Scalable Privacy-Preserving Neural Network Training over $\mathbb{Z}_{2^k}$ via RMFE-Based Packing and Mixed-Circuit Computation

Hengcheng Zhou

School of Computer Science, Shanghai Jiao Tong University
zhc12345@sjtu.edu.cn

Abstract

We introduce a novel framework for privacy-preserving multi-party neural network training over $\mathbb{Z}_{2^k}$ with semi-honest security in the honest-majority setting. Our work utilizes Shamir secret sharing scheme over Galois rings $GR(2^k, d)$ and is scalable in the number of participants. Our primary contribution is a generalization of existing data packing techniques used in private training through Reverse Multiplication-Friendly Embedding (RMFE), which enables a higher packing density and thus more efficient SIMD-style parallel computation. Notably, our work is the first to support a general form of RMFE, lifting a common restriction from previous approaches. To holistically optimize the training process, we further integrate mixed-circuit techniques to be fully compatible with our RMFE-based packing scheme. This enables our protocol to efficiently compute nonlinear functions, such as comparison, by leveraging bit-wise computations over $GR(2, d)$. We consolidate these advances into an end-to-end parallel training framework. Experimental results on both fully connected and convolutional neural networks validate the practical performance advantages of our framework compared to existing methods.

Introduction

Privacy-Preserving Machine Learning (PPML) has emerged as a critical field at the intersection of machine learning and cryptography, aiming to reconcile privacy concerns involved in data-driven models. Among the various tasks in PPML, achieving privacy-preserving training for deep neural networks is particularly significant yet challenging due to the complexity of the training algorithms and the scale of the data involved (Zhu et al. 2016). One of the most promising cryptographic primitives for this task is secret-sharing-based Secure Multi-Party Computation (MPC) (Yao 1982, 1986), which allows a group of mutually untrusting parties to collaboratively train a model on their combined data without revealing their private inputs.

There has been a great deal of research devoted to realizing practical MPC-based training protocols. Many of these works (Agrawal et al. 2019; Mohassel and Zhang 2017; Mohassel and Rindal 2018; Wagh, Gupta, and Chandran 2019; Wagh et al. 2021; Patra and Suresh 2020; Chaudhari, Rachuri, and Suresh 2020; Koti et al. 2022; Keller and Sun 2022) have concentrated on providing customized solutions for a fixed number of parties (typically 2-4). Scaling these protocols to arbitrary n parties presents a significant challenge. For example, protocols based on replicated secret sharing (Ito, Saito, and Nishizeki 1989; Wagh et al. 2021; Baccarini, Blanton, and Yuan 2023) are constrained by communication costs that grow exponentially, rendering them impractical for large n. Alternatively, general-purpose MPC frameworks (Damgård et al. 2012; Cramer et al. 2018; Damgård and Nielsen 2007) are theoretically applicable to an arbitrary number of parties but are hindered by the prohibitive cost of securely evaluating the nonlinear functions, *e.g.*, activation, required for machine learning, making them unsuitable for PPML. Zhou (Zhou 2023) introduced a training protocol for general n parties based on classic Shamir secret sharing scheme (Shamir 1979), achieving a more scalable communication complexity of $\mathcal{O}(n)$. However, this framework, along with its subsequent optimizations (Zhou 2025b) using the packed Shamir scheme (Franklin and Yung 1992), was inherently limited by the computational cost of modulo-p arithmetic, a necessary operation in protocols over $\mathbb{F}_p$ that is not efficiently supported by modern computer hardware.

To address the bottleneck of $\mathbb{F}_p$-based protocols, a promising direction is to leverage Shamir scheme over Galois rings (Abspoel et al. 2019), which enables the replacement of modulo-p arithmetic with fast modulo-2^k operation. The work in (Zhou 2025a) first realized this potential for private training, and used a similar approach as Overdrive2k (Orsini, Smart, and Vercauteren 2020) to enable parallel computation by packing the secrets into the coefficients of Galois ring elements. While this method provided substantial speedups compared to its $\mathbb{F}_p$-based counterparts, it introduced its own scalability challenge: the ring's degree d grows exponentially with the number of packed secrets, ρ.

In this paper, we introduce a novel secure training framework utilizing Shamir scheme over Galois rings. We generalize the packing method of (Zhou 2025a) by leveraging Reverse Multiplication-Friendly Embedding (RMFE) (Cramer, Rambaud, and Xing 2021), resulting in a higher data packing density and more efficient parallel computation. RMFE uses a map ϕ to map multiple elements into an extended domain for parallel processing and can maintain a linear dependency between ρ and d. Coral (Huang et al. 2024) first

proposed an RMFE-based computation framework but was restricted to the binary field $\mathbb{F}_2$ and did not support the complex operations required for a complete training process. The RMFE used in Coral imposes a constraint that the map ϕ has to preserve the multiplicative identity. Our construction lifts this restriction to support a broader class of RMFEs. Furthermore, we integrate mixed-circuit techniques (Rotaru and Wood 2019; Escudero et al. 2020) into our RMFE-based framework, enabling more efficient computation of nonlinear functions such as truncation and comparison within a bit-oriented representation using $GR(2, d)$. We then demonstrate how to construct key training components on this basis, including activation, pooling, and loss function.

Our main contributions are as follows:

- We propose an RMFE-based framework for privacy-preserving training over Galois rings, achieving a higher data packing density for more efficient parallel computation. Our construction supports a general form of RMFE.
- We adapt the mixed-circuit computation to be compatible with our RMFE-based framework, enabling efficient computation for nonlinear operations.
- We evaluate the performance of our framework on both fully connected and convolutional neural networks. The results show significant reductions in training time and communication compared to existing approaches.

Preliminaries

Shamir Scheme over Galois Rings

Galois Rings. The formal definition of the Galois ring $GR(2^k, d)$ used in this work is given below. More details about Galois rings can be found in (Wan 2003).

Definition 1 (Galois Ring) *A Galois ring $GR(2^k, d)$ is a ring of the form $\mathbb{Z}_{2^k}[x]/(h(x))$, where k is a positive integer, and $h(x) \in \mathbb{Z}_{2^k}[x]$ is a non-constant, monic polynomial such that its reduction modulo 2 is irreducible in $\mathbb{F}_2[x]$.*

Shamir Secret Sharing Scheme. To construct a Shamir scheme over $R = GR(2^k, d)$, we require a set of distinct evaluation points $\{\beta_1, \ldots, \beta_n\} \subset R$ that form an *exceptional sequence* (Abspoel et al. 2019), meaning the difference $\beta_i - \beta_j$ is invertible for any $i \neq j$. In $GR(2^k, d)$, there exists an exceptional sequence of length up to 2^d, which allows us to establish a Shamir scheme among $n < 2^d$ parties.

