EFFICIENT DETERMINISTIC TRUNCATION AND RMFE-NATIVE FIXED-POINT OPERATORS FOR PRIVATE NEURAL NETWORK TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Fixed-point arithmetic is widely used in privacy-preserving neural network training, but securely restoring the target scale of intermediate values remains costly. In the semi-honest, honest-majority setting with general n-party Shamir sharing, existing mask-and-reveal deterministic truncation approaches either introduce a statistical security parameter κ to prevent modular wraparound or explicitly correct wraparound at the cost of sequential secure comparisons. In this work, we develop a more efficient deterministic truncation protocol for Shamir secret sharing over Galois rings. We revisit modular wraparound in the mask-and-reveal paradigm and incorporate its correction directly into the truncation relation. For Galois rings of characteristic 2^k, the formulation further simplifies, yielding a κ-free deterministic truncation protocol that requires only one online secure comparison. We further integrate this truncation protocol into the RMFE-based training framework and develop new protocols for efficient fixed-point multiplication and gradient aggregation, two key components of RMFE-based private neural network training for general n parties. Experiments on fully connected and convolutional neural networks demonstrate up to a $1.70\times$ speedup in the LAN setting, together with substantial communication savings over the existing RMFE-based training framework.

1 INTRODUCTION

Privacy-preserving machine learning (PPML) enables multiple parties to jointly train neural networks without revealing their private data. Secret-sharing-based secure multi-party computation (MPC) provides a natural cryptographic foundation for this task (Yao, 1982; 1986). However, neural network computation inherently involves values over $\mathbb{R}$, whereas MPC protocols typically operate over discrete algebraic structures such as $\mathbb{F}_p$ (Ben-Or et al., 1988; Damgård et al., 2012) or $\mathbb{Z}_{2^k}$ (Cramer et al., 2018). Fixed-point representation bridges this gap by encoding real values as scaled integers (Catrina & Saxena, 2010), allowing neural network computation to be carried out using integer arithmetic.

A fundamental operation in fixed-point computation is rescaling. When two fixed-point values with f fractional bits are multiplied, their product carries a scale factor of 2^{2f} and must be rescaled to restore the target precision. Cryptographically, exact rescaling is typically realized through deterministic truncation, which securely computes $\lfloor a/2^m \rfloor$ for a secret-shared value a. Truncation also serves as an important building block for several operations in private neural network computation, including comparison (Escudero et al., 2020), ReLU (Catrina & de Hoogh, 2010), and pooling (Liu et al., 2024). Consequently, the efficiency of deterministic truncation has a direct impact on the practical performance of fixed-point private neural network computation.

1.1 CHALLENGES AND OUR APPROACH

A large body of work has developed efficient PPML frameworks based on fixed-point MPC (Mohassel & Rindal, 2018; Wagh et al., 2019; Patra & Suresh, 2020; Chaudhari et al., 2020; Koti et al., 2022; Wagh et al., 2021; Baccarini et al., 2023; Keller & Sun, 2022), although many of these systems are specialized to a small fixed number of parties. More recent works have also considered

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

private machine learning with a general number of participants using additive secret sharing (Aaraj et al., 2024; Lv et al., 2026) or Shamir sharing (Zhou, 2026). In this work, we focus on Shamir sharing over Galois rings, which enables information-theoretically secure general n-party MPC in the honest-majority setting and serves as the cryptographic foundation for the RMFE-based private training framework that we build upon (Zhou, 2026). Within this setting, we study how to improve fixed-point computation for private neural network training.

In the general n-party setting, a widely used approach to private truncation follows the mask-and-reveal paradigm of Catrina & de Hoogh (2010). The core idea is to mask the input with a shared random number, reveal this masked value, and calculate the result by subtracting the random number's truncation from the masked value's truncation. In its original form, the mask-and-reveal paradigm requires the input to be masked in a manner that guarantees no wraparound occurs. This necessitates a statistical security parameter κ, forcing protocols to operate over oversized mathematical structures and resulting in extra communication and computational overhead. The construction of Escudero et al. (2020) eliminates κ by permitting wraparound over tight rings. However, to ensure correctness, it requires two sequential invocations of secure bitwise comparisons in the online phase to correct modular wraparound, leading to increased latency. The RMFE-based training framework (Zhou, 2026) prioritizes low latency by adhering to the traditional mask-and-reveal paradigm and leveraging mixed-circuit techniques for comparison. While this approach demonstrates that a single secure comparison invocation over oversized rings with κ can outperform the multi-round logic of Escudero et al. (2020) over tight κ-free rings, it remains fundamentally constrained by the excessive communication overhead inherent to larger ring structures.

Secure truncation has also been studied extensively in other MPC settings, including efficient probabilistic truncation (Dalskov et al., 2020; Liu et al., 2024), which has been shown to exhibit security vulnerabilities (Li et al., 2023), as well as truncation protocols tailored to a fixed number of parties (Zou et al., 2023; Zbudila et al., 2024; Harth-Kitzerow et al., 2025). These works address different sharing schemes, security models, or party configurations from the general n-party Shamir setting considered here. Our key observation is that, in our setting, modular wraparound can be incorporated directly into the truncation relation rather than handled through a separate correction step. The wraparound state can be identified from the most significant bit of the random mask and that of the publicly revealed masked value, avoiding secure computation of the full-width modular carry. For Galois rings of characteristic 2^k, the formulation further simplifies, and the lower-bit carry correction becomes independent of the wraparound state, yielding a κ-free deterministic truncation protocol that requires only one online secure comparison. To connect our truncation protocol more directly to private neural network training, we further consider its use within the RMFE-based training framework (Zhou, 2026), which enables multiple fixed-point values to be packed into a single Galois ring element for SIMD-style computation. Building on this framework, we extend our deterministic truncation to RMFE-based packing and develop new protocols for efficient fixed-point multiplication and gradient aggregation. These constructions incorporate our κ-free truncation mechanism into key components of RMFE-based private neural network training.

1.2 Our Contributions

Our main contributions are summarized as follows:

- We develop a more efficient deterministic truncation protocol with perfect security over $\mathbb{Z}_{2^k}$ for general n-party Shamir sharing over Galois rings. Our key technique incorporates modular wraparound correction directly into the truncation relation. The resulting protocol requires only a 1-bit gap between the input bit length ℓ and the ring bit-width k, i.e., $\ell \leqslant k-1$, eliminates the statistical security parameter κ, and requires only one online secure comparison.
- Building on our deterministic truncation protocol, we extend it to RMFE-based packing and develop new protocols for efficient fixed-point multiplication and gradient aggregation. These constructions extend the benefit of our κ-free truncation mechanism to the existing RMFE-based training framework.
- We implement our protocols within the RMFE-based private training framework and evaluate the training performance on fully connected and convolutional neural networks. By integrating our efficient truncation and RMFE-native fixed-point operators, the resulting

framework substantially reduces both computation and communication compared with the previous RMFE-based framework, achieving up to a $1.70\times$ speedup in the LAN setting while retaining SIMD-style packing and scalability in the number of participants. Our implementation is available at `https://anonymous.4open.science/r/grtc/`.

2 PRELIMINARIES

2.1 SECURITY MODEL

We consider secure computation among n parties $P_1, \ldots, P_n$ ($n \geqslant 3$), assuming the existence of peer-to-peer secure channels. We focus on the semi-honest, honest-majority setting, where at most $t < n/2$ parties may be corrupted and adversaries strictly follow the protocol specification. The security of our protocols is analyzed under the standard Universal Composability (UC) framework (Canetti, 2001). A protocol Π securely realizes a functionality $\mathcal{F}$ if, for any environment $\mathcal{Z}$, the view of $\mathcal{Z}$ interacting with Π and an adversary $\mathcal{A}$ is indistinguishable from its view interacting with $\mathcal{F}$ and a simulator $\mathcal{S}$. A protocol Π is statistically secure if these two views are statistically indistinguishable, and perfectly secure if they are perfectly indistinguishable.

2.2 SHAMIR SECRET SHARING SCHEME

To establish a Shamir secret sharing scheme over a commutative ring R, we require n distinct nonzero elements $\beta_1, \ldots, \beta_n$ that, together with 0, form an exceptional sequence (Abspoel et al., 2019). This ensures that the difference between any two elements in the sequence is invertible in R. The maximum length of such a sequence is called the Lenstra constant of R, which bounds the number of participants n. To share a secret $u \in R$ among n parties against t adversaries, a dealer generates a random polynomial $f(x) \in R[x]$ of degree t with $f(0) = u$, and sends the share $u^i = f(\beta_i)$ to party P_i. The vector $(u^1, \ldots, u^n)$ is called a t-sharing of u, formally denoted by $[\![u]\!]_t$, from which the secret can be reconstructed via Lagrange interpolation using any $t+1$ shares.

Sharing Notations Shamir sharing over $\mathbb{Z}_{2^k}$ is infeasible because its Lenstra constant is only 2. We therefore work over $GR(2^k, d)$, whose Lenstra constant 2^d supports $n < 2^d$ parties (Wan, 2003). We adopt the unified notation $[\![\cdot]\!]_t^{\mathcal{T}}$ to jointly specify the underlying polynomial degree t and the algebraic setting. When $\mathcal{T} = \mathcal{R}$, $[\![\cdot]\!]_t^{\mathcal{R}}$ denotes an arithmetic sharing over $GR(2^k, d)$. When $\mathcal{T} = \mathcal{B}$, $[\![\cdot]\!]_t^{\mathcal{B}}$ denotes a Boolean sharing over $GR(2, d)$, corresponding to $k = 1$ for mixed-circuit evaluation. The superscript $\mathcal{T}$ is also consistently applied to secure protocols and ideal functionalities to indicate the corresponding algebraic setting. For the sake of conciseness, we omit the subscript t when the degree is self-evident, and any notation without an explicit superscript, such as $[\![\cdot]\!]$, denotes the arithmetic setting $[\![\cdot]\!]^{\mathcal{R}}$ unless otherwise specified.

Operations The Shamir scheme allows for local addition and scalar multiplication. Interactive primitives, such as secure reconstruction and multiplication via the double-sharing technique (Damgård & Nielsen, 2007), are abstracted as $\Pi_{\text{Reveal}}^{\mathcal{T}}$ and $\Pi_{\text{Mult}}^{\mathcal{T}}$, respectively, where $\mathcal{T} \in \{\mathcal{R}, \mathcal{B}\}$.

RMFE-Based Packing We follow the RMFE-based framework (Zhou, 2026) to pack ρ values into $GR(2^k, d)$ for SIMD computation. Specifically, let (ϕ, ψ) be a (ρ, d)-RMFE over $\mathbb{Z}_{2^k}$, where $\phi : \mathbb{Z}_{2^k}^{\rho} \to GR(2^k, d)$ and $\psi : GR(2^k, d) \to \mathbb{Z}_{2^k}^{\rho}$ are $\mathbb{Z}_{2^k}$-linear maps satisfying $\psi(\phi(\boldsymbol{x})\phi(\boldsymbol{y})) = \boldsymbol{x} \odot \boldsymbol{y}$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_{2^k}^{\rho}$. We write $\mathsf{Pack} = \phi$, $\mathsf{PackInv} = \phi^{-1}$ on valid packed elements, $\mathsf{Unpack} = \psi$ for extracting component-wise values after multiplication, and $\mathsf{ReEncode} = \phi \circ \psi$ to map a Galois ring element back to a valid packed representation, as required before subsequent packed multiplication.

2.3 PRIVATE TRUNCATION AND THE MASK-AND-REVEAL PARADIGM

Fixed-Point Representation and Notations Fixed-point representation (Catrina & Saxena, 2010) represents real numbers as scaled integers. A real number $y \in \mathbb{R}$ is transformed to a fixed-point approximation $x \in \mathbb{Z}$ by setting $x = \lfloor y \cdot 2^f \rfloor$, where f is the number of fractional bits. We assume the valid range of such fixed-point integers is bounded by a bit-length ℓ. Then, we use the function $\mathrm{Rep} : \mathbb{Z} \to \mathbb{Z}_N$, $\mathrm{Rep}(x) = x \bmod N$ to map an element in $\mathbb{Z}$ into $\mathbb{Z}_N$. For any $a \in \mathbb{Z}_N$, we

denote its canonical integer representative in $[0, N-1]$ as $\hat{a}$ and its most significant bit evaluated at index $\lceil \log_2 N \rceil - 1$ as a_{msb} (0-indexed). We introduce the notation $[\cdot]_m$ to represent the m least significant bits of its argument where $[x]_m = x \bmod 2^m$ for an integer $x \in \mathbb{Z}$ and $[a]_m = \hat{a} \bmod 2^m$ for a ring element $a \in \mathbb{Z}_N$. When $N \geqslant 2^\ell$, the embedding of integers from $[0, 2^\ell - 1]$ into $\mathbb{Z}_N$ is injective so that no overflow occurs. For brevity, we omit the explicit $\mathrm{Rep}(\cdot)$ mapping in ring expressions where the conversion is clear from the context. Once the data are represented as integers, truncation is required to facilitate fixed-point arithmetic, e.g., reversing the scaling effect of 2^f after multiplication. Definition 2.1 details the truncation we use in this work. We first define unsigned truncation over $\mathbb{Z}_N$. Signed inputs are handled in Section 3 and formally derived in Appendix A.

