DYNAFIX: Dynamic Fixed-Point Encoding for Arbitrary-Range MPC

Yuntian Chen, Tianpei Lu*, Zhanyong Tang, Bingsheng Zhang, Wenjing Yang, Zhuzhu Wang, Kui Ren

***Abstract*—Privacy-preserving computation over real numbers typically employs either floating-point or fixed-point arithmetic. While fixed-point methods are highly efficient, they struggle to handle wide dynamic ranges. Conversely, floating-point methods support a much larger numerical scope but incur overheads more than a hundred times higher than their fixed-point counterparts. In this paper, we propose DYNAFIX, a dynamic fixed-point computation scheme that strikes a balance between floating-point and fixed-point arithmetic. Compared to traditional fixed-point approaches, our scheme supports an arbitrary numerical range; compared to floating-point computation, it maintains performance comparable to fixed-point execution. Experimental results demonstrate that our method achieves a $24.1\times$ speedup over the state-of-the-art when evaluating high-precision functions, such as the exponential function.**

I. INTRODUCTION

Secure multi-party computation (MPC) [1], [2], [3] has become a vital tool for privacy-preserving data processing, enabling collaborative analysis of confidential datasets without compromising privacy. Nevertheless, a fundamental mismatch exists between the discrete nature of finite rings/fields—upon which MPC protocols are natively built—and the continuous real-number arithmetic required by modern applications. As privacy-centric tasks in machine learning [4], [5], [6], data mining [7], [8], [9], [10], and statistical modeling [11], [12], [13] become more prevalent, the community faces a persistent hurdle: devising efficient numerical representations and MPC-compatible algorithms that can seamlessly bridge the gap between fractional values and integer-based secure logic.

The most widely adopted approach for real-number arithmetic is fixed-point encoding [14], [8], [15], [16], [17], [18], [19], which is typically implemented by encoding real values onto a ring through scaling, allowing the underlying ring arithmetic to be used directly for computation. However, this approach usually requires an additional truncation protocol to handle the "double-scaling" effect that occurs after multiplications. Recent studies [20], [21], [22] have focused on minimizing the overhead introduced by these truncation steps. As a result, the cost of fixed-point computation in the MPC setting is now approaching that of standard ring operations (i.e., addition and multiplication).

However, due to their fixed fractional precision and bit-width, fixed-point numbers struggle with computations involving a dynamic range. In scenarios like high-degree polynomial evaluations, where the output range can far exceed the input domain, fixed-point representations are highly susceptible to overflow. To handle such cases effectively, floating-point arithmetic is typically required. However, floating-point arithmetic comes at a prohibitive cost in secure computation settings due to the complex non-linear operations required, such as mantissa alignment and normalization. Prior work [23], [24], [25], [26], [27], [22] has shown that floating-point-based secure inference can incur up to two orders of magnitude overhead compared to fixed-point alternatives, making it impractical for large-scale tasks.

A deeper analysis reveals a fundamental trade-off: while fixed-point arithmetic can theoretically support arbitrary real-number computations if embedded in a sufficiently large ring, such an oversized ring is impractical for real-world scenarios. The advantage of floating-point arithmetic over fixed-point lies in its use of exponents to dynamically adjust the radix point. However, in the context of MPC, the primary cost stems precisely from this dynamic adjustment of exponents and bit-widths. This observation leads us to the following research question:

Is there a way to bypass the dynamic adjustment of the radix point and bit-width in the ciphertext domain, thereby achieving an efficient real-number computation scheme that supports arbitrary ranges?

a) Our Approach: Motivated by this observation, we propose DYNAFIX, a **Dynamic Fixed-Point Computation framework** that introduces an adaptive encoding scheme: each wire has its own scaling factor and bit-width, while all intermediate arithmetic is carried out on a single, pre-determined suitably sized ring. Unlike fixed-point representations with static precision, our method dynamically adjusts per-wire quantization granularity without changing the underlying ring during online computation, enabling stable polynomial evaluation while preserving MPC-friendly integer arithmetic.

Specifically, for the values on each wire of the real-number arithmetic, we perform dynamic fixed-point encoding. That is, for each wire in the function $y = f(x_1, \ldots, x_n)$, we encode the pre-determined data range as $\hat{x}_i = \lfloor x_i \cdot s_{x_i} \rfloor$, where $\hat{x}_i$ can be regarded as a value on an ℓ_{x_i}-bit ring. Here, s_{x_i} is a scaling factor used to scale x_i onto the ℓ_{x_i}-bit ring. Similarly, for y, the encoding is $\hat{y} = \lfloor y \cdot s_y \rfloor$. Consequently, the real-number computation f is transformed into a dynamic fixed-point computation f':

$$\hat{y} = f'(\hat{x}_1, \ldots, \hat{x}_n) = \lfloor f(\hat{x}_1/s_{x_1}, \ldots, \hat{x}_n/s_{x_n}) \cdot s_y \rfloor$$

The scaling factor s for each wire is determined by the respective input and output domains. In many privacy-centric applications, such as neural network inference or statistical

* Corresponding author.

profiling, techniques like quantization-aware analysis can be employed to tighten these domains. This allows for higher precision within narrower bit widths, thereby optimizing both accuracy and efficiency.

Next, we need to evaluate f' on the ring via MPC correctly. We introduce a ring extension operation: first, since the bit-widths ℓ_{x_i} and ℓ_y vary across different wires, all inputs are extended to a sufficiently large ring ℓ_b, and f' is then evaluated on ℓ_b without overflow. Finally, we restore the output from the ℓ_b ring back to the ℓ_y ring.

However, the introduction of a large ℓ_b-bit ring may still encounter scalability bottlenecks if the underlying MPC protocol's cost depends on the ring size ℓ_b. To mitigate this, we employ a local ring extension technique inspired by Lu *et al* [22]. Specifically, we leverage a key property of Masked Secret Sharing (MSS): the online evaluation of a circuit can be decomposed into a local computation phase and a reconstruction phase, where communication occurs only in the latter. Under this paradigm, all input shares are first locally extended to the ℓ_b-bit ring without any network interaction. The polynomial f' is then evaluated locally on ℓ_b, yielding an intermediate result. Crucially, we truncate this result back to the target ℓ_y-bit ring before the reconstruction operation. Consequently, the online communication cost is decoupled from the intermediate bit-width ℓ_b and depends solely on the output bit-width ℓ_y. For non-linear computations, such as e^x, we realize them by fitting them into polynomials.

b) Contributions: Our main contributions are summarized as follows:

- **Dynamic fixed-point computation framework.** We propose a novel dynamic fixed-point encoding scheme that adjusts ring size and bit-width during computation, enabling efficient real-number approximation within MPC-friendly integer arithmetic.
- **Efficient 3-party secure computation protocol.** We design a communication-efficient three-party computation (3-PC) protocol tailored to dynamic-ring arithmetic, enabling stable and low-overhead execution of addition and multiplication under adaptive ring configurations.
- **System implementation and evaluation.** We implement a full prototype and conduct extensive experiments across diverse network settings and model architectures. Results demonstrate that our approach significantly reduces computation and communication overhead compared to fixed-point and floating-point baselines, while maintaining inference accuracy. Our source code is available at [28].

c) Performance: We evaluate our framework across diverse benchmarks. For fundamental operators, such as multiplication and inner product, our benchmark shows that DynaFIXachieves a 3.35× - 17.3× speedup in online time compared to the SOTA approach, ASTRA [29], in MAN settings. Even in high-latency WAN environments, DynaFIXconsistently maintains a performance lead of up to 11.4×. For the polynomial like the exponential operator, our approach achieves 24.1× and 11.3× performance improvement compared to the SOTA works—Secfloat and Nexus. Remarkably, DynaFIXmaintains near-constant precision as input scales increase, effectively decoupling error from calculation magnitude.

d) Paper Organization: Section. II provides the preliminaries. Section III presents a detailed description of the construction of our DFC. In Section IV, we implement and evaluate our DFC protocol, while Section. V discusses related work.

II. Preliminaries

Notation. Consider a three-party MPC setting denoted by $\mathcal{P} = \{P_0, P_1, P_2\}$. We represent an ℓ-bit ring as $\mathbb{Z}_{2^\ell} = \{0, 1, \ldots, 2^\ell - 1\}$. Given an element x within this ring, $x_{(i)}$ refers to the i-th bit of its binary representation in big-endian format. The most significant bit, or the sign bit, is denoted as $\mathsf{sign}(x) = x_{(0)}$. Let $x, y \in [0, 2^\ell - 1]$. We denote $b = \mathsf{wrap}(x, y, 2^\ell)$ as the wrap-around bit of the addition $x+y \pmod{2^\ell}$, which is defined as: $b = \lfloor (x+y)/2^\ell \rfloor$. Furthermore, we introduce the $\mathsf{drop}(x, d)$ function, which discards the d least significant bits of x. This operation is formally defined as $\mathsf{drop}(x, d) := \lfloor x \cdot 2^{-d} \rfloor \in \mathbb{Z}_{2^{\ell-d}}$. We use $\eta_{j,k}$ to denote the common seed shared by both P_j and P_k. Our protocol employs two types of secret sharing:

- Two-party additive secret share: We utilize a 2-out-of-2 additive secret sharing scheme over the ring $\mathbb{Z}_{2^\ell}$, denoted as $[\cdot]^\ell$. A secret x is partitioned into two components, $[x]_1^\ell$ and $[x]_2^\ell$, such that $x = [x]_1^\ell + [x]_2^\ell \pmod{2^\ell}$. In this setup, each party P_j (where $j \in \{1, 2\}$) holds the respective share $[x]_j^\ell$.
- Three-party masked secret share: A value x in the ring $\mathbb{Z}_{2^\ell}$ is represented via $\langle \cdot \rangle^\ell$-sharing as the pair $(m_x, [r_x]^\ell)$. Here, $r_x \in \mathbb{Z}_{2^\ell}$ serves as a uniform random mask generated by P_0, such that the masked value is $m_x = x - r_x \pmod{2^\ell}$. The distribution of data is as follows: the computing parties P_j ($j \in \{1, 2\}$) possess both the masked value m_x and their respective share $[r_x]_j^\ell$ of the mask. Meanwhile, P_0 acts as the dealer, maintaining the additive shares $([r_x]_1^\ell, [r_x]_2^\ell)$ of the mask r_x.

To streamline our notation, we omit the modulus $\mathbb{Z}_{2^\ell}$ when it is clear from the context, writing $[x]$ or $\langle x \rangle$ instead. We extend this notation to vectors $\mathbf{v} = (v_0, \ldots, v_{n-1})$ such that $[\mathbf{v}]$ represents a sequence of individual shares $([v_0], \ldots, [v_{n-1}])$, with a parallel definition for $\langle \mathbf{v} \rangle$. Both sharing formats support linear homomorphic operations. Specifically, for any secrets x, y and a public scalar c, the following identities hold over $\mathbb{Z}_{2^\ell}$: $[x + y] = [x] + [y]$, $[c \cdot x] = c[x]$, and $[x + c] = ([x]_1 + c, [x]_2)$. These additive and scaling properties are similarly preserved for the $\langle \cdot \rangle$ sharing scheme.

Secret Sharing Scheme. We denote the protocols for sharing $[\cdot]$ and $\langle \cdot \rangle$ shares as $\Pi_{[\cdot]}$ and $\Pi_{\langle \cdot \rangle}$, respectively. An instantiation $\Pi_{[\cdot]}(x)$ signifies that a specific party distributes shares of the input x. Conversely, when $\Pi_{[\cdot]}(\cdot)$ is invoked without a parameter, it refers to the collective generation of a secret-shared uniform random value. To optimize communication efficiency, we employ pseudorandom generators (PRGs) as per

the techniques in [30]. Specifically, when parties P_j and P_k are required to sample shared randomness, they execute a PRG using a pre-established common seed $\eta_{j,k}$. The following provides a concise summary of these secret-sharing frameworks. seed $\eta_{j,k}$. A brief overview of the secret-sharing schemes is given below.

- $[x]^\ell \leftarrow \Pi^\ell_{[\cdot]}(x)$ (Generate shares of x): If P_0 holds x, P_0 and P_1 jointly sample a random value $[x]_1 \in \mathbb{Z}_{2^\ell}$ with seed $\eta_{0,1}$; Party P_0 then sends $[x]_2 = x - [x]_1 \pmod{2^\ell}$ to P_2.
- $[x]^\ell \leftarrow \Pi^\ell_{[\cdot]}(\cdot)$: Parties P_0 and P_1 jointly sample a random value $[x]_1 \in \mathbb{Z}_{2^\ell}$ using the seed $\eta_{0,1}$, and parties P_0 and P_2 jointly sample a random value $[x]_2 \in \mathbb{Z}_{2^\ell}$ using the seed $\eta_{0,2}$.
- $\langle x\rangle^\ell \leftarrow \Pi^\ell_{\langle\cdot\rangle}(x, P_0)$ (Generate shares of x): In the offline phase, all parties execute $[r_x]^\ell \leftarrow \Pi^\ell_{[\cdot]}$, where P_0 knows both seeds corresponding to $[r_x]_1$ and $[r_x]_2$; Party P_0 computes $m_x = x - [r_x]_1 - [r_x]_2$ and sends m_x to P_1 and P_2.
- $\langle x\rangle^\ell \leftarrow \Pi^\ell_{\langle\cdot\rangle}(\cdot)$: In the offline phase, all parties execute $[r_x]^\ell \leftarrow \Pi^\ell_{[\cdot]}$. Parties P_1 and P_2 then jointly sample a random value m_x using $\eta_{1,2}$.

In our protocol, for the convenience of security proofs, we introduce ideal functionality $\mathcal{F}_{\mathsf{share}}[2^\ell]$ to represent the functionality corresponding to $[x]^{2^\ell} \leftarrow \Pi^\ell_{[\cdot]}$. Specifically, $\mathcal{F}_{\mathsf{share}}$ samples $[x]_1, [x]_2 \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, then sends $[x]_1$ to P_0 and P_1, and sends $[x]_2$ to P_0 and P_2.

Preprocessing. We follow the "preprocessing" paradigm [31], which splits the protocol into two phases: the preprocessing/offline phase is data-independent and can be executed without data input, and the online phase is data-dependent and is executed after data input. Specifically, all the items r_x of share $\langle x\rangle$ of our protocols can be generated in the circuit-depend offline phase. What the parties need to do in the online phase is to collaborate in computing m_x for P_1 and P_2.

Multiplication Over Ring. We use $\Pi^{2^\ell}_{\mathsf{mult}}$ to denote multiplication over the ring $\mathbb{Z}_{2^\ell}$. For sharings of the form $\langle\cdot\rangle$, we adopt the protocol proposed in ASTRA [32], which is secure in the semi-honest setting. For the multiplication $\langle z\rangle^{2^\ell} \leftarrow \Pi^{2^\ell}_{\mathsf{mult}}(\langle x\rangle^{2^\ell}, \langle y\rangle^{2^\ell})$ corresponding to $z = x \cdot y$, all parties first generate $[r_z]^\ell \leftarrow \Pi^\ell_{[\cdot]}$ for the output wire in the offline phase. In the online phase, the parties compute the masked value m_z, which can be written as

$$\begin{aligned} m_z = xy - r_z &= (m_x + r_x)(m_y + r_y) - r_z \\ &= m_x m_y + m_x r_y + m_y r_x + r_x r_y - r_z \ . \end{aligned}$$

The term $[\Gamma']^{2^\ell} = m_x m_y + m_x[r_y]^{2^\ell} + m_y[r_x]^{2^\ell}$ can be computed by P_1 and P_2 locally, while $[\Gamma]^{2^\ell} = [r_x \cdot r_y]^{2^\ell} - [r_z]^{2^\ell}$ can be secret shared by P_0 to P_1 and P_2 in the offline phase. In the online phase, P_1 and P_2 compute and reconstruct $[m_z]^{2^\ell} = [\Gamma']^{2^\ell} + [\Gamma]^{2^\ell}$.

