Efficient FHE-based Privacy-Enhanced Neural Network for AI-as-a-Service

Kwok-Yan Lam
Nanyang Technological University
Singapore
kwokyan.lam@ntu.edu.sg

Xianhui Lu
Chinese Academy of Sciences
China
luxianhui@iie.ac.cn

Linru Zhang
Nanyang Technological University
Singapore
linru.zhang@ntu.edu.sg

Xiangning Wang
Nanyang Technological University
Singapore
xiangning.wang@ntu.edu.sg

Huaxiong Wang
Nanyang Technological University
Singapore
hxwang@ntu.edu.sg

Si Qi Goh
Nanyang Technological University
Singapore
siqi005@e.ntu.edu.sg

ABSTRACT

AI-as-a-Service has emerged as an important trend for supporting the growth of the digital economy. Digital service providers make use of their vast amount of user data to train AI models (such as image recognitions, financial modelling and pandemic modelling etc) and offer them as a service on the cloud. While there are convincing advantages for using such third-party models, the fact that users need to upload their data to the cloud is bound to raise serious privacy concerns, especially in the face of increasingly stringent privacy regulations and legislations.

To promote the adoption of AI-as-a-Service while addressing the privacy issues, we propose a practical approach for constructing privacy-enhanced neural networks by designing an efficient implementation of fully homomorphic encryption. With this approach, an existing neural network can be converted to process FHE-encrypted data and produce encrypted output which are only accessible by the model users, and more importantly, within an operationally acceptable time (e.g. within 1 second for facial recognition in typical border control systems). Experimental results show that in many practical tasks such as facial recognition, text classification and so on, we obtained the state-of-the-art inference accuracy in less than one second on a 16 cores CPU.

KEYWORDS

Fully homomorphic encryption, Privacy-enhanced neural networks, Look-up table algorithm, Deep neural network, Digital trust

1 INTRODUCTION

AI-as-a-Service (AIaaS) has been experiencing rapid development because of strong demand in sharing of sophisticated and powerful AI models, which require not only technological advancement but also availability of vast amount of data resources. For digital service providers, it allows them to put their trained AI models on the cloud and offer the inference as a service, instead of disclosing their models to users. For users, with such cloud service, they just upload their inputs and get the inference results, instead of training the computationally expensive AI models in-house.

However, during AIaaS, cloud servers can access users' raw data, which may introduce privacy risks. Either users' input or the prediction result may contain privacy information, such as healthcare records and financial data. A cloud service that offers privacy-enhanced inference of AI models is needed, so that users can enjoy the service without disclosing their data to the cloud.

AI models, such as deep neural networks (DNNs), apply a sequence of evaluations on the input data and model parameters to obtain a prediction output. Fully Homomorphic encryption (FHE) is an useful tool in privacy-enhanced neural networks (PE-NN). HE provides a way to encrypt data while supporting computations through the encryption envelope[3]. Recent works[2, 3, 6, 9, 16, 21, 27] implement privacy-enhanced neural network inferences over encrypted data by applying FHE. The flow is shown in Figure 1.

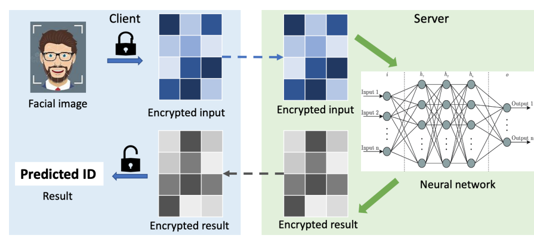

Figure 1: basic FHE-based privacy-enhanced neural network model (basic FHE-PE-NN)

In FHE-based PE-NN inferences, the user encrypts its sensitive data before sending it to the server. The server homomorphically evaluates the neural network over encrypted data and produces an encrypted inference output, then it returns the ciphertext to the user. The user who has the private key can decrypt the encrypted result. The server does not have the private key so it cannot decrypt neither the input nor the output. In this model, users can enjoy AI services on the cloud without disclosing their data, and AI service providers also do not disclose the trained AI models to users. However, high inference latency and low accuracy restrict existing works to be applied in real world AI services.

Inference latency. High inference latency is an obstacle to apply FHE-based PE-NN in DNNs. Existing works usually take quite long time per inference in simple and shallow networks. CryptoNets[16] takes over 200 seconds per inference on encrypted image from MNIST[20]. Faster CryptoNets improved the result. It takes around 40 seconds and 20000 seconds per inference on encrypted image from dataset MNIST and encrypted image from dataset CIFAR-10[19] respectively. Recently, SHE[21] further improved the latency and achieves around 10 seconds and 2000 seconds per inference on MNIST and CIFAR-10 respectively.

MNIST and CIFAR-10 are most commonly used datasets in existing works for benchmarks, which are simple and well studied. But in real-world applications, we usually anticipate neural network can solve more complex problems such as facial recognition and text classification. For such tasks, only SHE[21] mentioned that it use architectures of AlexNet, ResNet-18 and Shuffle Net for inferences on ImageNet and the result is 5 hours per inference with an accuracy of 69.4%.

To solve this problem, we analyze the architectures of various FHE schemes and find that there is not a 'perfect' FHE scheme which performs good in all homomorphic evaluations in neural networks. Evaluation of neural networks on encrypted data involves both arithmetic operations such as weighted sum and convolutions, evaluation of non-linear activations and bootstrapping after each neuron to enable further computations.

In practice, there are two main kinds of FHE schemes which are applied in FHE-based PE-NN. The *word-wise CKKS/BGV schemes*[4, 7] (adopted in many works such as CryptoNets[16]) are good at arithmetic operations but the bootstrapping and non-linear activation evaluations are significantly slow. The *bit-wise FHEW/TFHE schemes*[8, 13] provides a very efficient bootstrapping operation, but the arithmetic operations have to be computed gate by gate. Neither word-wise schemes nor bit-wise schemes is good at all evaluations of neural networks.

An optimized FHE scheme which combines the advantages of CKKS and TFHE schemes will help to reduce the inference latency.

Accuracy. The poor performance of homomorphic evaluations of activations causes the low accuracy. DiNN[3] and CryptoNets[16] only support sign function and square function as the activation function separately, where both of them are not commonly used in machine learning area. n-GraphHE[2], Lola[6] and Faster CryptoNets[9] have to use the low-degree polynomial approximation activations and thus fail to obtain the state-of-the-art inference accuracy. For example, Faster CryptoNet achieves 76.72% inference accuracy on CIFAR-10 dataset, while in an unencrypted network with ReLU activations, it is 93.72%.

The bottleneck to improve the accuracy is homomorphical evaluation of non-linear activations. ReLU and Sigmoid are widely used in modern neural networks. However, existing works in FHE-based PE-NN usually use sign function, square function, or low degree polynomials to approximate activations. Although it is possible to improve the inference accuracy of FHE-based PE-NN by enlarging the degree of polynomial approximation activations, the computing overhead increases exponentially with the degree and thus the time taken becomes unacceptable.

Efficient algorithms which can evaluate the activations in modern networks accurately over encrypted data are urgently needed.

In addition to the optimizations in FHE schemes, we review the structure of DNNs. DNNs, which may consists of hundreds of layers, are considered to be computationally expensive. To provide a DNN service, AI model owners usually start with pre-trained feature extractors and fine-tune them to solve users' task. Model owners are usually not willing to share the parameters they trained with users. However, if the whole DNN is evaluated on encrypted data in the AI model owner's server, then it is obvious that it will be too slow to be acceptable. The main reason is that homomorphically evaluation on encrypted data is much slower than the evaluation in plaintext, i.e., the more homomorphic evaluations on encrypted data, the slower it is. For FHE-based DNN evaluations, new structure and models, which can reduce the number of homomorphic computations and meanwhile protect the privacy of both users and AI model owners, are needed to improve the performance.

1.1 Our Results

Towards practical FHE-based PE-NN constructions, we propose a practical approach for constructing privacy-enhanced neural networks by designing an efficient implementation of fully homomorphic encryption.

We first propose a optimized fully homomorphic encryption scheme together with an efficient design for non-linear activation evaluation. We show that our FHE scheme achieves better results in both inference accuracy and time in the benchmark MNIST dataset.

Then we propose a new model which splits a deep neural network into a plaintext evaluation part and a ciphertext evaluation part. We deploy some pre-trained feature extraction layers to user's side and evaluate it in clear. The fine-tuned layers are retained at model owner's server and the input to the server is encrypted. In this way, we reduce the number of expensive homomorphic evaluations without disclosing either users' data and model providers' trained parameters. We call this model *Hybrid FHE-based privacy-enhanced neural network* (Hybrid FHE-based PE-NN).

As a result, we can perform facial recognition under one second, where it requires more than a day for basic FHE-based PE-NN model. We also show that our hybrid model can be used to solve practical tasks such as voice recognition, text classification and object classification. To the best of our knowledge, this is the first work to solve real-world problems by applying FHE-based PE-NN.

Optimized FHE scheme. Our optimized FHE scheme enables us the ability to perform weighted sums and convolutions on the approximate LWE-based additive homomorphic encryption schemes, and to evaluate non-linear activations on FHEW ciphertexts. Therefore, the evaluations of both linear and non-linear functions are very fast. Then the noise will be reduced during the evaluation of non-linear activations by applying homomorphic look-up table algorithm (LUT)[26].

We also observed the fact that the neural network is good at noise tolerating, i.e., the neural networks are usually not sensitive to the noise in less significant bits of input. This observation help us to reduce the size of encryption parameters and thus improves the efficiency while keeping high inference accuracy. It is achieved by: 1) On receiving of an input, we discard the less significant bits before encrypting; 2) In the beginning of LUT algorithm, we also discard the less significant bits of the LWE ciphertext.

In summary, our optimized FHE scheme has the following properties: 1) It can be applied to neural networks of arbitrary depth. 2) It supports many kinds of widely used activations, such as ReLU and Sigmoid. 3) When applied to inference of privacy-enhanced neural networks, it is fast and accurate.

Efficient design for non-linear activation evaluation. We further improve the evaluations of non-linear activations to speed up our system. Typically, a general method used to evaluate non-linear activations is homomorphic look-up-table algorithm (LUT)[3, 22, 26]. The core idea of LUT is to encode the "Table" containing the value of the non-linear activation into a polynomial, so that we can use ciphertext to locate the position of the desired output. In this work, we focus on improving the LUT algorithm. As a result, our improved system only takes 0.14s per inference on MNIST dataset, while 0.42s is needed without improvements.

We summarize our improvements as follows:

1) *RNS polynomial multiplication.* We use number theoretic transform (NTT) multiplication to speed up the multiplication. As a brief introduction, one NTT multiplication includes: 1) (NTT) turn two polynomials into two vectors (We call it residue numeral system (RNS) form); 2) (Position-wise multiplication) Perform position-wise multiplication between two vectors; 3) (Inverse-NTT (INTT)) turn the resulted vector into a polynomial. Lastly, we reduce the number of NTT and INTT to a third of before.