Definition 2.1 (Unsigned Truncation). *Let $a \in \mathbb{Z}_N$ with canonical representative $\hat{a} \in [0, N-1]$. For $1 \leqslant m < \lceil \log_2 N \rceil$, the truncation of a by m bits is defined as* $\mathrm{Trunc}_m(a) = \mathrm{Rep}\left(\left\lfloor \frac{\hat{a}}{2^m} \right\rfloor\right)$.

For the ℓ-bit fixed-point inputs considered in this work, we assume $1 \leqslant m \leqslant \ell - 1$.

The Mask-and-Reveal Paradigm The classic truncation approach (Catrina & de Hoogh, 2010) securely evaluates $\mathrm{Trunc}_m(a)$ using a precomputed random sharing pair $\{[\![r]\!], [\![\mathrm{Trunc}_m(r)]\!]\}$. The parties open the masked value $c = (a + r) \bmod N$ and compute $\mathrm{Trunc}_m(c) - [\![\mathrm{Trunc}_m(r)]\!]$ as the truncation result. The resulting value differs from $\mathrm{Trunc}_m(a)$ by at most one, depending on whether the lower m bits of a and r generated a carry. To eliminate this error and achieve a deterministic result, the protocol requires a secure comparison between $[c]_m$ and $[r]_m$. Mathematically, the mask-and-reveal relation holds only if no wraparound occurs during masking, i.e., $\hat{a} + \hat{r} < N$. To statistically hide the ℓ-bit input, traditional protocols sample $r \in [0, 2^{\ell+\kappa} - 1]$ with a statistical security parameter $\kappa \approx 40$, necessitating a ring size on the order of $2^{\ell+\kappa+1}$. The κ-free exact truncation approach of Escudero et al. (2020) eliminates this dependency by permitting wraparound over tight rings with bit-widths nearly matching the data length ℓ. However, its correctness requires two sequential secure comparisons to detect and correct modular wraparound, with the second depending on the corrected sharing of $[a]_m$ produced by the first, making this dependency the main bottleneck in online latency.

3 Efficient Deterministic Truncation

This section develops the theoretical basis of our deterministic truncation protocol, which eliminates the statistical security parameter κ while requiring only one online secure comparison.

3.1 Wraparound Identification

We first characterize the modular wraparound that arises in the mask-and-reveal procedure. Our analysis relies on a simple relation between the modulus N and the data bound 2^ℓ. Let $k = \lceil \log_2 N \rceil$ denote the bit length of the modulus. We require N to satisfy $2^\ell + 2^{k-1} - 2 < N \leqslant 2^k$. Under this condition, modular wraparound can be identified as follows.

Theorem 3.1 (Wraparound Identification). *Given a ring $\mathbb{Z}_N$, let $k = \lceil \log_2 N \rceil \geqslant 2$ and suppose that $2^\ell + 2^{k-1} - 2 < N \leqslant 2^k$. Let $a \in \mathbb{Z}_N$ satisfy $\hat{a} \in [0, 2^\ell - 1]$ and let $r \in_{\$} \mathbb{Z}_N$. When computing $c = (a + r) \bmod N$, wraparound occurs (i.e., $\hat{a} + \hat{r} \geqslant N$) if and only if $r_{\mathrm{msb}}(1 - c_{\mathrm{msb}}) = 1$.*

The formal proof appears in Appendix A.2. Heuristically, when $r_{\mathrm{msb}} = 0$, the integer sum $\hat{a} + \hat{r}$ is strictly below N due to the restricted range of a. Conversely, when $r_{\mathrm{msb}} = 1$, modular wraparound is uniquely characterized by the resulting value c falling into the lower range of the ring, so $c_{\mathrm{msb}} = 0$.

3.2 Truncation Formula

Using the above characterization, we incorporate modular wraparound directly into the truncation relation, in contrast to prior κ-free constructions that handle it through a separate correction step (Escudero et al., 2020). Let $a \in \mathbb{Z}_N$ satisfy $\hat{a} \in [0, 2^\ell - 1]$, and let $r \in_{\$} \mathbb{Z}_N$. We define the masked value as $c = (a + r) \bmod N$ and, by Theorem 3.1, the wraparound indicator as $s = r_{\mathrm{msb}}(1 - c_{\mathrm{msb}})$. With these definitions, the truncation result can be expressed as follows:

$$\mathrm{Trunc}_m(a) = \underbrace{\frac{[c]_m + N - [\hat{c} + N]_m}{2^m} s}_{\text{Wraparound correction}} + \mathrm{Trunc}_m(c) - \mathrm{Trunc}_m(r) - \delta. \tag{1}$$

The term $\delta \in \{0,1\}$ is an adjustment bit based on the wraparound state s: $\delta = \mathbb{1}([c]_m < [r]_m)$ if $s = 0$, and $\mathbb{1}([\hat{c} + N]_m < [r]_m)$ if $s = 1$, with $\mathbb{1}(\cdot)$ denoting the indicator function that evaluates to 1 if its condition is true and 0 otherwise. We clarify that the division by 2^m in the above expressions is performed over the corresponding integer representatives in $\mathbb{Z}$ before the result is embedded back into the ring. The formal derivation of Eq. 1 is provided in Appendix A.

3.3 Truncation over $\mathbb{Z}_{2^k}$

Eq. 1 applies to any N satisfying the condition of Theorem 3.1. When specialized to $\mathbb{Z}_{2^k}$, the ring underlying our private training framework, the formulation admits several useful algebraic simplifications: (i) the condition $2^\ell + 2^{k-1} - 2 < N \leqslant 2^k$ from Theorem 3.1 reduces to $\ell \leqslant k - 1$, necessitating only a 1-bit gap between data length and ring bit-width; (ii) since $m < k$, the relation $[\hat{c} + 2^k]_m = [c]_m$ allows δ to simplify directly to $\mathbb{1}([c]_m < [r]_m)$, becoming independent of the wraparound state s; and (iii) the correction term $\frac{[c]_m + 2^k - [\hat{c}+2^k]_m}{2^m}$ reduces to 2^{k-m}.

To support signed fixed-point values, we further adopt the standard offset technique used in prior truncation protocols (Escudero et al., 2020). For $a = \mathrm{Rep}(v) \in \mathbb{Z}_{2^k}$ representing a signed integer $v \in [-2^{\ell-1}, 2^{\ell-1} - 1]$, we define its signed truncation as $\mathrm{Trunc}^{\mathrm{s}}_m(a) = \mathrm{Rep}\left(\left\lfloor \frac{v}{2^m} \right\rfloor\right)$. Combining this transformation with the above simplifications yields the following formulation over $\mathbb{Z}_{2^k}$, which eliminates the statistical security parameter κ and requires only one online secure comparison. The proof is deferred to Appendix A.5.

Corollary 3.2 (Signed Truncation over $\mathbb{Z}_{2^k}$)**.** *Let $a = \mathrm{Rep}(v) \in \mathbb{Z}_{2^k}$ represent a signed integer $v \in [-2^{\ell-1}, 2^{\ell-1} - 1]$, and let $r \in_{\$} \mathbb{Z}_{2^k}$, where $\ell \leqslant k - 1$. Let the shifted masked value be $c = (2^{\ell-1} + a + r) \bmod 2^k$, and the wraparound indicator be $s = r_{\mathrm{msb}}(1 - c_{\mathrm{msb}})$. Then,*

$$\mathrm{Trunc}^{\mathrm{s}}_m(a) = 2^{k-m}s + \mathrm{Trunc}_m(c) - \mathrm{Trunc}_m(r) - \mathbb{1}([c]_m < [r]_m) - 2^{\ell-1-m}. \tag{2}$$

4 Main Protocols

In this section, we present our deterministic truncation protocol over $GR(2^k, d)$ and build on it to construct several key fixed-point primitives for RMFE-based private neural network training.

4.1 Mixed-Circuit Primitives

Our construction uses several mixed-circuit functionalities. The RMFE-based packed implementations of randomness generation and domain conversion are formalized in (Zhou, 2026). Here, we briefly review their non-packed counterparts together with the required bitwise comparison: (i) $\mathcal{F}_{\text{edaBit}}(m)$ generates an m-bit edaBit (Escudero et al., 2020), which consists of an arithmetic sharing $[\![r]\!]^{\mathcal{R}}$ of a random m-bit integer r alongside its bit-decomposition $([\![r_m]\!]^{\mathcal{B}}, \ldots, [\![r_1]\!]^{\mathcal{B}})$, satisfying $r = \sum_{i=1}^{m} 2^{i-1} r_i$. (ii) $\mathcal{F}_{\text{B2A}}$ converts a Boolean sharing $[\![b]\!]^{\mathcal{B}}$ into an arithmetic sharing $[\![b]\!]^{\mathcal{R}}$. (iii) $\mathcal{F}_{\text{BitLT}}$ performs secure bitwise comparison. Given m public bits $(a_i)_{i=1}^m$ and m sharings $([\![b_i]\!]^{\mathcal{B}})_{i=1}^m$ where $a_i, b_i \in \{0,1\}$, it returns $[\![\mathbb{1}(\sum_{i=1}^m 2^{i-1} a_i < \sum_{i=1}^m 2^{i-1} b_i)]\!]^{\mathcal{B}}$. The detailed constructions of these primitives are deferred to Appendix B.

4.2 Deterministic Truncation

We first consider deterministic truncation of a single secret over $\mathbb{Z}_{2^k}$ and formalize it via the functionality $\mathcal{F}_{\text{DTrun}}$. In this setting, the secret value a is embedded into the constant subring $\mathbb{Z}_{2^k} \subset GR(2^k, d)$. This alignment ensures that while the Shamir scheme leverages Galois ring to support an arbitrary number of parties n, the underlying truncation logic remains consistent with $\mathbb{Z}_{2^k}$ arithmetic. Given an arithmetically shared input $[\![a]\!]$ representing an ℓ-bit signed integer, i.e., $a = \mathrm{Rep}(v) \in \mathbb{Z}_{2^k}$ for some $v \in [-2^{\ell-1}, 2^{\ell-1} - 1]$, and a target truncation length m where $1 \leqslant m \leqslant \ell - 1$, $\mathcal{F}_{\text{DTrun}}$ computes the truncated output $[\![d]\!]$ satisfying $d = \mathrm{Trunc}^{\mathrm{s}}_m(a)$. The formal definition of $\mathcal{F}_{\text{DTrun}}$ is deferred to Appendix C.1. Protocol 1 shows the details of our protocol Π_{DTrun} for realizing this functionality.

Protocol 1: Deterministic Truncation Π_{DTrun}

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

1. $\{[\![g]\!]^{\mathcal{R}}, ([\![r_i]\!]^{\mathcal{B}})_{i=m+1}^{k}\} = \mathcal{F}_{\text{edaBit}}(k-m)$;
2. $\{[\![h]\!]^{\mathcal{R}}, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^{m}\} = \mathcal{F}_{\text{edaBit}}(m)$;
3. $[\![r_{\text{msb}}]\!]^{\mathcal{R}} = \mathcal{F}_{\text{B2A}}([\![r_k]\!]^{\mathcal{B}})$;
4. $[\![r]\!]^{\mathcal{R}} = 2^m[\![g]\!]^{\mathcal{R}} + [\![h]\!]^{\mathcal{R}}$;
5. $[\![\text{Trunc}_m(r)]\!]^{\mathcal{R}} = [\![g]\!]^{\mathcal{R}}$;

Online phase:

1. $c = \Pi_{\text{Reveal}}(2^{\ell-1} + [\![a]\!]^{\mathcal{R}} + [\![r]\!]^{\mathcal{R}})$;
2. $c_1 = \text{Rep}([c]_m)$;
3. $[\![s]\!]^{\mathcal{R}} = [\![r_{\text{msb}}]\!]^{\mathcal{R}}(1 - c_{\text{msb}})$;
4. The parties locally compute the m-bit decomposition of c_1, $(q_m, \ldots, q_1)$, satisfying $c_1 = \sum_{i=1}^{m} 2^{i-1} q_i$;
5. $[\![\delta]\!]^{\mathcal{B}} = \mathcal{F}_{\text{BitLT}}((q_i)_{i=1}^{m}, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^{m})$;
6. $[\![\delta]\!]^{\mathcal{R}} = \mathcal{F}_{\text{B2A}}([\![\delta]\!]^{\mathcal{B}})$;
7. $[\![d]\!]^{\mathcal{R}} = 2^{k-m}[\![s]\!]^{\mathcal{R}} + \text{Trunc}_m(c) - [\![\text{Trunc}_m(r)]\!]^{\mathcal{R}} - [\![\delta]\!]^{\mathcal{R}} - 2^{\ell-1-m}$;

The correctness of Π_{DTrun} follows from Corollary 3.2. For privacy, the only value revealed during the online phase is the shifted masked value c. Since the mask r is sampled uniformly at random from $\mathbb{Z}_{2^k}$, c perfectly blinds the input a, ensuring zero information leakage. We formalize this perfect security guarantee in the following Theorem 4.1, with the full proof deferred to Appendix C.2.

