Helix: Scalable Multi-Party Machine Learning Inference against Malicious Adversaries (Full Version)

Yansong Zhang[1,2,3], Xiaojun Chen[1,2,3], Qinghui Zhang[1,2,3], Ye Dong[4], and Xudong Chen[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, China
[2] State Key Laboratory of Cyberspace Security Defense, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, China
{zhangyansong,chenxiaojun,zhangqinghui,chenxudong}@iie.ac.cn
[4] National University of Singapore, Singapore
dongye@nus.edu.sg

Abstract. With the growing emphasis on data privacy, secure multi-party computation has garnered significant attention for its strong security guarantees in developing privacy-preserving machine learning (PPML) schemes. However, only a few works address scenarios with a large number of participants. The state of the art by Liu *et al.* (LXY24, USENIX Security'24) first achieves a practical PPML protocol for up to 63 parties but is constrained to semi-honest security. Although naive extensions to the malicious setting are possible, they would introduce significant overhead.

In this paper, we propose Helix, a scalable framework for maliciously secure PPML in the honest majority setting, aiming to enhance both the scalability and practicality of maliciously secure protocols. In particular, we report a privacy leakage issue in LXY24 during prefix OR operations and introduce a round-optimized alternative based on a single-round vectorized three-layer multiplication protocol. Additionally, by exploiting *reusability* properties within the computation process, we propose lightweight compression protocols that substantially improve the efficiency of multiplication verification. We also develop a batch check protocol to reduce the computational complexity of revealing operations in the malicious setting. For 63-party neural network inference, compared to the semi-honest LXY24, Helix is only $1.9\times$ $(1.1\times)$ slower in the online phase and $1.2\times$ $(1.1\times)$ slower in preprocessing under LAN (WAN) in the best case.

Keywords: secure multi-party computation · malicious security · honest majority · privacy-preserving machine learning.

1 Introduction

Machine learning (ML) is increasingly applied across diverse domains, including medicine, finance, and recommendation systems. However, this widespread success has raised significant privacy concerns regarding both models and personal data. As such, privacy-preserving techniques should be employed to ensure the privacy of the data used in machine-learning-as-a-service.

Secure multi-party computation (MPC) [38] is a notable approach for enabling privacy-preserving machine learning (PPML). MPC allows n parties to collaboratively compute a function over their private inputs while ensuring input privacy and output correctness. Nowadays, MPC-based PPML has made significant progress, particularly under the semi-honest setting [15,29,37], where adversaries honestly follow the protocol but try to learn secret values. Nevertheless, in real-world scenarios involving large numbers of participants, such as federated learning, expecting that all individual parties are semi-honest is overly strong and impractical. Therefore, there is an urgent need for scalable protocols that support efficient PPML with many parties in the malicious setting, where adversaries can arbitrarily deviate from the protocol.

However, existing malicious PPML protocols primally focus on 2-4 parties [10,19,21,28,32,39], where one party is corrupt. In the most recent work, Liu et al. [24] utilize Shamir secret sharing [35] to design scalable PPML protocols under the semi-honest secure honest majority model. Their approach enables efficient PPML inference with up to 63 parties. For ease of reference, we denote this work [24] as "LXY24". Trivially, the protocols in LXY24 can be adapted to malicious security using standard techniques [3,4,17,23], with the primary challenge being the verification of multiplication and revealing operations. Nevertheless, due to the inherent design limitations of the protocols in LXY24 and large scale of applications, the naive extensions face two critical issues as follows.

First, a privacy leakage issue exists in LXY24. The comparison protocol in LXY24 achieves both constant rounds and low communication costs without requiring a gap between the domains of shares and secrets. However, rigorous security analysis reveals a privacy leakage issue in the prefix OR protocol, which serves as the core and bottleneck of the comparison protocol. This vulnerability can expose all input values of the prefix OR operation. Consequently, it indicates that the n-party comparison protocols over fields, with no gap requirement, still encounter a trade-off[5] between rounds and communication costs.

Second, the verification step exhibits performance limitations. In large-scale applications, the number of verified multiplication triples N and revealing operations m are both considerable. In such cases, the multiplication verification method in [23] incurs significant communication overhead due to its complexity linear in N. Protocols that achieve sublinear communication [3, 4, 17] have higher computational complexity, which also grows linearly with N. Similarly, for reconstruction with Shamir secret sharing, the computational complexity of the correctness check is linear in m. As a result, in practical scenarios with large N and m, such as neural network (NN) inference, existing verification methods either suffer from high communication overhead or computational complexity, which causes a significant performance gap between malicious and semi-honest protocols.

[5] For further details, please refer to § 7.

To address the first issue, we propose a secure and round-optimized alternative solution for the prefix OR operation based on the DN [13] protocol. Specifically, by leveraging the Beaver triples [2], we extend the vectorized two-layer DN multiplication in LXY24 to a three-layer version without increasing online communication. The two-layer and three-layer multiplication protocols enable prefix multiplications with 2-4 inputs to be completed in only one online round. Ultimately, by utilizing these efficient building blocks and performing computations in groups of four, we construct an efficient prefix multiplication protocol that effectively resolves the dual problem of prefix OR.

To address the second issue, we primarily utilize the *random linear combination* technique [8, 14], which introduces randomness to reduce the verification scale while ensuring correctness with overwhelming probability. First, we employ the multiplication verification protocol in [17] to achieve sublinear communication and tailor it to machine learning applications based on the linearity of Shamir secret sharing. In addition, to further reduce computational complexity, our key insight is that large numbers of multiplication operations, such as those in matrix multiplication, share a common multiplier or involve multipliers related through affine transformations. Thus, rather than verifying such triples independently, we propose a lightweight compression protocol to consolidate them into a single triple with random linear combinations. In this way, we can significantly reduce N, thereby improving verification efficiency and enhancing practicality. Specifically, we reduce the verification count to $\frac{1}{mo}$ of the original requirement for two matrices of dimensions $m \times n$ and $n \times o$. Moreover, for the verification of revealed sharings, we optimize computational complexity by deferring the process to a batch execution after circuit evaluation, which is similar to the batch MAC check procedure in SPDZ [12, 14].

Building upon the above techniques, we enhance the protocols in LXY24 to efficiently achieve malicious security and introduce Helix, a scalable framework designed for multi-party machine learning inference with malicious security in the honest majority setting.

Contributions In brief, we summarize our main contributions as follows.

- We identify and address the privacy leakage issue in the prefix OR protocol proposed in LXY24. Specifically, we adopt a three-layer version of the vectorized two-layer multiplication protocol in LXY24, without increasing online communication. We further combine these two protocols to design an optimized prefix OR protocol, achieving the online complexity of $\log_4 \ell$ rounds with $O(\ell \log_4 \ell)$ field elements per party.
- We present a lightweight multiplication triple compression protocol by leveraging the reusability properties, leading to a significant reduction in the number of verified multiplications. This new protocol works for any verification technique, which is of independent interest. Additionally, we propose a batch check protocol for revealed sharings, which reduces the computational complexity of revealing operations. Furthermore, we extend the multiplication verification protocol in [17] to support machine learning applications.

– We implement Helix and evaluate its performance in secure NN inference with varying numbers of parties, ranging from 3 to 63. The results highlight the practicality and scalability of Helix. In the best case, Helix is $1.9\times$ ($1.1\times$) slower in the online phase and $1.2\times$ ($1.1\times$) slower in preprocessing under LAN (WAN) compared to the state-of-the-art semi-honest secure protocol, LXY24 [24]. Additionally, compared to the naive approach that directly employs the protocol from [17] for malicious security, Helix achieves at most $2.6\times$ faster online NN inference in LAN.

2 Preliminaries

Notations Let $P_1, \dots, P_n$ be the n parties to do the secure computation. We denote scalar, vector, and matrix by lowercase letter x, lowercase bold letter $\boldsymbol{x}$, and uppercase bold letter $\mathbf{X}$, respectively. Let $\mathbf{X}(i,j)$ denote the element at the i-th row and j-th column in matrix $\mathbf{X}$, and $\boldsymbol{x}(i)$ the i-th element in vector $\boldsymbol{x}$. Denote the functionality and protocol used in the semi-honest setting by $\mathcal{F}^{\mathsf{sh}}$ and Π^{sh}, respectively. Similarly, $\mathcal{F}^{\mathsf{mal}}$ ($\mathcal{F}$ for simplicity) and Π^{mal} (Π for simplicity) are employed in the malicious setting.

2.1 Shamir Secret Sharing

This work is based on Shamir's (t,n)-threshold scheme [35], where n is the number of parties, t is the number of corrupted parties, and $n \geq 2t+1$. For the rest of the paper, we assume maximal corruption in this setting, and thus $n = 2t+1$. To share a secret $x \in \mathbb{F}_p$ with degree t, a uniformly random polynomial $f(X)$ of degree t is chosen under the constraint that $f(0) = x$. Each party P_i holds the share $f(\alpha_i)$, where $\alpha_i \in \mathbb{F}_p$ is the unique identifier for P_i. For convenience, we set $\alpha_i = i$. Following LXY24, the finite field $\mathbb{F}_p$ is defined over a fixed Mersenne prime $p = 2^\ell - 1$, with examples including $\ell = 31, 61$.

We denote the degree-t sharing as $[\cdot]$-sharing. Multiplying two degree-t sharings yields a degree-$2t$ sharing, denoted by $[\![\cdot]\!]$-sharing, with no communication. Additionally, any linear function or addition by a constant can be performed locally. Thus, for simplicity, we write $[a + b \bmod p] = [a+b] = [a] + [b]$ and $[\![a \cdot b \bmod p]\!] = [\![ab]\!] = [a] \cdot [b]$ for shares $[a]$ and $[b]$.

