# On Probabilistic Truncation in Privacy-preserving Machine Learning

**Lijing Zhou[1], Bingsheng Zhang[2,3,*], Ziyu Wang[1], Tianpei Lu[2], Qingrui Song[1], Su Zhang[1], Hongrui Cui[4], Yu Yu[4]**

[1]Huawei Technology, China,
[2]The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China,
[3]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security,
[4]Shanghai Jiao Tong University, Shanghai,
{zhoulijing, songqingrui1, zhangsu14, wangziyu13, wangxianggui1}@huawei.com, {bingsheng, lutianpei}@zju.edu.cn, rickfreeman@sjtu.edu.cn, yuyu@cs.sjtu.edu.cn,

## Abstract

Probabilistic truncation has been widely used in a broad range of privacy-preserving machine learning (PPML) platforms, such as EdaBits (Crypto 20), ABY 2.0 (Usenix 21), Crypten (NIPS 21), Piranha-Falcon (Usenix 22), and Bicoptor (S&P 23), etc. In this work, we examine the problems of common probabilistic truncation protocols in PPML, and propose solutions from the perspectives of accuracy and efficiency.

With regard to accuracy, we found the recommended precision parameters in many existing works are incorrect, leading to extremely low inference accuracy. We conducted a thorough analysis of their open-source code and found that their errors were mainly caused by simplified implementation; more specifically, random numbers are not correctly sampled in probabilistic truncation protocols. Based on this, we provide a detailed theoretical analysis to validate our views.

With regard to efficiency, we identify limitations in the state-of-the-art secure comparison, Bicoptor's (S&P 2023) DReLU protocol, which relies on the probabilistic truncation and is heavily constrained by the security parameter to eliminate errors, significantly impacting its performance. To address these challenges, we introduce a non-interactive deterministic truncation technique, replacing the original probabilistic truncation. Additionally, we propose a new technique for speeding up the ReLU/DReLU evaluation, which can be applied to the other non-linear functions as well. When the input size of DReLU is reduced to 7 bits, we can speed up approximately 5x the ReLU protocols w.r.t. ABY$^3$, ABY 2.0, EdaBits, and Bicoptor without compromising model accuracy. The improved protocol can complete a ReLU evaluation within 2 rounds and 704 bits overall communication when the input/output is secretly shared over the 64-bit ring, which yields a 92% communication reduction on original Bicoptor. Compared to existing PPML platforms with GPU acceleration, our benchmark indicates a 10x improvement in the DReLU protocol, and a 6x improvement in the ReLU protocol over Piranha-Falcon and a 3.7x improvement over Bicoptor. As a result, the overall PPML model inference could be sped up by 3-4 times.

*Bingsheng Zhang is the corresponding author.


## Introduction

Privacy-preserving machine learning (PPML) is an emerging research area that enables model training and inference without revealing the underlying data as well as the model parameters to the PPML participants. To date, the most popular state-of-the-art (SOTA) PPML platforms, such as Orca (Jawalkar et al. 2023), Falcon (Wagh et al. 2021), ABY$^3$ (Mohassel and Rindal 2018), and SecureML (Mohassel and Zhang 2017), are based on the multi-party computation (MPC).

Existing research has explored various methods to reduce the communication overhead. However, these improvements can come with certain drawbacks. Taking truncation protocols as an example, a typical PPML uses fixed-point arithmetics, and the truncation protocol is often needed to re-scale the range of the underlying data. In practice, the non-interactive/less-interactive truncation protocols, e.g., SecureML (Mohassel and Zhang 2017) and ABY$^3$ (Mohassel and Rindal 2018) demonstrate better performance compared to CryptFlow2 (Rathee et al. 2020) and Cheetah (Huang et al. 2022). Nevertheless, these non-interactive/less-interactive truncation protocols may encounter truncation failure due to probabilistic errors. We refer to these types of truncation protocols as "probabilistic truncation protocols". The main focus of this paper is to comprehensively analyze and discuss, from the perspectives of accuracy and efficiency, the issues arising from using probabilistic truncation protocols, and we propose corresponding solutions to address these issues.

Furthermore, we propose a new technique to reduce the communication cost of ReLU while preserving the overall model accuracy. Based on the observation that ReLU/-DReLU evaluation is at least $10\times$ slower than secure multiplication, our main idea is to reduce the input size of the DReLU component at the cost of an extra multiplication. Our benchmark indicates that, for most models, using 7 effective bits can reduce over $70\%$ communication cost and speed up approximately 5x the ReLU evaluation protocols of ABY$^3$, ABY 2.0, EdaBits, and Bicoptor with no noticeable accuracy loss. Tab. 1 summarizes the communication cost of previous works on the ReLU protocol, with the improvement of adopting our new technique.

**Low communication ReLU protocol.** To the best of our knowledge, Bicoptor (Zhou et al. 2023) provides the most efficient ReLU/DReLU protocol construction. Besides the benefit of our new technique, we propose an improved Bicoptor ReLU protocol, further reducing its communication by $50\%$.

Bicoptor adopts the probabilistic truncation in (Mohassel and Zhang 2017) to realize the arithmetic right shift operation. This probabilistic truncation would cause a large error with probability $1/2^{\ell-\ell_x-1}$. Therefore, the underlying MPC must operate over a relatively large ring, say $\ell = 64$ to mitigate this drawback, significantly affecting the overall protocol communication. To address this issue, we apply our non-interactive deterministic cut protocol to replace the probabilistic truncation protocol, avoiding such a probabilistic error; therefore, the subsequent protocol can utilize a much smaller ring size. This technique can reduce the communication cost of a DReLU protocol in Bicoptor (Zhou et al. 2023) by approximately 50%.

Combining with our technique, the overall communication of the improved ReLU protocol is $(\ell_x+1)(\ell_x+1)+5\ell$. When $\ell_x = 7$ and $\ell = 64$, our protocol incurs a communication cost of only 30% of Bicoptor, significantly lower than the typical logarithmic-round protocols, e.g, its overall communication cost is only 79% of Falcon (Wagh et al. 2021) online phase communication. Our protocol can perform a ReLU evaluation within 2-round and 704-bit overall communication, when the input/output is secretly shared over $\mathbb{Z}_{2^{64}}$, which yields a 92% communication reduction on original Bicoptor. Compared to existing PPML platforms with GPU acceleration, our benchmark indicates a 10x improvement in DReLU protocol, and a 6x improvement in the ReLU protocol over Piranha-Falcon (Watson, Wagh, and Popa 2023) and 3.7x improvement over Bicoptor (Zhou et al. 2023). As a result, the overall PPML model inference could be sped up by 3-4 times.

## Preliminaries

**Notations.** We use $:=$ to denote the definition. Considering a secret input $x \in [0, 2^\ell)$ is positive or negative, if $x \in [0, 2^{\ell_x})$ or $x \in (2^\ell - 2^{\ell_x}, 2^\ell)$, respectively. $\ell$ is the bit length of an element in $\mathbb{Z}_{2^\ell}$. $\ell_x$ is the precision bit length of $x$. $[x]^\ell$ refers to the shares of $x$ over ring $\mathbb{Z}_{2^\ell}$. We use the following specific 2-out-of-2 secret sharing for all protocols in this paper:

$$[x]_0^\ell := x + R \bmod 2^\ell,\ [x]_1^\ell := -R \bmod 2^\ell,$$

and $x = [x]_0^\ell + [x]_1^\ell \bmod 2^\ell$, $R$ is a random number belongs to $\mathbb{Z}_{2^\ell}$. $\xi$ is defined as the "absolute value" of $x$. Where $\xi := x \bmod 2^\ell$ for positive $x$ and $\xi := 2^\ell - x \bmod 2^\ell$ for negative $x$. The input $x$ is a fix-point number consisting of two parts: the fractional part and the integer part. The bit length of the fractional part is denoted by $\ell_x^{\mathsf{frac}}$, the bit length of the integer part is denoted by $\ell_x^{\mathsf{int}}$, and the total precision length of the input is denoted by $\ell_x := \ell_x^{\mathsf{int}} + \ell_x^{\mathsf{frac}}$.

**The Cut Function.** The function $\mathsf{cut}(\alpha, k)$ is similar to the right shifting operation, which cuts off the last $k$ bits of $\alpha$. Considering an integer $\alpha := \sum_0^{\ell-1} \alpha_i \cdot 2^i$, whose binary decomposition is $\alpha := \{\alpha_{\ell-1}, \cdots, \alpha_0\}$. Then, the binary form of the result is $\mathsf{cut}(\alpha, k) := \sum_k^{\ell-1} \alpha_i \cdot 2^i = \{\alpha_{\ell-1}, \cdots, \alpha_k\}$.