For $x_1 \in \mathbb{Z}_{2^\ell}$ input by P_1 and $x_2 \in \mathbb{Z}_{2^\ell}$ input by P_2, we abuse the notation $[z]^{2^\ell} \leftarrow \Pi^{2^\ell}_{\mathsf{mult}}(x_1, x_2)$ to denote the multiplication of x_1 and x_2, yielding a secret-shared result in the $[\cdot]$. In the offline phase, for $i \in \{1, 2\}$, P_0 and P_i, jointly generate a random value r_i using same seed. Party P_0 computes $r = r_1 \cdot r_2$ and invokes $\Pi_{[\cdot]}(r)$ to secret share r to P_1 and P_2. In the online phase, each party P_i, for $i \in \{1, 2\}$, sends $a_i = x_i - r_i$ to P_{3-i}. Then P_1 sets $[z]^{2^\ell}_1 = a_1 \cdot a_2 + a_2 \cdot r_1 + [r]^{2^\ell}_1$ and P_2 sets $[z]^{2^\ell}_2 = a_1 \cdot r_2 + [r]^{2^\ell}_2$.

Fixed-point Representation. For a real number x with one sign-bit, $\ell_i - 1$ bits of integer precision and ℓ_f bits of fractional precision, we define its fixed-point representation (over ring) as $\hat{x} = \lfloor x \cdot 2^{\ell_f} \rfloor \pmod{2^\ell}$, while $\ell_i + \ell_f = \ell$. This encoding designates the most significant bit as the sign bit. Negative values are encoded into the interval $(2^\ell - 2^{\ell-2}, 2^\ell)$, while positive values are encoded into $[0, 2^{\ell-2})$. Compared to conventional fixed-point representation [14], [8] and [16], [33], [34], this encoding sacrifices one bit of precision. Under this representation, the addition of two fixed-point numbers can be directly implemented using ring addition.

Truncation and Drop. The purpose of truncation is to take a secret-shared value x in the ring $\mathbb{Z}_{2^\ell}$ and, by removing the ℓ' least significant bits, obtain a sharing of $\lfloor x/2^{\ell'} \rfloor$ while remaining within the same ring $\mathbb{Z}_{2^\ell}$. Truncation is a costly computation in MPC, with the most efficient known approach being probabilistic truncation [35]. Probabilistic truncation is directly achieved by locally right-shifting the secret shares by the corresponding number of bits. However, this technique introduces a non-negligible probability of high-order truncation errors. The rationale is that performing a simple right shift on a secret sharing allows the wrap-around bit, which would otherwise be discarded by the modulus, to migrate into a significant bit position. This shift-induced bit migration introduces an arithmetic error in the resulting value. In contrast, the drop operation follows a different computational logic: for a value x in the $\mathbb{Z}_{2^\ell}$ ring, it discards the ℓ' least significant bits to produce a sharing over a reduced ring $\mathbb{Z}_{2^{\ell-\ell'}}$. This approach inherently avoids high-order bit errors, as the modular wrap-around bit remains external to the new ring's bit-width and thus does not migrate into the significant bit positions.

Threat Model. We consider a synchronized three-party computing (3PC) setting comprising a set of parties $\mathcal{P} = \{P_0, P_1, P_2\}$. Our protocol achieves security against semi-honest adversaries, assuming a computationally bounded adversary can statically corrupt at most one of the three parties ($t = 1$).

III. Secure Dynamic Fixed-point for Arbitrary-Range calculation

In the context of the MPC setting, real-number arithmetic is predominantly implemented using either fixed-point or floating-point representations. Fixed-point arithmetic is favored for its efficiency, as it maps real numbers into a finite ring through a scaling factor. However, it struggles with dynamic range functions like e^x, where the codomain significantly exceeds the domain, which can easily trigger overflows even with valid inputs. While floating-point arithmetic mitigates this via exponents, the resulting overhead from operations like bit-alignment is often two orders of magnitude higher than that of fixed-point. In this section, we introduce a dynamic fixed-point evaluation scheme that bridges this gap.

It supports an arbitrary numerical range while maintaining an execution cost comparable to traditional fixed-point protocols. As illustrated in Figure 1, our framework follows a two-tier design. We first introduce the concept of dynamic fixed-point arithmetic in the encoding layer (Section III-A), which leverages an adaptive numeric scheme to map inputs into individual scales and bit-widths while executing operations over a unified intermediate ring. This is seamlessly integrated with our 3PC layer (Section III-B), which securely implements the evaluation by shifting the computational burden to a pre-processing phase. Specifically, the offline dealer P_0 precomputes wrap-bits and correlated randomness, enabling evaluators P_1 and P_2 to perform online arithmetic with minimal communication, thereby drastically reducing the execution critical path.

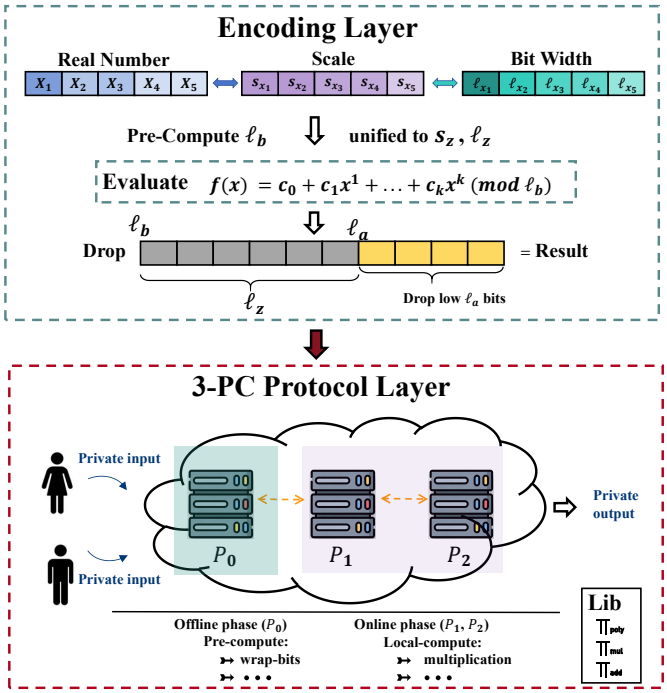

Fig. 1: The overview of DynaFix.

A. Dynamic Fixed-Point Computation (DFC) for Arbitrary-Range Rings

We observe that traditional fixed-point arithmetic relies on a globally predefined numerical range. That is, all intermediate values in the arithmetic are represented using a fixed ring $\mathbb{Z}_{2^\ell}$ and a constant scaling factor s. We propose to enhance the computational range by relaxing this constraint, defining each intermediate value in the fixed-point arithmetic over distinct scaling factors and rings. We define Dynamic Fixed-Point Computation (DFC) as follows.

Definition 1 (DFC). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a target mathematical function. A DFC is a ring-domain transformation that evaluates f over dynamic fixed-point encodings under a pre-determined numerical objective. For a real value x_i, let its encoding be $\hat{x}_i = \lfloor x_i \cdot s_{x_i} \rfloor \in \mathbb{Z}_{2^{\ell_{x_i}}}$, where s_{x_i} is the associated scaling factor and ℓ_{x_i} denotes its bit-width. Given a target output scaling factor s_y and bit-width ℓ_y, the ideal output is defined as:

$$\hat{y} = \left\lfloor f\left(\frac{\hat{x}_1}{s_{x_1}}, \ldots, \frac{\hat{x}_n}{s_{x_n}}\right) \cdot s_y \right\rfloor \pmod{2^{\ell_y}}.$$

A DFC instantiates this mapping via an integer-ring transformation $F : \mathbb{Z}_{2^{\ell_{x_1}}} \times \cdots \times \mathbb{Z}_{2^{\ell_{x_n}}} \to \mathbb{Z}_{2^{\ell_y}}$, such that $F(\hat{x}_1, \ldots, \hat{x}_n)$ realizes the aforementioned output within the prescribed numerical precision.

Technique Overview. For compatibility with the MPC setting, we implement these computations on a ring-based structure. The core logic is to embed the scaling factor $\frac{1}{s_{x_i}}$ into a high bit-width ring through left-shifting, execute the evaluation of $f(\hat{x}_1/s_{x_1}, \ldots, \hat{x}_n/s_{x_n}) \cdot s_y$ in this domain, and eventually truncate result to fit the target ring $\mathbb{Z}_{2^{\ell_y}}$.

Throughout the DFC constructions, we use an intermediate ring $\mathbb{Z}_{2^{\ell_b}}$ to accommodate the enlarged numerical range caused by scale transitions, multiplications, and accumulations. The parameter ℓ_r denotes the resolution factor $s_r = 2^{\ell_r}$ introduced to preserve precision when a real-valued scale ratio is converted into an integer coefficient. The parameter ℓ_a denotes an additional extraction offset, which can be used to reserve guard bits or to align the desired output with a prescribed bit window.

Given an intermediate value $u \in \mathbb{Z}_{2^{\ell_b}}$, DFC extracts the target output by $\hat{z} = \mathsf{drop}(u, \ell_a + \ell_r) \pmod{2^{\ell_z}}$, where $\ell_z = \ell_b - (\ell_a + \ell_r)$. Hence, the interval $[\ell_a + \ell_r, \ell_b)$ specifies the output bit window. In particular, ℓ_r removes the auxiliary resolution introduced by s_r, while ℓ_a provides an additional controllable offset for range management.

The core value of DFC lies in its pre-determined numerical flexibility. Its output scaling factor s_y is assigned a priori based on the functional characteristics (e.g., the range of e^x), enabling seamless transitions between non-uniform scaling factors (i.e., $s_{x_i} \neq s_y$). This mechanism bypasses the prohibitive dynamic alignment overhead associated with floating-point arithmetic while providing robust support for transcendental functions where the output magnitude shifts significantly. Based on this definition, we propose a set of dynamic ring-based computation paradigms for 1st-order terms $f(x, y) = ax + by$ (addition), 2nd-order terms $f(x, y) = xy$ (multiplication), and high-order terms ($d \geq 3$) encompassing both recursive polynomial expansions (e.g., $e^x \approx f(x) = a_k x^k + a_{k-1}x^{k-1} + \cdots + a_0$) and disparate-scale alignments (e.g., independent-scale inner products).

The First-order Terms: Addition ($f(x) = ax + by$). Let $\hat{x} \in \mathbb{Z}_{2^{\ell_x}}$ and $\hat{y} \in \mathbb{Z}_{2^{\ell_y}}$ be the dynamic fixed-point representations of real values x and y with scaling factors s_x and s_y, respectively. To compute the sum $\hat{z} \in \mathbb{Z}_{2^{\ell_z}}$ with the pre-assigned scaling factor s_z (i.e., $\hat{z} = \lfloor (x+y) \cdot s_z \rfloor$), we account for arbitrary scale transitions by introducing an intermediate resolution factor $s_r = 2^{\ell_r}$, which is used to shift the scaling factor to an integer representation. Let $C_x = \lfloor \frac{s_z}{s_x} \cdot s_r \rfloor$ and $C_y = \lfloor \frac{s_z}{s_y} \cdot s_r \rfloor$ be the integer coefficients. Given an interval

(ℓ_a, ℓ_b), the arbitrary-bit addition is performed as:

$$\hat{z} = \mathsf{drop}\left(C_x \cdot \hat{x} + C_y \cdot \hat{y} \mod 2^{\ell_b}, \ell_a + \ell_r\right) \pmod{2^{\ell_z}}$$

where $[\ell_a + \ell_r, \ell_b)$ specifies the bit-interval for extraction, and $\ell_b \geq \max(\ell_x + \lceil \log_2 C_x \rceil, \ell_y + \lceil \log_2 C_y \rceil) + \ell_a$. Here, $\ell_a + \ell_r$ defines the new drop boundary that simultaneously removes the fractional noise from the original encoding and the additional scaling introduced by s_r. The output bit-width satisfies $\ell_z = \ell_b - (\ell_a + \ell_r)$, ensuring that the final result $\hat{z}$ correctly represents the sum under the pre-assigned scale s_z within the target ring $\mathbb{Z}_{2^{\ell_z}}$.

Case Study. Consider two real values $x = 42.3$ and $y = 27.6$ with scaling factors $s_x = 64$ and $s_y = 32$. Their dynamic fixed-point representations are $\hat{x} = \lfloor 42.3 \cdot 64 \rfloor = 2707$ and $\hat{y} = \lfloor 27.6 \cdot 32 \rfloor = 883$, where $\hat{x} \in \mathbb{Z}_{2^{12}}$ and $\hat{y} \in \mathbb{Z}_{2^{10}}$. Let the target scaling factor be $s_z = 32$, and the target bit-width be $\ell_z = 12$.

To handle the potential precision loss during the scale transition ($\frac{s_z}{s_x} = 0.5$), we introduce an intermediate resolution factor $s_r = 2^2 = 4$ (i.e., $\ell_r = 2$), and calculate the integer scaling coefficients as $C_x = \lfloor \frac{s_z}{s_x} \cdot s_r \rfloor = \lfloor \frac{32}{64} \cdot 4 \rfloor = 2$ and $C_y = \lfloor \frac{s_z}{s_y} \cdot s_r \rfloor = \lfloor \frac{32}{32} \cdot 4 \rfloor = 4$. The calculation is performed in an intermediate ring $\mathbb{Z}_{2^{14}}$ (where $\ell_b = \ell_z + \ell_r = 14$) to accommodate the increased magnitude:

$$u = (C_x \cdot \hat{x} + C_y \cdot \hat{y}) \pmod{2^{14}} = (2 \cdot 2707 + 4 \cdot 883) = 8946.$$

To obtain the final encoding $\hat{z}$ at scale s_z, we apply the drop operator to remove the $\ell_r = 2$ bits introduced by s_r (assuming $\ell_a = 0$ for simplicity), we have $\hat{z} = \mathsf{drop}(8946, 2) = \lfloor 8946/2^2 \rfloor = 2236$. The result $\hat{z} = 2236$ correctly resides in the target ring $\mathbb{Z}_{2^{12}}$. The final real value is $z = \hat{z}/s_z = 2236/32 = 69.875$, which closely approximates the ideal sum $x + y = 69.9$ with an approximation deviation error of 0.025.