To share a secret $s \in R$ among n parties against t adversaries, a dealer defines a random polynomial $f(x) \in R[x]$ of degree t such that $f(0) = s$. The dealer then distributes the share $s_i = f(\beta_i)$ to the i-th party. The vector $(s_1, \ldots, s_n)$ is called a t-sharing of s, which is denoted by $[\![s]\!]_t$, and simply $[\![s]\!]$ when t is clear from the context. Any group of $t + 1$ parties can jointly reconstruct s via Lagrange interpolation, whereas any group of t or fewer parties learns nothing about s. For mixed-circuit computation, we need two kinds of sharings, differentiated by superscripts when necessary:

- $[\![\cdot]\!]^{\mathcal{A}}$: An *arithmetic sharing* over $GR(2^k, d)$ for $k > 1$.
- $[\![\cdot]\!]^{\mathcal{B}}$: A *binary sharing* over $GR(2, d)$.

We use Π_{Reveal} and $\Pi^{\mathcal{B}}_{\text{Reveal}}$ to represent the secret reconstruction protocol on sharing $[\![\cdot]\!]^{\mathcal{A}}$ and $[\![\cdot]\!]^{\mathcal{B}}$, respectively.

Reverse Multiplication-Friendly Embedding

In this paper, we employ RMFE over $\mathbb{Z}_{2^k}$ (Cramer, Rambaud, and Xing 2021) for efficient parallel computation.

Definition 2 ((ρ, d)-RMFE) *Let $\rho, d \in \mathbb{N}$. A pair of $\mathbb{Z}_{2^k}$-module homomorphisms (ϕ, ψ) with $\phi : \mathbb{Z}^{\rho}_{2^k} \to GR\left(2^k, d\right)$ and $\psi : GR\left(2^k, d\right) \to \mathbb{Z}^{\rho}_{2^k}$ is a reverse multiplication-friendly embedding, if, for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}^{\rho}_{2^k}$, it holds that $\boldsymbol{x} \odot \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}))$, where $\odot$ denotes the Hadamard product and $\cdot$ denotes multiplication in $GR(2^k, d)$.*

Intuitively, an RMFE provides a mechanism to embed a vector of ρ secrets into a single element in $GR(2^k, d)$ via ϕ. The crucial property is that the multiplication of two such embedded elements, when mapped back via ψ, corresponds exactly to the component-wise product of the original vectors. This property allows us to perform Single Instruction, Multiple Data (SIMD) style operations on the packed elements. Note that, if (ϕ, ψ) is an RMFE, then necessarily ϕ is injective and ψ is surjective, so in particular $\rho \leqslant d$.

RMFE Construction. There exists a constructive family of (ρ, d)-RMFE over $\mathbb{Z}_{2^k}$ for all $k \geqslant 1$ with $\rho \to \infty$ and $d/\rho \to 4.92$ (Corollary 3, (Escudero et al. 2023)). More details about the construction of RMFEs can be found in (Cramer, Rambaud, and Xing 2021). For practical applications, $\rho = 3$ is a good choice, in which case we can obtain a $(3, 5)$-RMFE. We give its detailed construction below.

Example 1 (Construction of a $(3, 5)$-RMFE) *Define the map $\phi : \mathbb{Z}^3_{2^k} \to GR(2^k, 5)$, $(a_0, a_1, a_2) \mapsto a_0 + (a_1 - a_0 - a_2)x + a_2x^2 + 0x^3 + 0x^4$ and $\psi : GR(2^k, 5) \to \mathbb{Z}^3_{2^k}$, $g(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + b_4x^4 \mapsto (g(0), g(1), b_4)$. Then it is easy to verify that (ϕ, ψ) forms a $(3, 5)$-RMFE.*

Recap of Coefficient-Packing. An element in $GR(2^k, d)$ can be uniquely represented as a polynomial of degree less than d with coefficients in $\mathbb{Z}_{2^k}$. The work in (Zhou 2025a) uses multiple coefficients to store secrets, thereby enabling parallel computation. Their construction actually conforms to the definition of RMFE. To see this, we give the RMFE corresponding to their packing method of two secrets.

Example 2 ($(2, 3)$-RMFE in (Zhou 2025a)) *Define the map $\phi : \mathbb{Z}^2_{2^k} \to GR(2^k, 3)$, $(a_0, a_1) \mapsto a_0 + a_1x + 0x^2$ and $\psi : GR(2^k, 3) \to \mathbb{Z}^2_{2^k}$, $b_0 + b_1x + b_2x^2 \mapsto (b_0, b_2)$. Then (ϕ, ψ) forms a $(2, 3)$-RMFE.*

Our work generalizes the packing method in (Zhou 2025a), which is a specific instance of (ρ, d)-RMFE constrained by $d \geqslant 2^{\rho} - 1$, an exponential dependency that is impractical for large ρ. For instance, they require $d = 7$ for $\rho = 3$. Our framework, however, leverages more general and efficient RMFE constructions like Example 1, needing only $d = 5$ for the same ρ and thus offering superior performance.

Notation

In the rest of this paper, we let $R = GR(2^k, d)$ and use ρ to represent the number of packed secrets.We write $u \in_{\$} \mathcal{S}$ to denote the uniform random sampling of u from a set $\mathcal{S}$. We define $\boldsymbol{e} = (1, \ldots, 1) \in \mathbb{Z}^{\rho}_{2^k}$ and $\boldsymbol{e_0} = (1, 0, \ldots, 0) \in \mathbb{Z}^{\rho}_{2^k}$. Further notation will be introduced as needed.

Framework Overview

Threat Model

We consider a setting with $n \geqslant 3$ parties who wish to jointly train a neural network. Our protocols operate in the honest-majority setting, where at most $t < n/2$ parties can be passively corrupted by a semi-honest adversary. Throughout the computation, all sensitive data are protected via secret sharing. The underlying Shamir scheme guarantees that any coalition of t or fewer corrupted parties learns nothing beyond what is revealed by the protocol's designated outputs. The security of our protocols is analyzed under the Universal Composability framework (Canetti 2001).

Data Representation and Packing

We use a standard fixed-point representation (Catrina and de Hoogh 2010) to convert real numbers into $\mathbb{Z}_{2^k}$ before packing them into Galois rings. A real number $\tilde{x} \in \mathbb{R}$ is transformed to a fixed-point approximation $\bar{x}$ by setting $\bar{x} = \lfloor \tilde{x} \cdot 2^f \rfloor \in \mathbb{Z}$, where f is the precision and $\lfloor \cdot \rfloor$ denotes the floor function. Then, we encode $\bar{x}$ into $\mathbb{Z}_{2^k}$ by computing $x = \bar{x} \pmod{2^k}$. In this paper, we limit $\bar{x}$ in $[-2^{\ell-1}, 2^{\ell-1} - 1]$ and require $k \geqslant \ell + \kappa + 1$, where κ is the statistical security parameter.