We further define a function $\mathsf{cut}(\alpha, k_1, k_2)$, which cuts the first $k_2$ bits and the last $k_1$ of $\alpha$. The binary form of the result is $\mathsf{cut}(\alpha, k_1, k_2) := \sum_{k_1}^{\ell-k_2-1} \alpha_i \cdot 2^i = \{\alpha_{\ell-k_2-1}, \cdots, x_{k_1}\}$.

**Truncation.** Truncation is used to recover the fixed-point decimal precision after a multiplication operation, which is a key component in approximate computation. There are several truncation protocols proposed in the literature, which could be distinguished into two types: probabilistic ($\mathsf{trc}_{\mathsf{prob.}}$) and deterministic ($\mathsf{trc}_{\mathsf{determ.}}$). Both types of truncation protocols suffer from a 1-bit error issue, caused by the carry bit generated by the truncated part, which is referred to as $e_0$.

Here is an example of using the probabilistic truncation protocol in SecureML (Mohassel and Zhang 2017) to truncate the last $k$ bits of the input $x \in \mathbb{Z}_{2^\ell}$ and resulting in $e_0$. [1]

$$\begin{aligned}
&x = 0100\ 1011, R = 1010\ 1010, \ell = 8, k = 4,\\
&[x]_0 = x + R \bmod 2^8 = 1111\ 0101,\\
&[x]_1 = -R \bmod 2^8 = 0101\ 0110\\
&\mathsf{trc}(x, 4)\\
=&(\mathsf{cut}([x]_0, 4) \bmod 2^8 - \mathsf{cut}(-[x]_1, 4) \bmod 2^8) \bmod 2^8\\
=&(0000\ 1111 - 0000\ 1010) \bmod 2^8 = 0000\ 0101
\end{aligned}$$

The expected outcome after truncation is $0000\ 0100$ and the real output is $0000\ 0101$. The occurrence of $e_0$ seems inevitable due to the nature of secret sharing.

In addition to $e_0$, probabilistic truncation also has another error, $e_1$, which can directly cause truncation failure. Here is another example to illustrate the significant deviation caused by $e_1$.

$$\begin{aligned}
&x = 0100\ 1011, R = 1110\ 0000, \ell = 8, k = 4,\\
&[x]_0 = x + R \bmod 2^8 = 0010\ 1011,\\
&[x]_1 = -R \bmod 2^8 = 0010\ 0000\\
&\mathsf{trc}(x, 4)\\
=&(\mathsf{cut}([x]_0, 4) \bmod 2^8 - \mathsf{cut}(-[x]_1, 4) \bmod 2^8) \bmod 2^8\\
=&(0000\ 0010 - 0000\ 1110) \bmod 2^8 = 1111\ 0100
\end{aligned}$$

The actual result of the truncation is $1111\ 0100$, which is far from the expected result of $0000\ 0100$.

To avoid the error of $e_1$ happening frequently and leading to more serious problems, we usually choose a larger ring size. This also means that when using probabilistic truncation, we need to be careful in selecting parameters and there will be some limitations, e.g., (Mohassel and Zhang 2017; Mohassel and Rindal 2018).

## Probabilistic Truncation in PPML

**The Truncation Proposed in Prior Works** The non-interactive probabilistic truncation protocol proposed in SecureML (Mohassel and Zhang 2017) is summarised in

[1] $\mathsf{trc}(x, 4)$ denotes truncating the last 4 bits of $x$ while preserving the sign. $\mathsf{cut}([x]_0, 4)$ denotes cutting the last 4 bits of the share $x_0$.

| Protocol | Offline | Round | Com | Original Com | 7-bit Com | Improvement |
|---|---|---|---|---|---|---|
| Falcon | Yes | $\log \ell_x + 4$ | $4(\log \ell_x + 1)\ell_x + 6\ell$ | $2,179$ bits | $884$ bits | $> 2\times$ |
| 3PC_Edabits | Yes | $\log \ell_x + 4$ | $72\ell_x + 12\ell_x \log \ell_x + 24\ell$ | $11,136$ bits | $3,444$ bits | $> 3\times$ |
| Orca | Yes | 1 | $2(\ell_x(\kappa{+}1{+}2){+}\kappa{+}1{+}4\ell_x){+}2$ | $17,280$ bits | $2,374$ bits | $> 7\times$ |
| ABY2.0 | Yes | 2 | $\ell_x\kappa + \ell_x + 4\ell$ | $8,512$ bits | $1,415$ bits | $> 6\times$ |
| Bicoptor | No | 2 | $2(\ell_x + 2) \cdot \ell + 5\ell$ | $8,768$ bits | $1,728$ bits | $> 5\times$ |
| Ours | No | 2 | $2(\ell_x + 1) \cdot (\ell_x + 1) + 5\ell$ | - | $704$ bits | - |

Table 1: The comparison of ReLU protocol communication cost between our ReLU protocol and that of other related works, adopting our new framework. The PPML runs in the ring size of $\ell = 64$. The scale precision used in PPML is $\ell_x = 7$. Original Com refers to the communication of the online phase using the original protocol. 7-bit Com refers to the communication that applies our framework to scale 7-bit DReLU input.

Alg. 1, and its correctness is illustrated by Theorem 1 is proved in (Mohassel and Zhang 2017, Sect. 4.1).

Algorithm 1: The Truncation Protocol Proposed in SecureML (Mohassel and Zhang 2017).

**Input**: shares of $x \in [0, 2^{\ell_x}) \bigcup (2^\ell - 2^{\ell_x}, 2^\ell)$ in $\mathbb{Z}_{2^\ell}$, number of bits to be truncated $k$

**Output**: shares of $\mathsf{trc}(x, k)$ in $\mathbb{Z}_{2^\ell}$

1: $P_0$ sets $[\mathsf{trc}(x, k)]_0 := \mathsf{cut}([x]_0, k) \bmod 2^\ell$.
2: $P_1$ sets $[\mathsf{trc}(x, k)]_1 := 2^\ell - \mathsf{cut}(2^\ell - [x]_1, k) \bmod 2^\ell$.

Theorem 1 describes the occurrence of $e_0$. We define the "slack" (Makri et al. 2021) as $\ell - \ell_x$. While invoking Alg. 1 in an MPC-based PPML protocol, we should choose a larger ring size to ensure enough slack.

**Theorem 1** *In a ring $\mathbb{Z}_{2^\ell}$, let $x \in [0, 2^{\ell_x}) \bigcup (2^\ell - 2^{\ell_x}, 2^\ell)$, where $\ell > \ell_x + 1$. Then the outputs of Alg. 1 satisfy the following results with probability $1 - \frac{1}{2^{\ell - \ell_x - 1}}$, where $\mathsf{bit} := \{0, 1\}$.*

- *For a positive $x$, $\mathsf{trc}(x, k) = \mathsf{cut}(\xi, k) + \mathsf{bit}$.*
- *For a negative $x$, $\mathsf{trc}(x, k) = 2^\ell - \mathsf{cut}(\xi, k) - \mathsf{bit}$*

ABY$^3$ (Mohassel and Rindal 2018) (Alg. 2) proposes an $n$-party interactive probabilistic truncation protocol. The participants first reconstruct the masked input, and locally truncate the opened value. Then, the participants remove the mask using a pre-shared element and obtain the final result. The protocol is summarised in Alg. 2, note that Alg. 2 is also probabilistic and hence constrained by the slack.

Algorithm 2: The Truncation Protocol Proposed in ABY$^3$ (Mohassel and Rindal 2018).

**Preprocessing**: the shares of $r' := \frac{r}{2^k}$

**Input**: shares of $x \in [0, 2^{\ell_x}) \bigcup (2^\ell - 2^{\ell_x}, 2^\ell)$ in $\mathbb{Z}_{2^\ell}$, number of bits to be truncated $k$

**Output**: shares of $\mathsf{trc}(x, k)$ in $\mathbb{Z}_{2^\ell}$

1: $P_i$ reconstructs $\alpha$ from $[x]_i + [r]_i$.[2]
2: $P_i$ sets $[\mathsf{trc}(x, k)]_i := \frac{\alpha}{2^k} - [r']$.