2) *Modulo function.* We design different modulo functions for different operations. We also reduce the number of modulo function by half compared to the previous method.

Hybrid FHE-based PE-NN model. The applications of basic FHE-based PE-NN is restricted by the fact that evaluation an entire DNN on encrypted data is too inefficient. In order to reduce the inference latency, we develop the idea that divide the network into two parts: an open network and a private network. We call it Hybrid FHE-based privacy-enhanced neural network model.

In this model, we make use of edge computing in a different purpose. The open network is distributed in the users' side and the private network is remained in the cloud server. The user first runs the open network in plaintext locally, then the results are encrypted and being sent to the server. The server evaluates the private network on encrypted input and returns a prediction output which is also encrypted. Only the user who has the secret key can decrypt the output and see the result. By replacing some computationally expensive ciphertext evaluations in the server side with low-cost plaintext computations in the user side, our model significantly reduce the inference latency.

The open network is made up of general feature extracting layers such as convolutional layers. Usually, it is chosen from well-known open-source networks such as FaceNet[31] (for facial recognition), TextCNN[17] (for text classification) and InceptionV3[33] (for object classification). The private model consists of some lower layers which is highly related to the dataset and the task. The private model is trained by the AI model owner. The training method and trained parameters are often considered as critical intellectual property by AI model owner, who are typically not willing to share them. Notice that when we set the whole network as private network, our model is exactly same as basic FHE-PE-NN model, so our model can be viewed as a generalization of basic FHE-PE-NN model.

As a future work, we will further improve our proposed techniques, and apply them in the area of privacy-enhancing machine learning to train AI models using encrypted data efficiently.

1.2 Overview of the paper

In section 2, we introduce backgrounds of our work. In section 3, we propose our optimized FHE scheme and propose a protocol for its application on PE-NN. Followed by section 4, where we further optimize our FHE scheme, and focus on improving the evaluation of non-linear activations. Next up in section 5, we propose a hybrid PE-NN model so that homomorphically evaluating deep complex neural networks becomes practical. Finally in section 6 we report the performance of our system and show our evaluation results of different PE-NN such as hand-writing digits recognition, facial recognition, voice recognition, text classification and object classification.

2 PRELIMINARIES

2.1 Notations

We denote all real numbers by $\mathbb{R}$. We denote all integers by $\mathbb{Z}$. For a positive integer q, we use $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$ to denote the multiplicative group of integers modulo q.

We use upper-case letter for matrix and use lower-case bold letters for vectors. Given a vector $\mathbf{x}$, we write $\boldsymbol{x}[i-1]$ for its i-th entry, i.e., $\boldsymbol{x}[0]$ is its first entry. We denote $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ the inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. We use lower-case letters for polynomials.

For a positive integer n, we write $[n]$ to denote the set $\{0, 1, 2, \ldots, n-1\}$. We write $a \xleftarrow{\$} S$ to denote that, a is sampled uniformly random from set S. Let $\mathcal{E}$ be a distribution, we use $e \xleftarrow{\$} \mathcal{E}$ to denote that e is randomly sampled according to $\mathcal{E}$.

Finally, we use $\mathbf{1}_{\mathcal{S}}$ to denote the indicator function:

$$\mathbf{1}_{\mathcal{S}} := \begin{cases} 1, & \mathcal{S} \text{ is TRUE} \\ 0, & \mathcal{S} \text{ is FALSE} \end{cases}.$$

2.2 (Ring) Learning with errors

The learning with errors (LWE) problem[30] was proposed as a generalization of learning parity with noise and it is widely used in construction of many cryptosystems.

For positive integers n and $q \geq 2$, a vector $\boldsymbol{s} \in \mathbb{Z}_q^n$, and a probability distribution $\mathcal{E} = \mathcal{E}(n)$ over $\mathbb{Z}_q$, let $\mathcal{A}_{\boldsymbol{s},\mathcal{E}}$ be the distribution obtained by choosing a vector $\boldsymbol{a} \xleftarrow{\$} \mathbb{Z}_q^n$ uniformly at random and a noise term $e \xleftarrow{\$} \mathcal{E}$, and outputting $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. The LWE problem is defined as follows.

Definition 2.1. (LWE) For an integer $q = q(n)$ and an error distribution $\mathcal{E} = \mathcal{E}(n)$ over $\mathbb{Z}_q$, the LWE problem $\mathrm{LWE}_{n,m,q,\mathcal{E}}$ is defined as: Given m independent samples from $\mathcal{A}_{\boldsymbol{s},\mathcal{E}}$, output $\boldsymbol{s}$ with non-negligible probability.

The decisional version is to distinguish between m samples chosen according to $\mathcal{A}_{\boldsymbol{s},\mathcal{E}}$ for some uniformly random $\boldsymbol{s}$ and m samples from the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

The Ring learning with errors (RLWE) problem[23] is a variant of LWE which is widely used to design homomorphic encryption schemes[4, 7, 11, 13]. The secret s is chosen from ring R. An RLWE sample $(a, as + e) \in R_q^2$ is generated by choosing $a \xleftarrow{\$} R_q$ uniformly at random and noise e from the error distribution $\mathcal{E}$ over R. Here $q \geq 2$ is an integer modulus. The decisional version is to distinguish between the RLWE sample for some secret s and a sample from uniform distribution over R_q^2.

2.3 Homomorphic encryption (HE)

A cryptosystem that supports computation on ciphertext without decryption is known as homomorphic encryption (HE)[8]. An encryption scheme consists of following PPT algorithms.

- Key generation. The algorithm takes security parameter λ as input and outputs a public key pk, and a secret key sk.
- Encryption. The algorithm takes a message m and the public key as input and outputs a ciphertext $\mathrm{ct} = \mathrm{Enc}(m)$.
- Decryption. The algorithm takes a ciphertext ct and the secret key as input and output a message $m' = \mathrm{Dec}(c)$.

If it is a secret key encryption scheme, then there is no public key and both encryption and decryption use the secret key.

Briefly speaking, a encryption scheme is **homomorphic** in an operation $\circ$, if there is another operation $\diamond$ such that $\mathrm{Enc}(m_1) \diamond \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 \circ m_2)$.

There are many sub-classes of homomorphic encryption scheme, depending on the types and number of operations it supports.

- *Partially homomorphic encryption.* It only supports the evaluation of circuits consisting of one type of gate. It is usually simple and fast. For example, *ElGamal cryptosystem*[14] can evaluate unbounded number of modular multiplications, and *Paillier cryptosystem*[28] can evaluate unbounded number of modular additions.
- *Somewhat homomorphic encryption.* It supports the evaluation of arbitrary circuits with multiple types of gates of pre-determined bounded depth. Typically, ciphertexts are "noisy", and the noise grows along with the increment of homomorphic computations, where ultimately the noise will make the resulting ciphertext indecipherable.
- *Fully homomorphic encryption (FHE).* It supports the evaluation of arbitrary circuits with multiple types of gates of unbounded depth. FHE solves the noise problem by using bootstrapping technique. In bootstrapping, the ciphertext is "refreshed" to reduce its noise level[15].

2.4 Number-theoretic transform (NTT)

In our encryption scheme, almost all computations involve high-degree polynomial multiplications. Naive polynomial multiplication costs $O(n^2)$ time, where n is the degree of polynomials. In field $\mathbb{R}$ or $\mathbb{C}$, using Faster Fourier Transforms (FFT)[10] for polynomial multiplication is a common technique. It changes the time cost from $O(n^2)$ to $O(n \log n)$, and this will have significant improvement when n is large, especially in the area of cryptography. Note that we only consider integers in cryptography, so a variant of FFT algorithm is widely used on finite field, which is called number-theoretic transform (NTT)[1].

The basic idea of NTT is that for some appropriately chosen prime q, $\mathbb{Z}_q[x]/(x^n+1)$ and $\mathbb{Z}_q^n$ are isomorphic. Therefore we can define a mapping $NTT(\cdot) : \mathbb{Z}_q[x]/(x^n+1) \to \mathbb{Z}_q^n$, which turns a polynomial into a integer vector. By definition of isomorphism, an inverse mapping is $INTT(\cdot) : \mathbb{Z}_q^n \to \mathbb{Z}_q[x]/(x^n+1)$, which turns a vector back to the polynomial. Based on the definition of isomorphism, we have:

- $NTT(a+b) = NTT(a) + NTT(b)$.
- $NTT(a \times b) = NTT(a) \times NTT(b)$.

Here the multiplication of vectors is position-wise multiplication. The vector $NTT(a)$ is called RNS form of polynomial a, and we use $\bar{a}$ to represent it in the following. Now we are ready to propose NTT multiplication. The time cost is $2O(n \log n) + O(n) + O(n \log n) = O(n \log n)$, which is better than the naive $O(n^2)$.

Algorithm 1: NTT multiplication

1: **Input:** Polynomials $a, b \in \mathbb{Z}_q[x]/(x^n+1)$.
2: Find $\bar{a} = NTT(a), \bar{b} = NTT(b)$. (One $NTT(\cdot)$ costs $O(n \log n)$.)
3: Calculate $\bar{c} = \bar{a} \times \bar{b}$. (Position-wise multiplication (PMUL), $O(n)$.)
4: Find $c = INTT(\bar{c})$. (One $INTT(\cdot)$ costs $O(n \log n)$.)
5: **Output:** Polynomial c.

3 OPTIMIZED FHE SCHEME

In FHE-based privacy-enhanced neural network inferences, the user encrypts the data before sends it to the server. The server homomorphically evaluates the neural network over encrypted data, and produces an encrypted inference output. It then returns the result's ciphertext to the user. Evaluation of neural networks on encrypted data involves both arithmetic operations such as weighted sum and convolutions, evaluation of non-linear activations and bootstrapping after each neuron to enable further computations. However, existing FHE schemes are not designed for evaluation neural networks on encrypted data. Most of them focus on high-precision evaluations and are very slow in inferencing of neural network. Some of them only support specific non-linear activations (e.g., [3],[16]). Our optimized FHE scheme which combines the advantages of CKKS and TFHE schemes will help to reduce the inference latency, and achieve the state-of-the-art inference accuracy.

3.1 Background: Neural network

Neural network is a artificial model inspired by biological brains. A neural network can be considered as population of artificial neurons arranged in layers. The raw data is encoded properly and fed into the input layer and the output layer will output the inference result. Each internal layer (i.e., hidden layer) receives the data generated by its previous layer and outputs the processed data for the next layer.

