


Article

A Study of Privacy-Preserving Neural Network Prediction Based on Replicated Secret Sharing

Yanru Zhang * and Peng Li 

Department of Mathematics and Physics, North China Electric Power University, Baoding 071003, China

* Correspondence: zyrlycoris@163.com

Abstract: Neural networks have a wide range of promise for image prediction, but in the current setting of neural networks as a service, the data privacy of the parties involved in prediction raises concerns. In this paper, we design and implement a privacy-preserving neural network prediction model in the three-party secure computation framework over secret sharing of private data. Secret sharing allows the original data to be split, with each share held by a different party. The parties cannot know the shares owned by the remaining collaborators, and thus the original data can be kept secure. The three parties refer to the client, the service provider and the third server that assist in the computation, which is different from the previous work. Thus, under the definition of semi-honest and malicious security, we design new computation protocols for the building blocks of the neural network based on replicated secret sharing. Experimenting with MNIST dataset on different neural network architectures, our scheme improves $1.3\times/1.5\times$ and $7.4\times/47.6\times$ in terms of computation time as well as communication cost compared to the Falcon framework under the semi-honest/malicious security, respectively.

Keywords: neural network; privacy-perserving; three-party secure computation; replicated secret sharing

MSC: 68P27; 68T07



Citation: Zhang, Y.; Li, P. A Study of Privacy-Preserving Neural Network Prediction Based on Replicated Secret Sharing. *Mathematics* **2023**, *11*, 1048. <https://doi.org/10.3390/math11041048>

Academic Editor: Angel Martín-del-Rey

Received: 15 January 2023
Revised: 13 February 2023
Accepted: 14 February 2023
Published: 19 February 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Machine learning is now widely used in computer science research and practical industrial applications, generating huge social impact and economic benefits. In particular, convolutional neural networks have great potential in the field of image processing, such as face recognition [1–3] and speech recognition [4]. Due to their wide applicability, service providers are starting to rent neural networks as a commodity for their services. In this scenario, the model is pre-trained by the service provider and the client simply uploads their own private data to obtain the desired output.

With the advent of the digital age, massive data infrastructures are being built and sensitive data is being generated. The end-to-end communication between the client and the service provider can create security vulnerabilities, allowing the leakage of the private data owned by the client as well as model parameters trained by the service provider at great human and material cost [5]. Therefore, there is a need to protect the privacy of parties during the inference process, and neural networks that supports secure computation become a hot topic of current research.

Secure multi-party computation (SMC) offers a promising solution for privacy-preserving neural network, which originates from the Millionaire’s Problem proposed by Yao in 1982 [6]. In the SMC framework, collaborating parties do not reveal their private data to each other. Depending on the number of members involved in the protocol, current research is divided into (1) two-party secure computation (2PC) [7–10]; (2) three-party secure computation (3PC) [11–15]; and (3) four-party secure computation (4PC) [16,17].

SecureML [18] is based on a two-server model and relies on techniques such as oblivious transfer [19] and garbled circuits to support secure computation of shared decimal numbers, implementing protocols for various machine learning algorithms such as linear regression, logistic regression and neural networks. A hybrid protocol framework is proposed in [20] that effectively combines arithmetic sharing, boolean sharing and Yao's garbled circuits to provide a solution for 2PC problem, which is known as ABY. Chandran et al. [21] build a compiler based on ABY to convert easy-to-write and high-level programs into efficient 2PC protocols using boolean or arithmetic circuits for better performance. Agrawal et al. [22] combine boolean sharing and arithmetic sharing to propose a ternary matrix-vector multiplication protocol with correlated oblivious transfer, and design a new fixed-point optimisation algorithm through a floating-point adaptive gradient optimisation procedure to improve prediction efficiency.

Homomorphic encryption is also an effective mean of protecting privacy, which is often used in 2PC. The first to use homomorphic encryption for secure inference is the CryptoNet [23] framework, where the user encrypts his private data and uploads it to the cloud server, which performs the corresponding computation directly on the ciphertext and returns the resulting ciphertext. The data exists in ciphertext form, which protects the user's privacy. However, in this framework, the non-linear layer needs to be calculated by approximation, e.g., the activation function is replaced by a square function, and therefore the accuracy of the model may be reduced due to the difference between the approximation function and the original function. Researches [24–28] based on this are very computationally expensive and have high latency due to the limitations of homomorphic encryption itself, so further improvements are still needed, e.g., in combination with secret sharing or garbled circuits. Gazelle [29] combines garbled circuits with homomorphic encryption to achieve homomorphic linear algebraic kernel which maps neural network layers to optimized homomorphic matrix multiplication and convolution computation, improving the inference time of neural networks. Delphi [30] builds on Gazelle and proposes a new planner that automatically generates neural network architecture configurations to navigate the trade-off between performance and accuracy to further improve performance.

Araki et al. [31] propose a 3PC framework for boolean circuits with honest majorities, which is secure in the presence of semi-honest adversaries. Furukawa et al. [32] build on this by constructing beaver multiplication triples [33] to verify the correctness of the computation, making the protocol secure in the presence of malicious adversaries at most one party. Mohassel et al. propose a three-server model, namely the ABY3 [34], which generalises and optimises the conversion between arithmetic sharing, binary sharing and Yao's garbled circuits on the basis of ABY, and propose new techniques for fixed-point multiplication with shared decimal values. Chameleon [35] is a hybrid protocol framework that utilises additive secret sharing, garbled circuits and the Goldreich-Micali-Wigderson (GMW) protocol [36] to improve performance, and introduces a semi-honest third party to design a more efficient oblivious transfer protocol for arithmetic triples. SecureNN [37] studies the computation of forward prediction and backpropagation training for deep learning models in a three-party computation environment, using only arithmetic addition and secret sharing, eliminating expensive cryptographic operations and guaranteeing semi-honest security. Malicious security is also presented in that paper, but the computation of its protocol is not accurate in this security setting. Falcon [38] proposes an end-to-end 3PC protocol for neural networks training and inference based on the ideas of SecureNN and ABY3, improving performance and ensuring semi-honest as well as malicious security, which is the main reference of this paper.

Trident [39] is a privacy-preserving machine learning framework that implements a four-server model. The fourth party in the protocol is inactive in the online phase except for input sharing and output reconstruction. Flash [40] guarantees output delivery security (all parties get the output regardless of the behaviour of the adversary), and optimises the dot product protocol to make its computation independent of vector size as well as truncation and highest significant bit extraction algorithms. QuantizedNN [41] proposes an efficient

privacy-preserving machine learning framework using the quantization scheme of Jacob et al. [42], and provides protocols under the semi-honest or malicious security setting.

In the above studies, the 2PC [18,20–22] requires both parties involved in the protocol to be semi-honest. The 4PC [39–41] is not of practical importance due to the limitation of the number of computing servers. Instead, the 3PC [32,34,37,38] is considered to be one of the most promising framework available due to its security and efficiency. The main techniques include secret sharing, garbled circuits [43], homomorphic encryption, GMW protocol, etc. Each technique has its advantages and disadvantages, for example, homomorphic encryption has the highest computational complexity, while secret sharing requires the least bandwidth compared to garbled circuits or GMW protocol. Therefore, we mainly use secret sharing to build a 3PC framework for privacy-preserving neural network prediction. Our contributions are shown below.

- A neural network prediction model for three-party secure computation is constructed. Experiments are conducted on pre-trained different model architectures with the MNIST dataset, achieving an accuracy of 96.64%, 97.04%, 98.70% and 96.50%, respectively, with errors of no more than 1% from the results predicted directly on the plaintext.
- We define two security models. In the semi-honest environment, collaborators behave honestly, performing calculations according to the protocol and ensuring that they do not learn the shares held by the remaining parties. In the malicious environment, the protocol is terminated if malicious activity is detected, ensuring correct computation. Depending on the application scenario and performance requirements, the choice between the two security environments can be made.
- New sub-protocol algorithms for neural networks are designed using secret sharing, and they can be combined to construct secure prediction model with different combinations, it was possible to construct secure prediction model, improving the time required for prediction and the cost of communication compared to previous work.

The rest of the paper is structured as follows: Section 2 focuses on the relevant theoretical knowledge; Section 3 introduces the prediction framework and the sub-protocol algorithms for the neural network; Section 4 presents the experimental results; and Section 5 provides the conclusion.

2. Preliminaries

In this section, we describe the concepts and notations needed in this paper.

2.1. Replicated Secret Sharing

The three parties are denoted by P_0, P_1, P_2 , respectively, then P_{i+1}, P_{i-1} denote the next and previous party of P_i , where the subscript is in modulo 3, i.e., the next party of P_0 is P_1 and the previous party is P_2 . Let $\llbracket x \rrbracket^L = (x_0, x_1, x_2)$ denotes the 2-out-of-3 replicated secret sharing [31,32,34] of x in the number field \mathbb{Z}^L , i.e., $x \equiv x_0 + x_1 + x_2 \pmod{L}$, where $L = 2^\ell$ (L is a number of data type with bit size ℓ). Then P_0 holds the pair (x_0, x_1) , P_1 holds the pair (x_1, x_2) and P_2 holds the pair (x_2, x_0) . Any two parties can combine to recover the original data x . Similarly, the 2-out-of-3 replicated secret sharing of y in the number field \mathbb{Z}^2 can be denoted by $\llbracket y \rrbracket^2 = (y_0, y_1, y_2)$, i.e., $y = y_0 \oplus y_1 \oplus y_2$.

