Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Efficient privacy-preserving scheme for secure neural network inference*

Liquan CHEN†‡[1,2], Zixuan YANG[1], Peng ZHANG[1], Yang MA[1]

[1]*School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China*

[2]*Purple Mountain Laboratories, Nanjing 211189, China*

†E-mail: Lqchen@seu.edu.cn

Received May 8, 2024; Revision accepted Nov. 21, 2024; Crosschecked

Abstract: The increasing adoption of smart devices and cloud services, coupled with limitations in local computing and storage resources, prompts extensive users to transmit private data to cloud servers for processing. However, the transmission of sensitive data in plaintext form raises concerns regarding user's privacy and security. To address these issues, this study proposes an efficient privacy-preserving secure neural network inference scheme based on homomorphic encryption and secure multi-party computation, which ensures the privacy of both the user and the cloud server while enabling fast and accurate ciphertext inference. First, we divided the inference process into three stages, including the merging stage for adjusting the network structure, the preprocessing stage for performing homomorphic computations, and the online stage for floating-point operations on the additive secret sharing of private data. Second, we proposed an approach of merging network parameters, thereby reducing the cost of multiplication levels and decreasing both ciphertext–plaintext multiplication and addition operations. Finally, we proposed a fast convolution algorithm to enhance computational efficiency. Compared with other literature, our scheme reduces linear operation time in the online stage by at least 11%, significantly reducing inference time and communication overhead.

Key words: Secure neural network inference; Convolutional neural network; Privacy-preserving; Homomorphic encryption; Secret sharing

https://doi.org/10.1631/FITEE.2400371 **CLC number:** TP

1 Introduction

With the rapid development of smart devices, multimedia data such as images are playing an increasingly prominent role in various fields and many users choose to outsource their data to cloud servers for processing and storage (Chai et al., 2022). Although cloud servers have provided convenient services, they also have introduced additional security concerns. When users upload private data containing sensitive information to cloud servers, the servers may potentially access user data without the user's awareness, leading to the leakage of user privacy and sensitive data (Riazi et al., 2018; Ng and Chow, 2021). Additionally, the model parameters stored on cloud servers are also considered private, posing a risk of users inferring the model parameters, thus requiring protection (Chaudhari et al., 2019; Ma et al., 2021; Wang et al., 2022).

To protect user privacy and ensure cloud server security, homomorphic encryption technology for computations under ciphertext has emerged as a preferable solution (Li et al., 2020). Users encrypt their privacy data before uploading them to cloud servers for inference calculations, and the cloud server returns encrypted prediction results for de-

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. U22B2026) and ZTE Industry University Research Cooperation Project

ORCID: Liquan CHEN, https://orcid.org/0000-0002-7202-4939

cryption by users, thereby achieving inference while protecting the privacy of both users and servers (Liu et al., 2017). In recent years, convolutional neural networks (CNNs) have flourished, demonstrating their capability to extract more abstract and complex features from data (Schroff et al., 2015) and thereby further enhancing inference accuracy. This advancement has led to its widespread applications in encrypted image prediction (Li et al., 2024). Dowlin et al. (2016) proposed CryptoNets, which encrypt multiple inputs into a single ciphertext for parallel computation, replacing max pooling with average pooling and employing the square function as the activation function, marking the first instance of utilizing CNN for encrypted image prediction. Hesamifard et al. proposed CryptoDL (Hesamifard et al., 2017), which replaced the activation functions in neural network models with polynomial functions and provided mathematical expressions for low-degree polynomial approximation activation functions. Chou et al. (2018) devised Faster CryptoNets, which further accelerated inference speed by integrating neural network pruning methods to reduce the parameter count in the original model. Chabanne et al. (2017) attempted to integrate batch normalization (BN) layers commonly used in deep learning with existing encryption schemes, effectively deepening the network layers. Ishiyama et al. (2020) proposed a scheme for normalizing inputs using BN layers and approximating Swish and ReLU with polynomial functions. They mitigated multiplication levels to reduce the multiplication cost in the ciphertext domain by preprocessing the coefficients of the highest-order terms in the polynomial. They provided evaluation results for different fitting intervals.

However, despite the advancements made by the aforementioned schemes, which leverage polynomial approximation and square function for non-linear operations, they often suffer from a notable drawback: the loss of inference accuracy (Li et al., 2020; Lou et al., 2020; Jha et al., 2021). In response to this challenge, the integration of secure multi-party computation for encrypted image inference has emerged as a promising solution (Shen et al., 2020). Juvekar et al. (2018) proposed the GAZELLE system, which improves algorithms to enhance inference efficiency while maintaining accuracy. GAZELLE proposes innovative convolution algorithms and matrix-vector multiplication algorithms for CNNs. Additionally, it integrates garbled circuits with homomorphic encryption to achieve secure neural network inference. Building on GAZELLE, Srinivasan et al. (2020) introduced DELPHI , which shifts heavy homomorphic encryption operations to the preprocessing stage and combines additive secret sharing technology to ensure inference security, significantly enhancing the efficiency of secure neural network inference in the online stage.

Although these methods ensure accuracy during inference, GAZELLE still requires long online ciphertext inference time, whereas DELPHI reduces user online waiting time but demands frequent interaction with the cloud server. Therefore, to alleviate user computational time and communication overheads, we proposed an efficient secure neural network inference scheme. Our scheme takes advantage of the benefits of homomorphic Single Instruction Multiple Data (SIMD) and efficient convolution algorithms, providing superior computational efficiency compared to other methods. Our main contributions are as follows:

1. We present improvements to the scheme of DELPHI by proposing a network parameters merging approach, targeting consecutive multi-layer linear operations. This approach merges the parameters of multiple layers from the preprocessing stage and the online stage into a single layer for computation, effectively reducing communication overhead and inference time.

2. We propose a fast convolution algorithm to address the heavy convolutional computations in the preprocessing stage. This method transforms homomorphic convolutional calculations into homomorphic matrix-vector multiplication calculations, adopting and improving diagonal schemes for convolutional calculations, significantly reducing the computational time in the preprocessing stage.

3. The efficiency of the proposed scheme is validated by the experimental analysis. Compared to the prior works, our scheme reduces linear operation time in the online stage by at least 11%, under the proposed network model. Additionally, the network parameters merging approach we proposed significantly improves both the computational time and communication overhead for consecutive linear layer calculations in the online stage, compared to the nonmerging method. Furthermore, the proposed

Table 1 Symbol descriptions

Symbol	Description
M_i	The weight of the i^{th} layer in the server model.
b_i	The bias of the i^{th} layer in the server model.
x_i	The private information input by the user at layer i.
$X^{IC}_{f_{Size}}$	Input matrix with convolutional kernel of size f_{Size} and input channels of IC.
Y^{OC}	Output vector Y with output channels of OC.
$W^{IC,OC}_{f_{Size}}$	Matrix with convolutional kernel of f_{Size}, input channels of IC, and output channels of OC.
$w^{j,k}_i$	The i^{th} plain vector at the j^{th} input channel and the k^{th} output channel of the convolutional kernel.
x^j_i	The i^{th} cipher vector extracted from the input matrix at the j^{th} input channel.

