M&M: Secure Two-Party Machine Learning through Modulus Conversion and Mixed-Mode Protocols

Ye Dong*, Wen-jie Lu†, Xiaoyang Hou†, Kang Yang‡, Jian Liu†
*National University of Singapore, Singapore
†Zhejiang University, China
‡State Key Laboratory of Cryptology, China

***Abstract*—Secure two-party machine learning has made substantial progress through the use of mixed-mode protocols, but existing approaches often suffer from efficiency bottlenecks due to inherent mismatch between optimal domains of various cryptographic primitives. In response to these challenges, we introduce framework M&M, which features an efficient modulus conversion protocol. This breakthrough enables seamless integration of the most suitable cryptographic subprotocols within their optimal modulus domains with a minimal modulus conversion overhead. We further establish new benchmarks and practical optimizations for the performance of fundamental primitives, namely comparison and multiplication, across various two-party techniques.By incorporating these techniques, M&M demonstrates significant performance enhancements over state-of-the-art solutions: i) we report a $6\times$-$100\times$ improvement for approximated truncations with 1-bit error tolerance; ii) an average of $5\times$ (resp. $4\times$) reduction in communication (resp. runtime) for machine learning functions; iii) and a 25%-99% improvement in cost-efficiency for private inference of deep neural networks and 50% improvement in private training of gradient boosting decision trees.**

***Index Terms*—Secure Two-Party Computation, Homomorphic Encryption, Oblivious Transfer, Privacy-Preserving Machine Learning**

I. Introduction

To alleviate privacy concerns arising from machine learning (ML), recent works have introduced cryptographic frameworks to enable privacy-preserving ML. Secure two-party computation (2PC) [1] is a powerful cryptographic technique that allows two parties to jointly compute a function over their inputs while keeping those inputs private. To optimize the performance of 2PC-based ML, modern approaches, such as those presented in [2]–[7], often use a combination of different cryptographic protocols, each suited to specific types of computations. For example, Homomorphic Encryption (HE) is typically used for linear operations that are best represented as arithmetic circuits, such as multiplications and convolutions. On the other hand, Boolean circuits, which are more efficient for nonlinear operations like comparisons, are usually handled by Garbled Circuits (GC) [8] or Oblivious Transfer (OT).

To maintain consistency and efficiency, these works generally use a single type of modulus across the entire ML computation. For instance, some methods [3], [9] operate under a prime modulus, which is well-suited for HE, while others [4], [6], [10] use a two-power modulus, which is more efficient for GC/OT protocols. Although some works dynamically adjust modulus size [5], [10], [11], they still adhere to a single type of modulus throughout the computation.

However, a significant challenge arises from the mismatch between the optimal domains for these cryptographic techniques. HE-based protocols typically perform better with a prime modulus, while GC/OT-based protocols intrinsically favor a two-power modulus. For example, secure multiplications using HE can enjoy up to *three times* higher throughput with a prime modulus compared to a two-power modulus [12]. Conversely, the size of a circuit for arithmetic addition over a prime modulus is nearly *double* that over a two-power modulus using GC/OT. This mismatch forces current frameworks to compromise: *they either suffer from inefficient GC/OT-based operations over a prime field [3], [9], or incur higher costs for HE-based operations over a two-power ring [4], [6].*

One major reason for this suboptimal performance is *the absence of an efficient protocol for converting between different moduli, particularly between prime and two-power moduli.* Existing conversion approaches are facing efficiency barriers or operational constraints: i) Modulus conversion methods such as [5], [13] rely on costly private comparison protocols to handle this conversion. These comparisons typically require extensive communication, either in the form of exchanging a large number of bits [8] or multiple rounds of communication [4], making them inefficient for practical use. ii) ABY [14], [15] and (extended) doubly-authenticated shared bits [7], [16] support share conversion between arithmetic and boolean circuits, but directly extending them to support different arithmetic moduli requires running binary addition circuits, which is also communication-intensive. This effectively negates the potential efficiency gains from mixed-modulus computation. iii) [17], [18] avoid private comparison by using two odd moduli; this limitation excludes the efficiency benefits of two-power rings that are crucial for GC/OT optimization. Besides, recent works [19], [20] also highlight the emergent efficiency requirements of 2PC-based ML, but few efforts have explicitly addressed the cost of modulus conversion between different domains, which is crucial for supporting different computation types in 2PC privacy-preserving ML.

Motivating Scenario. To illustrate the motivation for leveraging different sharing moduli in 2PC-based secure machine learning, consider the scenario of private neural network inference. Secure two-party computation techniques are em-

†Wen-jie Lu is the corresponding author.

Ye Dong (Email: dongye@nus.edu.sg). Wen-jie Lu (Email: fionser@gmail.com). Xiaoyao Hou (Email: xiaoyanghou@zju.edu.cn). Kang Yang (Email: yangk@sklc.org). Jian Liu (Email: liujian2411@zju.edu.cn).

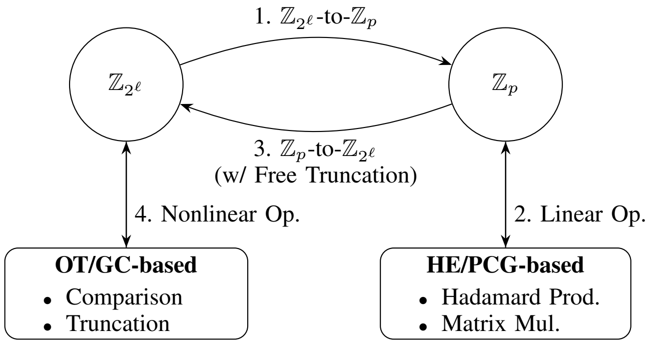

Fig. 1: Overview of our M&M framework that allows two mixings: mixed moduli and mixed-mode protocols.

ployed to safeguard both the client's input privacy and the confidentiality of the neural network model. A typical neural network computation consists of a repeating sequence of linear transformation layers followed by non-linear activation layers. Linear transformations are more efficiently represented using arithmetic circuits, while Boolean circuits are better suited for activation functions. A natural design choice is to utilize HE-based approaches for the linear layers, such as those in [9], [21], which are typically defined over a prime modulus. These can be combined with GC/OT-based approaches for evaluating the non-linear activation layers, as in [2], [4], [5], which are 2×–3× more efficient when the sharing is over a two-power modulus. When efficient modulus conversion is possible, it allows greater flexibility in choosing the most optimal subprotocols. Given the above analysis, we ask the following question:

Can we design an efficient two-party modulus conversion technique that the overall performance can be enhanced, as long as the overhead of the conversion is less than the performance gain achieved by avoiding suboptimal protocols?

We answer the above question affirmatively by proposing framework M&M as Fig 1, that allows mixed moduli and mixed-mode protocols.

A. Technical Overview

Efficient Modulus Conversion for Signed Numbers. We introduce an efficient conversion protocol applicable to two general moduli. Our protocol operates in *two rounds* and, through amortization, exchanges *only one element* over the destination modulus. Our key insights are as follows:

- We propose a uniform method to compute the wrap-bit for secret shares over a general modulus without using a private comparison protocol. Specifically, we compute the wrap-bit using a logical OR of two private bits when the underlying ***message is positive***. For instance, given $x = x_0 + x_1 \bmod 2^\ell$, the wrap-bit $w = \mathsf{sgn}(x_0) \vee \mathsf{sgn}(x_1)$ if $\mathsf{sgn}(x) = 0$, where $\mathsf{sgn}(\cdot)$ returns sign-bit, i.e., the most significant bit in $\mathbb{Z}_{2^\ell}$. For an odd modulus p, we naturally extend the sign-bit definition as $\mathsf{sgn}_p(y) = \mathbf{1}\{y \geq \lceil p/2 \rceil\}$ for $y \in \mathbb{Z}_p$. The wrap-bit for an odd modulus share, say $y = y_0 + y_1 \bmod p$, is similarly computed via a logical OR when the underlying message is positive, i.e., $\mathbf{1}\{y_0 + y_1 \geq p\} = \mathsf{sgn}_p(y_0) \vee \mathsf{sgn}_p(y_1)$ if $\mathsf{sgn}_p(y) = 0$.
- The logical OR should ideally be computed as an additive share over the destination modulus K. Previous approaches compute the logical OR as a Boolean share using 1-out-of-2 OT, followed by a conversion of the Boolean share to an arithmetic share using Delta OT, which requires a total of 4 rounds. In contrast, we directly obtain the logical OR as an arithmetic share using Delta OT, which requires only two rounds. Essentially, Delta OT can generate the logical AND correlation $\alpha \cdot \delta \bmod K$ between the two parties. We then obtain the logical OR correlation via the transformation $\alpha \vee \beta = 1 - (1-\alpha)\cdot(1-\beta) \bmod K$. By instantiating the Delta OT using Ferret OT [22], each logical OR only *exchanges one ring element in a sense of amortization*.
- For a share of $x \in \mathbb{Z}_M$ with an unknown sign, we propose *add-then-subtract* operation with the positive number L to facilitate a simple approach for modulus conversion of signed numbers: We first compute the wrap-bit of the share $x + L$, where L is a large positive number, such as $L = \lceil M/4 \rceil$. We heuristically assume that $x + L$ is positive, enabling us to apply the efficient logical-OR protocol for wrap-bit computation. We initially obtain a share of $x + L$ over the destination modulus K, and subsequently remove the offset by subtracting L. For example, let $M = 8$, $K = 11$, and $L = 2$. For $x = 6$ (which represents -2 in $\mathbb{Z}_8$), $y = (6 + 2 \bmod M) - 2 = 9 \bmod K$, which is also the representative of -2 in $\mathbb{Z}_{11}$.

Remark 1 (1-Bit Constraint)**.** *The positive heuristic reduces the representation of modulus M by one bit. While $\mathbb{Z}_M$ can natively represent values in $[\lceil M/2 \rceil - M, \lceil M/2 \rceil)$, the heuristic requires values to lie within narrower range $[-\lceil M/4 \rceil, \lceil M/4 \rceil)$ to ensure correctness. We systematize how to satisfy this constraint through two practical strategies:*

- ***Modulus Expansion****: Existing implementations [4], [6], [23] typically use well-tested bitwidths to ensure a safe bound $|x| < \lceil M/2 \rceil$ for fixed-point arithmetic. We can satisfy the constraint by expanding $M' = 2M$ (adding one bit) in M&M with negligible efficiency impact.*
- ***Intrinsic Bounds****: Empirical studies [2], [11], [24] show that intermediate values in ML models typically satisfy $|x| \ll M$. Even for advanced large language models (e.g., BERT, GPT-2), as in [11, Fig. 7, Appendix F], most of the hidden states in Transformer blocks are close to zero (with only a small proportion of outliers, but still < 500), well within practical modulus boundaries such as $M = 2^{64}$.*

Prior work [10], [25] has exploited this constraint in privacy-preserving ML. Our experimental validation in §VI-C confirms model performance remains unaffected by directly using well-tested modulus sizes and fractional precisions, demonstrating its practicality for real-world applications.

Efficient Modulus Conversion with Free Truncation. We also develop a specialized conversion protocol from a prime modulus p to a two-power modulus 2^ℓ, which allows for arithmetic right-shift (i.e., truncation) at no additional cost. Specifically, given a share $x = x_0 + x_1 \bmod p$, we compute a share $y = y_0 + y_1 \bmod 2^\ell$ such that y represents the arithmetic right-shift of x, e.g., $y = x \gg d$ for d units. We observe that the wrap-bit $w = \mathbf{1}\{x_0 + x_1 \geq p\}$, computed during the

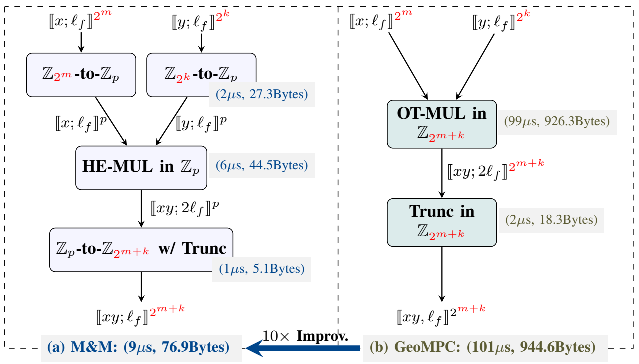
$[\![x;\ell_f]\!]^{2^m}$
$[\![y;\ell_f]\!]^{2^k}$
$\mathbb{Z}_{2^m}$-to-$\mathbb{Z}_p$
$\mathbb{Z}_{2^k}$-to-$\mathbb{Z}_p$
(2μs, 27.3Bytes)
$[\![x;\ell_f]\!]^p$
$[\![y;\ell_f]\!]^p$
HE-MUL in $\mathbb{Z}_p$
(6μs, 44.5Bytes)
$[\![xy;2\ell_f]\!]^p$
$\mathbb{Z}_p$-to-$\mathbb{Z}_{2^{m+k}}$ w/ Trunc
(1μs, 5.1Bytes)
$[\![xy;\ell_f]\!]^{2^{m+k}}$
$[\![x;\ell_f]\!]^{2^m}$
$[\![y;\ell_f]\!]^{2^k}$
OT-MUL in $\mathbb{Z}_{2^{m+k}}$
(99μs, 926.3Bytes)
$[\![xy;2\ell_f]\!]^{2^{m+k}}$
Trunc in $\mathbb{Z}_{2^{m+k}}$
(2μs, 18.3Bytes)
$[\![xy,\ell_f]\!]^{2^{m+k}}$
(a) M&M: (9μs, 76.9Bytes)
10× Improv.
(b) GeoMPC: (101μs, 944.6Bytes)

Fig. 2: Secure non-uniform fixed point multiplication of (a) our M&M and (b) GeoMPC. $m = 18$, $k = 20$, $\ell_f = 12$, and prime $p \approx 2^{m+k}$. Amortized communication and time of each subprotocol is tagged.

modulus conversion, can also be leveraged for the truncation. Informally, $y = (x_0 \gg d) + (x_1 \gg d) - w \cdot (p \gg d) \bmod 2^\ell$. This equation already demonstrates a modulus conversion with free truncation, given the wrap-bit w. By employing this specialized conversion, we can achieve optimal efficiency for both fixed-point multiplications over a prime modulus and nonlinear operations over a two-power modulus, thereby providing a highly efficient framework for secure two-party machine learning that leverages mixed-mode protocols and secret sharing moduli.

Remark 2 (Comparison with GeoMPC [10])**.** *The very recent work GeoMPC also adopts the 1-bit heuristic constraint for efficient modulus conversion in 2PC. However, M&M advances beyond GeoMPC by embracing a fundamentally different mixed-mode design philosophy, which leads to broader applicability and higher efficiency:*

- *GeoMPC operates entirely within the OT-based paradigm, which is inherently inefficient for linear operations and constrained to two-power moduli. It relies on the 1-bit constraint to eliminate costly comparisons in operations such as truncation, signed extension, and non-uniform multiplication—but only within this restricted setting.*
- *In contrast, M&M is built on a mixed-mode design that flexibly integrates GC/OT-based protocols over two-power rings with HE-based protocols over prime fields. This philosophy not only motivates our efficient heuristic modulus conversion independently but also enables a broader class of modulus conversion types and composition of cryptographic techniques.*

Serving as a bridge between distinct cryptographic backends, M&M significantly improves the performance of 2PC. For instance, Figure 2 shows our method achieves up to 90% reduction in communication and runtime for secure fixed-point non-uniform multiplication in a MAN setting (100 Mbps bandwidth, 40 ms latency), compared to GeoMPC.

B. Contributions

Efficient Modulus Conversion. In M&M, our core contribution is designing efficient modulus conversion protocols that efficiently bridge the $\mathbb{Z}_{2^\ell}$ and $\mathbb{Z}_p$ worlds in secure two-party computation, aiming at minimizing the overhead concerns. This enables M&M to seamlessly combine the most effective protocols from both worlds: nonlinear computations such as comparisons and truncations are performed in GC/OT over a two-power modulus 2^ℓ, while linear computations such as multiplications are executed in HE over a prime modulus.

Benchmarks & Optimizations. To choose the most optimal subprotocols, we comprehensively benchmark existing approaches for the most basic two primitives, comparison and multiplication, focusing on their *cost-efficiency*. Our results highlight that the OT-based approach [4] is the most cost-effective for comparison, and we also introduce an optimization for the comparison protocol [4], achieving an 11% reduction in communication costs by optimizing correlated AND gates. Specifically, we demonstrate that an extended Random OT-based approach [26] outperforms the 1-of-2 OT-based method in [4], reducing communication cost per gate *from 23 bits to 8 bits*. On the other hand, we find secure multiplication using lattice-based HE, as in [27], offers superior cost-efficiency over other methods. We establish new benchmarks for the performance of various two-party applications, which might be of independent interest.