RMFE Operations. The key for achieving parallelism in our framework is *packing*, where multiple secrets are embedded into a single Galois ring element. Let (ϕ, ψ) be an RMFE. We define the following fundamental operations:

- $\mathsf{Pack}(\cdot)$: This operation, defined by the map ϕ, embeds a vector of ρ secrets from $\mathbb{Z}_{2^k}^{\rho}$ into a single element in R.
- $\mathsf{PackInv}(\cdot)$: This operation, defined by ϕ^{-1}, retrieves the vector of secrets from $\mathrm{Im}(\phi)$ *before* multiplication.
- $\mathsf{Unpack}(\cdot)$: This operation, defined by the map ψ, retrieves the vector of secrets from R *after* multiplication.
- $\mathsf{ReEncode}(\cdot)$: This operation, defined as the composition $\phi \circ \psi$, maps a Galois ring element back into the valid image of ϕ. This is crucial for ensuring that the output of one multiplication can be correctly used as the input for the next, since ϕ is not necessarily surjective.

For Pack and Unpack, we use $\mathsf{Pack}^{\mathcal{B}}$ and $\mathsf{Unpack}^{\mathcal{B}}$ to represent the situation when $k = 1$, respectively.

RMFE-Based Computation Architecture

The overall architecture of our training framework is depicted in Figure 1. The process begins with data owners secret-sharing their private data after fixed-point encoding and RMFE-based packing. The target parameters are then trained interactively by all parties. It is worth emphasizing that training is not a fully parallel process. In subsequent sections, we will introduce our parallel computation framework and demonstrate its integration into the training procedure.

Building Blocks

Randomness

Throughout this paper, we need the following functionalities to generate the random sharings necessary for our protocols.

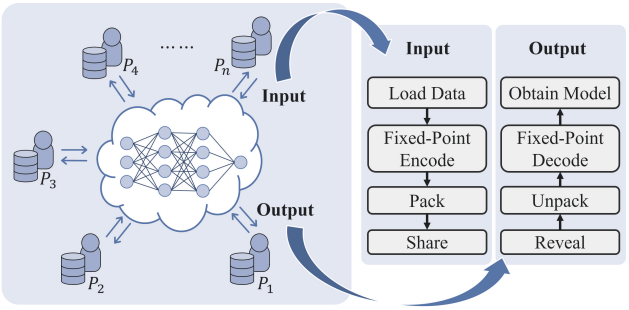

Figure 1: The high-level architecture of our framework.

- $\mathcal{F}_{\text{DouRan}}$. This functionality generates a pair of sharing $\{[\![r]\!]_{2t}, [\![\mathsf{ReEncode}(r)]\!]_t\}$, where $r \in_{\$} R$.
- $\mathcal{F}_{\text{RanPair}}$. This functionality generates a pair of sharing $\{[\![r]\!], [\![\mathsf{Pack}(r_0, \ldots, r_{\rho-1})]\!]\}$, where $r_0, \ldots, r_{\rho-1}$ and r are randomly sampled from $\mathbb{Z}_{2^k}$ such that $\sum_{i=0}^{\rho-1} r_i = r$.
- $\mathcal{F}_{\text{daBit}}$. This functionality generates $\{[\![r]\!]^{\mathcal{A}}, [\![r]\!]^{\mathcal{B}}\}$, where $r = \mathsf{Pack}(r_0, \ldots, r_{\rho-1})$ with each $r_i \in_{\$} \{0, 1\}$;
- $\mathcal{F}_{\text{edaBit}}$. Given a positive integer $m \leqslant k$, this functionality generates $\{[\![r]\!]^{\mathcal{A}}, ([\![r_{m-1}]\!]^{\mathcal{B}}, \ldots, [\![r_0]\!]^{\mathcal{B}})\}$, where $r = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$ with each $a_i \in \mathbb{Z}_{2^k}$ being a random number of length m, satisfying $r = \sum_{i=0}^{m-1} 2^i r_i$.

$\mathcal{F}_{\text{DouRan}}$ and $\mathcal{F}_{\text{RanPair}}$ can be implemented based on a hyper-invertible matrix (Beerliová-Trubíniová and Hirt 2008; Abspoel et al. 2019). $\mathcal{F}_{\text{daBit}}$ and $\mathcal{F}_{\text{edaBit}}$ generate our RMFE-based daBits and edaBits, which can be realized using standard techniques in (Escudero et al. 2020). Throughout this paper, functionalities and protocols designed for packed data, *e.g.*, $\mathcal{F}_{\text{edaBit}}$, have a corresponding non-packed version that handles secrets in $\mathbb{Z}_{2^k}$ rather than R, with elements of $\mathbb{Z}_{2^k}$ embedded into R as the constant term of the polynomial associated with a Galois ring element. We denote this version with a superscript 1, as in $\mathcal{F}_{\text{edaBit}}^{1}$.

Basic Arithmetic on Packed Data

Let $[\![a]\!], [\![b]\!]$ be sharings of two elements of R, where $a = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$, $b = \mathsf{Pack}(b_0, \ldots, b_{\rho-1})$. We introduce the following basic operations.

Linear Operations. Addition of $[\![a]\!], [\![b]\!]$ and multiplication of $[\![a]\!]$ by a public value $c \in R$ are performed locally by each party on their respective shares. Specifically, when $c \in \mathbb{Z}_{2^k}$, the parties can locally compute $[\![\mathsf{Pack}(ca_0, \ldots, ca_{\rho-1})]\!] = c[\![a]\!]$.

Multiplication. The element-wise multiplication of $[\![a]\!]$ and $[\![b]\!]$ can be performed in a similar way to the multiplication in (Damgård and Nielsen 2007), which leverages a double sharing to achieve degree reduction. Algorithm 1 details our protocol Π_{Mult}, where we utilize a double sharing not only for degree reduction but also for re-encoding. The reason for re-encoding is to ensure that the output is suitable for subsequent multiplications.