2.2 Useful Techniques in the Semi-Honest Setting

Sharing and Revealing As explained above, $\Pi^{\mathsf{sh}}_{\mathsf{Share}}$ works as follows: the dealer chooses a random polynomial $f(X)$ and sends each P_i the point $f(i)$. To reveal a secret $[x]$, at least t parties send their shares to P_{king}, who then reconstructs x and sends it back to other parties. We write $x \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([x])$ and $x \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([\![x]\!])$ for the revealing of degree-t and degree-$2t$ sharings[6], respectively. Following LXY24, we measure the round complexity of a protocol by the number of rounds of parallel invocations of $\Pi^{\mathsf{sh}}_{\mathsf{Reveal}}$.

[6] Similarly, for degree-$2t$ sharings, at least $2t$ parties send their shares to P_{king}.

Random Sharings Generation Based on the Vandermonde matrix and sharing protocol $\Pi^{\mathsf{sh}}_{\mathsf{Share}}$, random degree-$t$ sharings and random double sharings can be efficiently generated. We define the generation protocols as $[r] \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Rand}}$ and $([r], [\![r]\!]) \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{DoubleRand}}$, respectively.

DN Multiplication The multiplication between two degree-t sharings can be realized by DN [13] multiplication with the use of $([r], [\![r]\!])$. We write the DN multiplication protocol as $[xy] \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Mult}}([x], [y])$. In essence, the DN multiplication protocol performs a degree reduction from $[\![xy]\!]$ to $[xy]$ by revealing $[\![xy]\!] + [\![r]\!]$ and computing $[xy] = (xy + r) - [r]$. For two vectors of degree-t sharings $[\boldsymbol{x}]$ and $[\boldsymbol{y}]$, to compute $[\boldsymbol{x} \cdot \boldsymbol{y}]$, we can first compute $[\![\boldsymbol{x} \cdot \boldsymbol{y}]\!] = [\boldsymbol{x}] \cdot [\boldsymbol{y}]$ and then apply the same approach as the DN multiplication protocol to compute $[\boldsymbol{x} \cdot \boldsymbol{y}]$ from $[\![\boldsymbol{x} \cdot \boldsymbol{y}]\!]$. We define the protocol as $\Pi^{\mathsf{sh}}_{\mathsf{InnerProd}}$.

The consecutive multiplication of three degree-t sharings can be achieved in a single online round through the vectorized two-layer DN multiplication protocol [16, 24], denoted as $\{xyz_i\}_{i=1}^m \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{2L\text{-}DN}}([x], [y], \{[z_i]\}_{i=1}^m)$. In the preprocessing phase, parties generate $m + 1$ pairs of double sharings $([r], [\![r]\!])$ and $\{([r_i], [\![r_i]\!])\}_{i=1}^m$. In the online phase, parties first compute $[\![u]\!] = [x] \cdot [y] + [\![r]\!]$ and $[\![u_i]\!] = [r] \cdot [-z_i] + [\![r_i]\!]$ locally. Subsequently, they simultaneously reveal $[\![u]\!]$ and $[\![u_i]\!]$, and the result $[xyz_i]$ can be expressed as $u \cdot [z_i] + u_i - [r_i]$.

Optimization using PRG The communication complexity of the above protocols can be further reduced by utilizing pseudo-random generators (PRG) [16,23]. For a detailed theoretical complexity analysis, see [24]. In the complexity analysis across this work, we adopt the costs from the PRG-optimized version.

2.3 Useful Techniques in the Malicious Setting

Correctness Check of Revealed Degree-t Sharings In the malicious setting, revealing a degree-t sharing requires each party to distribute its share to all other parties. Then the correctness of the revealed sharings can be checked as follows: each party P_i uses any of the $t + 1$ shares to compute the unique degree-t polynomial, and checks that all other shares lie on the same polynomial. If not, then it outputs $\perp$ and aborts. We write the protocol for revealing degree-t sharings with correctness check as $\Pi_{\mathsf{Reveal\text{-}Check}}$.

Batch Correctness Check of Shares In [23], the correctness of m degree-t sharings can be checked in batch utilizing $\Pi_{\mathsf{Reveal\text{-}Check}}$. For $[x_1], \ldots, [x_m]$, the parties first generate m non-zero random field elements $\{\alpha_i\}_{i=1}^m$ and a random sharing $[r]$ using $\Pi^{\mathsf{sh}}_{\mathsf{Rand}}$. Then compute $[v] = \sum_{i=1}^m \alpha_i \cdot [x_i] + [r]$ locally. Finally, each party broadcasts its share of $[v]$ and checks the correctness using $\Pi_{\mathsf{Reveal\text{-}Check}}$. If no abort, then the parties accept all input sharings. We write the above process as $\Pi_{\mathsf{ShareCheck}}$. Based on $\Pi_{\mathsf{ShareCheck}}$, it is easy to obtain maliciously secure protocols $\Pi_{\mathsf{Share}}, \Pi_{\mathsf{Rand}}$ and $\Pi_{\mathsf{DoubleRand}}$. For more details, please refer to [23].

Random Coins Generation There are two methods to implement the protocol Π_{Coin}. First, as described in [17], [23], the parties invoke $\Pi^{\mathsf{sh}}_{\mathsf{Rand}}$ to generate a random sharing and then reveal the result with the use of $\Pi_{\mathsf{Reveal\text{-}Check}}$. The second method, as described in [12], requires each party to sample a seed and broadcast its commitment. The parties then reveal all commitments and XOR the seeds. Finally, random field elements can be generated by utilizing PRG.

2.4 Security Model

We consider security against static malicious adversaries in the honest-majority setting. Specifically, an adversary corrupts $t < n/2$ parties at the beginning of the protocol execution and can deviate from the protocol specification arbitrarily. The goal of our security model is to prove that the malicious adversary has no impact on the computational result or the privacy of the sensitive inputs. Formally, we model and prove the security of our protocols under the universal composition (UC) framework [5], and assume familiarity with this.

3 Improved Bitwise Primitives

In this section, we first analyze the potential privacy leakage issue associated with the prefix OR protocol introduced in LXY24 (§ 3.1). Next, we propose a semi-honest secure and round-optimized alternative solution (§ 3.4) leveraging two- and three-layer DN multiplication protocols (§ 3.2, § 3.3). In § 5, we apply these protocols to the malicious setting.

3.1 The Security Issue in LXY24

In LXY24, the prefix OR operation for inputs $\{a_i\}_{i=1}^{\ell}$, where $a_i \in \{0,1\}$, is calculated by solving its dual problem. This involves first computing $\bar{a}_i = 1 - a_i$, and then performing prefix multiplication $\mathcal{F}^{\mathsf{sh}}_{\mathsf{PreMult}}$ on $\{\bar{a}_i\}_{i=1}^{\ell}$. The resulting prefix OR values $\{b_j\}_{j=1}^{\ell}$ are given by $b_j = 1 - \prod_{i=1}^{j} \bar{a}_i$. The functionality $\mathcal{F}^{\mathsf{sh}}_{\mathsf{PreMult}}$ in LXY24 is implemented based on the constant-round prefix multiplication techniques proposed in [1, 6, 11, 30]. Specifically, in the preprocessing phase, parties prepare ℓ pairs of correlated random sharings $([r_i], [r'_i])$ where $r'_1 = r_1^{-1}$ and $r'_i = r_{i-1} r_i^{-1}$ for $i > 1$. In the online phase, for each $i \in [1, \ell]$, the parties compute and reveal $c_i \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{MultPub}}([\bar{a}_i], [r'_i])$ and then locally compute $[r_i] \cdot \prod_{j=1}^{i} c_j$, which yields the prefix products.

However, we observe that this instantiation fails to UC-securely realize the functionality $\mathcal{F}^{\mathsf{sh}}_{\mathsf{PreMult}}$ as defined in LXY24, because the public values $\{c_i\}_{i=1}^{\ell}$ are not uniformly distributed over $\mathbb{F}_p$ when handling binary inputs, leading to a potential privacy leakage issue [7]. To begin with, all the correlated random elements $\{(r_i, r'_i)\}_{i=1}^{\ell}$ are non-zero, as the computation of r'_i necessitates a multiplicative

[7] Previous works [1, 6, 11, 30] handle inputs over $\mathbb{F}_p^*$ and thus they do not encounter such issue.

inverse. Moreover, each element in $\{\bar{a}_i\}_{i=1}^{\ell}$ is either 0 or 1 in the prefix OR protocol. As a consequence, if $\bar{a}_i = 0$, the corresponding public value c_i is also 0, whereas if $\bar{a}_i = 1$, c_i becomes non-zero. Hence, an adversary can deduce the private values $\{\bar{a}_i\}_{i=1}^{\ell}$ by observing if the public values $\{c_i\}_{i=1}^{\ell}$ are equal to 0.

3.2 Vectorized Three-Layer DN Multiplication

We extend $\Pi^{\mathsf{sh}}_{\mathsf{2L\text{-}DN}}$ to a three-layer protocol using Beaver triples [2], without increasing online communication cost. For ease of description, we consider the general case where the last two multipliers are vectors of the same length, i.e., calculating $\{[xyz_iw_i]\}_{i=1}^{m}$. we begin by computing the standard DN multiplication $[\![u]\!] = [x]\cdot[y] + [\![r]\!]$ and $[\![u_i]\!] = [z_i]\cdot[w_i] + [\![r_i]\!]$, where $([r],[\![r]\!])$ and $\{([r_i],[\![r_i]\!])\}_{i=1}^{m}$ are random double sharings. Since $[r]$ and $[r_i]$ are generated in the preprocessing phase, $[r\cdot r_i]$ can be precomputed, which forms a Beaver triple $([r],[r_i],[r\cdot r_i])$. Next, u and u_i can be revealed in a single round, and then $[xyz_iw_i]$ is computed in the Beaver form as $u\cdot u_i - u\cdot[r_i] - u_i\cdot[r] + [c_i]$. Therefore, the vectorized three-layer DN multiplication $\Pi^{\mathsf{sh}}_{\mathsf{3L\text{-}DN}}$ can be realized with a single round of online complexity. The detailed protocol is described in Fig. 1.