As mentioned in previous sections, the truncation protocols of both SecureML (Mohassel and Zhang 2017) and ABY$^3$ (Mohassel and Rindal 2018) are probabilistic, and therefore, they both suffer from $e_1$. We have already discussed the serious consequences of $e_1$. Following, we analyze how $e_1$ is generated. We first introduce a more fundamental cut function to better understand the nature of $e_1$, and then explain how this fundamental cut function evolves in these truncation protocols.

**A Fundamental Cut Function** In our analysis, we find that truncation protocols are based on combinations or variations of cut functions. Since the essence of the occurrence of $e_1$ is also on the cut function, we first formally define the cut function and provide the key properties of the cut function in Theorem 2 to better understand the generation of $e_1$ and to better understand the essence of truncation protocols.

**Definition 1** *For $\alpha, \beta \in \mathbb{Z}_{2^\ell}$, we define $\mathsf{LT}(\alpha, \beta) := 1$ iff $\alpha < \beta$, and $0$ otherwise.*

**Theorem 2** *For $\alpha, \beta \in \mathbb{Z}_{2^\ell}$ and $\mathsf{bit} := \{0, 1\}$,*

- $\mathsf{cut}(\alpha + \beta \bmod 2^\ell, k) = \mathsf{cut}(\alpha, k) + \mathsf{cut}(\beta, k) - \mathsf{LT}(\alpha + \beta, a) \cdot \mathsf{cut}(2^\ell, k) + \mathsf{bit} \bmod 2^\ell$
- $\mathsf{cut}(\alpha - \beta \bmod 2^\ell, k) = \mathsf{cut}(\alpha, k) - \mathsf{cut}(\beta, k) + \mathsf{LT}(a, \alpha - \beta) \cdot \mathsf{cut}(2^\ell, k) - \mathsf{bit} \bmod 2^\ell$

The proof of Theorem 2 could be found in Appendix.D.

We recall that the truncation protocol of SecureML (Mohassel and Zhang 2017) transfers the truncation operation on plaintext $x$ into the cut operations on shares $[x]_0 := x + R \bmod 2^\ell$ and $[x]_1 := -R \bmod 2^\ell$, i.e., $\mathsf{trc}(x, k) = \mathsf{cut}([x]_0, k) - \mathsf{cut}(2^\ell - [x]_1, k) \bmod 2^\ell$. By substituting $[x]_0 = x + R \bmod 2^\ell$ and $[x]_1 = -R \bmod 2^\ell$, Theorem 1 and Theorem 2, we obtain Corollary 1 as the following.

**Corollary 1** *In a ring $\mathbb{Z}_{2^\ell}$, let $x \in [0, 2^{\ell_x}) \bigcup (2^\ell - 2^{\ell_x}, 2^\ell)$, where $\ell > \ell_x + 1$. Then the outputs of Alg. 1 satisfy the following results, where $\mathsf{bit} := \{0, 1\}$.*

- *For a positive $x$, $\mathsf{trc}(x, k) = \mathsf{cut}(x, k) - \mathsf{LT}(x + R \bmod 2^\ell, x) \cdot \mathsf{cut}(2^\ell, k) + \mathsf{bit} \bmod 2^\ell$.*
- *For a negative $x$, $\mathsf{trc}(x, k) = -\mathsf{cut}(-x, k) + \mathsf{LT}(x, x + R \bmod 2^\ell) \cdot \mathsf{cut}(2^\ell, k) - \mathsf{bit} \bmod 2^\ell$.*

For positive $x$, we expect $\mathsf{trc}(x, k) = \mathsf{cut}(x, k) + \mathsf{bit} \bmod 2^\ell$. However, we observe that there is an additional term $\mathsf{LT}(x + R \bmod 2^\ell, x) \cdot \mathsf{cut}(2^\ell, k)$ in Corollary 1. When $x + R \bmod 2^\ell > x$, $\mathsf{LT}(x + R \bmod 2^\ell, x) = 0$, this extra

term disappears, which makes $\mathsf{trc}(x,k) = \mathsf{cut}(x,k)$ consistent with our expectation. When $x + R \bmod 2^\ell < x$, $\mathsf{LT}(x + R \bmod 2^\ell, x) = 1$, the error term exists causing the failure of truncation, also known as $e_1$. Similarly, we expect $\mathsf{trc}(x,k) = -\mathsf{cut}(-x,k) - \mathsf{bit} \bmod 2^\ell$ for negative $x$. $e_1$ occurs when $\mathsf{LT}(x, x + R \bmod 2^\ell) = 1$. Hence, we have

$$\begin{aligned}
&\mathsf{P}(\text{SecureML truncation failure}) = \mathsf{P}(e_1)\\
=&\mathsf{P}(x + R \bmod 2^\ell < x \mid x \in [0, 2^{\ell_x}))\\
=&\mathsf{P}(x < x + R \bmod 2^\ell \mid x \in (2^\ell - 2^{\ell_x}, 2^\ell))\\
=&\frac{1}{2^{\ell-\ell_x-1}}.
\end{aligned}$$

We further notice that, the larger the difference between $\ell$ and $\ell_x$, the lower the probability of failure in truncations.

The truncation protocol of ABY$^3$ (Mohassel and Rindal 2018) (Alg. 2) also suffers from $e_1$ as the truncation protocol of SecureML (Mohassel and Zhang 2017) (Alg. 1), and the underlying cause of the errors and its probability are the same i.e.,

$$\begin{aligned}
&\mathsf{P}(\text{ABY}^3 \text{ truncation failure}) = \mathsf{P}(e_1)\\
=&\mathsf{P}(\alpha = x + r \bmod 2^\ell < x \mid x \in [0, 2^{\ell_x}))\\
=&\mathsf{P}(x < \alpha = x + r \bmod 2^\ell \mid x \in (2^\ell - 2^{\ell_x}, 2^\ell))\\
=&\frac{1}{2^{\ell-\ell_x-1}}.
\end{aligned}$$

In conclusion, minimizing the probability of the occurrence of $e_1$ is equivalent to maximizing $\ell - \ell_x$. Hence, the parameter $\ell_x$ should be sufficiently smaller than $\ell$, i.e., $\ell_x \ll \ell$. We would like to emphasize that the above conclusion is based on an assumption that $R$, $r$ are uniformly and randomly chosen from the ring, which is also a necessary condition for security.

## Impact of $e_1$ on Accuracy in PPML

Based on the previous analysis, we understand that the occurrence of $e_1$ can cause a substantial deviation from the desired result, which can further affect the training and inference accuracy. We would like to point out that some existing works have used fixed numbers instead of random numbers declared by the protocol to simplify their implementation, which inadvertently hides $e_1$.

Next, we will provide a comprehensive explanation of why using fixed numbers instead of random numbers conceals the manifestation of $e_1$. Furthermore, we will present the actual inference accuracy when employing random numbers under the parameter recommendations provided in these works.

**The Implementation Bug Related to $e_1$ in Existing Works.** Many PPML works choose probabilistic truncation due to its non-interactive/less-interactive property while existing deterministic truncation protocols require additional communication overhead. According to the preceding introduction to the errors, $e_0$ appears to be a very minor error with only 1 bit, and we believe its impact on the accuracy of PPML tasks is negligible. On the other hand, $e_1$ is more severe and complex. Previous studies (Wagh 2022; Mishra et al. 2020; Ryffel et al. 2022; Wagh et al. 2021; Byali et al. 2020; Patra and Suresh 2020; Chaudhari, Rachuri, and Suresh 2020; Watson, Wagh, and Popa 2022; Tan et al. 2021) have not extensively addressed $e_1$, typically controlling its occurrence probability through a security parameter to confine its impact to a small, acceptable range on computation tasks. However, this approach does not fundamentally solve the problem of $e_1$ and may give rise to other problems. For instance, selecting appropriate security parameters requires careful consideration of each computation within the entire task. In complex computations, overlooking certain computation processes might lead to erroneous security parameter choices, resulting in a decrease in the accuracy of the computation task. Additionally, security parameters could become bottlenecks or limiting factors in the performance of certain protocols. Some existing works fail to discover that the parameters they select are insufficient to support the correctness of inference, as they use fixed numbers instead of random numbers in their truncation protocols.

| Model | Type | Fraction precision $\ell_x^{\mathsf{frac}} = 26$ | | |
|---|---|---|---|---|
| | | **SecureML** | **Falcon** | **Fantastic** |
| CIFAR10_AlexNet | mult-trc (f) | 69.63% | 69.63% | 69.63% |
| | mult-trc (r) | 12.74% | 12.73% | 12.74% |
| | trc-mult (r) | 69.62% | 69.64% | 69.62% |
| Tiny_AlexNet | mult-trc (f) | 26.39% | 26.39% | 26.39% |
| | mult-trc (r) | 0.45% | 0.44% | 0.45% |
| | trc-mult (r) | 26.35% | 26.35% | 26.35% |
| CIFAR10_VGG16 | mult-trc (f) | 88.31% | 88.31% | 88.31% |
| | mult-trc (r) | 10.60% | 9.89% | 10.60% |
| | trc-mult (r) | 88.29% | 88.29% | 88.35% |
| Tiny_VGG16 | mult-trc (f) | 54.89% | 54.89% | 54.89% |
| | mult-trc (r) | 0.43% | 0.41% | 0.50% |
| | trc-mult (r) | 54.92% | 54.92% | 54.88% |

Table 2: Accuracy when applying randoms in ABY$^3$ for truncation and applying the truncate-then-multiply solution in Piranha PPML inference implementations (Watson, Wagh, and Popa 2022, 2023), including P-SecureML (2-Party), P-Falcon (3-Party) and P-Fantastic (4-Party). Entries with (f) indicate the use of fixed numbers, while entries with (r) indicate the use of random numbers.