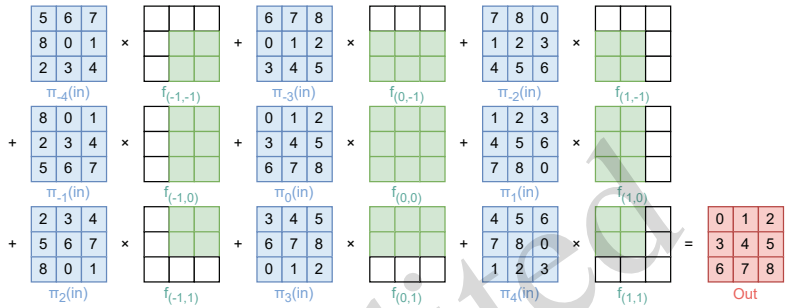


Fig. 1 Packed SC: Packed SISO convolution in GAZELLE.

fast convolution algorithm demonstrates a certain degree of improvement, compared to both the improved naive method and the GAZELLE convolution algorithm.

The remaining sections of this study are organized as follows: Section 2 introduces the preliminaries of this study. Section 3 elaborates on the proposed inference scheme. Section 4 describes the experimental design and presents the analysis of results. Finally, Section 5 concludes the study.

2 Preliminaries

For clarity, Table 1 summarizes some symbols used in this study along with their corresponding descriptions.

2.1 GAZELLE's homomorphic linear algorithm

Convolution operations commonly employ two padding schemes. In the "valid" scheme, no input padding is applied. In this case, the output size is smaller than the input size. In contrast, the "same" scheme involves zero-padding the input to ensure that the output size matches that of the input.

Juvekar et al. (2018) proposed two single-input single-output (SISO) convolution algorithms for computing convolutions in the "same" padding mode, where the input and the output are of equal size. The first one is the padded SISO convolution (Padded SC) algorithm, which generates an output matrix consistent in size with the input matrix by padding the outer edges of the input matrix with zeros. However, it suffers from the wastage of ciphertext "slots" due to padding with zeros.

Addressing this issue, Juvekar et al. (2018) introduced the second method, the packed SISO convolution (Packed SC) algorithm, which tightly packs and rotates the inputs to avoid wastage of ciphertext "slots" and multiply these rotated ciphertexts with plaintexts that have zeros in the appropriate locations as shown in Fig. 1 to reach the result.

However, both methods only support convolution under the "same" padding mode and exclusively for strides equal to 1. For computations with strides ≥ 1, the convolution operation needs to be decomposed into simple convolutions whose strides are equal to 1, resulting in additional homomorphic

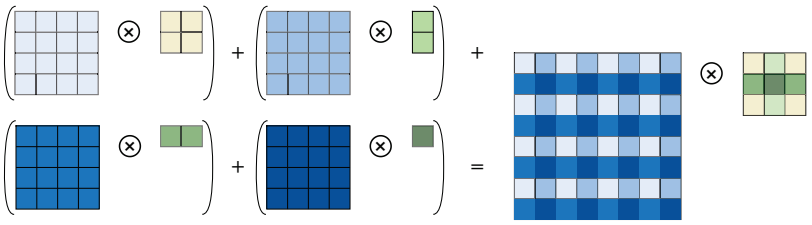


Fig. 2 Convolutional computation with strides set to 2 in GAZELLE.

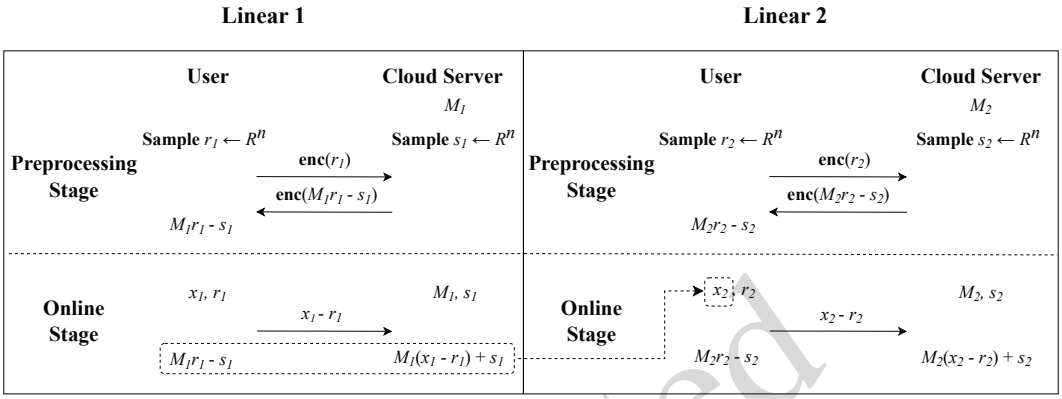


Fig. 3 Consecutive two-layer linear inference in DELPHI.

additions. Fig. 2 illustrates the implementation of a strided convolution with a $3 * 3$ kernel over an $8 * 8$ input matrix with strides set to 2.

For fully connected layers, Juvekar et al. (2018) proposed a hybrid approach for matrix-vector multiplication, combining both the original method and the diagonal method, which can handle cases where input and output sizes are inconsistent.

2.2 DELPHI's protocol

DELPHI's protocol offloads the heavy cryptographic operations in CNN to the preprocessing stage to alleviate the waiting time for users during the online stage. However, DELPHI requires users to communicate with the cloud server at every layer of the CNN.

Before communication, both the user and the cloud server pick random masking vectors r_i and s_i in i^{th} layer, where r_i and s_i are sampled from R^n.

In DELPHI's scheme, during the preprocessing stage of each layer, the computed result obtained by the user is $M_i r_i - s_i$, which constitutes an additive secret sharing of the linear layer computational result $M_i(x_i - r_i) + s_i$, along with the result received by the cloud server during the online stage.

In additive secret sharing (Koti et al., 2021), both the user and the cloud server hold additive secret sharing related to x, denoted as r and $x - r$, respectively. The calculation results of each layer are obtained by summing the secret sharing held by both the user and the cloud server. Through the additive secret sharing of computational results, neither the user nor the cloud server can get knowledge of the other's private data, thereby achieving dual protection of user and cloud server privacy. Fig. 3 illustrates the inference process of the DELPHI protocol for two consecutive linear layers. The input to linear layer 2 takes the form of additive secret sharing, which is constituted by the computational results obtained by the user during the preprocessing stage and the computational results from the cloud server during the online stage.