Furthermore, Let $\llbracket x \rrbracket^L = (x_0, x_1, x_2)$ and $\llbracket y \rrbracket^2 = (y_0, y_1, y_2)$ denote the 3-out-of-3 secret sharing in the \mathbb{Z}^L and \mathbb{Z}^2 , respectively, then P_0 holds the shares x_0 and y_0 , P_1 holds the shares x_1 and y_1 , and P_2 holds the shares x_2 and y_2 . Only a joint three-party effort can recover the original data x and y .

2.2. Random Number

We need to add a certain amount of noise to the sharing values to avoid channel attacks in peer-to-peer transmission. When the party P_i generates a random number seed k_i and shares it with the previous party P_{i-1} , each of the three parties has the peer-to-peer communication channel $(P_i, P_{i-1}), (P_i, P_{i+1})$ and a sharing seed pair (k_i, k_{i+1}) . A pseudo-

random generation function F_{k_i} is used to generate public random numbers as follows, where ct is a counter that automatically adds 1 after each call to the pseudo-random generation function.

- The party P_i generates the random number $\alpha_i = F_{k_i}(ct) - F_{k_{i+1}}(ct) (i = 0, 1, 2)$, then all parties can directly generate a 3-out-of-3 secret sharing of 0 without interaction, i.e., $\alpha_0 + \alpha_1 + \alpha_2 \equiv 0(mod L)$.
- The party P_i generates the random number pair $(r_i, r_{i+1}) = (F_{k_i}(ct), F_{k_{i+1}}(ct)) (i = 0, 1, 2)$, then (r_0, r_1, r_2) can form a 2-out-of-3 replicated secret sharing of r without interaction, i.e., $r_0 + r_1 + r_2 \equiv r(mod L)$.

Optimization. In the main protocols of Section 3, we need parties to have the 2-out-of-3 replicated secret sharing of 0, which is proposed in Section 3.1 in [34]. Then the secret sharing of x is denoted by $(x_0, x_1, x_2) = (\alpha_0, \alpha_1 + x, \alpha_2)$, which can reduce the number of interactions between the parties. The random numbers can be pre-generated in the offline phase, so we focus mainly on the online phase of the computation.

3. Framework and Protocols

In this section, we describe the prediction model for 3PC, the basic operations and some sub-protocols for the fully connected, convolutional, activation and pooling layers over replicated secret sharing.

3.1. Overview of the Framework

In this paper, we focus on the prediction phase of neural networks, where users query inference results by leasing pre-trained models. Our proposed 3PC prediction model consists of three roles, as shown in Figure 1. One is the data owner P_0 , which applies its own private data to the leased model; the second is the service provider P_2 , which provides pre-trained network models for users to perform query services; and the last is the helper service P_1 , which performs auxiliary computations.

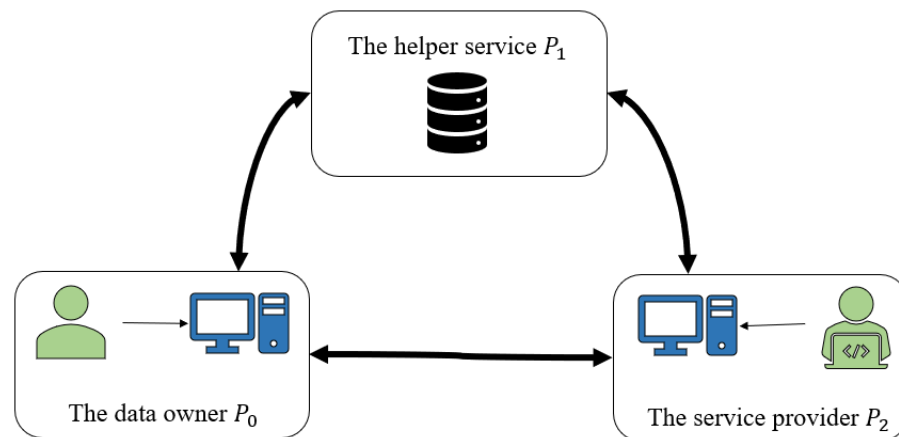


Figure 1. The prediction model for three-party secure computation.

First, P_0 shares private data $\llbracket x \rrbracket^L = (\alpha_0, x + \alpha_1, \alpha_2)$ and P_2 shares model parameters $\llbracket y \rrbracket^L = (\beta_0, \beta_1, y + \beta_2)$ in the form of 2-out-of-3 replicated secret sharing, where $\llbracket \alpha \rrbracket^L$ and $\llbracket \beta \rrbracket^L$ are both 2-out-of-3 replicated secret sharing of 0. Afterwards, the three parties jointly use their shared data for inference. The result is also in sharing form. Finally, the P_0 communicates with the P_1 and reconstructs to obtain the final inference result. In this way, the user’s input data, the specific parameters of the model, and the final inference output are all in secret sharing form, and no information about the original data can be leaked by any of the parties’ shares, thus allowing for complete privacy.

In most of the current studies [31,34,38], outsourced cloud computers are used to perform SMC. Therefore, the computation volume of each computer as well as its security

assumptions are the same, while in proposed model, P_0 , as the service renter, is semi-honest and does not modify the data in order to get the final expected value. Therefore, the proposed model has the following two definitions of security, based on the security assumptions of the party P_1 and P_2 .

Definition 1. *The model is defined as a semi-honest security model if both P_1 and P_2 are semi-honest.*

Definition 2. *The model is defined as a malicious security model if P_1 or P_2 is malicious.*

In the semi-honest security model, the parties will obey the protocol but will attempt to learn the shares owned by the remaining parties from the results of the intermediate calculations received. In a malicious security model, the malicious party may corrupt the correctness of the protocol by tampering with the intermediate data so that the model does not achieve the desired result. Therefore, we need a method of validation to ensure that the protocol can proceed correctly.

3.2. Basic Operation

Linear calculations. Assuming that a, b, c are common constant and $\llbracket x \rrbracket^L, \llbracket y \rrbracket^L$ are secret sharing, then the sharing of the linear calculation $\llbracket ax + by + c \rrbracket^L$ can be expressed as $(ax_0 + by_0 + c, ax_1 + by_1, ax_2 + by_2)$. The addition between secret sharing can be added directly corresponding to shares, whereas constant addition only requires adding the constant to a certain share. Constant multiplication requires that each share be multiplied by the constant. These three operations only need to be calculated locally without interaction, so does not require a security algorithm.

Multiplication. Let $\llbracket x \rrbracket^L = (x_0, x_1, x_2)$ and $\llbracket y \rrbracket^L = (y_0, y_1, y_2)$, then a total of two steps are required to obtain the secret sharing of $z = xy$. First, party P_i computes $z_i = x_i y_i + x_{i+1} y_i + x_i y_{i+1}$ locally, which gets the 3-out-of-3 secret sharing of z , i.e., $\llbracket z \rrbracket^L = (z_0, z_1, z_2) = (x_0 y_0 + x_1 y_0 + x_0 y_1, x_1 y_1 + x_2 y_1 + x_2 y_2, x_2 y_2 + x_0 y_2 + x_2 y_0)$, which is defined as $\llbracket z \rrbracket^L := \llbracket x \rrbracket^L \llbracket y \rrbracket^L$. Afterwards, the party P_i sends share $z_i + \alpha_i$ to party P_{i-1} , which gets the 2-out-of-3 replicated secret sharing of z , i.e., $\llbracket z \rrbracket^L = (z_0 + \alpha_0, z_1 + \alpha_1, z_2 + \alpha_2)$, where a 3-out-of-3 secret sharing of 0 is added for blinding to ensure the security of the data.

In the malicious security model, the information sent by P_1 and P_2 could be changed, so validation means need to be added to ensure the correctness of the data, as shown in Algorithm 1. An additional uniformly distributed tuple $(\llbracket a \rrbracket^L, \llbracket b \rrbracket^L, \llbracket c \rrbracket^L)$ is needed, which $c \equiv ab \pmod L$.

Algorithm 1 Multiplication protocol Π_{Mult}

Input: shares of $\llbracket x \rrbracket^L$ and $\llbracket y \rrbracket^L$ held by P_0, P_1 , and P_2 .

Output: shares of $\llbracket z \rrbracket^L$ held by P_0, P_1 , and P_2 .

Common Randomness: 3-out-of-3 secret sharing $\llbracket \alpha \rrbracket^L$ of 0, 2-out-of-3 replicated secret sharing $\llbracket \beta \rrbracket^L$ of 0, and uniformly distributed tuple $(\llbracket a \rrbracket^L, \llbracket b \rrbracket^L, \llbracket c \rrbracket^L)$, where $c \equiv ab \pmod L$.