The Second-order Terms: From Heterogeneous Inner Products to Atomic Multiplication ($f(x, y) = a_k x_k y_k + a_{k-1} x_{k-1} y_{k-1} + \cdots + a_0 x_0 y_0$). The DFC framework handles 2-degree operations by accumulating products within a suitably sized intermediate ring, followed by a single drop. By pre-calculating transition coefficients, DFC ensures that high-precision accumulation occurs within a secure intermediate ring, effectively eliminating intermediate rounding errors. In complex tensor operations, such as per-channel quantized convolutions or attention mechanisms, vector elements often possess independent scaling factors $s_{(x,i)}$ and $s_{(y,i)}$. This necessitates normalizing each product term to a unified high-precision ring before accumulation. DFC defines element-wise transition coefficients C_i that serve as linear scaling operators to transform each product term from its intrinsic scale to a unified high-precision plane defined by the target scale s_z and resolution factor s_r, we have $C_i = \left\lfloor \frac{s_z \cdot s_r}{s_{(x,i)} \cdot s_{(y,i)}} \right\rfloor$, for $i = 1, \ldots, n$. Given an extraction interval (ℓ_a, ℓ_b), the inner product is then performed by aligning each product to the target scale before accumulation, defined as:

$$\hat{z} = \mathsf{drop}\left(\sum_{i=1}^{n} (C_i \cdot \hat{x}_i \cdot \hat{y}_i) \mod 2^{\ell_b}, \ell_a + \ell_r\right) \pmod{2^{\ell_z}}$$

where the intermediate bit-width ℓ_b must satisfy $\ell_b \geq \max_i(\ell_{x,i} + \ell_{y,i} + \lceil \log_2 C_i \rceil) + \lceil \log_2 n \rceil + \ell_a$ to ensure overflow prevention.

Case Study. To demonstrate the capability of resolving spatial heterogeneity, we consider an inner product of length $n = 2$ where each element pair resides in a distinct quantization domain. This scenario simulates a dot product between a weight vector and an activation vector.

Consider the following real-valued vectors and their respective scaling factors:

- $\mathbf{x} = [0.75, 1.25]$ with $\mathbf{s}_x = [128, 64]$. Encodings: $\hat{\mathbf{x}} = [\lfloor 0.75 \cdot 128 \rfloor, \lfloor 1.25 \cdot 64 \rfloor] = [96, 80]$.
- $\mathbf{y} = [2.0, 0.5]$ with $\mathbf{s}_y = [32, 16]$. Encodings: $\hat{\mathbf{y}} = [\lfloor 2.0 \cdot 32 \rfloor, \lfloor 0.5 \cdot 16 \rfloor] = [64, 8]$.

We have two pairs of data with mismatched scales: $x_1 = 0.75$ ($s_{x,1} = 128$), $y_1 = 2.0$ ($s_{y,1} = 32$). Encoding: $\hat{x}_1 = 96, \hat{y}_1 = 64$. Intrinsic product scale: 4096, and $x_2 = 1.25$ ($s_{x,2} = 64$), $y_2 = 0.5$ ($s_{y,2} = 16$). Encoding: $\hat{x}_2 = 80, \hat{y}_2 = 8$. Intrinsic product scale: 1024. We target an output scale $s_z = 256$ and set a resolution factor $s_r = 256$ ($\ell_r = 8$) to protect precision. The transition coefficients C_i are pre-computed to align both terms to the same altitude ($s_z \cdot s_r = 65536$): $C_1 = \lfloor 65536/4096 \rfloor = 16$ and $C_2 = \lfloor 65536/1024 \rfloor = 64$. The intermediate sum is computed by multiplying the three components for each term, yielding $\text{term}_1 = 98304$ and $\text{term}_2 = 40960$, and thus $u = 139264$. Then we apply $\mathsf{drop}(u, 8)$ to remove the resolution bits, yielding $\hat{z} = \lfloor 139264/256 \rfloor = 544$. Decoding gives $z = 544/256 = 2.125$, which matches the floating-point result $(0.75 \times 2.0) + (1.25 \times 0.5) = 2.125$, demonstrating that DFC achieves exact alignment across heterogeneous scales in a single-pass computation.

For operations with uniform scaling factors (i.e., $s_{(x,i)} = s_x$ and $s_{(y,i)} = s_y$ for all i), the general model simplifies. The transition coefficients C_i collapse into a unified coefficient C_{mult}. Let $\hat{\mathbf{x}} = \{\hat{x}_i\}_{i=1}^n$ and $\hat{\mathbf{y}} = \{\hat{y}_i\}_{i=1}^n$ be vectors where each element $\hat{x}_i \in \mathbb{Z}_{2^{\ell_x}}$ and $\hat{y}_i \in \mathbb{Z}_{2^{\ell_y}}$ is dynamic fixed-point encoding with a unique scaling factor s_x and s_y, respectively. To compute the inner product $\hat{z} = \lfloor (\sum_{i=1}^n x_i y_i) \cdot s_z \rfloor \in \mathbb{Z}_{2^{\ell_z}}$, we introduce a resolution factor $s_r = 2^{\ell_r}$ and define element-wise transition coefficients $C_{\mathsf{mult}} = \left\lfloor \frac{s_z}{s_x \cdot s_y} \cdot s_r \right\rfloor$. Given an interval (ℓ_a, ℓ_b), the inner product is performed within the ring $\mathbb{Z}_{2^{\ell_b}}$ as:

$$\hat{z} = \mathsf{drop}\left(C_{\mathsf{mult}} \sum_{i=1}^{n} (\hat{x}_i \cdot \hat{y}_i) \mod 2^{\ell_b}, \ell_a + \ell_r\right) \pmod{2^{\ell_z}}$$

where ℓ_b must be analytically pre-determined to satisfy: $\ell_b \geq \ell_x + \ell_y + \lceil \log_2 n \rceil + \lceil \log_2 C_{\mathsf{mult}} \rceil + \ell_a$ to prevent overflow during the fused multiplication and accumulation. This paradigm

ensures that even when the product scale $s_x \cdot s_y$ differs from the target scale s_z, the result is normalized into a coherent representation without intermediate communication. The case study is in Appendix VIII-A.

The most fundamental case of a second-order term occurs when $n = 1$, DFC handles the product of two scalars with mismatched scales. Given an extraction interval (ℓ_a, ℓ_b), the atomic multiplication is defined as:

$$\hat{z} = \mathsf{drop}\left(C_{\mathsf{mult}} \cdot (\hat{x} \cdot \hat{y}) \mod 2^{\ell_b}, \ell_a + \ell_r\right) \pmod{2^{\ell_z}}$$

where the intermediate bit-width must satisfy $\ell_b \geq \ell_x + \ell_y + \lceil \log_2(C_{\mathsf{mult}}) \rceil + \ell_a$. This base case establishes how the transition coefficient C_{mult} and the resolution factor s_r work together to map a raw product to the target scale s_z in a single step. This special case significantly reduces the storage overhead for offline parameters in standard matrix-matrix multiplications. The case study is in Appendix VIII-A

High-order terms ($d \geq 3$): Recursive Polynomial Evaluation ($f(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_0$). While 1st- and 2nd-order primitives typically fit within small-bit rings, high-degree arithmetic ($d \geq 3$) introduces significant challenges for fixed-width MPC protocols. The architectural flexibility of DFC lies in its ability to evaluate any function $f(x)$ that can be approximated by a k-degree polynomial $\sum_{i=0}^{k} a_i x^i$. This is achieved by jointly handling recursive power generation and spatial scale alignment within a dynamic intermediate ring $\mathbb{Z}_{2^{\ell_b}}$. DFC evaluates the expression by jointly handling recursive power generation and spatial alignment. For any function $f(x)$ that can be approximated by a k-degree polynomial $\sum_{i=0}^{k} a_i x^i$, where a_i are publicly known coefficients, DFC evaluates the expression by jointly handling recursive power generation and scale alignment. Given a input $\hat{x} \in \mathbb{Z}_{2^{\ell_x}}$ with scaling factor s_x, and an interval (ℓ_a, ℓ_b), we define the recursive fixed-point powers as

$$\begin{cases} \hat{x}_q^{(1)} = \hat{x}, \quad \hat{x}_q^{(0)} = s_x, \\ \hat{x}_q^{(i)} = \mathsf{drop}\left(\hat{x}_q^{(i-1)} \cdot \hat{x}, \ell_f\right) \pmod{2^{\ell_b}}, \quad \text{for } i > 1, \end{cases}$$

where $\ell_f = \lfloor \log_2 s_x \rfloor$. This feedback shift prevents bit-width explosion by keeping each recursively generated power at the same effective scale s_x, namely $\hat{x}_q^{(i)} \approx x^i \cdot s_x$. To evaluate the weighted sum, we pre-compute transition coefficients C_i that absorb all public constants and target scaling requirements $C_i = \left\lfloor \frac{t_i \cdot s_z \cdot 2^{\ell_r + \ell_a}}{s_{\hat{x}_q^{(i)}}} \right\rfloor$. Here, t_i represents any arbitrary public scalar (e.g., $1/i!$ for Taylor series, or fitting weights for piecewise linear/polynomial approximations). By isolating a_i as a flexible parameter, DFC acts as a general-purpose evaluator for transcendental functions, kernels, or any power-law interactions. The final result is extracted via:

$$\hat{z} = \mathsf{drop}\left(\sum_{i=0}^{k} (C_i \cdot \hat{x}_q^{(i)}) \mod 2^{\ell_b}, \ell_a + \ell_r\right) \pmod{2^{\ell_z}}$$

where the bit-width ℓ_b is analytically pre-determined to accommodate the maximum sum, satisfying $\ell_b \geq \max_i(\lceil \log_2 C_i \rceil) + \ell_x + \lceil \log_2(k+1) \rceil + \ell_a$. This ensures that the non-linear growth of the exponential function is safely bounded and the target precision $\ell_z = \ell_b - (\ell_a + \ell_r)$ is extracted in a single overflow-free drop operation.

Case Study. We evaluate $e^{1.5}$ where $x = 1.5$ is encoded as $\hat{x} = 96$ ($s_x = 64$, $\ell_x = 7$). We target an output scale $s_z = 64$ and bit-width $\ell_z = 14$. The intermediate ring is $\mathbb{Z}_{2^{64}}$. We set $\ell_r = 18$ and $\ell_a = 32$. In this case study, $\max\lceil \log_2 C_i \rceil = 50$, $\ell_x = 7$, and $\lceil \log_2(4) \rceil = 2$. The sum $50 + 7 + 2 = 59$ bits is strictly less than $\ell_b = 64$, ensuring zero overflow. The final extraction uses $\ell_a + \ell_r = 50$, meaning we extract the bits from the 50-th position of the 64-bit sum. To maintain a constant dynamic range, we set the feedback shift $\ell_f = 6$ ($\log_2 s_x$). The first-, second-, and third-order terms are computed as follows: $\hat{x}_q^{(1)} = 96$; $u_2 = 96 \cdot 96 = 9216$ and $\hat{x}_q^{(2)} = \lfloor 9216/2^6 \rfloor = 144$; $u_3 = 144 \cdot 96 = 13824$ and $\hat{x}_q^{(3)} = \lfloor 13824/2^6 \rfloor = 216$. The coefficients C_i are pre-computed and shifted by $\ell_a = 32$ to align with the extraction window. Using the formula $C_i = \lfloor \frac{s_z \cdot 2^{\ell_a + \ell_r}}{i! \cdot s_x} \rfloor = \lfloor \frac{2^{50}}{i!} \rfloor$, the corresponding coefficients can be obtained directly. Then the summation u is performed in $\mathbb{Z}_{2^{64}}$:

$$\begin{aligned} u &= C_0 \cdot \hat{x}_q^{(0)} + C_1 \hat{x}_q^{(1)} + C_2 \hat{x}_q^{(2)} + C_3 \hat{x}_q^{(3)} \pmod{2^{64}} \\ &= (C_0 \cdot 64) + (C_1 \cdot 96) + (C_2 \cdot 144) + (C_3 \cdot 216) \\ &= 301741175033823088. \end{aligned}$$

The value u spans 59 bits ($\approx 3.02 \times 10^{17}$), which is safely within the 64-bit ring. We then apply the extraction $\mathsf{drop}(u, 50)$ to get $\hat{z} = \lfloor 301741175033823088/2^{50} \rfloor \approx 268$. The final result $\hat{z} = 268$ resides in $\mathbb{Z}_{2^{14}}$, yielding the decoded value $z = 268/64 = 4.1875$. The gap is primarily due to the truncation of the 3rd-order Taylor series, not the DFC conversion itself.

Power-of-two Alignment Optimization. A key advantage of the DFC framework is its flexibility in parameter selection. When the target scale s_z is a power-of-two multiple of the input scale(s) (e.g., $s_z/s_x \in 2^{\mathbb{Z}}$ for addition, or $s_z/(s_x s_y) \in 2^{\mathbb{Z}}$ for multiplication), the resolution factor s_r can be simplified to 1 ($\ell_r = 0$). In such scenarios, the fractional alignment is naturally satisfied by the ring structure, and the protocols reduce to direct weighted operations followed by a standard drop (truncation) to the target precision. This optimization eliminates the need for auxiliary resolution bits, further reducing the intermediate bit-width ℓ_b and improving overall computational efficiency.

While high-degree DFC primitives are rooted in foundational 1st-degree (addition) and 2nd-degree (multiplication) operations, they transcend simple algorithmic chaining through fused operator execution. Rather than evaluating k-th degree polynomials by sequentially invoking low-degree primitives—which often incurs compounding errors—our high-degree paradigm integrates spatial alignment and encoding within a unified dynamic ring $\mathbb{Z}_{2^{\ell_b}}$. This consolidation minimizes cumulative truncation noise and optimizes bit-width efficiency throughout the evaluation path. Consequently, by systematically nesting these primitives, our framework establishes a functionally complete arithmetic library capable of

evaluating arbitrary k-th degree polynomials $P(x) = \sum a_i x^i$ with high fidelity.

B. MPC Design and Implementation for DFC

To realize the DFC framework in a privacy-preserving setting, we develop a communication-efficient MPC protocol based on 3-party RSS.

In the secret-sharing setting, our objective is to compute $\langle z \rangle^{2^{\ell_z}}$ directly from input shares $\langle x \rangle^{2^{\ell_x}}$ and $\langle y \rangle^{2^{\ell_y}}$ without requiring explicit bit-length alignment. Consistent with the previous plaintext logic, the core strategy involves lifting these shares into a larger computation ring $\mathbb{Z}_{2^{\ell_b}}$. Directly employing ring extension is prohibitively expensive. As mentioned in the preliminaries, a naive reinterpretation of ℓ_x-bit shares in an ℓ_b-bit ring ($\ell_b > \max(\ell_x, \ell_y)$) makes the wrap-around bit observable, potentially corrupting the arithmetic result.

However, as demonstrated by Lu et al., the wrap-bit can be computed very efficiently from the MSBs of the shares when the MSB of the secret is known. Considering $x = a + b \pmod{2^\ell}$ and the first bit of x is zero (i.e., $x \in [0, 2^{\ell-1})$), the wrap-bit b (satisfying $a + b = x + b \cdot 2^\ell$) can be calculated by $b = \mathsf{sign}(a) \vee \mathsf{sign}(b)$, where $\mathsf{sign}(a)$ and $\mathsf{sign}(b)$ denote the most significant bits (MSBs) of a and b, respectively. We apply this intuition to our dynamic fixed-point computation. Specifically, our DFC scheme is constructed as follows:

- Interpret $\langle x \rangle^{2^{\ell_x}}$ and $\langle y \rangle^{2^{\ell_y}}$ as sharings over a ℓ_b-bit ring and perform multiplication. At this point, the errors e_x and e_y are introduced, respectively.
- Evalute $f(\hat{x}, \hat{y}) \cdot s_y$ over ℓ_b-bit ring, and subtract the error term under the polynomial f.
- Apply drop to obtain the final ℓ_z-bit result $\langle z \rangle^{2^{\ell_z}}$.