Looking purely from the perspective that $\ell_x$ should be much smaller than $\ell$, existing work has indeed chosen appropriate parameters. For example, in Piranha (Watson, Wagh, and Popa 2022, 2023), all P-SecureML, P-Falcon, and P-Fantastic implementations invoke the truncation protocol of ABY$^3$ (Mohassel and Rindal 2018) (Alg. 2) and the recommended precision $\ell_x^{\mathsf{frac}} = 26$ and $\ell = 64$ which does satisfy $\ell_x \ll \ell$. However, after a single multiplication operation, the size of $\ell_x$ is doubled. The new $\ell'_x = 2 \cdot \ell_x = 62$ if $\ell_x^{\mathsf{int}} = 5$, in which $\ell'_x$ is very close to $\ell = 64$. Theoretically, this would result in a very high probability of $e_1$ subsequently would lead to a decrease in inference accuracy. However, based

on the experimental results of the aforementioned work, this does not occur.

The reason for this is that the above works fix $r$ to be $2^{26}$ in Alg. 2. In this case, for positive $x$,

$$\mathsf{P}(e_1) = \mathsf{P}(\alpha = x + r \bmod 2^{\ell} < x | x \in [0, 2^{\ell_x})) = 0.\ ^{3}$$

Therefore, $e_1$ does not occur as expected. We replace the fixed $r$ in Alg. 2 with random numbers and rerun the experiments, and the results are exhibited in Tab. 2.

Fig. 4(see our full version) illustrates the impact of using random numbers(r) and fixed numbers(f) on inference accuracy under various models and parameters. When $\ell_x$ is getting close to $\ell$, $\mathsf{P}(e_1)$ increases and further leads to decreasing in inference accuracy. Fig. 4(a)-(d)(see our full version) illustrates the scenarios where $e_1$ may occur in a practical situation, that is, when random numbers are employed. It demonstrates that selecting the parameter combination $\ell_x^{\mathsf{frac}} = 26$ and $\ell = 64$, as claimed Piranha (Watson, Wagh, and Popa 2022, 2023), is not suitable. Similarly, Fig. 4(e) shows that $\ell_x^{\mathsf{frac}} = 13$ and $\ell = 32$ is also inappropriate. In addition, we demonstrate the impact of using fixed or random numbers on the inference accuracy for the 2-party, 3-party, and 4-party protocols when selecting $\ell_x^{\mathsf{frac}} = 26$ in Tab. 2.

To address the issue of $e_1$ occurring due to the product of two numbers approaching the length of $\ell$ and thereby affecting inference accuracy, an intuitive solution is to reduce the precision of both numbers before performing multiplication. In Appendix.B, we propose "truncate-then-multiply" as a replacement for "multiply-then-truncate". Our solution is described in Alg. 8(see our full version). Table. 2 demonstrates that the "truncate-then-multiply" approach can guarantee inference accuracy.

## Our New Technique

In this section, we propose a new technique to reduce the communication cost of ReLU while preserving the overall model accuracy. We first introduce a deterministic cut protocol as the building block of our new technique.

### A Non-interactive Deterministic Truncation Protocol

By Theorem 2, we know that the error term $e_1$ of the general cut function is always $\mathsf{LT}(\cdot,\cdot) \cdot \mathsf{cut}(2^{\ell}, k)$. The question is how could we eliminate this item. A natural way is to modulo the cut function by $2^{l-k}$, since $\mathsf{cut}(2^{\ell}, k) = 2^{l-k}$. Alg. 3 presents our new non-interactive deterministic truncation protocol. [4]

We reuse the example used in a previous section to demonstrate how $e_1$ can lead to serious errors, to better illustrate how the new truncation protocol (Alg. 3) eliminates $e_1$. Recall that, in the previous example, $x = 0100\ 1011$, $R = 1110\ 0000$, $x + R \bmod 2^8 = 00101011 < x$ and hence, $e_1$ occurs.

[3] If $x$ is a positive 62-bit long number and $r$ is $2^{26}$, $x+r \bmod 2^{26}$ will never be small than $x$.

[4] $\overline{\mathsf{trc}}(\cdot, k)$ truncates the last $k$ bits of the input over $\mathbb{Z}_{2^{\ell}}$, and the results are elements in $\mathbb{Z}_{2^{\ell-k}}$

Algorithm 3: The Non-interactive Deterministic Truncation Protocol.

**Input**: shares of $x \in [0, 2^{\ell_x}) \bigcup (2^{\ell} - 2^{\ell_x}, 2^{\ell})$ in $\mathbb{Z}_{2^{\ell}}$, number of bits to be truncated $k$
**Output**: shares of $\mathsf{trc}(x, k)$ in $\mathbb{Z}_{2^{\ell-k}}$

1: $P_0$ sets $[\overline{\mathsf{trc}}(x,k)]_0 := \mathsf{cut}([x]_0, k) \bmod 2^{\ell-k}$.
2: $P_1$ sets $[\overline{\mathsf{trc}}(x,k)]_1 := 2^{\ell} - \mathsf{cut}(2^{\ell} - [x]_1, k) \bmod 2^{\ell-k}$.

We further notice that all truncation protocols discovered so far are focusing on cutting the last few bits of input. Alg. 4 presents an extension of Alg. 3 which allows us to obtain the middle few bits of input, i.e., cut off the last few bits and the first few bits. Such truncation protocol remains non-interactive and deterministic.

Algorithm 4: The More General Non-interactive Deterministic Truncation Protocol.

**Input**: shares of $x \in [0, 2^{\ell_x}) \bigcup (2^{\ell} - 2^{\ell_x}, 2^{\ell})$ in $\mathbb{Z}_{2^{\ell}}$, first $k_1$ bits to be truncated and last $k_2$ bits to be truncated
**Output**: shares of $\mathsf{trc}(x, k)$ in $\mathbb{Z}_{2^{\ell-k_1-k_2}}$

1: $P_0$ sets

$$[\overline{\mathsf{trc}}(x, k_1, k_2)]_0 := \mathsf{cut}([x]_0, k_1, k_2) \bmod 2^{\ell-k_1-k_2}.$$

2: $P_1$ sets

$$[\overline{\mathsf{trc}}(x, k_1, k_2)]_1 := 2^{\ell} - \mathsf{cut}(2^{\ell} - [x]_1, k_1, k_2).$$

The correctness of Alg. 3 and Alg. 4 can be proven by the following theorem 3.

**Theorem 3** *For* $\alpha, \beta \in \mathbb{Z}_{2^{\ell}}$ *and* $\mathsf{bit} := \{0, 1\}$,

- $\mathsf{cut}(\alpha + \beta, k_1, k_2)$ *mod* $2^{\ell-k_1-k_2} = \mathsf{cut}(\alpha, k_1, k_2) + \mathsf{cut}(\beta, k_1, k_2) + \mathsf{bit}$ *mod* $2^{\ell-k_1-k_2}$;
- $\mathsf{cut}(\alpha - \beta, k_1, k_2)$ *mod* $2^{\ell-k_1-k_2} = \mathsf{cut}(\alpha, k_1, k_2) - \mathsf{cut}(\beta, k_1, k_2) - \mathsf{bit}$ *mod* $2^{\ell-k_1-k_2}$;

### Framework Overview

Our main observation is that most non-linear functions in PPML can use a scaled input instead of the original input without affecting the correctness of the overall model execution results. The typical cases are ReLU and Maxpool. For $\mathsf{ReLU}(x)$, which can be computed as $x \cdot \mathsf{DReLU}(x)$, it is easy to see that $\mathsf{DReLU}(x) = \mathsf{DReLU}(x \cdot s)$ for any positive $s$. For $\mathsf{Max}(x, y)$ which selects $x$ or $y$ depending on boolean check $x > y$, similarly, it is equivalent to check whether $x \cdot s > y \cdot s$. Without loss of generality, most nonlinear functions can be decomposed into similar structures.

