Scalable Honest-majority MPC for Machine Learning from Mixed Secret Sharings

Meilin Li* Meng Hao† Yu Chen*

Abstract

Secure multi-party computation (MPC) provides a promising approach for privacy-preserving machine learning (ML). Existing solutions generally fall into two categories but face scalability and efficiency limitations. Protocols based on Shamir secret sharing (SS) incur high communication costs, while those relying on packed Shamir secret sharing (PS) remain largely theoretical and often require costly secret routing, especially for complex ML tasks.

In this work, we propose a mixed secret sharing strategy that leverages PS sharing for non-linear layers with repeated and independent operations, and SS sharing for linear layers such as matrix multiplications. To efficiently support alternating linear and non-linear computations, we design generic conversions between SS and PS sharings and further integrate them into the corresponding ML protocols, thereby eliminating additional communication and computation overhead. Moreover, we develop efficient PS sharing-based protocols for primitive non-linear building blocks, which enable multiple non-linear operations to be executed with essentially the same communication cost as a single operation.

We implement our framework for secure multi-party ML inference and conduct extensive experiments. Compared to the SOTA work LXY24 (USENIX Security '24), our approach reduces communication by 3.6-6.1×, while achieving 1.5-4.3× runtime improvement in the WAN setting and comparable or up to 2.3× better performance in the LAN setting.

1 Introduction

Secure Multi-Party Computation (MPC) [4, 16] enables a group of n parties $P_1, \ldots, P_n$ with corresponding private inputs $x_1, \ldots, x_n$ to jointly compute the output of a public function $f(x_1, \ldots, x_n)$ without revealing any information about their inputs beyond what can be derived from the output itself. Due to its desirable security guarantees, MPC techniques have been widely applied to privacy-preserving machine learning (PPML) [30], enabling multiple parties to collaboratively train or evaluate machine learning models without compromising the confidentiality of their data. This paradigm is especially relevant in domains such as healthcare [28], finance [38], and personalized services [34], where sensitive information must be protected while still enabling the utility of modern machine learning techniques.

However, existing MPC-based PPML protocols are mainly customized for a small number of parties, typically two-party scenarios [8, 23, 32, 35, 36], three-party scenarios [11, 29, 39, 40], and four-party scenarios [5, 9, 24]. While such settings are suitable for scenarios like client-server inference, they fall short in large-scale collaborative environments involving tens or even hundreds of parties such as federated healthcare [1] networks or multi-institutional financial analytics [38], where many parties jointly contribute sensitive data.

To address this limitation, Liu et al. [27] recently proposed the honest-majority MPC framework for PPML, based on Shamir secret sharing (abbreviated as *SS sharing* hereafter), which scales to a large number of parties (e.g., 63). Their protocols leverage the special properties of Mersenne primes and introduces optimized protocols for key machine learning building blocks, such as truncation and comparison. However, SS sharing inherently incurs substantial communication overhead, which increases rapidly with the number of parties. In bandwidth-constrained environments, this overhead becomes a critical bottleneck, severely limiting

*School of Cyber Science and Technology, Shandong University. State Key Laboratory of Cryptography and Digital Economy Security, Shandong University. Email: lml@mail.sdu.edu.cn, yuchen@sdu.edu.cn

†Singapore Management University. Email: menghao@smu.edu.sg

practical deployment. Moreover, a core component of their comparison protocol, i.e., the Prefix OR, was recently identified to have security flaws [41].

An alternative approach is to employ packed Shamir secret sharing (abbreviated as *PS sharing* hereafter) [3, 15] for PPML. Unlike standard SS sharing, PS sharing encodes k secrets into a single sharing, thereby enabling the parallel evaluation of k operations at the cost of only one operation. This packing property can significantly reduce communication overhead. However, existing works on PS sharing remain largely theoretical and have not yet seen practical adoption in PPML, primarily due to the challenge of *secret routing* [18]. Specifically, since each PS sharing encodes multiple secrets in a fixed order, operations involving cross-term dependencies, such as matrix multiplications, require collecting and reordering secrets with PS sharing. This reordering even eliminates the benefit of using PS sharing [14]. While recent studies have attempted to mitigate the secret routing overhead, these solutions either incur prohibitively large offline overhead [18] or impose restrictive assumptions on the circuit structure [3].

In summary, SS sharing-based protocols are well-suited for PPML but suffer from high communication costs, while PS sharing-based protocols are communication-efficient yet remain challenging to apply in complex machine learning tasks. Motivated by these limitations, we ask the following central question: *Can we leverage the complementary strengths of both SS and PS sharings to achieve scalable and communication-efficient multi-party protocols for machine learning?*

1.1 Our Contributions

In this work, we answer the above question affirmatively. We propose a mixed secret sharing strategy that strategically employs PS sharing for non-linear layers to amortize communication costs, while retaining SS sharing for linear layers to avoid expensive secret routing. To facilitate seamless transitions between these layers, we introduce generic conversions between SS and PS sharings, which may be of independent interest. More importantly, we integrate the sharing conversions into computation protocols by merging the secret opening operations, thereby eliminating the communication overhead incurred by the conversions. Complementing this framework, we develop PS-based protocols for non-linear primitives (Prefix OR, Bitwise-LT, Truncation, ReLU) that enable k parallel operations at the cost of a single operation. Compared to the state-of-the-art protocols LXY24 [27], we achieve $3.6 \sim 6.1\times$ reduction in the communication cost. For the runtime, our protocols achieve an improvement of $1.5 \sim 4.3\times$ in the WAN setting, while achieving comparable or up to $2.3\times$ better performance in the LAN setting.

1.2 Overview of Techniques

This section presents a high-level description of our techniques. In the following, we use $[x]_d$ to denote a degree-d SS sharing of $x \in \mathbb{F}_p$, and $[\![\mathbf{x}]\!]_d$ to denote a degree-d PS sharing of $\mathbf{x} \in \mathbb{F}_p^k$, where k is the number of secrets packed in the PS sharing. Besides, n is the number of parties.

Multi-party Protocols for ML from Mixed Sharings. Machine learning models generally consist of two types of layers: linear and non-linear layers. Linear layers perform affine transformations such as matrix multiplications, while non-linear layers apply activation functions (e.g., ReLU) element-wise. Our key observations are: (1) Non-linear layers repeatedly and independently apply the same function to multiple inputs, making them well-suited for PS sharing, which amortizes both communication and computation overhead; (2) In contrast, linear layers, typically realized as matrix multiplications, are inherently cross-term operations that do not benefit from PS sharing in the same way. Furthermore, recent works [10, 20] have demonstrated efficient evaluation of matrix multiplications using SS sharing.

Based on these insights, we propose a mixed secret sharing strategy that employs PS sharing for non-linear layers and SS sharing for linear layers. This design not only significantly reduces the communication cost of non-linear layers through PS sharing-based parallel evaluation, but also preserves the efficiency of SS sharing-based matrix multiplications in linear layers while avoiding costly secret routing.

Generic Conversions between SS and PS Sharings. To bridge alternating linear and non-linear layers in ML, we propose generic conversions between SS and PS sharings. These conversions may also be of independent interest, as they can be applied to other settings that require interoperability between different sharing types. Central to our conversions is to generate mixed randomness in the form of $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_d)$, where $\mathbf{r} = (r_1, \ldots, r_k) \in \mathbb{F}_p^k$. Specifically, we first generate SS sharings $[r_i]_d$ and PS sharings $[\![r_i \cdot \mathbf{1}]\!]_{n-k}$ for $i \in [1,k]$, where $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}_p^k$. Then, we compute the PS sharing $[\![\mathbf{r}]\!]_{n-1} := \sum_{i=1}^k \mathbf{e}_i \cdot [\![r_i \cdot \mathbf{1}]\!]_{n-k}$, where $\mathbf{e}_i$ is the unit vector with 1 in the i-th position and 0 elsewhere. Finally, we can apply the degree reduction technique to obtain the PS sharing $[\![\mathbf{r}]\!]_d$ from $[\![\mathbf{r}]\!]_{n-1}$.

With the above mixed randomness $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_d)$, we propose efficient SS-to-PS and PS-to-SS conversion protocols. We take the SS-to-PS conversion with input $([x_1]_d, \ldots, [x_k]_d)$ as an example. For $i \in [1,k]$, parties first mask SS sharings $[x_i]_d$ with

$[r_i]_d$ and reveal the masked sharing to a designated party called P_{king}. Then, P_{king} reconstructs $x_i + r_i \in \mathbb{F}_p$ for $i \in [1,k]$ and re-distributes the PS sharing $[\![\mathbf{x}+\mathbf{r}]\!]_d$, where $\mathbf{x}+\mathbf{r} = (x_1+r_1,\ldots,x_k+r_k) \in \mathbb{F}_p^k$. Finally, all parties locally compute the PS sharing $[\![\mathbf{x}]\!]_d := [\![\mathbf{x}+\mathbf{r}]\!]_d - [\![\mathbf{r}]\!]_d$.

Efficient Protocols for Packed Non-linear Functions. We propose several efficient protocols for non-linear building blocks in ML based on PS sharing, including Prefix OR, Bitwise Less Than, Truncation, and ReLU. This allows k non-linear operations to be executed with the same communication cost as a single operation. Below, we introduce our Prefix OR protocol, which can be used in LXY24 [27] to address their security flaws.

Given inputs $a_0,\ldots,a_{\ell-1}$, Prefix OR computes $b_i := \vee_{j=0}^{i} a_j$ for $i \in [0,\ell-1]$. Prior works [7,41] require $O(\ell \log \ell)$ multiplications to achieve Prefix OR, leading to substantial communication overhead. We propose an efficient packed Prefix Product protocol with only $O(\ell)$ multiplications while maintaining comparable communication rounds. The starting point is the naive sequential Prefix Product evaluation, which only requires ℓ multiplications but results in $\ell-1$ rounds. We observe that the prefix computation structure [25] allows the use of an additional ℓ multiplications to decouple the inherent sequential dependencies of the original procedure. This enables the parallel evaluation of multiple multiplication gates, thereby reducing the round complexity to $O(\log \ell)$. Concretely, our Prefix OR protocol requires 2ℓ multiplications and $2\log \ell$ rounds.

Eliminating the Cost of Sharing Conversions in ML. We further eliminate the additional communication overhead from sharing conversions by integrating them into the respective ML protocols. Specifically, we embed the SS-to-PS conversion into the matrix multiplication of linear layers and the PS-to-SS conversion into the ReLU of non-linear layers. Our key insight is that degree reduction occurs in both linear and non-linear protocols, and both degree reduction and sharing conversion are performed by opening masked secrets. By carefully combining these steps, we can reduce two openings into one, thereby saving communication of sharing conversions.

Below, we describe the high-level idea of the sharing conversion–integrated ReLU protocol. The key is to generate mixed randomness in the form $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_{n-1})$, where $\mathbf{r} = (r_1,\ldots,r_k) \in \mathbb{F}_p^k$. Specifically, we first generate SS sharings $[r_i]_d$ and PS sharings $[\![r_i \cdot \mathbf{1}]\!]_{n-k}$ for $i \in [1,k]$. Then, all parties locally compute $[\![\mathbf{r}]\!]_{n-1} := \sum_{i=1}^k \mathbf{e}_i \cdot [\![r_i \cdot \mathbf{1}]\!]_{n-k} + [\![\mathbf{0}]\!]_{n-1}$, where $[\![\mathbf{0}]\!]_{n-1}$ is a random degree-$(n-1)$ PS sharing of $\mathbf{0} = (0,\ldots,0) \in \mathbb{F}_p^k$, ensuring that $[\![\mathbf{r}]\!]_{n-1}$ is a random PS sharing. Using this mixed randomness $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_{n-1})$, we can combine degree reduction and PS-to-SS conversion into a single opening. Before the final step of ReLU (i.e., degree reduction), the intermediate output is $[\![\mathbf{x}]\!]_{d'}$, where $d \le d' \le n-1$. Similar to sharing conversion, P_{king} reconstructs the masked sharing $[\![\mathbf{x}+\mathbf{r}]\!]_{n-1}$, where $\mathbf{x}+\mathbf{r} := (x_1+r_1,\ldots,x_k+r_k) \in \mathbb{F}_p^k$, and re-distributes the SS sharings $[x_i+r_i]_d$ for $i \in [1,k]$. Finally, all parties locally compute the SS sharings $[x_i]_d := [x_i+r_i]_d - [r_i]_d$ for $i \in [1,k]$.

2 Preliminary

2.1 Notation

In this paper, we use lowercase letters (e.g., a) to represent scalars and bold lowercase letters (e.g., $\mathbf{a}$) to represent vectors, where a_i denotes the i-th component of $\mathbf{a}$. Similarly, $\mathbf{0}$ and $\mathbf{1}$ denote a vector with all entries set to 0 and 1, respectively. We use uppercase letters (e.g., M) to represent matrices, where m_{ij} denotes the element in the i-th row and j-th column of M. Unless otherwise specified, all operations on vectors are element-wise. For example, $\mathbf{x} \cdot \mathbf{y} = (x_1 \cdot y_1, \ldots, x_k \cdot y_k)$, namely the Hadamard product of $\mathbf{x}$ and $\mathbf{y}$. Besides, $[1,k]$ denotes the set $\{1,\ldots,k\}$.

2.2 Security Model

We use $\mathcal{P} = \{P_1,\ldots,P_n\}$ to denote the set of n parties. For every pair of parties, assume there exists a secure (private and authentic) synchronous channel. Similar to prior works [27,29,30], we consider static semi-honest adversaries in the honest-majority setting. Specifically, the adversary corrupts[1] t parties at the beginning of the protocol, where $t < (\frac{1}{2} - \varepsilon)n$ for a constant $0 < \varepsilon < \frac{1}{2}$, and follows the protocol specification exactly. However, the adversary may attempt to learn additional information by analyzing the transcripts of messages that the corrupted parties received and their internal states. All our protocols are proven in the standard simulation-based security model [6].

[1] We note that our protocols do not achieve maximal corruption tolerance in the honest majority setting (i.e., $t < \frac{n}{2}$) due to the reliance on packed Shamir sharing and DN multiplication [13]. In principle, this limitation can be overcome by adopting (packed) Beaver triple–based multiplication [2,14,19], which would allow our protocols to be secure against maximal corruption. We leave this extension as future work.