To satisfy the aforementioned condition that the MSB remains zero, we restrict the effective bit length of $\hat{x}$ and $\hat{y}$ to $\ell_x - 1$ and $\ell_y - 1$, respectively.That is, $\hat{x} \in [0, 2^{\ell_x - 2})$ for positive value and $\hat{x} \in [2^{\ell_x} - 2^{\ell_x - 2}, 2^{\ell_x})$ for negative, and similarly for $\hat{y}$. We first apply independent shifts in the native rings: $\langle \hat{x}' \rangle = \langle \hat{x} \rangle + 2^{\ell_x - 2}$, $\langle \hat{y}' \rangle = \langle \hat{y} \rangle + 2^{\ell_y - 2}$. We then reinterpret these sharings over a larger ring by directly setting $\langle \hat{x}' \rangle^{2^{\ell_b}} := \langle \hat{x}' \rangle^{2^{\ell_x}}$, and analogously for $\hat{y}'$. It follows that $\hat{x}' = \hat{x} + 2^{\ell_x} \cdot w_x \pmod{2^{\ell_b}}$, where w_x is the wrap value of $\hat{x}'$, and $\hat{y}' = \hat{y} + 2^{\ell_y} \cdot w_y \pmod{2^{\ell_b}}$, where w_y is the wrap value of $\hat{y}'$.

Building on this basis, we further designed efficient MPC protocols to handle different DFC operations.

*1) **DFC-Addition Term** (Π_{add}):* The objective of Π_{add} is to compute the weighted sum $\langle \hat{z} \rangle^{2^{\ell_z}}$ from $\langle \hat{x} \rangle^{2^{\ell_x}}$ and $\langle \hat{y} \rangle^{2^{\ell_y}}$ in the target ring. We have $u = \hat{x} + \hat{y}$:

$$\begin{aligned} u = {} & C_x(\hat{x}' - 2^{\ell_x} w_x - 2^{\ell_x - 2}) \\ & + C_y(\hat{y}' - 2^{\ell_y} w_y - 2^{\ell_y - 2}) \pmod{2^{\ell_b}}. \end{aligned}$$

The wrap value can be decomposed as $w_{\hat{x}} = w_x^{(0)} + w_x^{(1)}$, where $w_x^{(0)} = \mathsf{wrap}([r_{\hat{x}'}]_0, [r_{\hat{x}'}]_1, 2^{\ell_x})$ is the wrap-bit of $[r_{\hat{x}'}]_0 + [r_{\hat{x}'}]_1$, which is known to P_0, $w_x^{(1)} = \mathsf{wrap}(r_{\hat{x}'}, m_{\hat{x}'}, 2^{\ell_x})$ is the wrap-bit of $m_{\hat{x}'} + r_{\hat{x}'}$, which can be calculated by $\mathsf{sign}(m_{\hat{x}'}) \vee \mathsf{sign}(r_{\hat{x}'}) = \mathsf{sign}(m_{\hat{x}'}) + \mathsf{sign}(r_{\hat{x}'}) - \mathsf{sign}(m_{\hat{x}'}) \times \mathsf{sign}(r_{\hat{x}'})$.

Expanding the sum yields,

$$\begin{aligned} u = {} & (C_x \hat{x}' + C_y \hat{y}') - (C_x 2^{\ell_x} w_x + C_y 2^{\ell_y} w_y) \\ & - (C_x 2^{\ell_x - 2} + C_y 2^{\ell_y - 2}) \pmod{2^{\ell_b}}. \end{aligned}$$

We observe that the remaining expression can be written as a linear combination of terms involving masked values (m-related terms, e.g., $m_{\hat{x}'}$ or $m_{\hat{y}'}$), with coefficients derived solely from random-mask-related terms(r-related terms, e.g., $[r_{\hat{x}'}]_1$ or $[r_{\hat{y}'}]_2$). As party P_0 knows the r-related terms, while parties P_1 and P_2 know the m-related terms, we let P_0 compute and secret-share the coefficients during the offline phase. As the m-related terms are public to P_1 and P_2, in the online phase, they only need to locally sum these secret-shared coefficients with the corresponding masked values to obtain the final secret sharing $\langle \hat{z} \rangle$.

For $C_x \hat{x}' + C_y \hat{y}'$, we have

$$\begin{aligned} C_x \hat{x}' + C_y \hat{y}' = {} & C_x \cdot (m_{\hat{x}'} + [r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2) + \\ & C_y \cdot (m_{\hat{y}'} + [r_{\hat{y}'}]_1 + [r_{\hat{y}'}]_2) \pmod{2^{\ell_b}}. \end{aligned}$$

In our 3-PC setting, while the coefficients C_x and C_y are globally public, the reconstructed values $m_{\hat{x}'}$ and $m_{\hat{y}'}$ are only revealed to the computing parties P_1 and P_2 during the online phase (P_0 remains oblivious to these values). Since the additive secret shares $[r_{\hat{x}'}]_j$ and $[r_{\hat{y}'}]_j$ are held locally by P_j ($j \in \{1, 2\}$), the parties P_1 and P_2 can independently compute their respective shares of the linear combination over the ring 2^{ℓ_b} without further interaction. Specifically,

- P_1 computes $[C_x \hat{x}' + C_y \hat{y}']_1 = C_x [r_{\hat{x}'}]_1 + C_y [r_{\hat{y}'}]_1$
- P_2 computes $[C_x \hat{x}' + C_y \hat{y}']_2 = (C_x m_{\hat{x}'} + C_y m_{\hat{y}'}) + C_x [r_{\hat{x}'}]_2 + C_y [r_{\hat{y}'}]_2$

For $C_x 2^{\ell_x} w_x$, we have

$$w_x = w_x^{(0)} + m_{\hat{x}',(0)} + r_{\hat{x}',(0)} - m_{\hat{x}',(0)} r_{\hat{x}',(0)},$$

where $r_{\hat{x}',(0)}$ and $m_{\hat{x}',(0)}$ denote the bit of $r_{\hat{y}'}$ at position 0. The correction term can be expanded as:

$$\begin{aligned} C_x 2^{\ell_x} w_x = {} & C_x 2^{\ell_x} (w_x^{(0)} + r_{\hat{x}',(0)}) + \\ & C_x 2^{\ell_x} m_{\hat{x}',(0)} (1 - r_{\hat{x}',(0)}) \pmod{2^{\ell_b}}. \end{aligned}$$

In the offline phase, P_0 computes the bits $r_{\hat{x}',(0)}$ and the combined terms $(w_x^{(0)} + r_{\hat{x}',(0)})$, then secretly shares them with P_1 and P_2. In the online phase, P_1 and P_2 use the reconstructed bit $m_{\hat{x}',(0)}$ to compute the shared value $[C_x 2^{\ell_x} w_x]$ over $\mathbb{Z}_{2^{\ell_b}}$:

- P_1 computes $[C_x 2^{\ell_x} w_x]_1 = C_x 2^{\ell_x} [w_x^{(0)} + r_{\hat{x}',(0)}]_1 + C_x 2^{\ell_x} m_{\hat{x}',(0)} [1 - r_{\hat{x}',(0)}]_1$;
- P_2 computes $[C_x 2^{\ell_x} w_x]_2 = C_x 2^{\ell_x} [w_x^{(0)} + r_{\hat{x}',(0)}]_2 + C_x 2^{\ell_x} m_{\hat{x}',(0)} [1 - r_{\hat{x}',(0)}]_2$.

The term $C_y 2^{\ell_y} w_y$ is computed symmetrically by sharing $(w_y^{(0)} + r_{\hat{y}',(0)})$ and $(1 - r_{\hat{y}',(0)})$.

For term $C_x 2^{\ell_x - 2} + C_y 2^{\ell_y - 2}$, since it consists purely of public parameters, it can be directly subtracted by one of the parties (e.g., P_2) during the online phase without any communication.

Put all together. In the offline phase, P_0 pre-calculates the carry-bits and generates secret shares of the correction coefficients over $\mathbb{Z}_{2^{\ell_b}}$, including $[w_i^{(0)} + r_{i,(0)}]$ and $[1 - r_{i,(0)}]$ for $i \in \{x, y\}$. In the online phase, P_1 and P_2 first reconstruct the masked values $m_{\hat{x}'}$ and $m_{\hat{y}'}$. Then, they locally compute the final shares of the sum $[u]$ over $\mathbb{Z}_{2^{\ell_b}}$ by combining the aligned input shares, the carry correction terms, and the constant offsets:

- P_1 computes:

$$\begin{aligned}[u]_1 = &- C_x 2^{\ell_x}\big([w_x^{(0)} + r_{x,(0)}]_1 + m_{x,(0)}[1 - r_{x,(0)}]_1\big) \\ &- C_y 2^{\ell_y}\big([w_y^{(0)} + r_{y,(0)}]_1 + m_{y,(0)}[1 - r_{y,(0)}]_1\big) \\ &+ (C_x[r_{\hat{x}'}]_1 + C_y[r_{\hat{y}'}]_1) \pmod{2^{\ell_b}}\end{aligned}$$

- P_2 computes:

$$\begin{aligned}[u]_2 = &(C_x(m_{\hat{x}'} + [r_{\hat{x}'}]_2) + C_y(m_{\hat{y}'} + [r_{\hat{y}'}]_2)) \\ &- C_x 2^{\ell_x}\big([w_x^{(0)} + r_{x,(0)}]_2 + m_{x,(0)}[1 - r_{x,(0)}]_2\big) \\ &- C_y 2^{\ell_y}\big([w_y^{(0)} + r_{y,(0)}]_2 + m_{y,(0)}[1 - r_{y,(0)}]_2\big) \\ &- (C_x 2^{\ell_x - 2} + C_y 2^{\ell_y - 2}) \pmod{2^{\ell_b}}\end{aligned}$$

Subsequently, since the true sum u is embedded in the higher bits of the computed value, P_1 and P_2 locally perform the truncation: $[\hat{z}]_j = \mathsf{drop}([u]_j, \ell_a + \ell_r) \pmod{2^{\ell_z}}$. Finally, they reveal $m_{\hat{z}} = \mathsf{Open}([\hat{z}] - [r_{\hat{z}}])$ to obtain the result $\hat{z}$.

Communication Cost. As shown in the Figure 2, the online phase of DFC-Addition is highly efficient, requiring only the reconstruction of masked input values ($m_{\hat{x}'}$ and $m_{\hat{y}'}$), which incurs a constant communication overhead of $\ell_x + \ell_y$ bits. Unlike traditional mixed-precision protocols that rely on interactive circuits for wrap-bit detection or scale alignment.

*2) **DFC-Multiplication Term (Π_{mult})**:* The objective of Π_{mult} is to compute the product $\langle \hat{z} \rangle^{2^{\ell_z}}$ from $\langle \hat{x} \rangle^{2^{\ell_x}}$ and $\langle \hat{y} \rangle^{2^{\ell_y}}$ in the target ring. We have:

$$\begin{aligned}\hat{x} \cdot \hat{y} = C_{\mathsf{mult}} \cdot \{&(\hat{x}' - 2^{\ell_x} w_x - 2^{\ell_x - 2}) \\ &\cdot (\hat{y}' - 2^{\ell_y} w_y - 2^{\ell_y - 2})\} \pmod{2^{\ell_b}}.\end{aligned}$$

Expanding the product yields,

$$\begin{aligned}\hat{x}\hat{y} = C_{\mathsf{mult}} \cdot \{&\hat{x}'\hat{y}' - 2^{\ell_y} w_y \hat{x}' - 2^{\ell_x} w_x \hat{y}' \\ &- 2^{\ell_y - 2}\hat{x}' - 2^{\ell_x - 2}\hat{y}' \\ &+ 2^{\ell_x + \ell_y - 2}(w_x + w_y) + 2^{\ell_x + \ell_y}(w_x \cdot w_y) \\ &+ 2^{\ell_x + \ell_y - 4}\} \pmod{2^{\ell_b}}\end{aligned}$$

As party P_0 knows the r-related terms, P_1 and P_2 know the m-related terms, we then let P_0 compute and secret-share the coefficients during the offline phase. As the m-related terms are public to P_1 and P_2. In the online phase, they only need to locally multiply these secret-shared coefficients with the corresponding masked values to obtain the final secret sharing $[\hat{x} \cdot \hat{y}]$.

For $C_{\mathsf{mult}} \cdot \hat{x}'\hat{y}'$, we have

$$\begin{aligned}C_{\mathsf{mult}} \cdot \hat{x}'\hat{y}' = C_{\mathsf{mult}} \cdot &(m_{\hat{x}'} m_{\hat{y}'} + m_{\hat{x}'}([r_{\hat{y}'}]_1 + [r_{\hat{y}'}]_2) \\ &+ m_{\hat{y}'}([r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2) \\ &+ ([r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2)([r_{\hat{y}'}]_1 + [r_{\hat{y}'}]_2)) \pmod{2^{\ell_b}}.\end{aligned}$$

Protocol $\langle \hat{z} \rangle^{2^{\ell_z}} \leftarrow \Pi_{\mathsf{add}}(\langle \hat{x} \rangle^{2^{\ell_x}}, \langle \hat{y} \rangle^{2^{\ell_y}})$

Common Input : $\ell_x, \ell_y, \ell_z, \ell_b, \ell_{a+r}$.
Input : $\langle \cdot \rangle^{2^{\ell_x}}$-shared value $\hat{x}$ and $\langle \cdot \rangle^{2^{\ell_y}}$-shared value $\hat{y}$.
Output : $\langle \cdot \rangle^{2^{\ell_z}}$-shared value $\hat{z}$.

Offline:

- P_0 calculates $w_x^{(0)} = \mathsf{wrap}([r_x]_1, [r_x]_2, 2^{\ell_x})$, $r_{x,(0)}$, $r_{y,(0)}$, and $w_y^{(0)} = \mathsf{wrap}([r_y]_1, [r_y]_2, 2^{\ell_y})$;
- P_0 secret-shares $[w_i^{(0)} + r_{i,(0)}]$ and $[1 - r_{i,(0)}]$ for $i \in \{x, y\}$ over $\mathbb{Z}_{2^{\ell_b}}$ to P_1, P_2.
- All parties invoke $[r_{\hat{z}}] \leftarrow \mathcal{F}_{\mathsf{share}}[2^{\ell_z}]$.

Online:

- P_j, for $j \in \{1, 2\}$ does:
  1) Calculate $m_{\hat{x}'} = m_{\hat{x}} + 2^{\ell_x - 2} \pmod{2^{\ell_x}}$ and $m_{\hat{y}'} = m_{\hat{y}} + 2^{\ell_y - 2} \pmod{2^{\ell_y}}$;
  2) Calculate the total intermediate share $[u]_j \pmod{2^{\ell_b}}$:
  $$\begin{aligned}[u]_j = \; &C_x([r_{\hat{x}'}]_j + (j-1) \cdot m_{\hat{x}'}) + C_y([r_{\hat{y}'}]_j + (j-1) \cdot \\ &m_{\hat{y}'}) - C_x 2^{\ell_x}([w_x^{(0)} + r_{x,(0)}]_j + m_{x,(0)}[1 - r_{x,(0)}]_j) \\ &- C_y 2^{\ell_y}([w_y^{(0)} + r_{y,(0)}]_j + m_{y,(0)}[1 - r_{y,(0)}]_j) \\ &- (j-1) \cdot (C_x 2^{\ell_x - 2} + C_y 2^{\ell_y - 2});\end{aligned}$$
  3) Drop the lower ℓ_{a+r} bits: $[\hat{z}]_j = \mathsf{drop}([u]_j, \ell_{a+r}) \pmod{2^{\ell_z}}$;
  4) Reveal $m_{\hat{z}} = \mathsf{Open}([\hat{z}] - [r_{\hat{z}}])$ to obtain the final result $\hat{z}$.

