

MCU: Algebraic and Transcendental Function Evaluation in MPC without Preprocessing

Min Yang, Jinxuan Du, Zihang Zhou, Dongcan Guo, and Qingshu Meng

Wuhan University, Hubei Province 430072, China qsmeng@126.com

Abstract. The rapid proliferation of privacy-sensitive machine learning and scientific computing applications has intensified the demand for secure multi-party computation (MPC) protocols that can efficiently evaluate complex nonlinear functions. Despite four decades of research, MPC systems face two fundamental barriers: heavy offline preprocessing and limited support for transcendental functions. Existing preprocessing-free frameworks are confined to comparison-related functions (e.g., Bicaptor) and cannot handle exponentials or trigonometric functions.

In this paper, we propose MCU (Mask-Compute-Unmask), a novel MPC architecture that directly supports a broad class of algebraic and transcendental functions without any offline preprocessing. MCU introduces a semi-honest, non-colluding helper party (HP) as an active computational engine: parties additively mask their inputs, the HP computes the target function on the aggregated masked data, and the parties then unmask using locally known masks. This paradigm enables constant-round, scalable protocols for general multiplication (2 rounds), power functions (2 rounds), exponentials (8 rounds), and trigonometric functions (2 rounds). All protocols achieve simulation-based security and rely solely on synchronized pseudorandom generators—no preprocessing, no polynomial approximations.

To the best of our knowledge, MCU is the first MPC framework that simultaneously achieves preprocessing-free, constant-round, exact evaluation of exponentials and trigonometric functions—functions essential for scientific computing, differential privacy mechanisms, and advanced neural network architectures. Our multiplication protocol scales better than Beaver-based multiplication in multi-party settings, and our exponential and trigonometric protocols achieve sub-ULP accuracy.

1 Introduction

1.1 The Privacy-Computation Dilemma

The proliferation of data-driven applications—from collaborative medical research to privacy-preserving machine learning—has created an urgent need for secure computation on sensitive data. Secure multi-party computation (MPC) provides a rigorous cryptographic solution, enabling multiple parties to jointly compute functions over their private inputs without revealing them.

Since Yao’s introduction of garbled circuits in the 1980s [23], MPC has witnessed remarkable advances. Today, MPC protocols can support various computation tasks with provable security guarantees. Yet two fundamental challenges remain: **efficiency** and **scalability**—particularly when evaluating complex non-linear functions.

1.2 The Preprocessing Bottleneck

Many modern MPC systems achieve impressive online performance by moving heavy cryptographic work to an offline preprocessing phase [5,13,4,15,10]. Parties generate correlated randomness (e.g., Beaver triples) before seeing actual inputs, making the online phase lightweight.

However, preprocessing introduces severe problems. First, it requires storage and communication of massive random data—for large-scale models with millions of parameters, the overhead becomes prohibitive. Second, the preprocessing phase itself is an MPC problem, often requiring either a trusted dealer or additional interaction rounds. Many systems introduce a semi-honest, non-colluding helper party (HP) solely for offline randomness generation [18,20,10,9,21,17], but this merely shifts the problem rather than eliminating it.

1.3 The Transcendental Function Gap

A smaller line of recent works have attempted to eliminate preprocessing entirely by using synchronized pseudorandom generators (PRGs) to generate masks on the fly [27,30]. However, these approaches are confined to comparison-related functions (e.g., Bicoprot [30] supports only sign and ReLU). Exponential, trigonometric, and other transcendental functions—which are critical for scientific computing, differential privacy, positional encodings, and advanced activation functions—remain unsupported in preprocessing-free frameworks. Most existing schemes [12,16,6,22,29,24,8,19,7,3,2] require preprocessing, and those that avoid it rely on polynomial approximations or iterative methods that introduce accuracy loss and increase communication rounds.

1.4 Our Contributions

We propose **MCU (Mask-Compute-Unmask)**, a new MPC architecture that directly supports algebraic and transcendental functions **without any offline preprocessing**. The core idea is to redefine the helper party as an active computational engine: parties additively mask their inputs using synchronized randomness, the HP computes the target function on the aggregated masked data, and the parties then unmask using locally known masks. (A detailed description of this workflow is given in Section 2.)

Based on the MCU framework, we present a comprehensive protocol suite with the following characteristics:

- **No offline preprocessing:** All randomness is generated online via synchronized PRGs. Parties exchange a one-time seed before any computation; no per-circuit correlated randomness is stored or communicated.
- **Exact computation:** For algebraic functions, results are exact over the ring $\mathbb{Z}_{2^t}$. For transcendental functions, we use standard floating-point arithmetic with no additional approximation beyond the inherent rounding of the math library.
- **Constant communication rounds** (independent of the number of parties n):
 - General multiplication and power x^k : **2 rounds**
 - Exponential a^x : **8 rounds**
 - Trigonometric functions $\sin(x), \cos(x)$: **2 rounds**

These round counts hold for *any* number of parties n . In contrast, previous preprocessing-free frameworks either support only comparison-related functions (e.g., Bicoptor [30]) or introduce additional rounds when scaling to multi-party settings.

- **Broad function coverage:** MCU handles multiplication, power, exponentials, and trigonometric functions—unlike prior preprocessing-free works confined to comparisons.
- **Scalability:** The HP bears most of the computational burden, and communication grows only linearly with the number of parties. Our multiplication protocol scales better than Beaver-based multiplication in multi-party settings (see Section 5).
- **Provable security:** All protocols are proven secure in the semi-honest model under the assumption that the HP does not collude with any party. The proofs follow the simulation paradigm.

To the best of our knowledge, MCU is the first MPC framework that simultaneously achieves **preprocessing-free**, **constant-round**, and **exact** evaluation of exponentials and trigonometric functions. Our multiplication and power protocols also accelerate polynomial evaluation, benefiting many approximation-based or M2A-based schemes.

1.5 Why This Matters Now

The rise of privacy-sensitive computing—from confidential clinical trials to proprietary financial modeling—has made efficient privacy-preserving computation an urgent industrial need. Current MPC solutions either suffer from prohibitive preprocessing or rely on polynomial approximations that sacrifice accuracy and round efficiency. MCU breaks this deadlock by enabling exact, constant-round evaluation of key transcendental functions without any preprocessing. Its multiplication protocol can also accelerate existing schemes that rely on multiplicative masks or M2A protocols [24,2,14].

1.6 Paper Organization

The remainder of this paper is organized as follows. Section 2 defines the MCU architecture, security model, and computational model. Section 3 presents the full suite of protocols. Section 4 provides security proofs. Section 5 reports experimental results. Section 6 discusses related work and addresses potential concerns. Section 7 concludes.

2 The MCU Architecture

2.1 Security Model and its Rationale

We consider the semi-honest (honest-but-curious) adversary model, in which parties follow the protocol specifications but may attempt to infer additional information from their views. The HP is assumed to be non-colluding. We argue that this assumption is rational from two aspects.

First, introducing a non-colluding HP to achieve efficiency gains and security guarantees is an accepted and cheap trade-off in the community. In the standard end-to-end MPC, even the simplest multiplication operation requires offline pre-generation of Beaver triples to eliminate the randomness introduced in additive masks. For complex nonlinear functions, solutions also necessarily require offline precomputation, implying great deployment cost. Many efficient solutions introduce a semi-honest, non-colluding third-party server [18,20,10,9,21,17] or cryptographic commodity server [1,28].

Second, this assumption is grounded in realistic deployment scenarios. Examples include:

- **Financial dark pools:** A securities regulator operates the trading platform, matching buy and sell orders without colluding with participating institutions.
- **Healthcare data sharing:** A health authority aggregates medical records from hospitals for research, inherently motivated to protect patient privacy.
- **Industry consortia:** A trade association collects and analyzes member companies' operational data to provide benchmarking services, with no incentive to favor any particular member.

In such scenarios, the HP—acting as a regulator, authority, or association—naturally fulfills the role of a fair, non-colluding, semi-honest party.

2.2 MCU Components

The MCU framework relies on two types of pseudorandom generators (PRGs) to achieve communication-free randomness generation:

- **PRG:** A cryptographically secure pseudorandom generator shared among all participating servers $P_1, \dots, P_n$ with identical seeds. This enables parties to generate the same sequence of random numbers without any communication.

- **ASPRG**: A unique pseudorandom generator shared between the HP and each individual party P_i , each instantiated with distinct seeds. This allows the HP to privately distribute shares to each party.

Initialization. Before protocol execution, a one-time seed establishment is performed for both the PRG and the ASPRGs. This setup is lightweight and does not involve per-circuit correlated randomness (e.g., Beaver triples), distinguishing MCU from preprocessing-heavy MPC systems.

2.3 Operational Flow

The MCU framework computes a function $f(x_1, \dots, x_n)$, where x_i is the private input of party P_i , and outputs additive secret shares of the result to all parties. The protocol proceeds in three steps:

1. **Mask**: Each party P_i uses PRG_0 to generate a random mask r_i . It computes $m_i = x_i + r_i \bmod 2^l$ (additive masking) and sends m_i to the HP.
2. **Compute**: The HP aggregates the received values to obtain $M = \sum_{i=1}^n m_i = x + r \bmod 2^l$, where $x = \sum x_i$ and $r = \sum r_i$. It then computes the desired value $v = g(M)$, where g is a function derived from f (e.g., if $f = a^x$, then $g(M) = a^M$). Next, the HP generates random shares $s_1, \dots, s_{n-1}$ using the ASPRG, sets $s_n = v - \sum_{i=1}^{n-1} s_i$, and sends s_n to P_n .
3. **Unmask**: Each party P_i obtains its share s_i (for $i < n$, s_i is generated locally via ASPRG_i ; for $i = n$, it is received from the HP). Using the local knowledge of r (reconstructed from the r_i via PRG_0), each party computes its final share $\langle f(x) \rangle_i = h(s_i, r)$, where h depends on the specific function being evaluated.

The key observation is that the HP only sees $M = x + r \bmod 2^l$ with r uniformly random, and thus learns nothing about x . This property forms the foundation of provable security for all additive-mask protocols in the MCU framework.

2.4 Computational Model

Our protocols operate in two distinct computational settings depending on the function type.

Algebraic Functions. For functions such as multiplication and power, computation is performed over a ring $\mathbb{Z}_{2^l}$. All operations—addition, multiplication, and reconstruction—are exact and deterministic. The protocols produce shares that reconstruct to the exact value (up to rounding) that would be obtained by evaluating the function in the same algebraic structure.