Since r is random in R, the revealed c in Step 3 leaks no information about a and b. As for correctness, we have

Algorithm 1: Multiplication Π_{Mult}

Input: $[\![a]\!]_t$, $[\![b]\!]_t$;
Output: $[\![g]\!]_t$ where $g = \mathsf{Pack}(a_0b_0, \ldots, a_{\rho-1}b_{\rho-1})$;

1: $\{[\![r]\!]_{2t}, [\![\mathsf{ReEncode}(r)]\!]_t\} \leftarrow \mathcal{F}_{\text{DouRan}}$;
2: $[\![h]\!]_{2t} = [\![a]\!]_t[\![b]\!]_t$;
3: $c = \Pi_{\text{Reveal}}([\![h]\!]_{2t} - [\![r]\!]_{2t})$;
4: $[\![g]\!]_t = \mathsf{ReEncode}(c) + [\![\mathsf{ReEncode}(r)]\!]_t$;

$g = \mathsf{ReEncode}(ab - r) + \mathsf{ReEncode}(r) = \mathsf{ReEncode}(ab) = \mathsf{Pack}(\mathsf{Unpack}(ab)) = \mathsf{Pack}(a_0b_0, \ldots, a_{\rho-1}b_{\rho-1})$.

Remark. A primary difference between our work and previous work like (Huang et al. 2024) is that we require the output $g = \mathsf{Pack}(a_0b_0, \ldots, a_{\rho-1}b_{\rho-1})$ instead of $\mathsf{Unpack}(g) = (a_0b_0, \ldots, a_{\rho-1}b_{\rho-1})$. Since for a vector $x \in \mathbb{Z}_{2^k}^{\rho}$, we don't guarantee that $\mathsf{Unpack}(\mathsf{Pack}(x)) = x$. This equation holds only when $\mathsf{Pack}(\boldsymbol{e}) = 1_R$, which was required by previous work (Huang et al. 2024). We remove this constraint to support a more general form of RMFE. It is noteworthy that the constructions detailed in both Example 1 and Example 2 are exempt from this constraint.

Mixed-Circuit Computation

Binary to Arithmetic. Our construction requires the conversion from binary sharing to arithmetic sharing, which can be achieved through a daBit (Escudero et al. 2020). Algorithm 2 details our protocol Π_{B2A} for this process. Given $[\![a]\!]^{\mathcal{B}}$, where $a = \mathsf{Pack}^{\mathcal{B}}(a_0, \ldots, a_{\rho-1})$, $a_i \in \{0, 1\}$, Π_{B2A} outputs $[\![b]\!]^{\mathcal{A}}$ satisfying $b = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$.

Algorithm 2: Π_{B2A}

Input: $[\![a]\!]^{\mathcal{B}}$;
Output: $[\![b]\!]^{\mathcal{A}}$;

1: $\{[\![r]\!]^{\mathcal{A}}, [\![r]\!]^{\mathcal{B}}\} \leftarrow \mathcal{F}_{\text{daBit}}$;
2: $c = \Pi^{\mathcal{B}}_{\text{Reveal}}([\![a]\!]^{\mathcal{B}} + [\![r]\!]^{\mathcal{B}})$;
3: $[\![b]\!]^{\mathcal{A}} = c + [\![r]\!]^{\mathcal{A}} - 2\Pi_{\text{Mult}}(c, [\![r]\!]^{\mathcal{A}})$;

Note that in Step 3, even if c is public, we still need to invoke Π_{Mult} since we need element-wise multiplication.

Truncation. Algorithm 3 details our protocol for parallel truncation. Given $[\![a]\!]$ where $a = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$ with each a_i being an integer in $\mathbb{Z}_{2^k}$ of length ℓ, and an integer m satisfying $1 \leqslant m \leqslant \ell - 1$, Π_{Trun} outputs $[\![d]\!]$ such that $d = \mathsf{Pack}(\lfloor a_0/2^m \rfloor, \ldots, \lfloor a_{\rho-1}/2^m \rfloor)$. Π_{Trun} adapts the truncation in (Zhou 2025a) by integrating a boolean-circuit functionality $\mathcal{F}_{\text{BitLT}}$ to compute the bit-wise comparison, which can be realized by following the steps of BitLTC1 in (Catrina and de Hoogh 2010) and using Π_{Mult} to compute the underlying prefix multiplication sequentially.

The way we calculate truncation, originally introduced in (Catrina and de Hoogh 2010), utilizes a random number to mask the input in a manner that no wraparound happens. The result is then computed based on subtracting the random number's truncation from the masked value's truncation. To ensure masking, the random number must be sufficiently large, necessitating a security parameter κ. Same as

Algorithm 3: Deterministic Truncation Π_{Trun}

Input: $[\![a]\!]$, m;
Output: $[\![d]\!]$;

1: $\{[\![r'']\!]^{\mathcal{A}}, ([\![r_i]\!]^{\mathcal{B}})_{i=m}^{\ell+\kappa-1}\} \leftarrow \mathcal{F}_{\text{edaBit}}(\ell + \kappa - m)$;
2: $\{[\![r']\!]^{\mathcal{A}}, ([\![r_i]\!]^{\mathcal{B}})_{i=0}^{m-1}\} \leftarrow \mathcal{F}_{\text{edaBit}}(m)$;
3: $c = \Pi_{\text{Reveal}}(\mathsf{Pack}(2^{\ell-1}\boldsymbol{e}) + [\![a]\!] + 2^m[\![r'']\!] + [\![r']\!])$;
4: $(c_0, \ldots, c_{\rho-1}) = \mathsf{PackInv}(c)$;
5: $c' = \mathsf{Pack}(c_0 \bmod 2^m, \ldots, c_{\rho-1} \bmod 2^m)$;
6: $[\![u]\!]^{\mathcal{B}} = \mathcal{F}_{\text{BitLT}}(c', ([\![r_i]\!]^{\mathcal{B}})_{i=0}^{m-1})$;
7: $[\![u]\!]^{\mathcal{A}} = \Pi_{\text{B2A}}([\![u]\!]^{\mathcal{B}})$;
8: $[\![d]\!] = (c - c')2^{-m} - [\![r'']\!] - [\![u]\!]^{\mathcal{A}} - \mathsf{Pack}(2^{\ell-m-1}\boldsymbol{e})$;

in (Catrina and de Hoogh 2010), we can remove the calculation of u to get an efficient probabilistic truncation. Correspondingly, we call Π_{Trun} the deterministic truncation.

Integer Comparison. The comparison between two numbers can be computed by extracting the most significant bit of their difference, which reduces to a deterministic truncation (Catrina and de Hoogh 2010; Damgård et al. 2019).

Secure Primitives for Neural Network Layers

Activation Function. We use ReLU as our activation function. To enable training, we also need to compute its derivative $\mathrm{DReLU}(x)$, which is defined to be 1 if $x \geqslant 0$ and 0 otherwise. DReLU can be implemented through a deterministic truncation (Catrina and de Hoogh 2010), and we can compute ReLU as $\mathrm{ReLU}(x) = x \cdot \mathrm{DReLU}(x)$.