The framework as illustrated in Fig. 1 takes the ReLU function as an example. We introduce a private parameter $s$ to each ReLU function, $s$ shall be calibrated at the training process. We call this step *post-training calibration*, and it should be performed on a calibration dataset that has the same distribution as the training dataset. In particular, we monitor the range of the input values to the ReLU function. For each intermediate input data $x_i$ of the non-linear operator with the $i^{\mathsf{th}}$ data in the calibration dataset, we denote the

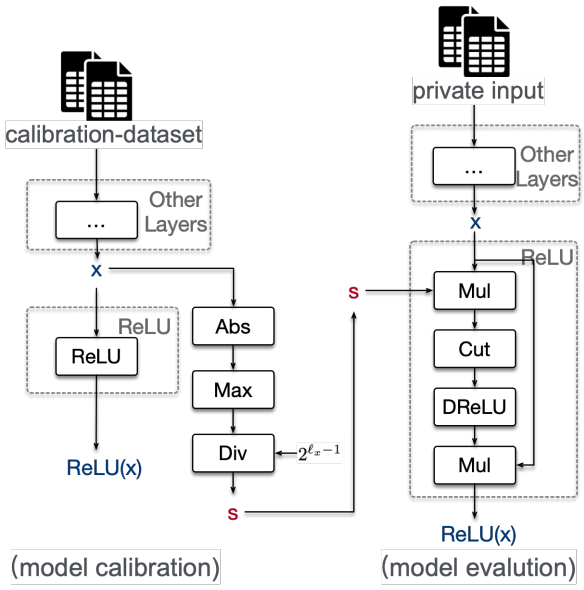


Figure 1: Structure of Our New Technique.

overall set of such input as $X := \{x_0, \ldots, x_{N-1}\}$, where $N$ is the size of calibration dataset. Let Max and Abs be the maximum and absolute functions, respectively. $s$ is calculated by

$$s = 2^{\ell_x - 1} / (\mathsf{Max}(\mathsf{Abs}(X))) \ ,$$

where $\ell_x$ is the bounded precision length, which will be used to scale the ReLU input to the lower effective range, denoted by $[-2^{\ell_x-1}, 2^{\ell_x-1}]$.

During the *privacy-preserving inference* procedure, we redefine $\mathsf{ReLU}(x)$ function is augmented to $\mathsf{ReLU}^*(x, s)$ with the scaling factor $s$. All parties first scale the input data, corresponding to the non-linear layer, to the smaller range $[-2^{\ell_x-1}, 2^{\ell_x-1}]$. After that, we utilize the aforementioned cut protocol to drop the insignificant bits and retain $\ell_x$ effective bits. Assume the private input of ReLU is $x$. Each party holds the secret sharing $[x]$ and the ReLU parameter $[s]$, where each share is in the ring $\mathcal{Z}_{2^\ell}$. We define $\mathsf{ReLU}^* := x \cdot \mathsf{DReLU}(\mathsf{cut}(x \cdot s, \ell - \ell_x - 2 \cdot \ell_x^{\mathsf{frac}}, 2 \cdot \ell_x^{\mathsf{frac}}))$. Instead of ReLU, we employ all parties to evaluate $\mathsf{ReLU}^*$.

Recall that $x \cdot s$ will duplicate the fractional part, corresponding to $2 \cdot \mathsf{frac}_x$. As mentioned previously, the integer range of $x \cdot s$ is $[-2^{\ell_x-1}, 2^{\ell_x-1}]$. If we cut the first $\ell - \ell_x - 2 \cdot \ell_x^{\mathsf{frac}}$ bits and the last $2 \cdot \ell_x^{\mathsf{frac}}$ of $x \cdot s$ to obtatin $x'$, it will retain the $\ell_x$ effective integer bits. Notice that $x'$ and $x$ keep the same sign bit, namely $\mathsf{DReLU}(x) = \mathsf{DReLU}(x')$. Consequently, all parties perform secure DReLU protocol on $\ell_x$ bits share $[x']$.

## Communication Reduction

We integrate our technique with several SOTA secure ReLU evaluation protocols (Wagh et al. 2021; Escudero et al. 2020; Jawalkar et al. 2023). Following our framework, for the 2PC protocols, all parties can perform the cut procedure locally with the non-interactive cut protocol as Alg. 4; with regard to 3PC protocols, the parties have to adopt the interactive cut protocol as Alg. 3. Table. 1 depicts the improvement of communication cost of the $\mathsf{ReLU}^*$ protocol of Falcon (Wagh et al. 2021), 3PC_Edabits (Escudero et al. 2020) , Orca (Jawalkar et al. 2023) , ABY2.0 (Patra et al. 2021), and Bicoptor (Zhou et al. 2023), respectively.

**Falcon (Wagh et al. 2021).** Falcon utilizes private comparison to realize the ReLU protocol. It requires $\log \ell_x + 4$ rounds and $4(\log \ell_x + 1)\ell_x + 6\ell$ bits communication, for the ReLU input size $\mathbb{Z}_{2^\ell}$ and the DReLU precision $\ell_x$ bits. Falcon uses 3PC replicated secret sharing, which introduces additional $3\ell$ communication overhead for the cut. When $\ell_x = 7$ and $\ell = 64$, the communication is $3 \cdot 64 + 4(\log 7 + 1) \cdot 7 + 6 \cdot 64 + 3 \cdot 64 = 880$ bits. Compared with their original protocol, communication volume has been reduced by 70%.

**3PC_Edabits (Escudero et al. 2020).** We analyze the communication of secure ReLU evaluation in 3PC Edabits under semi-honesty with share conversion. In order to apply our technology, we separately evaluated the DReLU protocol on $\mathbb{Z}_2$ and performed multiplication on $\mathbb{Z}_{2^\ell}$ to calculate ReLU. Since it is based on additive secret sharing, the cut protocol requires $6\ell$ bits communication of reconstruction, and the scaling of $s$ requires $12\ell$ bits communication. Therefore, its communication on 7 effective-bits is $72 \cdot 7 + 12 \cdot 7 \cdot \log 7 + 24 \cdot 64 + 6 \cdot 64 + 12 \cdot 64 = 3,444$ bits, which reduce 75% communication of its original version.

**Orca (Jawalkar et al. 2023)/ABY2.0 (Patra et al. 2021)/Bicoptor (Zhou et al. 2023).** For the 2PC protocols Orca, ABY2.0 and Bicoptor, our non-interactive cut protocol can be applied without any modification. After applying our technique, Orca, ABY2.0, and Bicoptor have achieved $8\times$, $7\times$, and $5\times$ communication optimization respectively.

# Low Communication ReLU Protocol

In this section, we propose a new ReLU protocol that is compatible with our framework. We will delve into how $e_1$ becomes a factor leading to performance bottlenecks in certain protocols due to the constraints imposed by parameter choices. Taking Bicoptor (Zhou et al. 2023) as an example, the work selected $\ell = 64$ to ensure enough slack, the one-pass dominating communication cost was about $\ell_x \cdot \ell = 31 \cdot 64 \approx 2,000$ bits. If there is no need to ensure a sufficiently large slack, $\ell$ could be chosen as 31, and the communication cost would be reduced by half. To address this problem, we adopt the aforementioned non-interactive deterministic cut protocol $\Pi_{\mathsf{cut}}$. We first revisit the core principle of using truncation protocol for sign determination in Bicoptor (Zhou et al. 2023), and theoretically demonstrate that $\Pi_{\mathsf{cut}}$ is also applicable to this principle. Addressing some potential security and correctness issues, we showcase our new DReLU protocol.

## The Principle of Sign Determination

We aim to replace the truncation protocol proposed by SecureML (Mohassel and Zhang 2017) in Bicoptor (Zhou et al. 2023) with our new protocol to address the issues caused by the probabilistic truncation protocol, and thus obtain a more efficient DReLU protocol. However, replacing the main sub-

Algorithm 5: Zero-preserving random mapping protocol.