3 The proposed scheme

3.1 System setup

Our system contains two parties involved as follows: the user, who owns the private images, encrypts all the images and uploads them to the cloud server for prediction; and the cloud server, which

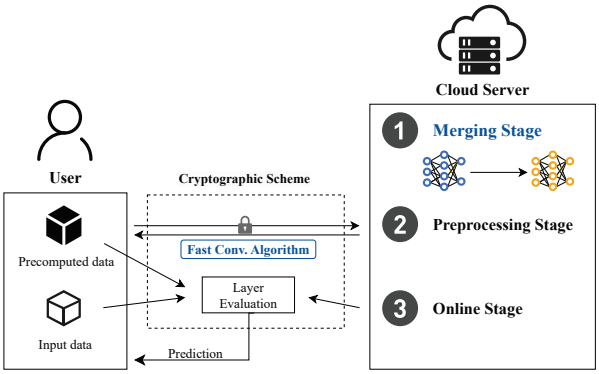


Fig. 4 System framework.

owns the neural network model parameters, predicts the ciphertext images uploaded by the user using certain neural network, and sends the encrypted prediction results back to the user.

Fig. 4 illustrates the detailed framework of our system. Our system comprises the following three stages: the merging stage, the preprocessing stage, and the online stage.

Before prediction, the cloud server adjusts the CNN network structure during the merging stage by merging consecutive linear layers to obtain the merged neural network structure and parameters. This is primarily done to reduce the number of communication rounds between the user and the cloud server to decrease communication costs.

To reduce the computational cost of online operations and decrease the user's waiting time, we move heavy cryptographic operations on ciphertexts to the preprocessing stage. This movement stems from the cloud server's prior knowledge of the input M_i before the transmission of the user's private data. During the preprocessing stage, users utilize the fast convolution algorithm to compute precomputed data, which then becomes part of the additive sharing for the final computational results.

In the online stage, transmission and computation can be performed directly over the secret sharing of the user's input data without encryption, significantly reducing both online communication and online computational costs. Additionally, the computational results of the online stage and precomputed data, which are the computational results of the preprocessing stage, form the additive secret sharing of the prediction result, with neither the user nor the cloud server knowing each other's private data, protecting both parties' privacy.

Our proposed scheme significantly reduces the computational time during the online stage when neural networks contain consecutive linear layers. In contrast, for neural network models that do not possess consecutive linear layers, the efficiency improvement of our scheme is relatively limited, as it merely reduces the convolutional computational time by utilizing the fast convolution algorithm.

3.2 Cryptographic scheme

Our scheme is based on DELPHI's protocol and focuses on the scenario involving consecutive multiple linear layers. Taking two layers as an example, in DELPHI's protocol, when computing two consecutive linear layers, after the computation of the first layer, the user and the cloud server, respectively, possess additive secret sharing concerning the computational result of the first layer, $M_1x + b_1$. Through communication interactions, the user obtains the result y, which serves both as the output of the first linear layer and the input for the second linear layer. After the computation of the second layer, the user and the cloud server, respectively, hold additive secret sharing concerning the second layer's result, $M_2y + b_2$. In DELPHI's protocol, two consecutive linear layers are computed sequentially, requiring two communications between the user and the cloud server during the computational process.

To mitigate communication overhead and computational burden, we proposed a ciphertext inference scheme that merges consecutive linear layers. In our scheme, for the computation of consecutive multiple linear layers, we merged them into one layer to reduce communication costs and computational time. Taking two layers as an example, Fig. 5 illustrates the detailed proposal for merging two consecutive linear layers.

First, in the merging stage before inference, the user and the cloud server negotiate the algorithm parameters and keys for homomorphic encryption and merge linear layers to optimize the network structure. Subsequently, the optimized network parameters are computed and encoded. Following this, the inference computation is conducted, and we divided the linear operation process into the following two stages: the preprocessing stage and the online stage.

During the preprocessing stage, the user encrypts r_1 using a homomorphic encryption algorithm and sends it to the cloud server. The cloud

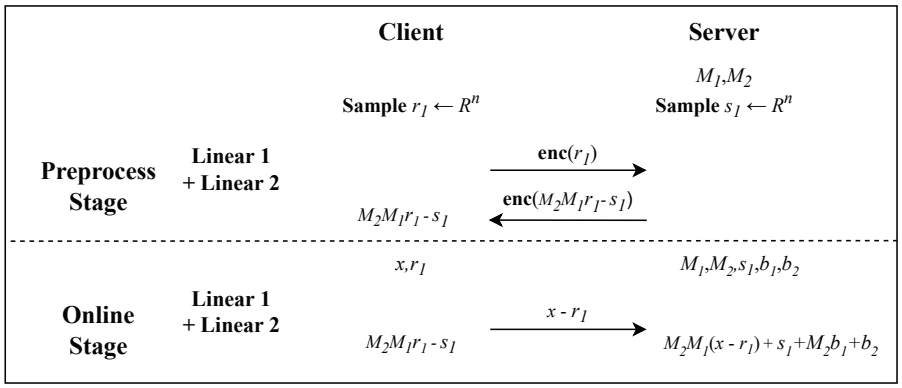


Fig. 5 Inference scheme of merging two consecutive linear layers.

server utilizes homomorphic operations to obtain the encrypted computational result $enc(M_2M_1r_1 - s_1)$, which is then transmitted to the user. On decryption, the user obtains $M_2M_1r_1 - s_1$.

In the online stage, the user sends $x - r_1$ to the cloud server. At this point, both the user and the cloud server possess an additive secret sharing concerning private data x. The cloud server computes the inference result of $x - r_1$ as $M_2M_1(x - r_1) + s_1 + M_2b_1 + b_2$. The additive secret sharing held by the user from the preprocessing stage, $M_2M_1r_1 - s_1$, combined with the cloud server's result, constitutes the additive secret sharing of the output result of the two-layer linear layers, $M_2(M_1x + b_1) + b_2$.

In addressing the nonlinear operations, unlike the polynomial approximations utilized in the studies by Chabanne et al. (2017), Hesamiard et al. (2017), Chou et al. (2018), Ishiyama et al. (2020), Xie et al. (2022), Kim and Guyot (2023) and Wang et al. (2023), we employed garbled circuits akin to Juvekar et al. (2018). The given encoded circuit is applied to the results held by both the user and the cloud server for evaluation, to obtain the additive secret sharing of the linear layer's output result, $M_2(M_1x+b_1)+b_2$, while ensuring that neither party can get knowledge of information about the circuit or the inputs of the other.

3.3 Network parameters merging

We used the Cheon-Kim-Kim-Song (CKKS) algorithm as the homomorphic encryption algorithm in our scheme, which is an approximate numerical homomorphic encryption scheme that is capable of performing high-precision computations on floating-point numbers (Cheon et al., 2017, 2019). To decrease the cost of CKKS multiplication levels during the preprocessing phase, thereby mitigating computational time, we proposed a network parameters merging approach. During the merging stage, while encoding the parameters of the trained network model, the cloud server combines the parameters of consecutive linear layers into a single layer.