Fig. 2: DFC-Addition protocol.

If P_0 computes $\gamma = ([r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2)([r_{\hat{y}'}]_1 + [r_{\hat{y}'}]_2) \pmod{2^{\ell_b}}$ and secret-shares it with P_1 and P_2, then P_1 and P_2 can directly compute the shared value $[\hat{x}'\hat{y}']$ over ring 2^{ℓ_b}. Specifically,

- P_1 computes $[C_{\mathsf{mult}} \cdot \hat{x}'\hat{y}']_1 = C_{\mathsf{mult}} \cdot (m_{\hat{x}'}[r_{\hat{y}'}]_1 + m_{\hat{y}'}[r_{\hat{x}'}]_1 + [\gamma]_1)$;
- P_2 computes $[C_{\mathsf{mult}} \cdot \hat{x}'\hat{y}']_2 = C_{\mathsf{mult}} \cdot (m_{\hat{x}'} m_{\hat{y}'} + m_{\hat{x}'}[r_{\hat{y}'}]_2 + m_{\hat{y}'}[r_{\hat{x}'}]_2 + [\gamma]_2)$.

For $C_{\mathsf{mult}} \cdot 2^{\ell_y} w_y \hat{x}'$, we have

$$\begin{aligned}w_y \hat{x}' &= (w_y^{(0)} + m_{\hat{y}',(0)} + r_{\hat{y}',(0)} - m_{\hat{y}',(0)} r_{\hat{y}',(0)})\hat{x}' \\ &= m_{\hat{y}',(0)}(1 - r_{\hat{y}',(0)}) m_{\hat{x}'} + (w_y^{(0)} + r_{\hat{y}',(0)}) m_{\hat{x}'} \\ &+ m_{\hat{y}',(0)}(1 - r_{\hat{y}',(0)})([r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2) \\ &+ (w_y^{(0)} + r_{\hat{y}',(0)})([r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2) \pmod{2^{\ell_b}}\end{aligned}$$

P_0 computes the values $r_{\hat{y}',(0)}$, $w_y^{(0)} + r_{\hat{y}',(0)}$, $(1 - r_{\hat{y}',(0)})([r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2)$, and $(w_y^{(0)} + r_{\hat{y}',(0)})([r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2)$, and secret shares these to P_1 and P_2, then P_1 and P_2 can directly compute the shared value $[w_y \hat{x}']$ over ring 2^{ℓ_b}. A similar method can be used to compute $[w_x \hat{y}']$.

For $C_{\mathsf{mult}} 2^{\ell_y - 2} \hat{x}'$, we have

$$\hat{x}' = m_{\hat{x}'} + [r_{\hat{x}'}]_1 + [r_{\hat{x}'}]_2$$

It can be directly calculated by P_1 and P_2. Similar, $C_{\mathsf{mult}} 2^{\ell_x - 2} \hat{y}'$ can also be obtained directly.

The wrap value can be decomposed as $w_{\hat{x}} = w_x^{(0)} + w_x^{(1)}$, where $w_x^{(0)} = \mathsf{wrap}([r_{\hat{x}'}]_0, [r_{\hat{x}'}]_1, 2^{\ell_x})$ is the wrap-bit of

$[r_{\hat{x}'}]_0 + [r_{\hat{x}'}]_1 \pmod{2^{\ell_x}}$, $w_x^{(1)} = \mathsf{wrap}(r_{\hat{x}'}, m_{\hat{x}'}, 2^{\ell_x})$ is the wrap-bit of $m_{\hat{x}'} + r_{\hat{x}'} \pmod{2^{\ell_x}}$.

For $\mathbf{2^{\ell_x+\ell_y-2}(w_x + w_y)}$, we have

$$w_y := w_y^{(0)} + m_{\hat{y}',(0)} + r_{\hat{y}',(0)} - m_{\hat{y}',(0)} r_{\hat{y}',(0)}$$

Since $r_{\hat{y}',(0)}, w_y^{(0)} + r_{\hat{y}',(0)}$ have already been shared by P_0 when calculating $w_y^{\hat{x}'}$, the value $[w_x] + [w_y]$ can be directly computed by P_1 and P_2.

For $\mathbf{2^{\ell_x+\ell_y} w_x \cdot w_y}$, we have

$$\begin{aligned} w_x w_y = {} & m_{\hat{x}',(0)} m_{\hat{y}',(0)} \cdot (1 - r_{\hat{x}',(0)})(1 - r_{\hat{y}',(0)}) \\ & + m_{\hat{x}',(0)} \cdot (1 - r_{\hat{x}',(0)})(w_y^{(0)} + r_{\hat{y}',(0)}) \\ & + m_{\hat{y}',(0)} \cdot (1 - r_{\hat{y}',(0)})(w_x^{(0)} + r_{\hat{x}',(0)}) \\ & + (w_x^{(0)} + r_{\hat{x}',(0)})(w_y^{(0)} + r_{\hat{y}',(0)}) \pmod{2^{\ell_b}}. \end{aligned}$$

Since the single bits $r_{i,(0)}$ and the combined terms $(w_i^{(0)} + r_{i,(0)})$ for $i \in \{x, y\}$ have already been shared by P_0 during the calculation of $w_y \hat{x}'$ and $w_x \hat{y}'$, P_0 only needs to further compute and secret-share the cross-products of these r-related terms: $(1-r_{\hat{x}',(0)})(1-r_{\hat{y}',(0)})$, $(1-r_{\hat{x}',(0)})(w_y^{(0)}+r_{\hat{y}',(0)})$, $(1-r_{\hat{y}',(0)})(w_x^{(0)}+r_{\hat{x}',(0)})$, and $(w_x^{(0)}+r_{\hat{x}',(0)})(w_y^{(0)}+r_{\hat{y}',(0)})$. With these pre-shared coefficients, P_1 and P_2 can directly compute the shared value $[w_x w_y]$ over $\mathbb{Z}_{2^{\ell_b}}$ by locally multiplying them with the reconstructed bits $m_{\hat{x}',(0)}$ and $m_{\hat{y}',(0)}$.

Put all together. In the offline phase, P_0 pre-computes and secret-shares the necessary r-related coefficients to enable P_1 and P_2 to locally evaluate the expanded product $\hat{x}\hat{y}$ over $\mathbb{Z}_{2^{\ell_b}}$. Specifically, P_0 distributes the shares of the following terms: 1). The cross-product mask γ, and the r-only coefficients for $w_y \hat{x}'$ and $w_x \hat{y}'$, such as $(1 - r_{\hat{i}',(0)})([r_{\hat{j}'}]_1 + [r_{\hat{j}'}]_2)$ and $(w_i^{(0)} + r_{\hat{i}',(0)})([r_{\hat{j}'}]_1 + [r_{\hat{j}'}]_2)$. 2). The cross-product terms of carry-bits, including $(1-r_{\hat{x}',(0)})(1-r_{\hat{y}',(0)})$, $(1 - r_{\hat{x}',(0)})(w_y^{(0)} + r_{\hat{y}',(0)})$, $(1 - r_{\hat{y}',(0)})(w_x^{(0)} + r_{\hat{x}',(0)})$, and $(w_x^{(0)} + r_{\hat{x}',(0)})(w_y^{(0)} + r_{\hat{y}',(0)})$. To minimize online overhead, P_0 consolidates all constant terms (those independent of $m_{\hat{x}'}$ and $m_{\hat{y}'}$) into a single offset share:

$$\begin{aligned} \Gamma = C_{\mathsf{mult}} \cdot \{ & \gamma - \sum_{i,j \in x,y, i \neq j} 2^{\ell_i} (w_i^{(0)} + r_{i,(0)})([r_{\hat{j}'}]_1 + [r_{\hat{j}'}]_2) \\ & - \sum_{i \in x,y} 2^{\ell_i - 2} ([r_{\hat{i}'}]_1 + [r_{\hat{i}'}]_2) + 2^{\ell_x + \ell_y - 2} \sum_{i \in x,y} (w_i^{(0)} + r_{i,(0)}) \\ & + 2^{\ell_x + \ell_y} (w_x^{(0)} + r_{x,(0)})(w_y^{(0)} + r_{y,(0)}) + 2^{\ell_x + \ell_y - 4} \} \pmod{2^{\ell_b}}. \end{aligned}$$

In the online phase, P_1 and P_2 first reconstruct the masked values $m_{\hat{x}'}$ and $m_{\hat{y}'}$. They then locally compute the linear combination of the pre-shared coefficients with the corresponding m-related terms (e.g., $m_{\hat{x}'} m_{\hat{y}'}$, $m_{\hat{i}'}$, and $m_{\hat{x}',(0)} m_{\hat{y}',(0)}$) to obtain the total intermediate shares $[u]_j \in \mathbb{Z}_{2^{\ell_b}}$. Finally, they perform the truncation $[\hat{z}]_j = \mathsf{drop}([u]_j, \ell_{a+r}) \pmod{2^{\ell_z}}$ and open the result to obtain the final product $\hat{z}$.

Communication Optimization and Generalization. As shown in the Figure 3, in the proposed DFC-Multiplication protocol, the online phase is designed to be succinct. Only a single round of communication is required for the computing parties to reconstruct the final result. Specifically, P_1 and P_2 exchange their intermediate shares to open the masked value $m_{\hat{z}}$ in the target ring $\mathbb{Z}_{2^{\ell_z}}$, resulting in a constant online communication cost of ℓ_z bits.

During the offline phase, P_0 distributes the secret shares of various r-related coefficients to P_1 and P_2. While a naive 3-PC implementation would necessitate sharing all coefficients over the full intermediate ring $\mathbb{Z}_{2^{\ell_b}}$, our protocol leverages the inherent power-of-two scaling factors in the expansion of $\hat{x}\hat{y}$ to perform *ring compression*. We categorize the offline communication into three tiers: 1). The consolidated constant

Protocol $\langle \hat{z} \rangle^{2^{\ell_z}} \leftarrow \Pi_{\mathsf{mult}}(\langle \hat{x} \rangle^{2^{\ell_x}}, \langle \hat{y} \rangle^{2^{\ell_y}})$

Common Input : $\ell_x, \ell_y, \ell_z, \ell_b, \ell_{a+r}$.
Input : $\langle \cdot \rangle^{2^{\ell_x}}$-shared $\hat{x}$ and $\langle \cdot \rangle^{2^{\ell_y}}$-shared $\hat{y}$.
Output : $\langle \cdot \rangle^{2^{\ell_z}}$-shared $\hat{z} = \lfloor C_{\mathsf{mult}} \hat{x}\hat{y} / 2^{\ell_{a+r}} \rfloor$.

Offline:

- P_0 calculates bits $r_{\hat{x}',(0)}, r_{\hat{y}',(0)}$ and local wrap-bits $w_x^{(0)}, w_y^{(0)}$ based on the shares $[r_{\hat{x}'}]$ and $[r_{\hat{y}'}]$;
- P_0 computes the following intermediate coefficients:
  - $\gamma_i = w_i^{(0)} + r_{i',(0)}$ for $i \in \{x, y\}$;
  - $\beta_{i,j} = (1 - r_{\hat{j}',(0)})([r_{\hat{i}'}]_1 + [r_{\hat{i}'}]_2)$ and $\alpha_{i,j} = \gamma_j \cdot r_{\hat{i}'}$ for $i, j \in \{x, y\}, i \neq j$;
  - $\theta_1 = (1 - r_{\hat{x}',(0)})(1 - r_{\hat{y}',(0)}), \theta_2 = (1 - r_{\hat{x}',(0)})\gamma_y, \theta_3 = (1 - r_{\hat{y}',(0)})\gamma_x, \theta_4 = \gamma_x \gamma_y$;
  - $\Gamma = C_{\mathsf{mult}} \cdot \Big[\gamma - \sum_{i \neq j} 2^{\ell_i} \alpha_{i,j} - \sum 2^{\ell_i - 2} r_{\hat{i}'} + 2^{\ell_x + \ell_y - 2} \sum \gamma_i + 2^{\ell_x + \ell_y} \theta_4 + 2^{\ell_x + \ell_y - 4}\Big]$, where $\gamma = r_{\hat{x}'} r_{\hat{y}'}$.
- P_0 distributes the calculated coefficients by invoking the sharing protocol $\Pi_{[\cdot]}$:
  - $[r_{\hat{x}',(0)}], [r_{\hat{y}',(0)}], [\gamma_x], [\gamma_y] \leftarrow \Pi_{[\cdot]}(\cdot)$;
  - $[\beta_{x,y}], [\beta_{y,x}], [\alpha_{x,y}], [\alpha_{y,x}] \leftarrow \Pi_{[\cdot]}(\cdot)$;
  - $[\theta_1], [\theta_2], [\theta_3], [\theta_4], [\Gamma] \leftarrow \Pi_{[\cdot]}^{\ell_b}(\cdot)$; *// over the intermediate ring $\mathbb{Z}_{2^{\ell_b}}$*
- All parties invoke $[r_{\hat{z}}] \leftarrow \mathcal{F}_{\mathsf{share}}[2^{\ell_z}]$.

Online:

- P_j, for $j \in \{1, 2\}$ does:
  1) Compute $m_{\hat{x}'} = m_{\hat{x}'} + 2^{\ell_x - 2} \pmod{2^{\ell_x}}$ and $m_{\hat{y}'} = m_{\hat{y}'} + 2^{\ell_y - 2} \pmod{2^{\ell_x}}$, and extract LSBs $m_{x,(0)}, m_{y,(0)}$;
  2) Compute intermediate share $[u]$ in $\mathbb{Z}_{2^{\ell_b}}$:

$$\begin{aligned} [u] = {} & C_{\mathsf{mult}} \cdot \big(m_{\hat{x}'} m_{\hat{y}'} + m_{\hat{x}'} [r_{\hat{y}'}] + m_{\hat{y}'} [r_{\hat{x}'}] + [\Gamma] \\ & - 2^{\ell_y} (m_{y,(0)} (1 - r_{y,(0)}) m_{\hat{x}'} + m_{y,(0)} [\beta_{x,y}] \\ & + m_{\hat{x}'} [\gamma_y] + [\alpha_{x,y}]) - 2^{\ell_x} (m_{x,(0)} (1 - r_{x,(0)}) m_{\hat{y}'} \\ & + m_{x,(0)} [\beta_{y,x}] + m_{\hat{y}'} [\gamma_x] + [\alpha_{y,x}]) \\ & - 2^{\ell_y - 2} (m_{\hat{x}'} + [r_{\hat{x}'}]) - 2^{\ell_x - 2} (m_{\hat{y}'} + [r_{\hat{y}'}]) \\ & + 2^{\ell_x + \ell_y - 2} ([\gamma_x] + m_{x,(0)} (1 - r_{x,(0)}) + [\gamma_y] \\ & + m_{y,(0)} (1 - r_{y,(0)})) + 2^{\ell_x + \ell_y} (m_{x,(0)} m_{y,(0)} [\theta_1] \\ & + m_{x,(0)} [\theta_2] + m_{y,(0)} [\theta_3] + [\theta_4])) \pmod{2^{\ell_b}} \end{aligned}$$

  3) $[\hat{z}] = \mathsf{drop}([u], \ell_{a+r}) \pmod{2^{\ell_z}}$.
  4) Reveal $[z] - [r_z]$ to get m_z;

Fig. 3: DFC-Multiplication Protocol.