Theorem 4.1. Π_{DTrun} *securely realizes* $\mathcal{F}_{\text{DTrun}}$ *in the* $(\mathcal{F}_{\text{edaBit}}, \mathcal{F}_{\text{B2A}}, \mathcal{F}_{\text{BitLT}})$*-hybrid model.*

Theoretical Comparison We compare Π_{DTrun} with two representative deterministic truncation approaches over $\mathbb{Z}_{2^k}$, namely the RMFE-based protocol Π_{RMFE} of Zhou (2026), which requires the statistical security parameter κ but only one online secure comparison, and the κ-free protocol Π_{edaBit} of Escudero et al. (2020), instantiated with Shamir sharing over Galois rings, which operates over a tight ring but requires two sequential secure-comparison invocations. For consistency, we consider the single-secret setting, as all three protocols are compatible with SIMD-style packing. Table 1 highlights two main factors that affect the efficiency of these protocols. The first is the reliance on the statistical security parameter κ, which increases the required ring bit-width and consequently the computation and communication costs of ring operations. The second is the number of sequential secure comparisons in the online phase. Although private truncation also involves other communication steps, such as secret reconstruction, secure comparisons constitute the main source of online latency due to their interactive nature.

Table 1: Qualitative comparison of deterministic truncation over $\mathbb{Z}_{2^k}$.

Protocol	κ-Free	Sequential Comp.[†]
Π_{RMFE} (Zhou, 2026)	✗	1
Π_{edaBit} (Escudero et al., 2020)	✓	2
Π_{DTrun} (Ours)	✓	1

[†] Number of sequential invocations to secure comparison.

As shown in Table 1, Π_{DTrun} simultaneously avoids the statistical security parameter and requires only one online secure-comparison invocation. Compared with Π_{RMFE}, eliminating κ reduces the required ring size and consequently lowers the computation and communication costs of the underlying ring operations. Compared with Π_{edaBit}, our protocol removes the sequential dependency between two secure comparisons and reduces the comparison depth of the online phase.

4.3 RMFE-BASED PROTOCOLS FOR PRIVATE TRAINING

We next extend our truncation mechanism to the RMFE-based training framework of Zhou (2026), where ρ fixed-point values are packed into a single Galois ring element for SIMD-style computation. In the original framework, the underlying truncation primitive relies on a statistical security parameter κ, requiring the ring bit-width used throughout training to be enlarged accordingly. Building on our κ-free truncation, we develop new RMFE-based components that remove this dependence on κ and allow the resulting training framework to operate with a tight ring bit-width.

Packed Deterministic Truncation Our protocol Π_{DTrun} naturally extends to the RMFE-based framework to truncate ρ fixed-point values in SIMD form. The packed construction remains κ-free and requires only one online comparison. The formal construction is provided in Appendix C.3.

Fixed-Point Multiplication In Shamir-based MPC, local multiplication of two degree-t sharings yields a degree-$2t$ sharing. Applying truncation directly to this intermediate optimizes fixed-point multiplication by avoiding a separate degree reduction step and reducing communication overhead (Catrina, 2018). In the RMFE-based packing setting, however, this optimization is nontrivial because the degree-$2t$ intermediate is generally not a valid packed representation, and a mask restricted to valid packed elements cannot securely hide an arbitrary Galois-ring intermediate. We therefore propose $\Pi_{\text{MulRescale}}$, which combines packed truncation masks with additional randomness over the full Galois ring. This construction uniformly masks the degree-$2t$ intermediate while preserving the component-wise truncation masks after unpacking, enabling direct truncation of the intermediate into a valid degree-t packed sharing without a separate degree reduction step. The complete construction is provided in Appendix C.4.

Gradient Aggregation Besides truncation itself, gradient aggregation is another component of the RMFE-based framework (Zhou, 2026) that relies on κ. Formally, given a Galois ring element a obtained by packing the ρ gradient values $(a_0, \ldots, a_{\rho-1})$, gradient aggregation first computes $\sum_{i=0}^{\rho-1} a_i$ and then divides the sum by the public mini-batch size to obtain the average. Following the aggregation strategy of Zhou (2026), this process can be formulated as a specialized form of truncation. Specifically, a single truncation mask r is decomposed into ρ values $(v_0, \ldots, v_{\rho-1})$ whose sum equals r, and these values are packed and added to the packed gradient. After revealing the masked packed element, the parties recover and sum the corresponding ρ masked values, obtaining a masked representation of the aggregate to which the truncation procedure can be applied. Consequently, the aggregation procedure inherits the same dependence on κ as the underlying truncation primitive. Building on our κ-free truncation, we adapt this construction accordingly and propose $\Pi_{\text{AggRescale}}$ to perform our gradient aggregation. The complete construction is provided in Appendix C.5.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate our deterministic truncation protocol and the resulting RMFE-based private neural network training framework integrating the protocols described in Section 4. All experiments were conducted on a server equipped with an AMD EPYC 9754 128-core processor and 512 GB of RAM. We consider both LAN and WAN settings. In the LAN setting, the parties communicate over a shared network connection with an average bandwidth of 4 Gbps and sub-millisecond round-trip time (RTT). In the WAN setting, we use Linux Traffic Control to emulate an RTT of 40 ms and a bandwidth of 100 MB/s. For baselines that require a statistical security parameter, we set $\kappa = 40$. We adopt the (ρ, d)-RMFE formulation of Zhou (2026) to pack ρ values into a single element of the Galois ring $GR(2^k, d)$, and follow the offline-online paradigm in which all input-independent random sharings are generated during the offline phase.

5.1 PERFORMANCE EVALUATION FOR PRIVATE TRUNCATION

To evaluate the efficiency of Π_{DTrun}, we compare it against Π_{RMFE} (Zhou, 2026) and Π_{edaBit} (Escudero et al., 2020). Specifically, we measure the execution time and communication overhead for $10,000$ parallel 13-bit truncations on 60-bit fixed-point data among 3 parties. All protocols are instantiated over $GR(2^k, 3)$ for arithmetic sharing, where the ring bit-width k varies depending on whether the protocol requires a statistical security parameter. Table 2 summarizes the results.

Table 2: Performance comparison for 10,000 parallel deterministic truncations. ρ and k denote the number of packed secrets and ring bit-width, respectively. Communication is reported as the amortized communication per party for a single truncation.

Protocol	ρ	k	Offline Phase			Online Phase		
			LAN (s)	WAN (s)	Comm. (KB)	LAN (ms)	WAN (s)	Comm. (KB)
Π_{edaBit}	1	64	0.63	17.64	14.16	64.40	5.29	0.69
Π_{RMFE}	1	128	0.85	26.51	34.27	17.83	1.30	0.20
	2	128	0.69	23.80	23.98	13.52	1.23	0.11
Π_{DTrun}	1	64	0.55	17.77	12.66	15.63	1.21	0.17
(Ours)	**2**	**64**	**0.44**	**17.05**	**9.26**	**10.20**	**1.18**	**0.08**

The results in Table 2 are consistent with the theoretical comparison in Table 1. Compared with Π_{RMFE}, eliminating κ allows us to replace oversized 128-bit rings with tight 64-bit rings, reducing offline communication by 63%. Although Π_{edaBit} also uses 64-bit rings, it requires two sequential invocations to secure comparison. In contrast, Π_{DTrun} requires only one comparison invocation, resulting in substantially lower online latency.

5.2 Application to Neural Network Training

We integrate our deterministic truncation protocol and the RMFE-native fixed-point operators into the training framework of Zhou (2026), while retaining the same neural network architectures and training configurations for a fair comparison.

Networks We perform private training on the MNIST dataset (Deng, 2012) using three representative networks established in prior literature (Wagh et al., 2019; Zhou, 2026): (i) Network A: a 3-layer Fully Connected Neural Network (FCN) with 128 neurons per hidden layer (Mohassel & Zhang, 2017); (ii) Network B: a Convolutional Neural Network (CNN) with two convolutional and two fully connected layers (Liu et al., 2017); (iii) Network C: the 4-layer LeNet-5 Caffe CNN architecture (LeCun et al., 1998). All three networks use ReLU activations. Networks B and C use max pooling in their convolutional layers.

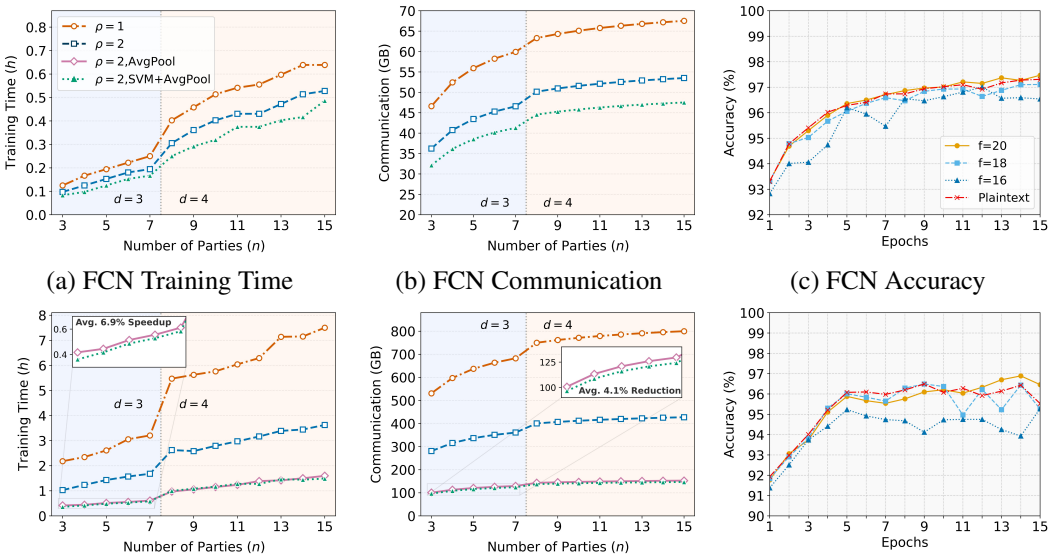
Performance Comparison We evaluate our framework in a 3-party setting using a consistent packing density of $\rho = 2$ and leverage the Eigen library to accelerate local matrix multiplications. We set the ring bit-width k to 64 and 128 for our work and (Zhou, 2026), respectively. For the fixed-point configuration, we set the bit-length bound ℓ to 60 and the precision f to 20. All network training sessions operate with a mini-batch size of 128 and a learning rate η of 2^{-8}. As detailed in Table 3, our integrated framework consistently outperforms the baseline across all 3 networks, achieving up to a $1.70\times$ speedup in the LAN setting and reducing communication by up to $1.26\times$.

Table 3: The time and amortized communication required for each party for an epoch of training. Baseline denotes the RMFE-based training framework of Zhou (2026).

Network	LAN Time (h)			WAN Time (h)			Communication (GB)		
	Ours	Baseline	Factor	Ours	Baseline	Factor	Ours	Baseline	Factor
A	0.11	0.15	$1.38\times$	19.95	20.10	$1.01\times$	36.23	42.20	$1.16\times$
B	0.96	1.29	$1.35\times$	29.03	31.34	$1.08\times$	280.79	354.08	$1.26\times$
C	1.68	2.86	$1.70\times$	32.04	35.83	$1.12\times$	486.94	605.98	$1.24\times$

Scalability and Accuracy We further evaluate the scalability of our framework on FCN (Network A) and CNN (Network B) as the number of parties n increases from 3 to 15. Figures 1(a,b,d,e) report the training time and communication overhead, where the number of semi-honest adversaries

is set to the maximum value $t = \lfloor (n-1)/2 \rfloor$. Across both networks, our framework exhibits near-linear scalability as the number of participants increases. The structural increase from $n = 7$ to $n = 8$ results from increasing the Galois ring extension degree from $d = 3$ to $d = 4$ to satisfy the constraint $n < 2^d$. For the CNN, we additionally evaluate average pooling in place of max pooling to avoid costly secure comparisons. We also explore the multiclass SVM loss (Weston & Watkins, 1999; Zhou, 2023) as an MPC-friendly alternative to cross-entropy, which avoids secure division in the output layer. As shown in Figures 1(a,b,d,e), these training-level choices provide additional efficiency improvements, with the benefit of the SVM loss being more pronounced for the FCN. For Network B, the SVM-based configuration further reduces training time and communication by an average of 6.9% and 4.1%, respectively, for $n = 3$ to 7. Implementation details of the SVM loss are provided in Appendix D.1.