2.3 (Packed) Shamir Secret Sharing

Our protocols utilize both standard Shamir Secret Sharing (SS sharing) [37] and packed Shamir secret sharing (PS sharing) [15]. We use $[x]_d$ and $[\![\mathbf{x}]\!]_d$ to denote degree-d SS and PS sharings, respectively, where k is the number of secrets packed in PS sharing.

Shamir Secret Sharing. A degree-d SS sharing scheme consists of two algorithms $(\mathsf{Share}, \mathsf{Reconst})$ as follows:

- $\mathsf{Share}(x)$: Given a secret $x \in \mathbb{F}_p$ as input, randomly sample a polynomial $f(\cdot) \in \mathbb{F}_p[X]$ of degree at most d such that $f(0) = x$, and output the share $[x]_d^i := f(i)$ to each party P_i for $i \in [1,n]$. We denote $[x]_d := ([x]_d^1, \ldots, [x]_d^n)$.
- $\mathsf{Reconst}([x]_d)$: Given as input any size-$(d+1)$ subset of $[x]_d$, denoted as $s_1, \ldots, s_{d+1}$, output $x := \sum_{i=1}^{d+1} \delta_i(0) \cdot s_i$, where $\delta(\cdot)$ are the Lagrange basis polynomials and $\delta_i(a) = \prod_{j \in [1,d+1], j \neq i} (a-j)/(i-j)$.

SS sharing has the following properties:

- Security: For a degree-d SS sharing $[x]_d$ of x, any d shares are independent of the secret x.
- Linear homomorphism: For all $1 \leq d_1 \leq d_2 < n$ and any $x, y \in \mathbb{F}_p$, $[x+y]_{d_2} := [x]_{d_1} + [y]_{d_2}$.
- Multiplicative property: For all d_1, d_2 subject to $d_1 + d_2 < n$ and any $x, y \in \mathbb{F}_p$, $[x \cdot y]_{d_1+d_2} := [x]_{d_1} \cdot [y]_{d_2}$ with the increased degree.

Packed Shamir Secret Sharing. A degree-d PS sharing scheme consists of two algorithms $(\mathsf{PSShare}, \mathsf{PSReconst})$:

- $\mathsf{PSShare}(\mathbf{x})$: Given a vector $\mathbf{x} \in \mathbb{F}_p^k$ as input, where $\mathbf{x} = (x_1, \ldots, x_k)$, randomly sample a polynomial $f(\cdot) \in \mathbb{F}_p[X]$ of degree at most d such that $f(-j+1) = x_j$ for $j \in [1,n]$, and output the share $[\![\mathbf{x}]\!]_d^i := f(i)$ to each party P_i for $i \in [1,n]$. We denote $[\![\mathbf{x}]\!]_d := ([\![\mathbf{x}]\!]_d^1, \ldots, [\![\mathbf{x}]\!]_d^n)$.
- $\mathsf{PSReconst}([\![\mathbf{x}]\!]_d)$: Given as input any size-$(d+1)$ subset of $[\![\mathbf{x}]\!]_d$, denoted as $s_1, \ldots, s_{d+1}$, output $x_i := \sum_{i=1}^{d+1} \delta_i(-i+1) \cdot s_i$, where $\delta_i(a) = \prod_{j \in [1,d+1], j \neq i} (a-j)/(i-j)$.

Similarly, PS sharing has the following properties:

- Security: For a degree-d PS sharing $[\![\mathbf{x}]\!]_d$ of $\mathbf{x}$, any $d-k+1$ shares are independent of the secret $\mathbf{x}$.
- Linear homomorphism: For all $1 \leq d_1 \leq d_2 < n$ and any $\mathbf{x}, \mathbf{y} \in \mathbb{F}_p^k$, $[\![\mathbf{x}+\mathbf{y}]\!]_{d_2} = [\![\mathbf{x}]\!]_{d_1} + [\![\mathbf{y}]\!]_{d_2}$.
- Multiplicative property: For all d_1, d_2 subject to $d_1 + d_2 < n$ and any $[\![\mathbf{x} \cdot \mathbf{y}]\!]_{d_1+d_2} = [\![\mathbf{x}]\!]_{d_1} \cdot [\![\mathbf{y}]\!]_{d_2}$.

The multiplicative property implies that, for a packed share $[\![\mathbf{x}]\!]_d$ and a public vector $c \in \mathbb{F}_p^k$, all parties can locally compute $[\![\mathbf{c} \cdot \mathbf{x}]\!]_{d+k-1}$, since $\mathbf{c}$ can be viewed as a degree-$(k-1)$ packed share $[\![\mathbf{c}]\!]_{k-1}$. This property is referred to as multiplication-friendliness [19].

Single packed Shamir Secret Sharing. A degree-d sPS sharing $[\![x \cdot \mathbf{1}]\!]_d$ is a specialization of PS sharing, where the secrets of $[\![x \cdot \mathbf{1}]\!]_d$ are k copies of the same value $x \in \mathbb{F}_p$. Thus, sPS shraring follows the properties of PS shraring.

2.4 DN Multiplication

2.4.1 DN Protocol

Damgård and Nielsen proposed the renowned DN protocol [13] in the honest majority setting for semi-honest security. To evaluate a multiplication gate, all parties first need to prepare a pair of random sharings $([r]_d, [r]_{2d})$ of the same secret r, where the first sharing is a degree-d SS sharing and the second sharing is a degree-$2d$ SS sharing. Such a pair of sharings is referred to as a pair of double sharings. In [13], preparing a pair of random double sharings requires the communication of $4n$ elements.

For a multiplication gate, suppose the input sharings are denoted by $[x]_d$, $[y]_d$. To compute $[z]_d := [x \cdot y]_d$, a pair of random double sharings $([r]_d, [r]_{2d})$ is consumed. All parties first agree on a special party P_{king}. Then, all parties run the following steps:

1. All parties locally compute $[xy+r]_{2d} := [x]_d \cdot [y]_d + [r]_{2d}$.
2. P_{king} collects all shares and reconstructs the secret $xy+r$. Then P_{king} sends the value $xy+r$ to all other parties.
3. After receiving $xy+r$ from P_{king}, all parties locally compute $[z]_d := xy + r - [r]_d$.

2.4.2 Packed DN Protocol

The above DN protocol can be generalized to packed Shamir secret sharing. To compute the multiplication of two PS sharings $[\![\mathbf{x}]\!]_d$ and $[\![\mathbf{y}]\!]_d$, it needs a pair of random double packed sharings $([\![\mathbf{r}]\!]_d, [\![\mathbf{r}]\!]_{2d})$. Preparing such sharings requires the communication of $4n$ elements [14]. The multiplication runs the following steps:

1. All parties locally compute $[\![\mathbf{xy}+\mathbf{r}]\!]_{2d} = [\![\mathbf{x}]\!]_d \cdot [\![\mathbf{y}]\!]_d + [\![\mathbf{r}]\!]_d$.
2. P_{king} collects all shares and reconstructs secrets $\mathbf{xy}+\mathbf{r}$. P_{king} generates degree-$(k-1)$ PS sharing $[\![\mathbf{xy}+\mathbf{r}]\!]_{k-1}$ and distributes the shares to other parties.
3. All parties locally compute $[\![\mathbf{z}]\!]_d = [\![\mathbf{xy}+\mathbf{r}]\!]_{k-1} - [\![\mathbf{r}]\!]_d$.

We write the above packed DN multiplication functionality as $[\![\mathbf{xy}]\!]_d \leftarrow \mathcal{F}_{\text{Mult}}([\![\mathbf{x}]\!]_d, [\![\mathbf{y}]\!]_d)$.

2.5 Random Sharings

Random sharing. We utilize the technique from [13] to prepare random SS and PS sharings. Below, due to the same process for both SS and PS, we take SS as an example. All parties agree on a Vandermonde matrix $M^T \in \mathbb{F}_p^{n\times(n-t)}$. Each party P_i samples a random $s^{(i)}$ and secret-shares $[s^{(i)}]_d$ to other parties. All parties locally compute $([r^{(1)}]_d,\ldots,[r^{(n-t)}]_d)^T = M([s^{(1)}]_d,\ldots,s^{(n)}]_d)^T$ and output $([r^{(1)}]_d,\ldots,[r^{(n-t)}]_d)$. The functionalities $\mathcal{F}_{\text{Rand}}^{\mathsf{SS},d}$ and $\mathcal{F}_{\text{Rand}}^{\mathsf{PS},d}$ denote generating random SS and PS shares, respectively. Note that preparing random SS sharing of 0 or random PS sharing of $\mathbf{0}$ [17] also follow the same idea of the above process, which are denoted by functionalities $\mathcal{F}_{\text{Zero}}^{\mathsf{SS},d}$ and $\mathcal{F}_{\text{Zero}}^{\mathsf{PS},d}$, respectively.

Double random sharings. We utilize the technique from [13, 14] to prepare double random SS $([r]_d,[r]_{2d})$ and PS shares $(\llbracket r\rrbracket_d,\llbracket r\rrbracket_{2d})$. Below, due to the same process for both SS and PS, we take SS as an example. All parties agree on a Vandermonde matrix $M^T \in \mathbb{F}_p^{n\times(n-t)}$. Each party P_i samples a random $s^{(i)}$ and secret-shares $[s^{(i)}]_d$, $[s^{(i)}]_{2d}$ to other parties. All parties locally compute $([r^{(1)}]_d,\ldots,[r^{(n-t)}]_d)^T = M([s^{(1)}]_d,\ldots,s^{(n)}]_d)^T$ and $([r^{(1)}]_{2d},\ldots,[r^{(n-t)}]_{2d})^T = M([s^{(1)}]_{2d},\ldots,s^{(n)}]_{2d})^T$, then output $([r^{(1)}]_d,\ldots,[r^{(n-t)}]_d)$ and $([r^{(1)}]_{2d},\ldots,[r^{(n-t)}]_{2d})$. The functionalities $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},d,2d}$ and $\mathcal{F}_{\text{DouRand}}^{\mathsf{PS},d,2d}$ denote generating double random SS and PS shares, respectively.

Random Bit-wise Sharing. The work [12] proposed the random bit-wise sharing protocol to generate a random Shamir sharing $[r]_d$ and Shamir sharings $[r_1]_d,\ldots,[r_\ell]_d$ of decomposed bits, where $r=\sum_{i=0}^{\ell-1}2^i\cdot r_{i+1} \bmod p$. We extend it to generate random packed bit-wise shares, namely, $\llbracket\mathbf{r}\rrbracket_d$ and $\llbracket\mathbf{r}_1\rrbracket_d,\ldots,\llbracket\mathbf{r}_\ell\rrbracket_d$, where each $\mathbf{r}_i\in\{0,1\}^k$, $\mathbf{r}\in\mathbb{F}_p^k$ and $\mathbf{r}=\sum_{i=0}^{\ell-1}2^i\cdot\mathbf{r}_{i+1} \bmod p$. Specifically, the parties generate each $\llbracket\mathbf{r}_i\rrbracket_d$ by computing $\frac{\mathbf{a}(\sqrt{\mathbf{a}^2})^{-1}+1}{2}$ on a packed random sharing $\llbracket\mathbf{a}\rrbracket_d \leftarrow \mathcal{F}_{\text{Rand}}^{\mathsf{PS},d}$, in which $\sqrt{\mathbf{a}^2}$ be the unique vector in $\{1,\ldots,(p-1)/2\}^k$ so that $\frac{\mathbf{a}(\sqrt{\mathbf{a}^2})^{-1}+1}{2}$ is uniformly random in $\{0,1\}^k$. Then the parties generate $\llbracket\mathbf{r}\rrbracket_d=\sum_{i=0}^{\ell-1}\mathbf{2}^i\cdot\llbracket\mathbf{r}_{i+1}\rrbracket_d$ locally. We write the above process as $(\{\llbracket\mathbf{r}_i\rrbracket_d\}_{i=1}^{\ell},\llbracket\mathbf{r}\rrbracket_d)\leftarrow\mathcal{F}_{\text{SolvedBits}}$, which can be done with communication of $2\ell(3n+4k-1)$ field elements in the preprocessing phase when $d=(n-1)/2$.

Reduce Communication Complexity with PRG. The communication complexity of distributing sharings can be reduced by utilizing pseudo-random generators (PRG) [17, 26, 31]. When a party P_i needs to distribute a random packed sharing $\llbracket\mathbf{r}\rrbracket_{d'}$, where $\mathbf{r}=(r_1,\ldots,r_k)$, P_i first sends random field elements to the first $d'-k+1$ parties $\{P_1,P_2,...,P_{d'-k+1}\}$(which is achieved by using PRG without communication). Then P_i reconstructs the whole sharing $\llbracket\mathbf{r}\rrbracket_{d'}$ using k secrets $(r_1,\ldots,r_k)$ and the shares of the first $d'-k+1$ parties, and send the shares to the rest parties.

3 Generic Conversions between (Packed) Shamir Secret Sharing

In this section, we present generic conversions between Shamir secret sharing (SS sharing) and packed Shamir secret sharing (PS sharing). These conversions are essential for bridging different layers in ML, where SS sharing is applied to linear layers and PS sharing to non-linear layers. As we will show in Section 5.2, the conversions can be integrated into the respective protocols for both layers without incurring additional communication overhead.

We first present the parameters used in SS and PS sharings. Let n be the number of parties and k be the number of packed secrets in PS sharing, where $1\le k\le d$. For both sharings, same as previous works [13, 15], we set the polynomial degree to $d:=\frac{n-1}{2}$, which aims to increase the corruption threshold while satisfying the DN07 multiplication constraint that $2d+1\le n$ [13]. The corruption threshold is $t:=d+1-k$.