Γ and the cross-product mask γ are independent of input-specific power-of-two scales and thus must be shared over the full intermediate ring $\mathbb{Z}_{2^{\ell_b}}$. The cost for distributing these two coefficients to P_1 and P_2 is $4\ell_b$ bits. 2). The coefficients associated with $2^{\ell_y} w_y \hat{x}'$ and $2^{\ell_x} w_x \hat{y}'$ (e.g., $r_{\hat{y}',(0)}$ and its interactions) are scaled by 2^{ℓ_y} and 2^{ℓ_x} respectively. These can be effectively shared over reduced rings $\mathbb{Z}_{2^{\ell_b-\ell_y}}$ and $\mathbb{Z}_{2^{\ell_b-\ell_x}}$. The communication cost for these first-order correction terms is $4\ell_b - 2(\ell_x + \ell_y)$ bits. 3). The terms required for the high-order product $2^{\ell_x+\ell_y} w_x w_y$, such as $(1 - r_{\hat{x}',(0)})(1 - r_{\hat{y}',(0)})$, are scaled by $2^{\ell_x+\ell_y}$. Consequently, these four coefficients are secret-shared over the minimal ring $\mathbb{Z}_{2^{\ell_b-(\ell_x+\ell_y)}}$. The corresponding cost is $8\ell_b - 8(\ell_x + \ell_y)$ bits.

During the online execution, P_1 and P_2 treat these small-ring shares as elements of $\mathbb{Z}_{2^{\ell_b}}$. Any potential wrap-bit errors introduced by this ring extension are mathematically eliminated because the subsequent multiplication by the respective power-of-two scales shifts the errors out of the ring modulus 2^{ℓ_b}. Unlike standard fixed-point protocols that often discard high-order terms (e.g., $2^{2\ell} w_x w_y$) to simplify calculation, our DFC protocol preserves these terms to support arbitrary target rings $\mathbb{Z}_{2^{\ell_z}}$. By employing scale-aware compression, the total offline communication cost is optimized as $\ell_b + 4(2\ell_b - \ell_x - \ell_y) + 8(\ell_b - \ell_x - \ell_y)$ bits. The communication burden of the DFC framework dynamically decreases as the precision of input the dynamic fixed-point representation (ℓ_x, ℓ_y) increases relative to the intermediate ring width ℓ_b.

Based on the above, we can easily implement the inner product operation. Given secret-shared inputs $\{\langle \hat{x}_i \rangle^{2^{\ell_{x,i}}}, \langle \hat{y}_i \rangle^{2^{\ell_{y,i}}}\}_{i=1}^n$ and public transition coefficients C_i, the goal is to compute $\langle \hat{z} \rangle^{2^{\ell_z}} \leftarrow \Pi_{\mathsf{poly}}^{\mathsf{agg}}(\{\langle \hat{x}_i \rangle, \langle \hat{y}_i \rangle, C_i\}_{i=1}^n)$, where $\hat{z} = \mathsf{drop}\left(\sum_{i=1}^n C_i \hat{x}_i \hat{y}_i \mod 2^{\ell_b}, \ell_a + \ell_r\right) \pmod{2^{\ell_z}}$. As in Π_{mult}, each input is first shifted in its native ring, we then have $\hat{x}'_i = \hat{x}_i + 2^{\ell_{x,i}-2}$, $\hat{y}'_i = \hat{y}_i + 2^{\ell_{y,i}-2}$. When reinterpreted over $\mathbb{Z}_{2^{\ell_b}}$, we have $\hat{x}_i = \hat{x}'_i - 2^{\ell_{x,i}} w_{x,i} - 2^{\ell_{x,i}-2} \pmod{2^{\ell_b}}$, and $\hat{y}_i = \hat{y}'_i - 2^{\ell_{y,i}} w_{y,i} - 2^{\ell_{y,i}-2} \pmod{2^{\ell_b}}$, where $w_{x,i}$ and $w_{y,i}$ are the corresponding wrap values. Therefore,

$$\begin{aligned}\sum_{i=1}^{n} C_i \hat{x}_i \hat{y}_i = &\sum_{i=1}^{n} C_i \{\hat{x}'_i \hat{y}'_i - 2^{\ell_{y,i}} w_{y,i} \hat{x}'_i - 2^{\ell_{x,i}} w_{x,i} \hat{y}'_i \\ &- 2^{\ell_{y,i}-2} \hat{x}'_i - 2^{\ell_{x,i}-2} \hat{y}'_i \\ &+ 2^{\ell_{x,i}+\ell_{y,i}-2}(w_{x,i} + w_{y,i}) \\ &+ 2^{\ell_{x,i}+\ell_{y,i}} w_{x,i} w_{y,i} + 2^{\ell_{x,i}+\ell_{y,i}-4}\} \pmod{2^{\ell_b}}\end{aligned}$$

Instead of independent Π_{mult} calls, P_0 pre-computes and fuses all r-only correction offsets into a single global share: $[\Gamma_{\mathsf{sum}}] = \mathsf{drop}(\sum C_i[\Gamma_i] \pmod{2^{\ell_b}}, \ell_{a+r}) \pmod{2^{\ell_z}}$.

Each summand follows the same correction structure as the scalar Π_{mult}, but is weighted by a public coefficient C_i and aggregated before extraction. Instead of invoking Π_{mult} per term, P_0 pre-computes all r-only corrections and combines them into a global share $[\Gamma_{\mathsf{sum}}] = \sum_{i=1}^n C_i[\Gamma_i] \bmod 2^{\ell_b}$. In the online phase, P_1 and P_2 reconstruct all masked values in one round, and locally evaluate the remaining products using public masked inputs and pre-shared randomness. Summing all contributions with $[\Gamma_{\mathsf{sum}}]$ yields shares $[u]^{2^{\ell_b}}$ of $u = \sum_{i=1}^n C_i \hat{x}_i \hat{y}_i$.

Communication Cost. In the online phase, the protocol achieves a single-round communication complexity by reconstructing all masked values $\{m_{\hat{x}'i}, m_{\hat{y}'i}\}_{i=1}^n$ in one interaction. While the round complexity remains constant, the total communication scales linearly with the vector dimension n, incurring $O(\sum_{i=1}^n(\ell_{x,i}+\ell_{y,i}))+\ell_z$ bits. In the offline phase, by leveraging correction fusion and a three-tier ring compression strategy, the assisting party P_0 minimizes bandwidth based on power-of-two scaling factors: Tier 1 (Full Ring): The consolidated constant Γ and cross-product masks γ are shared over $\mathbb{Z}_{2^{\ell_b}}$, costing $2n\ell_b + 2\ell_b$ bits. Tier 2 (Reduced Ring): Coefficients for first-order corrections (e.g., $\beta_{i,j}, \alpha_{i,j}$) are shared over compressed rings, costing $\sum_{i=1}^n[4\ell_b - 2(\ell_{x,i} + \ell_{y,i})]$ bits. Tier 3 (Minimal Ring): High-order terms (e.g., $\theta_{1\ldots4}$) are shared over the smallest rings, costing $\sum_{i=1}^n[8\ell_b - 8(\ell_{x,i} + \ell_{y,i})]$ bits.Using these optimizations, the total offline communication overhead is strictly bounded by Offline Comm $= (14n + 2)\ell_b - 10\sum_{i=1}^n(\ell_{x,i} + \ell_{y,i})$ bits.

*3) **Polynomial Evaluation (Π_{poly})**:* Building upon the DFC addition and multiplication primitives, the framework naturally extends to the efficient evaluation of high-degree polynomials $f(x) = \sum_{i=0}^k a_i x^i$. The core approach employs a recursive power generation and fused extraction paradigm: first, each monomial power $\hat{x}^i$ is recursively computed via Π_{mult}, where local alignment is performed at each step to prevent bit-width explosion within the intermediate ring $\mathbb{Z}_{2^{\ell_b}}$. Subsequently, the public coefficients a_i are utilized to pre-compute transition weights C_i that map each monomial to a unified precision plane determined by the target scale s_z and resolution factor s_r. The final result is obtained through a single weighted summation and global extraction: $\hat{z} = \mathsf{drop}\left(\sum_{i=0}^k (C_i \cdot \hat{x}^i) \mod 2^{\ell_b}, \ell_a + \ell_r\right)$. This fused processing architecture effectively circumvents the accumulation of dynamic fixed-point representation noise associated with independent term-wise truncation, providing a high-precision, low-communication foundation for approximating non-linear functions. The key term for Π_{poly} can be found in Appendix VIII-B.

Privacy. The privacy of our protocol can be intuitively verified. In the offline phase, all values transmitted by P_0 to P_1 and P_2 are distributed as secret shares. Given our non-collusion setting with at most one static corrupted party, these shares appear uniformly random to P_1 and P_2. During the online phase, the only information revealed to P_1 and P_2 is the opened masked value $m_z = z - r$. By virtue of the random mask r, which acts as a one-time pad, m_z perfectly hides all information regarding the underlying z.

IV. Benchmark

In this section, we benchmark our dynamic fixed-point protocols. We compare our dynamic fixed-point protocol against typical fixed-point solutions and float-point solutions, as shown in Table I.

TABLE I: Comparison baselines for RingAdaptive evaluation.

Method	Type	Scenario	Key Metrics
ASTRA [29]	fixed-point	Π_{mult}, inner product	Online latency, total time (MAN/WAN)
Secfloat [36]	Floating-point	exp	Latency, communication
BOLT [37]	Fixed-point	exp	Latency, Communication
Nexus [38]	FHE	exp	Latency
Lu et al. [?]	Quantized LUT	Inner product	Latency, Communication

A. Benchmark Setting

We implement our method DYNAFIXin C++, leveraging the EMP toolkit[1] [39] to facilitate communication between the participating parties. Our benchmark code is available at [28]. For our experiments, the benchmark is performed on a rack server equipped with an Intel(R) Xeon(R) Gold 6238R CPU @ 2.20GHz and 377 GB of RAM. All external libraries are compiled using GCC (version 11.4.0) on Ubuntu 20.04. To evaluate the performance under various network conditions, our experiments are conducted in a local area network (LAN) using Linux traffic control (tc) to simulate the following two network settings: Metropolitan-area setting (MAN): 256 Mbps bandwidth with 2 ms round-trip latency (RTT). Wide-area setting (WAN): 100 Mbps bandwidth with 20 ms round-trip latency (RTT).

B. Benchmark Results

Performance of Dynamic Fixed-Point Inner product. We conducted a comprehensive performance evaluation of our dynamic fixed-point inner product operation. We compared our protocol against SOTA approaches (ASTRA [29]) across varying data volumes (from 2^{16} to 2^{24}) and different bit-widths ($\ell_b \in \{16, 24, 32\}$), and ASTRA [29] needs the different bit-widths ($\ell_b \in \{47, 55, 62\}$). We provide a detailed analysis and the rationale for these ring-size selections in Appendix VIII-D. The experiments were carried out in both MAN and WAN settings. The detailed experimental results are summarized in Table II. As Table II shows, our protocol demonstrates substantial acceleration across all data scales, with performance improvements ranging from 1.4× to 17.3×. In the MAN setting, at a batch size of 2^{24}, the SOTA scheme requires 122.20s, whereas our optimized configuration ($\ell_b = 16$) completes the same task in only 7.07 ms, achieving a 17.3× speedup. While SOTA suffers from a drastic increase in execution time due to the ball fixed rings in large computation spaces, our protocol maintains excellent scalability. This proves that our fused multiplication-accumulation design effectively mitigates the computational burden associated with high-precision intermediate states. The performance advantage remains robust in WAN settings, where our protocol consistently maintains a lead of 1.4× to 11.4×. At the 2^{24} scale, our protocol significantly reduces total latency from 156.28 s to 13.65 s. For more details, see Figure 5. By merging the inner product and truncation into a single drop operation, our protocol markedly reduces sensitivity to network RTT, making it highly suitable for geographically distributed privacy-preserving computation scenarios. We also evaluated the performance of Π_{mult}, the detailed results are provided in the Appendix VIII-C.

[1]https://github.com/emp-toolkit/emp-tool

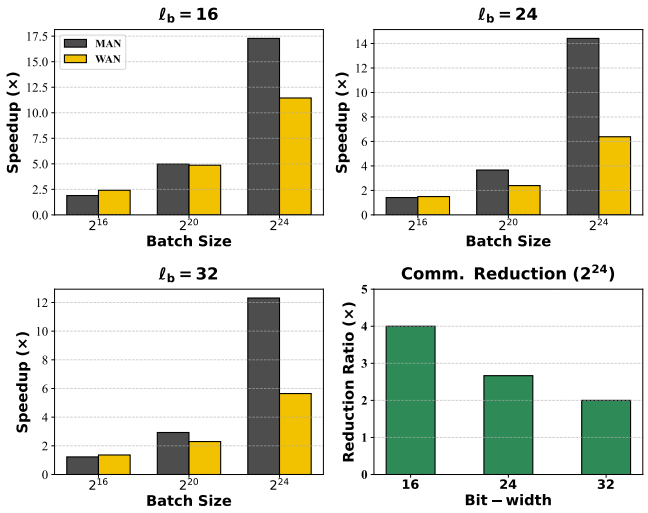

Fig. 4: Performance speedup and communication reduction. The ■ indicates the MAN setting speedup. The ■ indicates the WAN setting speedup. The ■ indicates the communication reduction ratio.

Performance of Dynamic Fixed-Point Polynomial (Exponential). We examine the error convergence of DFC by comparing it against high-precision plaintext baselines and theoretical limits across various configurations. Figure 5 illustrates the absolute error of DFC relative to the plaintext exponential function, showing that the fixed-point implementation nearly matches the theoretical accuracy. To further investigate the impact of bit-width allocation, Table III provides a detailed accuracy analysis across different polynomial degrees and input scales. In the experimental range of $x \in [0, 2]$, as shown

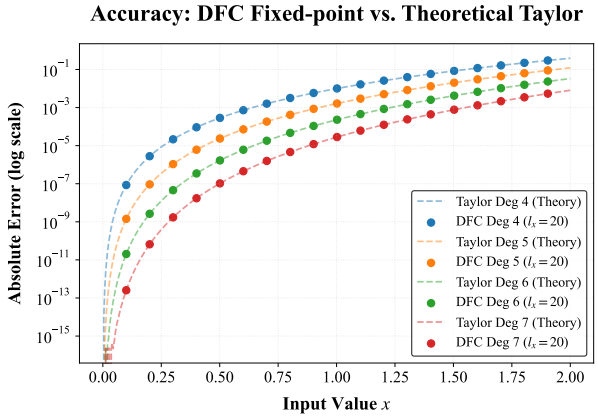

Fig. 5: Performance speedup and communication reduction.

in Table III, the results distinctly demonstrate a Precision-Scale Decoupling effect. As the input scale ℓ_x increases from 16 to 32 bits, the maximum absolute error remains nearly constant

TABLE II: Test Results from Dynamic Fixed Point Inner Product (Running Time in ms, Communication in bytes). The SOTA denotes ASTRA [29]. And the method requires different ring sizes for different batch sizes. For example, when the batch size is $2^{16}/2^{20}/2^{24}$, the corresponding ring size is $\ell_{\mathsf{size}} = 47/55/62$. We provide a detailed explanation in Appendix VIII-D.