Transcendental Functions. For functions such as $\exp(x)$, $\sin(x)$, and $\cos(x)$, each input is masked as $x_i + r_i \bmod 2^l$. The HP reconstructs a masked value $x + r \bmod 2^l$, converts it to a floating-point number, and evaluates the function using standard arithmetic (e.g., IEEE 754). We assume all parties have access to a deterministic floating-point implementation of these functions (e.g., from a

standard math library). The final reconstructed output is identical to evaluating the function on the plaintext input using the same floating-point arithmetic. No additional approximation is introduced by the MPC protocol.

3 Provably Secure Protocols

We now present the protocols. For each, we highlight the unmasking challenge and describe how we address it.

3.1 General Multiplication

Suppose $x_1, x_2, \dots, x_k$ are k variables. By inclusion-exclusion,

$$\prod_{j=1}^k x_j = \prod_{j=1}^k (x_j + r_j - r_j) = \sum_{S \subseteq \{1, \dots, k\}} (-1)^{|S|} \left(\prod_{j \in S} r_j \right) \left(\prod_{j \notin S} (x_j + r_j) \right). \quad (1)$$

Taking both sides of (1) modulo M also holds naturally. When $\prod_{j=1}^k x_j < M$, the correctness of the result is unaffected by whether any $x_j + r_j$ exceeds M .

Protocol 1. Multiplication Protocol

Input: Each P_i holds $\langle x_1 \rangle_i, \dots, \langle x_k \rangle_i$ with $x_j = \sum_i \langle x_j \rangle_i$.

Output: Each P_i obtains $\langle \prod_{j=1}^k x_j \rangle_i$.

1. Each party P_i (in parallel) compute $a_{j,i} = \langle x_j \rangle_i + \langle r_j \rangle_i \bmod 2^l$ for $j = 1, \dots, k$. Send k values $a_{1,i}, \dots, a_{k,i}$ to HP.
2. HP reconstructs $\sum_{m=1}^n a_{j,m} = x_j + r_j \bmod 2^l$, $j = 1, \dots, k$. For each $S \in \{1, 2, \dots, k\}$, HP calculates $A_S = \left(\prod_{j \notin S} (x_j + r_j) \right) \bmod 2^l$, generates shares $t_{S,1}, \dots, t_{S,n-1}$ via $ASPRG_1, \dots, ASPRG_{n-1}$ respectively, sets $t_{S,n} = A_S - \sum_{j=1}^{n-1} t_{S,j} \bmod 2^l$, and sends $t_{S,n}$ to P_n . There are 2^k different S , as the full set S is unnecessary to share in ASS way.
3. Each party P_i (in parallel) compute $\langle \prod_{j=1}^k x_j \rangle_i = \sum_{S \subseteq \{1, \dots, k\}} (-1)^{|S|} \left(\prod_{j \in S} r_j \right) t_{S,i} \bmod 2^l$ as its output share. P_1 additionally adds $(-1)^k r_1 r_2 \dots r_k \bmod 2^l$ to its share.

Properties: The communication rounds of the protocol is only 2 at the cost that the protocol needs to transfer $2^k - 1$ ring elements to P_n . It is practical for k not big, say $k \leq 5$.

The protocol has three special cases.

- First Case: When $k = 2$, the problem becomes the classic 2-party multiplication problem.
- Second Case: If each variable x_j is specially shared such that only one share is non-zero, the protocol becomes M2A (transforming multiplicative shares into additive shares).
- Third Case: If all k variables are equal, the protocol becomes a power protocol, which we simplify separately.

3.2 Power Function x^k

Protocol 2. Power Protocol

Input: Shares of x .

Output: Shares of x^k .

1. **Mask:** Each P_i uses PRG_0 to generate a random r_i and sends $m_i = x_i + r_i \bmod 2^l$ to the HP.
2. **Compute:** The HP computes $R = \sum m_i = x + r \bmod 2^l$, where $r = \sum r_i \bmod 2^l$. For each exponent $i = 2, \dots, k$, the HP computes $p_i = R^i \bmod 2^l$. For each i , the HP generates shares $s_{i,1}, \dots, s_{i,n-1}$ via ASPRG, sets $s_{i,n} = p_i - \sum_{j=1}^{n-1} s_{i,j} \bmod 2^l$, and sends $s_{i,n}$ to P_n .
3. **Unmask:** Parties compute $r = \sum r_i \bmod 2^l$ via PRG_0 . By the formula

$$x^k = (x + r - r)^k = \sum_{j=0}^k \binom{k}{j} (x + r)^j (-r)^{k-j} \bmod 2^l,$$

each party P_i computes $\langle x^k \rangle_i = \sum_{j=1}^k \binom{k}{j} (-r)^{k-j} s_{j,i} \bmod 2^l$.

Properties: The Beaver-based approach requires $O(\log k)$ rounds via repeated squaring while Protocol 2 requires 2 rounds, independent of the degree k . Using the power protocol, many nonlinear functions can be evaluated in low constant rounds.

3.3 Sign Function

Our sign protocol is built upon the deterministic truncation framework of Bicoprotor 2.0 [30]. In Bicoprotor 2.0, the input $x \in \mathbb{Z}_2^l$ is split into two additive shares $[x]_0 = x + R$ and $[x]_1 = -R$ over $\mathbb{Z}_{2^l}$, where R is a random mask. In our multi-party setting, the HP receives the masked sum $M = \sum_i (x_i + s_i) = x + s$ (where $s = \sum_i s_i$ is the aggregate mask known only to the parties), while the parties locally know s . By identifying $[x]_0 = M = x + s$ and $[x]_1 = -s$, the HP and the parties collectively emulate the two shares required by Bicoprotor 2.0. Consequently, the HP can execute the deterministic truncation and zero-detection steps of Bicoprotor 2.0 to compute $\text{sign}(x)$ without any probabilistic error. The HP returns no additional information to the parties, and the security follows from the additive masking in additive group, the multiplicative masking in multiplicative group and the non-colluding assumption.

Enhanced privacy via optional input blinding: To further prevent the HP from inferring any statistical information about the sign (even in scenarios where the aggregate mask s might be partially compromised), we introduce an *optional* random sign flip. This technique is adopted from Bicoprotor 2.0. The flip is controlled by a parameter $\text{flip} \in \{0, 1\}$ (default 0). When $\text{flip} = 1$, the parties jointly generate a random bit t and multiply their input shares by $(-1)^t$; the final

output is then corrected by XORing with t . Since the HP never learns t , its view becomes statistically independent of the true sign. This optional feature does not affect the correctness of the protocol when $\text{flip} = 0$ (which is used internally by the Wrap protocol).

Protocol 3. Sign Protocol (with optional input blinding)

Input: Each P_i holds $x_i \in \mathbb{Z}_{2^\ell}$, where $x = \sum_{i=1}^n x_i \bmod 2^\ell$ and $|x| < 2^{\ell_x}$ ($2\ell_x \leq \ell$). **Parameter:** $\text{flip} \in \{0, 1\}$ (default 0). **Output:** $\text{sign}(x)$.

Protocol:

1. **(Optional) Input blinding (no communication).** If $\text{flip} = 1$:
 - All parties generate a common random bit $t \in_R \{0, 1\}$ using PRG_0 .
 - Each party P_i locally computes its blinded input share:

$$x'_i = (1 - 2t) \cdot x_i \pmod{2^\ell}.$$

(This corresponds to $x' = (-1)^t x$.)

Otherwise, set $x'_i = x_i$ for all i (and t is effectively 0).

2. **Mask and send to HP (1 round).** Each P_i generates $s_i \in_R \mathbb{Z}_{2^\ell}$ from PRG_0 , computes $m_i = x'_i + s_i \bmod 2^\ell$, and sends m_i to the HP.
3. **HP computes decomposed bits (1 round).** The HP reconstructs $R = \sum_{i=1}^n m_i \bmod 2^\ell = x' + s \bmod 2^\ell$, where $s = \sum s_i \bmod 2^\ell$. For $j = 0, \dots, \ell_x$, HP computes

$$u_j = \text{cut}([x']_0, j, \ell - \ell_x - j) \bmod 2^{\ell_x},$$

where $[x']_0 = R$ and the second share is implicitly $[x']_1 = -s$ as in Bi-coptor 2.0. The HP generates additive shares $\langle u_j \rangle_i$ for $i = 1, \dots, n$ (using ASPRG_i for $i < n$, and sends $\langle u_j \rangle_n$ to P_n). (All shares are in $\mathbb{Z}_{2^{\ell_x}}$.)

4. **Local unmasking of the mask s .** All parties compute $s = \sum s_i \bmod 2^\ell$ (from PRG_0). For each $j = 0, 1, \dots, \ell_x$, compute $w_j = \left(2^\ell - \text{cut}(2^\ell - [x]_1, j, \ell - \ell_x - j)\right) \bmod 2^{\ell_x}$ using the same deterministic truncation rule and locally calculates one share $\langle w_j \rangle_i$ in $\mathbb{Z}_{2^{\ell_x}}$. Set $\langle u'_j \rangle_i = \langle u_j \rangle_i + \langle w_j \rangle_i \pmod{2^{\ell_x}}$. For each $j = 0, \dots, \ell_x - 1$:

$$\begin{aligned} \langle v_j \rangle_1 &= \langle u'_j \rangle_1 + \langle u'_{j+1} \rangle_1 - 1 \pmod{M}, \\ \langle v_j \rangle_i &= \langle u'_j \rangle_i + \langle u'_{j+1} \rangle_i \pmod{M}, \quad i = 2, \dots, n. \end{aligned}$$

For the boundary term $j = \ell_x$:

$$\begin{aligned} \langle v_{\ell_x} \rangle_1 &= \langle u'_{\ell_x} \rangle_1 - 1 \pmod{M}, \\ \langle v_{\ell_x} \rangle_i &= \langle u'_{\ell_x} \rangle_i \pmod{M}, \quad i = 2, \dots, n. \end{aligned}$$

5. **Shuffle.** All parties generate the same random permutation Π from PRG_0 and permute the sequence $(\langle v_0 \rangle_i, \dots, \langle v_{\ell_x} \rangle_i)$. The permuted values are still denoted $\langle v_j \rangle_i$.

6. **Modulus conversion and zero detection.** For each $j = 0, \dots, \ell_x$, let $V_j = \sum_{i=1}^n \langle v_j \rangle_i$ (integer sum, no modulus). Note that $V_j = v_j + t_j \cdot 2^{\ell_x}$ where $v_j \in [0, 2^{\ell_x}]$ and $t_j \in [0, n-1]$ is the carry.
- P_1 constructs n numbers: $q_{1,j}^{(k)} = \langle v_j \rangle_1 - k \cdot 2^{\ell_x}$ for $k = 0, \dots, n-1$.
 - For $i \geq 2$, each P_i uses $q_{i,j}^{(k)} = \langle v_j \rangle_i$ (same for all k).
 - All parties generate common non-zero random numbers $r_0, \dots, r_{n-1} \in \mathbb{Z}_p^*$ from PRG_0 , where p is the smallest prime greater than $n \cdot 2^{\ell_x}$.