3.1 Mixed Randomness

We propose an efficient method for generating randomness of the mixed SS and PS sharings $\left(\{[r_i]_{d_1}\}_{i=1}^k,\llbracket\mathbf{r}\rrbracket_{d_2}\right)$ such that $\mathbf{r}=(r_1,\ldots,r_k)\in\mathbb{F}_p^k$, where $d_1\ge t$ and $d_2\ge t+k-1$. This mixed randomness serves as a critical building block to design the conversion protocols in Section 3.2. The functionality is shown in Figure 1. Our key idea is that we first prepare k pairs $\left([r_i]_{d_1},\llbracket r_i\cdot\mathbf{1}\rrbracket_{n-k}\right)$ for $i\in[1,k]$, where $\mathbf{1}:=(1,\ldots,1)\in\mathbb{F}_p^k$. We then compute a degree-$(n-1)$ PS sharing as $\llbracket\mathbf{r}\rrbracket_{n-1}:=\sum_{i=1}^k\mathbf{e}_i\cdot\llbracket r_i\cdot\mathbf{1}\rrbracket_{n-k}$, where $\mathbf{e}_i\in\mathbb{F}_p^k$ denotes the unit vector with 1 in the i-th coordinate and 0 elsewhere. From $\llbracket\mathbf{r}\rrbracket_{n-1}$, we can efficiently obtain a degree-d_2 PS sharing $\llbracket\mathbf{r}\rrbracket_{d_2}$ using a degree-reduction protocol described in Section 4. As an optimization, when $d_2=n-1$, we can directly set $\llbracket\mathbf{r}\rrbracket_{d_2}:=\llbracket\mathbf{r}\rrbracket_{n-1}+\llbracket\mathbf{0}\rrbracket_{n-1}$, thereby avoiding the degree reduction step. Here, $\llbracket\mathbf{0}\rrbracket_{n-1}$ is obtained by invoking $\mathcal{F}_{\text{Zero}}^{\mathsf{SS},n-1}$.

To generate the randomness $\left([r_i]_{d_1},\llbracket r_i\cdot\mathbf{1}\rrbracket_{n-k}\right)$, we present an efficient protocol $\Pi_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$ in Figure 3. The functionality is shown in Figure 2. This protocol extends the classic double randomness generation protocol [13], which is limited to the SS sharing. Moreover, we incorporate PRGs to further optimize the protocol's communication overhead. We set the degree of

Functionality $\mathcal{F}_{\text{MixRand}}^{d_1,d_2}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d_1 of SS sharing and the degree d_2 of PS sharing, where $d_1 \geq t$, $d_2 \geq t+k-1$.

Functionality.

1. $\mathcal{F}_{\text{MixRand}}^{d_1,d_2}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{MixRand}}^{d_1,d_2}$ randomly samples a vector $\mathbf{r} = (r_1, \ldots, r_k) \in \mathbb{F}_p^k$.
3. $\mathcal{F}_{\text{MixRand}}^{d_1,d_2}$ receives from the adversary a set of shares $\{s_i^{(1)}, \ldots, s_i^{(k)}, s_i\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{MixRand}}^{d_1,d_2}$ randomly samples degree-d_1 SS sharings $([r_1]_{d_1}, \ldots, [r_k]_{d_1})$ and a degree-d_2 PS sharing $[\![\mathbf{r}]\!]_{d_2}$ such that for all $P_i \in \mathcal{C}$, the i-th share of $([r_1]_{d_1}, \ldots, [r_k]_{d_1})$ is $(s_i^{(1)}, \ldots, s_i^{(k)})$ and the i-th share of $[\![\mathbf{r}]\!]_{d_2}$ is s_i.
5. $\mathcal{F}_{\text{MixRand}}^{d_1,d_2}$ distributes the shares of $([r_1]_{d_1}, \ldots, [r_k]_{d_1})$ and $[\![\mathbf{r}]\!]_{d_2}$ to the honest parties.

Figure 1: Ideal functionality for mixed randomness.

Functionality $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d_3 of SS sharing and the degree d_4 of sPS sharing, where $d_3 \geq t$, $d_4 \geq t+k-1$.

Functionality.

1. $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$ randomly samples $r \in \mathbb{F}_p$.
3. $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$ receives from the adversary a set of shares $\{s_i, s_i'\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$ randomly sample a degree-d_3 SS sharings $[r]_{d_3}$ and a degree-d_4 sPS sharing $[\![r \cdot \mathbf{1}]\!]_{d_4}$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[r]_{d_3}$ is s_i and the i-th share of $[\![r \cdot \mathbf{1}]\!]_{d_4}$ is s_i'
5. $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$ distributes the shares of $[r]_{d_3}$ and $[\![r \cdot \mathbf{1}]\!]_{d_4}$ to the honest parties.

Figure 2: Ideal functionality for generalized double randomness.

Protocol $\Pi_{\text{DouRand}}^{\mathsf{SS},\mathsf{sPS},d_3,d_4}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d_3 of SS sharing and the degree d_4 of sPS sharing, where $d_3 \geq t$, $d_4 \geq t+k-1$. $M \in \mathbb{F}_p^{(t+1)\times n}$ is a Vandermonde matrix.

Protocol.

1. For $i \in [1,n]$, each party P_i samples a random value $s_i \in \mathbb{F}_p$ and distributes the SS sharing $[s_i]_{d_3}$ and the sPS sharing $[\![s_i \cdot \mathbf{1}]\!]_{d_4}$ to other parties.
2. All parties locally compute $([r_1]_{d_3}, \ldots, [r_{n-t}]_{d_3})^T := M([s_1]_{d_3}, \ldots, [s_n]_{d_3})^T$ and $([\![r_1 \cdot \mathbf{1}]\!]_{d_4}, \ldots, [\![r_{n-t} \cdot \mathbf{1}]\!]_{d_4})^T := M([\![s_1 \cdot \mathbf{1}]\!]_{d_4}, \ldots, [\![s_n \cdot \mathbf{1}]\!]_{d_4})^T$.

Figure 3: Protocol for generalized double randomness.

$[\![r_i \cdot \mathbf{1}]\!]_{n-k}$ to $n-k$ for two reasons: (1) with PRGs, higher-degree random sharings can be generated at lower communication cost, and (2) $n-k$ is the maximal degree required to correctly compute $\mathbf{e}_i \cdot [\![r_i \cdot \mathbf{1}]\!]_{n-k}$.

The mixed randomness protocol $\Pi_{\text{MixRand}}^{d_1,d_2}$ is described in Figure 4, which invokes the degree reduction protocol $\Pi_{\text{DegRed}}^{d_2,n-1}$ detailed in Section 4.

Protocol $\Pi^{d_1,d_2}_{\text{MixRand}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d_1 of SS sharing and the degree d_2 of PS sharing, where $d_1 \geq t$, $d_2 \geq t+k-1$.

Protocol.

1. All parties invoke $\mathcal{F}^{\mathsf{SS},\mathsf{sPS},d_1,n-k}_{\text{DouRand}}$ to receive $([r_i]_{d_1}, [\![r_i \cdot \mathbf{1}]\!]_{n-k})$ for $i \in [1,k]$.
2. All parties compute $[\![\mathbf{r}]\!]_{n-1} := \sum_{i=1}^{k} \mathbf{e}_i \cdot [\![r_i \cdot \mathbf{1}]\!]_{n-k}$, where each $\mathbf{e}_i \in \mathbb{F}^k$ is a unit vector with the i-th entry being 1.
3. All parties input $[\![\mathbf{r}]\!]_{n-1}$ to $\mathcal{F}^{d_2,n-1}_{\text{DegRed}}$ and receive $[\![\mathbf{r}]\!]_{d_2}$.

Figure 4: Protocol for generating mixed randomness.

Functionality $\mathcal{F}_{\text{SS-to-PS}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{SS-to-PS}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{SS-to-PS}}$ receives the shares of $[x_i]_d$, for $i \in [1,k]$, from honest parties and reconstructs the secrets $\mathbf{x} := (x_1, \ldots, x_k)$.
3. $\mathcal{F}_{\text{SS-to-PS}}$ receives from the adversary a set of shares $\{s_i\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{SS-to-PS}}$ samples a random degree-d PS sharing $[\![\mathbf{x}]\!]_d$ s.t. for all $P_i \in \mathcal{C}$, the i-th share of $[\![\mathbf{x}]\!]_d$ is s_i.
5. $\mathcal{F}_{\text{SS-to-PS}}$ distributes the shares of $[\![\mathbf{x}]\!]_d$ to honest parties.

Figure 5: Ideal functionality for SS-to-PS conversion.

Theorem 1. *Protocol $\Pi^{d_1,d_2}_{MixRand}$ securely computes $\mathcal{F}^{d_1,d_2}_{MixRand}$ in the $(\mathcal{F}^{\mathsf{SS},\mathsf{sPS}}_{DouRand}, \mathcal{F}^{d_2,n-1}_{DegRed})$-hybrid model against a semi-honest adversary who controls at most $t := d-k+1$ parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The simulator works as follows:

1. In step 1, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}^{\mathsf{SS},\mathsf{sPS}}_{\text{DouRand}}(d_1, n-k)$ and learns the shares of $([r_i]_{d_1}, [\![r_i \cdot \mathbf{1}]\!]_{n-k})$ of corrupted parties, for $i \in [1,k]$ and send them to $\mathcal{F}_{\text{MixRand}^{d_1,d_2}}$.
2. In step 2, $\mathcal{S}$ follows the protocol and computes the shares of $[\![\mathbf{r}]\!]_{n-1}$ of corrupted parties.
3. In step 3, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}^{d_2,n-1}_{\text{DegRed}}$ and receives the shares of $[\![\mathbf{r}]\!]_{d_2}$ of corrupted parties and send them to $\mathcal{F}^{d_1,d_2}_{\text{MixRand}}$.

Note that through the protocol, corrupted parties do not receive any messages from honest parties. Therefore, it is sufficient to argue that $\mathcal{S}$ perfectly simulates the behavior of honest parties. The secrets r_i for $i \in [1,k]$ and $\mathbf{r}$ in the real world have the same distribution as those computed by $\mathcal{F}^{d_1,d_2}_{\text{MixRand}}$. As for the shares held by corrupted parties, they are chosen by the adversary in both worlds. Thus, the distribution of the output of honest parties in both worlds are identical.

□

3.2 Conversions between SS and PS sharings

We present efficient protocols for conversions between SS and PS sharings. Let P_{king} denote a designated party (e.g., P_1) agreed upon by all parties in the beginning. The key idea is to prepare the mixed randomness $\left(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_d\right)$ in the offline phase. We provide detailed protocols in the following.

SS-to-PS Conversion. We present the SS-to-PS conversion functionality in Figure 5 and its protocol in Figure 6. The parties first generate mixed randomness $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_d)$, where $\mathbf{r} = (r_1, \ldots, r_k) \in \mathbb{F}^k$. The goal is to convert the SS sharings $([x_1]_d, \ldots, [x_k]_d)$ into a PS sharing $[\![\mathbf{x}]\!]_d$, where $\mathbf{x} := (x_1, \ldots, x_k)$. Specifically, for $i \in [1,k]$, the parties first reveal the shares $[x_i + r_i]_d$ to P_{king}. P_{king}

Protocol $\Pi_{\text{SS-to-PS}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Inputs. All parties input $[x_i]_d$, for $i \in [1,k]$.

Protocol.

1. All parties invoke $\mathcal{F}_{\text{MixRand}}^{d,d}$ to receive $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_d)$.
2. All parties locally compute $[x_i + r_i]_d := [x_i]_d + [r_i]_d$, for $i \in [1,k]$, and send their shares to P_{king}.
3. P_{king} reconstructs $x_i + r_i \in \mathbb{F}_p$ for $i \in [1,k]$, generates PS sharing $[\![\mathbf{x}+\mathbf{r}]\!]_d$ and distributes the shares to other parties.
4. All parties locally compute $[\![\mathbf{x}]\!]_d := [\![\mathbf{x}+\mathbf{r}]\!]_d - [\![\mathbf{r}]\!]_d$.

Figure 6: Protocol for SS-to-PS conversion.

reconstructs $x_i + r_i$ for $i \in [1,k]$ and re-distributes the packed Shamir share $[\![\mathbf{x}+\mathbf{r}]\!]_d$. Finally, all parties locally compute the packed Shamir share $[\![\mathbf{x}]\!]_d := [\![\mathbf{x}+\mathbf{r}]\!]_d - [\![\mathbf{r}]\!]_d$.

Theorem 2. *Protocol $\Pi_{SS\text{-}to\text{-}PS}$ securely computes $\mathcal{F}_{SS\text{-}to\text{-}PS}$ in the $\mathcal{F}_{MixRand}^{d,d}$-hybrid model against a semi-honest adversary who controls $t = d-k+1$ parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. The simulator works as follows:

1. In step 1, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{\text{MixRand}}^{d,d}$ and learns the shares of corrupted parties.
2. In step 2, $\mathcal{S}$ computes the shares of $[x_i]_d + [r_i]_d$ of corrupted parties for $i \in [1,k]$. $\mathcal{S}$ randomly samples a random vector $\mathbf{r}_i = (r_i, r_i^{(1)}, \ldots, r_i^{(k-1)})$ and then randomly samples the shares of honest parties based on the secrets $\mathbf{r}_i$ and the shares of corrupted parties for $i \in [1,k]$.
3. In step 3, if P_{king} is honest, $\mathcal{S}$ honestly follows the protocol. If P_{king} is corrupted, $\mathcal{S}$ sends the shares of $[x_i]_d + [r_i]_d$ of honest parties to P_{king}. P_{king} also receives the shares from corrupted parties. Then P_{king} can reconstruct $x_i + r_i$ for $i \in [1,k]$ and generate the packed sharing $[\![\mathbf{x}+\mathbf{r}]\!]_d$. After $\mathcal{S}$ receives the shares of $[\![\mathbf{x}+\mathbf{r}]\!]_d$ of honest parties, it recover the whole sharing $[\![\mathbf{x}+\mathbf{r}]\!]_d$ and learns the shares of corrupted parties.
4. In step 4, $\mathcal{S}$ follows the protocol and computes the shares of $[\![\mathbf{x}]\!]_d$ of corrupted parties and send them to $\mathcal{F}_{\text{SS-to-PS}}$.

Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties. Observe that $[r_i]_d$ is a random degree-d Shamir sharing. Therefore, $[x_i]_d + [r_i]_d$ is a random degree-d SS sharing which satisfies that the shares of honest parties are uniformly random given the shares of corrupted parties. Thus, the shares of honest parties generated by $\mathcal{S}$ have the same distribution in both worlds. After generating the shares of $[\![\mathbf{x}+\mathbf{r}]\!]_d$, $\mathcal{S}$ honestly follows step 3 and 4. Therefore, $\mathcal{S}$ perfectly simulates the behaviors of honest parties.

Then, we show that the output of honest parties in both worlds have the same distribution. Note that the whole sharing $[\![\mathbf{x}]\!]_d$ is determined by the secrets $\mathbf{x}$ and the shares held by corrupted parties, which are identical in both worlds. Therefore, the output of honest parties is identical in both worlds.

□

PS-to-SS Conversion. We can achieve the PS-to-SS conversion by following similar steps as above. The protocol $\Pi_{\text{PS-to-SS}}$ and the corresponding functionality are detailed in Algorithm 24 and Figure 15 in Appendix A.

Theorem 3. *Protocol $\Pi_{PS\text{-}to\text{-}SS}$ securely computes $\mathcal{F}_{PS\text{-}to\text{-}SS}$ in the $\mathcal{F}_{MixRand}^{d,d}$-hybrid model against a semi-honest adversary who controls $t = d-k+1$ parties.*

Proof. This theorem can be proved similarly to Theorem 3. Therefore, for simplicity, we omit the details. □

4 Efficient Protocols for Packed Non-linear Functions

In this section, we design efficient protocols for non-linear layers based on PS sharing. This allows k non-linear operations to be executed with the same communication cost as a single operation.

Funcitonality $\mathcal{F}_{\text{DegRed}}^{d_1,d_2}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{DegRed}}^{d_1,d_2}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{DegRed}}^{d_1,d_2}$ receives the shares of $[\![\mathbf{a}]\!]_{d_2}$ from honest parties, where $d-k+1 < d_1 \leq d_2 \leq n-1$, and reconstructs the secrets $\mathbf{a}$.
3. $\mathcal{F}_{\text{DegRed}}^{d_1,d_2}$ receives from the adversary a set of shares $\{s_i\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{DegRed}}^{d_1,d_2}$ samples a random degree-d_1 packed Shamir sharing $[\![\mathbf{a}]\!]_{d_1}$ such that the share of each $P_i \in \mathcal{C}$ is s_i.
5. $\mathcal{F}_{\text{DegRed}}^{d_1,d_2}$ distributes the shares of $[\![\mathbf{a}]\!]_{d_1}$ to honest parties.

Figure 7: Ideal functionality for degree reduction.

Protocol $\Pi_{\text{DegRed}}^{d_1,d_2}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d_1 and d_2.

Inputs. All parties input $[\![\mathbf{a}]\!]_{d_2}$ where $d-k+1 \leq d_1 \leq d_2 \leq n-1$.

Protocol.

1. All parties invoke $\mathcal{F}_{\text{DouRand}}^{\mathsf{PS},d_1,n-1}$ to receive $([\![\mathbf{r}]\!]_{d_1}, [\![\mathbf{r}]\!]_{n-1})$.
2. All parties locally compute $[\![\mathbf{b}]\!]_{n-1} := [\![\mathbf{a}]\!]_{d_2} + [\![\mathbf{r}]\!]_{n-1}$ and send its share to P_{king}.
3. P_{king} reconstructs $\mathbf{b}$ and distributes the degree-d_1 PS sharing $[\![\mathbf{b}]\!]_{d_1}$ to other parties.
4. All parties locally compute $[\![\mathbf{a}]\!]_{d_1} := [\![\mathbf{b}]\!]_{d_1} - [\![\mathbf{r}]\!]_{d_1}$.

Figure 8: Protocol for general degree reduction.

General Degree Reduction. We first introduce a general-purpose degree reduction protocol that reduces the degree of a PS sharing from d_2 to d_1, where $1 \leq d_1 \leq d_2 \leq n-1$. We present the functionality in Figure 7 and the protocol $\Pi_{\text{DegRed}}^{d_1,d_2}$ in Figure 8. This building block will be frequently invoked by subsequent packed non-linear protocols, since a common operation in such protocols (i.e., the multiplication of a public vector with a PS sharing) increases the degree of the PS sharing

Our key improvement is to generate a pair of packed random $(d_1, n-1)$-degree sharings $([\![\mathbf{r}]\!]_{d_1}, [\![\mathbf{r}]\!]_{n-1})$, rather than (d_1,d_2)-degree sharings $([\![\mathbf{r}]\!]_{d_1}, [\![\mathbf{r}]\!]_{d_2})$. The reason is that with PRGs, generating a random degree-$(n-1)$ PS sharing requires only sending $\frac{k}{n-t}$ field elements, while a random degree-d_2 PS sharing requires sending $\frac{k+(n-d_2)}{n-t}$ field elements. This optimization does not affect security, since for any $d_2 \leq n-1$, the random PS sharing $[\![\mathbf{r}]\!]_{n-1}$ can perfectly mask a degree-d_2 PS sharing without revealing any information about the underlying secrets.

Packed Prefix OR. We present PS sharing-based protocols for Prefix OR, a core building block of ReLU. Given inputs $a_0, a_1, \ldots, a_{\ell-1}$, Prefix OR computes $b_i := \vee_{j=0}^{i} a_j$ for $i \in [0, \ell-1]$. Similar to previous works [27], we first evaluate its dual problem, i.e., packed Prefix Product $\bar{\mathbf{b}}_i := \wedge_{j=0}^{i} \bar{\mathbf{a}}_j$ for $i \in [0, \ell-1]$, and then obtain $\mathbf{b}_i := \mathbf{1} - \bar{\mathbf{b}}_i$. The naive approach to computing the packed Prefix Product requires a sequence of ℓ packed DN multiplications, resulting in $\ell-1$ rounds of communication which is unacceptable during online inference phase. We present a round-efficient Prefix OR protocol with packed inputs, while maintaining minimal communication cost. We observe that the prefix computation structure [25] allows the use of an additional ℓ packed DN multiplications to decouple the inherent sequential dependencies of the original procedure. This enables the parallel evaluation of multiple multiplication gates, thereby reducing the round complexity to $O(\log \ell)$. We present the packed Prefix Product protocol in Figure 9. It only requires linear communication cost, i.e., $2\ell - \log \ell - 2$ multiplications, with $2\log \ell$ rounds.

Building on $\Pi_{\text{PrefixMult}}$, we present the packed Prefix OR protocol, which computes $\mathbf{b_i} := \vee_{j=0}^{i} \mathbf{a}_j$ for $i \in [0, \ell-1]$ The complete description of Π_{PrefixOR} is presented in Figure 10. In the online phase, the communication complexity is $2n(2\ell - \lceil \log \ell \rceil - 2)$ field elements, and in the offline phase, it is $(2\ell - \lceil \log \ell \rceil - 2)(n+4k)n/2(n-t)$ field elements.

Protocol $\Pi_{\text{PrefixMult}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d.

Inputs. All parties input $[\![\mathbf{a}_0]\!]_d$, $[\![\mathbf{a}_1]\!]_d$, …, $[\![\mathbf{a}_{\ell-1}]\!]_d$, where $\mathbf{a}_i \in \{0,1\}^k$.

Protocol.

1. For $i = 1$ to $\lceil \log \ell \rceil$:
   (a) Set $j = \lceil \log \ell \rceil$
   (b) For $s = 1$ to 2^{j-1} (in parallel): All parties compute $m := 2^i s - 1$, input $([\![\mathbf{a}_{m-2^{i-1}}]\!]_d, [\![\mathbf{a}_m]\!]_d)$ to $\mathcal{F}_{\text{Mult}}$ and receive $[\![\mathbf{a}_m]\!]_d$.
   (c) Update $j = j - 1$
2. For $i = \lceil \log \ell \rceil - 1$ down to 1:
   (a) Set $j = 1$
   (b) For $s = 1$ to $2^j - 1$ (in parallel): All parties get $m := 2^i s - 1$, input $([\![\mathbf{a}_m]\!]_d, [\![\mathbf{a}_{m+2^{i-1}}]\!]_d)$ to $\mathcal{F}_{\text{Mult}}$ and receive $[\![\mathbf{a}_{m+2^{i-1}}]\!]_d$.
   (c) Update $j = j + 1$
3. For $i \in [0, \ell - 1]$, all parties compute $[\![\mathbf{b}_i]\!]_d := [\![\mathbf{a}_i]\!]_d$

Figure 9: Protocol for packed Prefix Product.

Protocol Π_{PrefixOR}

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d.

Inputs. All parties input $[\![\mathbf{a}_0]\!]_d$, $[\![\mathbf{a}_1]\!]_d$, …, $[\![\mathbf{a}_{\ell-1}]\!]_d$, where $\mathbf{a}_i \in \{0,1\}^k$.

Protocol.

1. For $i \in [0, \ell - 1]$, all parties locally compute $[\![\mathbf{b}_i]\!]_d := \mathbf{1} - [\![\mathbf{a}_i]\!]_d$.
2. All parites input $([\![\mathbf{b}_0]\!]_d, \ldots, [\![\mathbf{b}_{\ell-1}]\!]_d)$ to $\mathcal{F}_{\text{PrefixMult}}$ and receive $([\![\mathbf{c_0}]\!]_d, \ldots, [\![\mathbf{c}_{\ell-1}]\!]_d)$.
3. For $i \in [0, \ell - 1]$, all parties locally compute $[\![\mathbf{d}_i]\!]_d := \mathbf{1} - [\![\mathbf{c}_i]\!]_d$.

Figure 10: Protocol for packed Prefix OR.

Protocol $\Pi_{\text{Bitwise-LT}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Inputs. All parties input public $\mathbf{a}$ and private $([\![\mathbf{b}_0]\!]_d, \ldots, [\![\mathbf{b}_{\ell-1}]\!]_d)$. Let $\mathbf{a}_i, \mathbf{b}_i \in \{0,1\}^k$ for $i \in [0, \ell - 1]$ be the bit-decomposed vectors of $\mathbf{a}$ and $\mathbf{b}$, respectively: $\mathbf{a} := \sum_{i=0}^{\ell-1} \mathbf{2}^i \mathbf{a}_i$ and $\mathbf{b} := \sum_{i=0}^{\ell-1} \mathbf{2}^i \mathbf{b}_i$.

Protocol.

1. For $i \in [0, \ell - 1]$, all parties locally compute $[\![\mathbf{c}_i]\!]_{d+k-1} := \mathbf{a}_i \oplus [\![\mathbf{b}_i]\!]_d$.
2. For $i \in [0, \ell - 1]$, all parties input $[\![\mathbf{c}_i]\!]_{d+k-1}$ to $\mathcal{F}_{\text{DegRed}}^{d+k-1,d}$ and receive $[\![\mathbf{c}_i]\!]_d$.
3. All parties input $([\![\mathbf{c}_{\ell-1}]\!]_d, \ldots, [\![\mathbf{c_0}]\!]_d)$ to $\mathcal{F}_{\text{PrefixOR}}$ and receive $([\![\mathbf{d}_{\ell-1}]\!]_d, \ldots, [\![\mathbf{d_0}]\!]_d)$.
4. For $i \in [0, \ell - 1)$, all parties locally compute $[\![\mathbf{e}_i]\!]_d := [\![\mathbf{d}_i - \mathbf{d_{i+1}}]\!]_d$, where $[\![\mathbf{e}_{\ell-1}]\!]_d := [\![\mathbf{d}_{\ell-1}]\!]_d$, and then get $[\![\mathbf{a} < \mathbf{b}]\!]_{d+k-1} := \sum_{i=0}^{\ell-1} (\mathbf{1} - \mathbf{a}_i)[\![\mathbf{e}_i]\!]_d$.

Figure 11: Protocol for packed Bitwise Less Than.

Packed Bitwise Less Than. The original Bitwise-LT protocol [12] computes $[a < b]_d$ for a public $a \in \mathbb{F}_p$ and $([b_0]_d, \ldots, [b_{\ell-1}]_d)$ of bits $(b_0, \ldots, b_{\ell-1}) \in \{0,1\}^\ell$ with $b := \sum_{i=0}^{\ell-1} 2^i b_i \in \mathbb{F}_p$. Building on Π_{PrefixOR} , we can extends it to a packed variant, which

Protocol Π_{DReLU}

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d.

Inputs. All parties input $[\![\mathbf{t}]\!]_d$.

Protocol.

1. All parties invoke $\mathcal{F}_{\text{SolvedBits}}$ to receive $([\![\mathbf{r}]\!]_d, \{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{\ell-1})$, where $\mathbf{r}_i \in \{0,1\}^k$ and $\mathbf{r} \in \mathbb{F}_p^k$.
2. All parties locally compute $[\![\mathbf{y}]\!]_d := [\![2\mathbf{t}]\!]_d + [\![\mathbf{r}]\!]_d$ and send it to P_{king}.
3. P_{king} reconstructs the secret $\mathbf{y}$ and sends it back to other parties.
4. All parties locally compute $[\![\mathbf{b}]\!]_{d+k-1} := \mathbf{y}_{\mathbf{lsb}} \oplus [\![\mathbf{r}_{\mathbf{lsb}}]\!]_d$.
5. All parties input $(\mathbf{y}, \{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{\ell-1})$ to $\mathcal{F}_{\text{Bitwise-LT}}$ and receive $[\![\mathbf{c}]\!]_{d+k-1}$.
6. All parties input $[\![\mathbf{b}-\mathbf{c}]\!]_{d+k-1}$ to $\mathcal{F}_{\text{DegRed}}^{d+k-1,d}$ and receive $[\![\mathbf{b}-\mathbf{c}]\!]_d$.
7. All parties input $([\![\mathbf{b}-\mathbf{c}]\!]_d, [\![\mathbf{b}-\mathbf{c}]\!]_d)$ to $\mathcal{F}_{\text{MULT}}$ and recieve $[\![\mathbf{e}]\!]_d$, then compute $[\![\mathbf{t}']\!]_d := \mathbf{1} - [\![\mathbf{e}]\!]_d$.