ℓ_{size}	Protocol	MAN setting			WAN setting			Communication		
	(Test batch)	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
$(47, 55, 62)$	SOTA	63.46	2021.61	122203.83	195.12	4586.42	156284.25	524288	8388608	134217728
16-bits	Ours^{+}	33.49	406.13	7070.84	80.96	942.39	13652.54	174385	2440322	33554432
24-bits	Ours*	44.80	551.62	8475.49	130.45	1917.1	24484.8	267721	3660483	50331648
32-bits	Ours*	51.71	689.8	9926.44	143.21	1994.94	27677.37	356062	4880644	67092480

across all approximation degrees. This empirical evidence confirms that the DFC protocol effectively eliminates the quantization noise bottleneck inherent in conventional fixed-point arithmetic. In practical privacy-preserving computation scenarios, such as MPC implemented via the EMP toolkit, this characteristic allows the system to operate under low-bandwidth configurations (e.g., $\ell_x = 16$ and $\ell_z = \ell_b - \ell_a + \ell_r$) while maintaining high-precision accuracy. Consequently, DFC can significantly reduce the communication complexity of MPC protocols without sacrificing numerical integrity. Such

TABLE III: Accuracy Analysis of DFC-based Exponential Approximation ($x \in [0, 2]$)

Degree	Input Scale (ℓ_x)	Alignment Bits (ℓ_a)	Max Error
x^4	16	40	3.89×10^{-1}
	20	36	3.89×10^{-1}
	24	32	3.89×10^{-1}
	32	24	3.89×10^{-1}
x^5	16	40	1.22×10^{-1}
	20	36	1.22×10^{-1}
	24	32	1.22×10^{-1}
	32	24	1.22×10^{-1}
x^6	16	40	3.35×10^{-2}
	20	36	3.35×10^{-2}
	24	32	3.35×10^{-2}
	32	24	3.35×10^{-2}
x^7	16	40	8.11×10^{-3}
	20	36	8.10×10^{-3}
	24	32	8.10×10^{-3}
	32	24	8.10×10^{-3}

robustness originates from DFC's dynamic extraction window compensation mechanism. Even in configurations with a narrow ℓ_x, DFC flexibly utilizes the remaining alignment bits ℓ_a within the ℓ_b-bit ring space. This ensures that infinitesimal Taylor coefficients (e.g., $1/5040$) are preserved during the quantization process, preventing them from vanishing due to fixed-point truncation. As a result, the computational error is predominantly bounded by the mathematical truncation error of the Taylor series itself rather than numerical noise. Our findings indicate that DFC achieves a near-optimal fit to the theoretical mathematical limits, providing a lossless conversion scheme for precision-sensitive operators like Softmax in Transformer models.

We also evaluate our DFC-based exp operator against three representative paradigms: Secfloat [36] (floating-point MPC), BOLT [37] (fixed-point MPC), and Nexus [38] (FHE-based approach using $n = 7$ limit approximation)[2], while maintaining a precision error within $1 \sim 2 \times 10^{-3}$. As demonstrated in Table IV, our protocol achieves substantial efficiency breakthroughs. In a MAN setting, our latency of 817.61 ms represents a 24.1× and 11.3× speedup over Secfloat and Nexus, respectively. The advantage of DFC is even more pronounced in bandwidth-constrained WAN environments; specifically, our communication overhead is only 832 MB, which is a 131× reduction compared to BOLT and 57.8× over Secfloat. By bypassing the expensive bit-decomposition costs of floating-point arithmetic and the heavy rotation overhead of FHE, DFC strikes an optimal balance between performance and accuracy for privacy-preserving Transformer inference.

TABLE IV: Performance analysis of exp (Running Time in ms, Communication in MB).

Method	Running Time		Communication
	MAN setting	WAN setting	-
Secfloat [36]	19777.69	52319.67	48090
BOLT [37]	18842.11	47702.14	109405
Nexus [38]	9288	9288	-
DynaFix	817.61	7971.43	832

Performance Benchmarking Against SOTA Quantization Frameworks. To evaluate the effectiveness of our dynamic fixed-point design, we benchmark inner product, the most computationally intensive component in PPML, against the SOTA quantized paradigm (Lu et al. [4]), which relies on LUTs. Based on Table V, our protocol consistently improves efficiency over the SOTA baseline across different precision and batch settings. In particular, it achieves up to 17.1× speedup in the MAN setting and up to 11.1× speedup in the WAN setting, especially under large-batch configurations where communication latency becomes dominant. Notably, their approach also suffers from the growth of the inner product's output space, necessitating a larger ring to accommodate the results as the dimensionality increases(Cf. Appendix. VIII-D). Meanwhile, communication cost is reduced by approximately 3×–4× compared to the baseline, mainly due to eliminating LUT-heavy data movement. These results demonstrate that our fused multiplication-accumulation design

[2]Nexus approximates the exponential function using the limit definition: $\exp(x) \approx \left(1 + \frac{x}{n}\right)^n$. We empirically evaluate the approximation error for $x \in [0, 2]$ with $n = 7$. The maximum absolute error in this interval is measured as 2.120×10^{-3}

TABLE V: Compare the dynamic Fixed Point Inner Product with Quantized Computation (Running Time in ms, Communication in bytes). For DYNAFIX, the input x is represented using $\ell_x = 8$ bits, while the computation is performed with $\ell_b = 16$ bits.

ℓ_{size}	Protocol (Test batch)	MAN setting			WAN setting			Communication		
		2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
8-bits	Lu et al., [4]	12.28	277.93	8194.43	66.84	1031.33	21737.40	3145728	50331648	1073741824
16-bits	Ours^{+}	11.26	33.49	406.13	28.79	80.96	942.39	10899	174385	2440322
24-bits	Ours*	14.45	44.80	551.62	37.34	130.45	1917.1	16733	267721	3660483
32-bits	Ours*	17.15	51.71	689.8	47.88	143.21	1994.94	22253	356062	4880644

effectively alleviates both memory and communication bottlenecks in dynamic fixed-point computation while maintaining strong scalability with increasing tensor dimensions.

V. RELATED WORK

Most prior work on fixed-point MPC [14], [8], [16], [17], [18], [19], [35] follows the standard multiply-then-truncate paradigm. To improve performance, some schemes use probabilistic truncation, which in turn requires zero-padding of significant bits to avoid overflows. Recent work instead focuses on architectural improvements. Dong *et al.* [20] propose a specialized two-party truncation protocol that enables a more compact ring representation at the cost of additional oblivious transfers. Liu *et al.* [21] design efficient truncation in the Shamir secret sharing setting, but their protocol is restricted to Mersenne prime fields. Using ring extensions, Dalskov *et al.* [40] and Lu *et al.* [22] develop efficient fixed-point frameworks. For a fixed-point multiplication with ℓ-bit precision and ℓ_f fractional bits, Dalskov's protocol uses 3 online rounds and 7ℓ bits of communication (2 rounds and 4ℓ bits for truncation, plus 1 round and 3ℓ bits for multiplication). In contrast, Lu *et al.* [22] achieve a 1-round online phase with 2ℓ bits by moving $6\ell_f+\ell$ bits to a single offline round, for a total of $3\ell+6\ell_f$ bits. Research on privacy-preserving floating-point (FP) arithmetic has progressed through several paradigms. Aliasgari *et al.* [23] first realized 3PC FP evaluation, but their bit-extraction-based design incurred high round complexity—over 34 rounds for FP32. This bit-level method was later adapted to 2PC [41] and strengthened against malicious adversaries [24]. In parallel, hybrid 2PC protocols emerged [42], [43], using secret sharing for linear FP operations and Garbled Circuits for non-linear ones (e.g., shifts). Sasaki *et al.* [26] refined this by combining GMW with Boolean secret sharing. Rathee *et al.* [25] then built a comprehensive 2PC FP library with many transcendental functions (e.g., e^x, $\log x$, $\sin x$), later used for secure model training [27], substantially improving efficiency and precision. Lu *et al.* [22] recently proposed an efficient constant-round FP protocol, achieving over $4\times$ speedup versus prior work. Nonetheless, their benchmarks still show FP operations are more than $100\times$ slower than fixed-point at the same bit-width. To our knowledge, no existing MPC scheme over real numbers simultaneously attains the speed of fixed-point arithmetic and the wide dynamic range of floating-point.

VI. DISCUSSION

In our 3-PC protocol design, the transition coefficients C_i within the DFC framework are currently configured as public values. However, the protocol can be further evolved to treat these coefficients as secret shares.

Protocol Feasibility: Transitioning C_i from Public to Secret. Transitioning the transition coefficients C_i from public scalars to secret shares is architecturally feasible within our framework. Once C_i is treated as a sensitive share, the operations previously executed as local scalar multiplications (e.g., $C_i \cdot \hat{x}$) evolve into interactive multiplication protocols between private shares. While this shift necessitates additional communication rounds to align scaling factors, our proposed Poly Protocol effectively supports this transition. Specifically, by leveraging high-order primitives designed for tri-input terms (e.g., $C_i \cdot \hat{x} \cdot \hat{y}$), which are computationally equivalent to degree-3 monomials like $x^2 \cdot x$, we can consolidate the complex coefficient alignment logic into a single-round polynomial evaluation.

Implementation Challenges: Bit-width Derivation. Parameter Derivation and Overflow Management. The primary challenge in secret-sharing C_i lies in the inherent uncertainty of parameter derivation within the intermediate ring, necessitating robust overflow detection protocols. When C_i is public, the bit-width of the intermediate ring $\mathbb{Z}_{2^{\ell_b}}$ can be precisely calculated based on its exact magnitude to optimize efficiency. Conversely, when C_i is hidden via secret sharing, the computing parties lack direct perception of the coefficient's scale. To strictly prevent overflows during dynamic scaling or high-order accumulation, the protocol must pre-allocate a more conservative bit-width based on a prior safety upper bound of C_i. Furthermore, the subsequent drop must be upgraded from a simple public right-shift to a sophisticated secret-shift protocol.

VII. CONCLUSION

The proposed DYNAFIX framework introduces a dynamic fixed-point encoding mechanism that successfully balances high arithmetic performance with a wide numerical dynamic range in privacy-preserving computation. Under the 3-PC protocol, it significantly reduces communication latency, achieving an online multiplication performance up to $17.3\times$ faster than ASTRA [29] in inner product tasks, while its computational precision and performance for complex operators like exponential functions far exceed existing floating-point schemes such as SecFloat [36] and FHE-based schemes like Nexus [38].

Ethics Considerations

The primary motivation of DynaFixis to enhance data privacy in large-scale numerical computations. By bridging the performance gap between efficient fixed-point arithmetic and flexible floating-point methods, our work facilitates the deployment of privacy-preserving solutions in domains that require high precision and wide dynamic ranges, such as healthcare and financial auditing. We acknowledge that any privacy-enhancing technology could theoretically be repurposed to obfuscate malicious activities. However, the core of our contribution is a fundamental cryptographic primitive designed to protect legitimate user data from unauthorized exposure. All experiments conducted in this study utilized either synthetic data or publicly available, de-identified benchmarking datasets, ensuring no individual privacy was compromised during the research process.

References

[1] A. C. Yao, "Protocols for secure computations," in *SFCS*, 1982.

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC*, 1987.

[3] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991.

[4] T. Lu, B. Zhang, X. Zhang, and K. Ren, "A new PPML paradigm for quantized models," in *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/a-new-ppml-paradigm-for-quantized-models/

[5] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "Flash: Fast and robust framework for privacy-preserving machine learning," in *PoPETs*, 2020.

[6] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," in *PoPETs*, 2021.

[7] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *ASIACCS*, 2018.

[8] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *CCS*, 2018.

[9] A. Patra and A. Suresh, "BLAZE: blazing fast privacy-preserving machine learning," in *NDSS*, 2020.

[10] W. Henecka, S. K ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Tasty: Tool for automating secure two-party computations," in *CCS*, 2010.

[11] K. Ji, B. Zhang, T. Lu, L. Li, and K. Ren, "Uc secure private branching program and decision tree evaluation," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2836–2848, 2023.

[12] T. Lu, B. Zhang, H. Li, and K. Ren, "A new paradigm for privacy-preserving decision tree evaluation," Cryptology ePrint Archive, Paper 2025/1447, 2025. [Online]. Available: https://eprint.iacr.org/2025/1447

[13] C. Wang, K. Ren, J. Wang, and Q. Wang, "Harnessing the cloud for securely outsourcing large-scale systems of linear equations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1172–1181, 2013.

[14] M. Payman and Z. Yupeng, "Secureml: A system for scalable privacy-preserving machine learning," in *S&P*, 2017.

[15] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 4961–4973. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/2754518221cfbc8d25c13a06a4cb8421-Paper.pdf

[16] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the gpu," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1021–1038.

[17] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko, "Motion–a framework for mixed-protocol multi-party computation," *ACM Transactions on Privacy and Security*, vol. 25, no. 2, pp. 1–35, 2022.

[18] Y. Chen, Z. Tang, T. Lu, B. Zhang, Z. Shi, and Z. Wang, "Accelerating private large transformers inference through fine-grained collaborative computation," *IEEE Transactions on Information Forensics and Security*, 2025.

[19] T. Lu, B. Zhang, L. Li, and K. Ren, " The Communication-Friendly Privacy-Preserving Machine Learning against Malicious Adversaries ," *IEEE Transactions on Dependable and Secure Computing*, no. 01, pp. 1–14, Oct. 5555. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TDSC.2025.3625999

[20] Y. Dong, W. jie Lu, X. Hou, K. Yang, and J. Liu, "M&m: Secure two-party machine learning through efficient modulus conversion and mixed-mode protocols," Cryptology ePrint Archive, Paper 2025/1551, 2025. [Online]. Available: https://eprint.iacr.org/2025/1551

[21] F. Liu, X. Xie, and Y. Yu, "Scalable multi-party computation protocols for machine learning in the honest-majority setting," in *USENIX Security 2024: 33rd USENIX Security Symposium*, D. Balzarotti and W. Xu, Eds. Philadelphia, PA, USA: USENIX Association, Aug. 14–16, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/liu-fengrun

[22] T. Lu, B. Zhang, Y. Feng, and K. Ren, "Secure computation for fixed-point and floating-point arithmetic," Cryptology ePrint Archive, Paper 2026/102, 2026. [Online]. Available: https://eprint.iacr.org/2026/102

[23] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele, "Secure computation on floating point numbers," *NDSS*, 2013.

[24] M. Aliasgari, M. Blanton, and F. Bayatbabolghani, "Secure computation of hidden markov models and secure floating-point arithmetic in the malicious model," *International Journal of Information Security*, vol. 16, no. 6, pp. 577–601, 2017.