Round 3: Each P_i sends to the HP:

$$\text{For } P_1 : r_k \cdot q_{1,j}^{(k)} \bmod p \quad (\forall j, k); \quad \text{For } P_i (i \geq 2) : r_k \cdot \langle v_j \rangle_i \bmod p \quad (\forall j, k).$$

The HP computes for each j, k :

$$T_{j,k} = \sum_{i=1}^n (r_k \cdot q_{i,j}^{(k)}) \bmod p = r_k \cdot (V_j - k \cdot 2^{\ell_x}) \bmod p.$$

Because $p > n \cdot 2^{\ell_x} \geq |V_j - k \cdot 2^{\ell_x}|$, the reduction modulo p does not wrap around; thus $T_{j,k} = 0$ iff $V_j = k \cdot 2^{\ell_x}$. The HP sets $\text{sign}(x') = 1$ if there exists any j, k with $T_{j,k} = 0$; otherwise $\text{sign}(x') = 0$. The HP shares the $\text{sign}(x')$ in ASS way among the n parties.

7. **Unblind (if flip was used).** Each party P_i computes $(1-2t) < \text{sign}(x') >_i$.

Properties: The protocol requires 3 communication rounds (Steps 2, 4, and 6), independent of the optional blinding. The optional blinding adds no communication overhead and only local multiplications by -1 and an XOR. When $\text{flip} = 0$ (default), the protocol behaves exactly as the original Sign protocol and is used internally by the Wrap protocol. When $\text{flip} = 1$, the HP's view is further strengthened: even if the HP were to guess the aggregate mask s , the random bit t ensures that the sign of the blinded input x' is equally likely to be flipped, rendering the HP's output statistically independent of the true sign. This provides an additional layer of defense against potential side-channel attacks on the random masks.

3.4 Wrap Protocol

The objective is to securely compute the overflow bit

$$w = \mathbf{1}_{x+r \geq M},$$

where $M = 2^\ell$, $x = \sum_{i=1}^n x_i \bmod M$ is the secret input, and $r = \sum_{i=1}^n r_i \bmod M$ is the aggregate mask known to all parties but not to the HP. This overflow bit is essential for reconstructing the true integer $x + r$ from its residue modulo M .

The core idea is to let the parties locally construct shares of $z = r - t$, where $t = (x + r) \bmod M$, and then securely determine the sign of z using the Sign protocol (Protocol 3). Since $z > 0$ exactly when $x + r \geq M$, the sign of z directly gives the overflow bit.

Protocol 4. Wrap Protocol

Input: Each party P_i holds $x_i \in \mathbb{Z}_M$ with $x = \sum_{i=1}^n x_i \bmod M$. Each party knows the aggregate mask $r = \sum_{i=1}^n r_i \bmod M$ via PRG_0 .

Output: The HP obtains the overflow bit $w = \mathbf{1}_{x+r \geq M}$.

Protocol:

1. **Round 1 (Parties $\rightarrow$ HP).** Each party P_i generates its mask share $r_i \in_R \mathbb{Z}_M$ from PRG_0 , computes

$$m_i = x_i + r_i \bmod M,$$

and sends m_i to the HP.

2. **Round 2 (HP computes t and shares it in N).** The HP reconstructs

$$t = \sum_{i=1}^n m_i \bmod M = (x + r) \bmod M.$$

The HP then additively shares t in the enlarged ring $N = M^2 = 2^{2\ell}$:

- Samples $t_1, \dots, t_{n-1} \in_R \mathbb{Z}_N$ using $\text{ASPRG}_1, \dots, \text{ASPRG}_{n-1}$;
- Computes $t_n = t - \sum_{i=1}^{n-1} t_i \pmod{N}$;
- Sends t_n to P_n . Each party P_i obtains its share t_i (via ASPRG_i for $i < n$, and directly for $i = n$).

Thus $\sum_{i=1}^n t_i \equiv t \pmod{N}$.

3. **Local construction of z shares (no communication).** Each party locally computes its share of $z = r - t$:

$$z_1 = r - t_1 \pmod{N}, \quad z_i = -t_i \pmod{N} \quad (i = 2, \dots, n).$$

Then $\sum_{i=1}^n z_i \equiv r - t \pmod{N}$. Since $r, t \in [0, M - 1]$, we have $r - t \in [-(M - 1), M - 1]$, and $N = M^2 \gg M$, so no modular wrap occurs and the sign of z is preserved.

4. **Round 3 (Invoke Sign Protocol).** The parties invoke the Sign protocol (Protocol 3, without the optional blind step) on input shares $z_1, \dots, z_n$ over the ring N , but only the HP knows the result $\text{sign}(z) = \text{sign}(r - t)$.

Properties: The protocol requires 1 round for masking, 1 round for sharing t , and the communication rounds of the Sign protocol (3 rounds), totaling **5 rounds**. The communication cost is dominated by the Sign protocol invocation. The protocol is secure in the semi-honest, non-colluding HP model: the HP learns only the overflow bit w , and cannot infer x or r from its view.

Correctness:

$$z = r - t = \begin{cases} r - (x + r) = -x \leq 0, & \text{if } x + r < M \text{ (no overflow),} \\ r - (x + r - M) = M - x > 0, & \text{if } x + r \geq M \text{ (overflow).} \end{cases}$$

Hence $w = 1$ exactly when $x + r \geq M$.

3.5 Exponential Function e^x

Evaluating the exponential function e^x in the MCU framework faces the same two fundamental challenges:

Challenge 1: Exact integer recovery from modular residue with privacy. The HP receives only $Y = (x_u + r) \bmod 2^L$. To compute $e^{(x_u+r)/2^{s_1}}$, it must reconstruct the true integer $X = x_u + r$, requiring the overflow bit $w = \mathbf{1}_{x_u+r \geq 2^L}$. We design the wrap protocol.

Challenge 2: Semantic ambiguity of signed inputs in two's complement. We adopt the translation method: add a public offset $\Delta = 2^{\ell-1}$ to map signed $x_s \in (-2^{\ell-1}, 2^{\ell-1})$ to non-negative $x_u = x_s + \Delta \in [0, 2^\ell)$.

Parameter Setup and Fixed-Point Scaling Let the real-valued input be $x_s \in (-B, B)$ with $B = 8$. We choose input scaling s_1 , output fractional precision s_{out} , and guard bits γ' . Define

$$\ell = \lceil \log_2 B \rceil + s_1 + 1, \quad \Delta = 2^{\ell-1}.$$

The real input is scaled and translated to an unsigned ring element:

$$x_{sf} = \lfloor x_s \cdot 2^{s_1} + 0.5 \rfloor \in (-2^{\ell-1}, 2^{\ell-1}), \quad x_u = x_{sf} + \Delta \pmod{2^\ell}.$$

Masking and overflow detection are performed in $\mathbb{Z}_{2^\ell}$. All secret shares and the final output reside in $\mathbb{Z}_{2^\kappa}$, where κ is a security parameter (in our implementation, $\kappa = 128$). To guarantee sub-ULP accuracy, we define two fixed-point scaling factors:

$$S_V = 2^{s_{\text{out}}}, \quad S_C = 2^{s_{\text{out}} + \gamma'},$$

where S_V is used for the base branch V_0 computed by the HP, and S_C is used for the public constants C_w and C_{off} computed locally by the parties. The extra γ' guard bits ensure that C_{off} never rounds to zero.

Modified Wrap Protocol for Shared Overflow We use a modified version of the Wrap protocol (based on Protocol ??) that outputs the overflow bit

$$w = \mathbf{1}_{x_u+r \geq 2^\ell}$$

as additive shares $\langle w \rangle_i \in \mathbb{Z}_{2^\kappa}$ to the parties, without revealing w to the HP. The modified Wrap protocol extends the standard Wrap protocol with one additional round to distribute the shares $\langle w \rangle_i$ to the parties. Thus it requires 6 rounds in total. Its first round is executed concurrently with the masking step of the exponential protocol, and its final round (distributing $\langle w \rangle_n$) is executed concurrently with the base branch sharing step (Round 6).

Protocol Description Protocol 5. Exponential Protocol (Final Engineering Version)

Input: Each party P_i holds a share $x_{u,i} \in \mathbb{Z}_{2^\ell}$ such that $\sum_i x_{u,i} \equiv x_u \pmod{2^\ell}$.

Output: Each party obtains $\langle e^{x_s} \rangle_i \in \mathbb{Z}_{2^\kappa}$ satisfying

$$\sum_{i=1}^n \langle e^{x_s} \rangle_i \equiv \lfloor e^{x_s} \cdot 2^{s_{\text{out}}} + 0.5 \rfloor \pmod{2^\kappa},$$

with error < 0.5 ULP.

Protocol (8 communication rounds):

1. **Round 1 (Masking and Wrap initiation).** Each P_i generates a random mask $r_i \in_R \mathbb{Z}_{2^\ell}$ via the shared PRG and sends

$$m_i = x_{u,i} + r_i \pmod{2^\ell}$$

to the HP. Simultaneously, the parties initiate the modified Wrap protocol to obtain additive shares $\langle w \rangle_i \in \mathbb{Z}_{2^\kappa}$ of the overflow bit $w = \mathbf{1}_{x_u + r \geq 2^\ell}$.

2. **Rounds 2–6 (Wrap completion).** The modified Wrap protocol completes. At its end, each party holds its share $\langle w \rangle_i$; the HP learns nothing about w .
3. **Round 6 (HP computes and shares the base branch).** The HP computes

$$Y = \left(\sum_{i=1}^n m_i \right) \bmod 2^\ell = (x_u + r) \bmod 2^\ell,$$

and evaluates the base-branch value with scaling $S_V = 2^{s_{\text{out}}}$:

$$V_0 = \left\lfloor \exp\left(\frac{Y}{2^{s_1}}\right) \cdot S_V + 0.5 \right\rfloor.$$

The HP additively shares V_0 over $\mathbb{Z}_{2^\kappa}$: it samples $v_1, \dots, v_{n-1} \in_R \mathbb{Z}_{2^\kappa}$, sets

$$v_n = V_0 - \sum_{i=1}^{n-1} v_i \pmod{2^\kappa},$$

and sends v_n to party P_n . Each party obtains $\langle V_0 \rangle_i = v_i$.

