

truncation on secret x

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

Take the same example, we have $a = x + r$ in $\mathbb{Z}_{2^8-1}$.

x=162	0 1 0 1 0 0 0 1 0	x is always positive = introduce 1-big gap between secrets and modulus
r=161	0 1 0 1 0 0 0 0 1	r is uniformly sampled from $\mathbb{F}_{2^\ell-1}$
+		
a=323	0 1 0 1 0 0 0 0 1 1	only positive overflow might happen

Q1: How to ensure only the positive overflow is allowed?

We can impose a constraint on the input x to ensure x is always positive.

Q2: How to detect and correct the error caused by positive overflow?

Tech1: Truncation in $\mathbb{F}_{2^\ell-1}$

We are able to fix the positive overflow if we only allow positive overflow

truncation on secret x

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

Take the same example, we have $a = x + r$ in $\mathbb{Z}_{2^8-1}$.

x=162 0 1 0 1 0 0 0 1 0

given MSB(r) r=161 0 1 0 1 0 0 0 0 1

\+

a is revealed, so is MSB(a) a=323 0 1 0 1 0 0 0 0 1 1

truncation d=3

Trunc(x)=20 0 0 0 0 1 0 1 0 0

\+

Trunc(r)=20 0 0 0 0 1 0 1 0 0

=

Trunc(a)=40 0 0 0 0 1 0 1 0 0 0

Expected Truncation

$- \Delta$ just remove the misfilled top d bits

Actual Result 1 1 1 1 0 1 0 0 0

Q1: How to ensure the only allowance of positive overflow?

We can impose a constraint on the input x to ensure x is always positive.

Q2: How to detect and correct the error caused by positive overflow?

elaborated in Thm 3.2

Positive overflow occurs iff MSB(r) = 0 and MSB(a) = 1, and we can then easily correct the error as above.

Tech1: Truncation in $\mathbb{F}_{2^\ell - 1}$

We are able to fix the positive overflow if we only allow positive overflow

truncation on secret x

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

Online Complexity: 1 round & 1-bit gap

Take the same example, we have $a = x + r$ in $\mathbb{Z}_{2^8 - 1}$.

	x=162	0 1 0 1 0 0 0 1 0
given MSB(r)	r=161	0 1 0 1 0 0 0 0 1
	+	
a is revealed, so is MSB(a)	a=323	0 1 0 1 0 0 0 0 1 1

truncation $\xrightarrow{}$ d=3

Trunc(x)=20	0 0 0 0 1 0 1 0 0	
+		
Trunc(r)=20	0 0 0 0 1 0 1 0 0	Expected Truncation
=		
Trunc(a)=40	0 0 0 0 1 0 1 0 0 0	

- Δ just remove the misfilled top d bits

Actual Result 1 1 1 1 0 1 0 0 0

Notation [x]: x is secret shared

Given ([r], [Trunc(r)], [MSB(r)]), we can perform truncation on [x] in $\mathbb{F}_{2^\ell - 1}$ as follows:

1. [a] = [x] + [r]
2. Reveal a and MSB(a)
3. [e] = (1 - [MSB(r)]) · MSB(a) — e = 1 indicating positive overflow occurs
4. [Trunc(x)] = Trunc(a) - [Trunc(r)] + [e] · $(2^{\ell-d} - 1)$ — holds for any x $\in [0, 2^{\ell-1})$ representing positive numbers

Tech1: Truncation in $\mathbb{F}_{2^\ell - 1}$

Seamlessly combined with DN protocol to obtain 1-round fixed-point mult

fixed-point mult
on secret x and y

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

Online Complexity:
1 round & 1-bit gap

Notations [x]: degree-t sharing
<x>: degree-2t sharing
[x] · [y] = <xy>

Suppose x and y represent two fixed-point numbers:

DN Protocol: [xy]

1. <a> = [x] · [y] + <r>
2. Reveal a
3. [xy] = a – [r]

Our Truncation Protocol: [Trunc(x)]