Taking two layers as an example, Table 2 illustrates the ciphertext–plaintext multiplication operations (C–P Multi. Op.), ciphertext–plaintext addition operations (C–P Add. Op.) and floating-point operations required for two consecutive linear layers in our merging approach and nonmerging approach. In our scheme, we employed the merging approach; whereas, other schemes, such as DELPHI, utilized the nonmerging approach. In our inference scheme described in Section 3.2, our approach of merging network layer parameters manifests as follows: during the merging stage, the cloud server computes $M_2M_1$and $s_1 + M_2b_1 + b_2$, encoding M_2M_1 subsequently. After merging network parameters, the cloud server reduces one C–P multiplication and one C–P addition compared to the nonmerging approach. In the online stage, only one floating-point multiplication and addition operation is necessary, further diminishing the computational load. In the preprocessing stage, although our merging approach results in an additional floating-point multiplication, the computational overhead of this added multiplication is significantly lower than the reduction in the time required for homomorphic operations. Consequently, our merging approach demonstrates a substantial improvement in computational efficiency compared

Table 2 Numbers of multiplication and addition operations in two consecutive linear layers

	Preprocessing stage	C-P operations	Online stage	Floating-point operations
Nonmerging approach (DELPHI)	$M_1 r_1 - s_1$; $M_2 r_2 - s_2$	2 Multi. Op.; 2 Add. Op.	$M_1(x_1 - r_1) + (b_1 + s_1)$; $M_2(x_2 - r_2) + (b_2 + s_2)$	2 Multi. Op.; 2 Add. Op.
Merging approach	$M_2 M_1 r_1 - s_1$	1 Multi. Op.; 1 Add. Op.	$(M_2 M_1)(x - r_1)$ $+(s_1 + M_2 b_1 + b_2)$	1 Multi. Op.; 1 Add. Op.

to the nonmerging approach.

We considered two consecutive layers as a convolutional layer followed by a BN layer, where the output $y_i = \sum kx+b$ of the convolutional layer serves as the input to the BN layer. In the BN layer, the normalized output $\hat{y_i}$ is computed from the input x_i, the mean μ_B, and the variance σ_B^2 of each batch of data (Ioffe and Szegedy, 2015). Additionally, to address the issue of gradient dispersion, the output is to be multiplied by the weight γ and then added by the bias β. The description of the BN layer is as follows:

$$\begin{cases} \mu_B \longleftarrow \dfrac{1}{m}\sum\limits_{i=1}^{m} x_i \\ \sigma_B^2 \longleftarrow \dfrac{1}{m}\sum\limits_{i=1}^{m}(x_i - \mu_B)^2 \\ \hat{x_i} \longleftarrow \dfrac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \\ BN_{\gamma,\beta}(x_i) \equiv \hat{y_i} \longleftarrow \gamma \hat{x_i} + \beta \end{cases} \tag{1}$$

In this case, the parameters of the first linear layer are $M_1 = k$ and $b_1 = b$, and the parameters of the second linear layer are $M_2 = \frac{\gamma}{\sqrt{\sigma_B^2+\varepsilon}}$ and $b_2 = \beta - \frac{\gamma\mu_B}{\sqrt{\sigma_B^2+\varepsilon}}$. According to the scheme proposed in Section 3.2 of this study, after two layers of computation, the user holds $\frac{k\gamma}{\sigma_B^2+\varepsilon} r_1 - s_1$, and the cloud server holds $\frac{k\gamma}{\sqrt{\sigma_B^2+\varepsilon}}(x-r_1) + s_1 + \frac{b\gamma}{\sigma_B^2+\varepsilon} + \beta - \frac{\gamma\mu_B}{\sqrt{\sigma_B^2+\varepsilon}}$.

During the merging stage, the cloud server computes the parameters $\frac{k\gamma}{\sigma_B^2+\varepsilon}$ for M_2M_1 and $s_1 + \frac{b\gamma}{\sigma_B^2+\varepsilon} + \beta - \frac{\gamma\mu_B}{\sqrt{\sigma_B^2+\varepsilon}}$ for $s_1 + M_2 b_1 + b_2$ and then encodes $\frac{k\gamma}{\sigma_B^2+\varepsilon}$. Thus, the cloud server can directly perform homomorphic multiplication and addition of the pre-calculated parameters $\frac{k\gamma}{\sigma_B^2+\varepsilon}$ with the ciphertext random vector r_1 and s_1 in the preprocessing stage, decreasing one homomorphic multiplication and addition operation, thereby reducing server's computational time in this stage. Moreover, in the online stage, the cloud server can directly perform floating-point multiplication of the merged parameters $\frac{k\gamma}{\sigma_B^2+\varepsilon}$ with $x - r_1$. Then, it adds the multiplication result to the pre-merged parameters $s_1 + \frac{b\gamma}{\sigma_B^2+\varepsilon} + \beta\frac{\gamma\mu_B}{\sqrt{\sigma_B^2+\varepsilon}}$ without intermediary steps, thereby reducing the computational time in the online stage.

3.4 Fast convolution algorithm

The preprocessing stage of inference involves heavy cryptographic operations, with convolutional calculations in the linear layers occupying a significant portion of the computational time. To reduce the computational time in the preprocessing stage and consequently decrease the overall inference time, we proposed a fast convolution algorithm based on this premise.

We divided the implementation of the fast convolution algorithm into three steps. The first one is the kernel processing step, which is implemented during the merging stage of inference, where the convolutional kernels are processed and encoded. The other two steps are the transformation step and the computational step, which are implemented during the preprocessing stage.

Due to the fast matrix-vector multiplication algorithm's ability to fully utilize ciphertext "slots" and leverage homomorphic SIMD features for vector batch processing, thereby enhancing computational efficiency, we designed a transformation step that converts the multiplication operations in the convolutional layer into matrix-vector multiplications. This method not only improves the computational efficiency of the convolutional layer but also supports fast convolutional computations in scenarios with "padding" modes and strides ≥ 1.

In the case of computing two-dimensional convolution, let r_i and c_i denote the number of rows and columns of the input matrix, respectively, and f_w and f_h denote the number of rows and columns of the convolutional kernel, respectively. Let r_{pad} and c_{pad} denote the number of rows and columns of the zeros padded to ensure the same size of output and input, respectively. In particular, we have the

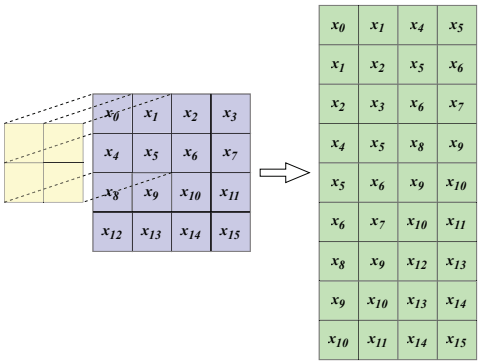


Fig. 6 Transformation step.