- 1: for $i = 0, 1, 2$, P_i computes locally $\llbracket u \rrbracket^L := \llbracket x \rrbracket^L - \llbracket a \rrbracket^L, \llbracket v \rrbracket^L := \llbracket y \rrbracket^L - \llbracket b \rrbracket^L$.
 - 2: for $i = 0, 1, 2$, P_i computes $\llbracket z \rrbracket^L := \llbracket x \rrbracket^L \llbracket y \rrbracket^L, \llbracket p \rrbracket^L := \llbracket u \rrbracket^L \llbracket b \rrbracket^L, \llbracket q \rrbracket^L := \llbracket a \rrbracket^L \llbracket v \rrbracket^L, \llbracket o \rrbracket^L := \llbracket u \rrbracket^L \llbracket v \rrbracket^L$ and $s_i = p_i + q_i + o_i + c_i$. These calculations can be performed locally.
 - 3: for $i = 1, 2$, P_i sends $(z_i + \alpha_i, s_i)$ to P_0 in the malicious security model, while P_i only needs to send $z_i + \alpha_i$ to P_0 in the semi-honest security model.
 - 4: P_0 computes $s' = \sum_i (z_i + \alpha_i - s_i)$. Output \perp if $s' \neq 0$, otherwise, continue the protocol.
 - 5: P_0 computes locally $z \equiv z_0 + \alpha_0 + z_1 + \alpha_1 + z_2 + \alpha_2 \pmod L$.
 - 6: P_0 transmits $z + \beta_1$ to P_1 .
 - 7: **return** $\llbracket z \rrbracket^L = (\beta_0, \beta_1 + z, \beta_2)$.
-

Theorem 1. *The protocol Π_{Mult} described in Algorithm 1 allows for correct and secure operation of the multiplication functionality.*

Proof of Theorem 1. In step 1 and 2, there are $u = x - a, v = y - b, z = xy, p = ub, q = av, o = uv$, then

$$\begin{aligned} s &= p + q + o + c \\ &= (x - a)b + a(y - b) + (x - a)(y - b) + ab \\ &= xb - ab + ay - ab + xy - ay - xb + ab + ab \\ &= xy \end{aligned}$$

So in step 4, $s' = z + \alpha - s = 0$, which means the messages sent by P_1 and P_2 are correct, without being tempered, and can continue to be calculated. Otherwise, terminal the protocol. Thus, in step 5, P_0 can correctly calculate $z = xy$ and send the share to get $\llbracket z \rrbracket^L$ after adding the random number. The introduction of common randomness during the transmission ensures the security of the data in Algorithm 1. \square

Reconstruction. In the semi-honest security model, denoted by $\llbracket \alpha \rrbracket^L$ for 2-out-of-3 replicated secret sharing of 0, P_1 sends $x_2 + \alpha_2$ to P_0 , then the party P_0 has shares $x_0 + \alpha_0, x_1 + \alpha_1$, and $x_2 + \alpha_2$, which can be recovered by computing $x \equiv x_0 + \alpha_0 + x_1 + \alpha_1 + x_2 + \alpha_2 \pmod L$ to recover the global value x , which only need 1 round communication and 1 message. In the malicious security model, because P_1 and P_2 may tamper with the data, in order to ensure the correctness of the final result, it is necessary for P_1 and P_2 to transmit both $x_2 + \alpha_2$ to P_0 . P_0 judges the received messages, and if the two messages are equal, it means that the correct value is transmitted in this round of communication without tampering, and the subsequent calculation can be continued. Otherwise, it is immediately terminated.

3.3. Fully Connected Layer

The computation of the fully connected layer is a matrix multiplication such that the input $X(n \times 1)$ and the weight $W(m \times n)$, then the output $Y = W \times X (y_i = \sum_{j=1}^n w_{ij}x_j, i = 1, \dots, m)$. Since the addition calculation can be performed directly locally, the main focus is on the multiplication between the shares and the matrix multiplication based on it. We use a fixed-point algorithm (as shown in Section 5.1 of [34]), so the matrix multiplication result needs to be truncated in the protocol Π_{MatMul} , which is shown in Algorithm 2, to ensure that the accuracy of the input and output remains consistent.

The two secret sharing of input are $\llbracket W \rrbracket^L = (W_0, W_1, W_2), \llbracket X \rrbracket^L = (X_0, X_1, X_2)$, and the sharing of output is $\llbracket Y = WX \rrbracket^L$. The protocol needs to generate a pair of shared random numbers $(\llbracket R \rrbracket^L, \llbracket R' \rrbracket^L)$, where $R = R' / 2^d$ (Each element of the matrix is divided by $2^d, d$ indicates the precision bit of the input). First, the parties compute locally $\llbracket Y' \rrbracket^L = \llbracket W \rrbracket^L \llbracket X \rrbracket^L$, where $Y = Y' / 2^d$. After that, P_1 transmits the blinded Y'_1 with R'_1 to P_0 , and P_2 transmits the blinded Y'_2 with R'_2 to P_0 , ensuring the privacy of the shared values in transmission. P_0 receives the messages from P_1 and P_2 and performs an addition operation for reconstruction, as shown in Line 5, to obtain the blinded Y' , so that the parameter of the party P_2 cannot be calculated by $W = (Y' - R')X^{-1}$. Because multiplication leads to a larger precision of the result, the truncation algorithm $Z = (Y' - R') / 2^d$ is needed to reduce the precision. In order to get the output sharing, P_0 needs to blind Z and send it to P_1 , then the final 2-out-of-3 replicated secret sharing $\llbracket Y \rrbracket^L = (R_0, Z + R_1, R_2)$ is obtained.

In the malicious security model, similarly, an additional uniformly distributed tuple $(\llbracket A \rrbracket^L, \llbracket B \rrbracket^L, \llbracket C \rrbracket^L)$ is needed, which $C \equiv AB \pmod L$, to ensure the correctness of the data.

Algorithm 2 Matrix Multiplication protocol Π_{MatMul}

Input: shares of $[[W]]^L$ and $[[X]]^L$ held by $P_0, P_1,$ and P_2 .

Output: shares of $[[Y]]^L$ held by $P_0, P_1,$ and P_2 .

Common Randomness: truncation pair of shared random number ($[[R]]^L, [[R']]^L$), where $R = R'/2^d$, and uniformly distributed tuple ($[[A]]^L, [[B]]^L, [[C]]^L$), where $C \equiv AB(mod L)$.

- 1: for $i = 0, 1, 2, P_i$ computes locally $[[U]]^L = [[W]]^L - [[A]]^L, [[V]]^L = [[X]]^L - [[B]]^L$.
 - 2: for $i = 0, 1, 2, P_i$ computes $[[Y']]^L = [[W]]^L [[X]]^L, [[P']]^L = [[U]]^L [[B]]^L, [[Q']]^L = [[A]]^L [[V]]^L, [[O']]^L = [[U]]^L [[V]]^L$ and $S_i = Y'_i - P'_i - Q'_i - O'_i - C_i$. These calculations can be performed locally.
 - 3: for $i = 1, 2, P_i$ transmits $(Y'_i - R'_i, S_i)$ to P_0 in the malicious security model, while P_i only needs to send $Y'_i - R'_i$ to P_0 in the semi-honest security model.
 - 4: P_0 computes $S' = \sum_i S_i$. Output \perp if $S' \neq 0$, otherwise, continue the protocol.
 - 5: P_0 computes $Y' - R' = \sum_i (Y'_i - R'_i)$ and truncates $Z = (Y' - R')/2^d$.
 - 6: P_0 transmits $Z + R_1$ to P_1 .
 - 7: **return** $[[Y]]^L = (R_0, Z + R_1, R_2)$.
-

Theorem 2. The protocol Π_{MatMul} described in Algorithm 2 allows for correct and secure operation of the matrix multiplication functionality.

Proof of Theorem 2. The proof of the theorem proceeds similarly to Theorem 1. \square

3.4. Convolutional Layer

The convolution computation can be viewed as the merging of multiple matrix multiplications, so the convolution can be expanded into a matrix of larger dimensionality. Suppose the convolution calculation between an input image of 3×3 and a kernel of size 2×2 (with the step of 1) can be converted into a matrix multiplication between $W(1 \times 4)$ and $X(4 \times 4)$, which as follows.

$$Conv \left(\begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix}, \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \right) = [k_1 \quad k_2 \quad k_3 \quad k_4] \times \begin{bmatrix} x_1 & x_2 & x_4 & x_5 \\ x_2 & x_3 & x_5 & x_6 \\ x_4 & x_5 & x_7 & x_8 \\ x_5 & x_6 & x_8 & x_9 \end{bmatrix}$$

Then we can call the protocol Π_{MatMul} directly to calculate it.

3.5. Wrap Function

The definition of the *wrap* functions $wrap_{2e}, wrap_{3e}$ and $wrap_3$ is given below, which can compute the carry efficiently when the shared values are summed as integers.

$$wrap_{2e}(a_0, a_1, L) = \begin{cases} 0 & \text{if } 0 \leq a_0 + a_1 < L \\ 1 & \text{Otherwise} \end{cases} \tag{1}$$

$$wrap_{3e}(a_0, a_1, a_2, L) = \begin{cases} 0 & \text{if } 0 \leq \sum_{i=0}^2 a_i < L \\ 1 & \text{if } L \leq \sum_{i=0}^2 a_i < 2L \\ 2 & \text{if } 2L \leq \sum_{i=0}^2 a_i < 3L \end{cases} \tag{2}$$

$$wrap_3(a_0, a_1, a_2, L) = wrap_{3e}(a_0, a_1, a_2, L)(mod 2) \tag{3}$$

Next, we describe the relationship between the most significant bit of a and $wrap_3$ function on the secret sharing (a_0, a_1, a_2) . Note that $a \equiv a_0 + a_1 + a_2(mod L)$ and a_i is a share within modulo L , and we can see the most significant bit $MSB(a) = MSB(a_0) + MSB(a_1) + MSB(a_2) + c(mod 2)$, where c is the carry of the previous index. The key insight here is that ignoring the most significant bit of a_i , the carry c of the previous

index is obtained by performing the $wrap_3$ function on a_i with modulo $L/2$. Furthermore, this operation is equivalent to performing $wrap_3$ function on $2a_i$ with modulo L . That is, $c = wrap_3(a_0, a_1, a_2, L/2) = wrap_3(2a_0, 2a_1, 2a_2, L)$.