(a) FCN Training Time (b) FCN Communication (c) FCN Accuracy

(d) CNN Training Time (e) CNN Communication (f) CNN Accuracy

Figure 1: End-to-end private training performance on FCN (a-c) and CNN (d-f) architectures. Subfigures (a, b, d, e) report training time and communication for one epoch as the number of parties increases from 3 to 15, where cross-entropy and max pooling are used as the default settings, with pooling choices relevant only to CNN. The shaded regions mark the transition of the underlying Galois ring extension degree from $d = 3$ to $d = 4$. Subfigures (c, f) compare private fixed-point training with plaintext training across different fractional precisions.

We further evaluate the training accuracy of the SVM-based configuration under different fractional precisions. The same multiclass SVM loss is used for both private and plaintext training so that the comparison focuses on the numerical effect of fixed-point secure computation. Figures 1(c,f) compare the corresponding training accuracy under different values of f with plaintext training. With $f = 20$, private training reaches accuracies of 97.47% and 96.46% after 15 epochs for Networks A and B, respectively. The gap between private fixed-point and plaintext training decreases as f increases, indicating that sufficient fractional precision is important for stable convergence, consistent with previous observations (Zhou, 2026; Zhou et al., 2025).

6 Conclusion

We presented a perfectly secure deterministic truncation protocol for general n-party Shamir sharing over Galois rings in the semi-honest, honest-majority setting. By incorporating modular wraparound correction directly into the truncation relation, our construction eliminates the statistical security parameter κ and requires only one online secure comparison over Galois rings of characteristic 2^k. We further extended the truncation mechanism to RMFE-based packing and developed new protocols for efficient fixed-point multiplication and gradient aggregation. Experiments demonstrate reduced truncation costs and improved efficiency in end-to-end private neural network training.

AI USE STATEMENT

In this work, we used generative AI tools solely for language polishing, including improving grammar, clarity, and writing style of text originally written by the authors. We did not use generative AI tools to generate research ideas, technical claims, proofs, experimental results, code, data analysis, or literature review. All AI-assisted edits were carefully reviewed by the authors to ensure that they did not alter the technical meaning or introduce unsupported claims. We take responsibility for the final content of this work, including text produced with the aid of generative AI.

REFERENCES

Najwa Aaraj, Abdelrahaman Aly, Tim Güneysu, Chiara Marcolla, Johannes Mono, Rogerio Paludo, Iván Santos-González, Mireia Scholz, Eduardo Soria-Vazquez, Victor Sucasas, and Ajith Suresh. FANNG-MPC: Framework for artificial neural networks and generic MPC. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025:1–36, 2024.

Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via Galois rings. In *Proceedings of the Theory of Cryptography Conference*, pp. 471–501, 2019.

Alessandro N. Baccarini, Marina Blanton, and Chen Yuan. Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning. *Proceedings on Privacy Enhancing Technologies*, 2023(1):608–626, 2023.

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

Ran Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 136–145, 2001.

Octavian Catrina. Round-efficient protocols for secure multiparty fixed-point arithmetic. In *2018 International Conference on Communications (COMM)*, pp. 431–436, 2018.

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

Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, pp. 35–50, 2010.

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

Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *Advances in Cryptology – CRYPTO 2018*, pp. 769–798, 2018.

Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355 – 375, 2020.

Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sára Zakariás. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, pp. 643–662, 2012.

Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, pp. 572–590, 2007.

Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Symposium on Network and Distributed System Security (NDSS)*, 2015.

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

Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *Advances in Cryptology – CRYPTO 2020*, pp. 823–852, 2020.

Christopher Harth-Kitzerow, Ajith Suresh, and Georg Carle. Sok: Truncation untangled: Scaling fixed-point arithmetic for privacy-preserving machine learning to large models and datasets. *Proceedings on Privacy Enhancing Technologies*, 2025(4):369–391, 2025.

Marcel Keller and Ke Sun. Secure quantized training for deep learning. In *ICML*, 2022.

Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively secure 4PC for secure training and inference. In *Symposium on Network and Distributed System Security (NDSS)*, 2022.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3PC for binary circuits with application to maliciously-secure DNN inference. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, 2023.

Fengrun Liu, Xie Xiang, and Yu Yu. Scalable multi-party computation protocols for machine learning in the honest-majority setting. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, 2024.

Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 619–631, 2017.

Yunlv Lv, Rui Zhang, Zhiyuan Zhang, Ziyi Wan, Lanxue Zhang, Minhui Xue, Jiangtao Li, and Yanan Cao. SMASH: Scalable maliciously secure hybrid multi-party computation framework for Privacy-Preserving large language models. In *Proceedings of the 35th USENIX Conference on Security Symposium*, pp. 1927–1946, 2026.

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

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

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

Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In *Progress in Cryptology – INDOCRYPT 2019*, pp. 227–249, 2019.

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

Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021(1):188–208, 2021.

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

Jason Weston and Christopher Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the 7th European Symposium on Artificial Neural Networks (ESANN)*, pp. 219–224, 1999.

Andrew Chi-Chih Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pp. 160–164, 1982.

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

Martin Zbudila, Erik Pohle, Aysajan Abidin, and Bart Preneel. Master: Maliciously secure truncation for replicated secret sharing without pre-processing. *Cryptology ePrint Archive*, 2024.

Hengcheng Zhou. Information-theoretically secure neural network training with flexible deployment. In *Artificial Neural Networks and Machine Learning – ICANN 2023*, pp. 324–336, 2023.

Hengcheng Zhou. Scalable privacy-preserving neural network training over $\mathbb{Z}_{2^k}$ via RMFE-based packing and mixed-circuit computation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(34):28964–28972, Mar. 2026.

Lijing Zhou, Bingsheng Zhang, Ziyu Wang, Tianpei Lu, Qingrui Song, Su Zhang, Hongrui Cui, and Yu Yu. On probabilistic truncation in privacy-preserving machine learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(21):22955–22964, Apr. 2025.

Huan Zou, Yuting Xiao, and Rui Zhang. Semi-honest 2-party faithful truncation from two-bit extraction. In *Information and Communications Security*, pp. 215–234, 2023.

A PROOFS FOR THE TRUNCATION FORMULATION

This section details the formal proofs for the integrated wraparound identification and correction mechanism introduced in Section 3. We formulate our analysis over the general ring $\mathbb{Z}_N$ assuming N to be an arbitrary integer.

A.1 TRUNCATION WITH WRAPAROUND

The truncation in (Catrina & de Hoogh, 2010) requires that wraparound cannot occur. For ease of demonstration, we summarize the following Lemma A.1 to handle this case:

Lemma A.1 (Truncation without Wraparound). *In $\mathbb{Z}_N$ where $N \geqslant 2^\ell$, let $a \in \mathbb{Z}_N$ where $\hat{a} \in [0, 2^\ell - 1]$ and $r \in_\$ \mathbb{Z}_N$. If wraparound does not happen when computing $c = a + r$, then we have*

$$\mathrm{Trunc}_m(a) = \mathrm{Trunc}_m(c) - \mathrm{Trunc}_m(r) - \mathbb{1}([c]_m < [r]_m). \tag{3}$$

Lemma A.1 shows how to compute the truncation when no wraparound occurs. Its correctness follows from the standard truncation analysis of Catrina & de Hoogh (2010). We now show that even if wraparound occurs, truncation can still be calculated. We first establish the corresponding integer relation.

Proposition A.2 (Integer Truncation with Wraparound). *Let $a \in [0, 2^\ell - 1]$ and $r \in_\$ [0, N-1]$, where $N \geqslant 2^\ell$. If $a + r \geqslant N$, we have*

$$\left\lfloor \frac{a}{2^m} \right\rfloor = \frac{[c]_m + N - [c+N]_m}{2^m} + \left\lfloor \frac{c}{2^m} \right\rfloor - \left\lfloor \frac{r}{2^m} \right\rfloor - \mathbb{1}([c+N]_m < [r]_m), \tag{4}$$

where $c = a + r - N \in \mathbb{Z}$.

Proof. Since $c = a+r-N$, we have $c+N = a+r$, and $[c+N]_m = [a+r]_m = (a+r) \bmod 2^m = [a]_m + [r]_m - \delta \cdot 2^m$, where $\delta \in \{0, 1\}$. It is easy to see that $\delta = 1$ if and only if $[a+r]_m < [r]_m$, i.e., $[c+N]_m < [r]_m$. Therefore, $\mathbb{1}([c+N]_m < [r]_m) = \delta$. We have

$$\begin{aligned}
&\frac{[c]_m + N - [c+N]_m}{2^m} + \left\lfloor \frac{c}{2^m} \right\rfloor - \left\lfloor \frac{r}{2^m} \right\rfloor - \mathbb{1}([c+N]_m < [r]_m) \\
=&\frac{[c]_m + N - [c+N]_m}{2^m} + \frac{c - [c]_m}{2^m} - \left\lfloor \frac{r}{2^m} \right\rfloor - \delta \\
=&\frac{c + N - [c+N]_m}{2^m} - \left\lfloor \frac{r}{2^m} \right\rfloor - \delta \\
=&\frac{a + r - ([a]_m + [r]_m - \delta \cdot 2^m)}{2^m} - \left\lfloor \frac{r}{2^m} \right\rfloor - \delta \\
=&\frac{a - [a]_m}{2^m} + \frac{r - [r]_m}{2^m} + \delta - \left\lfloor \frac{r}{2^m} \right\rfloor - \delta \\
=&\left\lfloor \frac{a}{2^m} \right\rfloor + \left\lfloor \frac{r}{2^m} \right\rfloor + \delta - \left\lfloor \frac{r}{2^m} \right\rfloor - \delta \\
=&\left\lfloor \frac{a}{2^m} \right\rfloor.
\end{aligned} \tag{5}$$

□

We next lift Proposition A.2 to truncation over $\mathbb{Z}_N$. Note that the computation of $\hat{c} + N$ below is performed over $\mathbb{Z}$ rather than $\mathbb{Z}_N$.

Lemma A.3 (Truncation with Wraparound over $\mathbb{Z}_N$). *In $\mathbb{Z}_N$ where $N \geqslant 2^\ell$, let $a \in \mathbb{Z}_N$ where $\hat{a} \in [0, 2^\ell - 1]$ and $r \in_\$ \mathbb{Z}_N$. If wraparound happens when computing $c = a + r$, then we have*

$$\mathrm{Trunc}_m(a) = \frac{[c]_m + N - [\hat{c} + N]_m}{2^m} + \mathrm{Trunc}_m(c) - \mathrm{Trunc}_m(r) - \mathbb{1}([\hat{c} + N]_m < [r]_m). \tag{6}$$

Proof. When wraparound occurs, the canonical integer representatives satisfy $\hat{c} = \hat{a} + \hat{r} - N$. Since $\hat{a} \in [0, 2^\ell - 1]$, $\hat{r} \in [0, N-1]$, and $N \geqslant 2^\ell$, the integers $\hat{a}$, $\hat{r}$, and $\hat{c}$ satisfy the conditions of Proposition A.2. Applying Proposition A.2 to these representatives and using the definition of $\mathrm{Trunc}_m(\cdot)$ directly yields the stated relation over $\mathbb{Z}_N$. □

A.2 Proof of Theorem 3.1 (Wraparound Identification)

Proof. We consider the following cases:

1. If $r_{\mathrm{msb}} = 0$, then $\hat{r} \in [0, 2^{k-1}-1]$. Therefore, $\hat{a}+\hat{r} \leqslant 2^{\ell}-1+2^{k-1}-1 < N$, so no wraparound occurs.

2. If $r_{\mathrm{msb}} = 1$, wraparound may or may not occur. Let $\hat{c} = \hat{a}+\hat{r}-sN$, where $s = 1$ if wraparound occurs and $s = 0$ otherwise. We consider the two cases:

   (a) If no wraparound occurs, then $\hat{c} = \hat{a}+\hat{r} \geqslant \hat{r} \geqslant 2^{k-1}$. Since $\hat{c} < N \leqslant 2^k$, it follows that $c_{\mathrm{msb}} = 1$.

   (b) If wraparound occurs, then $\hat{c} = \hat{a}+\hat{r}-N < \hat{a}$ because $\hat{r} < N$. Moreover, the condition $2^{\ell}+2^{k-1}-2 < N \leqslant 2^k$ implies $\ell \leqslant k-1$. Hence, $\hat{c} < \hat{a} \leqslant 2^{\ell}-1 < 2^{k-1}$, which gives $c_{\mathrm{msb}} = 0$.

Therefore, wraparound occurs if and only if $r_{\mathrm{msb}} = 1$ and $c_{\mathrm{msb}} = 0$, i.e., $r_{\mathrm{msb}}(1-c_{\mathrm{msb}}) = 1$. □

A.3 Truncation over $\mathbb{Z}_N$

Based on Lemma A.1, Lemma A.3, and Theorem 3.1, we prove Proposition A.4, which establishes the general truncation formulation presented in Eq. 1 of the main text for $\mathbb{Z}_N$ when $2^{\ell}+2^{k-1}-2 < N \leqslant 2^k$.