4. **Rounds 7–8 (Secure multiplication $w \cdot V_0$).** The parties invoke the generic multiplication protocol (Protocol 1, $k = 2$) on the secret-shared inputs $\langle w \rangle_i$ and $\langle V_0 \rangle_i$ over $\mathbb{Z}_{2^\kappa}$. This yields shares $\langle Q \rangle_i$ such that

$$\sum_{i=1}^n \langle Q \rangle_i \equiv w \cdot V_0 \pmod{2^\kappa}.$$

Since $w \cdot V_0 \leq 2^{s_{\text{out}}} \ll 2^\kappa$, the modular sum equals the integer product.

5. **Local combination (no communication, pure integer arithmetic).**

Each party reconstructs the aggregate mask $r = \sum_i r_i \bmod 2^\ell$ from the shared PRG, and computes the two public fixed-point constants using the scaling $S_C = 2^{s_{\text{out}} + \gamma'}$:

$$C_w = \left\lfloor \exp\left(\frac{2^\ell}{2^{s_1}}\right) \cdot S_C + 0.5 \right\rfloor, \quad C_{\text{off}} = \left\lfloor \exp\left(-\frac{\Delta + r}{2^{s_1}}\right) \cdot S_C + 0.5 \right\rfloor.$$

Each party then computes the following integer *without modular reduction*:

$$\text{Num} = \langle V_0 \rangle_i \cdot C_{\text{off}} \cdot S_C + (C_w - S_C) \cdot \langle Q \rangle_i \cdot C_{\text{off}}.$$

The final share is:

$$\boxed{\langle e^{x_s} \rangle_i = (\text{Num} \gg (2s_{\text{out}} + 3\gamma')) \bmod 2^\kappa},$$

where $\gg$ denotes arithmetic right shift.

Implementation note. The numerator Num is computed in the integer domain without reduction. A 2κ -bit unsigned integer type (e.g., 256-bit for $\kappa = 128$) suffices, as $\text{Num} < 2^{2\kappa}$ (indeed, $\text{Num} < 2^{205}$ for our concrete parameters). The final right shift reduces the value to below 2^κ , and the modulo operation produces a valid additive share in $\mathbb{Z}_{2^\kappa}$.

Correctness Proof Summing the final shares over all parties and ignoring rounding errors (using $\approx$):

$$\begin{aligned} \sum_i \langle e^{x_s} \rangle_i &= \frac{V_0 \cdot C_{\text{off}} \cdot S_C + (C_w - S_C) \cdot Q \cdot C_{\text{off}}}{S_C^2} \\ &= \frac{V_0 \cdot C_{\text{off}} \cdot S_C + (C_w - S_C) \cdot (wV_0) \cdot C_{\text{off}}}{S_C^2} \\ &= \frac{V_0 \cdot C_{\text{off}} \cdot (S_C + w(C_w - S_C))}{S_C^2}. \end{aligned}$$

Substituting $V_0 \approx e^{Y/2^{s_1}} S_V$, $C_w \approx e^{2^\ell/2^{s_1}} S_C$, and $C_{\text{off}} \approx e^{-(\Delta+r)/2^{s_1}} S_C$:

$$\begin{aligned} \sum_i \langle e^{x_s} \rangle_i &\approx \frac{(e^{Y/2^{s_1}} S_V)(e^{-(\Delta+r)/2^{s_1}} S_C)(S_C + w(e^{2^\ell/2^{s_1}} S_C - S_C))}{S_C^2} \\ &= e^{(Y+w \cdot 2^\ell - \Delta - r)/2^{s_1}} \cdot S_V. \end{aligned}$$

Since $Y + w \cdot 2^\ell = x_u + r = x_{sf} + \Delta + r$, we have:

$$Y + w \cdot 2^\ell - \Delta - r = x_{sf}.$$

Therefore:

$$\sum_i \langle e^{x_s} \rangle_i = e^{x_{sf}/2^{s_1}} \cdot S_V = e^{x_s} \cdot 2^{s_{\text{out}}},$$

which matches the desired output format.

Strict Error Analysis (Sub-ULP Guarantee) Let the true continuous values be

$$A = e^{Y/2^{s_1}}, \quad B = e^{2^\ell/2^{s_1}}, \quad D = e^{-(\Delta+r)/2^{s_1}},$$

and the true output be $Y_{\text{true}} = A \cdot B^w \cdot D$. The fixed-point approximations are:

$$V_0 = AS_V + e_V, \quad C_w = BS_C + e_B, \quad C_{\text{off}} = DS_C + e_D,$$

with $|e_V|, |e_B|, |e_D| \leq 0.5$.

The reconstructed value $\hat{Y}$ (scaled by S_V) is given by:

$$\hat{Y} = \frac{(AS_V + e_V)(DS_C + e_D)(S_C + w((BS_C + e_B) - S_C))}{S_C^2}.$$

The relative error can be bounded (neglecting second-order terms, which are $< 2^{-2s_{\text{out}}}$) by:

$$\frac{|\hat{Y} - Y_{\text{true}}|}{Y_{\text{true}}} \leq \frac{|e_V|}{AS_V} + \frac{|e_D|}{DS_C} + \frac{|e_B|}{BS_C} \quad (\text{when } w = 1; \text{ the third term vanishes when } w = 0).$$

Each term is at most $0.5/\min(S_V, S_C)$. With $S_C = 2^{s_{\text{out}}+\gamma'}$ and $S_V = 2^{s_{\text{out}}}$, the dominant term is from e_V , giving relative error $\leq 1.5 \cdot 2^{-s_{\text{out}}}$.

The absolute error in ULP is then:

$$\text{ULP error} \leq \text{relative error} \times Y_{\text{true}} \leq 1.5 \cdot 2^{-s_{\text{out}}} \cdot e^B,$$

where B is the input bound. For $B = 8$ and $s_{\text{out}} = 14$, this gives $\approx 1.5 \cdot 2^{-14} \cdot e^8 \approx 0.273$. Including the rounding error from the final right shift (which is < 0.01 ULP), the total error is $< 0.283 < 0.5$. Therefore, sub-ULP accuracy is rigorously guaranteed.

Implementation Notes All parties compute the same constants:

$$S_V = 2^{s_{\text{out}}}, \quad S_C = 2^{s_{\text{out}}+\gamma'}, \quad C_w = \left\lfloor \exp\left(\frac{2^\ell}{2^{s_1}}\right) \cdot S_C + 0.5 \right\rfloor.$$

The local combination requires 2κ -bit unsigned integer arithmetic for the numerator Num, which is well within the capabilities of modern processors (e.g., using `__int128` for multiplication and accumulating in a 256-bit structure for $\kappa = 128$). The final $2s_{\text{out}}$ -bit right shift yields a value that may still exceed 2^κ , so we reduce it modulo 2^κ to obtain the final share.

Communication and Security The protocol requires 8 communication rounds:

- Round 1: masking (concurrent with modified Wrap’s first round).
- Rounds 2–6: modified Wrap completion.
- Round 6: HP shares V_0 (concurrent with modified Wrap’s final round).
- Rounds 7–8: generic multiplication of w and V_0 .

In terms of security, the HP's view consists solely of $Y = (x_u + r) \bmod 2^\ell$. Since x_u and r belong to the same finite group $\mathbb{Z}_{2^\ell}$, and r is uniformly random and independent of x_u , the conditional distribution of x_u given Y is perfectly uniform over all 2^ℓ possible values. Thus the protocol achieves *information-theoretic* privacy for the input against the HP. The overflow bit w is never revealed to the HP, and all shares are uniformly random in $\mathbb{Z}_{2^\kappa}$. Security against corrupted parties follows from the security of the multiplication and Wrap subprotocols (Theorems 1 and 3).

3.6 Trigonometric Functions $\sin(x)$ and $\cos(x)$

Designing a preprocessing-free trigonometric protocol faces two fundamental challenges that stem from the ring representation of signed values.

Challenge 1: Exact integer recovery from modular residue with privacy. The HP receives only $Y = (x_u + r) \bmod 2^L$. To compute $\sin((x_u + r)/2^{s_1})$, it must reconstruct the true integer $X = x_u + r$, requiring the overflow bit $w = \mathbf{1}_{x_u + r \geq 2^L}$. Different from the exponential protocol, we will use the technique[26] to solve this challenge.

Challenge 2: Semantic ambiguity of signed inputs in two's complement. We adopt the translation method: add a public offset $\Delta = 2^{\ell-1}$ to map signed $x_s \in (-2^{\ell-1}, 2^{\ell-1})$ to non-negative $x_u = x_s + \Delta \in [0, 2^\ell)$.

Parameter Setup Let the real-valued input be $x_{real} \in (-B, B)$ with $B = 2\pi$. Define:

- $s_1 = 13$: input scaling factor (radians to fixed-point)
- $s_{out} = 14$: output precision (fractional bits)
- $\gamma = 2$: guard bits for error control
- $\ell = \lceil \log_2 B \rceil + s_1 + 1 = 17$: bit-length for signed representation
- $\Delta = 2^{\ell-1}$: offset to map signed values to unsigned domain

The Period-Matching Modulus. The key insight enabling our 2-round protocol is to eliminate the Wrap protocol by choosing a modulus M with a special property.

Recall that the HP receives $Y = (x_u + r) \bmod M$. If an overflow occurs, i.e., $x_u + r \geq M$, then the true value $X = x_u + r$ and the HP's observed value Y differ by exactly one multiple of M :

$$X = Y + w \cdot M, \quad w \in \{0, 1\}.$$

The HP uses $Y/2^{s_1}$ as the angle, while the true angle should be $X/2^{s_1}$. The difference between these two angles is:

$$\frac{X}{2^{s_1}} - \frac{Y}{2^{s_1}} = \frac{w \cdot M}{2^{s_1}}.$$

If we can ensure that $M/2^{s_1}$ is *exactly* an integer multiple of 2π , then this difference is $w \cdot 2\pi K$, which is a multiple of the sine/cosine period, and the overflow has *zero effect* on the computed result.

In practice, since 2π is irrational and $M/2^{s_1}$ is rational, exact equality is impossible. However, we can approximate it with arbitrary precision: we choose integers M and K such that

$$\left| \frac{M}{2^{s_1}} - 2\pi K \right| < 2^{-(s_{\text{out}} + \gamma)}.$$

The approximation error $\delta = |M - 2\pi K \cdot 2^{s_1}|$ translates to an angle error of $\delta/2^{s_1}$, which after scaling to the output precision contributes less than $2^{-\gamma}$ ULP to the final result. With $\gamma = 2$, this error is safely below 0.25 ULP, and the total error (including rounding) remains sub-ULP.