Figure 12: Protocol for packed DReLU.

simultaneously computes k Bitwise-LT with a similar cost as a single execution. The complete protocol of $\Pi_{\text{Bitwise-LT}}$ is presented in Figure 11. Based on Π_{PrefixOR}, the protocol achieves online communication complexity of $2n(3\ell - \lceil \log \ell \rceil - 2)$ field elements, and offline communication complexity of $(3\ell - \lceil \log \ell \rceil - 2)(n+4k)n/2(n-t)$ field elements.

Packed DReLU. For the original DReLU protocol with input t, which computes the derivative of the ReLU function, $\mathsf{DReLU}(t) = 0$ if $t < 0$, and $\mathsf{DReLU}(t) = 1$ otherwise. In other words, $\mathsf{DReLU}(t) = 1 - t_{msb}$. Following the idea in [39], since $t \in \mathbb{F}_p$ where p is odd, $t_{msb} = (2t)_{lsb}$. We extend the method of computing $[(2t)_{lsb}]_d$ in [27] to a packed variant, and then simultaneously computes the DReLU of k values with communication cost comparable to a single execution. The complete protocol is presented in Figure 12. Based on $\Pi_{\text{SolvedBits}}$ and $\Pi_{\text{Bitwise-LT}}$, the protocol achieves online communication complexity of $n(6\ell - 2\lceil \log \ell \rceil + k)$ field elements, and offline communication complexity of $(\ell(13n+20k-8t) - \lceil \log \ell \rceil (n+4k))n/2(n-t)$ field elements.

Packed Truncation. For the fixed-point arithmetic used in PPML, multiplying two ℓ bit numbers with b fractional bits results in the output with $2b$ fractional bits. To prevent overflow, we require executing a truncation operation [27, 33]. As shown in LXY24 [27], the b-bit truncation of an input $x \in \mathbb{F}_p$ can be computed as follows:

$$\mathsf{Trunc}_b(x) = \begin{cases} \lfloor x/2^b \rfloor, & 0 \le x \le (p-1)/2, \\ p - \lfloor (p-x)/2^b \rfloor, & (p-1)/2 < x \le p-1. \end{cases}$$

Lemma 1 ([27]). *Let $p = 2^\ell - 1$, and let $0 \le x \le (p-1)/2$. Let $z = x + r \mod p$ and $y = p - r \mod p$, where $r \in \mathbb{F}_p$. Then, $\mathsf{Trunc}_b(y) + \mathsf{Trunc}_b(z) + \delta(r,z) \cdot (2^{b-1} - 1) \in \mathsf{Trunc}_b(x) + E$ with E be the set $\{-1, 0, 1\}$, where $\delta(r,z) = 1$ if $r_{msb} = 0 \wedge z_{msb} = 1$, and 0 otherwise.*

With the above lemma, we present the packed truncation protocol in Figure 28 which operates on packed inputs $\mathbf{x} \in [0, (p-1)/2]^k$. Let $\mathcal{F}_{\text{Trunc-Triple}}$ be the functionality (protocol details in Figure 27) generating packed random truncation triples $([\![\mathbf{r}']\!]_d, [\![\mathbf{r}]\!]_{n-1}, [\![\mathbf{r}_{msb}]\!]_d)$ where $\mathbf{r}' := \mathsf{Trunc}_b(\mathbf{r})$.

Our protocol performs truncation specifically after the ReLU activation operation detailed in Figure 14, which can be computed as $\mathsf{ReLU}(\mathbf{t}) = \mathsf{DReLU}(\mathbf{t}) \cdot \mathbf{t} \in [0, (p-1)/2]^k$. There are two main reasons for this design. First, our packed truncation protocol Π_{Trunc} requires the input to be non-negative integers and ReLU outputs, non-negative integers in the range $[0, (p-1)/2]^k$, meet this requirement. Second, this design eliminates the communication overhead of the multiplication step in the ReLU computation. Specifically, computing $[\![\mathsf{ReLU}(\mathbf{t})]\!]_d$ via packed DN multiplication on $[\![\mathsf{DReLU}(\mathbf{t})]\!]_d$ and $[\![\mathbf{t}]\!]_d$ typically requires the masking and revealing process with communication cost. We bypass this by directly inputing the locally computed product $[\![\mathsf{DReLU}(\mathbf{t}) \cdot \mathbf{t}]\!]_{2d}$ to $\mathcal{F}_{\text{Trunc}}$, obtaining $[\![\mathsf{Trunc}_b(\mathsf{ReLU}(\mathbf{t}))]\!]_{d+k-1}$. This approach is secure because the random mask $[\![\mathbf{r}]\!]_{n-1}$ from the truncation triple has degree $n-1 = 2d$, which is sufficient to securely mask the degree-$2d$ sharing $[\![\mathsf{DReLU}(\mathbf{t}) \cdot \mathbf{t}]\!]_{2d}$.

5 Cost-free Sharing Conversions in ML

As discussed in Section 3, we employ a mixed secret sharing strategy for privacy-preserving machine learning. Since machine learning models alternate between linear and non-linear layers, we develop new techniques that integrate these conversions

directly into the respective protocols, thereby eliminating any additional communication overhead.

5.1 Integrating Sharing Conversion into Linear Protocols

In this section, we design efficient protocols for matrix-vector multiplication, the fundamental operation underlying linear layers such as fully connected and convolutional layers [21, 22]. The functionality is shown in Figure 16 in Appendix A. We begin with a basic protocol that leverages SS sharing-based DN07 multiplication [13], followed by an SS-to-PS conversion to support subsequent PS sharing-based non-linear protocols. We then optimize this construction by integrating the SS-to-PS conversion directly into the matrix-vector multiplication, thereby eliminating the additional communication overhead of a separate conversion step.

Basic protocol for linear layers with sharing conversion. The basic protocol computes matrix-vector multiplication by first leveraging the extended SS sharing-based DN multiplication protocol [10, 20]. We then apply our SS-to-PS conversion protocol to transform the Shamir-shared output into PS sharing. We present the detailed protocol in Figure 25 in Appendix B. Let N be the output size of matrix-vector multiplication. In the online phase, the communication complexity is $2nN$ field elements, while the conversion steps cost $nN(1+\frac{1}{k})$ field elements. In the offline phase, the communication cost is Nn field elements and the communication cost of the conversion is $N(2k^2+2n(n-t)+\frac{n(4k+n)}{2})/(n-t)$ field elements.

Improved conversion-integrated protocol for linear layers. We present an improved linear protocol to eliminate the additional cost of SS-to-PS conversion in Figure 13. The main observation is that both the degree reduction of multiplication and the sharing conversion require opening masked secrets, which can be integrated to reduce the communication cost. Specifically, the above basic approach first generates k pairs of double randomness $([r'_i]_d, [r'_i]_{2d})$ for $i \in [1,k]$, and a pair of mixed randomness $(\{[r''_i]_d\}_{i=1}^k, [\![\mathbf{r}'']\!]_d)$ in the offline phase. Then, it opens the masked secrets in multiplication and conversion within two rounds of communication in the online phase. In contrast, we squeeze them into a single round through preparing mixed randomness $(\{[r_i]_{2d}\}_{i=1}^k, [\![\mathbf{r}]\!]_d)$. The improved protocol reduces online communication cost to $nN(1+\frac{1}{k})$ field elements which achieving a 64% reduction when $k=8$, and offline communication cost to $N(2k^2+n(n-t)+\frac{n(4k+n)}{2})/(n-t)$ field elements, yielding total saving of $3nN$ field elements compared to the basic approach.

Theorem 4. *Protocol Π_{MatMul} securely computes $\mathcal{F}_{MatMul}$ against a semi-honest adversary who controls $t = d-k+1$ parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties. The simulator works as follows:

1. In step 1, $\mathcal{S}$ simulates $\Pi_{\text{MixRand}}^{2d,d}$ and learns the shares corrupted parties.
2. In step 2, $\mathcal{S}$ computes the shares of $[t_i]_{2d}+[r_i]_{2d}$ of corrupted parties and set the shares of honest parties to be uniform values for $i \in [1,N]$.
3. In step 3, if P_{king} is honest, $\mathcal{S}$ honestly follows the protocol. If P_{king} is corrupted, $\mathcal{S}$ sends the shares of $[t_i]_{2d}+[r_i]_{2d}$ of honest parties to P_{king}. P_{king} also receives the shares from corrupted parties. Then P_{king} can reconstruct $c_i = t_i + r_i$ for $i \in [1,N]$ and generate the packed sharings $[\![\mathbf{c}_j]\!]_d$ for $j \in [1,h]$ where $\mathbf{c}_j = (c_{(j-1)k+1}, \ldots, c_{jk})$. After $\mathcal{S}$ receives the shares of $[\![\mathbf{c}_j]\!]_d$ of honest parties, it recover the whole sharing $[\![\mathbf{c}_j]\!]_d$ and learns the shares of corrupted parties.
4. In step 4, $\mathcal{S}$ follows the protocol and computes the shares of $[\![\mathbf{t}_j]\!]_d$ of corrupted parties, for $j \in [1,h]$.

Now, we show that $\mathcal{S}$ perfectly simulates the behaviors of honest parties. We note that the only place where honest parties need to send messages to corrupted parties is in step 3 of the main protocol.

Observe that $[r_i]_{2d}$ is a random degree-$2d$ SS sharing. Therefore, $[t_i]_{2d}+[r_i]_{2d}$ is a random degree-$2d$ SS sharing which satisfies that the shares of honest parties are uniformly random given the shares of corrupted parties. Thus, the shares of honest parties generated by $\mathcal{S}$ have the same distribution in both worlds. After generating the shares of $[\![\mathbf{c}_j]\!]_d$ for $j \in [1,h]$, $\mathcal{S}$ honestly follows step 3 and 4. Therefore, $\mathcal{S}$ perfectly simulates the behaviors of honest parties.

Then, we show that the output of honest parties in both worlds have the same distribution. Note that the whole sharing $[\![\mathbf{t}_j]\!]_d$ is determined by the secrets $\mathbf{t}_j$ and the shares held by corrupted parties, which are identical in both worlds, for $j \in [1,h]$. Therefore, the output of honest parties is identical in both worlds. □

5.2 Integrating Sharing Conversion into Non-Linear Protocols

In ML, ReLU layers are usually followed by linear layers, meaning that it is necessary to convert the output of the ReLU layer from PS sharing to SS sharing. Similar to the linear protocols, we integrate the sharing conversion into these protocols at no additional overhead, enabling an efficient interface with the SS sharing-based protocols for the following linear layers.

Protocol Π_{MatMul}

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d, and the matrix dimensions $N \times M$ with $h := \frac{N}{k}$.

Inputs. All parties input $[X]_d$ and $[\mathbf{y}]_d$, where $X \in \mathbb{F}_p^{N \times M}$, $\mathbf{y} \in \mathbb{F}_p^M$.

Protocol.

1. For $j \in [1,h]$, all parties invoke $\mathcal{F}_{\text{MixRand}}^{2d,d}$ to receive $(\{[r_{(j-1)k+a}]_{2d}\}_{a=1}^k, [\![\mathbf{r_j}]\!]_d)$.
2. All parties locally compute $[t_i]_{2d} := \sum_{j=1}^M [x_{ij}]_d \cdot [y_j]_d$ and $[c_i]_{2d} := [t_i]_{2d} + [r_i]_{2d}$, for $i \in [1,N]$.
3. P_{king} collects all shares and reconstructs the secret values c_i, for $i \in [1,N]$. For $j \in [1,h]$, P_{king} generates PS sharing $[\![\mathbf{c}_j]\!]_d$, where $\mathbf{c}_j = (c_{(j-1)k+1}, \ldots, c_{jk})$ and distributes the shares to other parties.
4. For $j \in [1,h]$, all parties locally compute $[\![\mathbf{t}_j]\!]_d := [\![\mathbf{c}_j]\!]_d - [\![\mathbf{r}_j]\!]_d$.

Figure 13: Improved protocol for matrix-vector multiplication with SS-to-PS conversion.

Basic protocol for ReLU with sharing conversion. Using $\mathcal{F}_{\text{PS-to-SS}}$ described in 3.2 after the ReLU layer, we can prepare SS sharing from the output of the ReLU layer for the subsequent linear layer. The complete description is presented in Algorithm 26 in Appendix B. For the conversion operation, there are additional $n(k+1)$ field elements online communication cost and $n(k+2)+2(d+k)(k+1)$ field elements offline communication cost.

Improved conversion-integrated protocol for ReLU. We demonstrate an improved ReLU protocol to eliminate the additional cost of PS-to-SS conversion. We observe that in Figure 26, there is a degree reduction protocol before the PS-to-SS conversion. Since both protocols require a pair of random sharings to open the masked secrets separately, we integrate the degree reduction protocol and PS-to-SS conversion into a single step that requires only one round of communication. Specifically, in the preprocessing phase, we generate mixed randomness in the form of $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_{n-1})$. In the online phase, parties utilize $[\![\mathbf{r}]\!]_{n-1}$ to mask $[\![\mathbf{t}']\!]_{d+k-1}$, where $\mathbf{t}' := \mathsf{Trunc}_b(\mathsf{ReLU}(\mathbf{t}))$. After P_{king} unpacks secrets $\{t_i' + r_i\}_{i=1}^k$ and generates the SS sharing $\{[t_i' + r_i]_d\}_{i=1}^k$, parties unmask them with $\{[r_i]_d\}_{i=1}^k$. The complete description is presented in Figure 14. The functionality is shown in Figure 15 in Appendix A. By combining the last two steps of Figure 26 into one step, we reduce the communication complexity by $2n$ field elements in the online phase and $4n+6k$ field elements in the offline phase. For Figure 14, the communication complexity is $n(6\ell - 2\lceil \log \ell \rceil + 2k)$ field elements in the online phase and $nk + (7\ell n(6n+4k) - 2n\lceil \log \ell \rceil (2k+n))/2(n-t)$ field elements in the offline phase.