Neural networks are usually composed of layer of following types: Fully-connected layer (FC layer, every neuron of the layer takes all incoming data as inputs), convolutional layer (convolution evaluation is applied to its input), and so on.

At each neuron in layer l, the processing happens in two steps:

- Linear function: It is a weighted sum of inputs, which can be described as $\gamma_l = W_l \boldsymbol{x}_{l-1} + \boldsymbol{\beta}_l$. where W_l is the weight matrix of layer l and $\boldsymbol{\beta}_l$ is the bias of layer l.
- Activation function: The calculated weighted sum is passed to the activation function. An activation function is a mathematical function which adds non-linearity to the network. There are some commonly used activation functions: Sigmoid, ReLu and Softmax. It can be described as $\boldsymbol{x}_l = f_l(\gamma_l)$.

The neural network with one hidden layer can approximate any continuous function[12, 18]. Furthermore, a deep neural network with several layers of non-linearities has better ability in more complex tasks. In this work, our optimized FHE scheme is able to evaluate neural networks of arbitrary depth with possibly many hidden layers.

3.2 Overview

Our system is built on both a LWE-based secret key encryption scheme and a RLWE-based secret key encryption scheme. The entire part of our system run on integer, which provides us a possibility to apply faster algorithms in implementation (such as faster polynomial multiplication algorithm based on NTT).

The LWE-based secret key encryption scheme is used to encrypt the input test. Assume the length of input vector is l_{in}, we will generate l_{in} LWE ciphertexts. In each ciphertext, according to the good noise-tolerance of neural network, we can follow CKKS[7]'s method to add noise to the main message. As a result, we can choose relatively small ciphertext modulus and dimension.

At each neuron, our scheme performs:

Homomorphic evaluation for linear functions on LWE ciphertext. Since our LWE secret key encryption scheme is a type of additive homomorphic, we can simply evaluate the linear functions by homomorphic scale multiplication and addition.

Homomorphic evaluation for non-linear activations. We use the homomorphic look-up table algorithm (LUT) to evaluate non-linear activations and perform the bootstrapping at the same time. The core of LUT is to encode all possible output of the non-linear function $g(\cdot)$ into a polynomial $f(X)$ (denote by LUT function), so that we can "blindly" rotate the polynomial to locate the position of the desired output by homomorphically evaluating the decryption algorithm and re-encrypting the input ciphertext under the RLWE secret key. At the same time, the noise is reduced since we always "refresh" the ciphertext. Here, we use the RLWE-based secret key encryption scheme to encrypt the LWE secret key (denoted by evaluation key).

In order to achieve better efficiency, we hope the degree of polynomials in the LUT algorithm is small. So we make use of the noise tolerance of neural network again. Before entering LUT process, we squeeze the output range of the inner-product first. We only store a few number of most significant bits and discard some inaccurate least significant bits. Notice that the coefficients of the LUT function $f(X)$ store all possible value of the algorithm's output, so the number of coefficients (i.e., the degree of $f(X)$) is decided by the input range of the LUT algorithm (i.e., the output range of the inner-product). We will be able to achieve small degree of LUT function by squeezing the inner-product into a small range.

Key switching and homomorphic rounding algorithms. The LUT algorithm re-encrypts the input ciphertext, so that the underlying secret key and ciphertext modulus are changed. To ensure the network is extendable and can be applied to deep neural network, the output ciphertext should be in same size as the input ciphertext. Hence, we need key switching and homomorphic rounding algorithms to resize the output ciphertext.

In this way, the output of each neuron is in the same form as the input to the neuron, and the noise is reduced during bootstrapping. It ensures that the output of one neuron can be used for further computations in the next layer without an initially fixed limit on the number of layers one network has. Hence, our scheme can evaluate arbitrarily deep neural networks.

3.3 Encryption schemes

Additive homomorphic LWE encryption. We show our LWE-based secret key encryption scheme and related computations we used in our system. We write $ct \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m)$ to state that ct is a LWE ciphertext of m. Here q is the modulus, n is the dimension, and $\boldsymbol{s}$ is the secret key. And we use the same way for other encryption schemes. The detailed algorithms are Algorithm 2, 3, 4.

Algorithm 2: LWE Encryption Scheme

Input: plaintext m, modulus q, dimension n, secret key $\boldsymbol{s} \in \{-1, 0, 1\}^n$, error distribution $\mathcal{E}_{\mathrm{LWE}}$ on $\mathbb{Z}^q$.
Output: LWE ciphertext $(\boldsymbol{a}, b) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m)$.
Sample an error value $e \xleftarrow{\$} \mathcal{E}_{\mathrm{LWE}}$.
Sample vector $\boldsymbol{a} \xleftarrow{\$} \mathbb{Z}_n^q$.
Define $b := m + e - \langle \boldsymbol{a}, \boldsymbol{s} \rangle \mod q$.
Return $(\boldsymbol{a}, b)$.

Algorithm 3: Scale Multiplication

Input: LWE ciphertext $(\boldsymbol{a}, b) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m)$, plaintext $c \in \mathbb{Z}_n^q$.
Output: LWE ciphertext $(\boldsymbol{a}', b') \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(c \times m)$.
Compute $\boldsymbol{a}' := c \cdot \boldsymbol{a} \mod q$.
Compute $b' := c \cdot b \mod q$.
Return $(\boldsymbol{a}', b')$.

Algorithm 4: Addition

Input: LWE ciphertexts $(\boldsymbol{a}_1, b_1) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m_1)$ and $(\boldsymbol{a}_2, b_2) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m_2)$
Output: LWE ciphertext $(\boldsymbol{a}', b') \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m_1 + m_2)$.
Compute $\boldsymbol{a}' := \boldsymbol{a}_1 + \boldsymbol{a}_2 \mod q$.
Compute $b' := b_1 + b_2 \mod q$.
Return $(\boldsymbol{a}', b')$.

RLWE encryption and related operations. We show the RLWE-based secret key encryption scheme and related operations we used in our system. Our scheme is defined on ring $R_q := \mathbb{Z}_q[X]/(X^n + 1)$. The encryption scheme is Algorithm 5. And we can define scale multiplication and addition for RLWE ciphertext, in the same way as Algorithm 3, 4. Further, we include the definitions in [22] of the extended RLWE encryption in Algorithm 6, 7, 8 and 9.

3.4 Our framework

Our framework contains the following functions: Extraction, homomorphic rounding and key switching algorithms. We will briefly introduce them in this section, and the detailed algorithms are in

Algorithm 5: RLWE Encryption Scheme

Input: plaintext m, modulus q, dimension n, secret key s whose coefficients are in $\{-1, 0, 1\}$, error distribution $\mathcal{E}_{\text{RLWE}}$.
Output: RLWE ciphertext $(a, b) \in \text{RLWE}_s^{n,q}(m)$.
Sample error element $\boldsymbol{e} \xleftarrow{\$} \mathcal{E}_{\text{RLWE}}$.
Sample element $a \xleftarrow{\$} R_q$.
Define $b := m + e - a \times s \in R_q$.
Return (a, b).

Algorithm 6: Extended RLWE Encryption

Input: plaintext m, modulus q, dimension n, secret key $s \in \{-1, 0, 1\}^n$, error distribution $\mathcal{E}_{\text{RLWE}}$, decomposition base B.
Output: Extended RLWE ciphertext $\{(a_i, b_i)\}_{i \in [log(q)/log(B)]} \in \widetilde{\text{RLWE}}_s^{n,q}(m)$.
for $i = 0, 1, \ldots, log(q)/log(B) - 1$ **do**
  Find $(a_i, b_i) \in \text{RLWE}_s^{n,q}(m \times B^i)$.
end for
Return $\{(a_i, b_i)\}_{i \in [log(q)/log(B)]}$.

Algorithm 7: Extended RLWE Ciphertext Multiplication ($\diamond$)

Input: plaintext operand r, modulus q, dimension n, decomposition base B, extended RLWE ciphertext $\{(a_i, b_i)\}_{i \in [log(q)/log(B)]} \in \widetilde{\text{RLWE}}_s^{n,q}(m)$.
Output: RLWE ciphertext $(a, b) \in \text{RLWE}_s^{n,q}(r \times m)$.
Decompose r s.t. $r = \sum_{i=0}^{log(q)/log(B)-1} r_i \times B^i$.
Compute $(a, b) = \sum_{i=0}^{log(q)/log(B)-1} r_i \times (a_i, b_i) = \sum_{i=0}^{log(q)/log(B)-1} (r_i a_i, r_i b_i)$.
Ensure (a, b) in R_q^2.
Return (a, b).

Algorithm 8: RGSW Encryption based on Extended RLWE Encryption

Input: plaintext m, modulus q, dimension n, secret key $s \in \{-1, 0, 1\}^n$.
Output: RGSW ciphertext $(\beta, \alpha) \in \left(\widetilde{\text{RLWE}}_s^{n,q}(m), \widetilde{\text{RLWE}}_s^{n,q}(s \times m)\right)$.

Appendix A.1.
Extract0. Extract0 is to extract the constant term of polynomial m from its RLWE ciphertext $\text{RLWE}_s^{n,q}(m)$. It will output a LWE ciphertext which encrypts the 0-th coefficient of m. This extraction algorithm allows us to be able to convert a RLWE ciphertext to into LWE ciphertexts and will be frequently used in the following algorithms.
Rounding. The homomorphic rounding algorithm will change a LWE ciphertext with ciphertext modulus q' to a LWE ciphertext with ciphertext modulus q.

Algorithm 9: RLWE and RGSW Multiplication ($\odot$)

Input: $(a, b) \in \text{RLWE}_s^{n,q}(m_1)$, $(\beta, \alpha) \in \text{RGSW}_s^{n,q}(m_2)$.
Output: $(a', b') \in \text{RLWE}_s^{n,q}(m_1 \times m_2)$.
Return $(a', b') = a \diamond \alpha + b \diamond \beta$ and ensure $(a', b') \in R_q^2$.

Key switching. The key switching algorithm will change a LWE ciphertext with secret key $\boldsymbol{s'}$ and dimension n' to a LWE ciphertext with new secret key $\boldsymbol{s}$ and dimension n.

Homomorphic Look-up table algorithm. The look-up table algorithm takes a LWE ciphertext $ct \in \text{LWE}_s^{n,q}(m)$ and an evaluation function $F(\cdot)$ as input and it outputs a LWE ciphertext which encrypts $\Delta F(m)$, where Δ is a scale factor to limit noise.

Bit-by-bit Look-up table algorithm for general cases. We first show a bit-by-bit look-up table algorithm (Algorithm 10), which can be used in any cases, however not optimized.