Protocol $\Pi^{\mathsf{sh}}_{\mathsf{3L\text{-}DN}}$

Input: $[x],[y],\{[z_i]\}_{i=1}^{m}$ and $\{[w_i]\}_{i=1}^{m}$
Output: $\{[xyz_iw_i]\}_{i=1}^{m}$
Preprocessing:

1. Invoke $\mathcal{F}^{\mathsf{sh}}_{\mathsf{DoubleRand}}$ to generate $m+1$ pairs of double sharings $([r],[\![r]\!])$ and $\{([r_i],[\![r_i]\!])\}_{i=1}^{m}$.
2. For $1 \le i \le m$, all parties invoke $\mathcal{F}^{\mathsf{sh}}_{\mathsf{Mult}}$ on $([r],[r_i])$ to compute $[c_i]$.

Online:

1. Compute $[\![u]\!] = [x]\cdot[y] + [\![r]\!]$ and $[\![u_i]\!] = [z_i]\cdot[w_i] + [\![r_i]\!]$ for $1 \le i \le m$.
2. Reveal $u \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([\![u]\!])$ and $u_i \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([\![u_i]\!])$.
3. For $1 \le i \le m$, all parties locally compute $[xyz_iw_i] = u\cdot u_i - u\cdot[r_i] - u_i\cdot[r] + [c_i]$.

Fig. 1. Vectorized three-layer DN multiplication protocol.

The online complexity is 1 round with $2(m+1)$ field elements per party. As for the preprocessing phase, all random values can be generated in 2 rounds with $4m+1$ field elements per party (including $m+1$ for $\Pi^{\mathsf{sh}}_{\mathsf{DoubleRand}}$, and $3m$ for $\Pi^{\mathsf{sh}}_{\mathsf{Mult}}$).

Moreover, our extension approach can generalize DN multiplication to an arbitrary of inputs, albeit requiring exponential communication in the preprocessing phase [20]. We restrict our implementations to 2-4 inputs, as the protocols involving more inputs yield marginal online performance improvements for subsequent protocols while significantly increasing the preprocessing overhead.

3.3 Block Prefix Multiplication

Block prefix multiplication is defined as the prefix multiplication for a small number of inputs, such as 3 or 4, which can be realized using $\Pi^{\mathsf{sh}}_{\mathsf{2L\text{-}DN}}$ and $\Pi^{\mathsf{sh}}_{\mathsf{3L\text{-}DN}}$. Specifically, the three-element prefix multiplication protocol $\Pi^{\mathsf{sh}}_{\mathsf{PreMul3}}$ can be trivially derived from the protocol $\Pi^{\mathsf{sh}}_{\mathsf{2L\text{-}DN}}$, with $[xy]$ computed locally as $u-[r]$. The four-element prefix multiplication protocol $\Pi^{\mathsf{sh}}_{\mathsf{PreMul4}}$ described in Fig. 2 can be constructed by combining $\Pi^{\mathsf{sh}}_{\mathsf{PreMul3}}$ and $\Pi^{\mathsf{sh}}_{\mathsf{3L\text{-}DN}}$ in parallel, where $[xy]$ and $[xyz]$ are computed using $\Pi^{\mathsf{sh}}_{\mathsf{PreMul3}}$, and $[xyzw]$ is computed through $\Pi^{\mathsf{sh}}_{\mathsf{3L\text{-}DN}}$. Especially, the resulting protocol requires only 3 reveals since $[\![u]\!] = [x] \cdot [y] + [\![r]\!]$ needs to be computed and revealed just once.

For $\Pi^{\mathsf{sh}}_{\mathsf{PreMul3}}$, the complexity is identical to that of $\Pi^{\mathsf{sh}}_{\mathsf{2L\text{-}DN}}$. As for $\Pi^{\mathsf{sh}}_{\mathsf{PreMul4}}$ (in comparison to the PreOpL method in [34]), the online complexity is reduced to 1 round (from 2 rounds) with 6 field elements per party (reduced from 8 elements), while the preprocessing complexity increases to 2 rounds (from 1 round) with 6 field elements per party (increased from 4 elements).

Protocol $\Pi^{\mathsf{sh}}_{\mathsf{PreMul4}}$

Input: $[a_1], [a_2], [a_3]$ and $[a_4]$
Output: $[a_1], [a_1 \cdot a_2], [a_1 \cdot a_2 \cdot a_3], [a_1 \cdot a_2 \cdot a_3 \cdot a_4]$
Preprocessing:

1. Invoke $\mathcal{F}^{\mathsf{sh}}_{\mathsf{DoubleRand}}$ to generate 3 pairs of double sharings $([r], [\![r]\!])$, $([r_1], [\![r_1]\!])$ and $([r_2], [\![r_2]\!])$.
2. Invoke $\mathcal{F}^{\mathsf{sh}}_{\mathsf{Mult}}$ on $([r], [r_2])$ to compute $[c]$.

Online:

1. Compute $[\![u]\!] = [a_1]\cdot[a_2]+[\![r]\!]$, $[\![u_1]\!] = [r]\cdot[-a_3]+[\![r_1]\!]$ and $[\![u_2]\!] = [a_3]\cdot[a_4]+[\![r_2]\!]$.
2. Reveal $u \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([\![u]\!])$, $u_1 \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([\![u_1]\!])$ and $u_2 \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([\![u_2]\!])$.
3. Compute $[b] = u_1 - [r_1]$.
4. All parties locally compute $[a_1 \cdot a_2] = u - [r]$, $[a_1 \cdot a_2 \cdot a_3] = u \cdot [a_3] + [b]$ and $[a_1 \cdot a_2 \cdot a_3 \cdot a_4] = u \cdot u_2 - u \cdot [r_2] - u_2 \cdot [r] + [c]$.

Fig. 2. Four-element prefix multiplication protocol.

For $\Pi^{\mathsf{sh}}_{\mathsf{PreMul3}}$, the online complexity is 1 round with 4 field elements per party and the preprocessing complexity is 1 round with 2 field elements per party. As for $\Pi^{\mathsf{sh}}_{\mathsf{PreMul4}}$, the online complexity is 1 round with 6 field elements per party and the preprocessing complexity is 2 rounds with 6 field elements per party.

3.4 Prefix OR Protocol

Our prefix OR protocol $\Pi^{\mathsf{sh}}_{\mathsf{PreOR}}$ follows the method in LXY24, with the substitution of our secure prefix multiplication subprotocol for binary inputs. Fig. 3 shows our optimized prefix multiplication construction for 61-bit inputs, where operators ○, ◈, and ■ represent two-element, three-element, and four-element multiplications, respectively.

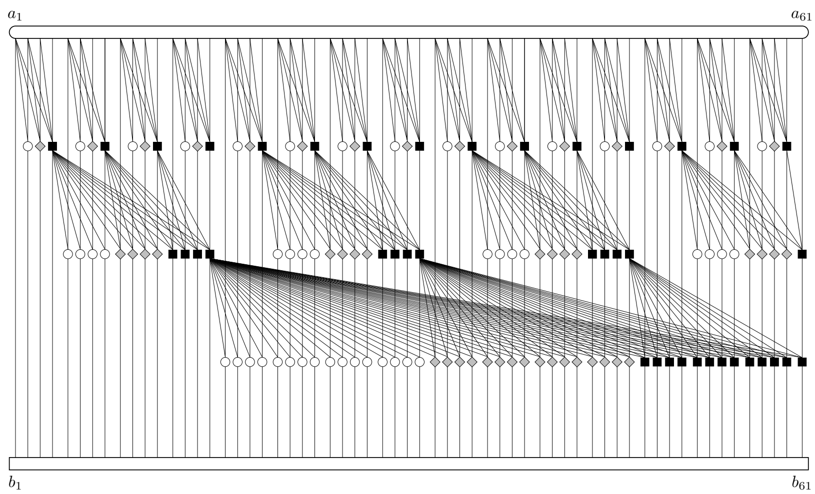

Fig. 3. Prefix multiplication circuit for 61-bit inputs, where $b_i = \prod_{j=1}^{i} a_j$ for $i \in [1, \ell]$.

In our construction, computations in the first round can be sequentially grouped into sets of four and efficiently executed using our block prefix multiplication protocols. From the second round, consecutive identical operators can be bundled together and computed collectively using vectorized DN multiplication protocols. As a result, by leveraging the protocols in § 3.2 and § 3.3, we derive our efficient prefix multiplication protocol $\Pi^{\mathsf{sh}}_{\mathsf{PreMult}}$.

Since $\Pi^{\mathsf{sh}}_{\mathsf{PreMult}}$ is fundamentally based on the DN multiplication protocol, it avoids the security issues encountered with binary inputs in [24]. Additionally, the associativity of field multiplication allows for a straightforward correctness proof of our prefix multiplication circuit. Hence, our protocol $\Pi^{\mathsf{sh}}_{\mathsf{PreMult}}$ UC-secure realizes the functionality $\mathcal{F}^{\mathsf{sh}}_{\mathsf{PreMult}}$ as described in [24].

Note that the prefix OR only requires one invocation of prefix multiplication, while other steps can be done locally. Therefore, the resulting protocol $\Pi^{\mathsf{sh}}_{\mathsf{PreOR}}$ achieves a better trade-off between online rounds and communication, with only a slight increase in preprocessing complexity. Specifically, the online complexity is $\log_4 \ell$ rounds with $O(\ell \log_4 \ell)$ field elements per party, while the PreOpL protocol in [34] requires $\log_2 \ell$ rounds and $O(\ell \log_2 \ell)$ field elements per party. Details are given in Table 1. Furthermore, using this efficient $\Pi^{\mathsf{sh}}_{\mathsf{PreOR}}$, we could design a fast comparison protocol with $O(\log_4 \ell)$ online rounds.