Proposition A.4 (Unsigned Truncation over $\mathbb{Z}_N$)**.** *Let $a, r \in \mathbb{Z}_N$ with $\hat{a} \in [0, 2^{\ell}-1]$, where $2^{\ell}+2^{k-1}-2 < N \leqslant 2^k$. Let $c = a+r$, $s = r_{\mathrm{msb}}(1-c_{\mathrm{msb}})$, $c_1 = [c]_m$, and $c_2 = [\hat{c}+N]_m$. We have*

$$\mathrm{Trunc}_m(a) = \frac{c_1+N-c_2}{2^m}s + \frac{c-c_1}{2^m} - \mathrm{Trunc}_m(r) - \delta, \tag{7}$$

where $\delta = \mathbb{1}(c_1 < [r]_m)$ if $s = 0$ and $\delta = \mathbb{1}(c_2 < [r]_m)$ if $s = 1$.

Proof. Since $s = r_{\mathrm{msb}}(1-c_{\mathrm{msb}})$, according to Theorem 3.1 wraparound happens if and only if $s = 1$. We have the following cases:

Case 1: $s = 0$, then there will be no wraparound when computing $a+r$, i.e., $\hat{c} = \hat{a}+\hat{r}$. We have $[c]_m = [a]_m + [r]_m - v_1 \cdot 2^m$ and $\delta = \mathbb{1}(c_1 < [r]_m) = \mathbb{1}([c]_m < [r]_m)$. It is easy to see that $v_1 = 1$ if and only if $[c]_m < [r]_m$, we have $v_1 = \mathbb{1}([c]_m < [r]_m) = \delta$. Then

$$\begin{aligned}
&\frac{c_1+N-c_2}{2^m}s + \frac{c-c_1}{2^m} - \mathrm{Trunc}_m(r) - \delta \\
=&\frac{c-c_1}{2^m} - \mathrm{Trunc}_m(r) - \delta \\
=&\,\mathrm{Trunc}_m(c) - \mathrm{Trunc}_m(r) - \mathbb{1}([c]_m < [r]_m) \\
=&\,\mathrm{Trunc}_m(a)\,. \quad \text{(by Lemma A.1)}
\end{aligned} \tag{8}$$

Case 2: $s = 1$, then $\hat{c} = \hat{a}+\hat{r}-N$. We have $\delta = \mathbb{1}([\hat{c}+N]_m < [r]_m)$ and $\hat{c}+N = \hat{a}+\hat{r}$. Therefore, $[\hat{c}+N]_m = [\hat{a}+\hat{r}]_m = [a]_m + [r]_m - v_2 \cdot 2^m$, and $v_2 = \mathbb{1}([\hat{c}+N]_m < [r]_m) = \delta$. We have

$$\begin{aligned}
&\frac{c_1+N-c_2}{2^m}s + \frac{c-c_1}{2^m} - \mathrm{Trunc}_m(r) - \delta \\
=&\frac{c_1+N-c_2}{2^m} + \frac{c-c_1}{2^m} - \mathrm{Trunc}_m(r) - \mathbb{1}(c_2 < [r]_m) \\
=&\frac{c_1+N-c_2}{2^m} + \mathrm{Trunc}_m(c) - \mathrm{Trunc}_m(r) - \mathbb{1}(c_2 < [r]_m) \\
=&\,\mathrm{Trunc}_m(a)\,. \quad \text{(by Lemma A.3)}
\end{aligned} \tag{9}$$

□

A.4 Extension to Negative Fixed-Point Numbers

The truncation mechanism presented in Proposition A.4 is natively defined for non-negative inputs. To accommodate signed fixed-point values, we apply the standard offset technique of Escudero et al. (2020). Following the signed truncation notation introduced in Section 3.3, let $a = \mathrm{Rep}(v) \in \mathbb{Z}_N$ represent a signed integer $v \in [-2^{\ell-1}, 2^{\ell-1} - 1]$. We first shift the represented integer into the non-negative range:

$$a_{\text{shifted}} = a + 2^{\ell-1} = \mathrm{Rep}\left(v + 2^{\ell-1}\right). \tag{10}$$

Its canonical integer representative therefore satisfies $\hat{a}_{\text{shifted}} = v + 2^{\ell-1} \in [0, 2^{\ell} - 1]$. Applying unsigned truncation to a_{shifted} and locally subtracting the rescaled offset gives

$$\mathrm{Trunc}^{\mathrm{s}}_m(a) = \mathrm{Trunc}_m(a_{\text{shifted}}) - 2^{\ell-m-1}. \tag{11}$$

This transformation inherently ensures floor rounding, i.e., rounding towards negative infinity, which is consistent with standard fixed-point arithmetic rules and requires no additional communication. The complete procedure is formalized in Lemma A.5.

Lemma A.5 (Signed Truncation over $\mathbb{Z}_N$)**.** *Let $a = \mathrm{Rep}(v) \in \mathbb{Z}_N$ represent a signed integer $v \in [-2^{\ell-1}, 2^{\ell-1} - 1]$, and $r \in_{\$} \mathbb{Z}_N$, where $2^{\ell} + 2^{k-1} - 2 < N \leqslant 2^k$. Let $c = 2^{\ell-1} + a + r$, $s = r_{\mathrm{msb}}(1 - c_{\mathrm{msb}})$, $c_1 = [c]_m$, and $c_2 = [\hat{c} + N]_m$. We have*

$$\mathrm{Trunc}^{\mathrm{s}}_m(a) = \frac{c_1 + N - c_2}{2^m} s + \frac{c - c_1}{2^m} - \mathrm{Trunc}_m(r) - \delta - 2^{\ell-m-1}, \tag{12}$$

where $\delta = \mathbb{1}(c_1 < [r]_m)$ if $s = 0$ and $\delta = \mathbb{1}(c_2 < [r]_m)$ if $s = 1$.

A.5 Proof of Corollary 3.2 (Signed Truncation over $\mathbb{Z}_{2^k}$)

Proof. We specialize Lemma A.5 to $N = 2^k$. Since $m < k$, we have $[\hat{c} + 2^k]_m = [c]_m$. Letting $c_1 = [c]_m$ and $c_2 = [\hat{c} + 2^k]_m$, it follows that $c_2 = c_1$, and the wraparound correction in Lemma A.5 simplifies to $\frac{c_1 + 2^k - c_2}{2^m} s = 2^{k-m} s$. Moreover, the two cases defining δ coincide, yielding $\delta = \mathbb{1}([c]_m < [r]_m)$. Finally, by the definition of unsigned truncation, $\frac{c - [c]_m}{2^m} = \mathrm{Trunc}_m(c)$. Substituting these identities into Lemma A.5 yields

$$\mathrm{Trunc}^{\mathrm{s}}_m(a) = 2^{k-m} s + \mathrm{Trunc}_m(c) - \mathrm{Trunc}_m(r) - \mathbb{1}([c]_m < [r]_m) - 2^{\ell-1-m}, \tag{13}$$

where $c = (2^{\ell-1} + a + r) \bmod 2^k$ and $s = r_{\mathrm{msb}}(1 - c_{\mathrm{msb}})$. This yields Corollary 3.2. □

B Mixed-Circuit Primitives

The mixed-circuit approach improves MPC performance by combining computation over different algebraic domains (Demmler et al., 2015; Mohassel & Rindal, 2018; Escudero et al., 2020). Our construction uses arithmetic sharings over $GR(2^k, d)$ and Boolean sharings over $GR(2, d)$. This section formalizes the mixed-circuit primitives used by our deterministic truncation and RMFE-based protocols. We use the superscripts $\mathcal{R}$ and $\mathcal{B}$ to denote the arithmetic and Boolean settings, respectively.

B.1 Randomness Generation: daBits and edaBits

In mixed-circuit computation, daBits (Rotaru & Wood, 2019) and edaBits (Escudero et al., 2020) bridge the arithmetic and Boolean domains. We formalize their generation for Shamir sharing over Galois rings through the following two functionalities. (i) For daBits, $\mathcal{F}^{\mathcal{R}}_{\text{daBit}}$ generates a pair of random sharings $\{[\![r]\!]^{\mathcal{B}}, [\![r]\!]^{\mathcal{R}}\}$, where $r \in \{0, 1\}$ is a random bit. It can be realized by having the parties locally generate random sharings and securely XOR them. Arithmetic XOR can be implemented using $\Pi^{\mathcal{R}}_{\text{Mult}}$, resulting in $\lceil \log n \rceil$ rounds and $n-1$ invocations to $\Pi^{\mathcal{R}}_{\text{Mult}}$. (ii) For edaBits, given $m \leqslant k$, $\mathcal{F}^{\mathcal{R}}_{\text{edaBit}}$ generates $\{[\![r]\!]^{\mathcal{R}}, ([\![r_m]\!]^{\mathcal{B}}, \ldots, [\![r_1]\!]^{\mathcal{B}})\}$, where $r \in [0, 2^m - 1]$ is uniformly random and $(r_m, \ldots, r_1)$ is its bit decomposition, satisfying $r = \sum_{i=1}^{m} 2^{i-1} r_i$. It can be realized using Π_{edaBits} from (Escudero et al., 2020), requiring fewer than $(m + \lceil \log n \rceil)(n - 1)$ invocations to $\Pi^{\mathcal{B}}_{\text{Mult}}$ in $\lceil \log n \rceil (m + \lceil \log n \rceil)$ rounds.

B.2 Domain Conversion

Let $\mathcal{F}_{\text{B2A}}^{\mathcal{R}}$ denote the functionality that converts a Boolean sharing $[\![\cdot]\!]^{\mathcal{B}}$ into an arithmetic sharing $[\![\cdot]\!]^{\mathcal{R}}$. It can be realized using a daBit (Escudero et al., 2020). Protocol 2 describes the corresponding protocol $\Pi_{\text{B2A}}^{\mathcal{R}}$, where $\oplus$ denotes addition over $GR(2,d)$.

Protocol 2: $\Pi_{\text{B2A}}^{\mathcal{R}}$

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

1. The parties invoke $\mathcal{F}_{\text{daBit}}^{\mathcal{R}}$ to obtain $\{[\![r]\!]^{\mathcal{B}}, [\![r]\!]^{\mathcal{R}}\}$;
2. $c = \Pi_{\text{Reveal}}^{\mathcal{B}}\left([\![a]\!]^{\mathcal{B}} \oplus [\![r]\!]^{\mathcal{B}}\right)$;
3. $[\![b]\!]^{\mathcal{R}} = c + [\![r]\!]^{\mathcal{R}} - 2c[\![r]\!]^{\mathcal{R}}$.

B.3 Bitwise Comparison

We now introduce our protocol for bitwise comparison over Shamir sharings over $GR(2,d)$. We denote the corresponding functionality by $\mathcal{F}_{\text{BitLT}}$. Given m public bits $(a_i)_{i=1}^m$ and m sharings $([\![b_i]\!]^{\mathcal{B}})_{i=1}^m$ where $a_i, b_i \in \{0,1\}$, $\mathcal{F}_{\text{BitLT}}$ returns $[\![\mathbb{1}(\sum_{i=1}^m 2^{i-1}a_i < \sum_{i=1}^m 2^{i-1}b_i)]\!]^{\mathcal{B}}$. We propose Π_{BitLT} to realize $\mathcal{F}_{\text{BitLT}}$ based on BitLTC1 in (Catrina & de Hoogh, 2010). Protocol 3 shows the details of Π_{BitLT}.

Protocol 3: Π_{BitLT}

Input: $(a_i)_{i=1}^m$, $([\![b_i]\!]^{\mathcal{B}})_{i=1}^m$;
Output: $[\![u]\!]^{\mathcal{B}}$;

1. **for** $i = 1$ to m **do**
2. $\quad [\![d_i]\!]^{\mathcal{B}} = a_i \oplus [\![b_i]\!]^{\mathcal{B}} \oplus 1$;
3. **end for**
4. $([\![p_m]\!]^{\mathcal{B}}, \cdots, [\![p_1]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{PreMult}}([\![d_m]\!]^{\mathcal{B}}, \cdots, [\![d_1]\!]^{\mathcal{B}})$;
5. **for** $i = 1$ to $m-1$ **do**
6. $\quad [\![s_i]\!]^{\mathcal{B}} = [\![p_i]\!]^{\mathcal{B}} \oplus [\![p_{i+1}]\!]^{\mathcal{B}}$;
7. **end for**
8. $[\![s_m]\!]^{\mathcal{B}} = [\![p_m]\!]^{\mathcal{B}} \oplus 1$;
9. $[\![u]\!]^{\mathcal{B}} = \oplus_{i=1}^m \left([\![s_i]\!]^{\mathcal{B}}(1 \oplus a_i)\right)$;

The calculation steps of Π_{BitLT} are the same as BitLTC1. The security and correctness of Π_{BitLT} directly follow from those of BitLTC1. Since Π_{BitLT} deals with binary sharings, we can omit the calculation of the least significant bit in the last step of BitLTC1. In Step 4, $\mathcal{F}_{\text{PreMult}}$ is the functionality for calculating the prefix multiplication. Given $\{[\![g_1]\!]^{\mathcal{B}}, \ldots, [\![g_l]\!]^{\mathcal{B}}\}$, $\mathcal{F}_{\text{PreMult}}$ calculates $\{[\![h_1]\!]^{\mathcal{B}}, \ldots, [\![h_l]\!]^{\mathcal{B}}\}$ with each $[\![h_j]\!]^{\mathcal{B}} = \prod_{i=1}^j [\![g_i]\!]^{\mathcal{B}}$ for $j \in [l]$. This functionality can be implemented by invoking $\Pi_{\text{Mult}}^{\mathcal{B}}$ sequentially, resulting in $l-1$ invocations to $\Pi_{\text{Mult}}^{\mathcal{B}}$ in $l-1$ rounds.