For example, with $s_1 = 13$, we have $2\pi \cdot 2^{13} = 51471.85 \dots$. Using continued fraction expansion, we can find integers M and K such that $|M - 51471.85K|$ is arbitrarily small. A 32-bit or 64-bit M suffices to make this error negligible for all practical purposes.

The Key Benefit: With this modulus choice, the HP can compute $\sin(Y/2^{s_1})$ and $\cos(Y/2^{s_1})$ *without knowing whether an overflow occurred*. The periodic nature of sine and cosine, combined with the period-matching modulus, ensures that any overflow-induced difference is absorbed into the function's period. The expensive Wrap protocol is therefore *completely eliminated*.

Protocol Description Protocol 6. $\Pi_{\text{SinCos}}^{\text{MCU}}$ (2 communication rounds)

Input: Each party P_i holds $\langle x \rangle_i \in \mathbb{Z}_M$, where $\sum_i \langle x \rangle_i \equiv x_u \pmod{M}$ with $x_u = \lfloor x_{\text{real}} \cdot 2^{s_1} \rfloor + \Delta$.

Output: Each party obtains $\langle \sin(x_{\text{real}}) \rangle_i$ and $\langle \cos(x_{\text{real}}) \rangle_i$ in $\mathbb{Z}_{2^{s_{\text{out}}}}$ (or $\mathbb{Z}_M$ for ring consistency).

Phase 1: Mask (Round 1)

Each party P_i :

1. Generates a random mask $r_i \in_R \mathbb{Z}_M$ using the shared PRG.
2. Computes the masked value:

$$m_i = \langle x \rangle_i + r_i \pmod{M}.$$

3. Sends m_i to the Helper Party (HP).

Phase 2: Compute (Round 2)

The HP performs the following steps:

1. Aggregates the received values:

$$Y = \sum_{i=1}^n m_i \pmod{M}.$$

- Converts Y to a real number representing the masked angle (in radians):

$$\theta_Y = \frac{Y}{2^{s_1}}.$$

Note: The HP does *not* need to know whether Y resulted from an overflow. The period-matching property ensures that any overflow (i.e., $Y = (x_u + r) \bmod M$ instead of $x_u + r$) changes θ_Y by at most $w \cdot M/2^{s_1}$, which is within $2\pi K + \delta$. Since the sine/cosine functions are 2π -periodic, the error introduced by this approximation is bounded by:

$$\frac{\delta}{2^{s_1}} \cdot 2^{s_{\text{out}}} < 2^{-\gamma}.$$

- Computes the sine and cosine values with fixed-point precision:

$$S = \lfloor \sin(\theta_Y) \cdot 2^{s_{\text{out}}} + 0.5 \rfloor, \quad C = \lfloor \cos(\theta_Y) \cdot 2^{s_{\text{out}}} + 0.5 \rfloor.$$

- Generates additive shares of S and C over the output ring $\mathcal{R}$ (choose $\mathcal{R} = \mathbb{Z}_{2^{s_{\text{out}}}}$ for compactness, or $\mathbb{Z}_M$ for ring compatibility):
 - Samples $s_1, \dots, s_{n-1} \in_R \mathcal{R}$ using $\text{ASPRG}_1, \dots, \text{ASPRG}_{n-1}$;
 - Computes $s_n = S - \sum_{i=1}^{n-1} s_i \pmod{|\mathcal{R}|}$;
 - Similarly, samples $c_1, \dots, c_{n-1} \in_R \mathcal{R}$ and sets $c_n = C - \sum_{i=1}^{n-1} c_i \pmod{|\mathcal{R}|}$.
- Sends (s_n, c_n) to party P_n . For $i < n$, party P_i 's share is generated locally via ASPRG_i (no communication).

Phase 3: Unmask (Local, no communication)

Each party P_i (including P_n who received its share) now holds:

- $\langle S \rangle_i$ and $\langle C \rangle_i$: shares of the HP-computed sine/cosine values.
- The aggregate mask $r = \sum_{i=1}^n r_i \pmod{M}$, reconstructable locally via the shared PRG.

To recover the sine/cosine of the original input x_{real} , each party locally computes the following fixed-point constants:

$$\sin_B = \left\lfloor \sin\left(\frac{\Delta + r}{2^{s_1}}\right) \cdot 2^{s_{\text{out}}} + 0.5 \right\rfloor, \quad \cos_B = \left\lfloor \cos\left(\frac{\Delta + r}{2^{s_1}}\right) \cdot 2^{s_{\text{out}}} + 0.5 \right\rfloor.$$

Then, using the angle-sum identity:

$$\sin(x_{\text{real}}) = \sin\left(\frac{Y - (\Delta + r)}{2^{s_1}}\right) = \sin\left(\frac{Y}{2^{s_1}}\right) \cos\left(\frac{\Delta + r}{2^{s_1}}\right) - \cos\left(\frac{Y}{2^{s_1}}\right) \sin\left(\frac{\Delta + r}{2^{s_1}}\right),$$

each party computes its final share:

$$\begin{aligned} \langle \sin \rangle_i &= \left\lfloor \frac{\langle S \rangle_i \cdot \cos_B - \langle C \rangle_i \cdot \sin_B}{2^{s_{\text{out}}}} + 0.5 \right\rfloor \bmod 2^{s_{\text{out}}}, \\ \langle \cos \rangle_i &= \left\lfloor \frac{\langle C \rangle_i \cdot \cos_B + \langle S \rangle_i \cdot \sin_B}{2^{s_{\text{out}}}} + 0.5 \right\rfloor \bmod 2^{s_{\text{out}}}. \end{aligned}$$

These shares satisfy:

$$\sum_{i=1}^n \langle \sin \rangle_i \equiv \lfloor \sin(x_{\text{real}}) \cdot 2^{s_{\text{out}}} + 0.5 \rfloor \pmod{2^{s_{\text{out}}}},$$

and analogously for cosine.

Correctness Argument The protocol’s correctness follows from two observations:

1. **Correct mask-unmask algebra:** The HP computes $Y = \sum_i m_i = (x_u + r) \bmod M$. The parties locally know r and Δ , so they can compute the difference $Y - (\Delta + r)$ modulo the period. The angle-sum identity ensures the combination yields the correct $\sin(x_{\text{real}})$.
2. **Overflow error is absorbed by periodicity:** If $x_u + r \geq M$, then $Y = x_u + r - M$. The angle used by the HP is $Y/2^{s_1}$, while the true angle is $(x_u + r)/2^{s_1}$. The difference is $M/2^{s_1}$, which by construction is within δ of $2\pi K$. Since the sine/cosine functions are 2π -periodic, this difference introduces an error bounded by:

$$\text{ULP error} \leq \frac{\delta}{2^{s_1}} \cdot 2^{s_{\text{out}}} + \text{rounding error} < 2^{-\gamma} + 0.5 < 0.75 \text{ ULP}.$$

With $\gamma = 2$, this is safely below the 1-ULP threshold; in practice, choosing M with $\delta < 2^{-10}$ gives sub-ULP accuracy.

In [25], GELU, Tanh, Sigmoid and softmax each is approximated by a general function $F_4(x) = a_4 \sin(a_3 \sin(a_2 \sin(a_1 \sin(a_0 x))))$. In MCU architecture, they also can be implemented in 4 rounds (in the definition of [25], 8 rounds in our definition). We will discuss them in another paper due to the space limitation.

4 Security Proof

We prove the security of all protocols in the semi-honest model with a non-colluding HP. The proofs follow the simulation paradigm: for any adversary controlling a subset of parties, we construct a simulator that, given only the inputs of the corrupted parties and the outputs, produces a view indistinguishable from the real execution. As the security is discussed in the semi-honest setting, we can only consider the simulation of received data.

Assumptions:

- PRG_0 and ASPRG_i are cryptographically secure (their outputs are indistinguishable from random).
- The HP does not collude with any party.

4.1 Security Analysis of the General Multiplication Protocol

For multiplication and x^k , in HP's view, the received $x_i + r_i$ can be simulated by random numbers and the aggregated value $x + r$ are simulable too. In the adversary's view who corrupted $t(t < n)$ parties, the only received data s_n (if party P_n is corrupted) is indistinguishable from random numbers because at least one share is unknown to the adversary.

The detailed proof for multiplication is as follows.

Theorem 1. *Under the semi-honest, non-colluding HP assumption, Protocol 1 securely computes $\prod_{j=1}^k x_j$ against any adversary corrupting $t < n$ parties.*

Proof. We construct a simulator $\mathcal{S}$ that, given the corrupted parties' inputs and the output product, produces an indistinguishable view.

Simulating the HP's view. The HP receives $a_{j,i} = \langle x_j \rangle_i + \langle r_j \rangle_i$ for each party i and each variable j . For honest parties, each $\langle r_j \rangle_i$ is uniformly random (generated by PRG) and unknown to the HP. Hence each $a_{j,i}$ is uniformly distributed over $\mathbb{Z}_{2^l}$. $\mathcal{S}$ simulates these values by sampling uniform random elements.

Simulating a corrupted party's view. A corrupted party P_i receives:

- If P_n is corrupted, the adversary receives $t_{S,n} = A_S - \sum_{j=1}^{n-1} t_{S,j}$ for each subset $S \subseteq \{1, \dots, k\}$, where $A_S = \prod_{j \notin S} (x_j + r_j)$ and the shares $t_{S,1}, \dots, t_{S,n-1}$ are generated uniformly by $ASPRG_i$ respectively. Since at least one honest party's share is missing, each $t_{S,n}$ seen by the adversary is uniformly random. $\mathcal{S}$ simulates them by sampling uniform random elements.
- No other messages are received (the output shares are computed locally).

The local unmasking step uses only these shares and known masks (reconstructed from PRG), involving no communication.

All simulated messages are uniformly random and independent of honest inputs. The real messages are also uniformly random due to secret sharing and unknown masks. Hence the real and simulated views are indistinguishable, proving the security of Protocol 1.

The security of the power protocol can be proved similarly.

4.2 Security of the Sign Protocol

Theorem 2. *Under the semi-honest, non-colluding HP assumption, Protocol ?? securely computes $\text{sign}(x)$ against any adversary corrupting $t < n$ parties.*

Proof. We construct a simulator $\mathcal{S}$ that, given the corrupted parties' inputs and the output $\text{sign}(x)$, simulates only the messages received by the HP and the corrupted parties.

Simulating the HP's view. The HP receives the following messages.