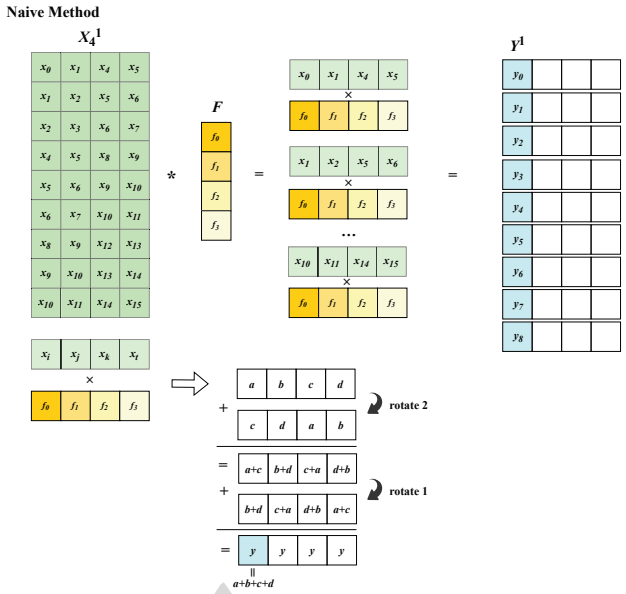


Fig. 7 Naive matrix-vector multiplication.

following:

$$\begin{aligned} r_{pad} &= \lfloor (f_w - 1)/2 \rfloor \\ c_{pad} &= \lfloor (f_h - 1)/2 \rfloor \end{aligned} \tag{2}$$

Let strides be str, and let r_o and c_o denote the number of rows and columns of the output matrix, respectively, which are calculated with the following formula:

$$\begin{aligned} r_o &= \lfloor (r_i - f_w + 2 * r_{pad})/str \rfloor + 1 \\ c_o &= \lfloor (c_i - f_h + 2 * c_{pad})/str \rfloor + 1 \end{aligned} \tag{3}$$

In the transformation step, a window of size $f_{Size}(f_{Size} = f_w * f_h)$ is first used to slide over the input matrix with a stride of str. During each slide of the window, the elements covered by the window in the input matrix are extracted and unfolded row-wise, generating $O_{Size}(O_{Size} = r_o * c_o)$ numbers of plaintext vectors, each of size f_{Size}. These plaintext vectors constitute the result matrix $X^1_{f_{Size}}$ of the transformation step, where the column number of columns of $X^1_{f_{Size}}$ is f_{Size}, and the number of rows is O_{Size}. The output matrix $X^1_{f_{Size}}$ of the transformation step serves as the input matrix for the computational step. Fig. 6 illustrates the operations performed in the transformation step of the fast convolution algorithm, using a $4 * 4$ input matrix and a $2 * 2$ convolutional kernel as an example.

After the transformation step, the convolution operation is converted into a matrix-vector multiplication.

In the naive matrix-vector multiplication approach (Juvekar et al., 2018), in the kernel processing step, the convolutional kernel is flattened row-wise to obtain a plaintext vector of size f_{Size}, which is then padded with zeros to make its length a power of 2, encoded, and denoted as F. The calculation step encrypts each row of the input matrix $X^1_{f_{Size}}$ into separate ciphertext vectors, with each ciphertext vector having a length of f_{Size} for the valid items, and is padded with zeros to match the length of F. Subsequently, each ciphertext vector is subjected to a homomorphic multiplication operation with F, and the O_{Size} numbers of results for multiplication are individually rotated and summed. Convolution result Y^{OC} is the result in the first "slot" of the corresponding ciphertexts. Fig. 7 illustrates the operations performed in the naive approach using a $4 * 4$ input matrix and a $2 * 2$ convolutional kernel as an example.

However, the naive approach entails multiple homomorphic multiplication operations and addition operations, as well as a waste of a considerable number of ciphertext "slots," resulting in an increased computational time. Therefore, an improved naive approach involves placing ciphertext vectors and multiple F separately in the unused ciphertext "slots" after the length of F in the ciphertext and

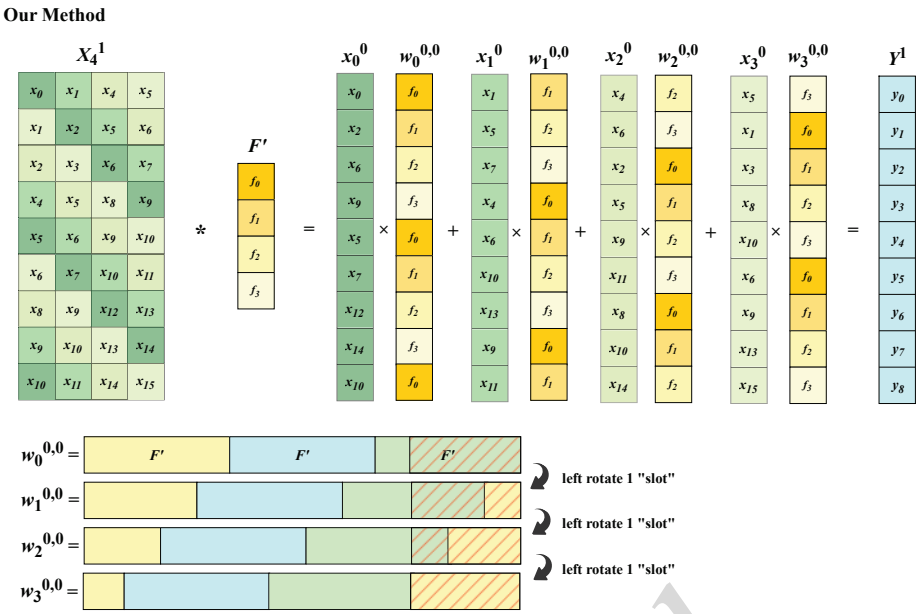


Fig. 8 Kernel processing and computational steps in fast convolution algorithm.

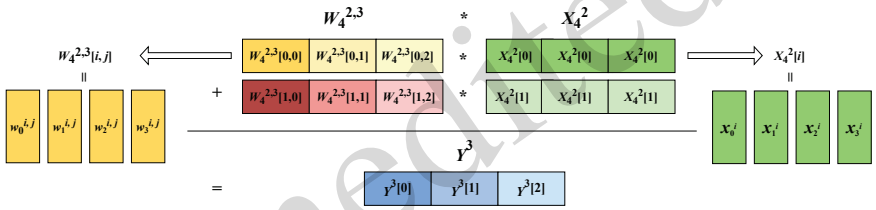


Fig. 9 Multidimensional fast convolution.

plaintext for computation. In the resulting ciphertext, the convolution result y_i intervals every length of F "slots." However, even with this improvement, the improved naive approach still suffers from the issue of unused ciphertext "slots" due to its zero-padding and rotating summation, which can lead to a decrease in computational efficiency, especially with larger input matrices.