2-bit Look-up table algorithm for single hidden layer. In the case where the neural network only has one hidden layer, we can skip the key switch and rounding operations after the look-up table evaluation, and proceed to the output layer directly. Applying this method, we are able to use the LWE secret key $\boldsymbol{s} \in \{0, 1\}^n$ instead of $\boldsymbol{s} \in \{-1, 0, 1\}^n$ during the encryption of the input test, while the security can still be proved[5, 25]. Then we can use the following Look-up table algorithm to reduce the number of external operations from n to $n/2$, compared with Algorithm 10. The details are in Algorithm 11.

2-bit Look-up table algorithm for multiple hidden layers. For multiple hidden layers, the key switching algorithm is required between each hidden layer, which consists of computations on RLWE ciphertexts. However, the binary secret for the Ring variant of LWE is still an open problem[8]. So, we cannot convert the LWE secret key from $\boldsymbol{s} \in \{-1, 0, 1\}^n$ to $\boldsymbol{s} \in \{0, 1\}^n$ here, but we can still construct a 2-bit look-up table algorithm. We will include the details in Algorithm 18 in Appendix A.2.

Now we are ready to show the full protocol for privacy-enhanced neural network using our optimized FHE scheme in Algorithm 12.

Theoretical analysis and experimental results of noise growing are shown in Section A.3.

4 EFFICIENT DESIGN FOR NON-LINEAR ACTIVATION EVALUATION

4.1 Overview

In this section, we further optimize our FHE scheme and focus on improving the evaluation of non-linear activations. Many previous work on FHE-based PE-NN use polynomial approximation activations to evaluate non-linear activations. However, it does not perform well. n-GraphHE[2], Lola[6] and Faster CryptoNet[9] have to use the low-degree polynomial approximation activations and thus fail to obtain the state-of-the-art inference accuracy. High-degree polynomial approximation activations can improve the accuracy, but the computing overhead increases exponentially with the degree, so the inference becomes inefficient.

In our scheme, we adopt homomorphic look-up-table (LUT) algorithm to evaluate non-linear activations. Although LUT algorithm

Algorithm 10: Bit-by-bit Look-up Table Evaluation.

Input: LWE ciphertext $(\boldsymbol{a}, b) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m)$ s.t. $|m| < q/4$, scale factor Δ, evaluation function $F(\cdot) : \mathbb{Z}_q \to \mathbb{Z}_q$, RLWE parameter set (n', q'), RLWE secret key s', a set of evaluation keys w.r.t. the LWE secret key $\boldsymbol{s} \in \{-1, 0, 1\}^n$:

$$\mathrm{EK}_{j,+} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[j]>0}), \mathrm{EK}_{j,-} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[j]<0}), j = 0, 1, \ldots, n-1 .$$

Output: LWE ciphertext $ct' \in \mathrm{LWE}_{\boldsymbol{s}'}^{n',q'}(\Delta F(m))$ where $\boldsymbol{s}'$ is the trivial vector form of polynomial s' (from high degree to low degree).

Let $\eta_k = kq/(2n')$ for $1 \le k \le n'/2$. Define a polynomial $f \in R_{n',q'}$ whose coefficients are:

$$f_j = \begin{cases} \lceil \Delta F(0) \rfloor & \text{if } j = 0 \\ \lceil \Delta F(-\eta_j) \rfloor & \text{if } 1 \le j \le n'/2 \\ \lceil -\Delta F(\eta_{n'-j}) \rfloor & \text{if } n'/2 < j < n' \end{cases} .$$

Let $b' = \lceil 2n'b/q \rfloor$, let $\boldsymbol{a}' = \lceil 2n'\boldsymbol{a}/q \rfloor$.

Initialize $\mathrm{AC} = (0, f \times X^{b'}) \in R_{n',q'}^2$.

for j=0,1,...,n-1 **do**

  $\mathrm{AC} \mathrel{+}= \left((X^{\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,+} + (X^{-\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,-} \right) \odot \mathrm{AC}$, note that all calculations are in $R_{n',q'}^2$.

end for

Return Extract0(AC).

Algorithm 11: 2-bit Look-up Table Evaluation for Single Hidden Layer Neural Network.

Input: LWE ciphertext $(\boldsymbol{a}, b) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m)$ s.t. $|m| < q/4$, scale factor Δ, evaluation function $F(\cdot) : \mathbb{Z}_q \to \mathbb{Z}_q$, RLWE parameter set (n', q'), RLWE secret key s', a set of evaluation keys w.r.t. the LWE secret key $\boldsymbol{s} \in \{0, 1\}^n$, and for $j = 0, 1, \ldots, n/2 - 1$:

$$\mathrm{EK}_{j,0} = \mathrm{RGSW}_{s'}^{n',q'}(\boldsymbol{s}[2j]\boldsymbol{s}[2j{+}1]), \mathrm{EK}_{j,1} = \mathrm{RGSW}_{s'}^{n',q'}(\boldsymbol{s}[2j](1{-}\boldsymbol{s}[2j{+}1])), \mathrm{EK}_{j,2} = \mathrm{RGSW}_{s'}^{n',q'}(\boldsymbol{s}[2j{+}1](1{-}\boldsymbol{s}[2j])) .$$

Output: LWE ciphertext $ct' \in \mathrm{LWE}_{\boldsymbol{s}'}^{n',q'}(\Delta F(m))$ where $\boldsymbol{s}'$ is the vector form of polynomial s' (from high degree coefficient to low degree coefficient).

Let $\eta_k = kq/(2n')$ for $1 \le k \le n'/2$. Define a polynomial $f \in R_{n',q'}$ whose coefficients are:

$$f_j = \begin{cases} \lceil \Delta F(0) \rfloor & \text{if } j = 0 \\ \lceil \Delta F(-\eta_j) \rfloor & \text{if } 1 \le j \le n'/2 \\ \lceil -\Delta F(\eta_{n'-j}) \rfloor & \text{if } n'/2 < j < n' \end{cases} .$$

Let $b' = \lceil 2n'b/q \rfloor$, let $\boldsymbol{a}' = \lceil 2n'\boldsymbol{a}/q \rfloor$.

Initialize $\mathrm{AC} = (0, f \times X^{b'}) \in R_{n',q'}^2$.

for j=0,1,...,n/2-1 **do**

  $\mathrm{AC} \mathrel{+}= \left((X^{\boldsymbol{a}'[2j]+\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,0} + (X^{\boldsymbol{a}'[2j]} - 1)\mathrm{EK}_{j,1} + (X^{\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,2} \right) \odot \mathrm{AC}$, all calculations are in $R_{n',q'}^2$.

end for

Return Extract0(AC).

performs better than polynomial approximation, it is the slowest part among all homomorphic evaluations in FHE-based PE-NN. To achieve better performance, we propose an efficient design for LUT-based non-linear activation evaluation.

We find that the NTT/INTT and the modulo calculations take around 70% time in one LUT. Therefore, we reduce the number of NTT and INTT in one LUT evaluation and optimize modulo calculations. As a result, our system becomes 3 times faster than before. Our improved system only takes 0.14s per input on MNIST dataset, while 0.42s is needed without improvements. We list our optimizations in below sections and an efficiency analysis to compare the difference in terms of the amount of time where we managed to bring down significantly.

4.2 Reduce the number of NTT and INTT

In this section, we show our optimizations on polynomial multiplications, which takes around 45% time in one LUT. We introduce 2 methods to reduce the number of NTT/INTTs in LUT algorithm. Section 4.2.1 is a general method and can be applied in any cases. Section 4.2.2 is designed for the neural network where very large modulus is necessary, such as hundreds of bits. In practical scenario, we should choose suitable method according to the concrete problems and neural networks.

Algorithm 12: Full Protocol for Privacy-enhanced Neural Network.

Input Layer parameters: LWE parameter set $(n, q, \boldsymbol{s})$.
Hidden Layer/Output Layer parameters: RLWE and RGSW related parameter set (n', q', s') where $n|n'$, scale factor Δ.
Public parameters: Decomposition base B, Key-Switching decomposition base B_{KS}.
Given input vector $\boldsymbol{x}$ of length N, define a set of N LWE ciphertexts $\{(\boldsymbol{a}_i, b_i)\}$ each of which is from $\mathrm{LWE}_{\boldsymbol{s}}^{n,q}(x[i])$.
Generate a set of evaluation keys w.r.t. the LWE secret key $\boldsymbol{s} \in \{-1, 0, 1\}^n$:
    $\mathrm{EK}_{j,+} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[j]>0}), \mathrm{EK}_{j,-} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[j]<0}), j = 0, 1, \ldots, n-1$.
Generate a set of LWE switching keys w.r.t. decomposition base B:
    $\mathrm{SK}_j \in \widetilde{\mathrm{RLWE}}_{\boldsymbol{s}}^{n,q}\left(\sum_{l=0}^{n-1} \boldsymbol{s}[jn+l]X^l\right), j = 0, 1, \ldots, n'/n - 1$.
Let input layer be Layer 0 and let output layer be Layer L.
for $l = 1, 2, \ldots, L$ **do**
    Let H_l be the number of neurons in layer l.
    Let $W_l, \boldsymbol{\beta}_l$ be the weight matrix and the bias vector from layer $l-1$ to layer l, respectively.
    for $h = 0, 1, 2, \ldots, H_l - 1$ **do**
        Let $\mathrm{LWE}_{l-1} := \{(\boldsymbol{a}_{l-1,i}, b_{l-1,i})\}_{i \in [H_{l-1}]}$, each of which is a LWE ciphertext $\mathrm{LWE}_{\boldsymbol{s}}^{n,q}(\cdot)$ from Layer $l-1$. For the sake of simplicity we omit the subscript of layer $l-1$ and write $(\boldsymbol{a}_i, b_i)$.
        Homomorphically evaluate the linear function in the h-th neuron in Layer l: $ip_h = \sum_{i \in [H_{l-1}]} W_l[h, i] \times (\boldsymbol{a}_i, b_i) + \boldsymbol{\beta}_l[h]$,i.e., $ip_h \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(\boldsymbol{\gamma}_l[h])$.
        Evaluate the look-up-table in the h-th neuron: $ct'_h = \mathrm{LUT}(ip_h) \in \mathrm{LWE}_{s'}^{n',q'}(\Delta \times \cdot)$.
        Switch from n' to smaller n: $\tilde{ct}_h = \mathrm{LWE\_}KS(ct'_h, B_{\mathrm{KS}}) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q'}(\Delta \times \cdot)$.
        Round from q' to smaller q and rescale: $ct_h = \mathrm{LWE\_}Rounding(\tilde{ct}_h) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(\cdot)$.
        Include ct_h in LWE_l.
    end for
end for
Return LWE_L to User. User can decrypt all ciphertexts in it with secret key $\boldsymbol{s}$, and then find the prediction result.

4.2.1 General method to reduce number of NTT/INTTs in LUT. . Taking our *2-bit look-up table evaluation for single hidden layer algorithm* (Algorithm 11) as an example, we will show how to reduce the number of times where NTT/INTTs are called.