The $wrap_{2e}$ function can be computed directly locally and therefore does not require a security algorithm. In addition, the $wrap_{2e}$ function allows us to write the following exact integer equation: if $a \equiv a_0 + a_1 \pmod L$, then $a = a_0 + a_1 - wrap_{2e}(a_0, a_1, L) \cdot L$. Where the former is a congruence relation and the latter is an integer relation which is exactly equal. Finally, Algorithm 3 gives the $wrap_3$ protocol on the 2-out-of-3 replicated secret sharing of a . It is necessary to blind the transmitted data in the communication to ensure data security, so two 2-out-of-3 replicated secret sharing of 0, denoted by $\llbracket r \rrbracket^L$ and $\llbracket \eta \rrbracket^2$, are introduced.

Algorithm 3 Wrap protocol \prod_{Wrap}

- Input:** shares of $\llbracket a \rrbracket^L$ held by P_0, P_1 , and P_2 .
Output: shares of a bit $\llbracket \theta \rrbracket^2$ held by P_0, P_1 , and P_2 , where $\theta = wrap_3(a_0, a_1, a_2, L)$.
Common Randomness: $\llbracket r \rrbracket^L$ (random shares of 0), $\llbracket \alpha \rrbracket^2$ where $\alpha = wrap_3(r_0, r_1, r_2, L)$, and $\llbracket \eta \rrbracket^2$ (a bit random shares of 0).
- 1: for $i = 0, 1, 2, P_i$ computes locally $x_i \equiv a_i + r_i \pmod L$ and $\beta_i = wrap_{2e}(a_i, r_i, L)$.
 - 2: In the semi-honest security setting, P_1 sends x_2 to P_0 . While in the malicious security setting, P_1 sends x_2 to P_0 and P_2 sends $H(x_2)$ to P_0 .
 - 3: In the malicious security setting, P_0 first needs to judge whether the hash value of x_2 from P_1 is equal to $H(x_2)$ from P_2 . Output \perp if it is not equal, otherwise, continue the protocol.
 - 4: P_0 computes $\delta = wrap_3(x_0, x_1, x_2, L)$ and sends $\delta + \eta_1$ to P_1 .
 - 5: **return** $\llbracket \theta \rrbracket^2 = (\beta_0 \oplus \alpha_0 \oplus \eta_0, \beta_1 \oplus \alpha_1 \oplus \delta \oplus \eta_1, \beta_2 \oplus \alpha_2 \oplus \eta_2)$.
-

Theorem 3. The protocol \prod_{Wrap} described in Algorithm 3 allows for correct and secure operation of the $wrap_3$ functionality.

Proof of Theorem 3. The following equations exist in Algorithm 3:

$$a = a_1 + a_2 + a_3 - \theta_e \cdot L \tag{4}$$

$$r = r_0 + r_1 + r_2 - \alpha_e \cdot L \tag{5}$$

$$x_i = a_i + r_i - \beta_i \cdot L, \forall i \in \{0, 1, 2\} \tag{6}$$

$$x = x_0 + x_1 + x_2 - \delta_e \cdot L \tag{7}$$

where θ_e, α_e and δ_e represent the exact $wrap_{3e}$ function, Equations (4), (5) and (7) follow the definition of the exact $wrap_{3e}$ function, while Equation (6) follows the definition of the $wrap_{2e}$ function. because $x \equiv a + r \pmod L$ and $r = 0$, so $x \equiv a \pmod L$. From the Eqs.4-7 we can get

$$\theta_e = \beta_0 + \beta_1 + \beta_2 - \delta_e - \alpha_e \tag{8}$$

Equation (8) modulo 2 is operated to get $\theta = \beta_0 + \beta_1 + \beta_2 - \delta - \alpha$, which is used to calculate the $wrap_3$ in Algorithm 3. Similarly, in the malicious security model, the verification method in step 3 ensures that the data is correct, so the protocol \prod_{Wrap} is correct. In addition, the introduction of random numbers ensures the security of Algorithm 3. \square

3.6. ReLU Activation Layer

In the decimal real number field, the activation function $ReLU(x)$ is defined as $ReLU(x) = x$ if $x > 0$, otherwise, $ReLU(x) = 0$. Under the C++ data type representation on the ring \mathbb{Z}^L , $x \geq L/2$ indicates a negative number whose most significant bit

$MSB(x) = 1$, and $0 \leq x < L/2$ indicates a positive number whose most significant bit $MSB(x) = 0$. Therefore, when using fixed-point encoding, the definition of $ReLU(x)$ can be converted as follows: if $MSB(x) = 1$, $ReLU(x) = 0$, otherwise, $ReLU(x) = x$, i.e., $ReLU(x) = x \cdot (1 - MSB(x))$.

When the secret shared values of x are known, the most significant bit $MSB(x)$ is given as follows.

$$MSB(x) = MSB(x_0) \oplus MSB(x_1) \oplus MSB(x_2) \oplus wrap_3(2x_0, 2x_1, 2x_2, L) \tag{9}$$

The secret sharing of $MSB(x)$ is obtained by first calling the Π_{Wrap} protocol to calculate the sharing of the carry bits, and then locally dissociating directly with the most significant bit of the shares of x , as shown in Line 2 of Algorithm 4. If the secret sharing of x and $MSB(x)$ are held in different number fields, i.e., $\llbracket x \rrbracket^L = (x_0, x_1, x_2)$ and $\llbracket MSB(x) \rrbracket^2 = \llbracket 0 \rrbracket^2 = (1, 1, 0)$, then calling the multiplication protocol on two sharing can obtain $\llbracket 2x \rrbracket^L = (2x_0 + x_1, x_1 + x_2, x_2)$, which is not the expected value. So we cannot call the multiplication protocol directly, but need to convert the secret sharing of $MSB(x)$ in \mathbb{Z}^2 to the secret sharing of $1 - MSB(x)$ in \mathbb{Z}^L and then call the protocol Π_{Mult} to get the 2-out-of-3 replicated secret sharing of $ReLU(x)$.

Algorithm 4 ReLU protocol Π_{RU}

Input: shares of $\llbracket x \rrbracket^L$ held by P_0, P_1 , and P_2 .

Output: shares of $\llbracket y \rrbracket^L$ held by P_0, P_1 , and P_2 , where $y = ReLU(x)$.

Common Randomness: random numbers $\llbracket r \rrbracket^L$ and $\llbracket r \rrbracket^2$.

- 1: Run Π_{Wrap} to get $\llbracket \theta \rrbracket^2$ where $\theta = wrap_3(2x_0, 2x_1, 2x_2, L)$.
 - 2: for $i = 0, 1, 2$, P_i computes locally $(\theta_i \oplus MSB(x_i), \theta_{i+1} \oplus MSB(x_{i+1}))$ to obtain $\llbracket MSB(x) \rrbracket^2 = (\theta_0 \oplus MSB(x_0), \theta_1 \oplus MSB(x_1), \theta_2 \oplus MSB(x_2))$.
 - 3: Run $\Pi_{Reconst}$ to get c where $c = (r \oplus MSB(x))$ and P_0 broadcasts c to P_1 and P_2 .
 - 4: If $c = 0$, set $\llbracket 1 - MSB(x) \rrbracket^L = 1 - \llbracket r \rrbracket^L$, otherwise, $\llbracket 1 - MSB(x) \rrbracket^L = \llbracket r \rrbracket^L$.
 - 5: Run Π_{Mult} over $\llbracket x \rrbracket^L$ and $\llbracket 1 - MSB(x) \rrbracket^L$ to get $\llbracket y \rrbracket^L$.
 - 6: **return** $\llbracket y \rrbracket^L$.
-

Theorem 4. The protocol Π_{RU} described in Algorithm 4 allows for correct and secure operation of the ReLU functionality.

Proof of Theorem 4. The protocol Π_{Wrap} called in step 1 is correct and secure. Step 2 is computed locally without the security algorithm. Step 3 invokes the protocol $\Pi_{Reconst}$ and then exposes c as the blinded $MSB(x)$, so ensuring that the true value of $MSB(x)$ is not known to the parties. Step 4 is computed locally and if $c = 0$, indicating that r is the same as $MSB(x)$, then $1 - MSB(x) = 1 - r$, otherwise $c \neq 0$ when r is complementary to $MSB(x)$, then $1 - MSB(x) = r$. The protocol Π_{Mult} called in step 5 is correct and secure. Therefore, it follows from Theorem 1.2 in [44] that combined protocol Π_{RU} is also correct and secure. \square

3.7. Max-Pooling Layer

In the max-pooling layer, we treat the elements in the local field as a row vector as follows:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 & x_2 & x_4 & x_5 \\ x_2 & x_3 & x_5 & x_6 \\ x_4 & x_5 & x_7 & x_8 \\ x_5 & x_6 & x_8 & x_9 \end{bmatrix}.$$

Then the computation at this layer is to find the maximum element in each row vector. The functionality of the protocol Π_{MP} is simply expressed as the input of a vector in the form of secret sharing and the output of secret sharing of its maximum value. At the intermediate layer, we only need to focus on the maximum value of each vector, so it is

sufficient to carry out the calculations in steps 3, 4 and 5 of Algorithm 5. Iterating through the input via a for loop, the difference between the two numbers is first calculated and the protocol Π_{RU} is called to determine if the difference is zero, which in turn gives the maximum value of the current traversal. In the final predictive classification, the index of the maximum value in the vector, i.e., the label, needs to be obtained. So we pre-generate a number of one-hot vectors \vec{e}_k with $k = \{1, 2, \dots, n\}$ and select the output in step 6 to get the index of the maximum value of the output vector.