We proposed a fast convolution algorithm that maximizes the utilization of ciphertext "slots" while reducing the number of homomorphic operations. The algorithm adopts the diagonal scheme for matrix-vector multiplication (Halevi and Shoup, 2014) for convolutional computation and improves it to enhance computational efficiency and maintain the same input-output format, facilitating direct computation in subsequent network layers. In the kernel processing step, the convolutional kernel is unfolded into a plaintext vector of size f_{Size}, denoted as F'. To fully utilize the ciphertext "slots" and ensure the correctness of the computational results, F' is duplicated T times and placed into $w_0^{0,0}$, where

$$T = \lceil (O_{Size}/f_{Size}) \rceil \tag{4}$$

Each subsequent $w_i^{0,0}$ undergoes a left rotation operation relative to $w_0^{0,0}$ with a step size of i. The top O_{Size} items are selected as valid items and encoded. In the computational step, to minimize the waste of ciphertext "slots" and to conserve the extraction of the result from the first "slot" of the ciphertext, which is necessary in traditional schemes, $X^1_{f_{Size}}$ is extracted diagonally and placed into the vector $x_t^i (t = 0, 1, ..., f_{Size} - 1)$ before encryption. The i^{th} element of x_t^0 corresponds to the i^{th} row and j^{th} column element of $X^1_{f_{Size}}$, where

$$t = (j - (i \mod f_{Size})) \mod f_{Size} \tag{5}$$

Finally, the result of the homomorphic multiplication between x_i^0 and $w_i^{0,0}$ is accumulated to obtain the final convolution result Y^1. Fig. 8 illustrates the

Table 3 Network description

	Layer	Description
1	Conv I	Input: $28*28*1$, kernel: $5*5*1$, strides: (1,1), filters: 16, and output: $24*24*16$
2	Activation I	Calculates ReLu for each input
3	Maxpool I	Pool size: (2,2) and strides: (2,2)
4	Conv II	Input: $12*12*16$, kernel: $5*5*16$, strides: (1,1), filters: 16, and output: $8*8*16$
5	BN	Input: $8*8*16$, and output: $8*8*16$
6	Activation II	Calculates ReLu for each input
7	Maxpool II	Pool size: (2,2) and strides: (2,2)
8	Flatten	Multidimensional arrays are expanded to one-dimensional arrays
9	FC I	Fully connects the incoming 256 nodes to the outgoing 100 nodes
10	Activation3	Calculates ReLu for each input
11	FC II	Fully connects the incoming 100 nodes to the outgoing 10 nodes
12	Softmax	/

Table 4 Inference accuracy of different models

Model	MINST accuracy (%)	Fashion-MNIST accuracy (%)
Ours	99.24	90.26
CryptoNets (Dowlin et al., 2016)	98.95	87.27
GAZELLE (Juvekar et al., 2018)	99.08	90.26
DELPHI (Srinivasan et al., 2020)	99.08	90.26
Approximated Mish (Yagyu et al., 2022)	99.30	84.68
Approximated SiLU (Lai et al., 2024)	97.17	89.35

operations performed in the kernel processing and computational steps of the proposed fast convolution algorithm using a $4*4$ input matrix and a $2*2$ convolutional kernel as an example.

In the case of computing multidimensional convolution, our proposed approach tightly packs the computational results across different output dimensions. We placed multiple $W^{IC,OC}_{f_{Size}}[i,j]$ corresponding to the same input dimension i but different output dimensions j into the same plaintext. Moreover, we replicated $X^{IC}_{f_{Size}}[i]$ with the same input dimension i multiple times and placed them into the same ciphertext for computation, aiming to reduce the number of computations for multi-dimensional convolutions and consequently decrease the computational time. The resulting computational outcome Y^{OC} represents the output of multi-dimensional fast convolution. Fig. 9 illustrates the operations of the multi-dimensional fast convolution algorithm, taking a $2*2$ convolutional kernel with input dimension 2 and output dimension 3 as an example. In Fig. 9, $W_4^{2,3}[i][j]$ encompasses 4 plaintext vectors $w_t^{i,j}(t=0,1,2,3)$ corresponding to input channel i and output channel j, processed during the kernel processing step. $X_4^2[i]$ comprises 4 ciphertext vectors $x_t^i(t=0,1,2,3)$ corresponding to input channel i, while $Y^3[j]$ contains the computational results Y^j from output channel 0 to j.

4 Experiment

In this section, we conducted experiments on the MNIST and Fashion-MNIST datasets to evaluate our proposed method and compared its inference performance with other methods presented in this study.

4.1 Experimental setup

All experiments were implemented using C++ language on a server equipped with an Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz.

In our proposed approach, we employed the CKKS algorithm for homomorphic encryption operations and implemented it using the Microsoft SEAL 3.3.1 library on the Visual Studio 2017 platform.

We conducted experiments using the MNIST and Fashion-MNIST datasets to evaluate our proposed approach. MNIST is a grayscale image dataset of handwritten digits, consisting of 60,000 images for training and 10,000 images for testing. The images in MNIST are of size $28*28$ pixels and cover a total of 10 classes from digit 0 to 9. Fashion-MNIST is

Table 5 Comparison of inference time for MNIST dataset

	Time in preprocessing stage (ms)		Time in online stage (ms)	
	Linear option	Nonlinear option	Linear option	Nonlinear option
Our Scheme	2059.10	1541.34	1.39	894.52
DELPHI (Mishraetal., 2020)	2089.51	1541.34	1.56	894.52
GAZELLE (Juvekaretal., 2018)	/	/	2089.51	2657.64
CryptoNets (Gilad-Bachrachetal., 2016)	/	/	264970.28	/

Table 6 Comparison of inference time for Fashion-MNIST dataset

	Time in preprocessing stage (ms)		Time in online stage (ms)	
	Linear option	Nonlinear option	Linear option	Nonlinear option
Our Scheme	2406.4	1789.82	1.47	1038.73
DELPHI (Mishraetal., 2020)	2411	1789.82	1.67	1038.73
GAZELLE (Juvekaretal., 2018)	/	/	2411	3066.56
CryptoNets (Gilad-Bachrachetal., 2016)	/	/	315,674.1	/

a grayscale image dataset of clothing items. Compared to the MNIST dataset, it presents a higher level of complexity, thereby offering a more accurate assessment of the neural network's performance.

Based on our optimization approach, we devised a neural network for feature extraction from the MNIST and Fashion-MNIST datasets. The specific network description is outlined in Table 3.

In the ciphertext inference phase, the network model we constructed requires five multiplication operations during the linear layer computation to obtain feature vectors. However, by adopting our proposed approach in this study, merging the Conv II layer with the BN layer can optimize the number of multiplication operations to four.

4.2 Accuracy and inference time analysis

Table 4 shows the comparison of different models' accuracy in ciphertext inference. Compared with other models, the network model we constructed exhibits high accuracy under ciphertext conditions. For the MNIST and Fashion-MNIST datasets, the model achieved inference accuracies of 99.24% and 90.26%, respectively. Among the 10,000 test images, none were misclassified due to encrypted computations.