Pooling. We consider both max pooling and average pooling. Max pooling can be computed based on ReLU (Liu, Xiang, and Yu 2024), and average pooling can be seen as a division by a public value, thus reduced to a truncation.

Loss Function. When using the softmax function and the cross-entropy loss, the corresponding gradient can be approximated by ReLU and division (Wagh, Gupta, and Chandran 2019). Here, the division can be calculated bit by bit, as detailed in (Zhou 2023).

Parallel Training Framework

We now introduce how to achieve parallel training on the basis of RMFE-based computation. We follow the training paradigm in (Zhou 2025b) to compute the gradient for each training sample in a mini-batch in parallel. Figure 2 shows the complete process of training a fully connected network.

Split and Expand. We first introduce two key subprotocols, Π_{Split} and Π_{Expand}. Given a sharing $[\![a]\!]$ where $a \in \mathbb{Z}_{2^k}$, Π_{Split} outputs $[\![\mathsf{Pack}(a_0, \ldots, a_{\rho-1})]\!]$ satisfying $a_0 + \cdots + a_{\rho-1} = a$, and Π_{Expand} outputs $[\![\mathsf{Pack}(a, \ldots, a)]\!]$. Π_{Expand} can be computed locally as $[\![\mathsf{Pack}(a\boldsymbol{e})]\!] = [\![a]\!]\mathsf{Pack}(\boldsymbol{e})$. As for Π_{Split}, we can compute it based on $\mathcal{F}_{\text{RanPair}}$, the details are shown in Algorithm 4.

Gradient Aggregation. Once the gradients for each sample are calculated, we need to aggregate and average these

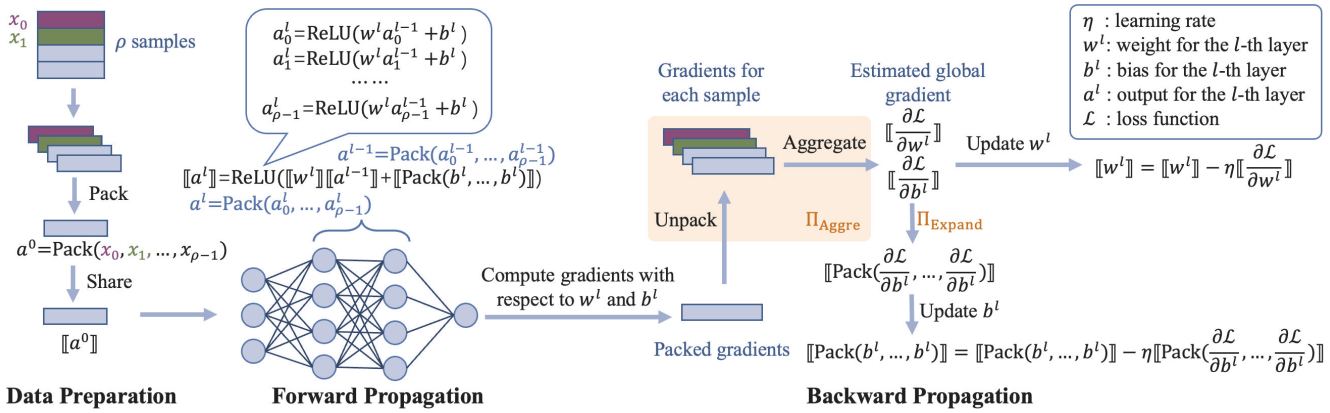

Figure 2: Parallel training framework.

Domain	Protocol	Circuit	(ρ, d)	LAN		WAN		Communication(KB)*	
				Offline(s)	Online(ms)	Offline(s)	Online(s)	Offline	Online
p	Π_{TruncF}	Arithmetic	–	11.92	144.35	83.71	4.48	55.02	0.63
	Π_{EdabitsF}	Mixed	–	1.50	29.99	45.53	3.39	32.92	0.18
2^k	Π_{EdabitsR}	Mixed	$(1,3)$	0.63	64.40	17.64	5.29	14.16	0.69
			$(2,3)$	0.43	43.59	15.82	5.16	9.86	0.35
	Π_{TrunGR}	Arithmetic	$(1,3)$	3.93	67.42	44.54	1.51	42.20	0.88
			$(2,3)$	2.63	40.42	34.86	1.35	32.49	0.44
	Π_{Trun} (Ours)	Mixed	$(1,3)$	**0.85**	**17.83**	**26.51**	**1.30**	**34.27**	**0.20**
			$(2,3)$	**0.69**	**13.52**	**23.80**	**1.23**	**23.98**	**0.11**

* The communication is reported as the amortized communication per party for a single truncation.

Table 1: Performance comparison for $10,000$ parallel deterministic truncations of 13 bits.

Algorithm 4: Π_{Split}

Input: $[\![a]\!]$;
Output: $[\![g]\!]$;

1: $\{[\![r]\!], [\![\mathsf{Pack}(r_0, \ldots, r_{\rho-1})]\!]\} \leftarrow \mathcal{F}_{\text{RanPair}}$;
2: $c = \Pi_{\text{Reveal}}([\![a]\!] + [\![r]\!])$;
3: $[\![g]\!] = \mathsf{Pack}(c\boldsymbol{e_0}) - [\![\mathsf{Pack}(r_0, \ldots, r_{\rho-1})]\!]$;

individual gradients. This process can be regarded as a special truncation. Based on Π_{Trun}, we propose Π_{Aggre} to handle this process. The details are shown in Algorithm 5.

Updating Parameters. As shown in Figure 2, for the l-th layer, the weight w^l and bias b^l require different sharing forms, as the former is applied via multiplication and the latter via addition. Therefore, after the gradients for b^l are computed, Π_{Expand} must be invoked on them before updating the value of b^l.

Experiments

Our experiments consist of a microbenchmark focused on truncation as well as end-to-end training of neural networks.