Algorithm 5 Maxpool function Π_{MP}

Input: shares of x_1, x_2, \dots, x_n in \mathbb{Z}^L held by P_0, P_1 , and P_2 .

Output: shares of x_k and \vec{e}_k held by P_0, P_1 , and P_2 , where $k = \text{argmax}\{x_1, x_2, \dots, x_n\}$ and $\vec{e}_k = \{e_1, e_2, \dots, e_n\}$ with $e_k = 1$ and $e_i = 0 \forall i \neq k$.

Common Randomness: No additional common randomness required.

- 1: Set $\text{max} \leftarrow a_1$ and $\vec{\text{ind}} \leftarrow \vec{e}_1 = \{1, 0, \dots, 0\}$.
 - 2: **for** $i = \{2, 3, \dots, n\}$ **do**
 - 3: Set $d \leftarrow (\text{max} - a_i)$.
 - 4: $b \leftarrow \text{MSB}(d), c \leftarrow \text{ReLU}(d)$.
 - 5: $\text{max} \leftarrow c + a_i$.
 - 6: $\vec{\text{ind}} \leftarrow (\vec{\text{ind}} \oplus \vec{e}_i)b \oplus \vec{\text{ind}}$.
 - 7: **end for**
 - 8: **return** $\text{max}, \vec{\text{ind}}$.
-

Theorem 5. *The protocol Π_{MP} described in Algorithm 5 allows for correct and secure operation of the Maxpool functionality.*

Proof of Theorem 5. Algorithm 5 calls the protocol Π_{RU} in step 4 (the computation of b is in steps 1 and 2 of Algorithm 4), and the protocol Π_{Mult} in step 6. Similarly, because sub-protocols are secure, the combined protocol is also secure. In step 6, if $b = 0$, $(\vec{\text{ind}} \oplus \vec{e}_i)b \oplus \vec{\text{ind}} = \vec{\text{ind}}$, so the current $\text{max} > a_i$ and output $\vec{\text{ind}}$. If $b = 1$, $(\vec{\text{ind}} \oplus \vec{e}_i)b \oplus \vec{\text{ind}} = \vec{\text{ind}} \oplus \vec{e}_i \oplus \vec{\text{ind}} = \vec{e}_i$, so $\text{max} < a_i$ and the one-hot vector e_i corresponding to a_i is outputted. In summary, Algorithm 5 is correct and secure. \square

3.8. Performance Analysis

We summarise the overhead of the protocol theoretically, with the number of rounds and the communication complexity shown in Table 1. The protocol $\Pi_{Reconst}$ requires the transmission of 1 message in the semi-honest security model and 2 messages in the malicious security model. the protocol Π_{MatMul} represents matrix multiplication of dimensions $x \times y$ and $y \times x$ and requires 2 communications in the semi-honest security model (e.g., steps 3 and 6), whereas in the malicious security model, although the same 2 communications are performed, the first communication complexity is $2 \times$. If the number of elements transmitted is the same, the communication complexity of Π_{Wrap} under the malicious security model can be the same as that of the protocol Π_{Mult} under the semi-honest security model. The focus of protocol Π_{RU} is on the combined computation of $\Pi_{Wrap}, \Pi_{Reconst}$ and Π_{Mult} . The protocol Π_{MP} only needs to compute the maximum value in the pooling layer of the neural network, i.e., only steps 3–5 in the for loop need to be performed. Let matrix dimension after local feature transformation be $x \times y$, then the protocol Π_{RU} needs to be called $y - 1$ times. The full protocol Π_{MP} is only called when the final label classification is performed and the one-hot encoding of each label needs to be calculated.

As can be seen from Table 1, the number of rounds under the malicious security model is essentially the same as that under the semi-honest security model, but the communication complexity is approximately $2 \times$, making it a much less efficient implementation.

Table 1. Theoretical overheads of protocols.

Protocol	Dependence	Semi-Honest		Malicious	
		Rounds	Comm.	Rounds	Comm.
$\Pi_{Reconst}$	n	1	$1kn$	1	$2kn$
Π_{Mult}	n	2	$3n$	2	$5n$
Π_{MatMul}	$(x \times y)(y \times z)$	2	$3kxz$	2	$5kxz$
Π_{Wrap}	n	2	$2kn$	2	$3kn$
Π_{RU}	n	6	$8kn$	6	$12kn$
Π_{MP}	$x \times y$	$8(y - 1)$	$11kn(y - 1)$	$8(y - 1)$	$17kn(y - 1)$

Note: k is the number of bytes occupied by each value.

We compared the theoretical complexity of the Astra [45], Blaze [46], Falcon [38], Flash [40] and Trident [39] frameworks with the proposed scheme. We compares the end-to-end overhead of the protocol Π_{RU} due to the different methods used in this framework to calculate the non-linear functions. Table 2 shows the comparison of the theoretical complexity. Astra is a third-party secure computation framework with semi-honest security. Blaze builds on Astra to achieve malicious security and fairness in a 3PC honest majority corruption model and uses an adder circuit approach for non-linear function computation. The framework which is most similar to the proposed scheme is Falcon and introduces a non-zero random number r in step 1 of the protocol Π_{Wrap} , so the PC protocol needs to be called in step 4 to compare the x and r , which is in Section 3.3 of the [38]. In contrast, we introduce a secret sharing of 0, so there is no need to call the PC protocol, resulting in a $\log \ell$ reduction in the number of rounds and a $4 \times$ reduction in communication complexity for the protocol Π_{RU} .

Table 2. Comparison of theoretical complexity of sub-protocols in each framework.

Framework	Multiplication		ReLU	
	Rounds	Comm.	Rounds	Comm.
Astra [45]	1	4ℓ	$3 + \log \ell$	45ℓ
Blaze [46]	1	3ℓ	4	$(\kappa + 7)\ell$
Falcon [38]	1	4ℓ	$5 + \log \ell$	32ℓ
Proposed	2	3ℓ	5	8ℓ
Flash [40]	1	3ℓ	$10 + \log \ell$	46ℓ
Trident [39]	1	3ℓ	4	$8\ell + 2$

Note: ℓ is the bit size of the data type, κ is the security parameter, and \log denotes the logarithm with base 2.

4. Results

Our experiments are written in C++ program on the virtual machine Ubuntu 22.04.1 LTS amd64 and focus on evaluating the performance overhead of the proposed scheme for private prediction on multiple neural networks.

The training is first performed on plaintext data using the torch library, with the accuracy of 98.06% on Network-A, 99.00% on Network-B, 99.56% on Network-C and 99.77% on Network-D (networks are described in Section 4.1). Afterwards, the performance of various sub-protocols under semi-honest and malicious security is benchmarked as shown in Section 4.2. Section 4.3 evaluates the prediction time and communication cost of the proposed scheme in this paper based on the model parameters obtained from training with the test dataset. Finally, the feasibility of this scheme is verified by comparing the accuracy under the secret sharing and plaintext in Section 4.4.

The inputs to the neural network as well as the parameters are generally floating point numbers, so they must be encoded in fixed point form, setting their precision bit to 13. Experiments in this paper are conducted using a smaller ring $\ell = 32$, allowing the entire framework to run on smaller data types with half the communication complexity compared

to the generic framework with $\ell = 64$. The Eigen library is used to speed up the matrix multiplication calculation.

4.1. Dataset and Networks

The dataset is MNIST [47], where each image is a 28×28 pixel handwritten digital image with labels between 0 and 9, consisting of 60,000 images in the training set and 10,000 images in the test set.

For comparisons with different neural network architectures, four standard network architectures are chosen for the experiments, which are shown in Figure 2.