C Deferred Constructions over $\mathbb{Z}_{2^k}$

C.1 Functionality

The formal definition of the functionality for deterministic truncation over $\mathbb{Z}_{2^k}$ is presented below.

Functionality 1: $\mathcal{F}_{\text{DTrun}}$

Input: $[\![a]\!]^{\mathcal{R}}, m$, where $a = \text{Rep}(v) \in \mathbb{Z}_{2^k} \subset GR(2^k, d)$ represents an ℓ-bit signed integer $v \in [-2^{\ell-1}, 2^{\ell-1} - 1]$, and $1 \leqslant m \leqslant \ell - 1$.
Output: The functionality performs the following steps:

1. Internally reconstructs a;
2. Identifies the signed integer v represented by a, and computes the truncated integer $v' = \lfloor v/2^m \rfloor$;
3. Generates a new sharing $[\![d]\!]^{\mathcal{R}}$, where $d = \text{Rep}(v') \in \mathbb{Z}_{2^k} \subset GR(2^k, d)$, and distributes the shares to all parties.

C.2 UC Proof of Theorem 4.1

Proof. The correctness of the protocol is mathematically guaranteed by Corollary 3.2. As for privacy, we construct a simulator $\mathcal{S}$ to simulate the behaviors of honest parties. The simulator $\mathcal{S}$ internally runs a copy of Π_{DTrun} where it plays the role of the honest parties and executes the protocol with the corrupted parties. The goal of the proof is to show that the simulated view of the adversary (run internally by $\mathcal{S}$) in the ideal world is perfectly indistinguishable for any environment $\mathcal{Z}$ from the view of the adversary $\mathcal{A}$ in the real world. In the ideal world, the simulator $\mathcal{S}$ also plays the role of the hybrid functionalities towards the adversary. We use I to represent the set of corrupted parties, where $I \subseteq [n]$ and $|I| \leqslant t$. The simulator $\mathcal{S}$ behaves as follows:

1. The simulator $\mathcal{S}$ receives the inputs and outputs, $\{a^i\}$ and $\{d^i\}$ respectively, for every $i \in I$ from the ideal functionality $\mathcal{F}_{\text{DTrun}}$, where the superscript i denotes the secret share corresponding to the i-th party.

2. $\mathcal{S}$ simulates the offline phase as follows: For every $i \in I$, the simulator $\mathcal{S}$ invokes $\mathcal{F}_{\text{edaBit}}$ and generates the valid random shares g^i, h^i, and $(r^i_k, \ldots, r^i_1)$. The simulator $\mathcal{S}$ then invokes $\mathcal{F}_{\text{B2A}}$ to set the arithmetic share output as r^i_{msb}.

3. $\mathcal{S}$ simulates each step of the online phase as follows:

   (a) Step 1. The simulator $\mathcal{S}$ samples a random value $\zeta \in_{\$} \mathbb{Z}_{2^k}$ and sets $c = \zeta + 0x + \cdots + 0x^{d-1} \in GR(2^k, d)$. That is, c is sampled uniformly from the constant subring $\mathbb{Z}_{2^k} \subset GR(2^k, d)$ while its higher-degree coefficients are set to zero. The simulator $\mathcal{S}$ then provides this unique public value to all corrupted parties $i \in I$. To simulate the reconstruction transcript, let P_{king} denote the designated party that reconstructs the secret. If $P_{\text{king}} \in I$, $\mathcal{S}$ additionally samples a random degree-t polynomial $f_c(x)$ satisfying $f_c(0) = c$ and $f_c(\beta_i) = 2^{\ell-1} + a^i + r^i$ for all $i \in I$, and provides $f_c(\beta_j)$ for each honest party $j \notin I$ to P_{king} as the simulated reconstruction shares. If $P_{\text{king}} \notin I$, no additional reconstruction shares from honest parties are observed by the corrupted parties.

   (b) Step 2-4. For every $i \in I$, the simulator $\mathcal{S}$ computes $c_1 = \text{Rep}([c]_m)$ where $\hat{c}$ is the integer representative of the constant term ζ. The simulator subsequently derives c_{msb} and the bit decomposition $(q_m, \ldots, q_1)$ alongside the wraparound indicator shares $s^i = r^i_{\text{msb}}(1 - c_{\text{msb}})$.

   (c) Step 5. $\mathcal{S}$ simulates the bitwise comparison functionality $\mathcal{F}_{\text{BitLT}}$. For every corrupted party $i \in I$, the simulator $\mathcal{S}$ provides a random element $\delta^{\mathcal{B},i} \in_{\$} GR(2, d)$ as its Boolean share. In the Shamir scheme over $GR(2, d)$, any t elements are perfectly consistent with a valid t-degree polynomial sharing of either $\delta^{\mathcal{B}} = 0$ or $\delta^{\mathcal{B}} = 1$ (embedded in the constant subring). Thus, these shares are identical in distribution to those in a real execution.

   (d) Step 6. $\mathcal{S}$ simulates the conversion functionality $\mathcal{F}^{\mathcal{R}}_{\text{B2A}}$. For every $i \in I$, the simulator $\mathcal{S}$ computes $\delta^i = 2^{k-m} s^i + \text{Trunc}_m(c) - g^i - 2^{\ell-1-m} - d^i$.

4. The simulator $\mathcal{S}$ outputs the view of the corrupted parties.

In the real protocol execution, the secret a and the mask r are both embedded into the constant subring $\mathbb{Z}_{2^k} \subset GR(2^k, d)$ as degree-0 terms where all higher-degree coefficients are fixed to zero. Since r is sampled uniformly at random from $\mathbb{Z}_{2^k}$ and added to a within this subring, the resulting value $c = a + r \pmod{2^k}$ is perfectly uniform within the constant subring with all higher-degree coefficients remaining zero. Consequently, the simulated transcript c generated by $\mathcal{S}$ in Step 1 is identical in distribution to the actual transcript observed in the real world. Furthermore, the subsequent online transcripts are deterministic functions of a^i, d^i, c, and the random shares $\delta^{\mathcal{B},i}$. The view generated by the simulator $\mathcal{S}$ is thereby *perfectly indistinguishable* from the real view.

□

C.3 Truncation with RMFE-Based Packing

We adopt the RMFE-based packing framework of Zhou (2026) to represent multiple values within a single Galois ring element and enable SIMD-style computation. Building on this packing framework, we extend our deterministic truncation protocol Π_{DTrun} to truncate ρ packed fixed-point values in parallel. To explicitly distinguish these parallelized operations from their single-secret counterparts presented in the main text, we append the packing dimension ρ to the sharing type indicator $\mathcal{T} \in \{\mathcal{R}, \mathcal{B}\}$. Specifically, we use the unified double-superscript notation, e.g., $\mathcal{F}^{\mathcal{R},\rho}_{\text{Mult}}$, for functionalities and protocols operating on RMFE-packed data. For primitives like bitwise comparison that inherently implies sharing types, we simply use the superscript ρ, e.g., $\mathcal{F}^{\rho}_{\text{BitLT}}$. As established by Zhou (2026), the secure multiplication of RMFE-packed sharings $\mathcal{F}^{\mathcal{R},\rho}_{\text{Mult}}$ inherently involves a re-encoding step to maintain packing consistency.

When packing ρ secrets, our truncation protocol operates on a packed sharing $[\![a]\!]^{\mathcal{R}}$ where $a = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$ with each $a_i \in \mathbb{Z}_{2^k}$ representing an ℓ-bit signed integer, and produces an output sharing $[\![d]\!]^{\mathcal{R}}$ satisfying $d = \mathsf{Pack}(\mathrm{Trunc}^{\mathrm{s}}_m(a_0), \ldots, \mathrm{Trunc}^{\mathrm{s}}_m(a_{\rho-1}))$. Protocol 4 outlines the explicit execution. Note that all functionalities invoked in Protocol 4 are the RMFE-packed versions of their single-secret counterparts, operating in a fully SIMD fashion. We emphasize that although the multiplication in Step 5 involves a public constant, it is implemented via the interactive functionality $\mathcal{F}^{\mathcal{R},\rho}_{\text{Mult}}$ because the re-encoding process requires communication. It is worth noting that, for the bit decomposition in Step 6, Pack and $\mathsf{PackInv}$ refer to the corresponding Boolean RMFE packing map and its inverse between $\mathbb{Z}_2^{\rho}$ and the valid packed elements of $GR(2, d)$. For notational simplicity, we use the same symbols for the corresponding arithmetic maps between $\mathbb{Z}_{2^k}^{\rho}$ and the valid packed elements of $GR(2^k, d)$.

Protocol 4: Truncation with RMFE-Based Packing

Input: $[\![a]\!]^{\mathcal{R}}, m$;
Output: $[\![d]\!]^{\mathcal{R}}$;
Offline phase:

1. $\{[\![g]\!]^{\mathcal{R}}, ([\![r_i]\!]^{\mathcal{B}})_{i=m+1}^{k}\} = \mathcal{F}^{\rho}_{\text{edaBit}}(k-m)$;
2. $\{[\![h]\!]^{\mathcal{R}}, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^{m}\} = \mathcal{F}^{\rho}_{\text{edaBit}}(m)$;
3. $[\![r_{\text{msb}}]\!]^{\mathcal{R}} = \mathcal{F}^{\rho}_{\text{B2A}}([\![r_k]\!]^{\mathcal{B}})$;
4. $[\![r]\!]^{\mathcal{R}} = 2^m [\![g]\!]^{\mathcal{R}} + [\![h]\!]^{\mathcal{R}}$;
5. $[\![\mathrm{Trunc}_m(r)]\!]^{\mathcal{R}} = [\![g]\!]^{\mathcal{R}}$;

Online phase:

1. Let $\boldsymbol{e} = (1, \ldots, 1) \in \mathbb{Z}_{2^k}^{\rho}$ be the all-ones vector;
2. $c = \Pi_{\text{Reveal}}\left(\mathsf{Pack}(2^{\ell-1}\boldsymbol{e}) + [\![a]\!]^{\mathcal{R}} + [\![r]\!]^{\mathcal{R}}\right)$;
3. $(c_0, \ldots, c_{\rho-1}) = \mathsf{PackInv}(c)$;
4. $c_{\text{msb}} = \mathsf{Pack}\left((c_0)_{\text{msb}}, \ldots, (c_{\rho-1})_{\text{msb}}\right)$;
5. $[\![s]\!]^{\mathcal{R}} = \mathcal{F}^{\mathcal{R},\rho}_{\text{Mult}}\left([\![r_{\text{msb}}]\!]^{\mathcal{R}}, \mathsf{Pack}(\boldsymbol{e}) - c_{\text{msb}}\right)$;
6. For each $j \in \{0, \ldots, \rho-1\}$, the parties locally compute the m least significant bits of c_j, denoted by $(q_{j,m}, \ldots, q_{j,1})$, satisfying $[c_j]_m = \sum_{i=1}^{m} 2^{i-1} q_{j,i}$ with $q_{j,i} \in \{0, 1\}$. For each

$i \in \{1, \ldots, m\}$, the parties pack the i-th bits of all components into $GR(2, d)$ as $q_i = \mathsf{Pack}(q_{0,i}, \ldots, q_{\rho-1,i})$. The resulting packed bit decomposition satisfies $([c_0]_m, \ldots, [c_{\rho-1}]_m) = \sum_{i=1}^{m} 2^{i-1} \mathsf{PackInv}(q_i)$;

7. $[\![\delta]\!]^{\mathcal{B}} = \mathcal{F}^{\rho}_{\text{BitLT}}\left((q_i)_{i=1}^{m}, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^{m}\right)$;
8. $[\![\delta]\!]^{\mathcal{R}} = \mathcal{F}^{\rho}_{\text{B2A}}([\![\delta]\!]^{\mathcal{B}})$;
9. $[\![d]\!]^{\mathcal{R}} = 2^{k-m}[\![s]\!]^{\mathcal{R}} + \mathsf{Pack}\left(\mathrm{Trunc}_m(c_0), \ldots, \mathrm{Trunc}_m(c_{\rho-1})\right) - [\![\mathrm{Trunc}_m(r)]\!]^{\mathcal{R}} - [\![\delta]\!]^{\mathcal{R}} - \mathsf{Pack}(2^{\ell-1-m}\boldsymbol{e})$;

Protocol 4 is a component-wise RMFE-packed instantiation of Π_{DTrun}. After unpacking the revealed masked value, each component follows the same truncation relation as in the single-secret protocol, including the wraparound correction s_i and the lower-bit carry δ_i. Hence, the output correctly packs $(\mathrm{Trunc}^{\mathrm{s}}_m(a_0), \ldots, \mathrm{Trunc}^{\mathrm{s}}_m(a_{\rho-1}))$. For security, the packed edaBit functionalities provide independent random masks for the packed components, so the revealed value is perfectly masked component-wise. All remaining operations are either local or invoke the corresponding secure packed functionalities. Therefore, the simulation argument follows directly from that of Π_{DTrun}, and the packed protocol inherits its perfect security in the semi-honest, honest-majority setting.