Theorem 5. *Protocol Π_{ReLU} securely computes $\mathcal{F}_{ReLU}$ in the $(\mathcal{F}_{DReLU}, \mathcal{F}_{Fixed\text{-}Mult})$-hybrid model against a semi-honest adversary who controls $t = d - k + 1$ parties.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. Let $\mathcal{C}$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The simulator works as follows:

1. In step 1, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{\text{MixRand}}^{d,n-1}$ and learns the shares of $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_{n-1})$ of corrupted parties.
2. In step 2, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{\text{DReLU}}$ and learns the shares of corrupted parties.
3. In step 3, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{\text{Fixed-Mult}}$ and learns the shares of corrupted parties.
4. In step 4, $\mathcal{S}$ generates a random element for each honest party as its share of $[\![\mathbf{c}]\!]_{n-1}$. $[\![\mathbf{r}]\!]_{2d}$ is a random degree-$2d$ packed sharing. The shares of $[\![\mathbf{c}]\!]_{2d}$ in the real world are uniform, the distribution of the shares of $[\![\mathbf{c}]\!]_{2d}$ of honest parties generated by $\mathcal{S}$ is identical to that in the real world.
5. In step 5, $\mathcal{S}$ follows the protocol. In step 6, $\mathcal{S}$ can compute the shares of $([c_1]_d, \ldots, [c_k])$ of corrupted parties by the shares of $[\![\mathbf{c}]\!]_d$ of honest parties $\mathcal{S}$ learnt in step 4. Then $\mathcal{S}$ computes the shares of $([t_1']_d, \ldots, [t_k']_d)$ of corrupted parties and send them to $\mathcal{F}_{\text{ReLU}}$.

It is clear that $\mathcal{S}$ perfectly simulates the behavior of honest parties. As for the output, note that whole sharing $([t_1']_d, \ldots, [t_k']_d)$ are determined by th secrets $\mathbf{t}'$ and the shares held by corrupted parties, which is identical in both worlds. Therefore, the output of honest parties is identical in both worlds.

□

Protocol Π_{ReLU}

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Inputs. All parties input $[\![\mathbf{t}]\!]_d$.

Protocol.

1. All parties invoke $\mathcal{F}_{\text{MixRand}}^{d,n-1}$ and receive $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_{n-1})$.
2. All parties input $[\![\mathbf{t}]\!]_d$ to $\mathcal{F}_{\text{DReLU}}$ and receive $[\![\mathbf{b}]\!]_d$.
3. All parties locally compute $[\![\mathbf{a}]\!]_{2d} = [\![\mathbf{b}]\!]_d \cdot [\![\mathbf{t}]\!]_d$, input $[\![\mathbf{a}]\!]_{2d}$ to $\mathcal{F}_{\text{Trunc}}$ and receive $[\![\mathbf{t}']\!]_{d+k-1}$.
4. All parties locally compute $[\![\mathbf{c}]\!]_{n-1} := [\![\mathbf{t}']\!]_{d+k-1} + [\![\mathbf{r}]\!]_{n-1}$ and send it to P_{king}.
5. P_{king} reconstructs the secret values $\mathbf{c}$ where $\mathbf{c} = (c_1, \ldots, c_k)$, generates SS sharings $[c_1]_d, \ldots, [c_k]_d$ and distributes the shares to other parties.
6. For $i \in [1,k]$, all parties locally compute $[t_i']_d := [c_i]_d - [r_i]_d$.

Figure 14: Improved protocol for ReLU with PS-to-SS conversion.

Table 1: Comparison of per-party's computation (seconds) and communication (MB) costs between our linear protocols and LXY24 [27].

Operations	# Parties	Threshold	Protocol	Time LAN (s)			Time WAN (s)			Comm (MB)		
				Offline	Online	Total	Offline	Online	Total	Offline	Online	Total
MatMul	11	2	LXY24	0.011	0.005	0.016	0.157	0.014	0.172	0.019	0.024	0.042
			Ours	0.003	0.006	0.009	0.217	0.028	0.245	0.030	0.019	0.049
		3	LXY24	0.012	0.006	0.018	0.172	0.015	0.187	0.021	0.025	0.046
			Ours	0.003	0.003	0.006	0.185	0.028	0.213	0.031	0.021	0.052
	31	5	LXY24	0.002	0.012	0.014	0.410	0.021	0.431	0.020	0.025	0.044
			Ours	0.021	0.025	0.046	1.122	0.041	1.162	0.024	0.016	0.040
		8	LXY24	0.002	0.015	0.016	0.476	0.026	0.502	0.022	0.026	0.049
			Ours	0.014	0.038	0.052	0.876	0.041	0.917	0.021	0.017	0.038
	51	9	LXY24	0.020	0.012	0.031	0.758	0.037	0.796	0.020	0.025	0.045
			Ours	0.051	0.127	0.178	1.857	0.117	1.974	0.024	0.016	0.040
		13	LXY24	0.025	0.039	0.063	0.863	0.054	0.917	0.023	0.027	0.049
			Ours	0.034	0.132	0.167	1.497	0.113	1.610	0.019	0.016	0.036
	71	12	LXY24	0.049	0.014	0.063	1.068	0.046	1.114	0.020	0.025	0.046
			Ours	0.116	0.173	0.289	2.704	0.197	2.901	0.023	0.016	0.039
		18	LXY24	0.069	0.041	0.110	1.221	0.053	1.273	0.023	0.027	0.049
			Ours	0.065	0.173	0.238	2.270	0.189	2.459	0.019	0.016	0.035
	91	15	LXY24	0.112	0.096	0.209	1.347	0.089	1.436	0.020	0.025	0.046
			Ours	0.229	0.204	0.434	3.447	0.251	3.698	0.023	0.016	0.039
		23	LXY24	0.089	0.131	0.220	1.540	0.120	1.660	0.023	0.027	0.050
			Ours	0.123	0.209	0.332	2.856	0.245	3.101	0.018	0.016	0.034

6 Evaluation

6.1 Experimental Setup

We implement our protocols in C++ and build upon the open-source framework hmmpc-public [27]. All experiments are conducted on cloud server instances ecs.g9i.16xlarge equipped with 64 vCPUs and 256 GB of RAM. Similar to prior works [22, 27], we evaluate our protocols under two network settings: a LAN setting with 15 Gbps bandwidth and 0.3ms round-trip time and a WAN setting with 200 Mbps bandwidth and 10ms round-trip time.

The performance of our scheme is benchmarked against Shamir secret sharing-based LXY24 [27], which represents the state-of-the-art in honest-majority privacy-preserving machine learning. We evaluate both our scheme and LXY24 on two 3-layer neural network architectures: Network-A [30] and Network-B [36]. For both schemes, we adopt the same finite field $\mathbb{F}_p$ with the Mersenne prime $p := 2^{31} - 1$ and set the fractional precision to 12 bits. In addition, we set the packing parameter $k := \frac{n}{3}$ and

Table 2: Comparison of per-party's computation (seconds) and communication (MB) costs between our non-linear protocols and LXY24 [27].

Operations	# Parties	Threshold	Protocol	Time LAN (s)			Time WAN (s)			Comm (MB)		
				Offline	Online	Total	Offline	Online	Total	Offline	Online	Total
RELU	11	2	LXY24	0.747	0.115	0.862	13.145	5.281	18.425	35.408	5.756	41.164
			Ours	0.840	0.270	1.110	7.337	3.397	10.734	14.308	5.237	19.545
		3	LXY24	0.732	0.106	0.838	14.100	5.496	19.595	37.889	5.747	43.636
			Ours	0.981	0.300	1.281	9.113	4.580	13.693	18.058	7.809	25.867
	31	5	LXY24	2.197	0.254	2.451	32.713	15.577	48.291	37.306	6.130	43.436
			Ours	1.103	0.391	1.494	9.514	3.389	12.903	7.028	1.737	8.765
		8	LXY24	1.961	0.253	2.214	39.518	16.241	55.759	40.242	6.121	46.363
			Ours	1.221	0.434	1.655	10.115	4.296	14.411	8.124	2.441	10.566
	51	9	LXY24	4.020	0.473	4.493	52.437	25.850	78.286	37.706	6.211	43.917
			Ours	1.589	0.536	2.125	12.141	3.574	15.715	5.433	1.074	6.507
		13	LXY24	3.822	0.499	4.321	61.089	26.983	88.073	40.750	6.209	46.959
			Ours	1.679	0.571	2.250	12.861	4.450	17.331	6.076	1.481	7.557
	71	12	LXY24	6.834	0.733	7.567	75.028	36.373	111.402	38.321	6.245	44.566
			Ours	2.081	0.735	2.816	15.600	3.899	19.498	4.779	0.822	5.601
		18	LXY24	6.536	0.829	7.365	81.331	37.591	118.922	40.971	6.536	47.507
			Ours	2.144	0.796	2.940	15.640	4.630	20.270	5.187	1.079	6.266
	91	15	LXY24	11.252	0.980	12.232	99.635	46.638	146.274	38.323	6.265	44.588
			Ours	2.544	0.961	3.505	18.626	4.068	22.693	4.362	0.654	5.016
		23	LXY24	9.838	1.097	10.935	112.095	50.881	162.976	41.958	6.257	47.353
			Ours	2.609	0.979	3.588	19.216	4.869	24.085	4.691	0.858	5.549
Trunc	11	2	LXY24	0.175	0.004	0.179	2.818	0.058	2.875	8.838	0.171	9.009
			Ours	0.280	0.004	0.285	3.163	0.078	3.241	6.659	0.2	6.859
		3	LXY24	0.176	0.004	0.180	3.112	0.082	3.193	9.280	0.171	9.451
			Ours	0.325	0.006	0.331	3.638	0.087	3.725	8.055	0.257	8.312
	31	5	LXY24	0.341	0.008	0.349	7.473	0.062	7.535	9.349	0.182	9.531
			Ours	0.453	0.008	0.461	6.517	0.063	6.580	4.376	0.127	4.503
		8	LXY24	0.315	0.007	0.322	8.453	0.081	8.534	9.878	0.182	10.060
			Ours	0.462	0.007	0.469	6.817	0.082	6.899	4.761	0.143	4.904
	51	9	LXY24	0.562	0.019	0.581	11.008	0.078	11.086	9.458	0.184	9.642
			Ours	0.831	0.012	0.843	9.556	0.064	10.020	3.896	0.114	4.010
		13	LXY24	0.518	0.019	0.537	13.880	0.097	13.977	10.008	0.184	10.192
			Ours	0.748	0.010	0.758	9.932	0.081	10.013	4.104	0.123	4.227
	71	12	LXY24	0.881	0.018	0.899	14.400	0.091	14.491	9.585	0.186	9.770
			Ours	1.350	0.016	1.366	13.345	0.065	13.410	3.700	0.109	3.809
		18	LXY24	0.819	0.014	0.833	17.361	0.105	17.466	10.064	0.186	10.250
			Ours	1.172	0.012	1.184	13.279	0.082	13.361	3.823	0.115	3.938
	91	15	LXY24	1.572	0.023	1.595	26.318	0.106	26.423	9.594	0.186	9.780
			Ours	2.270	0.021	2.291	16.843	0.070	16.913	3.582	0.105	3.687
		23	LXY24	1.283	0.027	1.310	28.257	0.114	28.370	10.096	0.186	10.282
			Ours	1.768	0.028	1.795	16.690	0.080	16.770	3.670	0.11	3.780

$k := \frac{n}{4}$, which corresponds to the corruption threshold $t := \frac{n}{6}$ and $t := \frac{n}{4}$, respectively, where n is the number of parties.

6.2 Performance of Building Blocks

Linear Function. We conduct a micro-benchmark of the linear layer by measuring the performance of our matrix multiplication protocol, which embeds SS-to-PS conversion, and compare it to the SS sharing-based matrix multiplication protocol in LXY24. As detailed in Table 1, despite performing an additional share conversion, our approach achieves a substantial reduction of approximately 40% in online communication, while maintaining comparable offline communication performance. Besides, our method incurs a moderate increase in computation cost, with online and offline time being roughly 2-4× slower than the baseline. This slowdown is primarily attributable to the SS-to-PS conversion. However, given that linear layers typically constitute no more than 5% of the total inference time, this trade-off is acceptable for the overall end-to-end latency of secure inference.

Non-Linear Function. We also benchmark our non-linear protocols, with the results presented in Table 2. We compare our

Table 3: End-to-end computation (seconds) and communication (MB) comparison for secure neural network inference. The best-performing protocol for each setting is highlighted in blue.