Implementations & Evaluations. We implement M&M in C++ and demonstrate how our techniques can significantly enhance the efficiency of 2PC ML computation, including: i) We achieve 6×-100× improvements for 1-bit error approximated truncations over [4], [6], [10]. ii) For ML functions like non-uniform multiplication and non-linear functions, we reduce the communication (resp. runtime) by 5× (resp. 4×) on average compared to [10], [23]. iii) Additionally, we report notable cost-efficiency gains of 25%–99% in the private inference of convolutional neural networks [6], [10], 28% in the private inference of large language models [23], and 50% in the private training of gradient boosting decision trees [28]. The source code is available https://github.com/CPS4AI/OpenMM.

Organization. The rest of this work is organized as follows: In § II, we introduce the preliminaries and background. We introduce our efficient modulus conversions in § III. In § IV, we review and benchmark existing approaches, as well as our optimizations for private comparisons and share multiplications. We propose several optimized ML building blocks in § V. We give our experimental evaluations and analysis in § VI, and summarize related works in§ VII. Finally, we give more discussion in § VIII and conclude this work in § IX.

II. Preliminaries & Background

A. Notations and Definitions

Besides the notations defined in Table I, we define the following functions. $\mathbf{1}\{\mathcal{P}\}$ denotes the indicator function that is 1 when the predicate $\mathcal{P}$ is true and 0 when $\mathcal{P}$ is false. The modulo operation $x' = x \bmod M$ maps $x \in \mathbb{Z}$ to a representation $x' \in \mathbb{Z}_M$. A division always results in a real number. The function $\mathsf{rep}_M : \mathbb{Z}_M \mapsto \mathbb{Z}$ maps the unsigned numbers in $\mathbb{Z}_M$ to the signed numbers in $\mathbb{Z}$ as follows.

$$\mathsf{rep}_M(x) = \begin{cases} x \in \mathbb{Z}^+ & \text{if } x \in [0, \lceil \frac{M}{2} \rceil) \\ x - M \in \mathbb{Z}^- & \text{if } x \in [\lceil \frac{M}{2} \rceil, M) \end{cases}.$$

TABLE I: Notations.

Symbols	Description
$\mathbb{R}, \mathbb{Z}, \mathbb{B}$	the sets of real, integral numbers, and binary
$x \xleftarrow{\$} \mathcal{D}$	x is taken from a set $\mathcal{D}$ uniformly at random
$[n]$	the set $\{0, 1, \cdots, n-1\}$ $(n \geq 1)$
$\mathbb{Z}_M$	the set of integers $\mathbb{Z} \cap [0, M)$ $(M > 2)$
$\wedge, \vee, \oplus$	logical AND, logical OR and logical XOR
$\mathbf{v}$, $\mathbf{v}[i]$	vector and the i-th entry
$\mathbf{A}$, $\mathbf{A}[i,j]$	two dim. matrix and the (i,j) entry
$\mathbf{v} \odot \mathbf{u}$, $\mathbf{A} \odot \mathbf{B}$	Hadamard product
$R_{N,q}$	the polynomial ring $\mathbb{Z}_q[X]/(X^N + 1)$
$\lceil\cdot\rceil$, $\lfloor\cdot\rfloor$, $\lfloor\cdot\rceil$	ceiling, flooring, and rounding to nearest

We define a sign function for the unsigned elements in $\mathbb{Z}_M$ as $\mathsf{sgn}_M(x) = \mathbf{1}\{\mathsf{rep}_M(x) < 0\} = \mathbf{1}\{x \geq \lceil M/2 \rceil\}$. Finally, we define truncation $\mathsf{trc}_M : \mathbb{Z}_M \times \mathbb{Z}^+ \mapsto \mathbb{Z}_M$ as

$$\mathsf{trc}_M(x; d) = \lfloor \frac{\mathsf{rep}_M(x)}{2^d} \rfloor \bmod M.$$

Recall that the flooring $\lfloor\cdot\rfloor$ on negative numbers would round to $-\infty$. For instance $\lfloor -0.4 \rfloor = -1$. This truncation is equivalent to an arithmetic right-shift when M is a two-power modulus. We will omit the subscript and write $\mathsf{rep}(x)$, $\mathsf{sgn}(x)$ and $\mathsf{trc}(x; d)$ if modulus M is clear from the context.

Additive Secret Sharing: $[\![x]\!]^M$ denotes an additive sharing of the value $x \in \mathbb{Z}_M$. We write $[\![x]\!]^M = ([\![x]\!]^M_0, [\![x]\!]^M_1)$ where P_0 holds $[\![x]\!]^M_0$ and P_1 holds $[\![x]\!]^M_1$ with $x = [\![x]\!]^M_0 + [\![x]\!]^M_1 \bmod M$. We drop the superscript when it is clear from context. $M = 2$ (a.k.a., bit-width $\ell = 1$) is for Boolean share, $M > 2$ (i.e., $M = 2^{64}$ or large prime) refers to Arithmetic share. By $[\![x]\!]^M_0$ distributes uniformly at random over $\mathbb{Z}_M$, probability $\Pr([\![x]\!]^M_0 + [\![x]\!]^M_1 \geq M) = \Pr(x < [\![x]\!]^M_0) = 1 - (x+1)/M$.

B. Threat Model

We consider secure two-party computation where both parties, designated as P_0 and P_1, provide their private inputs. Notably, we **do not** consider a third-party dealer as [29]–[32]. We operate under the semi-honest adversary model, assuming a static and computationally bounded adversary who attempts to infer additional information from the messages observed during execution [33]. This model is widely used due to its efficiency and reflects realistic settings where parties follow the protocol due to legal and regulatory compliance.

C. Objectives

We focus on the design and implementation of cost-effective secure two-party computation, accounting for both the costs incurred during the precomputation phase and online evaluation. Specially, we base our expenditure measurements on the current AWS on-demand pricing[1]. The computation cost is set at \$0.05132 per virtual CPU core per hour[2]. For data transfer, we distinguish between two types of communication costs: intra-AWS server communication, priced at \$0.02 per GB, and communication between an AWS server and the Internet, which costs \$0.09 per GB.

[1]https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls

[2]The on-demand pricing for the c7a.48xlarge instance is \$9.85344 per hour, equipped with 192 vCPU cores running at 3.7GHz.

TABLE II: (Amortized) Communication complexity of various OT protocols. λ denotes the security parameter, ℓ is the message width, and $n \geq 3$.

Construction	$\binom{n}{1}$-$\mathtt{OT}_\ell$	$\mathtt{OT}_\ell$	Δ-$\mathtt{OT}_\ell$	$\mathtt{ROT}_\ell$
IKNP OT [34]	$n\ell + 2\lambda$	$2\ell + \lambda$	$\ell + \lambda$	$\ell + \lambda$
Ferret OT [22]	$n\ell + \log_2 n$	$2\ell + o(1)$	$\ell + o(1)$	$o(1)$

We designate the expenditure (i.e., the total costs of computation and communication) under these two communication prices as the A2A cost and A2I cost, respectively. For instance, suppose both P_0 and P_1 each used c CPU cores to run a secure protocol for t hours, with P_0 sending s GB and P_1 sending r GB of data during the computation. The A2A cost for this protocol is calculated as $2 \cdot t \cdot c \cdot 0.05132\$ + (s + r) \cdot 0.02\$$, and the A2I cost as $2 \cdot t \cdot c \cdot 0.05132\$ + (s + r) \cdot 0.09\$$.

D. Oblivious Transfer

Our protocols rely heavily on oblivious transfer (OT) for nonlinear computations, such as comparisons. The ideal functionality for a single 1-out-of-2 OT on messages from a finite field $\mathbb{F}$ is specified as follows, along with the Delta OT and Random OT variants, where the sender's messages satisfy a fixed correlation or are sampled at random:

$$\begin{aligned}
&\mathcal{F}_{\mathtt{OT}} : ((m_0, m_1) \in \mathbb{F}^2, c \in \mathbb{B}) \mapsto (\perp, m_c \in \mathbb{F}) \\
&\mathcal{F}_{\Delta\text{-}\mathtt{OT}} : (\Delta \in \mathbb{F}, c \in \mathbb{B}) \mapsto (m_0 \in \mathbb{F}, m_0 + c \cdot \Delta \in \mathbb{F}) \\
&\mathcal{F}_{\mathtt{ROT}} : (\perp, c \in \mathbb{B}) \mapsto ((r_0, r_1) \in \mathbb{F}^2, r_c \in \mathbb{F})
\end{aligned}$$

We use Ferret protocol [22] for Delta OT over a two-power modulus and a prime modulus. For special case $\mathbb{F} = 2^\ell$, we simplify the notation to $\mathtt{OT}_\ell$, Δ-$\mathtt{OT}_\ell$, and $\mathtt{ROT}_\ell$. Additionally, we use $\binom{n}{1}$-$\mathtt{OT}_\ell$ to denote a general 1-out-of-n OT for $n > 2$ over ℓ-bit messages. Table II summarizes the communication complexities of the state-of-the-art OT implementations.

E. Lattice-based Homomorphic Encryption

A homomorphic encryption (HE) of x enables computing the encryption of $F(x)$ without the knowledge of the decryption key. In this work, we use an HE scheme that is based on Ring Learning-with-Error (RLWE) [35]. The RLWE scheme is defined by a set of public parameters $\mathsf{HE.pp} = \{N, q, p\}$. We leverage the following functions.

- **KeyGen.** Generate the RLWE key pair $(\mathsf{sk}, \mathsf{pk})$ where the secret key $\mathsf{sk} \in R_{N,q}$ and the public key $\mathsf{pk} \in R^2_{N,q}$.
- **Encryption.** An RLWE ciphertext is given as a polynomial tuple $(\hat{b}, \hat{a}) \in R^2_{N,q}$. We write $\mathsf{Enc}^{q,p}_{\mathsf{pk}}(\hat{m})$ to denote the encryption of $\hat{m} \in R_p$ under a key pk.
- **Addition** ($\boxplus$)**.** Given two RLWE ciphertexts $\mathsf{ct}_0 = (\hat{b}_0, \hat{a}_0)$ and $\mathsf{ct}_1 = (\hat{b}_1, \hat{a}_1)$ that respectively encrypts $\hat{m}_0, \hat{m}_1 \in R_p$ under a same key, the operation $\mathsf{ct}_0 \boxplus \mathsf{ct}_1$ computes the RLWE tuple $(\hat{b}_0 + \hat{b}_1, \hat{a}_0 + \hat{a}_1) \in R^2_{N,q}$ which can be decrypted to $\hat{m}_0 + \hat{m}_1 \bmod R_{N,p}$.
- **Multiplication** ($\boxtimes$)**.** Given an RLWE ciphertext $\mathsf{ct} = (\hat{b}, \hat{a})$ that encrypts $\hat{m} \in R_{N,p}$, and a plain polynomial $\hat{c} \in R_{N,p}$, the operation $\mathsf{ct} \boxtimes \hat{c}$ computes the tuple $(\hat{b} \cdot \hat{c}, \hat{a} \cdot \hat{c}) \in R^2_{N,q}$ which can be decrypted to $\hat{m} \cdot \hat{c} \bmod R_{N,p}$.

```
Known Sign-bit (Positive) to Wrap-bit $\Pi_{\texttt{wrap}^+}$
Inputs: $[\![x]\!]^M$ for a general modulus $M$ such that $\mathsf{sgn}(x) = 0$.
Outputs: $[\![\mathsf{wrap}\left([\![x]\!]^M\right)]\!]^{2^k}$ over the ring modulus $\mathbb{Z}_{2^k}$.
1: $P_0$ and $P_1$ invokes an instance of $\Delta$-$\texttt{OT}_k$ where $P_0$ is the sender inputting the correlation $\Delta = 1 - \mathsf{sgn}([\![x]\!]_0^M)$. $P_1$ is the receiver with the choice bit $c = 1 - \mathsf{sgn}([\![x]\!]_1^M)$.
2: After $\Delta$-$\texttt{OT}_k$, $P_0$ learns $m_0 \in \mathbb{Z}_{2^k}$ and $P_1$ gets $m_c \in \mathbb{Z}_{2^k}$.
3: $P_0$ outputs $m_0$, and $P_1$ outputs $1 - m_c \bmod 2^k$.
```

Fig. 3: Known Sign-bit to Wrap-bit Protocol $\Pi_{\texttt{wrap}^+}$.

- **SIMD Encoding.** The SIMD encoding $\mathsf{SIMD} : \mathbb{F}_p^N \mapsto R_{N,p}$ maps a vector over $\mathbb{F}_p$ to a polynomial over $R_{N,p}$ when $p = 1 \bmod 2N$ [36]. This encoding enables entry-wise additions and multiplications as follows.

$$\mathsf{SIMD}^{-1}(\mathsf{SIMD}(\mathbf{u}) + \mathsf{SIMD}(\mathbf{v}) \in R_{N,p}) = \mathbf{u} + \mathbf{v} \in \mathbb{F}_p^N,$$
$$\mathsf{SIMD}^{-1}(\mathsf{SIMD}(\mathbf{u}) \cdot \mathsf{SIMD}(\mathbf{v}) \in R_{N,p}) = \mathbf{u} \odot \mathbf{v} \in \mathbb{F}_p^N.$$

In the context of HE, this encoding can amortize the cost of N-lane homomorphic multiplication by $1/N$.

III. Efficient Modulus Conversions

For a shared value x over the modulus M, the wrap-bit indicates whether the sum of the two shares wraps around the underlying representation. Specifically, we define it as:

$$\mathsf{wrap}\left([\![x]\!]^M\right) = \mathbf{1}\{[\![x]\!]_0^M + [\![x]\!]_1^M \ge M\}.$$

The wrap-bit function is essential in many two-party computation protocols, including modulus conversions and fixed-point truncation. The wrap-bit can be computed using a private comparison protocol. Formally, this is expressed as: $\mathsf{wrap}([\![x]\!]^M) = \mathbf{1}\{[\![x]\!]_0^M > M - 1 - [\![x]\!]_1^M\}$.

We can reduce the overhead of computing the wrap-bit if the sign of the input is already known. For example,

$$\mathsf{wrap}([\![x]\!]^M) = \mathsf{sgn}([\![x]\!]_0^M) \vee \mathsf{sgn}([\![x]\!]_1^M) \text{ when } \mathsf{sgn}(x) = 0.$$

This means the wrap-bit of a positive value equals the logical OR of the two sign bits from each share. We now show the correctness. By definition, we have $[\![x]\!]_0^M + [\![x]\!]_1^M = x + \mathsf{wrap}([\![x]\!]^M) \cdot M$. For the case $\mathsf{wrap}([\![x]\!]^M) = 1$, $[\![x]\!]_0^M + [\![x]\!]_1^M \ge M$ indicating at least one of the share should be greater-or-equal than $\lceil M/2 \rceil$. On the other hand, we have $[\![x]\!]_0^M + [\![x]\!]_1^M = x < \lceil M/2 \rceil$ when $\mathsf{wrap}([\![x]\!]^M) = 0$. This means both shares are less than $\lceil M/2 \rceil$.

The logical OR of two private bits can be computed using Δ-$\texttt{OT}_k$. We present our wrap-bit protocol $\Pi_{\texttt{wrap}^+}$ in Figure 3 which assumes positive values as input. Note that $m_c = m_0 + (1 - \mathsf{sgn}([\![x]\!]_0^M)) \cdot (1 - \mathsf{sgn}([\![x]\!]_1^M)) \bmod 2^k$ by the definition of Δ-$\texttt{OT}_k$. Thus, $m_0 + 1 - m_c = 1 - (1 - \mathsf{sgn}([\![x]\!]_0^M)) \cdot (1 - \mathsf{sgn}([\![x]\!]_1^M)) \bmod 2^k$ which is the the logical OR of the two sign-bits. Our logical OR protocol is efficient than the ones described in [4]–[6]. Particularly, they use $\binom{2}{1}$-$\texttt{OT}_1$ to obtain logical OR as a boolean share, and then convert it to an arithmetic share using one Δ-$\texttt{OT}_k$.

```
Modulus Conversion $\Pi_{\texttt{mconv}}$.
Inputs: $[\![x]\!]^p$ over a ring $\mathbb{Z}_p$ such that $x \in [0, L) \cup [p - L, p)$ for some positive $L \le \lceil p/4 \rceil$.
Outputs: $[\![y]\!]^{2^k}$ such that $y = \mathsf{rep}_p(x) \bmod 2^k$.
1: $[\![x']\!]^p = [\![x]\!]^p + L \bmod p$  ▷ $\mathsf{sgn}_p(x') = 0$.
2: $[\![w]\!]^{2^k} \leftarrow \Pi_{\texttt{wrap}^+}([\![x']\!]^p)$. ▷ wrap-bit in the modulus $2^k$.
3: $P_0$ outputs $[\![y]\!]_0^{2^k} = [\![x']\!]_0^p - p \cdot [\![w]\!]_0^{2^k} \bmod 2^k$.
4: $P_1$ outputs $[\![y]\!]_1^{2^k} = [\![x']\!]_1^p - p \cdot [\![w]\!]_1^{2^k} - L \bmod 2^k$.
```

Fig. 4: From a prime Modulus p to Modulus 2^k.