C.4 RMFE-Based Fixed-Point Multiplication

We provide the complete construction and analysis of our protocol $\Pi_{\text{MulRescale}}$. Unlike standard packed truncation, which takes a degree-t sharing of a valid packed element as input, $\Pi_{\text{MulRescale}}$ operates directly on a degree-$2t$ sharing $[\![z]\!]^{\mathcal{R}}_{2t}$ of a Galois ring intermediate. Such intermediates naturally arise in RMFE-based neural network training after packed linear computation, e.g., when accumulating the multiplication terms of an affine layer before truncation, with the bias lifted to the corresponding fixed-point scale and RMFE representation. Unlike $\mathsf{PackInv}(\cdot)$, which is defined only for valid packed elements, $\mathsf{Unpack}(\cdot)$ is defined over the entire Galois ring. Hence, even when $z \notin \mathrm{Im}(\mathsf{Pack})$, we can write $\mathsf{Unpack}(z) = (z_0, \ldots, z_{\rho-1})$. Assuming that each z_i represents an ℓ-bit signed integer with $\ell \leqslant k-1$, $\Pi_{\text{MulRescale}}$ transforms $[\![z]\!]^{\mathcal{R}}_{2t}$ into a degree-t sharing $[\![d]\!]^{\mathcal{R}}_{t} = [\![\mathsf{Pack}(\mathrm{Trunc}^{\mathrm{s}}_m(z_0), \ldots, \mathrm{Trunc}^{\mathrm{s}}_m(z_{\rho-1}))]\!]^{\mathcal{R}}_{t}$. For fixed-point computation, setting $m = f$ restores the target fixed-point scale.

The remaining challenge is to securely mask the degree-$2t$ intermediate. Since z may lie outside $\mathrm{Im}(\mathsf{Pack})$, a mask restricted to valid packed elements does not provide sufficient randomness to hide the entire Galois ring intermediate. We therefore introduce additional randomness over the full Galois ring while preserving the component-wise truncation masks after unpacking. Following Zhou (2026), we use $\mathcal{F}_{\text{DouRan}}$ to generate a correlated pair $\{[\![\omega]\!]^{\mathcal{R}}_{2t}, [\![\mathsf{ReEncode}(\omega)]\!]^{\mathcal{R}}_{t}\}$ for uniformly random $\omega \in GR(2^k, d)$. Protocol 5 gives the resulting construction, where $\boldsymbol{e} = (1, \ldots, 1) \in \mathbb{Z}^{\rho}_{2^k}$ denotes the all-ones vector. In Step 5 of the online phase, Pack and PackInv follow the same Boolean RMFE convention as in Protocol 4.

Protocol 5: $\Pi_{\text{MulRescale}}$

Input: $[\![z]\!]^{\mathcal{R}}_{2t}, m$;
Output: $[\![d]\!]^{\mathcal{R}}_{t}$;
Offline phase:

1. $\{[\![g]\!]^{\mathcal{R}}_{t}, ([\![r_i]\!]^{\mathcal{B}})_{i=m+1}^{k}\} = \mathcal{F}^{\rho}_{\text{edaBit}}(k-m)$;
2. $\{[\![h]\!]^{\mathcal{R}}_{t}, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^{m}\} = \mathcal{F}^{\rho}_{\text{edaBit}}(m)$;
3. $[\![r_{\mathrm{msb}}]\!]^{\mathcal{R}}_{t} = \mathcal{F}^{\rho}_{\text{B2A}}([\![r_k]\!]^{\mathcal{B}})$;
4. $[\![r]\!]^{\mathcal{R}}_{t} = 2^m[\![g]\!]^{\mathcal{R}}_{t} + [\![h]\!]^{\mathcal{R}}_{t}$ and $[\![\mathrm{Trunc}_m(r)]\!]^{\mathcal{R}}_{t} = [\![g]\!]^{\mathcal{R}}_{t}$;
5. $\{[\![\omega]\!]^{\mathcal{R}}_{2t}, [\![\mathsf{ReEncode}(\omega)]\!]^{\mathcal{R}}_{t}\} = \mathcal{F}_{\text{DouRan}}()$;
6. The parties locally form $[\![\mu]\!]^{\mathcal{R}}_{2t} = \mathsf{Pack}(\boldsymbol{e})[\![r]\!]^{\mathcal{R}}_{t} + [\![\omega]\!]^{\mathcal{R}}_{2t} - \mathsf{Pack}(\boldsymbol{e})[\![\mathsf{ReEncode}(\omega)]\!]^{\mathcal{R}}_{t}$.

Online phase:

1. $c = \Pi_{\text{Reveal}}\left([\![z]\!]^{\mathcal{R}}_{2t} + \mathsf{Pack}(\boldsymbol{e})\mathsf{Pack}(2^{\ell-1}\boldsymbol{e}) + [\![\mu]\!]^{\mathcal{R}}_{2t}\right)$;

2. $(c_0, \ldots, c_{\rho-1}) = \mathsf{Unpack}(c)$;
3. $c_{\mathrm{msb}} = \mathsf{Pack}((c_0)_{\mathrm{msb}}, \ldots, (c_{\rho-1})_{\mathrm{msb}})$;
4. $[\![s]\!]_t^{\mathcal{R}} = \mathcal{F}_{\mathsf{Mult}}^{\mathcal{R},\rho}\left([\![r_{\mathrm{msb}}]\!]_t^{\mathcal{R}}, \mathsf{Pack}(\boldsymbol{e}) - c_{\mathrm{msb}}\right)$;
5. For each $j \in \{0, \ldots, \rho-1\}$, the parties locally compute the m least significant bits of c_j, denoted by $(q_{j,m}, \ldots, q_{j,1})$, satisfying $[c_j]_m = \sum_{i=1}^{m} 2^{i-1} q_{j,i}$ with $q_{j,i} \in \{0,1\}$. For each $i \in \{1, \ldots, m\}$, the parties pack the i-th bits of all components into $GR(2,d)$ as $q_i = \mathsf{Pack}(q_{0,i}, \ldots, q_{\rho-1,i})$. The resulting packed bit decomposition satisfies $([c_0]_m, \ldots, [c_{\rho-1}]_m) = \sum_{i=1}^{m} 2^{i-1}\mathsf{PackInv}(q_i)$;
6. $[\![\delta]\!]^{\mathcal{B}} = \mathcal{F}_{\mathsf{BitLT}}^{\rho}\left((q_i)_{i=1}^{m}, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^{m}\right)$;
7. $[\![\delta]\!]_t^{\mathcal{R}} = \mathcal{F}_{\mathsf{B2A}}^{\rho}([\![\delta]\!]^{\mathcal{B}})$;
8. $[\![d]\!]_t^{\mathcal{R}} = 2^{k-m}[\![s]\!]_t^{\mathcal{R}} + \mathsf{Pack}\left(\mathrm{Trunc}_m(c_0), \ldots, \mathrm{Trunc}_m(c_{\rho-1})\right) - [\![\mathrm{Trunc}_m(r)]\!]_t^{\mathcal{R}} - [\![\delta]\!]_t^{\mathcal{R}} - \mathsf{Pack}(2^{\ell-1-m}\boldsymbol{e})$.

Compared with Protocol 4, the main difference of $\Pi_{\mathsf{MulRescale}}$ lies in the masking and opening of the degree-$2t$ intermediate in Step 1. Since this intermediate may lie outside $\mathrm{Im}(\mathsf{Pack})$, it cannot in general be masked by a valid packed element alone. We therefore construct the degree-$2t$ mask as

$$\mu = \mathsf{Pack}(\boldsymbol{e})r + \omega - \mathsf{Pack}(\boldsymbol{e})\mathsf{ReEncode}(\omega), \tag{14}$$

where $\boldsymbol{e} = (1, \ldots, 1) \in \mathbb{Z}_{2^k}^{\rho}$, $r = \mathsf{Pack}(r_0, \ldots, r_{\rho-1})$, and ω is sampled uniformly from $GR(2^k, d)$. By the RMFE property, for any $\boldsymbol{x} \in \mathbb{Z}_{2^k}^{\rho}$, $\mathsf{Unpack}\left(\mathsf{Pack}(\boldsymbol{e})\mathsf{Pack}(\boldsymbol{x})\right) = \boldsymbol{e} \odot \boldsymbol{x} = \boldsymbol{x}$, where $\odot$ denotes component-wise multiplication. Moreover, since $\mathsf{ReEncode}(\omega) = \mathsf{Pack}(\mathsf{Unpack}(\omega))$, the $\mathbb{Z}_{2^k}$-linearity of Unpack and the RMFE property give

$$\begin{aligned}
&\mathsf{Unpack}\left(\omega - \mathsf{Pack}(\boldsymbol{e})\mathsf{ReEncode}(\omega)\right)\\
&= \mathsf{Unpack}(\omega) - \mathsf{Unpack}\left(\mathsf{Pack}(\boldsymbol{e})\mathsf{ReEncode}(\omega)\right)\\
&= \mathsf{Unpack}(\omega) - \mathsf{Unpack}\left(\mathsf{Pack}(\boldsymbol{e})\mathsf{Pack}(\mathsf{Unpack}(\omega))\right)\\
&= \mathsf{Unpack}(\omega) - \boldsymbol{e} \odot \mathsf{Unpack}(\omega)\\
&= 0.
\end{aligned} \tag{15}$$

Hence,

$$\begin{aligned}
\mathsf{Unpack}(\mu) &= \mathsf{Unpack}\left(\mathsf{Pack}(\boldsymbol{e})r + \omega - \mathsf{Pack}(\boldsymbol{e})\mathsf{ReEncode}(\omega)\right)\\
&= \mathsf{Unpack}\left(\mathsf{Pack}(\boldsymbol{e})r\right) + \mathsf{Unpack}\left(\omega - \mathsf{Pack}(\boldsymbol{e})\mathsf{ReEncode}(\omega)\right)\\
&= \mathsf{Unpack}\left(\mathsf{Pack}(\boldsymbol{e})r\right)\\
&= (r_0, \ldots, r_{\rho-1}).
\end{aligned} \tag{16}$$

Therefore, the additional randomization term in μ does not alter the component-wise truncation masks, since applying Unpack to μ still recovers $(r_0, \ldots, r_{\rho-1})$.

We next show that the resulting mask μ is uniformly distributed over the entire Galois ring. Let $\mathcal{K} = \ker(\mathsf{Unpack})$ and define the linear map

$$\begin{aligned}
S\colon \mathbb{Z}_{2^k}^{\rho} &\longrightarrow GR(2^k, d),\\
\boldsymbol{x} &\longmapsto \mathsf{Pack}(\boldsymbol{e})\mathsf{Pack}(\boldsymbol{x}).
\end{aligned}$$

The RMFE property gives $\mathsf{Unpack}(S(\boldsymbol{x})) = \boldsymbol{x}$, and hence S is injective. Moreover, suppose that $y \in \mathrm{Im}(S) \cap \mathcal{K}$. Since $y \in \mathrm{Im}(S)$, there exists $\boldsymbol{x} \in \mathbb{Z}_{2^k}^{\rho}$ such that $y = S(\boldsymbol{x})$. On the other hand, $y \in \mathcal{K} = \ker(\mathsf{Unpack})$ implies $\mathsf{Unpack}(y) = 0$. Therefore, $0 = \mathsf{Unpack}(y) = \mathsf{Unpack}(S(\boldsymbol{x})) = \boldsymbol{x}$. Hence, $\boldsymbol{x} = 0$ and consequently $y = S(\boldsymbol{x}) = S(0) = 0$. Thus, $\mathrm{Im}(S) \cap \mathcal{K} = \{0\}$. Since $\mathsf{Unpack} \circ S$ is the identity map on $\mathbb{Z}_{2^k}^{\rho}$, Unpack is surjective and every $x \in GR(2^k, d)$ admits the decomposition $x = S(\mathsf{Unpack}(x)) + (x - S(\mathsf{Unpack}(x)))$ where $S(\mathsf{Unpack}(x)) \in \mathrm{Im}(S)$ and $x - S(\mathsf{Unpack}(x)) \in \mathcal{K}$. Since every element of $GR(2^k, d)$ can be written as the sum of an element in $\mathrm{Im}(S)$ and an element in $\mathcal{K}$, and since $\mathrm{Im}(S) \cap \mathcal{K} = \{0\}$, we obtain

$$GR(2^k, d) = \mathrm{Im}(S) \oplus \mathcal{K}. \tag{17}$$

Now define $L(\omega) = \omega - \mathsf{Pack}(\boldsymbol{e})\mathsf{ReEncode}(\omega) = \omega - S(\mathsf{Unpack}(\omega))$. As shown above, $L(\omega) \in \mathcal{K}$. Furthermore, for every $x \in \mathcal{K}$, we have $\mathsf{Unpack}(x) = 0$, and therefore $L(x) = x - S(0) = x$. Thus, L is surjective onto $\mathcal{K}$. Since L is a surjective $\mathbb{Z}_{2^k}$-linear map onto $\mathcal{K}$, every element of $\mathcal{K}$ has the same number of preimages. Hence, uniformly random $\omega \in GR(2^k, d)$ induces a uniformly random $L(\omega) \in \mathcal{K}$. The packed edaBit generation provides a uniformly random vector $(r_0, \ldots, r_{\rho-1}) \in \mathbb{Z}_{2^k}^{\rho}$ that is independent of ω. Since S is injective, $\mathsf{Pack}(\boldsymbol{e})r = S(r_0, \ldots, r_{\rho-1})$ is uniformly distributed over $\mathrm{Im}(S)$. Therefore, μ is the sum of two independent uniform elements from the two direct summands:

$$\mu = \underbrace{\mathsf{Pack}(\boldsymbol{e})r}_{\in \mathrm{Im}(S)} + \underbrace{\omega - \mathsf{Pack}(\boldsymbol{e})\mathsf{ReEncode}(\omega)}_{\in \mathcal{K}}. \tag{18}$$

Since every element of $GR(2^k, d)$ has a unique decomposition into these two direct summands, it follows that $\mu \in_{\$} GR(2^k, d)$. Thus, μ uniformly masks the entire degree-$2t$ Galois-ring intermediate while preserving the component-wise truncation masks after applying Unpack.