Algorithm 5: Π_{Aggre}

Input: $[\![a]\!]$, m, where $a = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$;
Output: $[\![d]\!]$ satisfying $d = \lfloor (a_0 + \cdots + a_{\rho-1})/2^m \rfloor$;

1: $\{[\![r'']\!]^{\mathcal{A}}, ([\![r_i]\!]^{\mathcal{B}})_{i=m}^{\ell+\kappa-1}\} \leftarrow \mathcal{F}^1_{\text{edaBit}}(\ell + \kappa - m)$;
2: $\{[\![r']\!]^{\mathcal{A}}, ([\![r_i]\!]^{\mathcal{B}})_{i=0}^{m-1}\} \leftarrow \mathcal{F}^1_{\text{edaBit}}(m)$;
3: $[\![r]\!]^{\mathcal{A}} = 2^m [\![r'']\!]^{\mathcal{A}} + [\![r']\!]^{\mathcal{A}}$;
4: $[\![v]\!] = \Pi_{\text{Split}}([\![r]\!])$; //$v = \mathsf{Pack}(v_0, \ldots, v_{\rho-1})$
5: $c = \Pi_{\text{Reveal}}(\mathsf{Pack}(2^{\ell-1}\boldsymbol{e_0}) + [\![a]\!] + [\![v]\!])$;
6: $(c_0, \ldots, c_{\rho-1}) = \mathsf{PackInv}(c)$;
7: $c' = (c_0 + \cdots + c_{\rho-1}) \bmod 2^m$;
8: $[\![u]\!]^{\mathcal{B}} = \mathcal{F}^1_{\text{BitLT}}(c', ([\![r_i]\!]^{\mathcal{B}})_{i=0}^{m-1})$;
9: $[\![u]\!]^{\mathcal{A}} = \Pi^1_{\text{B2A}}([\![u]\!]^{\mathcal{B}})$;
10: $[\![d]\!] = (c_0 + \cdots + c_{\rho-1} - c')2^{-m} - [\![r'']\!] - [\![u]\!]^{\mathcal{A}} - 2^{\ell-m-1}$;

Experimental Setup

Execution Environment. All our experiments were conducted on a server equipped with an AMD EPYC 9754 128-Core Processor and 512 GB of RAM. We configured two network settings using the Linux Traffic Control:

- **LAN setting**: 4 Gbps bandwidth with sub-millisecond

Framework	Pooling	(ρ, d)	Network A (FCN)			Network B (CNN)			Network C (CNN)		
			LAN	WAN	Comm.	LAN	WAN	Comm.	LAN	WAN	Comm.
(Zhou 2025a)	Max	$(1,3)$	0.76	36.74	0.28	26.26	228.25	7.20	42.68	323.61	10.57
	Avg	$(1,3)$	–	–	–	8.68	84.03	1.91	17.20	114.91	2.97
	Max	$(2,3)$	0.44	35.18	0.14	12.19	120.38	3.60	23.88	165.32	5.30
	Avg	$(2,3)$	–	–	–	4.56	55.11	0.96	9.35	70.35	1.50
Ours	Max	$(1,3)$	0.34	34.78	0.07	7.84	81.30	1.79	15.69	103.18	2.65
	Avg	$(1,3)$	–	–	–	3.71	45.73	0.46	9.81	56.67	0.74
	Max	$(2,3)$	**0.16**	**34.00**	**0.04**	**3.86**	**63.71**	**0.90**	**8.34**	**75.82**	**1.34**
	Avg	$(2,3)$	–	–	–	**1.88**	**40.65**	**0.23**	**5.03**	**46.90**	**0.39**

Table 2: The time(h) and communication(TB) required for 3 epochs of online training between 3 parties.

round-trip time (RTT).

- **WAN setting**: 100 Mbps bandwidth with 40 ms RTT.

Implementation Details. We implemented our framework in C++ and used the Eigen library for matrix multiplication. We set the security parameter $\kappa = 40$ and the number of adversaries $t = \lfloor (n-1)/2 \rfloor$. All random sharings required for the protocols were generated in an offline phase. For fixed-point representation, we set $k = 128$ and the data range $\ell = 60$. Our experiments included two kinds of RMFE constructions. When $\rho = 2$, we used the construction in Example 2, which is applicable for $d \geqslant 3$. When $\rho = 3$, we used the construction in Example 1.

Baselines. For microbenchmark on truncation, we compare our protocol against its arithmetic-only counterpart in (Zhou 2025a), which we refer to as Π_{TrunGR}. We also consider the work in (Catrina and de Hoogh 2010) and (Escudero et al. 2020). The former is the first to consider truncation on fixed-point numbers over $\mathbb{F}_p$, and the latter is notable for enabling mixed-circuit computation and removing the security parameter over $\mathbb{Z}_{2^k}$. We refer to the protocol in (Catrina and de Hoogh 2010) as Π_{TruncF}, and the protocols in (Escudero et al. 2020) over $\mathbb{F}_p$ and $\mathbb{Z}_{2^k}$ as Π_{EdabitsF} and Π_{EdabitsR}, respectively. For end-to-end training, our main baseline is the coefficient-packing framework over Galois rings (Zhou 2025a). We also benchmark against two established three-party (3PC) frameworks, ABY^3 (Mohassel and Rindal 2018) and SecureNN (Wagh, Gupta, and Chandran 2019), to position our work within a broader PPML landscape.

Networks. We conduct training on the MNIST dataset (Deng 2012) using the same Network A, B, and C in (Wagh, Gupta, and Chandran 2019). Network A is a 3-layer fully connected neural network (FCN) with 128 neurons in each hidden layer. Network B and Network C are both convolutional neural networks (CNN), where Network B comes from (Liu et al. 2017), containing 2 convolution layers and 2 fully connected layers, and Network C is a variant of LeNet (Lecun et al. 1998). According to (Wagh, Gupta, and Chandran 2019), after 15 epochs of training, the accuracy of these three networks can reach 93.4%, 98.77%, and 99.15%, respectively.

Microbenchmark: Truncation Performance

We first benchmarked our protocol Π_{Trun} by executing 10,000 parallel 13-bit truncations between 3 parties. The results in Table 1 confirm that our protocol is the most efficient among all baselines. A key comparison is with Π_{EdabitsR}, which achieves minimal communication (using $k = 64$) by eliminating the security parameter. However, this comes at the cost of two online comparison rounds, hindering its practical performance. For instance, in the LAN setting with $\rho = 1$, while Π_{EdabitsR} is 25.9% faster offline, our protocol demonstrates a 72.3% speedup in the critical online phase.

End-to-End Training Performance

For our end-to-end evaluation, we benchmarked our framework against the work in (Zhou 2025a) across all three networks. We used probabilistic truncation with $f = 13$ and a batch size of 128. As detailed in Table 2, our framework consistently outperforms the baseline. Specifically, under the same packing density, we reduce communication by 71.4-75.1% and accelerate online training by 55.3-70.1% in LAN and 5.3-68.1% in WAN settings. This significant gain is directly attributable to our integration of mixed-circuit computation. The improvement in WAN is less pronounced for simpler Network A, where performance is primarily bottlenecked by the number of communication rounds. Table 2 also shows the impact when we switch from max pooling to average pooling. It can be seen that average pooling is significantly more efficient than max pooling because it circumvents the need for costly comparison operations.