**Input**: shares of $x$ in $\mathbb{Z}_{2^{\ell'}}$
**Output**: shares of $x$ in $\mathbb{Z}_p$, where $\log_2 p = \ell' + 1$.

1: $P_0$ sets $[x]_0 := 2^{\ell'} \bmod p$ if $[x]_0 = 0$, otherwise sets $[x]_0 := [x]_0 \bmod p$.
2: $P_1$ sets $[x]_1 := p + [x]_1 - 2^{\ell'} \bmod p$.

protocols in such a complex DReLU protocol is not a trivial task, and it needs to be further proven whether the new DReLU protocol with the replaced sub-protocol is still effective. In this sub-section, we will explain the core principles of the DReLU protocol in Bicoptor (Zhou et al. 2023) and provide the core theory, lemmas, and proofs for the DReLU protocol that apply to the new truncation protocol. We recall that the key idea is to determine the output of $\mathsf{trc}(x, \lambda)$ or $\mathsf{trc}(x, \lambda - 1)$, where $\lambda$ is the effective bit length of $\xi$ and $\xi$ is the "absolute value" of $x$. In the following theorem, we use $\overline{\mathsf{trc}}(x, \lambda - 1)$ instead of $\overline{\mathsf{trc}}(x, \lambda - 1, k)$ for $k \leq \ell - \lambda - 1$ because when truncating the effective bits, the outputs of are the same, either 0 or 1, regardless of the $k$. Hence, we can simply check the existence of 1 or $2^\ell - 1$ to determine the sign. i.e., Theorem 4, and the correctness is proved in (Zhou et al. 2023, Sect. 3.1).

**Theorem 4** *For an input $x \in \mathbb{Z}_{2^\ell}$ with precision of $\ell_x$, let $\xi := x$ if $x$ is positive, and let $\xi := 2^\ell - x \bmod 2^\ell$ if $x$ is negative. The binary form $\xi$ is defined as $\{\xi_{\ell_x - 1}, \xi_{\ell_x - 2}, \cdots, \xi_1, \xi_0\}$, where $\xi_i$ denotes the $i$-th bit of $\xi$. $\lambda$ is the effective bit length of $\xi$, i.e., $\xi_{\lambda-1} = 1$ and $\lambda + 1 < \ell$. Set $\xi := \xi' \cdot 2^k + \xi''$, where $\xi' \in [0, 2^{\ell_x - k})$ and $\xi'' \in [0, 2^k)$. We have that for any $\hat{\ell} \geq \lambda$, the following results hold:*

- *For a positive $x$, there exists positive numbers $\lambda'$ and $\lambda''$ ($\lambda' \leq \lambda'' \leq \ell_x$) satisfying $\overline{\mathsf{trc}}(\xi, j) = 1$ for $\lambda' \leq j \leq \lambda''$, and $\overline{\mathsf{trc}}(\xi, j) = 0$ for $j > \lambda''$.*
- *For a negative $x$, there exists positive numbers $\lambda'$ and $\lambda''$ ($\lambda' \leq \lambda'' \leq \ell_x$) satisfying $\overline{\mathsf{trc}}(2^\ell - \xi, j) = 2^\ell - 1$ for $\lambda' \leq j \leq \lambda''$, and $\overline{\mathsf{trc}}(2^\ell - \xi, j) = 0$ for $j > \lambda''$.*

To help us better understand Theorem 4, we provide the following example using the extended truncation protocol Alg. 4 for both positive and negative inputs. In this example, $\ell = 64$, $\ell_x = 7$, and $\lambda = 5$. When $x = 0...0010110$ is positive, $\overline{\mathsf{trc}}(x, 4, \ell - \ell_x - 4) = 0000001$, and $\overline{\mathsf{trc}}(x, k, \ell - \ell_x - k)$ for $k > 4$ would be 0 if no $e_0$ occurs. Similarly, for negative $x = 2^{64} - 0...0010110 = 1...1110110$, $\overline{\mathsf{trc}}(x, 4, \ell - \ell_x - 4) = 2^{64} - 1 \bmod 2^7 = 1111111$, and $\overline{\mathsf{trc}}(x, k, \ell - \ell_x - k)$ for $k > 4$ would be 0 if no $e_0$ occurs.

## Zero-preserving Random Mapping

In this subsection, we present the zero-preserving random mapping protocol and explain how it enhances our sign determination principle. From a theoretical perspective, we find that using the new truncation protocol is feasible to determine the sign of a number, again, sign determination is equivalent to DReLU. However, when it comes to a practically applicable DReLU protocol, replacing the truncation protocol with the new one still presents security and correctness issues. Therefore, we propose the zero-preserving random mapping protocol, which completes our new DReLU protocol.

From the security point of view, while working in $\mathbb{Z}_{2^{\ell'}}$ (e.g., $\ell' = \ell - k_1 - k_2$), the parity of a number remains after masking, more specifically, if the product of two numbers is odd, this means both two numbers are odd. Leaking the parity of data could cause other more serious issues. By switching the ring from $\mathbb{Z}_{2^{\ell'}}$ to $\mathbb{Z}_p$ for a prime $p$ could prevent this issue. From the correctness point of view, we want non-zero outputs remain non-zero after masking and zero outputs remain zero. However, while working in $\mathbb{Z}_{2^{\ell'}}$, one non-zero output could possibly become zero after multiplying a random number. For example, $16 \cdot 16 \bmod 2^8 = 0$. We present the zero-preserving random mapping protocol in Alg. 5:

It is easy to argue the correctness of Alg. 5. The aim of Alg. 5 is to ensure the output elements in $\mathbb{Z}_p$ are zero if and only if the input elements in $\mathbb{Z}_{2^{\ell'}}$ are zero. Considering $x = [x]_0 + [x]_1 \bmod 2^{\ell'} = 0$:

- If $[x]_0 \bmod 2^{\ell'} = 0$, then $[x]_1 \bmod 2^{\ell'} = 0$. $P_0$ sets $[x]_0 := 2^{\ell'} \bmod p$ and $P_1$ sets $[x]_1 := p - 2^{\ell'} \bmod p$. The output is $[x]_0 + [x]_1 = p = 0 \bmod p$.
- If $[x]_0 \bmod 2^{\ell'} \neq 0$, then $[x]_1 = 2^{\ell'} - [x]_0 \bmod 2^{\ell'}$. $P_0$ sets $[x]_0 := [x]_0 \bmod p$ and $P_1$ sets $[x]_1 := p + 2^{\ell'} - [x]_0 - 2^{\ell'} = p - [x]_0 \bmod p$. The output is $[x]_0 + [x]_1 =$

Algorithm 6: Improved DReLU Protocol.

**Setting**: $\ell$, $\ell_x$, and $p$. $P_0$ and $P_1$ share $\mathsf{seed}_{01}$.
**Input**: shares of $x$
**Output**: shares of DReLU($x$)

*// $P_0$,$P_1$ initialize.*
1: $P_0$ and $P_1$ generate a random bit $t$ using $\mathsf{seed}_{01}$.
2: $P_0$ and $P_1$ compute $[x] := (-1)^t \cdot [x] \bmod 2^\ell$.
3: $P_0$ and $P_1$ compute

$$[u_i] := [\overline{\mathsf{trc}}(x, i, \ell - \ell_x - i)] \bmod 2^{\ell_x}, \forall i \in [0, \ell_x].$$

4: $P_0$ and $P_1$ compute

$$[v_i] := [u_i] + [u_{i+1}] - 1 \bmod 2^{\ell_x}, \forall i \in [0, \ell_x - 1]$$

and $[v_{\ell_x}] := [u_{\ell_x}] - 1 \bmod 2^{\ell_x}$.
5: $P_0$ and $P_1$ run Alg. 5 to switch the ring from $\mathbb{Z}_{2^{\ell_x}}$ to $\mathbb{Z}_p$ for $[v_i]$.
6: $P_0$ and $P_1$ use $\mathsf{seed}_{01}$ to shuffle $[\{v_i\}] := \Pi([\{v_i\}])$.
7: $P_0$ and $P_1$ generate $\ell_x + 1$ numbers of random $\{r_i\}$ using $\mathsf{seed}_{01}$, where $r_i \in \mathbb{Z}_p^*, \mathbb{Z}_p^* := \mathbb{Z}_p / \{0\}$. Masking by performing $[\{w_i\}] := [\{v_i \cdot r_i\}] \bmod p$.
8: $P_0$ and $P_1$ send $[\{w_i\}]$ to $P_2$.
*// $P_2$ processes.*
9: $P_2$ reconstructs $\{w_i\}$, and sets DReLU$(x)' := 1$ if there is 0 in $\{w_i\}$, otherwise sets DReLU$(x)' := 0$.
10: $P_2$ responds $[\text{DReLU}(x)'] \in \mathbb{Z}_{2^\ell}$ to $P_0$ and $P_1$.
*// $P_0$ and $P_1$ finalize.*
11: $P_0$ and $P_1$ compute $[\text{DReLU}(x)] = [\text{DReLU}(x)' \oplus t] = t + (1 - 2t) \cdot [(\text{DReLU}(x)'] \bmod 2^\ell$.