First, we count the number of NTT/INTTs in one LUT. We use d to denote that in the beginning of each external product $\odot$, each polynomial in AC is decomposed to d polynomials. Regarding the computations of $(X^{\boldsymbol{a}'[2j]+\boldsymbol{a}'[2j+1]}-1)\mathrm{EK}_{j,0}$, $(X^{\boldsymbol{a}'[2j]}-1)\mathrm{EK}_{j,1}$ and $(X^{\boldsymbol{a}'[2j+1]}-1)\mathrm{EK}_{j,2}$, we discover that instead of calling NTT multiplication $3d$ times, we can simply implement it by rotating the polynomial in $\mathrm{EK}_{j,\cdot}$, which only involves polynomial addition and subtraction and is faster than NTT multiplication. We call it 'quick multiplication' and it was used in the implementation of the non-optimized LUT algorithm. Therefore, the NTT multiplications only appear in $\odot$. One $\odot$ includes $4d$ NTT polynomial multiplications. So one LUT calculation includes $4d \cdot (n/2) = 2dn$ NTT polynomial multiplications. Each NTT polynomial multiplication includes 3 NTT/INTT transformations, so one LUT has $6dn$ NTT/INTT transformations.

A general method is to store $\mathrm{EK}_{j,0}$, $\mathrm{EK}_{j,1}$ and $\mathrm{EK}_{j,2}$ in RNS form when generating them. Such evaluation keys are generated in the initialization phase and can be used repeatedly. When generating the evaluation keys, we can do NTT transformations after RGSW encryption to convert them into RNS form and store them.

In LUT calculation, when calculating $(X^{\boldsymbol{a}'[2j]+\boldsymbol{a}'[2j+1]}-1)\mathrm{EK}_{j,0}+(X^{\boldsymbol{a}'[2j]}-1)\mathrm{EK}_{j,1}+(X^{\boldsymbol{a}'[2j+1]}-1)\mathrm{EK}_{j,2}$, we first perform NTT transformation on $X^{\boldsymbol{a}'[2j]+\boldsymbol{a}'[2j+1]}-1$, $X^{\boldsymbol{a}'[2j]}-1$ and $X^{\boldsymbol{a}'[2j+1]}-1$. Then we proceed to compute position-wise multiplications and additions. The output of this part is $4d$ RNS form polynomials.

Next, we move to the calculation of $\odot$. The left side of $\odot$ is already in RNS form. So we first decompose the right side from 2 polynomials to $2d$ polynomials and do NTT transformations $2d$ times. The other calculations in $\odot$ can be done by position-wise multiplications and additions. Now the output of $\odot$ is 2 RNS form polynomials. We apply 2 INTT transformations on them and get 2 regular polynomials.

Therefore, in each loop, we only need $3+2d$ NTT transformations and 2 INTT transformations. So one improved LUT only has $(3+2d+2) \cdot (n/2) = dn + 2.5n$ NTT/INTT transformations.

For example, if we take $n = 512$ and $d = 2$ (same as our experiments), then the number of NTT/INTT transformations reduced from 6144 to 2304.

4.2.2 Further improvements on neural network with large modulus. When it is necessary to use hundreds bits modulus Q, a well known technique is to set Q as a product of L distinct and machine-word-sized primes: $Q = \Pi_{i=0}^{L-1} q_i$. Each q_i is also appropriate chosen so that NTT multiplication can be applied. The propose of this decomposition is, when modulus is very large, the multiplications and mod algorithm on computer/server become very slow. Using machine-word-sized is many times faster.

By Chinese remainder theorem (CRT), $R_{n,Q}$ and $\Pi_{i=0}^{L-1} R_{n,q_i}$ are isomorphic, which means that for each polynomial f in $R_{n,Q}$, it is equivalently L polynomials in $R_{n,q_i}, i = 0, \ldots, L-1$, so its RNS form $\bar{f}$ is L vectors in $\mathbb{Z}_{q_i}^n, i = 0, \ldots, L-1$.

For simplicity we write $EK_i := (X^{\boldsymbol{a}'[2i]+\boldsymbol{a}'[2i+1]} - 1) \cdot EK_{i,0} + (X^{\boldsymbol{a}'[2i]} - 1)EK_{i,1} + (X^{\boldsymbol{a}'[2i+1]} - 1)EK_{i,2}$.
Recall in Algorithm 11, in each loop we calculate $AC_{i+1} + = EK_i \odot AC_i$. We use overline to represent RNS form:

$$f(x) \in R_{n',Q} \to \overline{f} \in \Pi_{i=0}^{L-1} \mathbb{Z}_{q_i}^{n'},$$

which includes L vectors. The detailed algorithm is in Algorithm 13.

Algorithm 13: CRT based LUT for large modulus.

First turn AC_0, all RGSW ciphertexts into RNS form.
for $i = 0, 1, \ldots, n/2 - 1$ **do**
  Calculate $\overline{EK}_i$
  $\overline{AC}_{i+1} = \overline{EK}_i \odot \overline{AC}_i + \overline{AC}_i$
end for
Output: Extract0$(INTT(\overline{AC_{n/2}}))$, i.e., the LWE ciphertext of the constant term in the plaintext of $INTT(\overline{AC_{n/2}})$.

If we write $\overline{EK}_i = (\overline{\alpha}_i, \overline{\beta}_i)$, and $\overline{AC}_i = (\overline{a}_i, \overline{b}_i)$, then we have

$$\overline{AC}_{i+1} = \overline{EK}_i \odot \overline{AC}_i = \overline{\alpha}_i \diamond \overline{a}_i + \overline{\beta}_i \diamond \overline{b}_i .$$

Define $(\diamond)$ operator in RNS form Algorithm 14.

Algorithm 14: $(\diamond)$ operator in RNS form, $\bar{\alpha} \diamond \bar{f}$

Input: Extended RLWE ciphertext $\bar{\alpha}$ which includes L RLWE ciphertexts, i.e. $2L$ polynomials (in RNS form). According to CRT, find gadget vector $\vec{g} = (g_0, g_1, \ldots, g_{L-1}) \in \mathbb{Z}^L$ based on $\{q_i\}$. Let $\bar{\alpha} = \{(\bar{a}_i, \bar{b}_i)\}_{i=0}^{L-1}$, where each $(a_i, b_i) \in \mathrm{RLWE}_{s'}^{n',Q}(g_i \times \cdot)$.
A RNS form polynomial $\bar{f} \in \Pi_{i=0}^{L-1} \mathbb{Z}_{q_i}^{n'}$.
Decompose $\bar{f}$ into $\{\bar{f}_i = \bar{f} \mod q_i, i = 0, 1, \ldots, L-1\}$.
Calculate and output $\sum_{i=0}^{L-1} (\bar{f}_i \bar{a}_i, \bar{f}_i \bar{b}_i)$. (Position-wise multiplication)

4.3 Optimize modulo calculations

In this section, we present our optimizations on modulo calculations, which takes around 25% time in one LUT.

Different modulo calculations for different operations. Notice that the modulo calculation after addition is much easier than the modulo calculation after multiplication. The reason is that the size of coefficients grows slowly in addition. For example, we have two 59-bit coefficients c_1 and c_2, then $c_1 + c_2$ is at most 60-bit, while $c_1 \times c_2$ could be 118-bit. So we write two modulo calculation functions: one is for addition, the other is for multiplication.

Reduce the number of modulo calculation. We separate the modulo calculation from the polynomial addition. In most works, modulo calculation is followed by every polynomial addition, which means that at any time, when we do a polynomial addition, then we will do a modulo calculation. In fact, many modulus calculations are not necessary. For example, in our previous face recognition experiment, the modulus is a 59-bit prime. However in our implementation, we use 64-bit integer data type to store the coefficients.

From AC+=$\Big((X^{\boldsymbol{a}'[2j]+\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,0} + (X^{\boldsymbol{a}'[2j]} - 1)\mathrm{EK}_{j,1} + (X^{\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,2}\Big) \odot \mathrm{AC}$, we can see that we only need compute one modulo calculation after two additions in the left side of $\odot$. Similarly, by the definition of $\odot$ (Algorithm 9) and $\diamond$ (Algorithm 7), we only need one modulo calculation after 2-3 additions. As a result, the number of modulo calculations is half of before.

4.4 Efficiency analysis

In this section, we test and record the time taken of each part in one LUT, and to determine how much time our algorithm could save.

The parameters are same as those we used in our experiments. Length of LWE secret key is 512. Degree of LUT function/RLWE ciphertext is 2048. The modulus in our experiment is a 59-bit prime, which is not very large, so the method in Section 4.2.2 does not fit for our case. Therefore, we apply the general method (Section 4.2.1) to speed up our system.

We summarize our results in the Table 1. To observe the comparison clearly, we only ran it in single thread. From the table we can see that compared to our non-optimized LUT algorithm, our improved LUT saves 60% time.

5 HYBRID FHE-BASED PE-NN MODEL

5.1 Overview

The applications of basic FHE-based PE-NN is restricted by computationally expensive evaluations of DNNs on encrypted data. However, DNNs are widely used to solve many practical applications, such as facial recognition. Aiming for real-world scenarios, we develop Hybrid FHE-based PE-NN model, which could solve many practical problems within 1 second, such as facial recognition, text classification and object classification.

There are two privacy problems in this model: 1. For AI solution providers, private networks are often trained by private datasets, and owners do not want to share the parameters with others. 2. On users' side as well, the users of AI models are not willing to disclose both the input data and the prediction results to the server. To solve problem 1, model owners only publish open networks, and keep private networks in providers' cloud servers which allow users to make queries. To solve problem 2, users encrypt their data by homomorphic encryption, then the cloud servers evaluate the private networks on encrypted data and return encrypted prediction results.

In summary, we use edge computing to drive the model in two steps (see Figure 2): 1. User first runs the open network in plaintext locally; 2. The user encrypts the output from open network and sends the ciphertext to the server; 3. The server evaluates the private network on encrypted data and returns an encrypted prediction output. 4. Only the user who has the secret key can decrypt and see the result.