A. Heuristic Wrap-bit Computation

When the sign-bit of $x \in \mathbb{Z}_M$ is unknown, we can employ a heuristic approach by adding a large value, such as $L = \lceil M/4 \rceil$, and assume that the sign-bit $\mathsf{sgn}(x + L \bmod M)$ is zero. This heuristic holds true as long as x lies within the range $[0, L) \cup [M - L, M)$. This heuristic is generally acceptable for applications like privacy-preserving machine learning, with only a minimal loss of 1-bit capacity in the underlying representation. We first determine the wrap-bit of $x + L \bmod M$ using the efficient $\Pi_{\texttt{wrap}^+}$ protocol (Figure 3), then adjust for the additional L offset. Details on this adjustment are provided in the subsequent section.

Dalskov et al. [24] also utilizes a similar approach to accelerate truncation over $\mathbb{Z}_{2^\ell}$-shares in a three-party setting. This heuristic can also be applied in broader functions, including $\mathbb{Z}_p$-to-$\mathbb{Z}_{2^\ell}$ modulus conversion.

B. Heuristic Modulus Conversion

We consider a general conversion between two types of additive shares of **signed numbers**. Specifically, given a share $[\![x]\!]^M$ over a modulus M, our goal is to convert it to a share $[\![z]\!]^K$ over a different modulus K such that $z = \mathsf{rep}_M(x) \bmod K$. Recall that $\mathsf{rep}_M(x)$ is the signed representative of x over the modulus M. Our secure modulus conversion is based on the following equation:

$$[\![z]\!]^K = [\![x + L]\!]^M - L - M \cdot [\![\mathsf{wrap}([\![x + L]\!]^M)]\!]^K \bmod K \quad (1)$$

where L is some large value $L \le \lceil M/4 \rceil$. Again the sign-bit of $x + L$ is zero as long as $x \in [0, L) \cup [M - L, M)$. By this heuristic, we can compute the wrap-bit $[\![\mathsf{wrap}([\![x + L]\!]^M)]\!]^K$ using the efficient $\Pi_{\texttt{wrap}^+}$ protocol[3].

Basically (1) results in $z = (x + L \bmod M) - L \bmod K$. If $x \in [0, L)$ we have $x + L \bmod M = x + L$. Thus $z = x \bmod K$ in this case. On the other hand, when $x \in [M - L, M)$ we have $x + L \bmod M = x + L - M$, indicating $z = x - M \bmod K$ in this case. Then we can conclude that $z = \mathsf{rep}_M(x) \bmod K$. We can consider 4 conversions between different moduli.

- $M = p$ **and** $K = 2^k$**:** Figure 4 presents the modulus conversion from a prime modulus to a two-power modulus.
- $M = 2^\ell$ **and** $K = 2^k$ **for** $\ell < k$**:** When converting from a smaller two-power modulus to a larger two-power modulus, we can use the $\Pi_{\texttt{wrap}^+}$ protocol for computing the wrap-bit but using a smaller COT parameter. Because in this case, we

[3]The Δ-$\texttt{OT}$ is also well defined for a prime modulus. For example the Δ-$\texttt{OT}$ [37] can generate correlation over a prime modulus.

have $2^\ell \cdot [\![w]\!]_b^{2^k} \bmod 2^k = 2^\ell \cdot ([\![w]\!]_b^{2^k} \bmod 2^{k-\ell}) \bmod 2^k$ for any $w \in \mathbb{Z}_{2^k}$. Thus, we can first use Δ-$\texttt{OT}_{k-\ell}$ to obtain the share of the wrap-bit over modulus $2^{k-\ell}$. This reduces communication from $O(k)$ bits to $O(k-\ell)$ bits per share.

- $M = 2^\ell$ **and** $K = 2^k$ **for** $\ell \geq k$**:** When converting from a larger two-power modulus to a smaller two-power modulus, the computation of the wrap-bit can be even skipped, rendering a local procedure for this conversion. That is $[\![z]\!]_b^{2^k} = [\![x]\!]_b^{2^\ell} \bmod 2^k$ for $b = 0, 1$.
- $M = 2^\ell$ **and** $K = p$**:** We aim to compute $[\![z]\!]^p \leftarrow [\![x]\!]^{2^\ell}$ with $z = \mathsf{rep}_{2^\ell}(x) \bmod p$. We need to compute a wrap-bit $[\![\mathsf{wrap}([\![x]\!]^{2^\ell})]\!]^p$, which might transfer $O(\log_2 p)$ bits using Δ-$\texttt{OT}$. When p is large enough, e.g., $p > 2^{40}$, we present an optimization for this conversion. Specifically, we first extend $[\![x]\!]^{2^\ell}$ to $[\![x']\!]^{2^k}$ over a larger ring $k \geq \ell$ such that $x' = \mathsf{rep}_{2^\ell}(x) \bmod 2^k$, and then set $[\![z]\!]_0^p = [\![x']\!]_0^{2^k} \bmod p$ and $[\![z]\!]_1^p = [\![x']\!]_1^{2^k} - 2^k \bmod p$ **locally**. This results in $z = [\![z]\!]_0^p + [\![z]\!]_1^p = x' + w \cdot 2^k - 2^k \bmod p$ for the wrap-bit $w = \mathsf{wrap}([\![x']\!]^{2^k})$. We will show that $z = \mathsf{rep}_{2^\ell}(x) \bmod p$, **except** with a probability $|\mathsf{rep}_{2^\ell}(x)|/2^k$. By setting $k \geq \ell + 40$, this probability becomes negligibly small. Our insight is the communication costs of $\mathbb{Z}_{2^\ell}$-to-$\mathbb{Z}_{2^k}$ modulus conversion is about $O(k-\ell)$ bits per share which can be smaller than using Δ-$\texttt{OT}$ over modulus p, particularly $\log_2 p \geq 40$.
  There are two cases that we obtain $z = \mathsf{rep}_{2^\ell}(x) \bmod p$:
  1) $w = 1 \wedge x' = \mathsf{rep}_{2^\ell}(x)$. It means that the shares $[\![x']\!]^{2^k}$ wrap around 2^k, and value $\mathsf{rep}_{2^\ell}(x) \geq 0$ is positive.
  2) $w = 0 \wedge x' = 2^k + \mathsf{rep}_{2^\ell}(x)$. It means that shares $[\![x']\!]^{2^k}$ do not wrap around 2^k, and $\mathsf{rep}_{2^\ell}(x) < 0$ is negative.

  Recall the probability of a wrap-around $\Pr(w = 1) = 1 - (x'+1)/2^k$. For the 1st case $\Pr(w = 1) = 1 - (\mathsf{rep}_{2^\ell}(x) + 1)/2^k$. For the 2nd case, $\Pr(w = 0) = (x'+1)/2^k = 1 + (\mathsf{rep}_{2^\ell}(x)+1)/2^k = 1 - (|\mathsf{rep}_{2^\ell}(x)| - 1)/2^k$.

IV. Benchmarks & Optimizations

In this section, we review several existing approaches for two fundamental primitives: private comparisons and share multiplications. We benchmark these approaches in terms of total communication and computation time. Additionally, we identify the best approach based on minimal expenditure cost. We also propose optimizations for the runners-up.

A. Private Comparison

The private comparison (CMP) function in Figure 5 takes two unsigned ℓ-bit integers x and y from P_0 and P_1, respectively, and compute Boolean share $\langle \mathbf{1}\{x < y\}\rangle$. CMP can be implemented using different cryptographic primitives, including OT, HE, Garbled Circuit (GC), and Distributed Comparison Function (DCF). We prefer to leverage OT-based approach from [4] whose cost is smallest among the other approaches [15], [38]–[40], particularly for large inputs $\ell \geq 32$. We will summarize their performance differences later.

1) CMP Protocol from OT: We demonstrate how [4] implements the $\mathcal{F}_{\texttt{CMP}}^\ell$ functionality. Specifically, [4] first breaks the ℓ-bit value into $D = \lceil \ell/m \rceil$ blocks of m-bit each. That is $x = \sum_{i\in[D]} x_i \cdot 2^{i\cdot m}$ and $y = \sum_{i\in[D]} y_i \cdot 2^{i\cdot m}$ for

Private Comparison $\mathcal{F}_{\texttt{CMP}}^\ell$
Upon receiving $x \in \mathbb{Z}_{2^\ell}$ from P_0 and $y \in \mathbb{Z}_{2^\ell}$ from P_1, this functionality operates as follows: 1) Sample $b \xleftarrow{\$} \mathbb{B}$ and compute $c = \mathbf{1}\{x < y\} \oplus b$. 2) Send b to P_0 and c to P_1.

Fig. 5: CMP Functionality.

$x_i, y_i \in [0, 2^m)$. For simplicity, we assume D is a two-power value here. Then, [4] leverages D instances of $\binom{2^m}{1}$-$\texttt{OT}_2$ to compute equalities $\langle \texttt{eq}_{1,i} = \mathbf{1}\{x_i = y_i\}\rangle$ and less-than indicators $\langle \texttt{lt}_{1,i} = \mathbf{1}\{x_i < y_i\}\rangle$ for $i \in [D]$. After that, they update the less-than and equalities in a bottom-up manner:

$$\begin{aligned} \texttt{lt}_{j+1,i} &= \texttt{eq}_{j,2i+1} \wedge \texttt{lt}_{j,2i} \oplus \texttt{lt}_{j,2i+1} \\ \texttt{eq}_{j+1,i} &= \texttt{eq}_{j,2i+1} \wedge \texttt{eq}_{j,2i}, \end{aligned} \tag{2}$$

for $j = 1, 2, \cdots, \log_2 D - 1$. The $\texttt{lt}_{\log_2 D, 0}$ on the highest layer is the comparison result, i.e., $\texttt{lt}_{\log_2 D,0} = \mathbf{1}\{x < y\}$.

MSB and Secret Share Comparison. The CMP protocol can be used to compute the most-significant-bit (MSB) of an arithmetic share over $\mathbb{Z}_{2^\ell}$. We parse the share $[\![x]\!]_b^{2^\ell} = m_b \cdot 2^{\ell-1} + z_b$ where m_b is the highest bit, and $z_b \in [0, 2^{\ell-1})$. Then, we evaluate the secure MSB as $\langle \mathrm{MSB}(x)\rangle_b = m_b \oplus \langle \mathbf{1}\{z_0 > 2^{\ell-1} - 1 - z_1\}\rangle_b$ for $b = 0, 1$. The share of the bit $\mathbf{1}\{z_0 > 2^{\ell-1} - 1 - z_1\}$ is computed by calling $\mathcal{F}_{\texttt{CMP}}^{\ell-1}$ over inputs of $\ell - 1$ bits. The secure comparison of two unsigned values over the 2^ℓ modulus, $[\![x]\!]^{2^\ell}$ and $[\![y]\!]^{2^\ell}$, is evaluated as $\langle \mathbf{1}\{x < y\}\rangle = \langle \mathrm{MSB}(x - y \bmod 2^\ell)\rangle$, which requires one $\mathcal{F}_{\texttt{CMP}}$. In contrast, secure comparison over a prime modulus may require multiple $\mathcal{F}_{\texttt{CMP}}$ operations, as described by [4].

The protocol [4] is based on IKNP OT [34]. We now present two possible optimizations that can be achieved by switching from IKNP OT to the more efficient Ferret OT [22].

2) Correlated AND-triples Optimization: Rathee et al. [4] evaluate the two logical ANDs of (2) using a correlated AND-triple $(\langle x\rangle, \langle y\rangle, \langle y'\rangle, \langle z\rangle, \langle z'\rangle)$ such that $z = x \wedge y$ and $z' = x \wedge y'$. Particularly, they generate one correlated AND-triple using $\binom{8}{1}$-$\texttt{OT}_2$ by exchanging about 130 bits if using the IKNP OT or 17 bits if using the Ferret OT. To compare, we can generate one correlated AND-triple by exchanging less than 3 bits from Ferret OT as follows:

1) Run two instances of $\texttt{ROT}_2$. P_0 receives $(u_0, u_1) \in \mathbb{Z}_4 \times \mathbb{Z}_4$ and P_1 receives $(u_{c_1}, c_1) \in \mathbb{Z}_4 \times \mathbb{B}$ from the first instance. In the second instance, P_1 receives $(v_0, v_1) \in \mathbb{Z}_4 \times \mathbb{Z}_4$, and P_0 receives $(v_{c_0}, c_0) \in \mathbb{Z}_4 \times \mathbb{B}$.
2) Set $x_0 = c_0$, $y_0 = u_0 \oplus u_1$, and $z_0 = x_0 \wedge y_0 \oplus (u_0 \oplus v_{c_0})$. Set $x_1 = c_1$, $y_1 = v_0 \oplus v_1$, and $z_1 = x_1 \wedge y_1 \oplus (v_0 \oplus u_{c_1})$.
3) Parse the messages as Boolean shares. That is $x_b = \langle x\rangle_b$, $y_b = \langle y\rangle_b \| \langle y'\rangle_b$, and $z_b = \langle z\rangle_b \| \langle z'\rangle_b$.

Two instances of Ferret's $\texttt{ROT}_2$ (on average) exchanges less than 2 bits. Our correlated AND-triple generation can be viewed as an extension of [26, Algorithm 1]. We defer the correctness to Appendix B due to the space limit.

Complexity. For the ℓ-bits operands, the comparison protocol [4] requires $D = \lceil \ell/m \rceil$ instances of $\binom{2^m}{1}$-$\texttt{OT}_2$, $\lceil \log_2 D\rceil$ ANDs and $D - 1 - \lceil \log_2 D\rceil$ correlated ANDs. In total, it will exchange about $D \cdot (2^{m+1} + m) + 5 \cdot \lceil \log_2 D\rceil + 8 \cdot (D - 1 - \lceil \log_2 D\rceil)$ bits. For example, with $\ell = 32, m = 4$, this yeilds

TABLE III: Costs of $n = 2^{20}$ pair-comparisons on $\ell = 32$ messages. Time is measured by one CPU with AES-NI, under LAN (c.f. §VI). Numbers with '*' are estimated from corresponding paper. Smallest marked in bold.

Method	Time	Commu.	A2I Cost	A2A Cost
GC [41]	$< \mathbf{7}$s	1.5GB	13.520¢	3.020¢
DGK [39]†	$\approx$ 1.3h*	537GB*	$\approx 4,800$¢	$\approx 1,000$¢
DCF [40]‡	$>$ 400h	66.56MB	$\approx 4,105$¢	$\approx 4,106$¢
IKNP OT [15]	0.96h	20.96GB	51.77¢	198.47¢
IKNP OT [42]	$\approx$ 195s*	$\approx$ 2.5GB*	$\approx$ 23¢	$\approx$ 5.556¢
IKNP OT [4]	8.27s	505MB	4.462¢	1.010¢
Ferret OT [4]	10.69s	47.5MB	0.448¢	0.123¢
Ours	7.15s	**42.1**MB	**0.392**¢	**0.104**¢

† Used one V100 GPU which is 306¢ per hour, and bitwidth $\ell = 20$.
‡ We measure the costs of $n = 2^{16}$ pair-comparisons for 2PC DCF.

a theoretical communication cost of 335 bits per comparison when using the Ferret OT. Note that the communication cost of one correlated AND is 6 masked bits plus the cost of generating the correlated AND-triple.

3) Comparing the Costs of Different CMP Protocols: Table III summarizes the state of current approaches, including the GC implementation from the EMP-toolkit [41], the OT-based approach from [4], the DGK-based approach from [39], and the DCF-based approach from [30], [40].

The GC-based CMP protocol is computationally efficient but has high communication overhead, requiring $O(3\lambda\ell)$ bits of communication per ℓ-bit comparison. To mitigate the online communication of the GC-based CMP protocol, many works, such as [15], [39], [42], propose alternative protocols that shift some computation to the precomputation phase. These methods successfully reduce the online communication of the GC-based CMP protocol by more than 80%. However, they significantly increase precomputation communication.

Replacing IKNP OT with Ferret OT reduces the expenditure costs of [4] by about 95%. Moreover, our correlated AND optimization further reduces this cost by about 11%. With our optimizations, the OT-based CMP [4] offers the lowest expenditure cost among existing approaches. Interestingly, the expenditure cost of the GC-based method would become the lowest if the communication price drops below 1.25×10^{-5} US dollars per GB, which is 0.07% of the current price.