Round 1 (Masking): The HP receives $m_i = x'_i + s_i \bmod 2^\ell$ from each party. For honest parties, each s_i is uniformly random and unknown to the HP; hence each m_i is uniformly distributed over $\mathbb{Z}_{2^\ell}$ and independent of x'_i . $\mathcal{S}$ samples $m_i \in_R \mathbb{Z}_{2^\ell}$ for each honest party.

Round 3 (Zero detection): The HP receives $r_k \cdot q_{i,j}^{(k)} \bmod p$ from each party for all j, k . The blinding factors $r_k \in \mathbb{Z}_p^*$ are uniformly random and unknown to the HP. For each j, k , since at least one honest party contributes a uniformly random share, each blinded product is uniformly distributed over $\mathbb{Z}_p^*$ unless $V_j - k \cdot 2^{\ell_x} = 0$; that zero case precisely determines whether the output bit is 1. $\mathcal{S}$ simulates all non-zero products by sampling uniform elements from $\mathbb{Z}_p^*$, and for the unique (j, k) that makes $T_{j,k} = 0$ (if the simulated output is 1), it sets the corresponding product to 0. This simulation is statistically indistinguishable because the blinding factors r_k perfectly hide all other information.

The HP's local computations (computing u_j , determining $\text{sign}(x')$, and generating shares) are not messages and need not be simulated.

Simulating a corrupted party's view. A corrupted party receives messages from the HP in two steps.

Round 2 (Decomposed bits): If P_n is corrupted, it receives $\langle u_j \rangle_n \in \mathbb{Z}_{2^{\ell_x}}$ for each $j = 0, \dots, \ell_x$. These are generated as

$$\langle u_j \rangle_n = u_j - \sum_{i=1}^{n-1} \langle u_j \rangle_i \bmod 2^{\ell_x},$$

where $\langle u_j \rangle_i$ for $i < n$ are sampled uniformly by ASPRG_i and are unknown to the adversary. Since at least one honest party exists among $P_1, \dots, P_{n-1}$, each $\langle u_j \rangle_n$ is uniformly distributed over $\mathbb{Z}_{2^{\ell_x}}$ and independent of u_j . $\mathcal{S}$ simulates each by sampling a uniform element from $\mathbb{Z}_{2^{\ell_x}}$.

Final step (Output share): The HP shares the result $\text{sign}(x')$ in arithmetic secret sharing. If P_n is corrupted, it receives

$$\langle \text{sign}(x') \rangle_n = \text{sign}(x') - \sum_{i=1}^{n-1} \langle \text{sign}(x') \rangle_i \bmod M,$$

where $\langle \text{sign}(x') \rangle_i$ for $i < n$ are sampled uniformly by ASPRG_i and are unknown to the adversary. Since at least one honest party exists among $P_1, \dots, P_{n-1}$, $\langle \text{sign}(x') \rangle_n$ is uniformly distributed over $\mathbb{Z}_M$ and independent of $\text{sign}(x')$. $\mathcal{S}$ simulates it by sampling a uniform element from $\mathbb{Z}_M$.

If the adversary corrupts P_i with $i < n$, it receives no incoming messages (its share is generated locally via ASPRG_i). $\mathcal{S}$ reproduces its view directly from its input and local randomness.

The optional blinding step (if $\text{flip} = 1$) involves only local operations and no communication; it is reproduced by $\mathcal{S}$ using the common random bit t from PRG_0 .

Indistinguishability. All simulated messages for the HP are uniformly random (or identically zero only when the output dictates) and independent of honest inputs. All simulated messages received by a corrupted P_n (the shares $\langle u_j \rangle_n$ and $\langle \text{sign}(x') \rangle_n$) are uniformly random due to additive secret sharing over their respective rings. Since at least one honest party's share is unknown to the adversary, the real and simulated views are computationally indistinguishable. Hence Protocol ?? securely computes $\text{sign}(x)$ under the stated assumptions.

4.3 Security of the Wrap Protocol

Theorem 3. *Under the semi-honest, non-colluding HP assumption, Protocol ?? securely computes the overflow bit $w = \mathbf{1}_{x+r \geq M}$ and delivers it only to the HP, against any adversary corrupting $t < n$ parties.*

Proof. We construct a simulator $\mathcal{S}$ that, given the corrupted parties' inputs and the output w (which is the only value the HP learns), simulates only the messages received by the HP and the corrupted parties.

Simulating the HP's view. The HP receives messages in two rounds.

Round 1: The HP receives $m_i = x_i + r_i \bmod M$ from each party. For honest parties, r_i is uniformly random and unknown to the HP; hence each m_i is uniformly distributed over $\mathbb{Z}_M$ and independent of x_i . $\mathcal{S}$ samples $m_i \in_R \mathbb{Z}_M$ for each honest party. (The value B is no longer part of the protocol; the HP receives only m_i in this version.)

Round 3 (Sign subprotocol): The HP invokes the Sign protocol on the shares of $z = r - t$ to obtain $w = \text{sign}(z)$. By Theorem 2, the Sign protocol is secure; its transcript to the HP can be simulated given only the output bit w . $\mathcal{S}$ invokes the Sign simulator, which produces a transcript that is indistinguishable from the real execution.

The HP's local computations — computing $t = \sum_i m_i \bmod M$, generating the shares $t_1, \dots, t_n$ in Round 2, and invoking the Sign protocol — are not received messages and need not be simulated.

Simulating a corrupted party's view. The Wrap protocol itself sends only one message from the HP to the parties: in Round 2, P_n receives the share t_n of t .

If the adversary corrupts P_i with $i < n$, it receives no incoming messages from the Wrap protocol itself. Its view consists solely of its input x_i , its mask r_i , its local share t_i (generated locally via ASPRG_i), and its participation in the Sign subprotocol. $\mathcal{S}$ reproduces the local computations directly; the Sign subprotocol messages are handled below.

If P_n is corrupted, it receives:

$$t_n = t - \sum_{i=1}^{n-1} t_i \pmod{N},$$

where $t_1, \dots, t_{n-1}$ are sampled uniformly by ASPRG_i for $i < n$, and are unknown to the adversary. Since at least one honest party exists among $P_1, \dots, P_{n-1}$,

the sum contains at least one uniformly random term; hence t_n is uniformly distributed over $\mathbb{Z}_N$ and independent of t . $\mathcal{S}$ simulates t_n by sampling a uniform element from $\mathbb{Z}_N$.

The local construction of z shares (Step 3) involves no communication: P_1 computes $z_1 = r - t_1 \bmod N$, and P_i for $i \geq 2$ computes $z_i = -t_i \bmod N$. These are deterministic functions of the parties' local information and are reproducible by $\mathcal{S}$.

Sign subprotocol messages: In Step 4, the parties invoke the Sign protocol. Depending on which parties are corrupted, they may receive messages from the Sign protocol as specified in Protocol ?? (specifically, P_n receives shares $\langle u_j \rangle_n$ and the output share $\langle \text{sign}(z) \rangle_n$). These messages are simulated by invoking the Sign simulator, which is guaranteed by Theorem 2 to produce an indistinguishable view given the corrupted parties' inputs and the output bit w .

No other messages are received by any party.

Indistinguishability. All simulated messages for the HP — the m_i 's in Round 1 and the Sign transcript in Round 3 — are uniformly random or generated by the secure Sign simulator, and are independent of honest inputs.

For corrupted parties, the only Wrap-specific incoming message is t_n sent to P_n , which is uniformly random due to additive secret sharing over $\mathbb{Z}_N$ with at least one honest party's share unknown to the adversary. All other received messages belong to the Sign subprotocol and are simulated indistinguishably by the Sign simulator.

Hence the real and simulated views are computationally indistinguishable, proving the security of Protocol ??.

4.4 Security of the Trigonometric Protocol

Theorem 4. *Under the semi-honest, non-colluding HP assumption, Protocol $\Pi_{\text{SinCos}}^{\text{MCU}}$ securely computes $\sin(x_{\text{real}})$ and $\cos(x_{\text{real}})$ against any adversary corrupting $t < n$ parties.*

Proof. We construct a simulator $\mathcal{S}$ for any set of corrupted parties $\mathcal{C}$ with $|\mathcal{C}| < n$. Given the corrupted parties' inputs $x_{\text{real},i}$ for all $P_i \in \mathcal{C}$ and the protocol outputs $\sin(x_{\text{real}})$ and $\cos(x_{\text{real}})$, the simulator $\mathcal{S}$ produces a view that is computationally indistinguishable from the real execution. Since security is discussed in the semi-honest setting, we only consider the simulation of received messages.

Simulating the HP's view. The HP receives the following messages during protocol execution.

Round 1 (Mask): The HP receives the masked value $m_i = \langle x \rangle_i + r_i \pmod{M}$ from each party P_i . For honest parties ($P_i \notin \mathcal{C}$), the mask r_i is uniformly random over $\mathbb{Z}_M$ generated by the shared PRG_0 and is unknown to the HP. Hence each m_i is uniformly distributed over $\mathbb{Z}_M$ and independent of the honest party's input $\langle x \rangle_i$. $\mathcal{S}$ simulates these values by sampling $m_i \in_R \mathbb{Z}_M$ for each honest party.

Round 2 (Compute): The HP locally aggregates the received values to obtain $Y = \sum_{i=1}^n m_i \bmod M$, computes

$$S = \left\lfloor \sin\left(\frac{Y}{2^{s_1}}\right) \cdot 2^{s_{\text{out}}} + 0.5 \right\rfloor, \quad C = \left\lfloor \cos\left(\frac{Y}{2^{s_1}}\right) \cdot 2^{s_{\text{out}}} + 0.5 \right\rfloor,$$

and generates additive shares of S and C . These local computations involve no incoming messages and need not be simulated. The messages sent by the HP in this round (i.e., the shares (s_n, c_n) sent to P_n) are handled below when simulating a corrupted party's view.

Privacy of the masked input: The HP's view contains only $Y = (x_u + r) \bmod M$, where $r = \sum_{i=1}^n r_i \bmod M$ is uniformly random and independent of x_u . Since r is uniform over $\mathbb{Z}_M$, for any observed Y , the conditional distribution of x_u is perfectly uniform over $\mathbb{Z}_M$:

$$\Pr(x_u = x \mid Y = y) = \frac{1}{M}, \quad \forall x \in \mathbb{Z}_M.$$

Thus the HP's view is information-theoretically independent of x_u . $\mathcal{S}$ outputs only uniformly random values for the honest parties' m_i , producing a simulated view that is statistically indistinguishable from the real view.

Simulating a corrupted party's view. A corrupted party receives messages from the HP only if it is P_n .