Operations	LUT in our previous work			Improved LUT			Time saved
	Number of calculation	Time per calculation	Time in total	Number of calculation	Time per calculation	Time in total	
NTT	4096	15.29us	62628us	1792	15.29us	27399 us	35229us
INTT	2048	16.05us	32870us	512	16.05us	8192us	24678us
PMUL	2048	2.84us	5816us	8193	2.84us	23268us	-17452us
Modulo Calculation (After additions)	10625	5.9us	62687us	4994	4.5us	22473us	40214us
Polynomial addition	8192	1.48us	12124us	6144	1.48us	9093us	3031us
Bit-decomposition	512	9.5us	4864us	512	9.5us	4864us	0
Quick multiplication	6145	8us	49160us	0	-	0	49160us
Total time	230149us=230ms			95289us=95ms			135ms (60%)

Table 1: Analysis on LUT algorithm

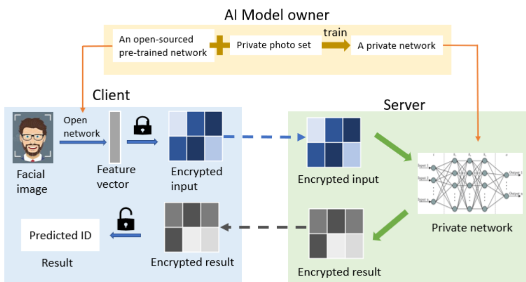

Figure 2: Hybrid FHE-based PE-NN model

In practical applications, such as facial recognition (FaceNet[31]) and object classification (InceptionV3[33]), there are many open-source pre-trained neural network projects. Since both the parameters and models of these projects are public, so we can set them as the open network (usually discard the fully connected layers). Meanwhile, the open project is usually not sufficient enough to solve their problems perfectly. There are two scenarios that might happen, either users' tasks differ too much from the open projects' tasks, or the open projects are not specific enough to solve the users' problem while running the inference on users' dataset. In the other words, the performance of open projects might tie to their own dataset. Inspired by transfer learning, AI solution provider can train a shallow network follows open network to solve user's task. Since the AI solution provider who trains the private network is usually not willing to share the parameters with users, so we set it as the private network and remain unseen in provider's cloud server. By transferring some computationally expensive ciphertext evaluations in the server side to lower-cost plaintext computations in the user side, our model significantly reduce the inference latency.

5.2 Training method and network structure

Transfer learning[29] is a good technique to train the network in our hybrid model. Transfer learning focuses on storing knowledge gained while solving problem and applying it to a different but related problem. From practical standpoints, transferring information from previously learned tasks for the learning of new tasks could significantly improve the sample efficiency.

We now talk about how to use transfer learning to train the network in our hybrid model. Assuming that we want to build a network to solve task B, and we have a dataset $\mathcal{D}_2$, we first search for a open-source pre-trained network on a related task A (trained by dataset $\mathcal{D}_1$). If we can find such a network, then we can apply transfer learning to obtain our network. There are two training methods to obtain the base of our network on task B from the pre-trained network on task A.

Freeze all parameters in pre-trained network. The idea of this method is that we use the pre-trained network as a feature extractor. Then we add a simple and shallow fully connected network (usually consists of 1 or 2 fully connected layer) and train the shallow network on these features. In this case, since the pre-trained network is open-source and we do not modify its parameters, so it can be set as the open network in our hybrid model. The shallow fully connected network is trained by ourselves and use our own dataset, so we can set it as the private network. The training method and network structure can be found in Figure 3.

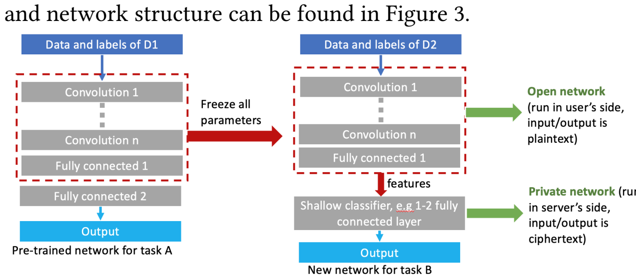

Figure 3: Freezing all parameters in pre-trained network

Fine-Tune some layers in pre-trained network. The method will be used when the difference gap of A and task B are huge, or our own dataset $\mathcal{D}_2$ is large, then we can consider this method. The idea of this method is that we only freeze the parameters of the top layers in the pre-trained network, and we add a simple and shallow fully connected network (usually consists of 1 or 2 fully connected layer). Then we train the lower layers in pre-trained network together with the shallow fully connected network on our own dataset. In this case, the frozen part of the open-source pre-trained network can be set as the open network in our hybrid model, since we do not modify it. But the parameters in the lower layers are re-trained by our own dataset, so both the lower layers and the shallow fully connected network should be set as the private network. The training method and network structure can be found in Figure 4.

Summary. When making a choice on training method, there are two criteria that we need to consider: (1) Task and data similarity; (2) Size of our dataset $\mathcal{D}_2$, and follow the instruction in Figure 5.

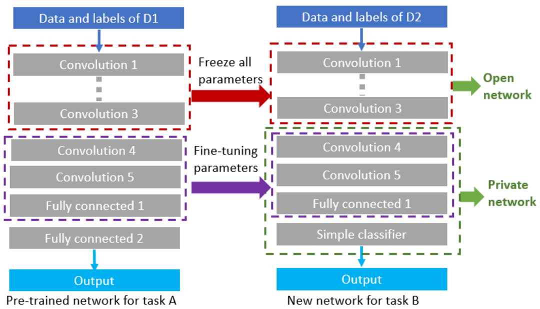

Figure 4: Fine-tuning some layers in pre-trained network

The training method depends on the problem and sometimes should be decided by experiments in plaintext.

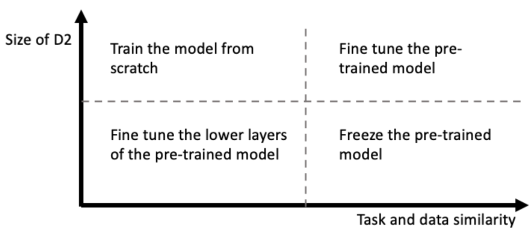

Figure 5: Training method instruction

The worst case is that we cannot find a suitable open-source pre-trained network. Then we need to train the whole network by ourselves and determine which part can be open to public.

6 EVALUATION RESULTS

In this section, we report the performance of our system. We evaluated all schemes on an AMD Milan 7313P 3.0GHz CPU which owns 16 cores. The security level is at least 80 bits. Our experiments include one of the most popular benchmark dataset, MNIST and practical applications such as facial recognition, voice recognition, text classification and object classification. For MNIST dataset, we train a BP network with one hidden layer and 30 hidden nodes, which is a commonly used network structure in the area of inference on encrypted data[3]. The average time for one inference is 0.14s and the accuracy loss compared to inference in plaintext is only less than 1%.

Beside that, we also show that our system can be used to solve practical problems, such as facial recognition and so on. Tasks like facial recognition are more difficult and requires a deep neural network, so we apply our hybrid FHE-based PE-NN for inference. The average time for one facial recognition is 0.18s, while basic FHE-based PE-NN model takes around 28 hours in the same server environment.

We also achieve excellent results in voice recognition, text classification and object classification, which show that our system can be used in real-world tasks. To the best of our knowledge, this is the first work to solve real-world problems by applying FHE-based privacy-enhanced neural networks.

6.1 Experiments on optimized FHE scheme

MNIST dataset. There are 60000 training samples and 10000 test samples in MNIST dataset for handwritten digit recognition. Each input is a 28×28 gray-level image, which is represented by a vector with length 784. For each point of the image, we set the value is 1 if the original value is > 0 and set it to be 0 otherwise.

We first train a BP network with one hidden layer and 30 hidden nodes, whose input is 784-dim vector and output has 10 classes. The training phase is in clear and the accuracy is 94.80%.

We present our results in table 2. The first line is the result that we evaluate our FHE scheme which equips with our LUT optimization. The second line is the result that we evaluate our FHE scheme without the LUT optimization. The results show that LUT optimization in Section 4 improves the efficiency for almost 3 times. Our inference accuracy on encrypted data is only 0.8% less than the inference accuracy on plain data. The third line shows the results in FHE-DiNN[3], which also evaluate their system in BP network with one hidden layer and 30 hidden nodes on MNIST dataset. Our system achieves better results in both inference accuracy and time on encrypted data.

FHE scheme	Activation function	Plaintext inference	Ciphertext inference	
		Accuracy	Accuracy	Time
Our Scheme with LUT optimization	ReLU (support others)	94.80%	94.04%	0.14s
Our Scheme without LUT optimization	ReLU (support others)	94.80%	94.04%	0.42s
FHE-DiNN[3]	Sign function	94.76%	93.71%	0.49s

Table 2: Inference on encrypted data in MNIST dataset

6.2 Experiments on hybrid PE-NN model

We also show that our hybrid FHE-based PE-NN system can be used to solve practical problems with help of transfer learning. In this section, we report performance of our system on facial recognition, voice recognition, text classification and object classification.

6.2.1 Facial recognition. In this experiment, we build a hybrid network to do facial recognition in a group of people. The network will identify the identity of the input photo. We first establish our own training and test dataset, which contains photos of 30 people. Note that our test dataset is different from the training dataset.

We use a pre-trained FaceNet (trained by VGGFace2) as the open network and train a private fully connected network on our own dataset by freezing all parameters in pre-trained network. Then we implement this system by our optimized FHE scheme and test it. The structure of the recognition network is shown in Figure 6.

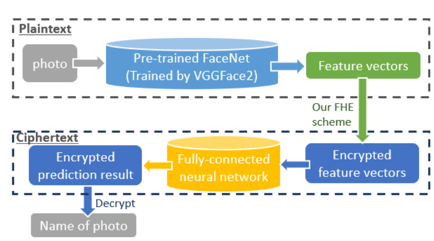

Figure 6: Structure of face recognition network

We present our evaluation results in Table 3. In this table, we compare 3 neural network models for face recognition: 1. Our hybrid neural network, where the open network runs in plaintext and the private network runs in ciphertext; 2. Traditional neural network, which is not privacy-enhanced and the whole network runs in plaintext; 3. Basic privacy-enhanced neural network, where the whole network is privacy-enhanced and runs in ciphertext.

We can observe that traditional neural network is very fast, but it does not consider the privacy problem. Basic privacy-enhanced neural network is does it well at privacy protection, but it is too slow to be applied to real-world. Our hybrid neural network only needs less than one second per recognition, while basic privacy-enhanced neural network needs 28 hours in the same server environment. Therefore, our hybrid neural network achieved good balance between privacy protection and efficiency, and can be used in real applications.

Network model	Private network structure	Number of activations in private network	Time	Accuracy
Hybrid PE-NN	2 FC layers	30	Open: 0.04s Private: 0.21s	100%
Traditinoal NN	none	0	0.04s	100%
Basic PE-NN	FaceNet+2 FC layers	$\approx 6M$	$\approx$ 28 hours (estimated)	-

Table 3: Inference on encrypted data in facial recognition

6.2.2 Voice recognition. Our scheme can be applied to process voice files as well. This can be applied when we do not want the neural network acquire the details of speech record while at the same time would require the running of neural network through the data to identify the identity of speaker. Our hybrid privacy-enhanced system can determine the owner of a speech record without knowing the speech content.