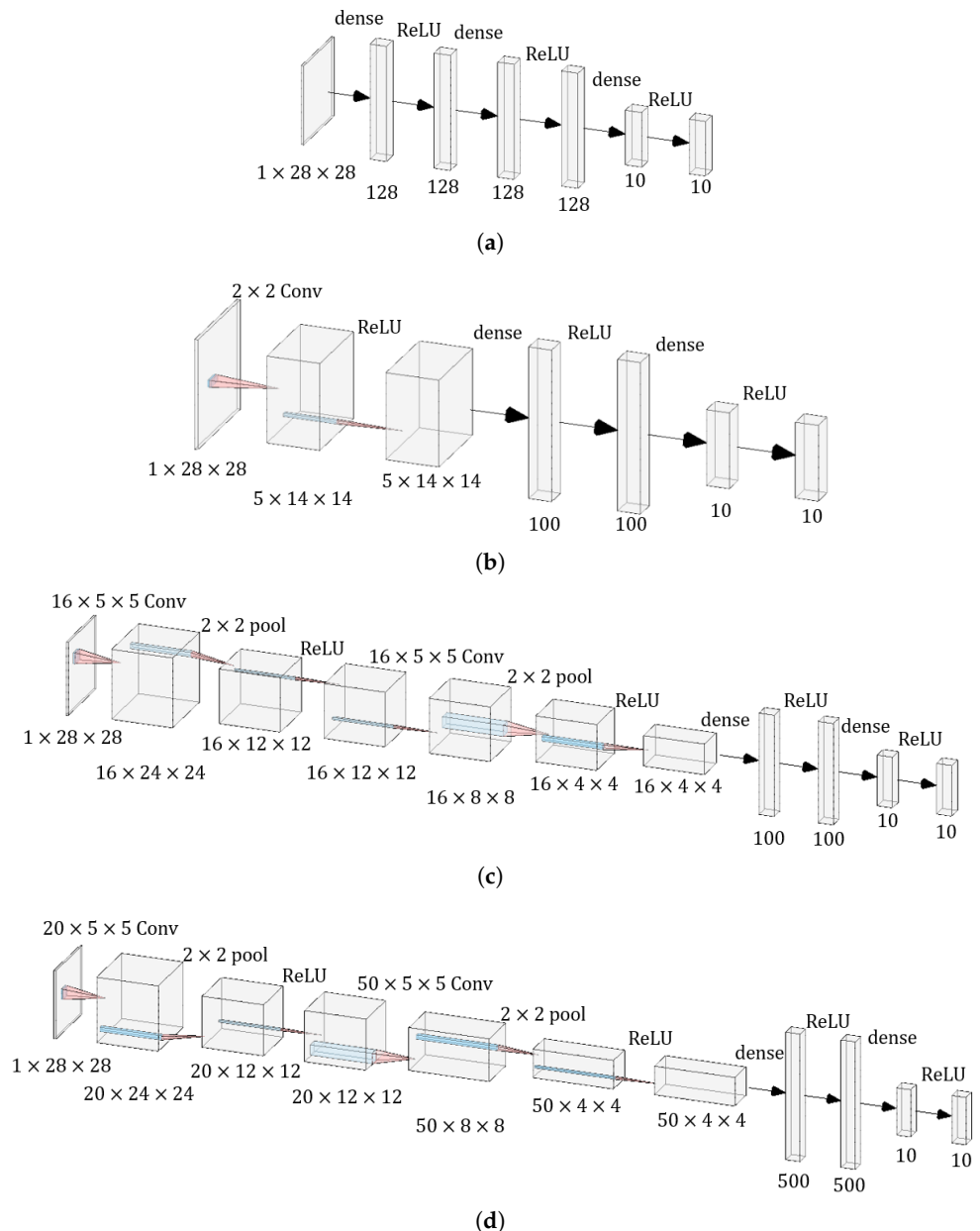


Figure 2. Four network architectures. (a) Network-A. (b) Network-B. (c) Network-C. (d) Network-D.

- Network-A: This is a neural network evaluated in the SecureML [18] framework. It consists of 3 fully connected layers, with an activation layer added after each layer. The number of nodes in each layer is 128×784 , 128×128 , 10×128 . This is the smallest network with 118,282 parameters.

- Network-B: This is a neural network evaluated in the Chameleon [35] framework. The network structure has 6 layers in total. The first layer is a convolutional layer with a kernel of $2 \times 2 \times 1 \times 5$. The third and fifth layers are fully connected layers with nodes of 100×980 , 10×100 , respectively. The second, fourth and sixth layers are the activation layers. Its total number of parameters is about 99,135.
- Network-C: This is a neural network evaluated in the MiniONN [48] framework. The network has a total of 10 layers. The first and fourth layers are convolutional layers with kernels of $5 \times 5 \times 1 \times 16$, $5 \times 5 \times 16 \times 16$, respectively. The second and fifth layers are pooling layers with kernels of 2×2 . Layers 7 and 9 are fully connected layers with nodes of 100×256 , 10 , respectively, and the rest of the layers are activation layers. Its total number of parameters is 33,542.
- Network-D: The network was first proposed in [47] for automatic detection of postal codes and digit recognition, which has 10 layers. The first and fourth layers are convolutional layers with kernels of $5 \times 5 \times 1 \times 20$, $5 \times 5 \times 20 \times 50$, respectively. The second and fifth layers both are pooling layers with kernels of 2×2 . Layers 7 and 9 are fully connected layers with nodes of 500×800 , 10×500 , respectively, and the rest of the layers are activation layers. It has a total of 431,080 parameters.

4.2. Microbenchmarks

In this section, we benchmark the computations of the fully connected, convolutional, activation and pooling layers. Three different parameter sets are used for the experiments, and the average of 100 experiments is taken as the result. A comparison of the sub-layers performance in a semi-honestly security model is shown in Table 3. The fully connected and convolutional layers mainly invoke the protocol Π_{MatMul} which reduces the computational complexity from 4ℓ to 3ℓ compared to Falcon, resulting in an improvement in communication cost of about $1/4$ under the same parameters. In the activation layer, compared to Falcon, there are fewer calls to the comparison protocol, resulting in performance improvement of about $2.7 \times -9.0 \times$ in computation time and about $10.95 \times$ in communication cost for the same parameters. In the pooling layer, we mainly calculate the maximum value in the local field without focusing on its index, and the protocol Π_{MP} is based on the protocol Π_{RU} , so the performance is also improved compared to Falcon, with the $5.1 \times -8.4 \times$ improvement in computation time and about $11.6 \times -13.4 \times$ in communication cost.

Table 3. Microbenchmarks in semi-honest security.

Layers	Dimension	FALCON		Proposed	
		Time (ms)	Comm (KB)	Time (ms)	Comm (KB)
Fully-connect1	784, 128, 10	1.55	122.50	1.41	91.88
Fully-connect2	1, 500, 100	0.44	1.56	0.36	1.17
Fully-connect3	1, 100, 1	0.18	0.02	0.17	0.01
Conv1	28, 5, 1, 20	0.63	180.00	0.53	135.00
Conv2	28, 3, 1, 20	0.50	211.25	0.42	158.44
Conv3	8, 5, 16, 50	0.67	12.50	0.61	9.38
ReLU1	128, 128	5.38	3504.00	0.60	320.00
ReLU2	576, 20	1.44	2463.75	0.50	225.00
ReLU3	64, 16	1.19	219.00	0.43	20.00
Maxpool1	$24 \times 24 \times 20, 2 \times 2$	7.12	1949.06	1.39	168.75
Maxpool2	$24 \times 24 \times 16, 2 \times 2$	6.64	1559.25	1.19	135.00
Maxpool3	$8 \times 8 \times 50, 4 \times 4$	3.43	782.23	0.41	58.59

In the malicious security model, the computational performance of the sub-layers is shown in Table 4. The performance is lower than in the semi-honest security model due to the need to verify the correctness of the data in the communication in each protocol. In our scheme, the introduction of the hash function leads to a significant reduction in communication cost. Compared to Falcon, the improvement is much higher than in the

semi-honest model. For example, with the same parameter setting of the fully connected and convolutional layers, the communication cost is improved by about $1.3\times$ with the semi-honest security setting, while it can be improved by about $1.8\times$ – $101.9\times$ with the malicious security setting. With the same parameter setting in the activation layer, the communication cost is increased by about $11.0\times$ with the semi-honest security setting, while it can be increased by $51.1\times$ with the malicious security setting. In the pooling layer with the same parameter setting, the communication cost is increased by approximately $12\times$ with the semi-honest security setting, while it is increased by approximately $54\times$ with the malicious security setting.

Table 4. Microbenchmarks in malicious security.

Layers	Dimension	FALCON		Proposed	
		Time (ms)	Comm (KB)	Time (ms)	Comm (KB)
Fully-connect1	784, 128, 10	7.10	809.88	4.72	153.13
Fully-connect2	1, 500, 100	1.03	198.63	0.59	1.95
Fully-connect3	1, 100, 1	0.30	1.20	0.27	0.02
Conv1	28, 5, 1, 20	2.93	402.31	1.67	225.00
Conv2	28, 3, 1, 20	2.06	406.39	0.95	264.06
Conv3	8, 5, 16, 50	2.90	176.56	1.78	15.63
ReLU1	128, 128	5.89	11,896.00	2.10	448.13
ReLU2	576, 20	3.65	16,098.80	1.62	315.13
ReLU3	64, 16	3.70	1431.00	0.53	28.13
Maxpool1	$24 \times 24 \times 20, 2 \times 2$	10.56	12,681.60	4.23	236.63
Maxpool2	$24 \times 24 \times 16, 2 \times 2$	9.15	10,145.20	1.53	189.38
Maxpool3	$8 \times 8 \times 50, 4 \times 4$	5.77	5036.13	1.29	83.91

After theoretical and experimental analysis, verify that the proposed scheme has better performance in terms of computation time as well as communication complexity. Experiments under both security definitions are also compared and show that the performance of the malicious model is lower than that of the semi-honest model, but the performance improvement of the malicious model is higher than that of the semi-honest model compared to other works.

4.3. Security Prediction

Table 5 shows the number of communication bytes (KB) and the end-to-end latency (ms) for the proposed scheme to perform a single inference query with the Falcon framework. We execute the queries in different network architectures as well as in semi-honest and malicious settings. In the semi-honest setting, predicting a single sample takes about 45.71 ms with an improvement of $1.18\times$, and its communication cost is about 8.90 KB with an improvement of $7.12\times$ on Network-A. On Network-B, it takes about 39.54 ms with an improvement of $1.14\times$, and its communication cost is about 34.65 KB with an improvement of $7.23\times$. On Network-C, it takes about 32.36 ms ($1.47\times$), and its communication cost is about 324.02 KB ($7.62\times$). Network-D takes about 155.28 ms ($1.38\times$), and its communication cost is about 476.52 KB ($7.61\times$). In a malicious security setting, our work is $1.29\times$ – $1.80\times$ faster, with a communication efficiency improvement of about $32.36\times$ – $81.77\times$ compared to Falcon.