4 Achieving Malicious Security

Helix uses batch verification techniques to detect any potential malicious behavior in multiplication and degree-t revealing operations. In particular, after the circuit evaluation step, Helix initially employs compression protocols (§ 4.1) to substantially reduce the number of required multiplication verifications. Then,

Table 1. Round and communication complexity of prefix OR protocol with Shamir secret sharing. Numbers of communication are reported in field elements per party. The naive approach performs the OR operation sequentially ℓ times for ℓ inputs.

Protocol	31 bits				61 bits			
	Rounds		Communication		Rounds		Communication	
	Online	Prep.	Online	Prep.	Online	Prep.	Online	Prep.
Naive	30	1	60	30	60	1	120	60
PreOpL [34]	5	1	150	75	6	1	352	176
Ours	3	2	130	107	3	2	290	268

it utilizes the general batch multiplication verification protocol (§ 4.2) to verify the compressed multiplication triples. Lastly, Helix introduces a batch checking for revealed degree-t sharings (§ 4.3).

Before delving into the details of our maliciously secure protocols, we first provide an important lemma that will be applied in the proofs of almost all subsequent lemmas.

Lemma 1. *Let $\delta_1, \delta_2, \ldots, \delta_m \in \mathbb{F}_p$, where not all the δ_i's are zero, and suppose that each δ_i is independent from the uniform distribution sampling α_j, for any $j \in [1, m]$. Then we have*

$$\Pr_{\alpha_1,\ldots,\alpha_m \leftarrow_R \mathbb{F}_p}\left(\sum_{i=1}^{m} \alpha_i \cdot \delta_i = 0\right) \leq \frac{1}{p}.$$

Proof. Suppose that $\delta_k \neq 0$. Thus, in this case, $\sum_{i=1}^{m} \alpha_i \cdot \delta_i = 0$ if and only if

$$\alpha_k = -\delta_k^{-1} \cdot (\sum_{i=1, i\neq k}^{m} \alpha_i \cdot \delta_i). \tag{1}$$

Since α_k is randomly sampled and chosen independently of all other values, the probability that Eq. (1) holds is at most $\frac{1}{p}$.

4.1 Lightweight Compression protocol

In this subsection, we propose lightweight multiplication triple compression protocols based on the property of *reusability.* These protocols effectively reduce the number of triples to be verified while ensuring security.

Intuition We begin with the definition of *reusable multiplication triples* and *reusable inner-product triples.*

Definition 1 (Reusable Multiplication Triples). *For m triples (x_1, y_1, z_1), ..., (x_m, y_m, z_m), where $z_i = x_i \cdot y_i$ for all $i \in [1, m]$, if each x_i can be expressed* *as a public invertible affine transformation of a common value x over $\mathbb{F}_p$, such that $x_i = f_i(x) = a_i \cdot x + b_i$ with $a_i \neq 0$, and each y_i is unique, then these m triples are defined as reusable multiplication triples with a reuse degree of m.*

The definition of reusable inner-product triples can be derived straightforwardly from Definition 1 by substituting multiplication triples with inner-product triples. When existing multiplication verification protocols [17, 23] are applied to verify the correctness of reusable triples, they treat these triples as independent entities, ignoring the inherent interdependence among them [26,32]. This introduces considerable redundant computations, resulting in substantial verification overhead. To address this, our intuition is to *compress these reusable triples into a single one*, inspired by the batch MAC check procedure in SPDZ [12,14]. Taking reusable multiplication triples as an example, since all affine transformations are publicly known and invertible, each x_i can be converted into the common value x using the inverse transformation $f_i^{-1}(x_i) = a_i^{-1}(x_i - b_i)$. Additionally, y_i and z_i can be processed locally to compute $z'_i = a_i^{-1} z_i - a_i^{-1} b_i y_i$, which forms the new triple (x, y_i, z'_i). Furthermore, we employ the linear combination technique with random coefficients α_i to compress y_i and z'_i, such that $y = \sum_{i=1}^{m} \alpha_i y_i$ and $z = \sum_{i=1}^{m} \alpha_i z'_i$. Since all these triples share the common value x, the resulting combined values naturally satisfy the multiplication relationship $z = x \cdot y$.

It is worth noting that the computations in machine learning, such as neural network inference, exhibit the strong reusability property. Specifically, matrix multiplication, which is widely used in fully connected and convolutional layers of NN, contains reusable inner-product triples, while comparison used in almost all of the activation functions of NN, includes reusable multiplication triples.

Protocols Our compression protocol for reusable multiplication triples is illustrated in Fig. 4. According to the reuse degree m, the number of multiplication triples to be verified can be reduced to $\frac{1}{m}$ of the original amount.

Protocol Π_{ReComp}

Input: $([x_1], [y_1], [z_1]), ..., ([x_m], [y_m], [z_m])$, where $x_i = a_i \cdot x + b_i$ with $a_i \neq 0$
Output: $([x], [y], [z])$
The Procedure:
1. For $1 \leq i \leq m$, all parties set $[z'_i] = a_i^{-1}[z_i] - a_i^{-1} b_i [y_i]$.
2. Invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate m random elements $\alpha_1, ..., \alpha_m$.
3. All parties locally compute: $[y] = \sum_{i=1}^{m} \alpha_i \cdot [y_i]$ and $[z] = \sum_{i=1}^{m} \alpha_i \cdot [z'_i]$.

Fig. 4. Compression protocol for reusable multiplication triples.

The compression protocol for reusable inner-product triples can be easily realized by substituting inputs with a set of reusable inner-product triples. As for matrix multiplication, we first highlight the inherent *two-dimensional reusability property*. Specifically, for a matrix triple $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, where $\mathbf{X} \in \mathbb{F}_p^{m \times n}$, $\mathbf{Y} \in$

$\mathbb{F}_p^{n\times o}$ and $\mathbf{Z} \in \mathbb{F}_p^{m\times o}$, row-wise reusability property exists in $\mathbf{X}$ and column-wise reusability in $\mathbf{Y}$. This means that any row vector in $\mathbf{X}$ forms a set of reusable inner-product triples with all column vectors in $\mathbf{Y}$, while any column vector in $\mathbf{Y}$ and all row vectors in $\mathbf{X}$ are also reusable inner-product triples.

Thus, to compress matrix triples, we extend Π_{ReComp} to Π_{ReComp2D} based on a two-stage approach. First, using the row-wise reusability, we linearly combine the m row vectors in $\mathbf{X}$ and $\mathbf{Z}$, separately, with random coefficients $\{\alpha_i\}_{i=1}^m$. This yields an n-dimensional vector $\boldsymbol{x}$ and an o-dimensional vector $\boldsymbol{z}$. Next, we exploit the column-wise reusability by linearly combining the o column vectors in $\mathbf{Y}$ and each element in $\boldsymbol{z}$ with new random coefficients $\{\beta_i\}_{i=1}^o$, resulting in an n-dimensional vector $\boldsymbol{y}$ and a scalar z. This procedure yields $(\boldsymbol{x}, \boldsymbol{y}, z)$, which satisfies the inner-product operation and thus compresses a matrix triple into a single inner-product triple. Details are illustrated in Fig. 5.

Protocol Π_{ReComp2D}

Input: $([\mathbf{X}], [\mathbf{Y}], [\mathbf{Z}])$, where $\mathbf{X} \in \mathbb{F}_p^{m\times n}$, $\mathbf{Y} \in \mathbb{F}_p^{n\times o}$ and $\mathbf{Z} \in \mathbb{F}_p^{m\times o}$
Output: $([\boldsymbol{x}], [\boldsymbol{y}], [z])$ which is a triple of n-dimensional $[\cdot]$-shared inner product
The Procedure:

1. Invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate $m + o$ random elements $\alpha_1, ..., \alpha_m$ and $\beta_1, ..., \beta_o$.
2. For $1 \le k \le n$, all parties locally compute:
   - $[\boldsymbol{x}(k)] = \sum_{i=1}^m \alpha_i \cdot [\mathbf{X}(i,k)]$ and $[\boldsymbol{y}(k)] = \sum_{j=1}^o \beta_j \cdot [\mathbf{Y}(k,j)]$
3. All parties compute $[z] = \sum_{i=1}^m (\sum_{j=1}^o \alpha_i \cdot \beta_j \cdot [\mathbf{Z}(i,j)])$.

Fig. 5. Compression protocol for matrix triples.

For the given matrix triple $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, the naive approach requires verification of mo inner-product triples of length n. However, after compression through our Π_{ReComp2D}, only a single n-dimensional inner-product triple is required, reducing the verification count to $\frac{1}{mo}$ of the naive requirement.

Security The security of Π_{ReComp} and Π_{ReComp2D} are established in Lemma 2 and Lemma 3, respectively.

Lemma 2. *If at least one multiplication triple is incorrect, then the compressed multiplication triple output by* Π_{ReComp} *is correct with probability at most* $\frac{1}{p}$.

Proof. Suppose the adversary introduces incorrect values $z_i = x_i \cdot y_i + e_i$ for all $i \in [1, m]$ and at least one $e_i \in \mathbb{F}_p$ is nonzero. Then we have

$$\begin{aligned} z_i' &= a_i^{-1} z_i - a_i^{-1} b_i y_i = a_i^{-1}(x_i \cdot y_i + e_i) - a_i^{-1} b_i y_i \\ &= a_i^{-1}(x_i - b_i) \cdot y_i + a_i^{-1} e_i = x \cdot y_i + a_i^{-1} e_i \end{aligned}$$

Next,

$$z = \sum_{i=1}^m \alpha_i \cdot z_i' = \sum_{i=1}^m \alpha_i \cdot (x \cdot y_i + a_i^{-1} e_i) = x \cdot y + \sum_{i=1}^m \alpha_i \cdot a_i^{-1} \cdot e_i.$$

Since $a_i^{-1} \neq 0$ and α_i's are randomly sampled by $\mathcal{F}_{\mathsf{Coin}}$, the probability that $z = x \cdot y$ holds is at most $\frac{1}{p}$, as established by Lemma 1.