Case 1: Corrupted party is P_i with $i < n$. Such a party receives no incoming messages from the HP in this protocol. Its shares s'_i and c'_i are generated locally via ASPRG_i , and the final output shares are computed locally. Its view consists solely of its own input, local randomness, and locally generated shares. $\mathcal{S}$ reproduces this view directly without additional simulation.

Case 2: Corrupted party is P_n . In Round 2, party P_n receives from the HP:

$$s_n = S - \sum_{i=1}^{n-1} s_i \pmod{2^{s_{\text{out}}}}, \quad c_n = C - \sum_{i=1}^{n-1} c_i \pmod{2^{s_{\text{out}}}},$$

where $s_1, \dots, s_{n-1}$ and $c_1, \dots, c_{n-1}$ are sampled uniformly via $\text{ASPRG}_1, \dots, \text{ASPRG}_{n-1}$, respectively, and are unknown to the adversary. Since at least one honest party exists among $P_1, \dots, P_{n-1}$, the corresponding shares s_j and c_j are uniformly random and unknown to the adversary. Therefore, both s_n and c_n are uniformly distributed over $\mathbb{Z}_{2^{s_{\text{out}}}}$ and independent of S and C . $\mathcal{S}$ simulates them by sampling uniform elements from $\mathbb{Z}_{2^{s_{\text{out}}}}$.

The local unmasking step involves no communication. Each party P_i computes its final share using the received (or locally generated) shares and the aggregate mask $r = \sum_i r_i \bmod M$, which is known locally via the shared PRG_0 . Since $\mathcal{S}$ knows the aggregate mask r (as it can reconstruct it from all parties' masks), and has simulated s_n, c_n as uniform random values, it can correctly reproduce the entire local computation for P_n , including the final output shares.

Indistinguishability. For the HP’s view, all simulated m_i values (for honest parties) are uniformly random over $\mathbb{Z}_M$, identically distributed to the real messages in which each m_i is hidden by a uniform random mask r_i . Hence the simulated HP view is statistically indistinguishable from the real view.

For a corrupted party’s view:

- If P_n is not corrupted, no incoming messages are received; the simulated view is identical to the real view.
- If P_n is corrupted, the received shares s_n and c_n in the real execution are uniformly random (because at least one honest party’s share is unknown), and in the simulation they are also uniform random samples. The distributions are identical. All local computations are deterministic functions of the party’s input and simulated randomness, and are reproduced exactly by $\mathcal{S}$.

Since all simulated messages are identically distributed (for the masking and sharing steps) or computationally indistinguishable from the real messages, and $\mathcal{S}$ uses only the corrupted parties’ inputs and the protocol outputs, Protocol $\Pi_{\text{SinCos}}^{\text{MCU}}$ securely computes $\sin(x_{\text{real}})$ and $\cos(x_{\text{real}})$ in the semi-honest model.

4.5 Security of the Exponential Protocol

Theorem 5. *Under the semi-honest, non-colluding HP assumption, Protocol 5 securely computes e^{x_s} against any adversary corrupting $t < n$ parties, with information-theoretic privacy against the HP.*

Proof. We construct a simulator $\mathcal{S}$ that, given the corrupted parties’ inputs and the output $Y = \lfloor e^{x_s} \cdot 2^{s_{\text{out}}} + 0.5 \rfloor$, simulates only the messages received by the HP and the corrupted parties.

Simulating the HP’s view. The HP receives messages in three phases.

Round 1 (Masking): The HP receives $m_i = x_{u,i} + r_i \pmod{2^\ell}$ from each party. For honest parties, r_i is uniformly random over $\mathbb{Z}_{2^\ell}$ and unknown to the HP; hence each m_i is uniformly distributed over $\mathbb{Z}_{2^\ell}$ and independent of $x_{u,i}$. $\mathcal{S}$ samples $m_i \in_R \mathbb{Z}_{2^\ell}$ for each honest party.

Rounds 2–5 (Modified Wrap): The HP participates in the modified Wrap protocol, which outputs $\langle w \rangle_i$ to the parties without revealing w to the HP. By the security of the modified Wrap (a straightforward adaptation of Theorem 3 with the output direction changed from HP to parties), its entire transcript to the HP is simulable given no output. $\mathcal{S}$ invokes the modified Wrap simulator, which produces an indistinguishable view for the HP.

Rounds 7–8 (Multiplication $w \cdot V_0$): The HP participates in the generic multiplication protocol (Protocol 1) on inputs $\langle w \rangle_i$ and $\langle V_0 \rangle_i$. By Theorem 1, the multiplication transcript to the HP is simulable given the product $Q = w \cdot V_0$. $\mathcal{S}$ invokes the multiplication simulator.

The HP’s local computations in Round 6 — computing $Y = \sum_i m_i \pmod{2^\ell}$, evaluating $V_0 = \lfloor \exp(Y/2^{s_1}) \cdot S_V + 0.5 \rfloor$, and generating its shares — are not messages and need not be simulated.

We must argue that the HP's knowledge of $Y = (x_u + r) \bmod 2^\ell$ does not leak x_u . Since r is uniformly random over $\mathbb{Z}_{2^\ell}$ and independent of x_u , the mapping $x_u \mapsto M = x_u + r \pmod{2^\ell}$ is a bijection for any fixed r . Thus for any observed M , the conditional distribution of x_u is perfectly uniform over $\mathbb{Z}_{2^\ell}$:

$$\Pr(x_u = x \mid M) = \frac{1}{2^\ell} \quad \text{for all } x \in \mathbb{Z}_{2^\ell}.$$

Therefore, the HP's view is information-theoretically independent of x_u .

Simulating a corrupted party's view. A corrupted party receives messages from the HP in the following steps.

If the adversary corrupts P_i with $i < n$: It receives no direct messages from the HP in the exponential protocol itself. Its shares of V_0 and Q are generated locally via ASPRG_i ; its share of w is obtained from the modified Wrap protocol (handled by the Wrap simulator); and its participation in the multiplication protocol involves only local computations and sending messages to the HP (which are not received messages). $\mathcal{S}$ reproduces its view directly from its input and local randomness.

If P_n is corrupted: It receives the following messages.

Round 6 (V_0 share): It receives

$$v_n = V_0 - \sum_{i=1}^{n-1} v_i \pmod{2^\kappa},$$

where $v_1, \dots, v_{n-1}$ are sampled uniformly by ASPRG_i for $i < n$, and are unknown to the adversary. Since at least one honest party exists among $P_1, \dots, P_{n-1}$, v_n is uniformly distributed over $\mathbb{Z}_{2^\kappa}$ and independent of V_0 . $\mathcal{S}$ simulates v_n by sampling a uniform element from $\mathbb{Z}_{2^\kappa}$.

Rounds 7–8 (Multiplication shares): The multiplication protocol sends to P_n the share $t_{\{1,2\},n}$ (or more generally, the final share of the product). By Theorem 1, these shares are uniformly distributed over $\mathbb{Z}_{2^\kappa}$ when at least one honest party exists. $\mathcal{S}$ simulates them by sampling uniform elements from $\mathbb{Z}_{2^\kappa}$.

Modified Wrap output: The modified Wrap protocol sends to P_n its share $\langle w \rangle_n$ of the overflow bit. This share is uniformly distributed over $\mathbb{Z}_{2^\kappa}$ (as it is the complement of a sum containing at least one honest party's uniform share). $\mathcal{S}$ simulates it by sampling a uniform element from $\mathbb{Z}_{2^\kappa}$.

No other messages are received. The local combination step — computing Num, right-shifting, and reducing modulo 2^κ — involves no communication and is reproduced by $\mathcal{S}$ using the corrupted party's inputs and simulated randomness.

Indistinguishability. All simulated messages for the HP — the m_i 's, the modified Wrap transcript, and the multiplication transcript — are uniformly random or generated by the respective secure simulators, and are independent of honest inputs.

All simulated messages received by a corrupted P_n — the share v_n , the multiplication shares, and the Wrap output share $\langle w \rangle_n$ — are uniformly random

due to additive secret sharing over $\mathbb{Z}_{2^k}$ with at least one honest party’s share unknown to the adversary.

Since the real and simulated views are identically distributed (for the masking and sharing steps) or computationally indistinguishable (for the PRG/ASPRG outputs and the subprotocol simulators), the protocol securely computes e^{x_s} under the stated assumptions.

5 Experimental Evaluation

We conduct performance experiments to evaluate the multiplication protocol’s scalability and precision experiments for the exponential protocol. For multiplication, we compare against Beaver-based multiplication, which is a standard baseline for secret-sharing-based MPC. Additional experiments, including performance of sign and wrap protocols, are planned for the full version of this work.

5.1 Multiplication Scalability

We evaluate the scalability of the multiplication protocol (Protocol 1) with respect to the number of parties n . The runtime is measured as the total wall-clock time from start to completion, and we compare it against Beaver-based multiplication.

As shown in Figure 1, although MCU requires two communication rounds compared to the single round of Beaver-based multiplication, it achieves lower overall latency. This is because Beaver-based multiplication incurs significantly higher communication volume due to the need to distribute multiple correlated random shares among all pairs of parties. In contrast, our protocol sends only $2n + 1$ ring elements and leverages an active helper to perform the bulk of the computation.

The runtime of our multiplication protocol exhibits near-linear growth with a small slope, remaining consistently low as the number of parties increases. Beaver-based multiplication, however, shows a more rapid increase, becoming substantially slower in medium- to large-party settings. These results indicate that Protocol 1 scales better in multi-party environments and is less sensitive to the number of participants.

5.2 Precision of Exponential Protocol

We test the precision of the exponential protocol when x is from $[-16, 16]$, covering the typical range for machine learning and scientific computing applications. Our experiments show only rounding error exists, confirming sub-ULP accuracy.

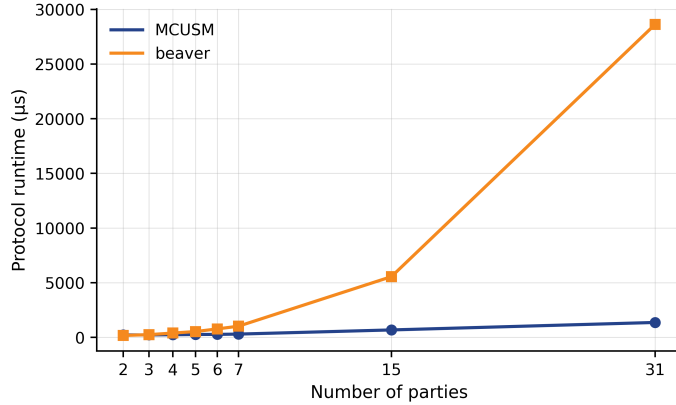


Fig. 1. Protocol runtime versus number of parties for Π_{MCUSM} and the Beaver-based method.