1. [a] = [x] + [r]
2. Reveal a and MSB(a)
3. [e] = (1 – [MSB(r)]) · MSB(a)
4. [Trunc(x)] = Trunc(a) – [Trunc(r)] + [e] · $(2^{\ell-d} - 1)$

combined

offline: random truncation triple: (<r>, [Trunc(r)], [MSB(r)])

Our Fixed-point Multiplication Protocol: [Trunc(xy)]

1. [a] = [x] · [y] + <r>
2. Reveal a and MSB(a)
3. [e] = (1 – [MSB(r)]) · MSB(a)
4. [Trunc(xy)] = Trunc(a) – [Trunc(r)] + [e] · $(2^{\ell-d} - 1)$

Tech2: Improved Bitwise Primitives

Efficient Prefix-OR and Bitwise Comparison

Q: Why do we want to optimize bitwise comparison?

It underpins arithmetic comparison crucial for various non-linear operations.

various non-linear functions → arithmetic comparison (a < 0) → bitwise comparison (y_B < r_B)

various non-linear functions

$$\mathrm{DReLU}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$\mathrm{ReLU}(x) = \mathrm{DReLU}(x) \cdot x$$

$$\mathrm{Max}(a, b) = \mathrm{ReLU}(a - b) + b$$

arithmetic comparison (a < 0)

MSB(a) = LSB (2a) holds in odd rings

1. y = 2a + r
2. Reveal y
3. LSB(2a) = LSB(y) ⊕ LSB(r) ⊕ (y_B < r_B)

public secret

bitwise comparison (y_B < r_B)

look for the first different bit

(public) y_B 0 0 1 0 1 0 0 0 1 0 1 0 1

(secret) r_B 0 0 1 0 1 0 1 0 0 0 1 0 0

* XOR

(secret) 0 0 0 0 0 0 1 0 1 0 0 0 1

* Prefix-OR

(secret) 0 0 0 0 0 0 1 1 1 1 1 1 1

(secret) e_B 0 0 0 0 0 0 1 0 0 0 0 0 0

(y_B < r_B) = <e_B, r_B>

* XOR is free between secret bits and public bits: [a] ⊕ b = a + b - 2[a] · b

* OR involves a multiplication between secrets: [a] ∨ [b] = a + b - [a] · [b]

* Prefix-OR involves ℓ multiplications: compute $b_j = \bigvee_{i=1}^{j} a_i$ for $j = 1, \ldots, \ell$

How to efficiently compute Prefix-OR?

Tech2: Improved Bitwise Primitives

Efficient Prefix-OR and Bitwise Comparison

Prefix-OR on secrete bits

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

Online Complexity: 1 round

(secret)	0 0 0 0 0 0 1 0 1 0 0 0 1	(secret)	1 1 1 1 1 1 0 1 0 1 1 1 0
Prefix-OR	→	Prefix-AND	
(secret)	0 0 0 0 0 0 1 1 1 1 1 1 1	(secret)	1 1 1 1 1 1 0 0 0 0 0 0 0

1. locate the first 1-bit's position starting from MSB
2. set all the following bits to 1

involved with OR operation [a] ∨ [b] = a + b - [a] · [b]

* Prefix-OR: compute $b_j = \bigvee_{i=1}^{j} a_i$ for $j = 1, \ldots, \ell$

Online Complexity of [NO07]: 5 rounds

→

1. locate the first 0-bit's position starting from MSB
2. set all the following bits to 0

(zero out all the following bits)

only involved with multiplication

* Prefix-MULT: compute $b_j = \Pi_{i=1}^{j} a_i$ for $j = 1, \ldots, \ell$

Online Complexity of [BB89]: 1 round

Tech2: Improved Bitwise Primitives

Efficient Prefix-OR and Bitwise Comparison

Bitwise Comparison between public bits and secret bits

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

arithmetic comparison (a < 0) → bitwise comparison (y_B < r_B)

MSB(a) = LSB (2a)   holds in odd rings

1. y = 2a + r
2. Reveal y
3. LSB(2a) = LSB(y) ⊕ LSB(r) ⊕ (y_B < r_B)
   public   secret

Q: Can we do better than 2 rounds?