Lemma 3. *If the matrix triple is incorrect, then the compressed inner-product triple output by* Π_{ReComp2D} *is correct with probability less than* $\frac{2}{p}$*.*

Proof. Suppose the adversary causes incorrect matrix triple $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y} + \mathbf{E}$ and $\mathbf{E} \in \mathbb{F}_p^{m \times o}$ is not a zero matrix. Then we have

$$z = \sum_{i=1}^{m} \alpha_i \cdot (\sum_{j=1}^{o} \beta_j \cdot \mathbf{Z}(i,j)) = \boldsymbol{x} \cdot \boldsymbol{y} + \sum_{i=1}^{m} \alpha_i \cdot (\sum_{j=1}^{o} \beta_j \cdot \mathbf{E}(i,j)).$$

Assume that there exists a non-zero element in the k-th row of $\mathbf{E}$. Thus, in this case, $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$ if and only if

$$\alpha_k \cdot (\sum_{j=1}^{o} \beta_j \cdot \mathbf{E}(k,j)) = - \sum_{i=1, i \neq k}^{m} \alpha_i \cdot (\sum_{j=1}^{o} \beta_j \cdot \mathbf{E}(i,j)). \tag{2}$$

Our aim is to show that Eq. (2) holds with probability $2/p$. Let $\Delta = \sum_{j=1}^{o} \beta_j \cdot \mathbf{E}(k,j)$. We have the following cases.

Case 1 ($\Delta \neq 0$). Eq. (2) holds if and only if

$$\alpha_k = \Delta^{-1} \cdot \left(- \sum_{i=1, i \neq k}^{m} \alpha_i \cdot (\sum_{j=1}^{o} \beta_j \cdot \mathbf{E}(i,j)) \right).$$

Since α_k is randomly sampled by $\mathcal{F}_{\mathsf{Coin}}$ and chosen independently of all other values, the probability that Eq. (2) holds is at most $1/p$.

Case 2 ($\Delta = 0$). In this case, the equality may hold. Nevertheless, the probability that $\Delta = 0$ is at most $1/p$, since β_j is distributed uniformly over $\mathbb{F}_p$ and not known to the adversary before introducing $\mathbf{E}(k,j)$.

In summary, the probability that Eq. (2) holds is at most $\frac{1}{p} + (1 - \frac{1}{p}) \cdot \frac{1}{p} < \frac{2}{p}$.

Usage Our compression protocols are executed in batches after the circuit evaluation step and before the multiplication verification step. As a result, the multiplication verification protocol only needs to perform batch verification on the compressed triples, thereby reducing the complexity of the verification step. Note that when using the batch strategy above, the random coefficients used in compression protocols can be reused across separate compression instances for both reusable multiplication triples and matrix triples without compromising security. Moreover, our compression protocols are *verification-method independent*, meaning that they can be integrated with any verification approach to narrow the performance gap between malicious and semi-honest protocols.

4.2 General Batch Multiplication Verification

In this subsection, we adapt the multiplication verification protocol from [17] to the machine learning setting, which enables it to handle an arbitrary number of multiplication triples and various types of triples, including $[\cdot]$-shared multiplication triples, inner-product triples, and multiplication triples with public outputs.

Normalization We replace the de-linearization protocol in [17] with a normalization protocol Π_{Norm} shown in Fig. 6. Instead of computing exponentiations, we use $\mathcal{F}_{\mathsf{Coin}}$ to generate multiple independent random elements for securely combining triples. The inner product triple $([\boldsymbol{x}^\star], [\boldsymbol{y}^\star], [z^\star])$ is integrated into our protocol by scaling $[\boldsymbol{x}^\star]$ and $[z^\star]$ with a random value $r^\star$ and appending each $r^\star[\boldsymbol{x}^\star(i)]$ to the final output triple. For multiplication triples with public outputs $([x^\zeta], [y^\zeta], z^\zeta)$, we leverage the *linearity* of Shamir secret sharing to add the public $r^\zeta z^\zeta$ to the secret-shared $[z]$, incorporating it into Π_{Norm} as well.

Protocol Π_{Norm}

Input: Let $N = m_1 + \sum_{i=1}^{m_2} n_i + m_3, N' = m_1 + m_2 + m_3$.
- m_1 pairs of $[\cdot]$-shared multiplication: $\{([x_i], [y_i], [z_i])\}_{i=1}^{m_1}$
- m_2 pairs of $[\cdot]$-shared inner product: $\{(\{[\boldsymbol{x}_i^\star(j)], [\boldsymbol{y}_i^\star(j)]\}_{j=1}^{n_i}, [z_i^\star])\}_{i=1}^{m_2}$
- m_3 pairs of $[\cdot]$-shared multiplication with public output: $\{([x_i^\zeta], [y_i^\zeta], z_i^\zeta)\}_{i=1}^{m_3}$

Output: $([\boldsymbol{a}], [\boldsymbol{b}], [c])$ which is a triple of N-dimensional $[\cdot]$-shared inner product

The Procedure:
1. All parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate N' random field elements $\{r_i\}_{i=1}^{m_1}, \{r_i^\star\}_{i=1}^{m_2}, \{r_i^\zeta\}_{i=1}^{m_3}$.
2. For $1 \leq i \leq m_1$, all parties set $[x_i] = r_i \cdot [x_i]$.
3. For $1 \leq i \leq m_2, 1 \leq j \leq n_i$, all parties set $[\boldsymbol{x}_i^\star(j)] = r_i^\star \cdot [\boldsymbol{x}_i^\star(j)]$.
4. For $1 \leq i \leq m_3$, all parties set $[x_i^\zeta] = r_i^\zeta \cdot [x_i^\zeta]$.
5. Compute $[z] = \sum_{i=1}^{m_1} r_i \cdot [z_i]$, $[z^\star] = \sum_{i=1}^{m_2} r_i^\star \cdot [z_i^\star]$ and $z^\zeta = \sum_{i=1}^{m_3} r_i^\zeta \cdot z_i^\zeta$.
6. All parties set:
   $[\boldsymbol{a}] = (..., [x_{i_1}], ..., [\boldsymbol{x}_{i_2}^\star(j)], ..., [x_{i_3}^\zeta], ...)$, $[\boldsymbol{b}] = (..., [y_{i_1}], ..., [\boldsymbol{y}_{i_2}^\star(j)], ..., [y_{i_3}^\zeta], ...)$, $[c] = [z] + [z^\star] + z^\zeta$, where $i_1 \in [1, m_1], i_2 \in [1, m_2], j \in [1, n_{i_2}], i_3 \in [1, m_3]$.

Fig. 6. Normalization protocol.

Lemma 4. *If at least one input tuple is incorrect, then the resulting inner-product triple output by Π_{Norm} is correct with probability at most $\frac{1}{p}$.*

Proof. Suppose the adversary causes incorrect values $z_i = x_i \cdot y_i + e_i, z_i^\star = \boldsymbol{x}_i^\star \cdot \boldsymbol{y}_i^\star + e_i^\star, z_i^\zeta = x_i^\zeta \cdot y_i^\zeta + e_i^\zeta$ and at least one of $e_i, e_i^\star, e_i^\zeta$ is non-zero. Then we have

$$c = \sum_{i=1}^{m_1} r_i \cdot z_i + \sum_{i=1}^{m_2} r_i^\star \cdot z_i^\star + \sum_{i=1}^{m_3} r_i^\zeta \cdot z_i^\zeta = \boldsymbol{a} \cdot \boldsymbol{b} + \sum_{i=1}^{m_1} r_i e_i + \sum_{i=1}^{m_2} r_i^\star e_i^\star + \sum_{i=1}^{m_3} r_i^\zeta e_i^\zeta.$$

Since the three $\sum$ have a symmetric structure, we can uniformly represent them as $\sum_{i=1}^{N'} r_i \cdot e_i$. Therefore, in this case, $c = \boldsymbol{a} \cdot \boldsymbol{b}$ if and only if

$$\sum_{i=1}^{N'} r_i \cdot e_i = 0. \tag{3}$$

Since at least one e_i is non-zero and the r_i values are distributed uniformly over $\mathbb{F}_p$, the probability that Eq. (3) holds is at most $\frac{1}{p}$ by Lemma 1.

General Dimension-Reduction When $N = m_1\tau + m_2$ with m_2 less than the compression parameter τ, a simple approach for dimension reduction is as follows. First, divide the N triples into $\tau + 1$ groups, where the first τ groups are m_1-dimensional inner-product triples and the last is m_2-dimensional. Then, compress the first τ groups to one m_1-dimensional inner-product triple according to the Extend-Compress protocol in [17]. Finally, combine the compressed triple with the last m_2-dimensional triple using uniformly random numbers, as in Π_{Norm}, resulting in an $(m_1 + m_2)$-dimensional inner-product triple.

To further reduce communication rounds, we implement the following optimizations.

(1) Polynomial calculations, $\boldsymbol{f}(\cdot)$ and $\boldsymbol{g}(\cdot)$, in Extend-Compress are independent of the inner product $\mathcal{F}^{\mathsf{sh}}_{\mathsf{InnerProd}}$ invoked in the dimension-reduction procedure. Thus, two invocations of $\mathcal{F}^{\mathsf{sh}}_{\mathsf{InnerProd}}$ can be executed in parallel, thereby reducing one communication round.

(2) When combining the m_1- and m_2-dimensional triples, the generation of the random number can be merged into the last step of the Extend-Compress protocol.

Consequently, based on rigorous security analysis, we construct a generalized version, $\Pi_{\mathsf{DimReduce}}$, without adding communication rounds. Details of $\Pi_{\mathsf{DimReduce}}$ are provided in Fig. 7.

Lemma 5. *If the input inner-product tuple is incorrect, then the resulting inner-product tuple output by* $\Pi_{\mathsf{DimReduce}}$ *is correct with probability at most* $\frac{2\tau-1}{p}$.