A 2PC CMP can be obtained by combining techniques from [30] and [40]. [30] compares two private inputs using DCF and leverages a trusted third party to generate and distribute the DCF keys to P_0 and P_1. [40] proposes an efficient two-party protocol for DCF key generation, it indicates that generating the DCF key **for one comparison** in 2PC requires $O(2^{\ell+1})$ local AES, resulting in a significantly high computation cost. For example, it might take more than 400 hours for 2^{16} pair-comparisons, demonstrating significantly slower performance compared to other end-to-end 2PC solutions.

B. Secret Share Multiplication

To multiply two shares $[\![xy]\!]^M \leftarrow \mathcal{F}_{\mathtt{mul}}([\![x]\!]^M, [\![y]\!]^M)$, we can leverage Oblivious Linear Evaluation (OLE) [43]. Particularly, we use a batch of OLEs to handle the multiplication of many inputs. In Figure 6, the OLEs we use work as a two-party function that takes a private vector $\mathbf{a} \in \mathbb{Z}_M^n$ from P_0 and another private vector $\mathbf{b} \in \mathbb{Z}_M^n$ from P_1, and generates the share $[\![\mathbf{a} \odot \mathbf{b}]\!]^M$ between them. For the multiplication of two secret shares, e.g., $[\![\mathbf{x}]\!]^M, [\![\mathbf{y}]\!]^M$, we utilize two OLEs:

Oblivious Linear Evaluation $\mathcal{F}_{\mathtt{OLE}}^{M,n}$
Upon receiving $\mathbf{a} \in \mathbb{Z}_M^n$ from P_0 and $\mathbf{b} \in \mathbb{Z}_M^n$ from P_1, this functionality operates as follows: 1) Sample $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_M^n$, and send $\mathbf{u}$ to P_0. 2) Send $\mathbf{a} \odot \mathbf{b} - \mathbf{u} \bmod M$ to P_1.

Fig. 6: OLE for vector Hadamard product.

$$\begin{aligned}&([\![\mathbf{x}]\!]_0^M + [\![\mathbf{x}]\!]_1^M) \odot ([\![\mathbf{y}]\!]_0^M + [\![\mathbf{y}]\!]_1^M)\\ =&\underbrace{[\![\mathbf{x}]\!]_0^M \odot [\![\mathbf{y}]\!]_0^M + [\![\mathbf{x}]\!]_1^M \odot [\![\mathbf{y}]\!]_1^M}_{\text{local}} + \underbrace{[\![\mathbf{x}]\!]_0^M \odot [\![\mathbf{y}]\!]_1^M + [\![\mathbf{x}]\!]_1^M \odot [\![\mathbf{y}]\!]_0^M}_{2\times\text{OLEs}}.\end{aligned}$$

We consider the $\mathbb{Z}_p$-OLE for a prime p which can be more efficiently constructed than the $\mathbb{Z}_{2^\ell}$-OLE counterpart.

1) $\mathbb{Z}_p$-OLE Constructions from RLWE: The classic construction of $\mathcal{F}_{\mathtt{OLE}}^{p,n}$ for a prime modulus p using RLWE is basically follows the ones in [27], [44] (c.f. Appendix C-A). We propose a strategy to reduce communication cost when using RLWE-based OLE for secret share multiplications.

In this strategy, P_1 sends two ciphertexts, corresponding to $[\![\mathbf{x}]\!]_1^p$ and $[\![\mathbf{y}]\!]_1^p$, to P_0. P_0 now only needs to respond with a single ciphertext of $[\![\mathbf{x}]\!]_0^p \odot [\![\mathbf{y}]\!]_1^p + [\![\mathbf{x}]\!]_1^p \odot [\![\mathbf{y}]\!]_0^p$ to P_1 for batched secret share multiplications. However, this strategy may leave P_1 idle while P_0 performs HE computation, leading to wasted CPU resources on P_1's side. To address this, when the input length is large, we switch the role of the sender. Specifically, P_0 can process the first half of inputs, while P_1 processes the remaining. This strategy effectively leverages CPU resources on both sides, reducing idle time and improving overall efficiency (detailed in § VI-A).

2) $\mathbb{Z}_p$-OLE Construction from PCG: Boyle et al. [45] present an alternative method for constructing $\mathbb{Z}_p$-OLE using a primitive called Pseudorandom Correlation Generator (PCG). While we do not delve into the construction details provided in [45], we highlight two notable properties of the PCG-based construction. First, the (minimum) batch size of the PCG-based approaches is significantly larger than that of RLWE-based ones. For instance, they [45] need to generate a batch of $n \geq 2^{21}$ OLEs at once. This renders a relatively long computation time for one exeuction. Second, the communication complexity of the PCG-based OLE is sublinear with the input length. However, the input vectors, i.e., $\mathbf{a}$ and $\mathbf{b}$ in Figure 6, are determined by the OLE protocol itself rather than by the parties. Consequently, we **can not** directly use PCG-based OLE to multiply two secret shares but need to use it to generate Beaver's triples first. Thus, we still need to exchange 4 ring elements per multiplication using PCG-based OLE.

3) Costs of Different OLEs: Table IV presents the concrete costs for performing $n = 2^{21}$ secret share multiplications using different OLE constructions. We highlight four key observations: **1).** Currently, RLWE-based approaches offer the lowest cost among the methods due to a better balance between CPU time and communication overhead. **2).** The amortized cost of secret share multiplication over the 2^ℓ modulus is over four times as expensive as that of multiplication over the prime modulus. **3).** PCG-based approaches may benefit from

TABLE IV: The costs of $n = 2^{21}$ share multiplications using different OLE constructions with parameters $p \approx 2^{60}$ and $\ell = 64$. Time is measured by one CPU core under LAN (c.f. §VI). Numbers with'*' are taken from corresponding paper.

Method	Mod.	Time	Commu.	A2I Cost	A2A Cost
RLWE [46]	p	4.91s	314.6MB	2.779¢	0.628¢
PCG [40], [45]	p	500.0s	**60.0**MB	1.953¢	1.543¢
OT [43]	2^ℓ	55.1s	6.0GB	54.157¢	12.078¢
OT [43]	2^8	5.45s	544MB	4.797¢	1.078¢
RLWE [12]	2^ℓ	12.9s	896.0MB	7.859¢	1.775¢
DGK [14]	2^ℓ	1.18h*	1.5GB*	25.612¢	15.112¢
Ours	p	**1.52**s	155.6MB	**1.373**¢	**0.310**¢

lower communication overhead. However, they also increase computation time, and thus also the computation costs, by two orders of magnitude compared to RLWE-based methods. **4).** RLWE-based approaches is still more cost-effective than OT-based approaches, even for a small modulus. For instance, the $\mathbb{Z}_{2^8}$-OLE can be realized by a $\mathbb{Z}_p$-OLE by setting $p \geq 2^{57}$.

Note that if the computation cost drops below 0.666¢ per CPU core per hour (i.e., $\approx 13\%$ of the current price) the PCG-based approaches become more cost-effective than the RLWE-based ones under the A2A cost metric. Nevertheless, PCG-based methods still require significantly more execution time, making them more suitable for use in the precomputation phase rather than just-in-time evaluation.

V. Secure Machine Learning Building Blocks

Machine learning algorithms commonly operate on real numbers. However, in secure computation, real numbers are typically represented as fixed-point values, following the approach used in many previous works. Particularly, a real value $\tilde{x} \in \mathbb{R}$ is encoded as a fixed-point value $x = \lfloor \tilde{x} \cdot 2^{\ell_f} \rfloor \bmod M$ under a fixed-point precision $\ell_f > 0$. $[\![x; \ell_f]\!]^M$ denotes the share of the fixed-point number with ℓ_f-bit precision.

We demonstrate how to optimize several useful machine learning building blocks using the proposed techniques, including fixed-point truncation, non-uniform multiplication, constant-round exponentiation, and general approximations using piecewise polynomials for non-linear functions. The pioneering works [5], [10], [47], [48] introduced a set of protocols for one or several of the above blocks, but our protocols are more efficient due to our optimizations.

A. Approximated Truncation

We perform the secure fixed-point multiplication in two steps. First, we compute the multiplication of two shared fixed-point numbers to get a fixed-point number of double precision. Then we apply a truncation step to reduce the fixed-point precision. We accelerate the secure fixed-point truncation over a modulus M using the wrap-bit heuristic.

Theorem 1 (Approximated Truncation Over a General M)**.** *Suppose M is the modulus and let $\eta = \lfloor \log_2 M \rfloor - 2$. For the shared value $[\![x]\!]^M$ that $x \in [0, 2^\eta) \cup [M - 2^\eta, M)$. The following steps result in a share $[\![z]\!]^M$ such that $z = \mathsf{trc}_M(x; \ell_f) - E \bmod M$ for a bounded error $E \in \{0, \pm 1\}$:*

1) Compute $[\![y = x + 2^\eta]\!]^M$ so that $\mathsf{sgn}(y) = 0$.

2) Compute the wrap-bit $[\![w]\!]^M \leftarrow \mathcal{F}_{\texttt{wrap+}}([\![y]\!]^M)$.

3) Set $[\![z]\!]_0^M = \lfloor \frac{[\![y]\!]_0^M}{2^{l_f}} \rfloor - \lfloor \frac{M}{2^{l_f}} \rfloor \cdot [\![w]\!]_0^M \bmod M$.

4) Set $[\![z]\!]_1^M = \lfloor \frac{[\![y]\!]_1^M}{2^{l_f}} \rfloor - 2^{\eta - l_f} - \lfloor \frac{M}{2^{l_f}} \rfloor \cdot [\![w]\!]_1^M \bmod M$.

Proof. Recall that $y = x + 2^\eta \bmod M$. When $x \in [0, 2^\eta)$, we have $\lfloor y/2^{l_f} \rfloor = \lfloor x/2^{l_f} \rfloor + 2^{\eta - l_f} = \mathsf{trc}_M(x; \ell_f) + 2^{\eta - l_f} \bmod M$. On the other hand when $x \in [M - 2^\eta, M)$, we have $y = x + 2^\eta - M$. In this case $\lfloor y/2^{l_f} \rfloor = \lfloor (x + 2^\eta - M)/2^{l_f} \rfloor = \lfloor (x - M)/2^{l_f} \rfloor + 2^{\eta - l_f} \bmod M$ which also equals to $\mathsf{trc}_M(x; \ell_f) + 2^{\eta - l_f} \bmod M$. In other words, $\lfloor y/2^{l_f} \rfloor - 2^{\eta - l_f} = \mathsf{trc}_M(x; \ell_f) \bmod M$.

Let $r_0 = [\![y]\!]_0^M \bmod 2^{l_f}$, $r_1 = [\![y]\!]_1^M \bmod 2^{l_f}$, and $r_2 = M \bmod 2^{l_f}$. Also, we write $w = \mathsf{wrap}\left([\![y]\!]^M\right)$. We compute

$$
\begin{aligned}
\lfloor y/2^{l_f} \rfloor &= \lfloor ([\![y]\!]_0^M + [\![y]\!]_1^M - w \cdot M)/2^{l_f} \rfloor \\
&= \lfloor [\![y]\!]_0^M / 2^{l_f} \rfloor + \lfloor [\![y]\!]_1^M / 2^{l_f} \rfloor - w \cdot \lfloor M/2^{l_f} \rfloor + E \\
&= z + 2^{\eta - l_f} + E \bmod M,
\end{aligned}
$$

where the 1-bit error $E \in \{0, \pm 1\}$. Particularly, when the wrap-bit $w = 0$, we have $\mathbf{1}\{r_0 + r_1 \geq 2^{l_f}\} = 1 \Leftrightarrow E = 1$. On the other hand when $w = 1$, we have $\mathbf{1}\{r_0 + r_1 - r_2 < 0\} = 1 \Leftrightarrow E = -1$ and $\mathbf{1}\{r_0 + r_1 - r_2 \geq 2^{l_f}\} = 1 \Leftrightarrow E = 1$. Otherwise, $E = 0$. □

Compared to the Current 2PC Truncations. The local truncation procedure [2] simply sets the wrap-bit $w = 1$, leading to a probabilistic error since $\Pr[w \neq 1] = \frac{y+1}{M}$. Note that this error could be as large as $O(M/2^{\ell_f})$. The error-free truncation protocol [4] computes the wrap-bit w and the value E using multiple instances of $\mathcal{F}_{\texttt{CMP}}$. Huang et al. [6] proposed that ignoring the last-bit error E can make the truncation protocol far more lightweight, but it still requires computing wrap-bit w using the $\mathcal{F}_{\texttt{CMP}}$ functionality. To compare, we compute the wrap-bit w with a significantly smaller overhead than the $\mathcal{F}_{\texttt{CMP}}$ at the cost of losing 1-bit capacity of the representation modulus. Also, we introduce a 1-bit error by setting $E = 0$. In concurrent GeoMPC [10], Guo et al. proposed a similar 1-bit approximated truncation, but it invokes a communication of $O(\lambda + \ell)$ bits using IKNP OT, while we only require $O(\ell)$ bits by using Ferret OT.

1) Modulus Conversion with Free Truncation: In some applications, such as neural networks, nonlinear operations (e.g., activation functions) immediately follow linear operations (e.g., multiplication). For such cases, we propose a specialized secure fixed-point multiplication that takes secret shares as input from a modulus p but directly produces a truncated result in modulus 2^ℓ. Instead of sequentially executing the $\mathbb{Z}_p$-to-$\mathbb{Z}_{2^\ell}$ share conversion protocol followed by a truncation protocol over 2^ℓ, we propose an optimized approach for this special multiplication. Our insight is that wrap-bit computed during the $\mathbb{Z}_p$-to-$\mathbb{Z}_{2^\ell}$ conversion can also be leveraged for truncation over the $\mathbb{Z}_{2^\ell}$ shares. This leads to a $\mathbb{Z}_p$-to-$\mathbb{Z}_{2^\ell}$ share conversion protocol with free truncation in Figure 7. The correctness basically follows Theorem 1 except here we compute wrap-bit as a share over destination modulus 2^ℓ.

```
Inputs: $[\![x]\!]^p$ such that $x \in [0, 2^\eta) \cup [p - 2^\eta, p)$ for $2^\eta < \lceil p/4 \rceil$.
Outputs: $[\![z]\!]^{2^\ell}$ with $\mathsf{rep}_{2^\ell}(z) = \lfloor \frac{\mathsf{rep}_p(x)}{2^{l_f}} \rfloor + E$ for $E \in \{0, 1\}$.
1: $[\![y]\!]^p \leftarrow [\![x]\!]^p + 2^\eta \bmod p \triangleright \mathsf{sgn}_p(y) = 0$.
2: $[\![w]\!]^{2^\ell} \leftarrow \Pi_{\mathtt{wrap}^+}([\![y]\!]^p) \triangleright$ wrap-bit in the modulus $2^\ell$.
3: $P_0$: $[\![z]\!]_0^{2^\ell} = \lfloor \frac{[\![y]\!]_0^p}{2^{l_f}} \rfloor - \lfloor \frac{p}{2^{l_f}} \rfloor \cdot [\![w]\!]_0^{2^\ell} \bmod 2^\ell$.
4: $P_1$: $[\![z]\!]_1^{2^\ell} = \lfloor \frac{[\![y]\!]_1^p}{2^{l_f}} \rfloor - 2^{\eta - l_f} - \lfloor \frac{p}{2^{l_f}} \rfloor \cdot [\![w]\!]_1^{2^\ell} \bmod 2^\ell$.
```

Fig. 7: Modulus Conversion with Free Truncation.

```
Hyper-parameters: The evaluation range $T > 0$, e.g., $T = 16$.
Inputs: A share of a fixed-point number $[\![x; f]\!]^{2^\ell}$ where the underlying message $\tilde{x} = \mathsf{rep}_{2^\ell}(x)/2^{\ell_f}$ ranges over $\tilde{x} \in [-T, 0)$.
Outpts: $[\![u; \ell_f]\!]^{2^\ell}$ such that $\mathsf{rep}_{2^\ell}(u)/2^{\ell_f} \approx 2^{\tilde{x}}$.
1: Adding an offset $[\![x' = x + z; \ell_f]\!]^{2^\ell}$ where $z = (1 + T) \cdot 2^{\ell_f}$
   so that the underlying message $\mathsf{rep}_{2^\ell}(x')/2^{\ell_f} \geq 1$.
2: $P_b$ locally seperates the integral and fractional part as:
   $t_b = \lfloor \frac{[\![x'; \ell_f]\!]_b^{2^\ell}}{2^{\ell_f}} \rfloor \in \mathbb{Z}_{2^{\ell - \ell_f}}$ and
   $\tilde{m}_b = \frac{[\![x'; \ell_f]\!]_b^{2^\ell} \bmod 2^{\ell_f}}{2^{\ell_f}} \in \mathbb{R}$. That is
   $(t_0 + t_1 \bmod 2^{\ell - \ell_f}) + \tilde{m}_0 + \tilde{m}_1 = \tilde{x} + 1 + T + r$ for some
   roundings error $|r| < 2^{-(\ell_f + 1)}$.
3: Convert the modulus of $t_0, t_1 \in \mathbb{Z}_{2^{\ell - \ell_f}}$ using the $\Pi_{\mathtt{mconv}}$
   protocol (Figure 4), and obtain $h_0, h_1 \in \mathbb{Z}_{p-1}$ for a prime
   number $p$. That is
   $h_0 + h_1 \bmod (p - 1) = t_0 + t_1 \bmod 2^{\ell - \ell_f}$.
4: $P_b$ sends $u_b = 2^{h_b} \cdot \lfloor 2^{\tilde{m}_b + \ell_f} \rfloor \bmod p$ to $\mathcal{F}_{\mathtt{OLE}}^{p,1}$. Then $P_b$
   obtains $[\![u; 2\ell_f + 1 + T]\!]_b^p$ from $\mathcal{F}_{\mathtt{OLE}}^{p,1} \triangleright u = u_0 \cdot u_1 \bmod p$.
5: Invoke a $\mathbb{Z}_p$-to-$\mathbb{Z}_{2^\ell}$ conversion (Figure 7) to bring back the
   share $[\![u; 2\ell_f + 1 + T]\!]^p$ to $[\![u; \ell_f]\!]^{2^\ell}$ as final output.
```

Fig. 8: Base-2 Exponentiation for Negative Inputs.