These experiments show that the proposed method has lower latency and fewer communication cost in single-sample prediction. Depending on the trust assumptions, malicious model requires more communication as well as higher runtimes compared to the semi-honest model. Figure 3 shows that the time performing batch prediction on different networks is roughly linear in the number of samples. Similarly, compared to the Falcon framework, our scheme performs better when predicting the same number of samples.

Table 5. Comparison of security predictions under different architectures.

Networks		FALCON		Proposed	
		Semi-Honest	Malicious	Semi-Honest	Malicious
Network-A	Time (ms)	54.01	78.81	45.71	61.20
	Comm (KB)	63.38	1090.87	8.90	13.34
Network-B	Time (ms)	45.02	75.56	39.54	47.60
	Comm (KB)	250.34	2144.46	34.65	52.42
Network-C	Time (ms)	47.56	77.21	32.36	42.98
	Comm (KB)	2467.58	15,733.20	324.02	486.19
Network-D	Time (ms)	214.83	266.90	155.28	194.38
	Comm (KB)	3626.88	25,292.3	476.52	714.94

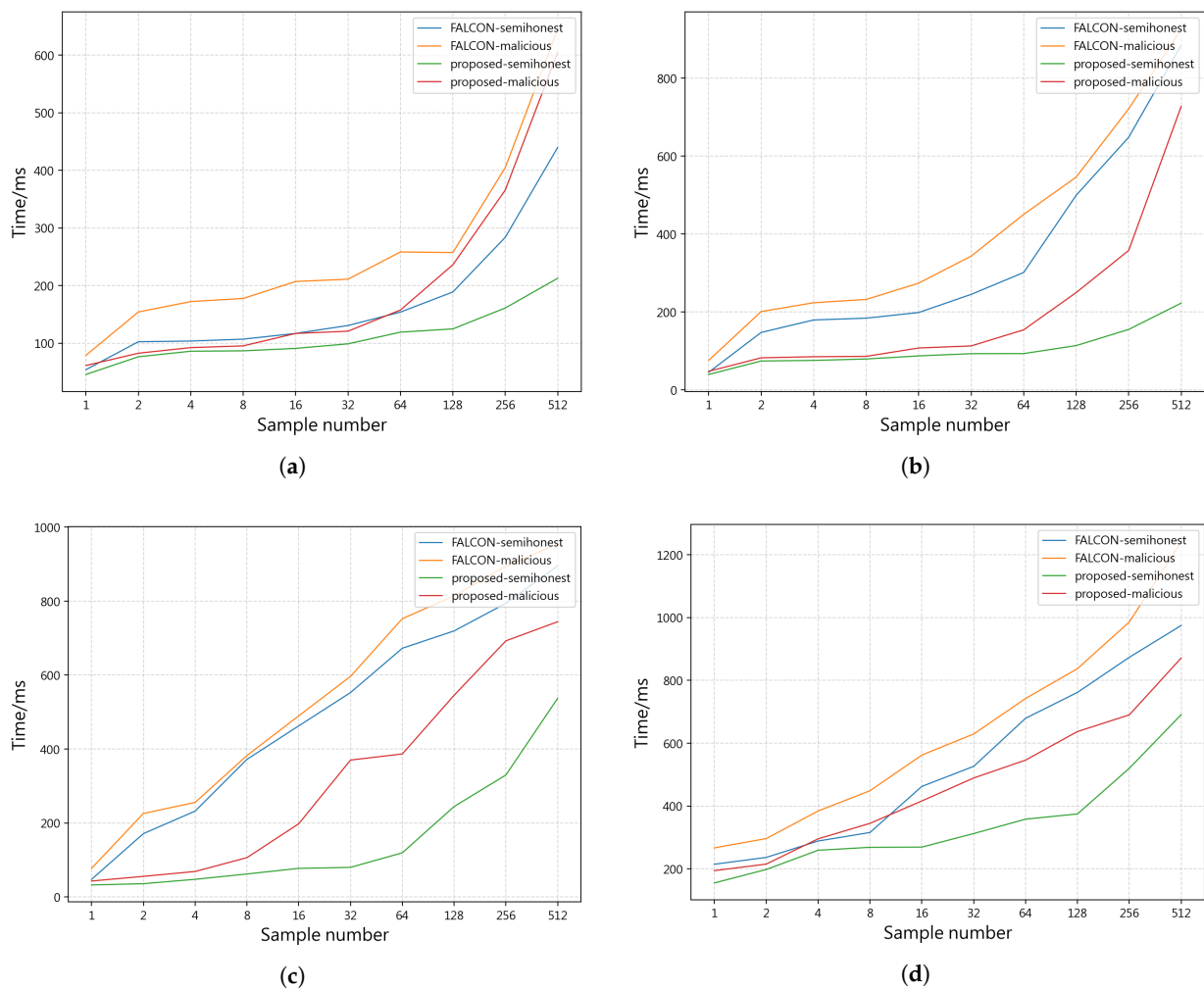


Figure 3. Prediction timings for batch size on Networks A-D over MNIST. (a) Network-A. (b) Network-B. (c) Network-C. (d) Network-D.

4.4. Comparison vs. Plaintext Computation

We perform experiments comparing secure prediction using secret sharing with traditional plaintext prediction using PyTorch. Results with the 64-bit floating-point data type on PyTorch (plaintext) and the 32-bit data type (secret sharing) are shown in Table 6. The second and third columns show the accuracy with the training set and test set obtained by setting the learning rate to 0.1 and iterating 15 times using traditional prediction methods. The fourth column is the accuracy of the test set under three-party secure computation. The fifth column is the error values for prediction accuracy, with differences all less than 1% be-

tween secure computation and plaintext computation. The results show that most networks have no/low loss in accuracy when the computation is performed as a fixed-point integer with the 13-bit precision in our work, confirming the effectiveness of the proposed scheme.

Table 6. Comparison of accuracy under secure prediction and plaintext.

Networks	Training Accuracy	Plaintext Inference	Security Inference	Relative Error
Network-A	98.06%	97.12%	96.64%	0.48%
Network-B	99.00%	97.90%	97.04%	0.86%
Network-C	99.56%	99.02%	98.70%	0.32%
Network-D	97.77%	97.22%	96.50%	0.72%

5. Conclusions

This paper proposes a neural network model for privacy prediction based on a 3PCthree-party secure computing framework. While traditional 3PCthree-party secure computing outsources all computations to three non-colluding servers, our three-party refers to the client, the service provider and the third-party server that assists in the computation. Because of the inclusion of the client in the computation, the definition of security in the model is slightly different than before. Therefore, depending on the different trust assumptions, this paper proposes two definitions of security to choose from, namely semi-honest and malicious security.

The introduction of the secret sharing technique ensures that the model parties do not have access to the original data of the client as well as the model parameters of the service provider, thus ensuring data security. Therefore, we give sub-protocols such as $\Pi_{Reconst}$, Π_{Mult} , Π_{MatMul} , Π_{Wrap} , Π_{RU} and Π_{MP} through the secret sharing technique. Compared to existing works, our scheme offers an improvement in both prediction time and communication cost.

The times reported in our experiments are all online times, and the performance of the framework would be further improved if the multiplicative triples computation is divided into the offline phase. As the entire codebase is parallelizable, improvements may be made later by parallelization or using GPUs in future work.

Author Contributions: Conceptualization, Y.Z. and P.L.; Formal analysis, Y.Z.; Funding acquisition, P.L.; Methodology, Y.Z.; Writing—original draft, Y.Z.; Writing—review & editing, P.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Natural Science Foundation of Hebei Province (Grant number: F2019502173), National Natural Science Foundation of China (Grant number: 61602173) and the Fundamental Research Funds for Central Universities (Grant number: 2019MS116).

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the editor and the anonymous reviewers for their valuable comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Li, H.X.; Lin, Z.; Shen, X.H.; Brandt, J.; Hua, G. A Convolutional Neural Network Cascade for Face Detection. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 5325–5334.
- Huang, H.; Wang, L.Y. Efficient privacy-preserving face verification scheme. *J. Inf. Secur. Appl.* **2021**, *63*, 103055. [[CrossRef](#)]
- Jiang, R.; Chazot, P.; Pavese, N.; Crookes, D.; Bouridane, A.; Celebi, M.E. Private Facial Prediagnosis as an Edge Service for Parkinson’s DBS Treatment Valuation. *IEEE J. Biomed. Health Inform.* **2022**, *26*, 2703–2713. [[CrossRef](#)]
- Trancoso, I.; Correia, J.; Teixeira, F.; Raj, B.; Abad, A. Analysing Speech for Clinical Applications. In Proceedings of the Statistical Language and Speech Processing, Mons, Belgium, 15–16 October 2018; Dutoit, T., Martín-Vide, C., Pironkov, G., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, pp. 3–6.
- Jadeja, Y.; Modi, K. Cloud computing—Concepts, architecture and challenges. In Proceedings of the 2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET), Nagercoil, India, 21–22 March 2012; pp. 877–880. [[CrossRef](#)]