Proof. According to Lemma 9 in [17], at least one tuple among the $\tau + 1$ inner-product tuples, $\{([\boldsymbol{a}_i], [\boldsymbol{b}_i], [c_i])\}_{i=1}^{\tau}, ([\boldsymbol{u}], [\boldsymbol{v}], [w])$, is incorrect. First, if any of the first τ inner-product tuples are incorrect, then by Lemma 7 in [17], the tuple $([\boldsymbol{a}'], [\boldsymbol{b}'], [c'])$ is also incorrect with probability $1 - \frac{2\tau-2}{p}$. Second, if the first τ tuples are all correct, it must be that $\boldsymbol{u} \cdot \boldsymbol{v} \neq w$. Thus, with probability $1 - \frac{2\tau-2}{p}$, either $([\boldsymbol{a}'], [\boldsymbol{b}'], [c'])$ or $([\boldsymbol{u}], [\boldsymbol{v}], [w])$ is incorrect.

Suppose the adversary induces incorrect values $c' = \boldsymbol{a}' \cdot \boldsymbol{b}' + e_1$ and $w = \boldsymbol{u} \cdot \boldsymbol{v} + e_2$, with at least one $e_i \in \mathbb{F}_p$ not equal to zero. Then we have $c = \boldsymbol{a} \cdot \boldsymbol{b} + r' \cdot e_1 + e_2$. In this case, $c = \boldsymbol{a} \cdot \boldsymbol{b}$ if and only if $r' \cdot e_1 + e_2 = 0$. Since r' is randomly sampled by $\mathcal{F}_{\mathsf{Coin}}$ and chosen independently of e_1 and e_2, the probability of this condition holding is at most $\frac{1}{p}$.

In summary, the probability that the resulting inner-product tuple is correct is at most $\frac{2\tau-2}{p} + (1 - \frac{2\tau-2}{p}) \cdot \frac{1}{p} < \frac{2\tau-1}{p}$.

Protocol $\Pi_{\mathsf{DimReduce}}$

Input: A N-dimensional $[\cdot]$-shared inner product $([\boldsymbol{x}], [\boldsymbol{y}], [z])$. Let τ denote the compression parameter. Let $N = m_1\tau + m_2$, where $m_2 < \tau$.
Output: A $(m_1 + m_2)$-dimensional $[\cdot]$-shared inner product $([\boldsymbol{a}], [\boldsymbol{b}], [c])$.
The Procedure:

1. All parties interpret $[\boldsymbol{x}], [\boldsymbol{y}]$ as:
   $[\boldsymbol{x}] = ([\boldsymbol{a}_1], [\boldsymbol{a}_2], ..., [\boldsymbol{a}_\tau], [\boldsymbol{u}])$
   $[\boldsymbol{y}] = ([\boldsymbol{b}_1], [\boldsymbol{b}_2], ..., [\boldsymbol{b}_\tau], [\boldsymbol{v}])$,
   where $\{[\boldsymbol{a}_i], [\boldsymbol{b}_i]\}_{i=1}^{\tau}$ are vectors of dimension m_1 and $[\boldsymbol{u}], [\boldsymbol{v}]$ are vectors of dimension m_2.
2. Locally compute $[\boldsymbol{f}(\cdot)]$ and $[\boldsymbol{g}(\cdot)]$ by using $\{[\boldsymbol{a}_i]\}_{i=1}^{\tau}$ and $\{[\boldsymbol{b}_i]\}_{i=1}^{\tau}$ respectively. $\boldsymbol{f}(\cdot)$ and $\boldsymbol{g}(\cdot)$ are vectors of degree-$(\tau - 1)$ polynomials such that $\forall i \in [1, \tau], \boldsymbol{f}(i) = \boldsymbol{a}_i, \boldsymbol{g}(i) = \boldsymbol{b}_i$.
3. For $i \in [\tau + 1, 2\tau - 1]$, all parties locally compute $[\boldsymbol{a}_i] = [\boldsymbol{f}(i)], [\boldsymbol{b}_i] = [\boldsymbol{g}(i)]$.
4. For $i \in [1, 2\tau - 1]$, all parties invoke $\mathcal{F}_{\mathsf{InnerProd}}^{\mathsf{sh}}$ on $([\boldsymbol{a}_i], [\boldsymbol{b}_i])$ to compute $[c_i]$ and set $[w] = [z] - \sum_{i=1}^{\tau}[c_i]$.
5. Locally compute $[h(\cdot)]$ by using $\{[c_i]\}_{i=1}^{2\tau-1}$.
6. Invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate two random field elements r and r'.
7. Compute $[\boldsymbol{a}'] = [\boldsymbol{f}(r)], [\boldsymbol{b}'] = [\boldsymbol{g}(r)], [c'] = [h(r)]$.
8. All parties output $[\boldsymbol{a}] = (r' \cdot [\boldsymbol{a}'], [\boldsymbol{u}])$, $[\boldsymbol{b}] = ([\boldsymbol{b}'], [\boldsymbol{v}])$ and $[c] = r' \cdot [c'] + [w]$.

Fig. 7. General dimension reduction protocol.

Multiplication Verification with Sub-linear Communication We construct a general batch multiplication verification protocol $\Pi_{\mathsf{MultVerify}}$ by combining normalization, general dimension reduction, and randomization processes, following the approach of [17]. $\Pi_{\mathsf{MultVerify}}$ achieves sub-linear communication complexity, and its security can be easily derived from the results in [17].

4.3 Batch Checking for Revealed Degree-t Sharings

To reduce the computational complexity of correctness checks in degree-t revealing, we defer the check procedure to be completed in batches and propose Π_{Reveal} in Fig. 8. In the online phase, only revealing under semi-honest security is required. In the verification phase, by using random linear combinations, the correctness of all revealed values can be checked in batch through the Lagrange interpolation method. The computational complexity in our Π_{Reveal} is $O(m+t^2)$, which is more efficient compared to the original method with a complexity of $O(mt^2)$.

Note that after revealing degree-t or degree-$2t$ sharing, view comparisons are essential. Otherwise, a malicious adversary can launch a differential attack during the next degree-t reveal. We now analyze this potential attack, similar to the one in [20]. Consider a circuit that initially contains a revealing operation with input $[a]$ or $[\![a]\!]$. Then, a is used in a local linear operation with other secret values, for example, computing $[c] = a \cdot [b] + [r]$. Finally, the degree-t sharing $[c]$ is revealed. Suppose P_{king} is controlled by a malicious adversary and chooses to

Protocol Π_{Reveal}

Input: $[x_1], [x_2], ..., [x_m]$
Online:
1. Reveal $x'_i \leftarrow \Pi^{\mathsf{sh}}_{\mathsf{Reveal}}([x_i])$.
2. Compare the hash of the view†. If inconsistent, then abort.

Verification:
1. Invoke $\mathcal{F}_{\mathsf{Coin}}$ to generate m random elements $\alpha_1, ..., \alpha_m$.
2. The parties locally compute: $[y] = \sum_{i=1}^{m} \alpha_i \cdot [x_i]$ and $y' = \sum_{i=1}^{m} \alpha_i \cdot x'_i$. Let $[z] = [y] - y'$.
3. Reveal $z \leftarrow \Pi_{\mathsf{Reveal\text{-}Check}}([z])$ with correctness check.
4. If z is not equal to 0 then abort, otherwise continue.

† Each party stores a string of its view on all broadcasted values up to now [23].

Fig. 8. Batch checking for revealed degree-t sharings.

send two different values, such as a and $a+e$, where e is known to the adversary, to two honest parties (say P_1 and P_2). Following this, each party performs the local linear operations with $a+e$ computed as $[c]+e\cdot[b]$. Consequently, there are two different secret shares within the network: P_1 holds a share of $[c]$, while P_2 holds a share of $[c]+e\cdot[b]$. In the final phase, both P_1 and P_2 send their shares to P_{king}. Since only $t+1$ shares are needed to reconstruct a degree-t sharing, a malicious adversary controlling t parties can simultaneously reconstruct c and $c+e\cdot b$. Thus, by calculating the difference, the adversary can learn b in clear.

Especially, when only $2t$ parties send their shares to P_{king} for revealing degree-$2t$ sharings, which is a common approach for optimized performance, an opportunity for optimization arises. Specifically, since revealing a degree-$2t$ sharing requires $2t+1$ shares, P_{king} needs at least $2t+2$ different shares to recover two distinct values, which makes differential attacks infeasible for degree-$2t$ revealing. Therefore, the view comparison process can be executed before the degree-t revealing (and also before $\Pi_{\mathsf{Reveal\text{-}Check}}$). For degree-$2t$ revealing, the comparison can be performed after the circuit evaluation step and before the verification step, as proposed in [23]. This strategy optimizes both the communication rounds and overhead.

We provide the correctness of Π_{Reveal} in Lemma 6.

Lemma 6. *If at least one of the public values x'_i does not equal x_i, then the probability that the check phase passes is at most $\frac{1}{p}$.*

Proof. The proof relies on the security of degree-t reconstruction with correctness check, and the probability analysis is similar to that in Lemma 2. Therefore, for simplicity, we omit the details.

5 Applications to Machine Learning

In this section, we present our construction of maliciously secure machine learning building blocks, including matrix multiplication, truncation, and ReLU.

5.1 Matrix Multiplication

Under Shamir secret sharing, matrix multiplication can be easily implemented using vector inner product. In the malicious security setting, we can further leverage Π_{ReComp2D} and $\Pi_{\mathsf{MultVerify}}$ to achieve efficient correctness verification, which yields Π_{MatMult}.