To evaluate our inference scheme, we utilized the testing images in the MNIST and Fashion-MNIST datasets as the testing dataset and compared the inference time of our approach with DELPHI (Mishra et al., 2020), GAZELLE (Juvekar et al., 2018), and CryptoNets (Dowlin et al., 2016). The network model used is built in Section 4.1. Tables 5 and 6, respectively, present the linear and nonlinear operation times for each inference scheme during the preprocessing and online stages on the MNIST and Fashion-MNIST datasets. In our inference scheme, DELPHI and GAZELLE, the time for linear operations in the preprocessing stage and the online stage is composed of the computational time of convolutional layers, BN layers, and fully connected layers. As CryptoNets transforms non-linear operations into linear computations, its linear operation time accounts for the entire inference process. For linear operations in the online stage, our scheme improves computational efficiency by approximately 11% compared to DELPHI and outperforms both GAZELLE and CryptoNets. Additionally, for the non-linear operations in the preprocessing stage, our optimization scheme also shows a slight improvement compared to the DELPHI approach.

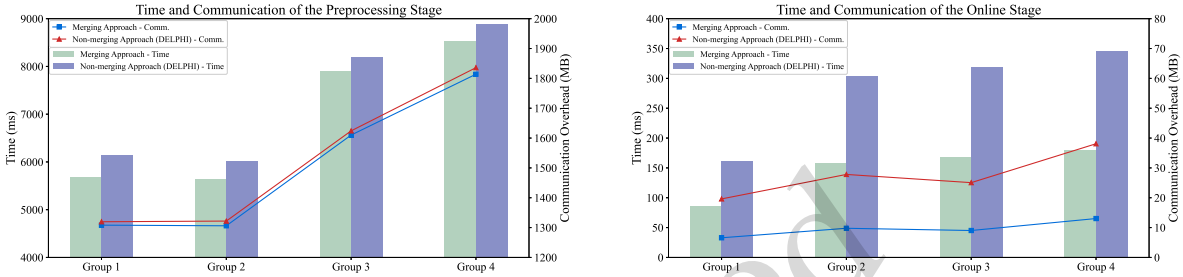
4.3 Efficiency analysis

We analyzed the efficiency of the network parameters merging approach proposed in Section 3.3 and compared it with the nonmerging approach. Taking consecutive calculations of convolution and BN as an example, Groups 1–4 in Table 7 represent the convolutional layers used in this experiment.

Figs 10a,b, respectively, illustrate the results of inference time and communication overhead in the preprocessing stage and the online stage for the four groups of data described above, under both merging and nonmerging approaches. In the preprocessing

Table 7 Description of convolutional layers in the experiment

	Convolutional Layer Description				
	Input	Kernel	Strides	Filters	Padding
Group 1	$32*32*8$	$5*5$	(1,1)	64	Valid
Group 2	$78*78*16$	$3*3$	(2,2)	48	Same
Group 3	$128*128*8$	$5*5$	(2,2)	16	Valid
Group 4	$64*64*16$	$3*3$	(1,1)	24	Same
Group 5	$8*8*6$	$3*3$	(1,1)	4	Same
Group 6	$16*16*4$	$5*5$	(1,1)	2	Valid
Group 7	$24*24*3$	$9*9$	(2,2)	4	Valid
Group 8	$32*32*3$	$7*7$	(2,2)	2	Same



(a) Preprocessing Stage **(b)** Online Stage

Fig. 10 Comparison of efficiency between merging and nonmerging approaches.

stage, the merging approach exhibits slightly lower computational time and communication overhead compared to the nonmerging approach. In the online stage, the merging approach demonstrates a reduction of approximately 48% in computational time and 66% in communication overhead, compared to the nonmerging approach, significantly reducing the user's waiting time and communication costs.

To better evaluate the computational efficiency of the merging approach proposed in Section 3.3, we compared it with the method of combining network parameters of consecutive layers in a neural network into a matrix (Matrix Method). Our merging approach combines adjacent linear layers into a single computational layer, utilizing homomorphic SIMD and fast convolution algorithms to accelerate the computations. In contrast, the Matrix Method achieves combined computation through the formulation of matrix-vector multiplication. Groups 5–8 in Table 7 represent the convolutional layers used in this experiment.

Table 8 presents the computational times for Groups 5–8 under two merging approaches. Compared to the Matrix Method, our proposed approach demonstrates varying degrees of reduction in computational time across all cases.

Subsequently, we implemented the fast convolution algorithm proposed in Section 3.4, as well as the improved naive method and Pack SC algorithm, within the framework of the proposed scheme, and tested them under the following four convolutional layer configurations:

Convolution I: Padding: same; Strides: (2,2); Input: $228*228*3$; Kernel: $3*3*3$; Filters: 64.

Convolution II: Padding: same; Strides: (2,2); Input: $64*64*32$; Kernel: $3*3*32$; Filters: 128.

Convolution III: Padding: valid; Strides: (2,2); Input: $128*128*3$; Kernel: $3*3*3$; Filters: 64.

Convolution IV: Padding: valid; Strides: (2,2); Input: $32*32*16$; Kernel: $5*5*16$; Filters: 24.

Table 9 illustrates the comparison of computational times among different convolution algorithms under the convolutional layer configurations mentioned before. It can be observed that the proposed fast convolution algorithm significantly surpasses the improved naive method and slightly outperforms the Packed SC algorithm in terms of computational time.

Table 8 Computing time comparison of merging approaches

	Group 5	Group 6	Group 7	Group 8
Ours	651.2 ms	1184.5 ms	2945.5 ms	1762.6 ms
Matrix Method	3075.6 ms	3490.2 ms	3118.7 ms	6117.8 ms

Table 9 Computing time comparison of convolution algorithms

	Convolution I	Convolution II	Convolution III	Convolution IV
Ours	13.58 s	23.57 s	4.13 s	1.93 s
Improved naive	129.55 s	615.61 s	42.95 s	21.66 s
Packed SC (Juvekar et al., 2018)	13.93 s	24.84 s	4.14 s	1.94 s

5 Conclusions

This study presents an efficient privacy-preserving secure neural network inference scheme, in which we proposed an approach of merging network parameters for consecutive linear layers and a fast convolution algorithm to reduce the computational time of linear layers in the preprocessing stage. The proposed scheme attains accuracies of 99.24% and 90.26% on the MNIST and Fashion-MNIST datasets, with linear operation time during the online stage measured at 1.39 ms and 1.47 ms, respectively. This demonstrates an 11% improvement in computational efficiency compared to DELPHI. The network parameters merging approach we proposed, respectively, reduces the computational time and communication overhead by approximately 48% and 66%, compared to the nonmerging approach during the online stage. The fast convolution algorithm presented in this study outperforms the improved naive method and slightly surpasses the state-of-the-art Packed SC method. Thus, our experiments demonstrate that our scheme provides superior performance in secure neural network inference under the premise of protecting privacy for both parties.

Contributors

Zixuan YANG designed the research. Zixuan YANG and Peng ZHANG processed the data. Zixuan YANG drafted the manuscript. Liquan CHEN, Peng ZHANG and Yang MA helped organize the manuscript. Zixuan YANG and Peng ZHANG revised and finalized the paper.

Conflicts of interest

All the authors declare that they have no conflict of interest.