6. Yao, A.C. Protocols for secure computations. In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982), Chicago, IL, USA, 3–5 November 1982; pp. 160–164. [\[CrossRef\]](#)
7. Patra, A.; Schneider, T.; Suresh, A.; Yalame, H. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In Proceedings of the 30th USENIX Security Symposium, USENIX Security 2021, Vancouver, BC, Canada, 11–13 August 2021.
8. Rathee, D.; Rathee, M.; Kumar, N.; Chandran, N.; Gupta, D.; Rastogi, A.; Sharma, R. CRYPTFLow2: Practical 2-Party Secure Inference. In Proceedings of the CCS '20: Proceedings of the 2020 ACM Sigsac Conference on Computer and Communications Security, Virtual Event, USA, 9–13 November 2020; pp. 325–342. [\[CrossRef\]](#)
9. Zhou, Z.P.; Fu, Q.; Wei, Q.J.; Li, Q. LEGO: A hybrid toolkit for efficient 2PC-based privacy-preserving machine learning. *Comput. Secur.* **2022**, *120*, 102782. [\[CrossRef\]](#)
10. Rouhani, B.D.; Riazi, M.S.; Koushanfar, F. DeepSecure: Scalable Provably-Secure Deep Learning. In Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 24–29 June 2018.
11. Bogdanov, D.; Laur, S.; Willemson, J. Sharemind: A Framework for Fast Privacy-Preserving Computations. In Proceedings of the 13th European Symposium on Research in Computer Security, Malaga, Spain, 6–8 October 2008; pp. 192–206.
12. Damgard, I.; Pastro, V.; Smart, N.; Zakarias, S. Multiparty Computation from Somewhat Homomorphic Encryption. In Proceedings of the 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012; pp. 643–662.
13. Duan, J.; Zhou, J.T.; Li, Y.M. Privacy-Preserving distributed deep learning based on secret sharing. *Inf. Sci.* **2020**, *527*, 108–127. [\[CrossRef\]](#)
14. Mohassel, P.; Rosulek, M.; Zhang, Y. Fast and Secure Three-party Computation: The Garbled Circuit Approach. In Proceedings of the CCS'15: Proceedings of the 22nd ACM Sigsac Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; pp. 591–602. [\[CrossRef\]](#)
15. Zhang, Z.X.; Lu, Q.; Xu, H.S.; Xu, G.B.; Kong, F.Y.; Yu, Y. Privacy-preserving deep learning for electricity consumer characteristics identification. *Front. Energy Res.* **2022**, *10*, 1273. [\[CrossRef\]](#)
16. Dalskov, A.; Escudero, D.; Keller, M. Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security. In Proceedings of the 30th Usenix Security Symposium, Vancouver, BC, Canada, 11–13 August 2021; pp. 2183–2200.
17. Koti, N.; Patra, A.; Rachuri, R.; Suresh, A. Tetrad: Actively Secure 4PC for Secure Training and Inference. *arXiv* **2021**, arXiv:2106.02850.
18. Mohassel, P.; Zhang, Y.P. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 19–38. [\[CrossRef\]](#)
19. Peikert, C.; Vaikuntanathan, V.; Waters, B. A framework for efficient and composable oblivious transfer. In Proceedings of the 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2008; pp. 554–571.
20. Demmler, D.; Schneider, T.; Zohner, M. ABY—A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS 2015), San Diego, CA, USA, 8–11 February 2015. [\[CrossRef\]](#)
21. Chandran, N.; Gupta, D.; Rastogi, A.; Sharma, R.; Tripathi, S. EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, 17–19 June 2019.
22. Agrawal, N.; Shamsabadi, A.S.; Kusner, M.J.; Gascon, A. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In Proceedings of the 2019 ACM Sigsac Conference on Computer and Communications Security (CCS'19), London, UK, 11–15 November 2019; pp. 1231–1247. [\[CrossRef\]](#)
23. Dowlin, N.; Gilad-Bachrach, R.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York, NY, USA, 19–24 June 2016.
24. Dathathri, R.; Saarikivi, O.; Chen, H.; Laine, K.; Lauter, K.; Maleki, S.; Musuvathi, M.; Mytkowicz, T. CHET: An Optimizing Compiler for Fully-Homomorphic Neural-Network Inferencing. In Proceedings of the 40th ACM Sigplan Conference on Programming Language Design and Implementation (PLDI '19), Phoenix, AZ, USA, 22–26 June 2019; pp. 142–156. [\[CrossRef\]](#)
25. Jiang, X.; Kim, M.; Lauter, K.; Song, Y. Secure Outsourced Matrix Computation and Application to Neural Networks. *Conf. Comput. Commun. Secur.* **2018**, *2018*, 1209–1222. [\[CrossRef\]](#)
26. Brutzkus, A.; Elisha, O.; Gilad-Bachrach, R. Low Latency Privacy Preserving Inference. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, CA, USA, 9–15 June 2019.
27. Sun, X.Q.; Zhang, P.; Liu, J.K.; Yu, J.P.; Xie, W.X. Private Machine Learning Classification Based on Fully Homomorphic Encryption. *IEEE Trans. Emerg. Top. Comput.* **2020**, *8*, 352–364. [\[CrossRef\]](#)
28. Yue, Z.J.; Ding, S.; Zhao, L.; Zhang, Y.T.; Cao, Z.H.; Tanveer, M.; Jolfaei, A.; Zheng, X. Privacy-preserving Time-series Medical Images Analysis Using a Hybrid Deep Learning Framework. *ACM Trans. Internet Technol.* **2021**, *21*, 57. [\[CrossRef\]](#)
29. Juvekar, C.; Vaikuntanathan, V.; Chandrakasan, A. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In Proceedings of the 27th Usenix Security Symposium, Baltimore, MD, USA, 15–17 August 2018; pp. 1651–1668.
30. Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.T.; Popa, R.A. DELPHI: A Cryptographic Inference Service for Neural Networks. In Proceedings of the 29th Usenix Security Symposium, Boston, MA, USA, 12–14 August 2020; pp. 2505–2522.

31. Araki, T.; Furukawa, J.; Lindell, Y.; Nof, A.; Ohara, K. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In Proceedings of the CCS'16: Proceedings of the 2016 ACM Sigsac Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 805–817. [[CrossRef](#)]
32. Furukawa, J.; Lindell, Y.; Nof, A.; Weinstein, O. High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority. In Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, 30 April–4 May 2017; pp. 225–255. [[CrossRef](#)]
33. Beaver, D. Efficient Multiparty Protocols Using Circuit Randomization. In Proceedings of the Advances in Cryptology—CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, CA, USA, 11–15 August 1991.
34. Mohassel, P.; Rindal, P. ABY(3): A Mixed Protocol Framework for Machine Learning. In Proceedings of the 2018 ACM Sigsac Conference on Computer and Communications Security (CCS'18), Toronto, ON, Canada, 15–19 October 2018; pp. 35–52. [[CrossRef](#)]
35. 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 ACM Asia Conference on Computer and Communications Security (Asiaccs'18), Incheon, Republic of Korea, 4–8 June 2018; pp. 707–721. [[CrossRef](#)]
36. Goldreich, O.; Micali, S.; Wigderson, A. How to play ANY mental game. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 25–27 May 1987.
37. Wagh, S.; Gupta, D.; Chandran, N. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* **2019**, *2019*, 26–49. [[CrossRef](#)]
38. Wagh, S.; Tople, S.; Benhamouda, F.; Kushilevitz, E.; Mittal, P.; Rabin, T. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proc. Priv. Enhancing Technol.* **2020**, *2021*, 188–208. [[CrossRef](#)]
39. Chaudhari, H.; Rachuri, R.; Suresh, A. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS 2020), San Diego, CA, USA, 23–26 February 2020. [[CrossRef](#)]
40. Byali, M.; Chaudhari, H.; Patra, A.; Suresh, A. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *Proc. Priv. Enhancing Technol.* **2019**, *2020*, 459–480. [[CrossRef](#)]
41. Barak, A.; Escudero, D.E.; Dalskov, A.; Keller, M. Secure Evaluation of Quantized Neural Networks. *Proc. Priv. Enhancing Technol.* **2019**, *2020*, 355–375.
42. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.L.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the 2018 Ieee/Cvf Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 2704–2713. [[CrossRef](#)]
43. Riazi, M.S.; Samragh, M.; Chen, H.; Laine, K.; Lauter, K.; Koushanfar, F. XONN: XNOR-based Oblivious Deep Neural Network Inference. In Proceedings of the 28th Usenix Security Symposium, Santa Clara, CA, USA, 14–16 August 2019; pp. 1501–1518.
44. Kushilevitz, E.; Lindell, Y.; Rabin, T. Information-Theoretically Secure Protocols and Security under Composition. *Siam J. Comput.* **2010**, *39*, 2090–2112. [[CrossRef](#)]
45. Chaudhari, H.; Choudhury, A.; Patra, A.; Suresh, A. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In Proceedings of the CCSW'19: Proceedings of the 2019 ACM Sigsac Conference on Cloud Computing Security Workshop, London, UK, 11 November 2019; pp. 81–92. [[CrossRef](#)]
46. Patra, A.; Suresh, A. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS 2020), San Diego, CA, USA, 23–26 February 2020. [[CrossRef](#)]
47. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [[CrossRef](#)]
48. Liu, J.; Juuti, M.; Lu, Y.; Asokan, N. Oblivious Neural Network Predictions via MiniONN Transformations. In Proceedings of the CCS'17: Proceedings of the 2017 ACM Sigsac Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 619–631. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.