5.2 Truncation

We extend the fixed-point multiplication protocol $\Pi^{\mathsf{sh}}_{\mathsf{Fixed\text{-}Mult}}$ in LXY24 to against malicious adversaries. First, we adapt the semi-honest secure protocol $\Pi^{\mathsf{sh}}_{\mathsf{SolvedBits}}$ for malicious security by incorporating maliciously secure $\mathcal{F}_{\mathsf{Rand}}$ and $\Pi_{\mathsf{MultVerify}}$. Next, we extend $\Pi_{\mathsf{Trunc\text{-}Triple}}$ to $\Pi_{\mathsf{Trunc\text{-}Quadruple}}$ to generate the truncation quadruple $([r'], [\![r]\!], [r], [r_{\mathrm{msb}}])$, which can be easily achieved since $[r]$ is already generated by $\Pi_{\mathsf{SolvedBits}}$ in $\Pi_{\mathsf{Trunc\text{-}Triple}}$. Finally, based on $\Pi_{\mathsf{Trunc\text{-}Quadruple}}$, we can construct $\Pi_{\mathsf{Fixed\text{-}Mult}}$ by verifying the correctness of the multiplication triple $([a], [b], c - 2^{\ell-2} - [r])$ using $\Pi_{\mathsf{MultVerify}}$.

5.3 ReLU

We first leverage $\mathcal{F}_{\mathsf{DoubleRand}}$ and $\Pi_{\mathsf{MultVerify}}$ to obtain maliciously secure protocols $\Pi_{\mathsf{2L\text{-}DN}}$, $\Pi_{\mathsf{3L\text{-}DN}}$, Π_{PreMul3} and Π_{PreMul4}. Subsequently, based on these subprotocols, Π_{PreOR} can be implemented with malicious security. Ultimately, we can realize Π_{ReLU} using Π_{PreOR}, $\Pi_{\mathsf{2L\text{-}DN}}$ and Π_{Reveal}, following the method described in LXY24.

To further improve efficiency, we utilize the protocol Π_{ReComp} before $\Pi_{\mathsf{MultVerify}}$ in our prefix multiplication protocol Π_{PreMult}. As shown in § 3.4, the first-round computation utilizes Π_{PreMul4}, where triples $(a_3, -r, u_1 - r_1)$ and $(a_3, a_4, u_2 - r_2)$ constitute reusable multiplication triples. Starting from the second round, the inputs to m consecutive ○, ◈, and ■ operators can be expressed as $\{x, \{y_i\}_{i=1}^m\}$, $\{x, y, \{z_i\}_{i=1}^m\}$, and $\{x, y, z, \{w_i\}_{i=1}^m\}$, respectively. Since ○ operators are implemented through DN protocol, the set $\{(x, y_i, xy_i)\}_{i=1}^m$ forms reusable multiplication triples with degree m. Similarly, ◈ operators realized through $\Pi_{\mathsf{2L\text{-}DN}}$, yield reusable triples $\{(r, -z_i, u_i - r_i)\}_{i=1}^m$ in step 1 of $\Pi_{\mathsf{2L\text{-}DN}}$. For ■ operators implemented via $\Pi^{\mathsf{sh}}_{\mathsf{3L\text{-}DN}}$, the set $\{(z, w_i, u_i - r_i)\}_{i=1}^m$ in step 1 of $\Pi_{\mathsf{3L\text{-}DN}}$ also constitutes reusable multiplication triples. Consequently, after compression by Π_{ReComp}, the complexity of the number of triples to be verified can be reduced from $O(\ell \log_4 \ell)$ to $O(\ell)$. In particular, the 31-bit (resp. 61-bit) protocol requires verification of only 16 (resp. 55) triples, indicating a reduction in the number of verified triples to 24% (resp. 38%) of the original 66 (resp. 145) triples.

6 Evaluation

6.1 Evaluation Setup

We implement Helix in C++[8] using the open-source framework hmmpc-public [24]. Consistent with LXY24, our evaluation focuses on scenarios involving 3PC, 7PC, 11PC, 21PC, 31PC, and 63PC. The experiments are conducted on Aliyun ECS using 11 ecs.c6a.8xlarge machines, each equipped with 32 vCPUs and 64 GB of RAM. We consider two network settings: LAN and WAN. These are simulated using the Linux `tc` command, with a bandwidth of 10Gbps (100Mbps) in LAN (WAN), and an average round-trip time of approximately 0.3ms (40ms) in LAN (WAN).

In our implementation, we utilize the Mersenne prime $p = 2^{61} - 1$ for Shamir secret sharing scheme to ensure 40-bit statistical security in the malicious setting. For fixed-point multiplication, the number of fractional bits is set to 13. In terms of the Eigen library [18], we allocate 8 cores to each party to accelerate matrix multiplications.

We assess the secure multi-party inference capabilities of Helix and compare its performance against two baselines: i) the semi-honest scheme LXY24 with the substitution of our secure prefix OR subprotocol. ii) a naive maliciously secure scheme, refer to as "LXY24+", that does not employ the optimized methods described in § 4. We evaluate 3 standard neural networks on the MNIST dataset [22]: a 3-layer DNN derived from SecureML [29] (Network-A), a 3-layer CNN derived from Chameleon [33] (Network-B), and a 4-layer CNN derived from MiniONN [25] (Network-C).

6.2 Evaluation on Neural Network Inference

The Number of Verified Multiplication Triples We first discuss the reduction in the number of multiplication triples to be verified achieved by Helix compared to LXY24+. The experimental results are presented in Table 2. These results illustrate that, after applying our compression protocols in § 4.1, the number of verified triples required in the online phase can be reduced to 9.9% for Network A, 23.4% for Network B, and 26.2% for Network C. This significant reduction in the problem scale of multiplication verification highlights the effectiveness of our compression protocols.

Running Time in the LAN Setting In Table 3, we evaluate the online and preprocessing time for NN inference in the LAN setting. On average, our maliciously secure Helix is 2.5× slower in the online phase and 1.5× slower in preprocessing compared to the semi-honest secure LXY24. Notably, for 63 parties, our protocol achieves an online (preprocessing) performance of 0.736 (5.512) seconds for Network-C, which is 1.9× (1.2×) slower than LXY24. To the best of

[8] The implementation will be open-sourced on GitHub upon publication.

Table 2. Comparing the number of verified triples between LXY24+ and Helix.

Protocol	Network-A		Network-B		Network-C	
	Online	Preprocessing	Online	Preprocessing	Online	Preprocessing
LXY24+	157184	43874	268160	183130	2248620	1748070
Helix	15632	36962	62644	153970	590161	1468890
Factor	9.9%	84.2%	23.4%	84.1%	26.2%	84%

Table 3. Online and Preprocessing time for NN inference in the LAN setting. All numbers are reported in seconds. The compression parameter τ in the multiplication verification protocol is 4.

Stage	#PC	Network-A			Network-B			Network-C		
		LXY24	LXY24+	Helix	LXY24	LXY24+	Helix	LXY24	LXY24+	Helix
Online	3PC	0.009	0.055	0.022	0.017	0.092	0.04	0.121	0.693	0.262
	7PC	0.01	0.057	0.026	0.018	0.09	0.044	0.126	0.652	0.268
	11PC	0.011	0.061	0.029	0.019	0.091	0.048	0.13	0.638	0.283
	21PC	0.014	0.075	0.043	0.023	0.104	0.065	0.167	0.708	0.345
	31PC	0.02	0.094	0.062	0.029	0.135	0.086	0.212	0.824	0.43
	63PC	0.048	0.199	0.152	0.062	0.278	0.18	0.392	1.403	0.736
Prep.	3PC	0.023	0.048	0.041	0.096	0.169	0.155	0.924	1.529	1.347
	7PC	0.031	0.059	0.053	0.134	0.204	0.187	1.262	1.86	1.775
	11PC	0.041	0.07	0.065	0.167	0.242	0.23	1.662	2.244	2.158
	21PC	0.056	0.098	0.094	0.231	0.319	0.314	2.219	2.879	2.786
	31PC	0.066	0.127	0.122	0.265	0.375	0.373	2.471	3.141	3.16
	63PC	0.179	0.331	0.315	0.508	0.699	0.704	4.431	5.468	5.512

our knowledge, this is the first maliciously secure PPML protocol to support 63 parties.

Compared to LXY24+, our protocol achieves a 1.3× to 2.6× improvement in online inference time, with the improvement positively correlated with the parameter size of neural networks. This is because larger network parameters result in more multiplication triples requiring verification, increasing the computational complexity when using the verification protocol in [17]. Consequently, the benefits of our compression protocols become more pronounced as the parameter size grows. Moreover, for preprocessing running time, the improvement is less significant, as the compression factor stabilizes at approximately 84% (compared to a 10% to 30% improvement in the online phase) according to Table 2.

Running Time in the WAN Setting Table 4 presents the online and preprocessing time of the three protocols in the WAN setting. The online (preprocessing) running time of Helix is, on average, 1.5× (1.6×) that of LXY24. Compared to LXY24+, Helix is slightly faster in both phases, though the difference is less significant than in the LAN setting.

Table 4. Online and preprocessing time for NN inference in the WAN setting. All numbers are reported in seconds. The compression parameter τ in the multiplication verification protocol is 32.

Stage	#PC	Network-A			Network-B			Network-C		
		LXY24	LXY24+	Helix	LXY24	LXY24+	Helix	LXY24	LXY24+	Helix
Online	3PC	0.563	1.147	1.013	0.69	1.305	1.233	3.348	4.684	4.063
	7PC	0.576	1.164	1.029	0.799	1.409	1.339	4.72	5.665	5.352
	11PC	0.582	1.171	1.037	0.765	1.38	1.313	5.038	6.057	5.544
	21PC	0.636	1.24	1.102	0.985	1.735	1.627	7.284	8.585	7.704
	31PC	0.697	1.323	1.18	1.273	2.051	1.932	9.547	11.267	10.004
	63PC	0.961	1.928	1.826	3.529	4.48	4.451	19.512	24.031	21.624
Prep.	3PC	0.386	1.037	0.813	0.756	1.549	1.427	6.365	7.726	7.396
	7PC	0.355	1.019	0.795	0.913	1.642	1.503	7.999	9.023	9.477
	11PC	0.39	1.054	0.831	0.901	1.675	1.508	7.97	9.706	9.236
	21PC	0.427	1.124	0.897	1.399	2.294	2.13	12.752	13.517	13.281
	31PC	0.518	1.242	1.013	1.754	2.906	2.763	16.757	19.1	18.742
	63PC	1.66	2.403	2.267	4.252	6.47	6.533	30.302	33.483	32.322

Note that our compression protocols combined with the verification protocol in [17], primarily reduce computational complexity, with limited impact on communication costs. However, in the WAN setting, communication complexity becomes the dominant factor affecting inference performance rather than computational cost. As a result, the performance of Helix and LXY24+ is similar in this scenario. Furthermore, thanks to the sublinear communication of [17], the communication overheads of the maliciously secure Helix and LXY24+ are comparable to those of the semi-honest LXY24. Overall, the three protocols exhibit closer performance in the WAN setting than in the LAN setting.