The dataset we used is *online VoxForge speech database*, whose test set consists of 5 speech utterances (5 seconds for each one) for each speaker and 34 speakers in total. We use Mel Frequency Cepstrum Coefficients (MFCC)[24] to do the feature extraction.

We present our results in Table 4. Even for a 20-30s speech utterances, we can identify the owner in few seconds with more than 90% accuracy. While in basic PE-NN, MFCC is too complicated to be homomorphically evaluated over encrypted data.

Network model	Private network structure	Number of activations in private network	Time	Accuracy
Hybrid PE-NN	2 FC layers	30	MFCC: 0.35s Private: 10.2s	91.2% (31/34)
Traditinoal NN	none	0	0.35s	97.0% (33/34)
Basic PE-NN	MFCC+2 FC layers	MFCC is too complicated to be implemented on encrypted data.		

Table 4: Inference on encrypted data in voice recognition

6.2.3 Text classification. Text files can also be processed by our hybrid PE-NN scheme. Nowadays, we receive lots of emails and SMS. Many of them are advertisement or spam mail. It takes us lots of time to check such messages everyday. Text classification is a process of providing labels to the set of texts or words and those labels will tell us about the sentiment of the set of words. We can build a hybrid privacy-enhanced text classification system, which can add labels to text files without knowing the plaintext.

For this application, we tested our scheme on *Movie Review dataset*, which consists of positive and negative sentences/snippets. We use TextCNN[17] as the open network.

We present our results in Table 5. Our scheme takes 0.016s per classification and does not lose the original accuracy compared to test in traditional neural network. Basic PE-NN takes around 6 minutes per inference, which is much slower. Notice that 84% accuracy is acceptable since the classifications are subjective and even human cannot achieve very high accuracy, e.g., [32].

Network model	Private network structure	Number of activations in private network	Time	Accuracy
Hybrid PE-NN	1 FC layer	0	Open: 0.013s Private: 0.003s	84%
Traditional NN	none	0	0.013s	84%
Basic PE-NN	TextCNN +1 FC layer	$\approx$ 20000	$\approx$ 6 minutes (estimated)	-

Table 5: Inference on encrypted data in text classification

6.2.4 Object classification. Finally we show our system works well in object classification. For this application, we tested our scheme on *Cat and Dog dataset*(A famous competition on Kaggle.com). We use InceptionV3[33] as the open network to extract the features.

We present our results in Table 6. Our scheme takes 0.019s per classification and achieved 99% accuracy. Training the 1-layer fully-connected network by transfer learning improve the inference accuracy significantly. When we use InceptionV3 directly, the inference accuracy is only 85%. After training by transfer learning, the inference accuracy becomes 99%.

Network model	Network structure	Number of activations in private network	Time	Accuracy
Hybrid PE-NN	Open: InceptionV3 Private: 1 FC layer	0	Open: 0.012s Private: 0.007s	99%
Traditional NN	Open: InceptionV3 +1 FC layer No private network	0	0.012s	99%
Traditional NN without TL	Open: InceptionV3 No private network	0	0.012s	85%
Basic PE-NN	No open network Private: InceptionV3 +1 FC layer	$\approx$ 5M	$\approx$ 1 day (estimated)	-

Table 6: Inference on encrypted data in object classification

7 CONCLUSION

This paper presented a practical approach for constructing privacy-enhanced neural networks by designing an efficient implementation of fully homomorphic encryption. As part of the efforts towards building a trusted digital economy, we aim to promote the adoption of AIaaS, which has emerged as an important trend for supporting the growth of the digital economy, by allowing AI models to process encrypted data.

In the global trend of digitalization, digital service providers make use of their vast amount of user data to train AI models (such as image recognitions, financial modelling and pandemic modelling etc) and offer them as a service on the cloud. While there are convincing advantages for using such third-party models, the fact that users need to upload their data to the cloud is bound to raise serious privacy concerns, especially in the face of increasingly stringent privacy regulations and legislations.

With this approach, an existing neural network can be converted to process FHE-encrypted data and produce encrypted output which are only accessible by the model users, and more importantly, within an operationally acceptable time (e.g. within 1 second for facial recognition in typical border control systems). Note that, in our work, we apply our FHE technique to existing proven neural networks instead of building proprietary neural networks. Hence, allowing privacy issues to be addressed separately from the accuracy of the AI models.

Experimental results show that in many practical tasks such as facial recognition, text classification and so on, we obtained the state-of-the-art inference accuracy in less than one second on a 16 cores CPU. In conclusion, our experiments show that in various practical tasks, our scheme achieved the state-of-the-art inference accuracy efficiently. It shows the feasibility of applying FHE-based PE-NN in real-world AI services to protect users' data.

Beside privacy-enhanced neural network inferences, the proposed technique can be used to achieve privacy-enhanced model training; though further performance improvement is needed, which is another technical challenge of our future work.

REFERENCES

[1] R. Agarwal and C. Burrus. 1974. Fast Convolution using fermat number transforms with applications to digital filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 22, 2 (1974), 87–97. https://doi.org/10.1109/TASSP.1974.1162555

[2] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 3–13.

[3] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*. Springer, 483–512.

[4] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* 6, 3 (2014), 1–36.

[5] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. 2013. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 575–584.

[6] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low latency privacy preserving inference. In *International Conference on Machine Learning*. PMLR, 812–821.

[7] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 409–437.

[8] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* 33, 1 (2020), 34–91.

[9] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. 2018. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953* (2018).

[10] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301.

[11] Ana Costache and Nigel P Smart. 2016. Which ring based somewhat homomorphic encryption scheme is best?. In *Cryptographers' Track at the RSA Conference*. Springer, 325–340.

[12] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[13] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 617–640.

[14] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.

[15] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.

[17] Bao Guo, Chunxia Zhang, Junmin Liu, and Xiaoyi Ma. 2019. Improving text classification with weighted word embeddings via a multi-channel TextCNN model. *Neurocomputing* 363 (2019), 366–374.

[18] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[20] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

[21] Qian Lou and Lei Jiang. 2019. She: A fast and accurate deep neural network for encrypted data. *Advances in Neural Information Processing Systems* 32 (2019).

[22] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. 2021. Pegasus: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1057–1073.

[23] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 1–23.

[24] Jorge Martinez, Hector Perez, Enrique Escamilla, and Masahisa Mabo Suzuki. 2012. Speaker recognition using Mel frequency Cepstral Coefficients (MFCC) and Vector quantization (VQ) techniques. In *CONIELECOMP 2012, 22nd International Conference on Electrical Communications and Computers*. IEEE, 248–251.

[25] Daniele Micciancio. 2018. On the hardness of learning with errors with binary secrets. *Theory of Computing* 14, 1 (2018), 1–17.

[26] Daniele Micciancio and Yuriy Polyakov. 2021. Bootstrapping in FHEW-like cryptosystems. In *Proceedings of the 9th on Workshop on Encrypted Computing &*

Applied Homomorphic Cryptography. 17–28.

[27] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*. 2505–2522.

[28] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*. Springer, 223–238.

[29] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[30] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 1–40.

[31] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[32] Amit Kumar Sharma, Sandeep Chaurasia, and Devesh Kumar Srivastava. 2020. Sentimental short sentences classification by using CNN deep learning model with fine tuned Word2Vec. *Procedia Computer Science* 167 (2020), 1139–1147.

[33] Xiaoling Xia, Cui Xu, and Bing Nan. 2017. Inception-v3 for flower classification. In *2017 2nd international conference on image, vision and computing (ICIVC)*. IEEE, 783–787.

A SUPPLEMENTARY MATERIALS FOR SECTION 3

A.1 Some building blocks of our framework

Our framework contains the following functions: Extraction, homomorphic rounding and key switching algorithms. We include the detailed algorithms are in Algorithm 15, Algorithm 16 and Algorithm 17.

Algorithm 15: Extract0, to Extract the Constant Term of an Encrypted Polynomial.

Input: $(a, b) \in \mathrm{RLWE}_s^{n,q}(m)$, where m is a polynomial.
Output: $(\boldsymbol{a}', b') \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m(0))$ where $\boldsymbol{s}$ is the vector form of polynomial s (from high degree coefficient to low degree coefficient). $m(0)$ is the constant term of m.
Let $s = s_{n-1}X^{n-1} + \ldots + s_1X + s_0$, $a = a_{n-1}X^{n-1} + \ldots + a_1X + a_0$ and $b = b_{n-1}X^{n-1} + \ldots + b_1X + b_0$.
Set vector $\boldsymbol{s} = (s_{n-1}, \ldots, s_1, s_0)$.
Set vector $\boldsymbol{a}' = (-a_1, \ldots, -a_{n-1}, a_0)$ and $b' = b_0$.
Return $(\boldsymbol{a}', b')$.

Algorithm 16: LWE Rounding and Rescale

Input: LWE ciphertext $(\boldsymbol{a}, b) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q'}(m)$, another modulus q.
Output: LWE ciphertext $ct \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(\lceil qm/q' \rfloor)$.
Return $(\lceil q\boldsymbol{a}/q' \rfloor, \lceil qb/q' \rfloor)$.

A.2 2-bit Look-up table algorithm for multiple hidden layers.

In this case, the key switching algorithm is required between each hidden layer, which consists of computations on RLWE ciphertexts. We elaborate the details in Algorithm 18.

A.3 Noise analysis of optimized FHE scheme

In this section, we analyze the growing of noise throughout our system.

Algorithm 17: LWE Key Switch

Input: LWE ciphertext $(\boldsymbol{a}, b) \in \mathrm{LWE}_{\boldsymbol{s}'}^{n',q}(m)$, another dimension n, decomposition base B, a set of LWE switching keys w.r.t. B and another LWE secret key $\boldsymbol{s}$:

$$\mathrm{SK}_j \in \widetilde{\mathrm{RLWE}}_{\boldsymbol{s}}^{n,q}\left(\sum_{l=0}^{n-1} \boldsymbol{s}[jn+l]X^l\right), j = 0, 1, \ldots, n'/n - 1 .$$

Output: LWE ciphertext $ct' \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m)$.
Define a set of polynomials $\{\tilde{a}_j\}$ where

$$\tilde{a}_j := \boldsymbol{a}[jn] - \sum_{l=1}^{n-1} \boldsymbol{a}[jn+l]X^{n-l}, j = 0, 1, \ldots, n'/n - 1 .$$

Compute a RLWE ciphertext $\tilde{ct} := \sum_{j=0}^{n'/n-1} \tilde{a}_j \diamond \mathrm{SK}_j$.
Return $(\boldsymbol{0}, b) + \mathrm{Extract0}(\tilde{ct})$.

Notations. Let σ^2_{LWE} be the variance of noise used in the LWE encryption. Define $\sigma^2_{\mathrm{RGSW}}, \sigma^2_{\mathrm{KS}}$ in the same way.