We further compared our framework against ABY^3 and SecureNN, two classic 3PC frameworks, and the relevant results for Network A are presented in Table 3. The results demonstrate that our framework, while designed for the more challenging setting of a general number of parties, achieves a runtime competitive with these specialized 3PC protocols.

Scalability and Packing Efficiency. To validate the scalability of our framework, we conducted private training using different RMFEs on Network A, B, and C, with the number of parties n ranging from 3 to 21. We set the batch size to 120 and the fixed-point precision f to 20. We used average pooling and only our deterministic truncation Π_{Trun} to avoid

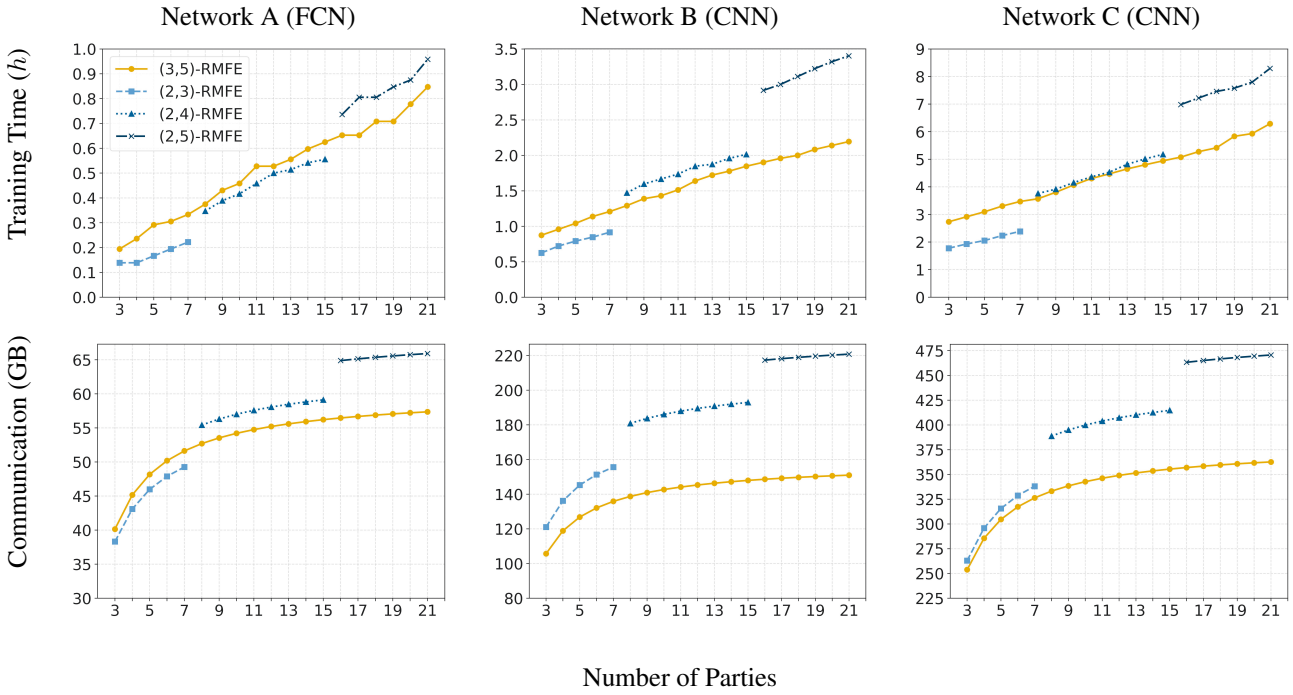

Figure 3: The time and amortized communication required for each party for an epoch of online training.

Framework	ρ	Time(h)	Communication(TB)
ABY^3	1	0.75	N/A
SecureNN	1	1.03	0.11
(Zhou 2025a)	2	2.21	0.72
Ours	2	**0.78**	**0.19**

Table 3: The time and communication required for 15 training epochs among 3 parties on Network A.

the security issue of probabilistic truncation identified in (Li et al. 2023). As shown in Figure 3, the near-linear growth in training time with the number of parties confirms the excellent scalability of our framework.

Figure 3 also reveals the benefits of supporting more general RMFE constructions. We compared a $(3, 5)$-RMFE against various $(2, d)$-RMFEs, which corresponds to the packing method used in (Zhou 2025a). The different settings of d are to ensure that $n < 2^d$. In terms of execution time, for $n \geqslant 16$, the $(3, 5)$-RMFE offers significant improvements. When $8 \leqslant n \leqslant 15$, there is a trade-off between d and ρ. For Network A, $(3, 5)$-RMFE is slightly slower than $(2, 4)$-RMFE due to the increased packing-related computation. However, on complex CNN networks that require more communication, $(3, 5)$-RMFE still maintains an advantage. From the perspective of communication, we can see that for experiments on CNNs, the $(3, 5)$-RMFE outperforms all $(2, d)$-RMFEs across all tested participant numbers, even though the theoretical packing density ρ/d of $(3, 5)$-RMFE is actually lower than that of $(2, 3)$-RMFE. This practical advantage stems from a low-level implementation detail. When using `std::bitset<d>` to represent an element of $GR(2, d)$, setting $d = 3, 4$, and 5 are all aligned to the same byte boundary, occupying identical memory and thus neutralizing the theoretical disadvantage of a larger d.

Accuracy Analysis. The accuracy of our protocols depends on the fixed-point precision f. Table 4 compares plaintext and ciphertext training accuracy on Network A, demonstrating that ciphertext accuracy approaches plaintext levels as f increases. This highlights the necessity of using a higher precision, *e.g.*, $f = 20$, for reliable convergence.

Epochs	Plaintext	Fixed-Point Precision f		
		13	16	20
5	77.50%	55.19%	81.38%	75.49%
10	92.35%	55.40%	69.91%	91.33%
15	92.97%	52.63%	80.11%	91.78%

Table 4: The impact of f on the training accuracy.

Conclusion

We introduce a scalable framework for privacy-preserving neural network training over Galois rings, which combines RMFE for higher packing density and mixed-circuit for fast nonlinear computation. Our construction supports a general form of RMFE. The experiments validate the efficiency and scalability of our framework.

Acknowledgments

The work was supported in part by the National Natural Science Foundation of China under Grants 12426302, 12361141818, 12031011 and the National Key Research and Development Program of China under Grants 2022YFA1004900.

References

Abspoel, M.; Cramer, R.; Damgård, I.; Escudero, D.; and Yuan, C. 2019. Efficient information-theoretic Secure Multiparty Computation over $\mathbb{Z}/p^k\mathbb{Z}$ via Galois rings. In *Proceedings of the Theory of Cryptography Conference*, 471–501.