6 Related Work and Discussion

6.1 Existing Approaches

Helper-assisted MPC. As summarized in Table 1, a number of works introduce a semi-honest helper party (HP) to improve efficiency [18,20,10,9,21,17,30,27]. In most of these systems [18,20,10,9,21,17], the HP is responsible for offline preprocessing (e.g., generating Beaver triples) and provides auxiliary assistance during the online phase. Wang et al. [21] construct several multiplication and exponentiation protocols, but their approach differs fundamentally from ours. In the work of Lu et al. [17], the HP participates as one of the parties in the online phase of a two-party computation protocol, serving a different role than in MCU.

In contrast, Zheng et al. [27] and Zhou et al. [30] take a more direct approach: they let the HP compute directly on masked data. Zheng et al. design protocols for ReLU, sigmoid, and tanh using random sign flipping and permutation as masking techniques. Permutation alone does not hide the numerical values; sorting the permuted inputs recovers the same sorted order as the original inputs, which can be used as a fingerprint for target recognition. Thus, permutation alone offers no provable security. Bicoprot [30] focuses exclusively on comparison operations and ReLU; it does not support exponentials or trigonometric functions. Despite these limitations, their insight—allowing the HP to compute on masked values—is highly inspiring. Building on this idea, we propose MCU, which systematically redefines the HP as an active computational engine that operates on additively masked data, enabling constant-round, exact evaluation of a broad class of algebraic and transcendental functions.

Non-linear functions in MPC. Recent years have seen growing interest in evaluating complex nonlinear functions [16,12,6,22,29,24,8] such as GeLU and softmax

Table 1. Helper-assisted MPC works with non-colluding HP assumption

Work	Venue	HP Main Role
SecureML [18]	SP 2017	Triple generation
SecureNN [20]	PETS 2019	MSB computation
Asterisk [10]	SP 2024	Preprocessing & online
Asynchronous MPC [11]	ePrint 2025	Preprocessing & online
Wang et al. [21]	arXiv 2025	Preprocessing & online
Malicious MPC [17]	TDSC 2023	Preprocessing & online participant
Zheng et al. [27]	KBS 2022	Direct computation
Bicopter [30]	SP 2023	Direct computation

within MPC, driven by the need to protect data privacy in deep learning applications. Most of these works rely on polynomial approximations or iterative methods, which inevitably introduce accuracy loss and require many communication rounds. In contrast, MCU evaluates exponentials and trigonometric functions exactly (up to floating-point rounding) without polynomial approximations and without preprocessing.

6.2 Discussion

Potential concerns. A natural question regarding the MCU architecture is whether the introduction of an active HP introduces a single point of failure or a new trust assumption. We acknowledge that the non-colluding HP assumption is central to the security of MCU; however, this assumption is widely adopted in the MPC literature and aligns with many real-world deployment scenarios, such as regulated financial markets, healthcare data aggregators, and industry consortia, where a neutral third party naturally exists.

Another potential concern is the reliance on synchronized pseudorandom generators for mask generation. While this eliminates offline preprocessing, it requires that all parties share a common PRG seed. This is a lightweight setup that can be established once before any computation, and the use of cryptographic PRGs ensures security under standard assumptions.

Limitations and Future Work. MCU currently supports multiplication, power, exponentials, and trigonometric functions. Functions requiring division (e.g., softmax, sigmoid, GeLU, $1/x$, $\log(x)$, $\sqrt{x}$) are not directly supported under the additive-mask paradigm. These functions require secure division or reciprocal protocols—a direction we plan to explore in future work by integrating standard MPC techniques (e.g., Newton iteration with our multiplication protocol).

7 Conclusion

We introduced MCU, a novel MPC architecture that fundamentally redefines the role of a helper party from a passive randomness source to an active computational engine. In the MCU framework, parties additively mask their inputs with

synchronized randomness, the HP computes the target function directly on the aggregated masked data, and the parties then unmask the result using locally known masks. This approach enables scalable, low constant-round, and provably secure evaluation of algebraic and transcendental functions without any offline preprocessing or polynomial approximations.

We presented a suite of protocols under the MCU framework, covering multiplication, power, comparison, wrap, exponential, and trigonometric functions. The multiplication and power protocols accelerate polynomial evaluation, benefiting many approximation-based or M2A-based schemes. Our exponential and trigonometric protocols achieve sub-ULP accuracy with 40-bit statistical security, setting a new standard for preprocessing-free transcendental function evaluation.

While the core idea of MCU—mask, compute, unmask—appears simple, its instantiation requires careful algebraic insights to derive unmasking identities for each target function, and for composite functions, sophisticated mask isolation techniques to prevent information leakage.

MCU offers a new paradigm for efficient and scalable privacy-preserving computing, particularly for scientific computing, differential privacy mechanisms, and advanced neural network inference. We believe that the active-helper model opens up new avenues for research in helper-assisted MPC, and the algebraic unmasking techniques developed in this work may inspire further exploration of direct computation on masked data for other complex functions in standard MPC settings.

References

1. Donald Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 446–455, 1997.
2. Yuntian Chen, Xianjia Meng, Zhiying Shi, Zhiyuan Ning, and Jingzhi Lin. SecureTm: Private inference for transformer-based large model with mpc. *Information Sciences*, 667:120429, 2024.
3. Ke Cheng, Yuheng Xia, Anxiao Song, Jiakuan Fu, Wenjie Qu, Yulong Shen, and Jiaheng Zhang. Mosformer: Maliciously secure three-party inference framework for large transformers. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, page 4124–4138, New York, NY, USA, 2025. Association for Computing Machinery.
4. Ivan. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology – CRYPTO 2007*, 2007.
5. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference*, Lecture Notes in Computer Science. Springer, 2012.
6. Ye Dong, Wen-Jie Lu, Yancheng Zheng, et al. PUMA: Secure inference of LLaMA-7B in five minutes. *Security and Safety*, 4(2025014), 2025.
7. Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. *Proc. Priv. Enhancing Technol.*, 2024:61–79, 2024.

8. Mazharul Islam et al. Compact: Approximating complex activation functions for secure computation. *Proceedings on Privacy Enhancing Technologies*, 2024:25–41, 2023.
9. Banashri Karmaka, Aniket Kate, and et al Shravani Patil. Breaking the barrier for asynchronous mpc with a friend. IACR ePrint Archive, Report 2025/???, 2025.
10. B. Karmakar et al. asterisk: Superfast MPC with a friend. In *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.
11. Banashri Karmakar, Aniket Kate, Shravani Patil, Arpita Patra, Sikhar Patranabis, Protik Paul, and Divya Ravi. Breaking the barrier for asynchronous mpc with a friend, 2025. <https://eprint.iacr.org/2025/1736>.
12. A. Y. L. Kei and H. S. M. Chow. SHAFT: Secure, handy, accurate, and fast transformer inference. In *Network and Distributed System Security Symposium (NDSS) 2025*, 2025.
13. M. Keller et al. MasCOT: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.
14. Zhongkai Li, Shuyang Fan, and Lingfei Jin. Protocol design of non-linear function in secure multi-party computation based on secret sharing. *Journal of Information Security and Applications*, 96:104293, 2026.
15. Fengrun Liu, Xiang Xie, and Yu Yu. Scalable multiparty computation protocols for machine learning in the honest-majority setting. In *USENIX Security Symposium*, 2024.
16. Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. Bumblebee: Secure two-party inference framework for large transformers. In *Proceedings of the 2025 Network and Distributed System Security Symposium (NDSS)*, 2025.
17. Yibiao Lu, Bingsheng Zhang, and Kui Ren. Maliciously secure mpc from semi-honest 2pc in the server-aided model. *IEEE Transactions on Dependable and Secure Computing*, 21(4):3109–3125, 2024.
18. Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.
19. Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4753–4771, 2024.
20. S. Wagh, D. Gupta, and N. Chandran. SecureNN: 3-party secure computation for neural network training. In *Proceedings on Privacy Enhancing Technologies (PETS) 2019*, 2019.
21. Kaiwen Wang, Xiaolin Chang, Junchao Fan, and Yuehan Dong. Efficient private inference based on helper-assisted malicious security dishonest majority mpc, 2025.
22. Shuya Yang, Xiaodong Li, and Jianyi Zhang. Privacy-preserving computing scheme for ciphertext neural network training. In *Proceedings of the 2024 3rd International Conference on Cryptography, Network Security and Communication Technology (CNSCT '24)*, pages 148–152, New York, NY, USA, 2024. Association for Computing Machinery.
23. A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, 1982.
24. C. Zeng, D. He, Q. Feng, X. Yang, and Q. Luo. SecureGPT: A framework for multi-party privacy-preserving transformer inference in GPT. *IEEE Transactions on Information Forensics and Security*, 19:9480–9493, 2024.

25. Jinghao Zhao, Hongwei Yang, Xiaoyu Song, Meng Hao, Hui He, and Weizhe Zhang. CRAfT: Constant-round non-polynomial approximation and P2P network for secure transformer inference. Cryptology ePrint Archive, Paper 2026/1292, 2026. URL: <https://eprint.iacr.org/2026/1292>.
26. Jinghao Zhao, Hongwei Yang, Bobo Wang, Lichunxi Yang, Juncheng Li, Xiangrui Zeng, Meng Hao, Desheng Wang, Hui He, and Weizhe Zhang. PrivaLean: Low-latency and high-accuracy system for secure 2PC inference. Cryptology ePrint Archive, Paper 2026/554, 2026. URL: <https://eprint.iacr.org/2026/554>.
27. Fei Zheng, Chaochao Chen, Xiaolin Zheng, et al. Towards secure and practical machine learning via secret sharing and random permutation. *Knowledge-Based Systems*, 245:108609, 2022.
28. Yu Zheng, Qizhi Zhang, Sherman S. M. Chow, Yuxiang Peng, Sijun Tan, Lichun Li, and Shan Yin. Secure softmax/sigmoid for machine-learning computation. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, page 463–476, 2023.
29. Li Zhengyi, Yang Kang, Tan Jin, et al. Nimbus: Secure and efficient two-party inference for transformers. In *Advances in Neural Information Processing Systems*, volume 37, pages 21572–21600, 2024.
30. Lijing Zhou, Qingrui Song, Su Zhang, Ziyu Wang, Xiang-Yu Wang, and Yong Li. Bicaptor 2.0: Addressing challenges in probabilistic truncation for enhanced privacy-preserving machine learning. *ArXiv*, abs/2309.04909, 2024. URL: <https://api.semanticscholar.org/CorpusID:261682275>.