Operations	# Parties	Threshold	Protocol	Time LAN (s)			Time WAN (s)			Comm (MB)		
				Offline	Online	Total	Offline	Online	Total	Offline	Online	Total
Network-A	11	2	LXY24	0.558	0.187	0.745	9.623	3.068	12.690	24.267	3.229	27.497
			Ours	0.669	0.249	0.918	6.984	4.638	11.621	11.516	3.024	14.540
		3	LXY24	0.567	0.197	0.764	10.276	3.203	13.479	25.867	3.225	29.092
			Ours	0.828	0.289	1.117	8.339	5.009	13.348	14.320	4.462	18.782
	31	5	LXY24	1.441	0.331	1.772	24.341	8.797	33.139	25.589	3.439	29.028
			Ours	1.007	0.335	1.342	11.095	4.720	15.815	6.290	1.071	7.361
		8	LXY24	1.386	0.341	1.727	30.325	9.176	39.501	27.486	3.434	30.921
			Ours	1.043	0.358	1.401	11.295	5.028	16.323	7.089	1.466	8.555
	51	9	LXY24	2.738	0.464	3.200	39.224	14.572	53.796	25.868	3.485	29.353
			Ours	1.864	0.502	2.363	15.887	5.020	20.906	5.163	0.702	5.865
		13	LXY24	2.697	0.458	3.155	45.279	15.188	60.467	27.836	3.480	31.316
			Ours	1.893	0.490	2.382	15.666	5.283	20.949	5.612	0.931	6.543
	71	12	LXY24	4.487	0.646	5.133	56.914	20.463	77.377	26.275	3.504	29.779
			Ours	2.794	0.667	3.461	20.289	5.330	25.619	4.706	0.562	5.268
		18	LXY24	4.350	0.669	5.019	66.260	21.211	87.472	27.988	3.499	31.488
			Ours	2.505	0.676	3.181	20.155	5.516	25.671	4.969	0.706	5.675
	91	15	LXY24	7.202	0.888	8.090	72.827	26.242	99.069	26.281	3.515	29.796
			Ours	3.701	0.906	4.606	25.443	5.496	30.939	4.417	0.469	4.886
		23	LXY24	7.387	0.878	8.265	81.905	27.229	109.135	28.074	3.510	31.584
			Ours	3.506	0.959	4.465	24.602	5.754	30.356	4.610	0.582	5.192
Network-B	11	2	LXY24	2.429	0.363	2.792	39.415	12.522	51.937	101.773	13.612	115.385
			Ours	2.647	0.626	3.273	24.134	8.602	32.736	48.559	12.747	61.306
		3	LXY24	2.451	0.377	2.827	41.884	13.063	54.947	108.492	13.592	122.084
			Ours	3.205	0.774	3.979	29.574	11.302	40.876	60.399	18.814	79.214
	31	5	LXY24	5.976	0.992	6.968	117.732	36.797	154.529	107.315	14.497	121.812
			Ours	3.810	0.972	4.782	35.599	8.889	44.488	26.522	4.513	31.035
		8	LXY24	6.021	1.020	7.041	126.213	38.348	164.561	115.282	14.476	129.758
			Ours	4.115	1.117	5.232	39.148	10.989	50.137	29.873	6.175	36.048
	51	9	LXY24	11.081	1.613	12.694	196.455	61.136	257.591	108.485	14.688	123.173
			Ours	5.692	1.355	7.046	49.013	9.607	58.620	21.746	2.953	24.699
		13	LXY24	10.895	1.663	12.558	211.813	63.721	275.534	116.747	14.667	131.414
			Ours	5.840	1.453	7.293	52.244	11.479	63.723	23.635	3.916	27.551
	71	12	LXY24	18.689	2.264	20.952	–	–	–	110.191	14.768	124.959
			Ours	7.757	1.968	9.725	63.010	10.670	73.681	19.773	2.362	22.135
		18	LXY24	17.766	2.353	20.119	–	–	–	117.386	14.750	132.136
			Ours	7.565	2.031	9.596	64.844	12.294	77.138	20.935	2.969	23.904
	91	15	LXY24	29.834	3.051	32.885	–	–	–	110.217	14.815	125.032
			Ours	10.552	2.366	12.918	76.656	11.353	88.009	18.541	1.967	20.508
		23	LXY24	26.225	3.109	29.334	–	–	–	117.745	14.796	132.541
			Ours	10.072	2.531	12.602	78.244	13.175	91.419	19.422	2.448	21.870

packed ReLU and truncation protocols embedded PS-to-SS conversion, with LXY24's pure ReLU and truncation implementation. The experimental results demonstrate that our protocols demonstrate significant performance improvements, particularly in large-scale deployments.

Specifically, our ReLU protocol exhibits pronounced scalability. For example, for more than 51 parties, our protocol reduces total communication by 80-90%. In the WAN environment, this translates to online and offline speed-ups of approximately 6-11.4× and 4.3-5.8×, respectively. Notably, with 91 parties and a threshold of 15, the online phase is 11.4× faster and the offline phase is 5.3× faster. Even in the high-bandwidth LAN setting, our protocol maintains 2.3-4.4× online advantages with comparable offline time when the number of parties is large. More importantly, our protocol addresses the privacy leakage issues identified in LXY24's RELU implementation, delivering both enhanced security and superior performance. For the truncation protocol, our approach reduces communication by 30-40% for $n \geq 51$ parties, despite the overhead of the PS-to-SS conversion. In the WAN setting, this leads to a 1.1-1.5× speedup for both the online and offline phases.

6.3 Performance of End-to-End Evaluation

We present the end-to-end evaluation results of our mixed SS-based scheme against LXY24 on two neural networks in Table 3. We observe that for a small number of parties, our approach exhibits comparable online and offline time compared to LXY24. With larger parties, our scheme achieves a 5.5-6× reduction in total communication cost compared to LXY24. In the WAN setting, our communication-optimized protocol delivers substantial computational improvement. For Network-A, as the number of parties increases, the online phase is accelerated by 3-4.7×, while the offline phase improves by 2.8-3.2×. For Network-B, which is more communication-intensive, our protocol accelerates the online phase by 4.1× at 31 parties and by 6.3× at 51 parties, with the offline phase improving by 3.3-4×. Even in the LAN environment, where communication is less of a bottleneck, our approach maintains competitive performance. For instance, with 71 parties on Network-B, our online phase is about 1.2× faster and the offline phase is about 2.8× faster than LXY24.

7 Conclusion

In this work, we present scalable and communication-efficient multi-party protocols for machine learning in the honest-majority setting. At the core of our approach is a mixed secret sharing strategy that employs PS sharing for non-linear layers with repeated and independent operations, while using SS sharing for linear layers such as matrix multiplications to avoid costly secret routing. To seamlessly support alternating linear and non-linear computations, we introduce generic conversions between SS and PS sharings and integrate them into the respective ML protocols, thereby eliminating the communication overhead typically incurred by sharing conversions. We further design efficient PS-sharing-based protocols for fundamental non-linear building blocks. Finally, we implement our framework for secure multi-party ML inference, and our extensive evaluation shows that it reduces communication by 3.6-6.1× and improves runtime by 1.5-4.3× in WAN settings compared to the state of the art. As future work, we plan to extend our semi-honest protocols to achieve security against malicious adversaries.

References

[1] Hendrik Ballhausen, Stefanie Corradini, Claus Belka, Dan Bogdanov, Luca Boldrini, Francesco Bono, Christian Goelz, Guillaume Landry, Giulia Panza, Katia Parodi, Riivo Talviste, Huong Elena Tran, Maria Antonietta Gambacorta, and Sebastian Marschner. Privacy-friendly evaluation of patient data with secure multiparty computation in a european pilot study. *npj Digit. Medicine*, 7(1), 2024.

[2] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[3] Gabrielle Beck, Aarushi Goel, Abhishek Jain, and Gabriel Kaptchuk. Order-c secure multiparty computation for highly repetitive circuits. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 663–693. Springer, 2021.

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[5] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: fast and robust framework for privacy-preserving machine learning. *Proc. Priv. Enhancing Technol.*, 2020(2):459–480, 2020.

[6] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[7] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 2010.

[8] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: Programmable and efficient secure two-party computation for machine learning. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 496–511. IEEE, 2019.

[9] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[10] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 34–64. Springer, 2018.

[11] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proc. Priv. Enhancing Technol.*, 2020(4):355–375, 2020.

[12] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.

[13] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[14] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Turbopack: Honest majority MPC with constant online communication. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 951–964. ACM, 2022.

[15] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 699–710. ACM, 1992.

[16] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[17] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: efficient and scalable MPC in the honest majority setting. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 244–274. Springer, 2021.

[18] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall's marriage theorem. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 275–304. Springer, 2021.

[19] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2022.

[20] Vipul Goyal and Yifan Song. Malicious security comes free in honest-majority MPC. *IACR Cryptol. ePrint Arch.*, page 134, 2020.

[21] Jiaxing He, Kang Yang, Guofeng Tang, Zhangjie Huang, Li Lin, Changzheng Wei, Ying Yan, and Wei Wang. Rhombus: Fast homomorphic matrix-vector multiplication for secure two-party inference. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 2490–2504. ACM, 2024.

[22] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 809–826. USENIX Association, 2022.

[23] Siam Umar Hussain, Mojan Javaheripi, Mohammad Samragh, and Farinaz Koushanfar. COINN: crypto/ml codesign for oblivious inference via neural networks. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 3266–3281. ACM, 2021.

[24] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively secure 4pc for secure training and inference. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022.

[25] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *J. ACM*, 27(4):831–838, 1980.

[26] Yehuda Lindell and Ariel Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 259–276. ACM, 2017.

[27] Fengrun Liu, Xiang Xie, and Yu Yu. Scalable multi-party computation protocols for machine learning in the honest-majority setting. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[28] David Melanson, Ricardo Maia, Hee-Seok Kim, Anderson C. A. Nascimento, and Martine De Cock. Secure multi-party computation for personalized human activity recognition. *Neural Process. Lett.*, 55(3):2127–2153, 2023.

[29] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

[30] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.

[31] Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority MPC by batchwise multiplication verification. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 321–339. Springer, 2018.

[32] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure RNN inference. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1003–1020. IEEE, 2021.

[33] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 325–342. ACM, 2020.

[34] Devin Reich, Ariel Todoki, Rafael Dowsley, Martine De Cock, and Anderson C. A. Nascimento. Privacy-preserving classification of personal text messages with secure multi-party computation. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3752–3764, 2019.

[35] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: xnor-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1501–1518. USENIX Association, 2019.

[36] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 707–721. ACM, 2018.

[37] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[38] Marie Beth van Egmond, Vincent Dunning, Stefan van den Berg, Thomas Rooijakkers, Alex Sangers, Ton Poppe, and Jan Veldsink. Privacy-preserving anti-money laundering using secure multi-party computation. In Jeremy Clark and Elaine Shi, editors, *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part II*, volume 14745 of *Lecture Notes in Computer Science*, pages 331–349. Springer, 2024.

[39] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proc. Priv. Enhancing Technol.*, 2019(3):26–49, 2019.

[40] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proc. Priv. Enhancing Technol.*, 2021(1):188–208, 2021.

[41] Yansong Zhang, Xiaojun Chen, Qinghui Zhang, Ye Dong, and Xudong Chen. Helix: Scalable multi-party machine learning inference against malicious adversaries. Cryptology ePrint Archive, Paper 2025/347, 2025.

Appendix A Functionality Description

Functionality $\mathcal{F}_{\text{PS-to-SS}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{PS-to-SS}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{PS-to-SS}}$ receives the shares of $[\![\mathbf{x}]\!]_d$ from honest parties and reconstructs the secrets $\mathbf{x} = (x_1, \ldots, x_k)$.
3. $\mathcal{F}_{\text{PS-to-SS}}$ receives from the adversary a set of shares $\{s_i^{(1)}, \ldots, s_i^{(k)}\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{PS-to-SS}}$ samples random degree-d SS sharings $[x_1]_d, \ldots, [x_k]_d$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[x_j]_d$ is $s_i^{(j)}$ for $j \in [1,k]$.
5. $\mathcal{F}_{\text{PS-to-SS}}$ distributes the shares of $([x_1]_d, \ldots, [x_k]_d)$ to honest parties.

Figure 15: Ideal functionality for PS-to-SS conversion

Appendix B Useful Packed Protocols

Functionality $\mathcal{F}_{\text{MatMul}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d and the matrix dimensions $N \times M$ with $h := \frac{N}{k}$.

Functionality.

1. $\mathcal{F}_{\text{MatMul}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{MatMul}}$ receives the shares of $[X]_d$ and $[\mathbf{y}]_d$ from honest parties, where $X \in \mathbb{F}_p^{N \times M}$, $\mathbf{y} \in \mathbb{F}_p^M$.
3. $\mathcal{F}_{\text{MatMul}}$ reconstructs the secrets X, $\mathbf{y}$ to compute $t_i := \sum_{j=1}^{M} x_{ij} \cdot y_j$ for $i \in [1, N]$.
4. $\mathcal{F}_{\text{MatMul}}$ receives from the adversary a set of shares $\{s_i^{(1)}, \ldots, s_i^{(h)}\}_{P_i \in \mathcal{C}}$.
5. $\mathcal{F}_{\text{MatMul}}$ samples random degree-d PS sharings $(\llbracket \mathbf{t}_1 \rrbracket_d, \ldots, \llbracket \mathbf{t}_h \rrbracket_d)$ such that for all $P_i \in \mathcal{C}$, the i-th share of $\llbracket \mathbf{t}_j \rrbracket_d$ is $s_i^{(j)}$ for $j \in [1, h]$, where $\mathbf{t_j} = (t_{(j-1)k+1}, \ldots, t_{jk})$.
6. $\mathcal{F}_{\text{MatMul}}$ distributes the shares of $\llbracket \mathbf{t}_j \rrbracket_d$ for $j \in [1, h]$ to honest parties.

Figure 16: Ideal functionality for matrix-vector multiplication.

Functionality $\mathcal{F}_{\text{ReLU}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{ReLU}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{ReLU}}$ receives the shares of $\llbracket \mathbf{t} \rrbracket_d$ from honest parties.
3. $\mathcal{F}_{\text{ReLU}}$ recovers the whole PS sharings $\llbracket \mathbf{t} \rrbracket_d$, reconstructs the secrets $\mathbf{t} = (t_1, \ldots, t_k)$ and computes $\mathbf{t}' := \mathsf{Trunc}_b(\mathsf{ReLU}(\mathbf{t}))$.
4. $\mathcal{F}_{\text{ReLU}}$ receives from the adversary a set of shares $\{s_i^{(1)}, \ldots, s_i^{(k)}\}_{P_i \in \mathcal{C}}$.
5. $\mathcal{F}_{\text{ReLU}}$ samples random degree-d SS sharings $[t'_1]_d, \ldots, [t'_k]_d$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[t'_j]_d$ is $s_i^{(j)}$, for $j \in [1, k]$
6. $\mathcal{F}_{\text{ReLU}}$ distributes the shares of $([t'_1]_d, \ldots, [t'_k]_d)$ to honest parties.

Figure 17: Ideal functionality for ReLU operation.

Funcitonality $\mathcal{F}_{\text{PrefixMult}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{PrefixMult}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{PrefixMult}}$ receives the shares of $\llbracket \mathbf{a}_0 \rrbracket_d$, $\llbracket \mathbf{a}_1 \rrbracket_d$, …, $\llbracket \mathbf{a}_{\ell-1} \rrbracket_d$, where $\mathbf{a}_i \in \{0,1\}^k$ from honest parties, reconstructs the secrets $\mathbf{a}_0$, $\mathbf{a}_1$, …, $\mathbf{a}_{\ell-1}$, and computes $\mathbf{b}_i := \wedge_{j=0}^{i} \mathbf{a}_j$ for $i \in [0, \ell-1]$.
3. $\mathcal{F}_{\text{PrefixMult}}$ receives from the adversary a set of shares $\{s_i^{(0)}, \ldots, s_i^{(\ell-1)}\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{PrefixMult}}$ samples random degree-d packed Shamir sharings $\llbracket \mathbf{b}_0 \rrbracket_d$, $\llbracket \mathbf{b}_1 \rrbracket_d$, …, $\llbracket \mathbf{b}_{\ell-1} \rrbracket_d$ such that for all $P_i \in \mathcal{C}$, the i-th share of $\llbracket \mathbf{b}_j \rrbracket_d$ is $s_i^{(j)}$ for $j \in [0, \ell-1]$.
5. $\mathcal{F}_{\text{PrefixMult}}$ distributes the shares of $\llbracket \mathbf{b}_0 \rrbracket_d$, $\llbracket \mathbf{b}_1 \rrbracket_d$, …, $\llbracket \mathbf{b}_{\ell-1} \rrbracket_d$ to honest parties.