- comparison between public bits and secret bits
  (y_B < r_B) = <e_B, r_B>
  (secret)
  1 round (mult)
- comparison between secret bits and public bits
  (r_B < y_B) = <e_B, y_B>
  (public)
  free
- (y_B < r_B) = (p − r_B < p − y_B)
  (public) (secret)   (secret)   (public)
  p − r_B is free when p is Mersenne prime!

look for the first different bit

(public) y_B   0 0 1 0 1 0 0 0 1 0 1 0 1

(secret) r_B   0 0 1 0 1 0 1 0 0 0 1 0 0

free   * XOR

(secret)   0 0 0 0 0 0 1 0 1 0 0 0 1

1 round   * Prefix-OR

(secret)   0 0 0 0 0 0 1 1 1 1 1 1 1

free

(secret) e_B   0 0 0 0 0 0 1 0 0 0 0 0 0

1 round (mult)

(y_B < r_B) = <e_B, r_B>
(secret) (secret) (secret)

Tech2: Improved Bitwise Primitives

Efficient Prefix-OR and Bitwise Comparison

Bitwise Comparison between public bits and secret bits

In the field setting $\mathbb{F}_p$ ($p = 2^{\ell} - 1$)

Online Complexity: 1 round

---

arithmetic comparison (a < 0)

MSB(a) = LSB (2a) holds in odd rings

1. y = 2a + r
2. Reveal y
3. LSB(2a) = LSB(y) ⊕ LSB(r) ⊕ (y_B < r_B)
   public secret

Q: Can we do better than 2 rounds?

- comparison between public bits and secret bits
  (y_B < r_B) = <e_B, r_B> (secret) — 1 round (mult)
- comparison between secret bits and public bits
  (r_B < y_B) = <e_B, y_B> (public) — free
- (y_B < r_B) = (p - r_B < p - y_B)
  (public) (secret) (secret) (public)
  p - r_B is free when p is Mersenne prime!

(y_B < r_B) = (p - r_B < p - y_B)

bitwise comparison (p - r_B < p - y_B)

look for the first different bit

(public) p - r_B	1 1 0 1 0 1 1 1 0 1 0 1 0
(secret) p - y_B	1 1 0 1 0 1 0 1 1 1 0 1 1

free — * XOR

(secret)	0 0 0 0 0 0 1 0 1 0 0 0 1

1 round — * Prefix-OR

(secret)	0 0 0 0 0 0 1 1 1 1 1 1 1

free

(secret) e_B	0 0 0 0 0 0 1 0 0 0 0 0 0

free — (p - r_B < p - y_B) = <e_B, p - y_B>
(public) (secret) (public)

Tech2: Improved Bitwise Primitives

Other Building Blocks

In the field setting $\mathbb{F}_p$ ($p = 2^\ell - 1$)

$$\mathrm{DReLU}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$\mathrm{ReLU}(x) = \mathrm{DReLU}(x) \cdot x$$

This multiplication is saved by using the techniques of two-layer DN multiplication [ATLAS]

Protocols	Rounds		Communication	
	Online	Prep.	Online	Prep.
$\Pi_{\text{Fixed-Mult}}$	1	2	2	3ℓ
Π_{PreMult}	1	2	2ℓ	7ℓ
Π_{PreOR}	1	2	2ℓ	7ℓ
$\Pi_{\text{Bitwise-LT}}$	1	2	2ℓ	7ℓ
Π_{DReLU}	3	2	$4+2\ell$	$1+10\ell$
$\Pi_{\text{2L-DN}}$	1	1	$2(m+1)$	$m+1$
Π_{ReLU}	3	2	$6+2\ell$	$2+10\ell$
Π_{Maxpool}	$3\log m$	2	$(m-1)(6+2\ell)$	$(m-1)(2+10\ell)$