B. Secure Non-Uniform Multiplication

The secure multiplication with non-uniform bitwidths requires that the multiplication result is shared over a larger ring than the input shares [5], [10]. Specifically, they perform multiplication as $[\![xy]\!]^{2^{m+k}} \leftarrow [\![x]\!]^{2^m} \cdot [\![y]\!]^{2^k}$. SiRNN [5] utilizes IKNP OT for this share multiplication, involve exchanging $O(\lambda(3\mu+\tau+4)+\mu(\mu+2\tau+1)+16(m+k))$ bits, and recent work GeoMPC [10] leverages the 1-bit heuristic constraint to reduce the communication to $O(\lambda(2\mu + 14) + \mu(\mu + 3) + 2mk+6(m+k))$, where $\mu = \min(m, k)$ and $\tau = \max(m, k)$. In contrast, as illustrated in Figure 2, M&M works as:

- First, we lift the modulus of $[\![x]\!]^{2^m}$ and $[\![y]\!]^{2^k}$ to a prime modulus such that $p \approx 2^{m+k}$.
- We then perform multiplication using our $\mathbb{Z}_p$-OLE protocol and obtain $[\![xy]\!]^p$.
- Through a final $\mathbb{Z}_p$-to-$\mathbb{Z}_{2^{m+k}}$ conversion, we bring back the modulus to $[\![xy]\!]^{2^{m+k}}$ (along with the free truncation if necessary).

Our total communication cost is $O(13(m+k))$ bits, achieving *one magnitude* improvements over [5], [10].

C. Constant-Round Exponentiation

In Figure 8, we present an efficient two-party protocol for computing the base-2 exponentiation $2^{\tilde{x}}$ from a secret share of a fixed-point number $\lfloor \tilde{x} \cdot 2^{\ell_f} \rfloor$. We are particularly interested in negative inputs that are close to the origin, e.g., $\tilde{x} \in [-12, 0]$, which is useful for functions such as softmax. Our protocol requires 6 rounds of communication, and is optimized based on [49]. The key insight of Figure 8 is Step 4, where we compute the arithmetic multiplication $2^{h_0} \cdot 2^{h_1} \bmod p$ even though h_0 and h_1 lie within a smaller modulus $p - 1$. By definition, $h_0 + h_1 = h + w \cdot (p - 1)$ for some message $h \in \mathbb{Z}_{p-1}$ and a wrap-bit $w \in \{0, 1\}$. Therefore, we have $2^{h_0} \cdot 2^{h_1} = 2^{h + w \cdot (p-1)} = 2^h \bmod p$ by Fermat's Little Theorem. The correctness is analyzed in Appendix A.

We now highlight the differences we have made compared to the original protocol of [49]. Kelkar et al. [49] perform all modulus conversions (i.e., Step 3 and Step 5) and the truncation (i.e., Step 5) using local procedures. For instance, they use the local truncation procedure [2]. The advantage is that their exponentiation protocol requires only two rounds of communication for the OLE in Step 4[4]. The disadvantage is that they have to set a relatively large ring size to achieve a negligible failure rate due to the local procedures. For instance, they need to set $\log_2(p) \approx \ell > 87$ for base-2 exponentiation over the range $\tilde{x} \in [-16, 0)$ with the fixed-point precision $\ell_f = 15$. In contrast, our base-2 exponentiation protocol does not fail with any probability since we leverage interactive but highly efficient protocols to achieve exact share conversions in Step 3 and 5. Therefore, we can use a significantly smaller parameter p for the OLE, e.g., $\log_2 p \approx 48$. This greatly reduces the costs of OLE.

Softmax. Exponentiation $\exp(\tilde{x})$ on negative inputs $\tilde{x} \leq 0$ is particularly useful for the Softmax activation function. This exponentiation can be rewritten as scaled base-2 exponentiation: $\exp(\tilde{x}) = 2^{\log_2(e) \cdot \tilde{x}}$, which can be evaluated securely using the protocol described in Figure 8. However, the multiplication with the real number $\log_2(e)$ requires a secure fixed-point multiplication that includes an extra truncation step. Indeed, this truncation can be saved since Step 1 to Step 3 of Figure 8 also work properly for inputs of a double fixed-point precision.

D. Approximations via Piecewise Polynomials

We denote by $g_{d,m} : \mathbb{R} \to \mathbb{R}$ a piecewise function defined over $d + 1$ intervals, each represented by a polynomial of degree m, as shown in (3) below.

$$g_{d,m}(x) = \begin{cases} F_0(x), & \text{if } x \leq I_1, \\ F_1(x), & \text{if } x \in (I_1, I_2), \\ \cdots \\ F_d(x), & \text{if } x > I_d. \end{cases} \quad (3)$$

Each polynomial is defined $F_i(x) = a_{i,0} + \sum_{j=1}^{m} a_{i,j} x^j$.

Many existing works, such as [3], [21], [23], [51], have utilized piecewise functions for approximated secure computation

[4]Indeed they evaluate the expression $u = u_0 \cdot u_1 \bmod p$ in a single round using a multiplicative-to-additive share conversion protocol [50], which requires two OLEs in the precomputation phase.

TABLE V: Benchmarking our modulus conversions with a batch size of $n = 10^7$ (single thread). The run time includes the setup time for the Ferret OT. Specifically, we convert shares from a modulus 2^ℓ to a different modulus 2^k.

$k-\ell$	Commu. (MB)		Time (sec)		
	$P_0 \to P_1$	$P_0 \leftarrow P_1$	LAN	MAN	WAN
16	1.192	19.607	1.405	1.824	3.117
32	1.192	38.681	1.499	2.126	4.739
40	1.192	48.217	1.579	2.256	5.537
64	1.192	76.828	1.820	2.657	7.982

on fixed-point numbers. For example, [21], [23] approximate the GeLU function using a piecewise function with parameters $(d = 2, m = 4)$. However, few existing approaches have proposed optimizations to accelerate the secure evaluation of (3), particularly in the 2PC setting. We present our secure evaluation of (3) in Appendix D. We postpone the truncation in Step 4 until after we have summed the results in Step 5. This saves us d secure truncations and modulus conversions.

The secure evaluation of (3) is also referred to as Oblivious Piecewise Polynomial Evaluation (OPPE). The most significant difference between our OPPE and existing algorithms, such as those in [3], [21], [23], [51], is that we leverage mixed sharing moduli during the evaluation to minimize the overheads associated with both the linear and non-linear computations. In contrast, existing OPPE consistently uses a single sharing modulus throughout their evaluation. For example, [23], [51] consistently use a power-of-two modulus.

1) Indicator Batching: The d instances of comparisons of OPPE are done between one secret value and multiple plain thresholds. We can optimize these secret share comparisons using the one-vs-many optimization as follows: Specifically, on the i-th instance of $\mathcal{F}^{\ell-1}_{\mathtt{CMP}}$, P_0 inputs $[\![I_i - x]\!]^{2^\ell}_0 \bmod 2^{\ell-1}$. We can set P_0's shares $[\![I_1 - x]\!]^{2^\ell}_0 = [\![I_2 - x]\!]^{2^\ell}_0 = \cdots = [\![I_d - x]\!]^{2^\ell}_0 = 2^\ell - [\![x]\!]^{2^\ell}_0$ due to the asymmetric property of the substraction between a secert share and a plaintext. In this way, the problem is reduced to comparing one private value from P_0 with a batch of private values from P_1, which can be optimized with *one-to-many comparisons* by 30%–60% of the computation time (Appendix D-A).

VI. Implementations and Evaluations

Code Base. We base our implementations over the Secure Processing Unit Framework (SPU) [52] which has already implemented and integrated many cryptographic building blocks such as the Ferret OT [22] with the half-tree optimization [40], the lattice-based HE from the SEAL libraray [53] with the HEXL AVX512 acceleration [54]. All these programs are written in C++. More specifically, it realizes the $\binom{2^m}{1}$-$\mathtt{OT}$ from m instances of $\binom{2}{1}$-$\mathtt{OT}$ following the method provided by [43].
Testbed Environments. The experiments are majorly conducted on two servers equipped with 64 vCPUs and 256 GB of RAM, CPU is AMD EPYC 9754 (2.25GHz), and Operating System is Ubuntu 20.04.6 LTS with Linux kernel 5.4.0-144-generic. We employ the same ring sizes $\mathbb{Z}_{2^\ell}$ and fractional precisions ℓ_f as established in [6], [23], and the prime modulus is set $p \approx 2^\ell$. We evaluate implementations under three network conditions: **LAN:** 1 Gbps bandwidth with 2 ms latency. **MAN:** 1 Gbps bandwidth with 40 ms latency. **WAN:** 100 Mbps bandwidth with 40 ms latency.

TABLE VI: Benchmarking our private comparison protocol with the parameter $m = 4$, and a batch size of $n = 2^{20}$ (single thread). The run times include Ferret's setup time.

ℓ	Commu. (MB)		Time (sec)		
	$P_0 \to P_1$	$P_0 \leftarrow P_1$	LAN	MAN	WAN
32	6.504	35.520	7.145	7.472	7.974
64	13.633	71.410	15.226	15.574	16.564
128	27.762	143.311	33.429	34.108	36.332

TABLE VII: Benchmarking the RLWE-based $\mathbb{Z}_p$-OLE (single thread) with a batch size of $n = 2^{20}$. The first column shows the bit-width of each RLWE parameter.

$\log_2(N, q, p)$	Commu. (MB)		Time (sec)		
	$P_0 \to P_1$	$P_0 \leftarrow P_1$	LAN	MAN	WAN
$(12, 109, 16)$	15.21	8.20	0.649	0.685	2.375
$(13, 135, 32)$	19.69	12.45	0.942	0.978	3.298
$(13, 173, 60)$	23.38	31.05	0.992	1.027	4.956

A. Microbenchmarks

Modulus Conversions. The benchmarking results in Table V demonstrate that our modulus conversion protocols are highly efficient, processing 10^6 shares per second even under MAN.
Private Comparisons The results presented in Table VI demonstrate that our private comparison protocol is practical, even with its higher round complexity compared to existing methods. The protocol maintains reasonable communication costs and computation times across various network environments, showing that the increased round complexity does not result in prohibitive performance penalties. These findings confirm that the OT-based protocol remains a viable and cost-effective option for secure computation.
Prime OLE. The benchmarking results for our RLWE-based $\mathbb{Z}_p$-OLE implementation in Table VII show that the volume of response ciphertexts from P_1 is generally smaller due to the use of the common modulus switching technique [55]. This technique effectively reduces the size of the ciphertexts by switching to a smaller modulus during the homomorphic operations. However, for larger plaintext sizes, such as $p \approx 2^{60}$, a significantly larger noise must be used to maintain the circuit privacy [56]. This larger noise limits the effectiveness of the modulus switching technique.

B. Benchmarks of Building Blocks

Approximated Truncation. The experimental results in Table VIII highlight the efficiency of our secure fixed-point truncation techniques. CrypTFlow2 and our method can work over both ring $\mathbb{Z}_{2^\ell}$ and prime modulus, and Cheetah is specialized for $\mathbb{Z}_{2^\ell}$. Compared to CrypTFlow2 [4] and Cheetah [6], our two approaches achieve significant reductions in both communication overhead and runtime, i.e., by upto two magnitudes. Notably, our truncation implementation over the ring $\mathbb{Z}_{2^\ell}$ outperforms the prime modulus variant. This is mainly due to the wrap-bit computation being more efficient in the ring than prime modulus. Compared to recent [10], our truncation

TABLE VIII: Benchmarking truncation with a batch of $n = 10^7$ (single thread). CrypTFlow2 and GeoMPC utilize IKNP OT, Cheetah and our truncations employ Ferret OT. The run times include IKNP/Ferret's setup time.

Protocol	Commu.		Time (sec)		
	$P_0 \rightarrow P_1$	$P_0 \leftarrow P_1$	LAN	MAN	WAN
CrypTFlow2 (p) [4]	17.883 GB	3.744 GB	530.368	707.251	2101.040
CrypTFlow2 (2^ℓ) [4]	9.047 GB	1.062 GB	219.882	308.356	970.639
Cheetah (2^ℓ) [6]	739.966 MB	153.347 MB	177.448	212.967	782.022
GeoMPC (2^ℓ) [10]	152.750 MB	21.458 MB	3.828	10.278	21.044
Ours (p)	76.548 MB	1.192 MB	2.885	11.592	16.930
Ours (2^ℓ)	28.864 MB	1.192 MB	2.621	11.038	13.059

TABLE IX: Benchmarking the secret share multiplication with non-uniform bitwidths. We set parameters $m = 18$ and $k = 20$ bits, and the output is of $m + k$ bits. The communication and time are for 2^{16} runs of protocols. The run times include IKNP/Ferret's setup time.

Protocol	Commu. (MB)		Time (sec)		
	$P_0 \rightarrow P_1$	$P_0 \leftarrow P_1$	LAN	MAN	WAN
SIRNN [5]	29.000	70.000	5.531	8.044	14.784
GeoMPC [10]	27.000	31.000	3.187	4.385	8.056
Ours	2.462	2.345	2.186	2.810	3.044

over $\mathbb{Z}_{2^\ell}$ reduces the communication by approximately $6\times$ and achieves better running time in most cases.

Secure Multiplication with Non-Uniform Bitwidths. We set $m = 18$ and $k = 20$ for multiplication with non-uniform bitwidths, and the fractional bit is set $\ell_f = 12$. As shown in Table IX, compared to SIRNN and GeoMPC, our method reduces the communication cost by $16\times$ and $12\times$, respectively. Our approach is also faster than the other approaches; in particular, it improves runtime by over $4\times$ and $2\times$ in MAN.

Exponentiation & Softmax. Lu et al. [23] approximates the exponentiation using the Taylor expansion, i.e., $\exp(x) \approx (1 + x/32)^{32}$ for $x \in (-14, 0]$. Experimental results in Table X show that our constant-round approximation method reduces the communication by around $10\times$ and runtime by $7\times$ over [23]. The efficiency improvements in exponentiation further lead to more efficient Softmax. Concretely, we reduce the communication and runtime by about $3\times$.

Approximated GeLU. We benchmark our piecewise polynomial approximation method using function GeLU, which is the most widely used activation function in Transformer models. Following work [23], given a ring share $[\![x; \ell_f]\!]^{2^\ell}$ of $\ell = 32$ (ring size) and $\ell_f = 12$ (fixed-point precision), we evaluate function $[\![\mathsf{GeLU}(x); \ell_f]\!]^{2^\ell}$ using approximation:

$$\mathsf{GeLU}(x) \approx \begin{cases} x - \epsilon & x \geq 3 \\ 0.5x + P^4(|x|) & x \in [-3, 3) \\ \epsilon & x < -3 \end{cases} . \quad (4)$$

Here $P^4(\cdot)$ is a degree-4 polynomial. The computation consists of two parts: 1) branch selection which needs performing share comparisons and 2) polynomial evaluation which consists of share multiplications. For the branch selection, we compute a batch of three comparisons, including $b_0 = \mathbf{1}\{x < -3\}$, $b_1 = \mathbf{1}\{x < 0\}$ and $b_2 = \mathbf{1}\{x < 3\}$. Specifically, $b_0 \oplus b_2 = \mathbf{1}\{x \in [-3, 3)\}$ and $1 \oplus b_2 = \mathbf{1}\{x \geq 3\}$. We can reuse the bit b_1 to obtain the absolute value $|x| = b_1 \cdot (-2x) + x$. The experimental results are in Table X: i) Lu et. al. [23] perform

TABLE X: Benchmarking secure non-linear functions performance (single thread), batchsize is 2^{20}. For Exponentiation and Softmax, $\ell = 64$ and $\ell_f = 18$. For GeLU, $\ell = 32$ and $\ell_f = 12$. For Softmax, the input shape is $2^{10} \times 2^{10}$.