Agrawal, N.; Shahin Shamsabadi, A.; Kusner, M. J.; and Gascón, A. 2019. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, 1231–1247. New York, NY, USA: Association for Computing Machinery. ISBN 9781450367479.

Baccarini, A. N.; Blanton, M.; and Yuan, C. 2023. Multi-Party Replicated Secret Sharing over a Ring with Applications to Privacy-Preserving Machine Learning. *Proceedings on Privacy Enhancing Technologies*, 2023: 608–626.

Beerliová-Trubíniová, Z.; and Hirt, M. 2008. Perfectly-Secure MPC with Linear Communication Complexity. In *Theory of Cryptography*, 213–230. Berlin, Heidelberg: Springer Berlin Heidelberg.

Canetti, R. 2001. Universally composable security: a new paradigm for cryptographic protocols. In *Proceeding of Foundations of Computer Science*, 136–145.

Catrina, O.; and de Hoogh, S. 2010. Improved primitives for secure multi-party integer computation. In *International Conference on Security and Cryptography for Networks*, 182–199.

Chaudhari, H.; Rachuri, R.; and Suresh, A. 2020. Trident: efficient 4PC framework for privacy preserving machine learning. In *Symposium on Network and Distributed System Security (NDSS)*.

Cramer, R.; Damgård, I.; Escudero, D.; Scholl, P.; and Xing, C. 2018. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *Advances in Cryptology – CRYPTO 2018*, 769–798. Cham: Springer International Publishing. ISBN 978-3-319-96881-0.

Cramer, R.; Rambaud, M.; and Xing, C. 2021. Asymptotically-Good Arithmetic Secret Sharing over $\mathbb{Z}/p^\ell\mathbb{Z}$ with Strong Multiplication and Its Applications to Efficient MPC. In *Advances in Cryptology – CRYPTO 2021*, 656–686. Cham: Springer International Publishing. ISBN 978-3-030-84252-9.

Damgård, I.; Escudero, D.; Frederiksen, T.; Keller, M.; Scholl, P.; and Volgushev, N. 2019. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, 1102–1120.

Damgård, I.; and Nielsen, J. B. 2007. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, 572–590.

Damgård, I.; Pastro, V.; Smart, N.; and Zakarias, S. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2012*, 643–662. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-32009-5.

Deng, L. 2012. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6): 141–142.

Escudero, D.; Ghosh, S.; Keller, M.; Rachuri, R.; and Scholl, P. 2020. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, 823–852. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-56879-5.

Escudero, D.; Hong, C.; Liu, H.; Xing, C.; and Yuan, C. 2023. Degree-D Reverse Multiplication-Friendly Embeddings: Constructions and Applications. In *Advances in Cryptology – ASIACRYPT 2023*, 106–138. Singapore: Springer Nature Singapore. ISBN 978-981-99-8721-4.

Franklin, M.; and Yung, M. 1992. Communication Complexity of Secure Computation (Extended Abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, 699–710.

Huang, Z.; Lu, W.; Wang, Y.; Hong, C.; Wei, T.; and Chen, W. 2024. Coral: Maliciously Secure Computation Framework for Packed and Mixed Circuits. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, 810–824. New York, NY, USA: Association for Computing Machinery. ISBN 9798400706363.

Ito, M.; Saito, A.; and Nishizeki, T. 1989. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan Part III-Fundamental Electronic Science*, 72: 56–64.

Keller, M.; and Sun, K. 2022. Secure Quantized Training for Deep Learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 10912–10938. PMLR.

Koti, N.; Patra, A.; Rachuri, R.; and Suresh, A. 2022. Tetrad: Actively Secure 4PC for Secure Training and Inference. In *Symposium on Network and Distributed System Security (NDSS)*.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Li, Y.; Duan, Y.; Huang, Z.; Hong, C.; Zhang, C.; and Song, Y. 2023. Efficient 3PC for binary circuits with application to maliciously-secure DNN inference. In *Proceedings of the 32nd USENIX Conference on Security Symposium*.

Liu, F.; Xiang, X.; and Yu, Y. 2024. Scalable Multi-Party Computation Protocols for Machine Learning in the Honest-Majority Setting. In *Proceedings of the 33rd USENIX Conference on Security Symposium*.

Liu, J.; Juuti, M.; Lu, Y.; and Asokan, N. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 619–631. New York, NY, USA: Association for Computing Machinery. ISBN 9781450349468.

Mohassel, P.; and Rindal, P. 2018. ABY^3: a mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 35–52.

Mohassel, P.; and Zhang, Y. 2017. SecureML: a system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, 19–38.

Orsini, E.; Smart, N. P.; and Vercauteren, F. 2020. Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption. In *CT-RSA*, 254–283. ISBN 978-3-030-40186-3.

Patra, A.; and Suresh, A. 2020. BLAZE: blazing fast privacy-preserving machine learning. In *Symposium on Network and Distributed System Security (NDSS)*.

Rotaru, D.; and Wood, T. 2019. MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security. In *Progress in Cryptology – INDOCRYPT 2019: 20th International Conference on Cryptology in India, Hyderabad, India, December 15–18, 2019, Proceedings*, 227–249. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-35422-0.

Shamir, A. 1979. How to share a secret. *Communications of the ACM*, 22(11): 612–613.

Wagh, S.; Gupta, D.; and Chandran, N. 2019. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3): 26–49.

Wagh, S.; Tople, S.; Benhamouda, F.; Kushilevitz, E.; Mittal, P.; and Rabin, T. 2021. FALCON: honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021(1): 188–208.

Wan, Z. 2003. Lectures on Finite Fields and Galois Rings. *World Scientific Publishing Company*.

Yao, A. C.-C. 1982. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, 160–164.

Yao, A. C.-C. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science*, 162–167.

Zhou, H. 2023. Information-Theoretically Secure Neural Network Training with Flexible Deployment. In *Artificial Neural Networks and Machine Learning – ICANN 2023*, 324–336. Cham: Springer Nature Switzerland. ISBN 978-3-031-44192-9.

Zhou, H. 2025a. Private Multi-Party Neural Network Training over $\mathbb{Z}_{2^k}$ via Galois Rings. Cryptology ePrint Archive, Paper 2025/331.

Zhou, H. 2025b. Private Neural Network Training with Packed Secret Sharing. In *Computing and Combinatorics*, 66–77. ISBN 978-981-96-1090-7.

Zhu, X.; Vondrick, C.; Fowlkes, C. C.; and Ramanan, D. 2016. Do We Need More Training Data? *Int. J. Comput. Vision*, 119(1): 76–92.