Figure 18: Ideal functionality for prefix product.

Funcitonality $\mathcal{F}_{\text{PrefixOR}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{PrefixOR}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{PrefixOR}}$ receives the shares of $[\![\mathbf{a}_0]\!]_d$, $[\![\mathbf{a}_1]\!]_d$, $\ldots$, $[\![\mathbf{a}_{\ell-1}]\!]_d$, where $\mathbf{a}_i \in \{0,1\}^k$ from honest parties, reconstructs the secrets $\mathbf{a}_0$, $\mathbf{a}_1$, $\ldots$, $\mathbf{a}_{\ell-1}$, and computes $\mathbf{b}_i := \vee_{j=0}^{i} \mathbf{a}_j$ for $i \in [0, \ell-1]$.
3. $\mathcal{F}_{\text{PrefixOR}}$ receives from the adversary a set of shares $\{s_i^{(0)}, \ldots, s_i^{(\ell-1)}\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{PrefixOR}}$ samples random degree-d packed Shamir sharings $[\![\mathbf{b}_0]\!]_d$, $[\![\mathbf{b}_1]\!]_d$, $\ldots$, $[\![\mathbf{b}_{\ell-1}]\!]_d$ such that for all $P_i \in \mathcal{C}$, the i-th share of $[\![\mathbf{b}_j]\!]_d$ is $s_i^{(j)}$ for $j \in [0, \ell-1]$.
5. $\mathcal{F}_{\text{PrefixOR}}$ distributes the shares of $[\![\mathbf{b}_0]\!]_d$, $[\![\mathbf{b}_1]\!]_d$, $\ldots$, $[\![\mathbf{b}_{\ell-1}]\!]_d$ to honest parties.

Figure 19: Ideal functionality for prefix or.

Funcitonality $\mathcal{F}_{\text{Bitwise-LT}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{Bitwise-LT}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{Bitwise-LT}}$ receives the public vector $\mathbf{a}$ and shares of $[\![\mathbf{b}_0]\!]_d$, $[\![\mathbf{b}_1]\!]_d$, $\ldots$, $[\![\mathbf{b}_{\ell-1}]\!]_d$, reconstructs the secrets $\mathbf{b}_0$, $\mathbf{b}_1$, $\ldots$, $\mathbf{b}_{\ell-1}$.
3. $\mathcal{F}_{\text{Bitwise-LT}}$ receives from the adversary a set of shares $\{s_i\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{Bitwise-LT}}$ samples a random degree-d packed Shamir sharing $[\![\mathbf{a} < \mathbf{b}]\!]_d$ such that the share of each $P_i \in \mathcal{C}$ is s_i.
5. $\mathcal{F}_{\text{Bitwise-LT}}$ distributes the shares of $[\![\mathbf{a} < \mathbf{b}]\!]_d$ to honest parties.

Figure 20: Ideal functionality for bitwise-LT.

Funcitonality $\mathcal{F}_{\text{DReLU}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Functionality.

1. $\mathcal{F}_{\text{DReLU}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{DReLU}}$ receives the shares of $[\![\mathbf{t}]\!]_d$, reconstructs the secrets $\mathbf{t}$ and computes $\mathbf{t}' = \mathsf{DReLU}(\mathbf{t})$
3. $\mathcal{F}_{\text{DReLU}}$ receives from the adversary a set of shares $\{s_i\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{DReLU}}$ samples a random degree-d packed Shamir sharing $[\![\mathbf{t}']\!]_d$ such that the share of each $P_i \in \mathcal{C}$ is s_i.
5. $\mathcal{F}_{\text{DReLU}}$ distributes the shares of $[\![\mathbf{t}']\!]_d$ to honest parties.

Figure 21: Ideal functionality for DReLU.

Funcitonality $\mathcal{F}_{\text{Trunc}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d and the truncation bit parameter b.

Functionality.

1. $\mathcal{F}_{\text{Trunc}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{Trunc}}$ receives the shares of $[\![\mathbf{x}]\!]_d$, reconstructs the secrets $\mathbf{x}$ and computes $\mathsf{Trunc}_b(\mathbf{x})$
3. $\mathcal{F}_{\text{Trunc}}$ receives from the adversary a set of shares $\{s_i\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{Trunc}}$ samples a random degree-d packed Shamir sharing $[\![\mathsf{Trunc}_b(\mathbf{x})]\!]_d$ such that the share of each $P_i \in \mathcal{C}$ is s_i.
5. $\mathcal{F}_{\text{Trunc}}$ distributes the shares of $[\![\mathsf{Trunc}_b(\mathbf{x})]\!]_d$ to honest parties.

Figure 22: Ideal functionality for Truncation.

Funcitonality $\mathcal{F}_{\text{Trunc-Triple}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d and the truncation bit parameter b.

Functionality.

1. $\mathcal{F}_{\text{Trunc-Triple}}$ receives the set of corrupted parties denoted by $\mathcal{C}$.
2. $\mathcal{F}_{\text{Trunc-Triple}}$ receives the shares of $\{[\![\mathbf{r}_i]\!]_d\}_{i=0}^{\ell-1}$, reconstructs the secrets $\{\mathbf{r}_i\}_{i=0}^{\ell-1}$ and computes $\mathbf{r} = \sum_{i=0}^{\ell-1} 2^i \cdot \mathbf{r}_i$ and $\mathbf{r}' := \mathsf{Trunc}_b(\mathbf{r})$
3. $\mathcal{F}_{\text{Trunc-Triple}}$ receives from the adversary a set of shares $\{s_i^{(0)}, s_i^{(1)}, s_i^{(2)}\}_{P_i \in \mathcal{C}}$.
4. $\mathcal{F}_{\text{Trunc-Triple}}$ samples a random degree-d packed Shamir sharing $[\![\mathbf{r}']\!]_d$ such that the share of each $P_i \in \mathcal{C}$ is $s_i^{(0)}$, a random degree-$n-1$ packed Shamir sharing $[\![\mathbf{r}]\!]_{n-1}$ such that the share of each $P_i \in \mathcal{C}$ is $s_i^{(1)}$ and a random degree-d packed Shamir sharing $[\![\mathbf{r}_{msb}]\!]_d$ such that the share of each $P_i \in \mathcal{C}$ is $s_i^{(3)}$.
5. $\mathcal{F}_{\text{Trunc-Triple}}$ distributes the shares of $[\![\mathbf{r}']\!]_d$, $[\![\mathbf{r}]\!]_{n-1}$, $[\![\mathbf{r}_{msb}]\!]_d$to honest parties.

Figure 23: Ideal functionality for Trunc-Triple.

Protocol $\Pi_{\text{PS-to-SS}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Inputs. All parties input $[\![\mathbf{x}]\!]_d$ with $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{F}_p^k$.

Protocol.

1. All parties invoke $\mathcal{F}_{\text{MixRand}}^{d,d}$ to receive $(\{[r_i]_d\}_{i=1}^k, [\![\mathbf{r}]\!]_d)$.
2. All parties locally compute $[\![\mathbf{x}+\mathbf{r}]\!]_d := [\![\mathbf{x}]\!]_d + [\![\mathbf{r}]\!]_d$ and send their shares to P_{king}.
3. P_{king} reconstructs $\mathbf{x}+\mathbf{r}$. For $i \in [1,k]$, P_{king} generates SS sharing $[x_i + r_i]_d$ and distributes the shares to other parties.
4. For $i \in [1,k]$, all parties locally compute $[x_i]_d := [x_i + r_i]_d - [r_i]_d$.

Figure 24: Protocol for PS-to-SS conversion.

Protocol $\Pi_{\text{BasicMatMul}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d, and the matrix dimensions $N \times M$ with $h := \frac{N}{k}$.

Inputs. All parties input $[X]_d$ and $[\mathbf{y}]_d$, where $X \in \mathbb{F}_p^{N \times M}$, $\mathbf{y} \in \mathbb{F}_p^M$.

Protocol.

1. For $i \in [1,N]$, all parties invoke $\mathcal{F}_{\text{DouRand}}^{\mathsf{SS},d,2d}$ to receive $([r_i]_d, [r_i]_{2d})$.
2. All parties locally compute $[t_i]_{2d} := \sum_{j=1}^M [x_{ij}]_d \cdot [y_j]_d$ and $[c_i]_{2d} := [t_i]_{2d} + [r_i]_{2d}$, for $i \in [1,N]$.
3. P_{king} collects all shares and reconstructs the secret values c_i, for $i \in [1,N]$, then generates SS sharing $[c_i]_d$ and distributes the shares to other parties.
4. All parties locally compute $[t_i]_d := [c_i]_d - [r_i]_d$ for $i \in [1,N]$.
5. For $j \in [1,h]$, all parties input $([t_{(j-1)k+1}]_d, \ldots, [t_{jk}]_d)$ to $\mathcal{F}_{\text{SS-to-PS}}$ and receive $[\![\mathbf{t}_j]\!]_d$.

Figure 25: Basic protocol for matrix-vector multiplication with SS-to-PS conversion

Protocol $\Pi_{\text{BasicReLU}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, and the degree d.

Inputs. All parties input $[\![\mathbf{t}]\!]_d$.

Protocol.

1. All parties input $[\![\mathbf{t}]\!]_d$ to $\mathcal{F}_{\text{DReLU}}$ and receive $[\![\mathbf{b}]\!]_d$.
2. All parties input $([\![\mathbf{b}]\!]_d, [\![\mathbf{t}]\!]_d)$ to $\mathcal{F}_{\text{Fixed-Mult}}$ and receive $[\![\mathbf{t}']\!]_{d+k-1}$.
3. All parties locally compute $[\![\mathbf{a}]\!]_{2d} = [\![\mathbf{b}]\!]_d \cdot [\![\mathbf{t}]\!]_d$, input $[\![\mathbf{a}]\!]_{2d}$ to $\mathcal{F}_{\text{Trunc}}$ and receive $[\![\mathbf{t}']\!]_{d+k-1}$.
4. All parties input $[\![\mathbf{t}']\!]_d$ to $\mathcal{F}_{\text{PS-to-SS}}$ and receive $([t'_1]_d, \ldots, [t'_k]_d)$.

Figure 26: Protocol for ReLU layer with PS-to-SS conversion.

Protocol $\Pi_{\text{Trunc-Triple}}$

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d, and the Mersenne prime $p = 2^\ell - 1$ with bit length ℓ.

Protocol.

1. All parties invoke $\mathcal{F}_{\text{SolvedBits}}$ to receive $(\llbracket \mathbf{r} \rrbracket_d, \{\llbracket \mathbf{r}_i \rrbracket_d\}_{i=0}^{\ell-1})$, where $\mathbf{r}_i \in \{0,1\}^k$ and $\mathbf{r} \in \mathbb{F}_p^k$.
2. All parties invoke $\mathcal{F}_{\text{Zero}}^{\mathsf{PS},n-1}$ to receive $\llbracket \mathbf{0} \rrbracket_{n-1}$.
3. All parties locally set $\llbracket \mathbf{r}_{msb} \rrbracket_d := \llbracket \mathbf{r}_{\ell-1} \rrbracket_d$.
4. All parties locally compute $\llbracket \mathbf{r}' \rrbracket_d := \sum_{i=d}^{\ell-1} 2^{i-d} \cdot \llbracket \mathbf{r}_i \rrbracket_d + \sum_{i=\ell-d}^{\ell-1} 2^i \cdot \llbracket \mathbf{r}_{msb} \rrbracket_d$.
5. All parties locally compute $\llbracket \mathbf{r} \rrbracket_{n-1} := \llbracket \mathbf{r} \rrbracket_d + \llbracket \mathbf{0} \rrbracket_{n-1}$.

Figure 27: Protocol for Trunc-Triple

Protocol Π_{Trunc}

Parameters. A finite field $\mathbb{F}_p$, the number k of packed secrets, the degree d and the truncation bit parameter b.

Inputs. All parties input $\llbracket \mathbf{x} \rrbracket_d$.

Protocol.

1. All parties invoke $\mathcal{F}_{\text{Trunc-Triple}}$ to receive $(\llbracket \mathbf{r}' \rrbracket_d, \llbracket \mathbf{r} \rrbracket_{n-1}, \llbracket \mathbf{r}_{msb} \rrbracket_d)$, where $\mathbf{r}' := \mathsf{Trunc}_b(\mathbf{r})$.
2. All parties locally compute $\llbracket \mathbf{c} \rrbracket_{n-1} := \llbracket \mathbf{x} \rrbracket_d + \llbracket \mathbf{r} \rrbracket_{n-1}$ and send it to P_{king}.
3. P_{king} reconstructs $\mathbf{c}$ and generate degree-$(k-1)$ PS sharing $\llbracket \mathbf{c}_{msb} \rrbracket_{k-1}$, $\llbracket \mathsf{Trunc}_b(\mathbf{c}) \rrbracket_{k-1}$. Then P_{king} distributes them to other parties.
4. All parties locally compute $\llbracket \mathbf{e} \rrbracket_{d+k-1} := (\mathbf{1} - \llbracket \mathbf{r}_{msb} \rrbracket_d) \llbracket \mathbf{c}_{msb} \rrbracket_{k-1}$ and $\llbracket \mathsf{Trunc}_b(\mathbf{x}) \rrbracket_{d+k-1} := \llbracket \mathsf{Trunc}_b(\mathbf{c}) \rrbracket_{k-1} - \llbracket \mathbf{r}' \rrbracket_d + \llbracket \mathbf{e} \rrbracket_{d+k-1}(\mathbf{2}^{\ell-b} - \mathbf{1})$.

Figure 28: Protocol for packed truncation