[25] D. Rathee, A. Bhattacharya, D. Gupta, R. Sharma, and D. Song, "Secure {Floating-Point} training," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6329–6346.

[26] K. Sasaki and K. Nuida, "Efficiency and accuracy improvements of secure floating-point addition over secret sharing," in *IWSEC 20: 15th International Workshop on Security, Advances in Information and Computer Security*, ser. Lecture Notes in Computer Science, K. Aoki and A. Kanaoka, Eds., vol. 12231. Fukui, Japan: Springer, Cham, Switzerland, Sep. 2–4, 2020, pp. 77–94.

[27] D. Rathee, A. Bhattacharya, D. Gupta, R. Sharma, and D. Song, "Secure floating-point training," in *USENIX Security 2023: 32nd USENIX Security Symposium*, J. A. Calandrino and C. Troncoso, Eds. Anaheim, CA, USA: USENIX Association, Aug. 9–11, 2023, pp. 6329–6346. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/rathee

[28] "Dynafix," https://anonymous.4open.science/r/dynamfix_anony, May 2026.

[29] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "Astra: High throughput 3pc over rings with application to secure prediction," in *CCSW*, 2019.

[30] L. Yehuda and N. Ariel, "A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority," in *CCS*, 2017.

[31] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *EUROCRYPT*, 2011.

[32] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High throughput 3PC over rings with application to secure prediction," Cryptology ePrint Archive, Report 2019/429, 2019. [Online]. Available: https://eprint.iacr.org/2019/429

[33] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 597–616.

[34] M. Keller and K. Sun, "Secure quantized training for deep learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 10 912–10 938.

[35] L. Zhou, B. Zhang, Z. Wang, T. Lu, Q. Song, S. Zhang, H. Cui, and Y. Yu, "On probabilistic truncation in privacy-preserving machine learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 21, pp. 22 955–22 964, Apr. 2025. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/34458

[36] D. Rathee, A. Bhattacharya, R. Sharma, D. Gupta, N. Chandran, and A. Rastogi, "Secfloat: Accurate floating-point meets secure 2-party

computation," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 576–595.
[37] Q. Pang, J. Zhu, H. Mollering, Z. Wenting, and T. Schneider, "Bolt: Privacy-preserving, accurate and efficient inference for transformers," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.
[38] J. Zhang, J. Liu, X. Yang, Y. Wang, K. Chen, X. Hou, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," in *NDSS*, 2025.
[39] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.
[40] A. Barak, D. E. Escudero, A. Dalskov, and M. Keller, "Secure evaluation of quantized neural networks," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, pp. 355 – 375, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:75136732
[41] M. Aliasgari and M. Blanton, "Secure computation of hidden markov models," in *2013 International Conference on Security and Cryptography (SECRYPT)*, 2013, pp. 1–12.
[42] P. Pullonen and S. Siim, "Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations," in *FC 2015 Workshops*, ser. Lecture Notes in Computer Science, M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds., vol. 8976. San Juan, Puerto Rico: Springer Berlin Heidelberg, Germany, Jan. 30, 2015, pp. 172–183.
[43] B. Hemenway, S. Lu, R. Ostrovsky, and W. Welser, IV, "High-precision secure computation of satellite collision probabilities," in *SCN 16: 10th International Conference on Security in Communication Networks*, ser. Lecture Notes in Computer Science, V. Zikas and R. De Prisco, Eds., vol. 9841. Amalfi, Italy: Springer, Cham, Switzerland, Aug. 31 – Sep. 2, 2016, pp. 169–187.

VIII. Appendix

A. Case Study.

Atomic Scalar Multiplication *($n = 1$).* Consider $x = 6.5$ ($s_x = 16, \hat{x} = 104 \in \mathbb{Z}_{2^8}$) and $y = 4.2$ ($s_y = 8, \hat{y} = 33 \in \mathbb{Z}_{2^6}$). The natural product scale would be $s_x \cdot s_y = 128$. However, we pre-assign a target scale $s_z = 32$ and $\ell_z = 10$. To handle the transition $s_z/(s_x \cdot s_y) = 32/128 = 0.25$, we set $s_r = 2^2 = 4$ ($\ell_r = 2$) for precision alignment. First, we calculate the integer transition coefficient as $C_{\mathsf{mult}} = \lfloor \frac{s_z}{s_x \cdot s_y} \cdot s_r \rfloor = \lfloor \frac{32}{128} \cdot 4 \rfloor = 1$. For our parameters ($\ell_x = 8, \ell_y = 6, C_{\mathsf{mult}} = 1$), the safe intermediate bit-width is $\ell_b = 14$. In this specific instance, the product $u = C_{\mathsf{mult}} \cdot (\hat{x} \cdot \hat{y}) = 1 \cdot (104 \cdot 33) = 3432$. Although the theoretical bound ℓ_b is 14 bits, the value u is effectively contained within 12 bits since $2^{11} < 3432 < 2^{12}$. Following the QFC rule, the calculation is performed in the intermediate ring $\mathbb{Z}_{2^{14}}$, we then have $u = (1 \cdot (104 \cdot 33)) \pmod{2^{14}} = 3432$. Since u represents the product at an effective scale of $s_x \cdot s_y \cdot (1/C_{\mathsf{mult}})$ (more precisely, u is already at the scale $s_z \cdot s_r$), we drop $\ell_r = 2$ bits (assuming $\ell_a = 0$ for simplicity):

$$\hat{z} = \mathsf{drop}(3432, 2) = \lfloor 3432/4 \rfloor = 858.$$

Decoding $\hat{z}$, we obtain $z = \hat{z}/s_z = 858/32 = 26.8125$. The ideal product is $6.5 \cdot 4.2 = 27.3$. The deviation is within the expected quantization bounds.

Unified Vector Multiplication *($s_{x_i} = s_{x_j}$)* Consider the inner product of two vectors $\mathbf{x} = [1.2, 0.5]$ and $\mathbf{y} = [2.1, 1.4]$ with uniform scaling factors $s_x = 64$ and $s_y = 32$. Their dynamic fixed-point encodings are:

- $\hat{\mathbf{x}} = [\lfloor 1.2 \cdot 64 \rfloor, \lfloor 0.5 \cdot 64 \rfloor] = [76, 32]$, where $\hat{x}_i \in \mathbb{Z}_{2^8}$.
- $\hat{\mathbf{y}} = [\lfloor 2.1 \cdot 32 \rfloor, \lfloor 1.4 \cdot 32 \rfloor] = [67, 44]$, where $\hat{y}_i \in \mathbb{Z}_{2^7}$.

The current product scale is $s_x \cdot s_y = 64 \cdot 32 = 2048$. We target an output scale $s_z = 128$ and bit-width $\ell_z = 12$. To handle the scale transition $\frac{s_z}{s_x \cdot s_y} = \frac{128}{2048} = 0.0625$, we introduce a resolution factor $s_r = 2^8 = 256$ (i.e., $\ell_r = 8$). We calculate the unified transition coefficient $C_{\mathsf{mult}} = \lfloor 0.0625 \cdot 256 \rfloor = 16$. The computation is performed in an intermediate ring $\mathbb{Z}_{2^{\ell_b}}$. The sum of products is:$\sum_{i=1}^{2} (\hat{x}_i \cdot \hat{y}_i) = (76 \cdot 67 + 32 \cdot 44) = 5092 + 1408 = 6500$. The weighted result $u = C_{\mathsf{mult}} \cdot \sum (\hat{x}_i \cdot \hat{y}_i) = 16 \cdot 6500 = 104000$. To accommodate this magnitude ($\approx 2^{16.6}$), we set $\ell_b = 20$, thus $u = (C_{\mathsf{mult}} \cdot \sum \hat{x}_i \hat{y}_i) \pmod{2^{20}} = 104000$. To obtain the final encoding $\hat{z}$ at scale s_z, we apply the drop operator to remove the $\ell_r = 8$ bits: $\hat{z} = \mathsf{drop}(104000, 8) = \lfloor 104000/256 \rfloor = 406$. The result $\hat{z} = 406$ correctly resides in the target ring $\mathbb{Z}_{2^{12}}$. The final real value is $z = \hat{z}/s_z = 406/128 = 3.171875$. This closely approximates the ideal inner product $(1.2 \cdot 2.1) + (0.5 \cdot 1.4) = 2.52 + 0.7 = 3.22$ with a quantization error of 0.048.

B. Computation of Terms for Π_{poly}.

Succinct DFC-Polynomial Term (Π_{poly}) The objective of Π_{poly} is to compute the power $\langle \hat{z} \rangle^{2^{\ell_z}}$ from $\langle \hat{x} \rangle^{2^{\ell_x}}$ in the target ring. We have:

$$\hat{x} \cdot \hat{x} = C_{\mathsf{mult}} \cdot \{(\hat{x}' - 2^{\ell_x} w_x - 2^{\ell_x - 2})^2\} \pmod{2^{\ell_b}}.$$

Expanding the eq yields,

$$\begin{aligned} \hat{x}^2 = C_{\mathsf{mult}} \cdot \{ & \underbrace{(\hat{x}')^2}_{\text{Input Square}} - \underbrace{2 \cdot \hat{x}' \cdot 2^{\ell_x} w_x}_{\text{Cross Term}} + \underbrace{(2^{\ell_x} w_x)^2}_{\text{MSB Square}} \\ & - \underbrace{2 \cdot 2^{\ell_x - 2} \cdot \hat{x}'}_{\text{Linear Shift}} + \underbrace{2 \cdot 2^{\ell_x - 2} \cdot 2^{\ell_x} w_x}_{\text{MSB Correction}} \\ & + \underbrace{(2^{\ell_x - 2})^2}_{\text{Constant Offset}} \} \pmod{2^{\ell_b}}. \end{aligned}$$

As illustrated in the derivation above, the expansion of $\hat{x}^2$ involves fewer intermediate terms compared to the general product $\hat{x} \cdot \hat{y}$. By leveraging this symmetry, we can evaluate higher-order powers using the same optimized multiplication paradigm, thereby streamlining the power generation process within our protocol.

C. Benchmark Results (Π_{mult})

Performance Dynamic fixed-point multiplication. **Performance of Dynamic Fixed-Point Multiplication.** We compare our protocol with SOTA truncation-based approaches (ASTRA [29]) under varying data volumes (2^{16} – 2^{24}) and truncation bit-widths ($\ell_b \in 16, 24, 32$), in both MAN and WAN settings. The results are summarized in Table VI. Across all configurations, our protocol significantly reduces online latency, achieving 2×–4× speedup over SOTA. For instance, at $(\ell_x, \ell_y, \ell_b) = (6, 4, 16)$, the online time drops from 38.21 ms to 18.58 ms (2.05×) at 2^{16}, and from 7471.85 ms to 2228.70 ms (3.35×) at 2^{24} in MAN. The improvement is even more pronounced in WAN, with specific speedup ratios illustrated in Figure 6, indicating that our design effectively reduces communication-intensive truncation overhead. As the data volume increases, the performance gap consistently widens.

TABLE VI: Test Results from Dynamic Fixed Point Multiplication (Time in ms). The SOTA denotes ASTRA [29].

(ℓ_x, ℓ_y, ℓ_b)	Protocol	MAN setting			WAN setting		
	Test batch	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
(6, 4, 16)	SOTA^{+}	38.21	591.44	7471.85	85.47	1309.32	19015.27
	Ours^{+}	**18.58**	**143.55**	**2228.70**	**30.26**	**318.35**	**5606.19**
	SOTA*	62.11	853.99	11313.3	127.65	1970.8	28620.01
	Ours*	**51.05**	**522.16**	**7353.88**	**128.8**	**1180.33**	**18046.67**
(8, 12, 24)	SOTA^{+}	41.68	692.78	9598.15	109.28	1634.84	24461.68
	Ours^{+}	**24.71**	**198.24**	**3274.69**	**42.72**	**583.87**	**8325.47**
	SOTA*	65.58	1003.61	14518.84	160	2573.99	37185.48
	Ours*	**57.61**	**850.6**	**12728.99**	**138.94**	**2135.06**	**32119.23**
(10, 14, 32)	SOTA^{+}	65.65	720.45	10643.85	116.81	1849.37	27261.42
	Ours^{+}	**34.67**	**272.72**	**4367.71**	**53.33**	**784.54**	**11587.85**
	SOTA*	86.46	1103.28	16127.89	173.29	2819.05	40969.84
	Ours*	**75.63**	**952.42**	**14061.02**	**154.89**	**2459.84**	**36,509.8**

$+$: Online phase; $*$: Total time (including offline phase).

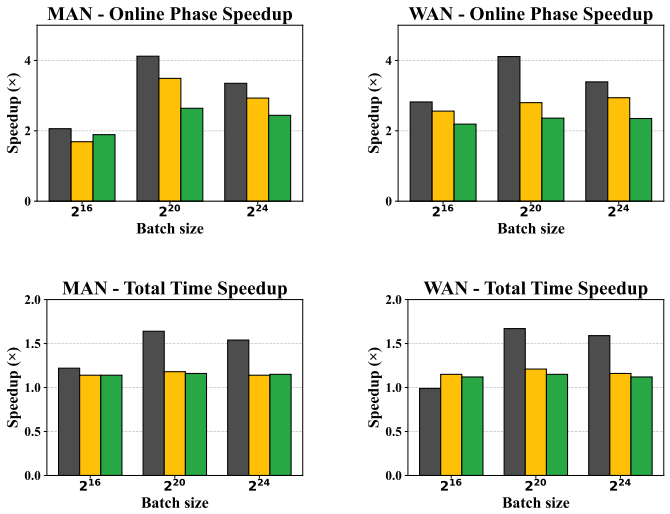

Fig. 6: Speedup of our protocol over SOTA under different network settings and batch sizes. The ■ indicates $\ell_b = 16$. The ■ indicates $\ell_b = 24$. The ■ indicates $\ell_b = 32$.

While SOTA methods exhibit near-linear growth with a large constant factor due to costly truncation, our protocol scales more efficiently, suggesting improved amortization and reduced per-element overhead. This trend holds across both network settings. For total latency (offline + online), our protocol still consistently outperforms SOTA, although the margin is smaller compared to the online phase. This is expected, as offline costs are less impacted by truncation optimization. Nevertheless, the overall gains demonstrate that our approach improves end-to-end efficiency.

D. Impact of Vector Dimension on Ring Size.

The codomain of an inner product is fundamentally distinct from its domain, as the output range depends on the vector dimension. ASTRA [29] employs a standard fixed-point encoding where the required maximum ring size depends on the input dimension; consequently, the bit-width must be increased as the test batch size grows to prevent overflow. Similarly, in the quantized PPML framework by Lu et al., [4], the summation following a LUT necessitates a larger ring to accommodate the accumulated results as the dimension expands. In contrast, DYNAFIXanalytically pre-calculates an optimal intermediate bit-width ℓ_b for the calculation and produces a fixed ℓ_z-bit output via a one-time drop operation, regardless of the accumulation depth.

Example Calculation. For a protocol like ASTRA [29] to remain overflow-free during an inner product of dimension n, the ring size ℓ_{size} must satisfy $\ell_{size} \geq \ell_{\mathsf{data}} + \lceil \log_2 n \rceil$. Assuming a base data bit-width (ℓ_{data}) of 31 bits. For Batch 2^{16}, need $31 + 16 = \mathbf{47}$ bits.