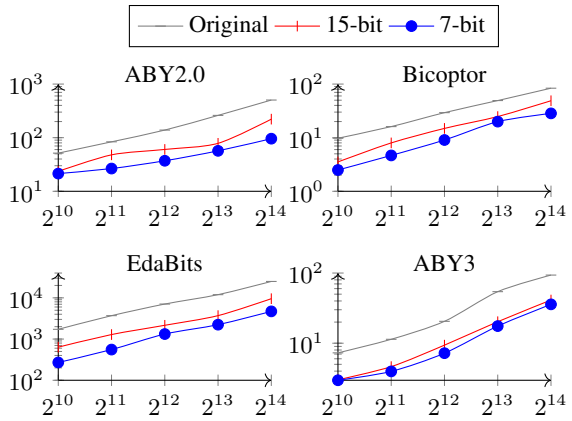


Figure 2: Overall run-time (ms) of ReLU in LAN setting, x-axis represents the number of ReLU inputs. 7-bit/15-bit refer to the 7-bit/15-bit quantization protocol.

$p = 0 \bmod p$.

## DReLU Protocol

With both deterministic cut protocol and zero-preserving random mapping protocol, we now present our new DReLU protocol in Alg. 6 and we will provide a step-by-step explanation. The overall system setting is that $P_0$ and $P_1$ locally compute arrays $[\{w_i\}]_0$ and $[\{w_i\}]_1$ then sent them to $P_2$. $P_2$ as an assisting party, helps reconstruct $\{w_i\}$ and returns an intermediate DReLU result to $P_0$ and $P_1$, who compute the final DReLU result.

**Step-by-Step Explanation**

- In step 1-2, $P_0$ and $P_1$ blind their input shares $[x]_0$ and $[x]_1$ using random bit $t$ generated from preshared $\mathsf{seed}_{01}$, i.e., the sign of the input has been randomly flipped.
- In step 3, as mentioned in Sect. , $\lambda$ is unknown and hence, $P_0$ and $P_1$ repeatedly compute $\ell_x$ number of times of truncations and obtain $[\{u_i\}]_0$ and $[\{u_i\}]_1$. Note that $[\{u_i\}]_0$ and $[\{u_i\}]_1$ are elements in $\mathbb{Z}_{2^{\ell_x}}$ instead of that in $\mathbb{Z}_{2^\ell}$ proposed in Bicoptor (Zhou et al. 2023), due to the usage of our new truncation protocol. We remove the $u_*$ term which was originally included in Bicoptor (Zhou et al. 2023). The purpose of $u_*$ was to help determine $\mathrm{DReLU}(0)$, but our ultimate goal is to improve the performance of ReLU and thus the end-to-end inference performance. Since $\mathrm{ReLU}(x) = \mathrm{DReLU}(x) \cdot x$, when $x = 0$, $\mathrm{ReLU}(0) = \mathrm{DReLU}(0) \cdot 0$, regardless of the result of $\mathrm{DReLU}(0)$, the final result of $\mathrm{ReLU}(0)$ is always 0, therefore removng the $u_*$ term will not effect the E2E PPML inference.
- In step 4, $P_0$ and $P_1$ locally perform adjacent pairwise addition on $[u_i]$, i.e., $[u_i] + [u_{i+1}]$ $\forall i \in [0, \ell_x - 1]$. Recall that the original summation proposed in Bicoptor (Zhou et al. 2023) was recursive summation, i.e., $\Sigma_{k=i}^{\ell_x}[u_k]$, $\forall i \in [0, \ell_x]$. This modification was made to enable better parallelization of the protocol and to reduce computational overhead.
- In step 5, $P_0$ and $P_1$ run Alg. 5 to switch the ring from $\mathbb{Z}_{2^{\ell_x}}$ to $\mathbb{Z}_p$ for $[v_i]$.
- In step 6-7, shuffling and masking are performed to avoid information leakage of the input. [5]
- In step 8, $P_0$ and $P_1$ reshare and send $[\{w_i\}]_0$ and $[\{w_i\}]_1$ to $P_2$, note that the one-pass dominating communication overhead is now reduced from $\ell_x \cdot \ell$ to $(\ell_x + 1) \cdot (\ell_x + 1)$ bits. In the case of $\ell = 64$ and $\ell_x = 31$, the communication overhead has been reduced by half.
- In step 9-10, $P_2$ reconstructs $\{w_i\}$ and outputs the intermediate result $[\mathrm{DReLU}(x)'] \in \mathbb{Z}_{2^\ell}$. $P_2$ then sends $[\mathrm{DReLU}(x)']_0$ and $[\mathrm{DReLU}(x)']_1$ back to $P_0$ and $P_1$, respectively.
- Finally, in step 11, $P_0$ and $P_1$ unblind $[\mathrm{DReLU}(x)']$ using the random bit $t$ and construct $[\mathrm{DReLU}(x)]$ locally, i.e., $[\mathrm{DReLU}(x)] = [\mathrm{DReLU}(x)' \oplus t] = t + (1 - 2t) \cdot [(\mathrm{DReLU}(x)'] \bmod 2^\ell$.

| Model | Protocol | $\mathrm{Time}^{\mathrm{PPML}}_{\mathrm{Total}}$ | $\mathrm{Time}^{\mathrm{PlainML}}_{\mathrm{Total}}$ |
|---|---|---|---|
| CIFAR10 | P-Falcon | 1.75s | 0.03s |
| AlexNet | Ours ($\ell_x = 7$) | 0.61s | |
| Tiny_ | P-Falcon | 7.81s | 0.03s |
| AlexNet | Ours ($\ell_x = 7$) | 2.41s | |
| CIFAR10 | P-Falcon | 30.48s | 0.29s |
| VGG16 | Ours ($\ell_x = 7$) | 7.12s | |
| Tiny_ | P-Falcon | out of memory | 0.29s |
| VGG16 | Ours ($\ell_x = 7$) | 28.80s | |
| CIFAR10 | P-Falcon | 14.77s | 0.13s |
| ResNet18 | Ours ($\ell_x = 7$) | 5.12s | |
| Tiny_ | P-Falcon | 20.46s | 0.14s |
| ResNet18 | Ours ($\ell_x = 7$) | 6.33s | |

Table 3: Performance comparison of E2E PPML inference and plaintext inference in LAN1 with batch size 128.

## The ReLU protocol

After obtaining the $\mathrm{DReLU}(x)$, computing $\mathrm{ReLU}(x) = x \cdot \mathrm{DReLU}(x)$ becomes relatively straightforward. A common approach is to use a Beaver triple (Beaver 1991) and work (Zhou et al. 2023) introduces a method to generate the shares of a triple using pre-shared seeds ($\mathsf{seed}_{02}$ and $\mathsf{seed}_{12}$). Algorithm. 7(see our full version) depicts our algorithm, where the multiplication is accomplished via preprocessed triples. For more information on how to generate triples during the online phase in ReLU protocols, please refer to Bicoptor (Zhou et al. 2023).

# Performance Evaluation

In this section, we benchmark our new technique with regard to performance improvement and the effect of accuracy on model inference. We then compare the performance

[5] More explanation can be found in Bicoptor (Zhou et al. 2023, Sect. 3.3).