Functions		Commu. (MB)		Time (sec)		
		$P_0 \rightarrow P_1$	$P_0 \leftarrow P_1$	LAN	MAN	WAN
Exp	BumbleBee [23]	391.167	319.906	22.245	35.828	63.795
	Ours	32.274	31.805	3.991	5.687	10.113
Softmax	BumbleBee [23]	555.624	425.585	25.705	46.582	80.944
	Ours	198.759	137.450	7.431	16.482	27.285
GeLU	BumbleBee [23]	257.306	155.052	21.778	34.927	76.977
	Ours	173.677	104.657	20.811	33.423	57.331

TABLE XI: End-to-end performance improvements over private CNN inference frameworks and LLM inference frameworks. We used 4 cores and 64 cores for respective CNN and LLM inference in the LAN setting.

System		Commu.	Time	A2I Costs
ResNet50	CrypTFlow2 [4]	32.43GB	545.35s	391.38¢
	Cheetah [6]	2.44GB	149.6s	23.66¢
	GeoMPC [10]	360.28GB	428.7s	725.44¢
	Ours	1.85GB	112.4s	17.93¢
DenseNet121	CrypTFlow2 [4]	35.56GB	463.2s	404.56¢
	Cheetah [6]	2.60GB	149.6s	25.11¢
	GeoMPC [10]	206.40GB	291.6s	416.13¢
	Ours	2.01GB	131.3s	15.59¢
BERT-base	IRON [57]	76.5GB	2041.1s	1060.04¢
	BOLT [21]	63.6GB	676.3s	695.81¢
	NEXUS [58] (CPU)	0.2GB	13985.2s	2553.69¢
	BLB [59] (CPU)	3.0GB	201.6s	63.79¢
	BumbleBee [23]	6.40GB	153.0s	85.52¢
	Ours	4.45GB	128.4s	63.47¢
ViT-base	BumbleBee [23]	11.56GB	239.4s	147.72¢
	Ours	7.72GB	192.1s	104.53¢

both the branch selection, and the polynomial evaluation over the ring $\mathbb{Z}_{2^{32}}$. Empirically, they can perform 2^{20} GeLUs within 21.8 seconds using 1 CPU core, and exchange about 412.9 MB messages over the LAN. Specifically, the branch selection part is about 13.9 seconds and 99 MB. ii) We perform the branch selection as [23] but we perform the polynomial evaluation over a 32-bit prime using our conversion protocols of ring and prime shares. Empirically, we can perform 2^{20} GeLUs within 20.8 seconds using 1 CPU core, and exchange about 278.7 MB messages over the LAN. More experimental results under WAN and MAN can be seen in Table X.

C. Performance of 2PC Machine Learning

We achieve comparable or identical model performance to cleartext and state-of-the-art 2PC baselines across deep neural networks and boosting decision trees (c.f., Appendix E). This validates the practicality of our positive heuristic.

Private Inference of Deep Neural Networks. Following CrypTFlow2 [10], Huang et al. [6] and Lu et al. [23] present secure two-party inference frameworks for convolutional neural networks (CNN), and large language models (LLM), respectively. In particular, they both implement the $\mathbb{Z}_{2^\ell}$-OLE functionality using RLWE.

We propose four optimizations for [6], [23] as follows: **1)** Implement secure truncation using the wrap-bit heuristic. This approach eliminates the need for the relatively expensive $\mathcal{F}_{\mathtt{CMP}}$ functionality in the fixed-point truncation operation. Notably, the $\mathcal{F}_{\texttt{CMP}}$-based truncation accounts for more than 25% of the computation time in [6]. **2)** Replace the $\mathbb{Z}_{2^\ell}$-OLE with a $\mathbb{Z}_p$-OLE where $p \approx 2^\ell$. This can reduce the costs of Batch Normalizations and Layer Normalizations in deep learning inference by about 60%. By utilizing our $\mathbb{Z}_p$-to-$\mathbb{Z}_{2^\ell}$ modulus conversion, this $\mathbb{Z}_p$-OLE can be seamlessly integrated into the other components of [6] and [23] that are defined over the 2^ℓ modulus. **3)** Leverage mixed sharing moduli during the evaluation of piecewise polynomials approximated activation functions to minimize the overheads associated with linear and non-linear computations. **4)** Especially, we replace Taylor approximation with our exponentiation protocol in § V-C to reduce the number of OLEs from 32 to 1 OLE per input point.

Although [10] proposed a truncation method comparable to our approach, we achieve $> 100\times$ reduction in communication, $2 \sim 3\times$ speedup, and more than $20\times$ improvement in monetary cost. This is primarily because [10] relies exclusively on IKNP OT for both linear and nonlinear computations, whereas we adopt a hybrid approach and utilize more efficient Ferret OT. It is worth noting that our experiments are conducted in LAN, which inherently favors [10]. Even so, both our method and [6] outperform it by combining HE and OT, validating our claims in § I. Several follow-up works to [6], such as [60], [61], have introduced various optimizations by *shifting certain computations to a precomputation phase*. However, the overall costs of these approaches remain higher than those of the original private CNN protocol presented in [6]. For example, Balla et al. [61] report that ResNet50 inference takes 183 seconds and requires more than 17GB of communication. And we focus on optimizing total costs in Table XI.

In terms of private LLM inference, IRON [57] extended Cheetah to support secure Transformer operators, and framework BOLT [21] employs IKNP OT for share multiplication with non-uniform bitwidths, following the approach in [5]. In contrast, we adopt our hybrid method that combines modulus conversion and $\mathbb{Z}_p$-OLE, as §V-B. When setting $\ell = k = 16$, this results in approximately 95% reduction in communication cost compared to the IKNP-based approach used by [21]. Framework NEXUS [58] utilized fully HE scheme CKKS to achieve secure non-interactive inference, but incurs substantially higher runtime overhead. The concurrent framework BLB [59] remodels Transformer architectures and integrates CKKS with 2PC for improved efficiency. However, its remodeling technique is non-trivial to generalize to models beyond Transformers

The empirical results in Table XI demonstrate our significant performance improvements in private deep neural networks inference over most existing frameworks. By implementing secure truncation using a wrap-bit heuristic and replacing $\mathbb{Z}_{2^\ell}$-OLE with $\mathbb{Z}_p$-OLE, we achieved notable reductions in both computation time and communication costs. These enhancements lead to over 25% – 99% savings in overall expenditure, making our optimized framework more cost-effective for PPML tasks. The only exceptions are the NEXUS and BLB frameworks, which incur lower communication costs but require considerably longer execution time. Nevertheless, our approach achieves comparable or even superior performance in terms of both runtime and monetary efficiency.

TABLE XII: End-to-end performance improvements over Squirrel [28]. The performance is for one tree ($T = 1$) with 31 tree nodes ($D = 5$) using 8 CPU cores in LAN setting.

Problem Size		Commu.	Time	A2I Costs
10^5 Samples, $(T = 1, D = 5)$	Squirrel [28]	310MB	12.1s	3.00¢
	Ours	174MB	11.0s	1.79¢
10^6 Samples, $(T = 1, D = 5)$	Squirrel [28]	2600MB	94.0s	25.00¢
	Ours	1248MB	84.4s	12.89¢

Private Training of Boosting Trees. Lu et al. [28] present a 2PC framework Squirrel for training a boosting decision tree. They utilize $\mathbb{Z}_{2^\ell}$-OLE with $\ell = 64$ for both the approximation of sigmoid function using Fourier series and the computation of the second-order gradient, i.e., $[\![\mathbf{g}-\mathbf{g}\odot\mathbf{g}; \ell_f]\!]^{2^\ell} \leftarrow [\![\mathbf{g}; \ell_f]\!]^{2^\ell}$. Our optimizations are as follows: i) we optimize Sigmoid approximations by piecewise polynomials (c.f. Appendix D-B) with our techniques §V-D, and ii) the computation of the second-order gradient can also benefit from replacing $\mathbb{Z}_{2^\ell}$-OLE with $\mathbb{Z}_p$-OLE. The empirical results in Table XII demonstrate that our optimizations effectively reduce the cost of the secure decision tree training [28] by half.

VII. Related Work

Wrap-bit Computations & Modulus Conversion. Kikuchi et al. [17] compute the wrap-bit for secret sharing over an odd modulus p under the assumption that the underlying message is even. Specifically, the wrap-bit is calculated as $\texttt{wrap}([\![x]\!]^p) = ([\![x]\!]^p_0 \bmod 2) \oplus ([\![x]\!]^p_1 \bmod 2)$ for an even $x \in [0, p)$. For any input y with the parity unknown, one can first compute the wrap-bit over $2y \bmod p$, assuming $y \in [0, \lceil p/2 \rceil)$. However, this approach does not work for $y' \in [p - \lceil p/2 \rceil, p)$ because the parity of $2y' \bmod p$ is odd. This limitation is why the method in [17] is commonly used for "positive" only, as seen in [18], [62]. [17] requires exchanging $O(\lambda + 2\log_2 p)$ bits per share since they compute the logical XOR via multiplication.

Mixed Modulus Secure Computation. The most closely related framework is the SecureNN framework [13], which leverages two sharing moduli: a two-power modulus 2^ℓ and an odd modulus $2^\ell - 1$. They present conversion protocol for these two moduli. Notably, the SecureNN framework operates within a special secure two-party computation setting where an additional trusted third party assists in the computation throughout the entire process. Even with the help from the third party, their conversion protocol still need to exchange $O(34\ell)$ bits message per share. In contrast, our modulus conversion protocols can operate with a broader range of moduli and require less communication, exchanging only $O(\ell)$ bits per share. ABY [14], [15] and (extended) daBits [7], [16] are specifically designed for efficient conversions between arithmetic and boolean circuits. However, extending these approaches to handle conversions between different arithmetic moduli efficiently is not straightforward. For instance, directly integrating a Boolean circuit between arithmetic circuits with different moduli would necessitate the execution of costly binary addition circuits, which are of $O(\ell \log_2 \ell)$ communication and $O(\log_2 \ell)$ round complexity. Moreover, they are designed following the precomputation/online paradigm, which requires an expensive input-independent precomputation phase.

In the private LLM application [21], they also mix a prime and a two-power moduli using a local conversion procedure as the one described in [49, Appendix A].

Secure Truncation Protocols. Recent work by [25] presents an efficient, secure, and probabilistic truncation protocol for Shamir's secret sharing. Their protocol requires the sharing modulus to be a Mersenne prime, e.g., $p = 2^{60} - 1$. Unlike our approach, they do not apply a positive heuristic; however, similar to our method, they also require the underlying message to be at least 1-bit smaller than the sharing modulus. Moreover, their truncation protocol introduces a larger error range up to ± 2. Interestingly, this error gap may be due to a subtle difference in the truncation definition for negative inputs. Specifically, we define truncation for negative inputs $x \in \mathbb{Z}_M$ as $\lfloor \frac{x-M}{2^d} \rfloor \bmod M$, where the flooring operation first handles negative numbers, and it will round to $-\infty$. In contrast, they define it as $M - \lfloor \frac{M-x}{2^d} \rfloor$, where the flooring is applied to positive numbers only, and thus rounding towards 0. Guo et al. [10] propose similar techniques for 2PC that also require $|x|$ to be two bits smaller than the sharing modulus. However, we achieve significantly better efficiency, particularly in communication complexity. [10] is restricted to the two-power modulus and does not support prime modulus.

Constant-Round Secure Exponentiation. In addition to the protocol proposed by [49], Lu et al. [18] also introduce a constant-round secure two-party protocol, specifically for integral exponentiation over a prime secret sharing modulus. Specifically, they compute $[\![a^x]\!]^p \leftarrow [\![x]\!]^p$, for a public base $a \in \mathbb{Z}_p$ via a private selection on $a^{[\![x]\!]_0^p} \cdot a^{[\![x]\!]_1^p} \bmod p$ or $a^{[\![x]\!]_0^p} \cdot a^{[\![x]\!]_1^p} \cdot a^{-p} \bmod p$ according to the bit $\mathsf{wrap}([\![x]\!]^p)$. They require a prime sharing modulus because they leverage the method of [17] to compute the wrap-bit. It is important to note that integral exponentiation serves different purposes compared to the fixed-point exponentiation.

VIII. Discussion

Security Arguments. We capture the security of Π_{mconv} and approximated truncation as Theorem 2.

Theorem 2. *The protocols Π_{mconv} (Fig. 3) and the approximated truncation (§ V-A) securely realize modulus conversion and approximated truncation, respectively, in the presence of a static probabilistic polynomial-time semi-honest adversary under the Δ-$\mathtt{OT}$ hybrid model.*

The proposed heuristic modulus conversion builds upon the well-known sign-bit (wrap-bit) computation protocol Π_{wrap^+}, which itself relies on the well-established 2PC primitive Δ-$\mathtt{OT}$. Except for the interactive operations inherent to 2PC, all other computations in these protocols are performed locally. Therefore, the security of the positive heuristic protocols follows directly from the security guarantees of the underlying Δ-$\mathtt{OT}$ primitive. The remaining components (e.g., probabilistic approximations) are composed of standard 2PC protocols combined with our heuristic modulus conversion. Consequently, their security is preserved in the Δ-$\mathtt{OT}$ hybrid model.

Generally, the 2PC protocols work on k-bit field/ring can also work on m-bit field/ring by our protocols, where $k \neq m$.

2PC in Precomputation Model. In this model, 2PC is divided into two phases: an input-independent precomputation phase and an online evaluation phase, as [2], [9], [14], [15], [30], [39]. The expensive asymmetric cryptographic protocols can be executed only during the precomputation phase. This paradigm can decrease the latency time for the evaluation phase. Besides the latency time, we highlight three important considerations that should not be overlooked when developing applications using secure two-party computation techniques.

1). Input-independent precomputations are typically distinct from a one-time setup such as keys generation. For example, the precomputation may be proportional to the circuit size, and the precomputed materials such as Beaver's multiplication triples [63], can only be used once.

2). *The total costs* of secure computation maybe increased when applying the precomputation model. For instance, secure multiplication can be computed with a small online cost via Beaver's triples. These triples are commonly generated using HE. However, we can use HE to perform secure multiplications at the same cost as triples generation. While the online costs of using triples are low, their total costs are slightly higher than those of using HE directly.

3). In practice, the cost of network transfer can be more expensive than computing resources. For instance, Amazon's Elastic Cloud charges about 0.09 US dollars ($) per outbound gigabyte, while the rental price for one CPU is just 0.054$ per hour. For some 2PC applications, such as Ad tech [62], [64] and private-aggregation system [65], [66], the expenditure cost is also an important factor to consider.

It is essential to consider the total costs holistically by balancing both computation and communication expenses. Thoughtful consideration should be given to both the precomputation phase and the evaluation phase.

IX. Conclusion

We propose a new modulus conversion protocol that facilitates the seamless integration of the most effective two-party techniques across prime and two-power moduli. Building on this, we introduced the M&M framework, a highly efficient solution for 2PC ML that leverages mixed-mode protocols and secret sharing moduli. M&M achieved significant performance improvements in privacy-preserving ML applications, setting a new benchmark in the field. For future work, we plan to design modulus conversion protocols with malicious security to defend against active adversaries, and to investigate the error probability and bias magnitude under heuristic bounds in tight bit-width settings.

References

[1] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 351–371.

[2] P. Mohassel and Y. Zhang, "SecureML: A System for Scalable Privacy-Preserving Machine Learning," in *IEEE S&P*, 2017, pp. 19–38.

[3] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious Neural Network Predictions via MiniONN Transformations," in *CCS*, 2017, pp. 619–631.

[4] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-Party Secure Inference," in *CCS*, 2020, pp. 325–342.

[5] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "SiRnn: A Math Library for Secure RNN Inference," in *IEEE S&P*, 2021, pp. 1003–1020.

[6] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference," in *USENIX Security*, 2022, pp. 809–826.

[7] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits," in *CRYPTO*, 2020, pp. 823–852.

[8] A. C. Yao, "How to Generate and Exchange Secrets," in *FOCS*, 1986, pp. 162–167.

[9] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A Cryptographic Inference Service for Neural Networks," in *USENIX Security*, 2020, pp. 2505–2522.