Communication We report only the communication overhead of Helix with $\tau = 4$ in Table 5, as the three protocols demonstrate similar communication costs. The values in brackets in Table 5 roughly represent the total communication overhead introduced by the verification step in the malicious setting. Specifically, for Network-C with 63PC, the communication overhead of the verification step is only 0.057 MB (1.927 MB) in the online (preprocessing) phase, accounting for just 0.02% (3.6%) of the total communication overhead of Helix.

7 Related Works

For comparison protocols over fields in the n-party setting, Catrina and de Hoogh [6] achieve both small constant rounds and low communication costs by assuming a large gap between secrets and shares. Damgård *et al.* [11] first study the comparison without the gap requirement, and propose a constant-round prefix OR protocol that is the critical component for realizing gap-free

Table 5. Online and preprocessing communication cost per party of Helix for NN inference. All numbers are reported in MB. The compression parameter τ in the multiplication verification protocol is 4. Increments compared to LXY24 are given in brackets.

#PC	Network-A		Network-B		Network-C	
	Online	Prep.	Online	Prep.	Online	Prep.
3PC	0.438(0.002)	0.913(0.024)	1.842(0.002)	3.814(0.093)	17.629(0.003)	36.41(0.867)
7PC	0.565(0.004)	1.176(0.04)	2.371(0.005)	4.906(0.155)	22.664(0.007)	46.815(1.435)
11PC	0.602(0.007)	1.25(0.048)	2.517(0.007)	5.206(0.176)	24.039(0.01)	49.655(1.616)
21PC	0.636(0.013)	1.315(0.057)	2.643(0.014)	5.459(0.198)	25.191(0.019)	52.026(1.78)
31PC	0.652(0.019)	1.342(0.065)	2.692(0.02)	5.554(0.211)	25.606(0.028)	52.871(1.843)
63PC	0.682(0.038)	1.383(0.085)	2.757(0.04)	5.668(0.238)	26.067(0.057)	53.788(1.927)

comparison. Nishide and Ohta [30] improve the efficiency of the protocols in [11] without using the bit-decomposition protocol. However, the comparison protocols in [11, 30] both incur impractically high communication costs. The most recent work, Rabbit [27] leverages the OR-version of PreOpL protocol in [34] to bring communication costs to a practical level while depending on logarithmic-round complexity. As a result, for n-party comparison protocols over fields with no gap requirement, existing works exhibit a trade-off between rounds and communication costs, primarily due to the complexity of prefix OR operations.

In the malicious and honest-majority setting, the primary challenge lies in verifying the correctness of multiplications. Lindell and Nof [23] propose a verification protocol using the triple sacrifice technique [14], which offers low computational complexity while requiring communication linear to the number of multiplication gates. Boneh *et al.* [3] first employ distributed zero-knowledge proofs to achieve sublinear communication at the expense of higher computational complexity. Building on this work, Boyle *et al.* [4] subsequently develop a practical verification protocol based on 3PC replicated secret sharing. Goyal and Song [17] extend the polynomial interpolation-based compression protocol in [31], designing a verification protocol that also achieves sublinear communication while improving practical performance.

In the context of MPC-based PPML, existing works that support malicious security in the honest-majority setting, mainly focus on 3-4 parties with one corrupt party. ABY3 [28], BLAZE [32] and Falcon [36] employ 3PC replicated secret sharing (RSS) to provide ML inference with malicious security. Works [7,9, 19,21] aim to further enhance efficiency in a 4PC setting. More recently, MPClan [20] leverages the RSS scheme to provide maliciously secure ML inference with up to 9 parties. However, RSS-based approaches incur exponential storage overhead, so it is infeasible to scalable to a large number of parties, e.g., 63, as us.

8 Conclusion

In this paper, we propose a scalable framework, Helix, for multi-party machine learning inference with malicious security in the honest majority setting. We design a series of novel protocols to narrow the performance gap between maliciously and semi-honestly secure protocols. Experimental results demonstrate the practicality and scalability of Helix by implementing NN inference with up to 63 parties. Future work could explore migrating our constructions to other maliciously secure frameworks and enhancing efficiency with GPUs.

References

1. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing. pp. 201–209 (1989)
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference. pp. 420–432. Springer (1992)
3. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear pcps. In: Annual International Cryptology Conference. pp. 67–97. Springer (2019)
4. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 869–886 (2019)
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
6. Catrina, O., De Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Security and Cryptography for Networks. pp. 182–199. Springer (2010)
7. Chaudhari, H., Rachuri, R., Suresh, A.: Trident: Efficient 4pc framework for privacy preserving machine learning. In: Network and Distributed System Security Symposium (2020)
8. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD$\mathbb{Z}_{2^k}$: efficient MPC mod for dishonest majority. In: Annual International Cryptology Conference. pp. 769–798. Springer (2018)
9. Dalskov, A., Escudero, D., Keller, M.: Fantastic four: Honest-majority four-party secure computation with malicious security. In: 30th USENIX Security Symposium. pp. 2183–2200 (2021)
10. Damgård, I., Escudero, D., Frederiksen, T., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure mpc over rings with applications to private machine learning. In: 2021 IEEE Symposium on Security and Privacy. pp. 1102–1120. IEEE (2019)
11. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Theory of Cryptography Conference. pp. 285–304. Springer (2006)
12. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In: European Symposium on Research in Computer Security. pp. 1–18. Springer (2013)

13. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Annual International Cryptology Conference. pp. 572–590. Springer (2007)
14. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Annual International Cryptology Conference. pp. 643–662. Springer (2012)
15. Dong, Y., Chen, X., Jing, W., Li, K., Wang, W.: Meteor: improved secure 3-party neural network inference with reducing online communication costs. In: Proceedings of the ACM Web Conference. pp. 2087–2098 (2023)
16. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: Atlas: efficient and scalable mpc in the honest majority setting. In: Annual International Cryptology Conference. pp. 244–274. Springer (2021)
17. Goyal, V., Song, Y.: Malicious security comes free in honest-majority mpc. Cryptology ePrint Archive (2020)
18. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010), `http://eigen.tuxfamily.org`
19. Koti, N., Pancholi, M., Patra, A., Suresh, A.: SWIFT: Super-fast and robust privacy-preserving machine learning. In: 30th USENIX Security Symposium. pp. 2651–2668 (2021)
20. Koti, N., Patil, S., Patra, A., Suresh, A.: Mpclan: Protocol suite for privacy-conscious computations. Journal of Cryptology **36**(3), 22 (2023)
21. Koti, N., Patra, A., Rachuri, R., Suresh, A.: Tetrad: Actively secure 4pc for secure training and inference. In: Network and Distributed System Security Symposium (2022)
22. LeCun, Y., Cortes, C., Burges, C.J.: Mnist handwritten digit database (2010), `http://yann.lecun.com/exdb/mnist`, Accessed: 2010
23. Lindell, Y., Nof, A.: A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 259–276 (2017)
24. Liu, F., Xie, X., Yu, Y.: Scalable multi-party computation protocols for machine learning in the honest-majority setting. In: 33th USENIX Security Symposium (2024)
25. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. pp. 619–631 (2017)
26. Lu, T., Zhang, B., Li, L., Ren, K.: Aegis: A lightning fast privacy-preserving machine learning platform against malicious adversaries. Cryptology ePrint Archive (2023)
27. Makri, E., Rotaru, D., Vercauteren, F., Wagh, S.: Rabbit: Efficient comparison for secure multi-party computation. In: International Conference on Financial Cryptography and Data Security. pp. 249–270. Springer (2021)
28. Mohassel, P., Rindal, P.: ABY3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 35–52 (2018)
29. Mohassel, P., Zhang, Y.: SecureML: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy. pp. 19–38. IEEE (2017)
30. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: 10th International Conference on Practice and Theory in Public Key Cryptography. pp. 343–360. Springer (2007)

31. Nordholt, P.S., Veeningen, M.: Minimising communication in honest-majority mpc by batchwise multiplication verification. In: International Conference on Applied Cryptography and Network Security. pp. 321–339. Springer (2018)
32. Patra, A., Suresh, A.: BLAZE: blazing fast privacy-preserving machine learning. In: Network and Distributed System Security Symposium (2020)
33. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia conference on computer and communications security. pp. 707–721 (2018)
34. SecureSCM: Deliverable D9.2, EUFP7 Project Secure Supply Chain Management. Security analysis (2009)
35. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
36. Wagh, S., Gupta, D., Chandran, N.: Falcon: Honest-majority maliciously secure framework for private deep learning. Proceedings on Privacy Enhancing Technologies (2021)
37. Watson, J.L., Wagh, S., Popa, R.A.: Piranha: A GPU platform for secure computation. In: 31st USENIX Security Symposium. pp. 827–844 (2022)
38. Yao, A.C.: Protocols for secure computations. In: Annual Symposium on Foundations of Computer Science. pp. 160–164. IEEE (1982)
39. Yuan, B., Yang, S., Zhang, Y., Ding, N., Gu, D., Sun, S.F.: {MD-ML}: Super fast {Privacy-Preserving} machine learning for malicious security with a dishonest majority. In: 33rd USENIX Security Symposium. pp. 2227–2244 (2024)