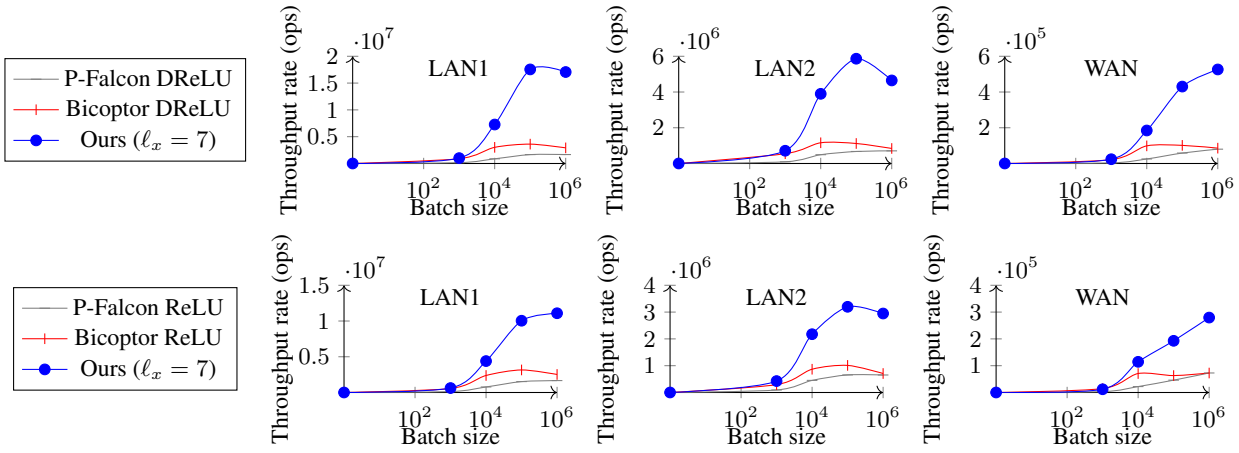


Figure 3: Performance comparisons of P-Falcon (Watson, Wagh, and Popa 2022, 2023) and Our DReLU and ReLU protocols on the different networks and batch sizes. (Graph)

of our DReLU/ReLU protocols based on our new technique with the performance of the original DReLU/ReLU protocol used in Piranha-Falcon (Watson, Wagh, and Popa 2022, 2023) and Piranha-Bicoptor (Zhou et al. 2023). Finally, we demonstrate the end-to-end PPML inference performance of the overall system through experimental results in Sect. .

We use three cloud server nodes to simulate three parties, each node with the following configuration: two Intel(R) Xeon(R) E5-2690 v4 @ 2.60GHz CPUs, 64 GiB memory, and one independent Nvidia Tesla P100 GPU. They are equipped with Ubuntu 16.04.7 and CUDA 11.4. We also simulate three different network environments: LAN1, LAN2, and WAN corresponded to 5Gbps/1Gbps/100Mbps bandwidth and 0.2ms/0.6ms/40ms round trip latency, respectively. Finally, we finish our experiments in several DNN models with parameter $\ell = 64$, $\ell_x = 7$ or $\ell_x = 15$.

**Benefit of Our Technique.** As mentioned before, our technique can significantly reduce the communication cost, attaining a diminution in communication cost surpassing 70%. Our benchmark validates that in real-world performance, our technique consistently yields benefits, with only marginal accuracy losses incurred. In our experiments, we validate that employing our technique to scale 15-bit DReLU does not affect the accuracy of our tested models. For applying 7-bit DReLU in our technique, we assessed the impact on model accuracy as depicted in Fig 5 (see our full version). In particular, under 7-bit ReLU, the accuracy loss from employing our technique can be contained within approximately 0.2%. In terms of performance, as depicted in Fig **??**, we evaluated the running time of EdaBits (Escudero et al. 2020), ABY$^3$ (Mohassel and Rindal 2018), ABY2.0 (Patra et al. 2021) and Bicoptor (Zhou et al. 2023) under 7-bit and 15-bit cuts. All protocols achieved performance improvements exceeding $5\times$ under 7-bit cut and over $3\times$ under 15-bit cut.

**DReLU/ReLU Unit Experiments.** We implement our DReLU and ReLU protocols and evaluate their performance with different batch sizes in various network environments. We compare our results with the DReLU/ReLU protocols in Piranha-Falcon (Watson, Wagh, and Popa 2022, 2023) and Piranha-Bicoptor (Zhou et al. 2023), which are shown in Tab. 4, Tab. 5(see our full version) and Fig. 3, respectively. We observe that our ultimate ReLU/DReLU protocol achieves over $10\times$ speed-up compared to Piranha-Falcon in all settings. For the Bicoptor scheme under the same 7-bit cut, our protocols achieve over $3\times$ performance improvement, of both ReLU and DReLU.

**PPML Inference Experiments.** We conduct experiments on end-to-end PPML inference using our ultimate ReLU protocol, comparing it to the original P-Falcon (Watson, Wagh, and Popa 2022, 2023) on sex different models under different network environments. The results of the experiments are presented in Fig. 6 and Tab. 6 (see our full version). We achieve a 3-4x improvement under different network conditions for different models. For example, by using our ReLU protocol in CIFAR10_AlexNet under LAN1, we achieved a total inference time of $4.4s$ while the original Piranha (Watson, Wagh, and Popa 2022, 2023) requires $16.72s$, demonstrating a 4x of improvement. This improvement narrows the performance gap between the PPML inference and plaintext inference to only a factor of 20 in model CIFAR10_AlexNet, and between 20-100x in other models (Tab. 3).

To further demonstrate the performance improvement brought about by our optimized ReLU protocol from a different perspective, we would like to recall that the cost of the ReLU layer accounts for around $80\%$ of the total inference cost. With our optimizations, this portion has been reduced to around $50\%$. Fig. 6 (see our full version) illustrates the percentage of ReLU cost for different models under LAN1 when using our optimized ReLU protocol.

## Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 62072401 and Grant 62232002.

## References


Beaver, D. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO 1991*, 420–432.

Byali, M.; Chaudhari, H.; Patra, A.; and Suresh, A. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *Proc. Priv. Enhancing Technol.*, 2020(2): 459–480.

Chaudhari, H.; Rachuri, R.; and Suresh, A. 2020. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS 2020*.

Escudero, D.; Ghosh, S.; Keller, M.; Rachuri, R.; and Scholl, P. 2020. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In *CRYPTO 2020*, 823–852.

Huang, Z.; Lu, W.; Hong, C.; and Ding, J. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *USENIX Security 2022*, 809–826.

Jawalkar, N.; Gupta, K.; Basu, A.; Chandran, N.; Gupta, D.; and Sharma, R. 2023. Orca: FSS-based Secure Training with GPUs. Cryptology ePrint Archive, Paper 2023/206.

Makri, E.; Rotaru, D.; Vercauteren, F.; and Wagh, S. 2021. Rabbit: Efficient Comparison for Secure Multi-Party Computation. In *FC 2021*, volume 12674, 249–270.

Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; and Popa, R. A. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *USENIX Security 2020*, 2505–2522.

Mohassel, P.; and Rindal, P. 2018. ABY$^3$: A Mixed Protocol Framework for Machine Learning. In *CCS 2018*, 35–52.

Mohassel, P.; and Zhang, Y. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *S&P 2017*, 19–38.

Patra, A.; Schneider, T.; Suresh, A.; and Yalame, H. 2021. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *USENIX Security 21*, 2165–2182.

Patra, A.; and Suresh, A. 2020. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS 2020*.

Rathee, D.; Rathee, M.; Kumar, N.; Chandran, N.; Gupta, D.; Rastogi, A.; and Sharma, R. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS 2020*, 325–342. ACM.

Ryffel, T.; Tholoniat, P.; Pointcheval, D.; and Bach, F. R. 2022. AriaNN: Low-Interaction Privacy-Preserving Deep Learning via Function Secret Sharing. *Proc. Priv. Enhancing Technol.*, 2022(1): 291–316.

Tan, S.; Knott, B.; Tian, Y.; and Wu, D. J. 2021. CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU. In *S&P 2021*, 1021–1038.

Wagh, S. 2022. Pika: Secure Computation using Function Secret Sharing over Rings. *Proc. Priv. Enhancing Technol.*, 2022(4): 351–377.

Wagh, S.; Tople, S.; Benhamouda, F.; Kushilevitz, E.; Mittal, P.; and Rabin, T. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proc. Priv. Enhancing Technol.*, 2021(1): 188–208.

Watson, J.; Wagh, S.; and Popa, R. A. 2022. Piranha: A GPU Platform for Secure Computation. In *USENIX Security 2022*, 827–844.

Watson, J.-L.; Wagh, S.; and Popa, R. A. 2023. Piranha source code. https://github.com/ucbrise/piranha/commit/dfbcb59d4e24ab69eb3606b49a102e602fdbee87. Accessed: 2014-10-25.

Zhou, L.; Wang, Z.; Cui, H.; Song, Q.; and Yu, Y. 2023. Bicoptor: Two-round Secure Three-party Non-linear Computation without Preprocessing for Privacy-preserving Machine Learning. In *S&P 2023*, 1295–1312.