[10] H. Guo, L. Peng, H. Xue, L. Peng, W. Liu, Z. Liu, and L. Hu, "Improved secure two-party computation from a geometric perspective," in *USENIX Security*, 2025.

[11] H. Wu, W. Fang, Y. Zheng, J. Ma, J. Tan, and L. Wang, "Ditto: Quantization-aware Secure Inference of Transformers upon MPC," in *ICML*, 2024, pp. 53 346–53 365.

[12] D. Rathee, T. Schneider, and K. K. Shukla, "Improved Multiplication Triple Generation over Rings via RLWE-Based AHE," in *CANS*, 2019, pp. 347–359.

[13] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-Party Secure Computation for Neural Network Training," *Proc. Priv. Enhancing Technol.*, no. 3, pp. 26–49, 2019.

[14] D. Demmler, T. Schneider, and M. Zohner, "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation," in *NDSS*, 2015.

[15] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation," in *USENIX Security*, 2021, pp. 2165–2182.

[16] D. Rotaru and T. Wood, "Marbled circuits: Mixing arithmetic and boolean circuits with active security," in *International Conference on Cryptology in India*. Springer, 2019, pp. 227–249.

[17] R. Kikuchi, D. Ikarashi, T. Matsuda, K. Hamada, and K. Chida, "Efficient Bit-Decomposition and Modulus-Conversion Protocols with an Honest Majority," in *ACISP*, 2018, pp. 64–82.

[18] Y. Lu, K. Hara, K. Ohara, J. C. N. Schuldt, and K. Tanaka, "Efficient Two-Party Exponentiation from Quotient Transfer," in *ACNS*, vol. 13269, 2022, pp. 643–662.

[19] L. K. Ng and S. S. Chow, "Sok: Cryptographic neural-network computation," in *IEEE S&P*. IEEE, 2023, pp. 497–514.

[20] W. Zeng, T. Xu, Y. Chen, Y. Zhou, M. Zhang, J. Tan, C. Hong, and M. Li, "Towards efficient privacy-preserving machine learning: A systematic review from protocol, model, and system perspectives," 2025. [Online]. Available: https://arxiv.org/abs/2507.14519

[21] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers," *In IEEE S&P*, p. 1893, 2023.

[22] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast Extension for Correlated OT with Small Communication," in *CCS*, 2020, pp. 1607–1626.

[23] W. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, K. Ren, C. Hong, T. Wei, and W. Chen, "BumbleBee: Secure Two-party Inference Framework for Large Transformers," in *NDSS*, 2025.

[24] A. P. K. Dalskov, D. Escudero, and M. Keller, "Secure Evaluation of Quantized Neural Networks," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 4, pp. 355–375, 2020.

[25] F. Liu, X. Xie, and Y. Yu, "Scalable Multi-Party Computation Protocols for Machine Learning in Honest-Majority Setting," in *USENIX Security*, 2024.

[26] G. Asharov, Y. Lindell, T. Schneider, M. Zohner, and M. Yung, "More Efficient Oblivious Transfer and Extensions for Faster Secure Computation," in *CCS*, 2013, pp. 535–548.

[27] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ Great Again," in *EUROCRYPT*, vol. 10822, 2018, pp. 158–189.

[28] W. Lu, Z. Huang, Q. Zhang, Y. Wang, and C. Hong, "Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree," in *USENIX Security*, 2023, pp. 6435–6451.

[29] 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, pp. 707–721.

[30] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation," in *EUROCRYPT*, 2021, pp. 871–900.

[31] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "SIGMA: secure GPT inference with function secret sharing," *Proc. Priv. Enhancing Technol.*, no. 4, pp. 61–79, 2024.

[32] K. Gupta, D. Kumaraswamy, N. Chandran, and D. Gupta, "LLAMA: A Low Latency Math Library for Secure Inference," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 4, pp. 274–294, 2022.

[33] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[34] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending Oblivious Transfers Efficiently," in *CRYPTO*, 2003, pp. 145–161.

[35] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *EUROCRYPT*, vol. 6110, 2010, pp. 1–23.

[36] N. P. Smart and F. Vercauteren, "Fully Homomorphic SIMD Operations," *IACR Cryptol. ePrint Arch.*, p. 133, 2011.

[37] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits," in *IEEE S&P*, 2021, pp. 1074–1091.

[38] O. Catrina and S. de Hoogh, "Improved Primitives for Secure Multiparty Integer Computation," in *SCN*, 2010, pp. 182–199.

[39] L. Ng and S. S. M. Chow, "GForce: GPU-Friendly Oblivious and Rapid Neural Network Inference," in *USENIX Security*, 2021, pp. 2147–2164.

[40] X. Guo, K. Yang, X. Wang, W. Zhang, X. Xie, J. Zhang, and Z. Liu, "Half-Tree: Halving the Cost of Tree Expansion in COT and DPF," in *EUROCRYPT*, 2023, pp. 330–362.

[41] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2023.

[42] T. Lu, X. Kang, B. Zhang, Z. Ma, X. Zhang, Y. Liu, and K. Ren, "Efficient 2PC for Constant Round Secure Equality Testing and Comparison," in *USENIX Security*, 2025.

[43] M. Naor and B. Pinkas, "Oblivious Transfer and Polynomial Evaluation," in *STOC*, 1999, pp. 245–254.

[44] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," in *CRYPTO*, 2012, pp. 643–662.

[45] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient Pseudorandom Correlation Generators from Ring-LPN," in *CRYPTO*, 2020, pp. 387–416.

[46] C. Baum, D. Escudero, A. Pedrouzo-Ulloa, P. Scholl, and J. R. Troncoso-Pastoriza, "Efficient Protocols for Oblivious Linear Function Evaluation from Ring-LWE," in *SCN*, 2020, pp. 130–149.

[47] O. Catrina and A. Saxena, "Secure Computation with Fixed-Point Numbers," in *FC*, 2010, pp. 35–50.

[48] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient comparison for secure multi-party computation," in *FC*. Springer, 2021, pp. 249–270.

[49] M. Kelkar, P. H. Le, M. Raykova, and K. Seth, "Secure Poisson Regression," in *USENIX Security*, 2022, pp. 791–808.

[50] H. Ghodosi, J. Pieprzyk, and R. Steinfeld, "Multi-party computation with conversion of secret sharing," *Des. Codes Cryptogr.*, vol. 62, no. 3, pp. 259–272, 2012.

[51] X. Fan, K. Chen, G. Wang, M. Zhuang, Y. Li, and W. Xu, "NFGen: Automatic Non-linear Function Evaluation Code Generator for General-purpose MPC Platforms," in *CCS*, 2022, pp. 995–1008.

[52] J. Ma, Y. Zheng, J. Feng, D. Zhao, H. Wu, J. Tan, C. Yu, B. Zhang, and L. Wang, "SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning," in *USENIX ATC*, 2023, pp. 17–33.

[53] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023.

[54] F. Boemer, S. Kim, G. Seifu, F. D. de Souza, V. Gopal *et al.*, "Intel HEXL (release 1.2)," https://github.com/intel/hexl, 2021.

[55] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Leveled Fully Homomorphic Encryption without Bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 13:1–13:36, 2014.

[56] Y. Ishai and A. Paskin, "Evaluating Branching Programs on Encrypted Data," in *TCC*, 2007, pp. 575–594.

[57] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," *Advances in neural information processing systems*, vol. 35, pp. 15 718–15 731, 2022.

[58] J. Zhang, X. Yang, L. He, K. Chen, W.-j. Lu, Y. Wang, X. Hou, J. Liu, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," in *NDSS*, 2025.

[59] T. Xu, W.-j. Lu, J. Yu, Y. Chen, C. Lin, R. Wang, and M. Li, "Breaking the layer barrier: Remodeling private transformer inference with hybrid {CKKS} and {MPC}," in *USENIX Security*, 2025, pp. 2653–2672.

[60] Q. Zhang, T. Xiang, C. Xin, and H. Wu, "From Individual Computation to Allied Optimization: Remodeling Privacy-Preserving Neural Inference with Function Input Tuning," in *IEEE S&P*, 2024, pp. 104–104.

[61] S. Balla and F. Koushanfar, "HELiKs: HE Linear Algebra Kernels for Secure Inference," in *CCS*, 2023, pp. 2306–2320.

[62] F. B. Durak, C. Weng, E. Anderson, K. Laine, and M. Chase, "Precio: Private Aggregate Measurement via Oblivious Shuffling," in *CCS*, 2024.

[63] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," in *CRYPTO*, 1991, pp. 420–432.

[64] K. Zhong, Y. Ma, and S. Angel, "Ibex: Privacy-preserving ad conversion tracking and bidding," in *CCS*, 2022, pp. 3223–3237.

[65] M. Rathee, Y. Zhang, H. Corrigan-Gibbs, and R. A. Popa, "Private Analytics via Streaming, Sketching, and Silently Verifiable Proofs," in *IEEE S&P*, 2024, p. 666.

[66] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares," in *SCN*, 2022, pp. 516–539.

[67] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *FOCS*, 2009, pp. 169–178.

[68] C. Guo, J. Katz, X. Wang, and Y. Yu, "Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers," in *IEEE S&P*, 2020, pp. 825–841.

APPENDIX A
BASE-2 EXPONENTIATION

We show the correctness of our base-2 exponentiation protocol. Recall that we compute a share of $2^{\tilde{x}}$ from the share of $\tilde{x} \in [-T, 0)$ in this protocol. In Step 1, we ensure the shared value is greater-or-equal than one by adding an offset of $T+1$. In Step 2, we seperate the integral and fraction parts, i.e., $t + \tilde{m}_0 + \tilde{m}_1 = \tilde{x} + 1 + T + r$ for some roundings error $|r| < 2^{-(\ell_f+1)}$. As a result, we can evaluate the approximated base-2 exponentiation as $2^{\tilde{x}} \approx 2^t \cdot 2^{\tilde{m}_0} \cdot 2^{\tilde{m}_1} \cdot 2^{-(1+T)}$. Recall that $t = t_0 + t_1 \bmod 2^{\ell-\ell_f}$ can be viewed as an additive share over the modulus $2^{\ell-\ell_f}$. To evaluate the modulo $2^{\ell-\ell_f}$ over the exponentiation, we first convert the share of $[\![t]\!]^{2^{\ell-\ell_f}}$ to a share $[\![h]\!]^{p-1} = (h_0, h_1)$ for a prime number p in Step 3. That is $t = h_0 + h_1 \bmod p-1$. Then the arithmetic multiplication in Step 4 computes $2^{h_0}\lfloor 2^{\tilde{m}_0+\ell_f} \rfloor \cdot 2^{h_1}\lfloor 2^{\tilde{m}_1+\ell_f} \rfloor \bmod p$ resulting the share of $2^{t+\tilde{m}_0+\tilde{m}_1+2\ell_f} \bmod p$ due to the Fermat's Little Theorem. Indeed, this step generates a $\mathbb{F}_p$-share of $2^{\tilde{x}}$ with a fixed-point precision of $T+1+2\ell_f$ bits. The final step converts the modulus with the free truncation, and results at the share of $2^{\tilde{x}}$ under the 2^ℓ modulus.

APPENDIX B
CORRELATED AND-TRIPLE GENERATION

Our correlated AND-triple generation is as follows.

1) Run two instances of $\texttt{ROT}_2$. P_0 receives $(u_0, u_1) \in \mathbb{Z}_4 \times \mathbb{Z}_4$ and P_1 receives $(u_{c_1}, c_1) \in \mathbb{Z}_4 \times \mathbb{B}$ from the first instance. In the second instance, P_1 receives $(v_0, v_1) \in \mathbb{Z}_4 \times \mathbb{Z}_4$, and P_0 receives $(v_{c_0}, c_0) \in \mathbb{Z}_4 \times \mathbb{B}$.
2) Set $x_0 = c_0$, $y_0 = u_0 \oplus u_1$, and $z_0 = x_0 \wedge y_0 \oplus (u_0 \oplus v_{c_0})$. Set $x_1 = c_1$, $y_1 = v_0 \oplus v_1$, and $z_1 = x_1 \wedge y_1 \oplus (v_0 \oplus u_{c_1})$.
3) Parse the messages as Boolean shares. That is $x_b = \langle x \rangle_b, y_b = \langle y \rangle_b \| \langle y' \rangle_b$, and $z_b = \langle z \rangle_b \| \langle z' \rangle_b$.

Upon receiving $x_0 \in \mathbb{Z}_M$ from P_0 and $x_1 \in \mathbb{Z}_M$ from P_1, this functionality operates as follows: 1) Sample $y_0 \xleftarrow{\$} \mathbb{Z}_K$ and compute $y_1 = \mathsf{rep}_M(x_0 + x_1 \bmod M) - y_0 \bmod K$. 2) Give y_0 to P_0 and give y_1 to P_1.

Fig. 9: Modulus Conversion $\mathcal{F}_{\texttt{mconv}}$.

Upon receiving $x_0 \in \mathbb{Z}_M$ from P_0 and $x_1 \in \mathbb{Z}_M, d \in \mathbb{Z}^+$ from P_1, this functionality operates as follows: 1) Sample $y_0 \xleftarrow{\$} \mathbb{Z}_K$, and $E \xleftarrow{\$} \{0, \pm 1\}$. 2) Compute $y_1 = \lfloor \frac{\mathsf{rep}_M(x_0 + x_1 \bmod M)}{2^d} \rfloor + E - y_0 \bmod K$. 3) Give y_0 to P_0 and give y_1 to P_1.

Fig. 10: Truncation $\mathcal{F}_{\texttt{tr}}$.

For the correctness, it suffices to show

$$\begin{aligned}
z_0 \oplus z_1 &= (x_0 \wedge y_0) \oplus (v_0 \oplus v_{c_0}) \oplus (u_0 \oplus u_{c_1}) \oplus (x_1 \wedge y_1) \\
&= (x_0 \wedge y_0) \oplus (c_0 \wedge (v_0 \oplus v_1)) \oplus (c_1 \wedge (u_0 \oplus u_1)) \oplus (x_1 \wedge y_1) \\
&= (x_0 \wedge y_0) \oplus (x_0 \wedge y_1) \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1) \\
&= (x_0 \wedge (y_0 \oplus y_1)) \oplus (x_1 \wedge (y_0 \oplus y_1)) \\
&= (x_0 \oplus x_1) \wedge (y_0 \oplus y_1).
\end{aligned}$$

APPENDIX C
MISSING FUNCTIONALITIES AND PROTOCOLS

Figure 9 illustrates the modulus conversion functionality that we implement in this work, specifically for signed numbers. Figure 10 depicts the modulus conversion with truncation functionality that we also realize in this work.

A. $\mathbb{Z}_p$-OLE Constructions from RLWE

A classic construction of $\mathcal{F}_{\texttt{OLE}}^{p,N}$ for a prime modulus p, such as [27], [44], is given as follows[5]. We assume P_1 holds the decryption key. Recall N is one of the RLWE parameter.

1) P_1 sends P_0 a ciphertext $\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}}^{q,p}(\mathsf{SIMD}(\mathbf{b}))$.
2) P_0 responses to P_1 a ciphertext $\mathsf{ct}' = \mathsf{ct} \boxtimes \mathsf{SIMD}(\mathbf{a}) \boxminus \widetilde{\mathsf{Enc}}_{\mathsf{pk}}^{q,p}(\mathsf{SIMD}(\mathbf{u}))$ with $\mathbf{u} \xleftarrow{\$} \mathbb{F}_p^N$.
3) P_1 outputs $\mathbf{w} = \mathsf{SIMD}^{-1}(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}'))$, P_0 outputs $\mathbf{u}$.

The correctness simply reduces to the SIMD encoding and the homomorphic properties. The encryption $\widetilde{\mathsf{Enc}}_{\mathsf{pk}}^{q,p}(\cdot)$ is identical to the normal RLWE encryption $\mathsf{Enc}_{\mathsf{pk}}^{q,p}(\cdot)$ except using a statistically larger noise to achieve the circuit privacy [56]. In slightly more detail, we need to ensure that P_1 who holds the decryption key and one of the operand $\mathbf{b}$, can learn only the masked vector $\mathbf{w}$ but no other information about the other operand $\mathbf{a}$ and the random mask $\mathbf{u}$. The difficulty is that the noise contained in the evaluated RLWE ciphertext could reveal information about $\mathbf{a}$ and $\mathbf{u}$. The simple idea to achieve circuit privacy is a technique called noise-flooding [67]. That is, sample from a sufficiently wide distribution so as to flood the noise contained in the evaluated ciphertext.

[5]These two protocols were designed for active security. We present the passively secure version here.