References

Chabanne H, De Wargny A, Milgram J, et al., 2017. Privacy-preserving classification on deep neural network. *IACR Cryptol. ePrint Arch.*, 2017:35. https://eprint.iacr.org/2017/035

Chai XL, Wang YJ, Gan ZH, et al., 2022. Preserving privacy while revealing thumbnail for content-based encrypted image retrieval in the cloud. *Inf Sci*, 604:115-141. https://doi.org/10.1016/j.ins.2022.05.008

Chaudhari H, Rachuri R, Suresh A, 2020. Trident: Efficient 4PC framework for privacy preserving machine learning. Proc 27^{th} Annual Network and Distributed Systems Security, p.1-18. https://doi.org/10.14722/ndss.2020.23005

Cheon JH, Kim A, Kim M, et al., 2017. Homomorphic encryption for arithmetic of approximate numbers. Proc 23^{rd} Int Conf Theory and Applications of Cryptology and Information Security, p.409-437. https://doi.org/10.1007/978-3-319-70694-8_15.

Cheon JH, Han K, Kim A, et al., 2019. A full RNS variant of approximate homomorphic encryption. Proc 25^{th} Int Conf Selected Areas in Cryptography, p.347-368. https://doi.org/10.1007/978-3-030-10970-7_16

Chou E, Beal J, Levy D, et al., 2018. Faster CryptoNets: leveraging sparsity for real-world encrypted inference. https://arxiv.org/abs/1811.09953

Dowlin N, Gilad-Bachrach R, Laine K, et al., 2016. CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. Proc 33^{rd} Int conf Machine Learning, p.201-210.

Halevi S, Shoup V, 2014. Algorithms in HElib. Proc 34^{th} Annual Cryptology Conf, p.554-571. https://doi.org/10.1007/978-3-662-44371-2_31

Hesamifard E, Takabi H, Ghasemi M, 2017. CryptoDL: deep neural networks over encrypted data. https://arxiv.org/abs/1711.05189

Ioffe S, Szegedy C, 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. Proc 32^{nd} Int conf Machine Learning, p.448-456. https://proceedings.mlr.press/v37/ioffe15.html

Ishiyama T, Suzuki T, Yamana H, 2020. Highly accurate CNN inference using approximate activation functions over homomorphic encryption. Proc IEEE Int Conf Big

Data, p.3989-3995.
https://doi.org/10.1109/BigData50022.2020.9378372

Iha NK, Ghodsi Z, Garg S, et al., 2021. DeepReDuce: ReLu reduction for fast private inference. Proc 38^{th} Int Conf Machine Learning, p.4839-4849.

Juvekar C, Vaikuntanathan V, Chandrakasan AP, 2018. GAZELLE: a low latency framework for secure neural network inference. Proc 27^{th} USENIX Security Symp, p.1651-1669.

Kim D, Guyot C, 2023. Optimized privacy-preserving CNN inference with fully homomorphic encryption. *IEEE Trans Inf Forensics Secur*, 18:2175-2187.
https://doi.org/10.1109/TIFS.2023.3263631

Koti N, Pancholi M, Patra A, et al., 2021. SWIFT: Super-fast and robust privacy-preserving machine learning. Proc 30^{th} USENIX Security Symp, p.2651-2668.

Lai ZZ, Zhou YF, Zheng PJ, et al.,2024. Efficient privacy-preserving KAN inference using homomorphic encryption. https://arxiv.org/abs/2409.07751

Li JS, Liu IH, Tsai CJ, et al., 2020a. Secure content-based image retrieval in the cloud with key confidentiality. *IEEE Access*, 8:114940-114952.
https://doi.org/10.1109/ACCESS.2020.3003928

Li QF, Huang ZC, Lu Wj, et al., 2020b. HomoPAI: a secure collaborative machine learning platform based on homomorphic encryption. Proc IEEE 36^{th} Int Conf Data Engineering, p.1713-1717.
https://doi.org/10.1109/ICDE48307.2020.00152

Li Y, Yan HY, Huang T, et al., 2024. Model architecture level privacy leakage in neural networks. *Sci China Inf Sci*, 67(3):132101.
https://doi.org/10.1007/s11432-022-3507-7

Liu J, Juuti M, Lu Y, et al., 2017. Oblivious neural network predictions via MiniONN transformations. Pro ACM SIGSAC Conf Computer and Communications Security, p.619-631. https://doi.org/10.1145/3133956.3134056

Lou Q, Shen YL, Jin HX, et al., 2021. SAFENet: a secure, accurate and fast neural network inference. Proc 9^{th} Int Conf Learning Representations, p.1-13.

Ma JPK, Tai RKH, Zhao YJ, et al., 2021. Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. Proc 28^{th} Annual Network and Distributed System Security Symp, p.1-18.
https://doi.org/10.14722/ndss.2021.23166

Mishra P, Lehmkuhl R, Srinivasan A, et al., 2020. Delphi: a cryptographic inference system for neural networks. Pro Workshop on Privacy-Preserving Machine Learning in Practice, p.27-30.
https://doi.org/10.1145/3411501.3419418

Ng LKL, Chow SSM, 2021. GForce:GPU-friendly oblivious and rapid neural network inference. Proc 30^{th} USENIX Security Symp, p.2147-2164.

Riazi MS, Weinert C, Tkachenko O, et al., 2018. Chameleon: a hybrid secure computation framework for machine learning applications. Pro Asia Conf Computer and Communications Security, p.707-721.
https://doi.org/10.1145/3196494.3196522

Schroff F, Kalenichenko D, Philbin J, 2015. FaceNet: a unified embedding for face recognition and clustering. Pro IEEE Conf Computer Vision and Pattern Recognition, p.815-823.
https://doi.org/10.1109/CVPR.2015.7298682

Shen M, Cheng GH, Zhu LH, et al., 2020. Content-based multi-source encrypted image retrieval in clouds with privacy preservation. *Fut Gener Comput Syst*, 109:621-632. https://doi.org/10.1016/j.future.2018.04.089

Wang J, He DB, Castiglione A, et al., 2023. $\mathrm{PCNN_{cec}}$: efficient and privacy-preserving convolutional neural network inference based on cloud-edge-client collaboration. *IEEE Trans Netw Sci Eng*, 10(5):2906-2923.
https://doi.org/10.1109/TNSE.2022.3177755

Wang Y, Chen LQ, Wu G, et al., 2023. Efficient and secure content-based image retrieval with deep neural networks in the mobile cloud computing. *Comput Secur*, 128:103163.
https://doi.org/10.1016/j.cose.2023.103163

Xie TY, Yamana H, Mori T, 2022. CHE: channel-wise homomorphic encryption for ciphertext inference in convolutional neural network. *IEEE Access*, 10:107446-107458.
https://doi.org/10.1109/ACCESS.2022.3210134

Yagyu K, Takeuchi R, Nishigaki M, et al., 2023. Improving Classification Accuracy by Optimizing Activation Function for Convolutional Neural Network on Homomorphic Encryption. Proc 17^{th} Int Conf Broad-Band Wireless Computing, Communication and Applications, P.102–113.
https://doi.org/10.1007/978-3-031-20029-8_10