We next incorporate the signed offset. By the RMFE property, $\mathsf{Unpack}\left(\mathsf{Pack}(\boldsymbol{e})\mathsf{Pack}(2^{\ell-1}\boldsymbol{e})\right) = 2^{\ell-1}\boldsymbol{e}$. Therefore, after the masked intermediate is revealed and unpacked, the parties obtain $c_i = (z_i + 2^{\ell-1} + r_i) \bmod 2^k$ for each $i \in \{0, \ldots, \rho-1\}$. After the masked opening in Step 1, the remaining operations are identical to those in Protocol 4. Together with the masking properties established above, this yields the correctness and security of $\Pi_{\text{MulRescale}}$.

C.5 RMFE-Based Gradient Aggregation

We next adapt our deterministic truncation to the RMFE-based gradient aggregation of Zhou (2026). Given a packed sharing $[\![a]\!]^{\mathcal{R}}$ with $a = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$, the goal is to obtain a scalar sharing $[\![d]\!]^{\mathcal{R}}$ satisfying $d = \mathrm{Trunc}_m^{\mathrm{s}}\left(\sum_{j=0}^{\rho-1} a_j\right)$, where the signed aggregate lies in $[-2^{\ell-1}, 2^{\ell-1}-1]$ and $\ell \leqslant k-1$. Following (Zhou, 2026), we use the protocol Π_{Split} to decompose a scalar truncation mask $r \in \mathbb{Z}_{2^k}$ across the ρ packed components, producing $v = \mathsf{Pack}(v_0, \ldots, v_{\rho-1})$ such that $\sum_{j=0}^{\rho-1} v_j = r \pmod{2^k}$. After revealing the masked packed value, the parties recover and sum its components to obtain a masked representation of the aggregate. We then apply our κ-free deterministic truncation mechanism to this scalar masked value. Protocol 6 gives the complete construction.

Protocol 6: $\Pi_{\text{AggRescale}}$

Input: $[\![a]\!]^{\mathcal{R}}$, m, where $a = \mathsf{Pack}(a_0, \ldots, a_{\rho-1})$;
Output: $[\![d]\!]^{\mathcal{R}}$ satisfying $d = \mathrm{Trunc}_m^{\mathrm{s}}\left(\sum_{j=0}^{\rho-1} a_j\right)$;
Offline phase:

1. $\{[\![g]\!]^{\mathcal{R}}, ([\![r_i]\!]^{\mathcal{B}})_{i=m+1}^{k}\} = \mathcal{F}_{\text{edaBit}}(k-m)$;
2. $\{[\![h]\!]^{\mathcal{R}}, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^{m}\} = \mathcal{F}_{\text{edaBit}}(m)$;
3. $[\![r_{\mathrm{msb}}]\!]^{\mathcal{R}} = \mathcal{F}_{\text{B2A}}([\![r_k]\!]^{\mathcal{B}})$;
4. $[\![r]\!]^{\mathcal{R}} = 2^m [\![g]\!]^{\mathcal{R}} + [\![h]\!]^{\mathcal{R}}$;
5. $[\![\mathrm{Trunc}_m(r)]\!]^{\mathcal{R}} = [\![g]\!]^{\mathcal{R}}$;
6. $[\![v]\!]^{\mathcal{R}} = \Pi_{\text{Split}}([\![r]\!]^{\mathcal{R}})$, where $v = \mathsf{Pack}(v_0, \ldots, v_{\rho-1})$ satisfies $\sum_{j=0}^{\rho-1} v_j = r \pmod{2^k}$.

Online phase:

1. Let $\boldsymbol{e_0} = (1, 0, \ldots, 0) \in \mathbb{Z}_{2^k}^{\rho}$;
2. $c = \Pi_{\text{Reveal}}\left(\mathsf{Pack}(2^{\ell-1}\boldsymbol{e_0}) + [\![a]\!]^{\mathcal{R}} + [\![v]\!]^{\mathcal{R}}\right)$;
3. $(c_0, \ldots, c_{\rho-1}) = \mathsf{PackInv}(c)$;
4. The parties locally compute the aggregated masked value $c_\Sigma = \mathrm{Rep}\left(\sum_{j=0}^{\rho-1} \hat{c}_j\right)$;
5. $c_1 = \mathrm{Rep}([c_\Sigma]_m)$;
6. $[\![s]\!]^{\mathcal{R}} = [\![r_{\mathrm{msb}}]\!]^{\mathcal{R}} (1 - (c_\Sigma)_{\mathrm{msb}})$;
7. The parties locally compute the m-bit decomposition of c_1, $(q_m, \ldots, q_1)$, satisfying $c_1 = \sum_{i=1}^{m} 2^{i-1} q_i$;

8. $[\![\delta]\!]^{\mathcal{B}} = \mathcal{F}_{\text{BitLT}}\left((q_i)_{i=1}^m, ([\![r_i]\!]^{\mathcal{B}})_{i=1}^m\right)$;
9. $[\![\delta]\!]^{\mathcal{R}} = \mathcal{F}_{\text{B2A}}([\![\delta]\!]^{\mathcal{B}})$;
10. $[\![d]\!]^{\mathcal{R}} = 2^{k-m}[\![s]\!]^{\mathcal{R}} + \text{Trunc}_m(c_\Sigma) - [\![\text{Trunc}_m(r)]\!]^{\mathcal{R}} - [\![\delta]\!]^{\mathcal{R}} - 2^{\ell-1-m}$.

The key observation is that the aggregation procedure can reduce the packed computation to a standard scalar truncation instance. Let $r \in \mathbb{Z}_{2^k}$ denote the scalar truncation mask. The protocol invokes Π_{Split} to encode this mask across the packed components as $(v_0, \ldots, v_{\rho-1})$ such that $\sum_{j=0}^{\rho-1} v_j = r \bmod 2^k$. These mask components are added to the packed gradients before the masked value is revealed. To preserve the signed interpretation of the aggregate, the offset $2^{\ell-1}$ is inserted only once using $\boldsymbol{e_0} = (1, 0, \ldots, 0)$. After unpacking the revealed value into $(c_0, \ldots, c_{\rho-1})$ and summing the components modulo 2^k, the parties obtain $c = (\sum_{j=0}^{\rho-1} a_j + 2^{\ell-1} + r) \bmod 2^k$. Hence, the packed aggregation has been transformed into exactly the masked scalar form required by Π_{DTrun}.

The subsequent wraparound and carry corrections can therefore reuse the same truncation logic as Π_{DTrun}. The relation $\sum_{j=0}^{\rho-1} v_j = r \bmod 2^k$ guarantees correctness of the masking transformation, while the security of the revealed packed value follows from Π_{Split} and the underlying RMFE functionalities. The remaining security argument is inherited from Π_{DTrun}.

D Supplementary Details for Neural Network Training

D.1 Implementation of the SVM Loss

To bypass the expensive secure division required by the standard cross-entropy loss, we adopt the multiclass Support Vector Machine (SVM) loss (Weston & Watkins, 1999; Zhou, 2023). Mathematically, the gradient of the SVM loss with respect to output logits mainly relies on an indicator function that determines whether the score difference exceeds a margin Δ. Consequently, the secure evaluation of this loss can be fully formulated via the derivative of ReLU, denoted as $\mathcal{F}_{\text{DReLU}}$. Because our truncation mechanism rounds towards negative infinity, the derivative of ReLU for an ℓ-bit signed value a can be expressed as $\text{DReLU}(a) = 1 + \text{Trunc}^{\text{s}}_{\ell-1}(a)$, thereby bypassing the need for secure division. We assume that the input to $\mathcal{F}_{\text{DReLU}}$ lies within the signed ℓ-bit range, and its output is an unscaled binary value in $\{0, 1\}$ rather than a fixed-point value encoded with scale 2^f.

We derive our protocol Π_{SVM} by instantiating the gradient evaluation technique from (Zhou, 2023) on Galois rings. The concrete procedure is detailed in Protocol 7, where y denotes the one-hot ground-truth label represented in fixed-point form, z represents the neural network output logits, Δ denotes the unscaled margin, C is the number of neurons in the output layer, and G represents the resulting gradient of the loss function with respect to z. The subscript i indicates the i-th element of a given vector. Accordingly, the one-hot label entries satisfy $y_i \in \{0, 2^f\}$, while the fixed-point representation of the margin is $2^f\Delta$. To facilitate a clear and intuitive presentation, Protocol 7 illustrates the execution using non-packed data. We use Π_{FPMul} to denote scalar fixed-point multiplication, where two degree-t sharings are multiplied locally to obtain a degree-$2t$ sharing, followed by deterministic truncation by f bits to restore the fixed-point scale and obtain a degree-t sharing.

Protocol 7: Π_{SVM}

Input: $[\![z]\!], [\![y]\!]$;
Output: $[\![G]\!]$;
1. **for** $i = 1$ to C **do**
2. $\quad [\![n_i]\!] = \Pi_{\text{FPMul}}\left([\![z_i]\!], [\![y_i]\!]\right)$;
3. **end for**
4. $[\![a]\!] = \sum_{i=1}^{C} [\![n_i]\!]$;
5. **for** $i = 1$ to C **do**
6. $\quad [\![m_i]\!] = [\![z_i]\!] - [\![a]\!] + 2^f\Delta - (\Delta + 1)[\![y_i]\!]$;

7. **end for**
8. Given $[\![m]\!]$, the parties invoke $\mathcal{F}_{\text{DReLU}}$ to compute $[\![k]\!]$;
9. $[\![t]\!] = \sum_{i=1}^{C} [\![k_i]\!]$;
10. **for** $i = 1$ to C **do**
11. $\quad [\![G_i]\!] = \Pi_{\text{Mult}}([\![-t]\!], [\![y_i]\!]) + 2^f [\![k_i]\!]$;
12. **end for**

Protocol 7 is presented using non-packed data for clarity. In our implementation, the element-wise operations are evaluated in SIMD form using RMFE-based packing. In particular, the fixed-point multiplication primitive Π_{FPMul} naturally corresponds to $\Pi_{\text{MulRescale}}$ in the RMFE-packed setting, where local multiplication produces a degree-$2t$ intermediate that is directly truncated and converted into a valid degree-t packed sharing. The truncation-based nonlinear operations are instantiated with their corresponding RMFE-based primitives. Thus, no extra RMFE-specific construction is required for the SVM loss.

D.2 Training Procedure

Our private training follows the standard mini-batch forward and backward propagation procedure of the RMFE-based framework of Zhou (2026). Rather than being invoked only as a standalone truncation operation, our deterministic truncation mechanism serves as a building block for several components throughout the training process, including ReLU, max pooling, average pooling, and cross-entropy loss. We follow the corresponding constructions of Zhou (2026), while replacing the underlying truncation operations with our κ-free mechanism.

For fixed-point multiplication in both forward and backward propagation, we use $\Pi_{\text{MulRescale}}$ to directly process degree-$2t$ intermediates and output valid degree-t packed sharings. For gradient aggregation, we use our protocol $\Pi_{\text{AggRescale}}$ to obtain the truncated aggregate before updating the secret-shared model parameters. All other training operations follow the original RMFE-based framework of Zhou (2026).