Ferret's OT Extension:
Initialize: Receive a global key $\delta \in \mathbb{F}_{2^\lambda}$ from a sender.
Extend: Samples $\mathbf{v} \xleftarrow{\$} \mathbb{F}^n_{2^\lambda}$ and $\mathbf{u} \leftarrow \mathbb{F}^n_2$. Compute $\mathbf{w} = \mathbf{v} + \mathbf{u} \cdot \delta \in \mathbb{F}^n_{2^\lambda}$. Send $\mathbf{v}$ to sender, and $(\mathbf{u}, \mathbf{w})$ to receiver.

Fig. 11: Ferret's OT Extension $\mathcal{F}_{\mathtt{OTe}}$.

B. Secure Matrix Multiplication and Convolution

The current RLWE-based secure matrix multiplication and convolution protocols, such as those presented in [6], [23], support both two-power and prime moduli without incurring additional costs. Therefore, these protocols can be seamlessly integrated into our M&M framework.

Additionally, the matrix multiplication protocol by Pang et al. [21] may outperform the approach in [23] for specific matrix shapes. However, Pang et al.'s method is limited to supporting only a prime modulus. Despite this, we can apply our proposed modulus conversion protocol, allowing Pang et al.'s approach to be integrated into our framework without significantly increasing the cost.

C. Ferret's Delta OT

Yang et al. provide a communication-efficient way to achieve the functionality described in Figure 11 in their Ferret paper [22]. Basically this functionality allows to generate the correlation $\mathbf{w} = \mathbf{v} + \mathbf{u} \cdot \delta$ between the two parties for a fixed key $\delta \in \mathbb{F}_{2^\lambda}$ from the sender. The elements $\mathbf{w}, \mathbf{v}$ are in the Galois field $\mathbb{F}_{2^\lambda}$ where λ is the security level, e.g., $\lambda = 128$. Note the random choices of $\mathbf{u}$ are in the field $\mathbb{F}_2$. To achieve correlation in other domains such as $\mathbb{Z}_{2^\ell}$, Yang et al. use a tweakable correlation robust (TCR) hash function, such as the one described in [34], [68], that maps from $\mathbb{F}_{2^\lambda} \times \{0,1\}^\lambda$ to $\mathbb{Z}_{2^\ell}$ where the second input to the hash function is an tweakable index. In Figure 12, we provide a generalized version over a general modulus M. By setting $M = 2^\ell$ in Figure 12, it becomes Yang et al.'s Delta OT. The correctness can be seen as follows:

- When random choice generated by $\mathcal{F}_{\mathtt{OTe}}$ equals to the receiver's choice $\mathbf{u}[i] = \mathbf{c}[i]$, sender outputs $\mathbf{a}[i] = \mathsf{H}(\mathbf{v}[i], i)$. Particularly, when $\mathbf{c}[i] = 0$, receiver outputs $\mathbf{b}[i] = \mathsf{H}(\mathbf{w}[i], i)$ which equals to $\mathsf{H}(\mathbf{v}[i], i)$ in this case. When $\mathbf{c}[i] = 1$, receiver outputs $\mathbf{b}'[i] - \mathsf{H}(\mathbf{w}[i], i) = \mathbf{b}'[i] - \mathsf{H}(\mathbf{v}[i] + \delta, i) = \mathsf{H}(\mathbf{v}[i], i) + \Delta[i] \bmod M$.
- When choices $\mathbf{u}[i] \neq \mathbf{c}[i]$, sender outputs $\mathbf{a}[i] = \mathsf{H}(\mathbf{v}[i] + \delta, i)$. Particularly, when $\mathbf{c}[i] = 0$, receiver outputs $\mathbf{b}[i] = \mathsf{H}(\mathbf{w}[i], i)$ which equals to $\mathsf{H}(\mathbf{v}[i] + \delta, i)$ in this case. When $\mathbf{c}[i] = 1$, receiver outputs $\mathbf{b}'[i] - \mathsf{H}(\mathbf{w}[i], i) = \mathbf{b}'[i] - \mathsf{H}(\mathbf{v}[i], i) = \mathsf{H}(\mathbf{v}[i] + \delta, i) + \Delta[i] \bmod M$.

Two-power Modulus Delta OT. For a two-power modulus $M = 2^\ell$, the TCR hash function can be securely and efficiently instantiated using two AES calls, as described by [68]. That is $\mathsf{H}(x, i) = \pi(\pi(x) \oplus i) \oplus \pi(x)$ where $\pi(\cdot)$ is AES-128 with a fixed-key.

Prime Modulus Delta OT. We present a method to modify the two-power modulus Delta OT to obtain a Delta OT over a prime modulus. Instead of identifying a new TCR hash function suitable for a prime modulus, we can reuse the TCR hash function already employed for the two-power modulus Delta OT. Specifically, we define two variables: $y = \mathsf{H}_\lambda(x, i) \in \mathbb{Z}_{2^\lambda}$ and $y' = y \bmod p$. The key insight is that when the security parameter $\lambda \gg \log_2 p$ is sufficiently large—such as when $\lambda = 128$ and $p < 2^{64}$—the distribution of y' is statistically close to uniform over $\mathbb{F}_p$, assuming that y is uniformly random over $\mathbb{Z}_{2^\lambda}$. To further explain, we can express $2^\lambda = m \cdot p + r$, where $r \in [0, p)$. The total variation distance between the distribution of y' and a uniform distribution over $\mathbb{Z}_p$ is given by $\frac{r}{2^{\lambda+1}} < \frac{p}{2^{\lambda+1}}$ which should be small enough for a prime $p < 2^{64}$.

The communication complexity of the prime Delta OT remains identical to that of the two-power modulus case, requiring one ring element plus one choice bit per correlation.

Inputs: Sender (S) inputs a correlation vector $\Delta \in \mathbb{Z}^n_M$, and receiver (R) inputs a choice list $\mathbf{c} \in \mathbb{B}^n$. A TCR hash $\mathsf{H} : \mathbb{F}_{2^\lambda} \times \{0,1\}^\lambda \mapsto \mathbb{Z}_M$. Note $M \leq 2^\lambda$ and $1 \leq n \leq 2^\lambda$.
Outputs: S obtains $\mathbf{a} \in \mathbb{Z}^n_M$ and R obtains $\mathbf{b} \in \mathbb{Z}^n_M$ such that $\mathbf{b}[i] = \mathbf{a}[i] + \mathbf{c}[i] \cdot \Delta[i] \bmod M$ for $i \in [n]$.

```
1: Invoke $\mathcal{F}_{\mathtt{OTe}}$ which gives $\mathbf{v} \in \mathbb{F}^n_{2^\lambda}$ to S, and gives $\mathbf{w} \in \mathbb{F}^n_{2^\lambda}$
   and $\mathbf{u} \in \mathbb{F}^n_2$ to R. $\triangleright$ $\mathbf{w} = \mathbf{v} + \mathbf{u} \cdot \delta \in \mathbb{F}^n_{2^\lambda}$.
2: R sends the masked choices $\mathbf{c}' = \mathbf{u} \oplus \mathbf{c}$ to S. $\triangleright$ $\mathbb{F}_2 \cong \mathbb{B}$.
3: S sends $\mathbf{b}'[i] = \mathsf{H}(\mathbf{v}[i], i) + \mathsf{H}(\mathbf{v}[i] + \delta, i) + \Delta[i] \bmod M$ for
   $i \in [n]$ to R.
4: S outputs $\mathbf{a}[i] = \mathsf{H}(\mathbf{v}[i] + \mathbf{c}'[i] \cdot \delta \in \mathbb{F}_{2^\lambda}, i)$ for $i \in [n]$.
5: R sets each $\mathbf{b}[i]$ so if $\mathbf{c}[i] = 0$ then $\mathbf{b}[i] = \mathsf{H}(\mathbf{w}[i], i)$.
   Otherwise, $\mathbf{b}[i] = \mathbf{b}'[i] - \mathsf{H}(\mathbf{w}[i], i) \bmod M$ if $\mathbf{c}[i] = 1$.
```

Fig. 12: Δ-$\mathtt{OT}$ over a general modulus M from $\mathcal{F}_{\mathtt{OTe}}$.

TABLE XIII: Performance improvements via the fused OTs. Input length is 2^{18} with $\ell = 32$. Single thread. Here $\mathtt{OT}^n_2$ means n instances of the general OT of 2-bit messages.

Batch n	2	4	8	16	32
$\binom{16}{1}$-$\mathtt{OT}^n_2$	2.64s	4.91s	9.62s	18.80s	37.44s
$\binom{16}{1}$-$\mathtt{OT}_{2n}$	1.87s	2.53s	4.22s	7.12s	13.72s
Reduction	29%	48%	56%	62%	63%
Batch n	2	4	8	16	32
$\binom{16}{1}$-$\mathtt{OT}^n_2$	709b	1373b	2710b	5375b	10714b
$\binom{16}{1}$-$\mathtt{OT}_{2n}$	669b	1253b	2437b	4790b	9510b
Reduction	5%	8%	10%	10%	11%

Appendix D
Approximation via Piecewise Polynomial

A. One-to-Many Comparison Optimization

Suppose we want to compare one private value $x \in \mathbb{Z}_{2^\ell}$ from P_0 with a batch of private values $y_1, y_2, \cdots, y_n \in \mathbb{Z}_{2^\ell}$ from P_1. Particularly, the $\binom{2^m}{1}$-$\mathtt{OT}_2$ instances used to compute the inequalities and equalities for the digits of x, say x_i, and the corresponding digits of $y_1, y_2, \cdots y_n$ share the identical choice, i.e., the digit $x_i \in [0, 2^m)$. This motivates us to collapse the n instances of $\binom{2^m}{1}$-$\mathtt{OT}_2$ into one instance of $\binom{2^m}{1}$-$\mathtt{OT}_{2n}$, that is 1-of-2^m OT of $2n$-bits messages. This OT fusion can save significant computation time, particularly for

```
Hyper-parameters: The piecewise function $g_{d,m}$ defines $d$ pivot points $I_1, I_2, \cdots, I_d$, and the coefficients $\{a_{i,j}\}_{j\in[m+1]}$.
Inputs: A share of a fixed-point number $[\![x;\ell_f]\!]^{2^\ell}$ with the underlying message $\tilde{x} = \mathsf{rep}_{2^\ell}(x)/2^{\ell_f}$.
Outputs: $[\![u;\ell_f]\!]^{2^\ell}$ s.t. $\mathsf{rep}_{2^\ell}(u)/2^{\ell_f} \approx g_{d,m}(\tilde{x})$ (c.f. Eq. (3)).
1: Compute the comparison bits $\langle b_i = \mathbf{1}\{x > I_i\}\rangle$ using $d$ instances of $\mathcal{F}^{\ell-1}_{\mathtt{CMP}}$ for $i = 1, 2, \ldots, d$. The indicator bits are given by $\langle \mathbb{I}_i \rangle = \langle b_i \rangle \oplus \langle b_{i+1} \rangle$ for $1 \le i < d$. Additionally, $\langle \mathbb{I}_0 \rangle = 1 \oplus \langle b_1 \rangle$ and $\langle \mathbb{I}_d \rangle = \langle b_d \rangle$.
2: Convert the modulus $[\![x;\ell_f]\!]^p \leftarrow \Pi_{\mathtt{mconv}}([\![x;\ell_f]\!]^{2^\ell})$ for a prime $p \approx 2^\ell$ using the conversion protocol of Figure 3.
3: Compute the powers $[\![x^j;\ell_f]\!]^p$ for $j = 2, \ldots, m$ using the secure fixed-point multiplication protocol in §V-A.
4: Locally set the shares of the polynomial evaluations with double fixed-point precision. Specifically, set $[\![F_i(x);2\ell_f]\!]^p = \lfloor a_{i,0} \cdot 2^{2\ell_f} \rceil + \sum_{j=1}^m [\![x^j;\ell_f]\!]^p \cdot \lfloor a_{i,j} \cdot 2^{\ell_f} \rceil$ for the $i$-th polynomial $F_i(x) = a_{i,0} + \sum_{j=1}^m a_{i,j} x^j$.
5: Sum-up the multiplexers $[\![u;2\ell_f]\!]^p = \sum_{i=0}^d [\![\mathbb{I}_i \cdot F_i(x)]\!]^p$.
6: Invoke the modulus conversion protocol described in Figure 6 to obtain $[\![u;\ell_f]\!]^{2^\ell} \leftarrow [\![u;2\ell_f]\!]^p$. That is $\mathbb{F}_p$-to-$\mathbb{Z}_{2^\ell}$ conversion along with a free truncation.
```

Fig. 13: Oblivious Piecewise Polynomial Evaluation.

the Ferret OT, whose *computation time* for one $\binom{2^m}{1}$-$\mathtt{OT}_{128}$ is almost identical to that of one $\binom{2^m}{1}$-$\mathtt{OT}_2$. Empirically, as shown in Table XIII, this OT fusion can reduce about 30% – 60% time for the one-vs-many comparisons.

B. Piecewise Polynomials for GeLU and Sigmoid

Coefficients of GeLU. For function GeLU, we approximate it as $0.5x + P^4(|x|)$ in the narrow range $x \in [-3, 3)$. The coefficients of $P^4(|x|) = \sum_{i=0}^4 a_i|x|^i$ are:

$$a_0 = 0.001620808531841547,\ a_1 = -0.03798164612714154,$$
$$a_2 = 0.5410550166368381,\ a_3 = -0.18352506127082727,$$
$$a_4 = 0.020848611754127593.$$

Approximated Sigmoid. Given a ring share $[\![x;f]\!]^{2^\ell}$ of $\ell = 32$ and $f = 11$, we compute the ring share of $[\![\sigma(x);f]\!]^{2^\ell}$ using $\sigma(|x|) \approx 1 - \epsilon$ if $|x| > 5$, and $P^4(|x|)$ otherwise. Here $P^4(\cdot)$ is a degree-4 polynomial that approximates $\sigma(x)$ over the positive range $x \ge 0$. The coefficients of polynomials $P^4(|x| = \sum_{i=0}^4 b_i|x|^i)$ are as follows:

$$b_0 = 0.49352908920523014,\ b_1 = 0.3028510171823098,$$
$$b_2 = -0.06808619212463336,\ b_3 = 0.006747908074344234,$$
$$b_4 = -0.00024727769787340 74$$

We apply $\sigma(x) = 1 - \sigma(|x|)$ for $x < 0$. We need three comparisons i.e., $1\{x < 0\}$, $1\{x < -5\}$ and $1\{x < 5\}$.

C. Boolean-Multiply-Arithmetic

A classic way to multiply a boolean share by an arithmetic share using two instances of Δ-$\mathtt{OT}$ as follows:

1) P_0 sets $\Delta = (1 - 2\langle w \rangle_0) \cdot [\![a]\!]_0^M \bmod M$, and P_1 inputs choice $\langle w \rangle_1$ for a Δ-$\mathtt{OT}$. P_0 and P_1 obtains $u_0, u_1 \in \mathbb{Z}_M$, respectively. That is $u_1 - u_0 = \langle w \rangle_1 \cdot (1 - 2\langle w \rangle_0) \cdot [\![a]\!]_0^M$.
2) P_1 sets $\Delta = (1 - 2\langle w \rangle_1) \cdot [\![a]\!]_1^M \bmod M$, and P_0 inputs choice $\langle w \rangle_0$ for a Δ-$\mathtt{OT}$. P_0 and P_1 obtains $v_0, v_1 \in \mathbb{Z}_M$, respectively. That is $v_0 - v_1 = \langle w \rangle_0 \cdot (1 - 2\langle w \rangle_1) \cdot [\![a]\!]_1^M$.
3) P_i outputs $\langle w \rangle_i \cdot [\![a]\!]_i^M - (-1)^i \cdot u_i + (-1)^i \cdot v_i \bmod M$.

TABLE XIV: Model performance. (T=10, D=5) for GBDT.

Model	Dataset	Plaintext	Baselines	M&M
ResNet50	ImageNet-1K	0.765	0.764	0.764
DenseNet121	ImageNet-1K	0.743	0.742	0.742
BERT-base	GLUE/QNLI	0.903	0.902	0.902
ViT-base	ImageNet-1K	0.732	0.726	0.726
GBDT	phishing	0.957	0.957	0.957

Appendix E
Model Performance

Table XIV presents our model performance comparison across various models: i) For ResNet-50 and DenseNet121 on ImageNet-1K, we report Top-1 classification accuracy, comparing against Cheetah [6]. For BERT-base on GLUE/QNLI and ViT-base on ImageNet-1K, we measure Top-1 prediction and classification accuracy, with BumbleBee [23] as baseline. ii) For GDBT on the phishing dataset from Libsvm (11,055 samples, 67 features)[6], we apply 5-fold cross-validation and report average F1-scores on validation sets, benchmarking against Squirrel [28]. M&M achieves comparable or identical performance to both plaintext and cryptographic baselines. This equivalence validates the practicality of our positive heuristic in privacy-preserving ML tasks.

[6]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html, June 2022