By the widely used assumptions, we assume that in each polynomial all the coefficients behave like independent zero-mean random variables of the same variance [11] (weaker than i.i.d.), and central limit heuristic [13]. The procedures are similar as in [22]. Further, note that it suffices to find the change of error variance within one neuron of each layer. Fix Layer l, assume that $\mathrm{LWE}_{l-1} := \{(\boldsymbol{a}_i, b_i)\}_{i\in[H_{l-1}]}$ has an error whose variance is σ^2 in each LWE ciphertext $(\boldsymbol{a}_i, b_i)$.

Linear function. $ip_h = \sum_{i\in[H_{l-1}]} W_l[h,i] \times (\boldsymbol{a}_i, b_i) + \boldsymbol{\beta}_l[h] \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(\cdot)$ and thus the noise becomes e_{ip} with variance $\sigma^2_{ip} := \sum_{i\in[H_{l-1}]} W_l^2[h,i]\sigma^2 \le ||W_l||^2\sigma^2$. Here we define

$$||W_l||^2 := \max_{h\in[H_l]} \left\{ \sum_{i\in[H_{l-1}]} W_l^2[h,i] \right\}$$

.

LUT. . As in [22], the LUT evaluations outputs a LWE ciphertext that decrypts to $\Delta F(m+e_1)+e_2$ for input m. And the LUT input is the above ciphertext ip_h of linear function. We first find e_1, which is the error of look-up index. Our analysis is based on the Algorithm 10, and for other proposed algorithms the analysis is almost identical.

- By our LUT algorithm (Algorithm 10), $(\boldsymbol{a}', b')$ has decryption result $b' + \langle \boldsymbol{a}', \boldsymbol{s} \rangle = \lceil 2n'm/q \rfloor + e$. By the central limit heuristic, e has variance $4n'^2\sigma^2_{ip}/q^2 + (||\boldsymbol{s}||_2^2 + 1)/12$, where the second term is from the rounding $\lceil \cdot \rfloor$. Note that ciphertext is generated from uniformly random distribution, thus the loss of $\lceil \cdot \rfloor$ is from $U[-1/2, 1/2]$, i.e., the uniform distribution on $[-1/2, 1/2]$.
- Next, note the definition of polynomial f's coefficients, the above error e is scaled up by a factor of $q/(2n')$, which results in an error with variance $\sigma^2_{ip}+q^2(||\boldsymbol{s}||_2^2+1)/(48n'^2)$. Also note that the error of $\lceil 2n'm/q \rfloor$ is scaled up by a factor of $q/(2n')$, this error has variance $(1/12) \times q^2/(4n'^2)$ if assuming the loss of $\lceil \cdot \rfloor$ is from $U[-1/2, 1/2]$. Summing up both parts we

Algorithm 18: 2-bit Look-up Table Evaluation for Multiple Hidden Layers Neural Network.

1: **Input:** LWE ciphertext $(\boldsymbol{a}, b) \in \mathrm{LWE}_{\boldsymbol{s}}^{n,q}(m)$ s.t. $|m| < q/4$, scale factor Δ, evaluation function $F(\cdot) : \mathbb{Z}_q \to \mathbb{Z}_q$, RLWE parameter set (n', q'), RLWE secret key s', a set of evaluation keys w.r.t. the LWE secret key $\boldsymbol{s} \in \{-1, 0, 1\}^n$:

$$\mathrm{EK}_{j,0} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=1,\boldsymbol{s}[2j+1]=0}), \mathrm{EK}_{j,1} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=1,\boldsymbol{s}[2j+1]=1}), \mathrm{EK}_{j,2} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=1,\boldsymbol{s}[2j+1]=-1}),$$
$$\mathrm{EK}_{j,3} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=0,\boldsymbol{s}[2j+1]=1}), \mathrm{EK}_{j,4} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=0,\boldsymbol{s}[2j+1]=-1}), \mathrm{EK}_{j,5} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=-1,\boldsymbol{s}[2j+1]=1}),$$
$$\mathrm{EK}_{j,6} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=-1,\boldsymbol{s}[2j+1]=0}), \mathrm{EK}_{j,7} = \mathrm{RGSW}_{s'}^{n',q'}(\mathbf{1}_{\boldsymbol{s}[2j]=-1,\boldsymbol{s}[2j+1]=-1}), j = 0, 1, \ldots, n/2-1 .$$

2: **Output:** LWE ciphertext $ct' \in \mathrm{LWE}_{\boldsymbol{s}'}^{n',q'}(\Delta F(m))$ where $\boldsymbol{s}'$ is the trivial vector form of polynomial s' (from high degree to low degree).

3: Let $\eta_k = kq/(2n')$ for $1 \le k \le n'/2$. Define a polynomial $f \in R_{n',q'}$ whose coefficients are:

$$f_j = \begin{cases} \lceil \Delta F(0) \rfloor & \text{if } j = 0 \\ \lceil \Delta F(-\eta_j) \rfloor & \text{if } 1 \le j \le n'/2 \\ \lceil -\Delta F(\eta_{n'-j}) \rfloor & \text{if } n'/2 < j < n' \end{cases} .$$

4: Let $b' = \lceil 2n'b/q \rfloor$, let $\boldsymbol{a}' = \lceil 2n'\boldsymbol{a}/q \rfloor$.

5: Initialize $\mathrm{AC} = (0, f \times X^{b'}) \in R_{n',q'}^2$.

6: **for** j=0,1,...,n/2-1 **do**

7:

$$\mathrm{AC}{+}{=}\Big((X^{\boldsymbol{a}'[2j]} - 1)\mathrm{EK}_{j,0} + (X^{\boldsymbol{a}'[2j]+\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,1} + (X^{\boldsymbol{a}'[2j]-\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,2} + (X^{\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,3}$$
$$+ (X^{-\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,4} + (X^{-\boldsymbol{a}'[2j]+\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,5} + (X^{-\boldsymbol{a}'[2j]} - 1)\mathrm{EK}_{j,6} + (X^{-\boldsymbol{a}'[2j]-\boldsymbol{a}'[2j+1]} - 1)\mathrm{EK}_{j,7} \Big) \odot \mathrm{AC} ,$$

all calculations are in $R_{n',q'}^2$.

8: **end for**

9: Return Extract0(AC).

have

$$var(e_1) = \sigma_{ip}^2 + q^2(||s||_2^2+2)/(48n'^2) = ||W_l||^2\sigma^2 + q^2(||s||_2^2+2)/(48n'^2) .$$

Next, we proceed to find e_2. In each step j we calculate the external product

$$\mathrm{AC}{+}{=}\Big((X^{\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,+} + (X^{-\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,-} \Big) \odot \mathrm{AC} .$$

As in [11] and [13], under their assumptions we have: Given polynomials a, b, whose variances of the coefficients are σ_a^2 and σ_b^2 respectively, then:

- the variance of coefficients of polynomial $a + b$ is $\sigma_a^2 + \sigma_b^2$;
- the variance of coefficients of polynomial ab is $n'\sigma_a^2\sigma_b^2$.

Based on the above we are able to find the variance of polynomial calculations. For the sake of simplicity, when we say the variance of a polynomial we mean the variance of the coefficients of this polynomial.

- Let $\mathrm{RGSW}_j := (X^{\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,+} + (X^{-\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,-}$. First we find the variances of the coefficients of the polynomial in each slot of RGSW_j. By our definition of algorithm, $\mathrm{EK}_{j,+}$ and $\mathrm{EK}_{j,-}$ have the same error variance: σ_{RGSW}^2 in each slot. Then note that $(X^{\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,+}$ is a sum of $X^{\boldsymbol{a}'[j]}\mathrm{EK}_{j,+}$ and $-\mathrm{EK}_{j,+}$. They both have variances σ_{RGSW}^2. The other part $(X^{-\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,-}$ is the same, so the variance in each slot of $(X^{\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,+} + (X^{-\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,-}$ is $4\sigma_{\mathrm{RGSW}}^2$.
- Suppose in step j, $\mathrm{AC} = (a_j, b_j)$ where both a_j and b_j are polynomials from the same ring. Given decomposition base B, they have decomposition

  $$a_j \to (\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{d-1}), b_j \to (\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_{d-1})$$

  where $a_j = \sum_{i=0}^{d-1} B^i \hat{a}_i$, $b_j = \sum_{i=0}^{d-1} B^i \hat{b}_i$. We also write

  $$\begin{aligned} \mathrm{RGSW}_j &= (\beta, \alpha) \\ &= \Big((\beta[0], \beta[1] \ldots, \beta[d-1]), (\alpha[0], \alpha[1] \ldots, \alpha[d-1]) \Big) . \end{aligned}$$

  By definition of our RGSW each $\beta[j]$ or $\alpha[j]$ is a pair of polynomials. Then

  $$\mathrm{RGSW}_j \odot \mathrm{AC} = a_j \diamond \alpha + b_j \diamond \beta = \sum_{i=0}^{d-1} \Big(\hat{a}_i\alpha[i] + \hat{b}_i\beta[i] \Big) .$$

  Since all $\hat{a}_j$ and $\hat{b}_j$ have B-bounded coefficients, the variance of each slot in RGSW $\odot$ AC is bounded by

  $$d \times 2 \times n'B^2 \times (4\sigma_{\mathrm{RGSW}}^2) = 8n'dB^2\sigma_{\mathrm{RGSW}}^2 .$$

- Finally, $\mathrm{RGSW}_j = (X^{\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,+} + (X^{-\boldsymbol{a}'[j]} - 1)\mathrm{EK}_{j,-}$ in each step j has the same distribution, and thus has the same error variance. The total variance in n steps is $var(e_2) = 8nn'dB^2\sigma_{\mathrm{RGSW}}^2$.

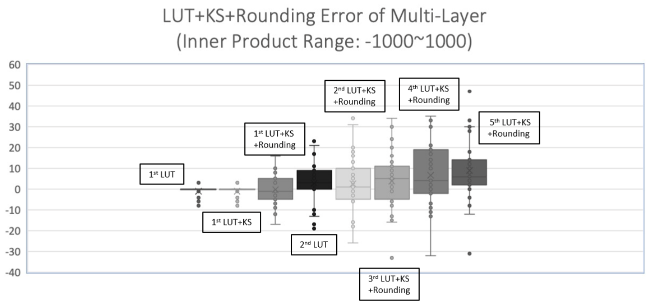

Figure 7: Test result on noise growing in LUT+KS+Rounding

Key Switch. By our definitions of Key-Switching key $\{\mathrm{SK}_j\}$ and $\widetilde{\mathrm{RLWE}}$, each SK_j has d RLWE ciphertexts, each of which has error of variance σ^2_{KS}. The error e_{KS} is from $\sum_{j=0}^{n'/n-1} \tilde{a}_j \diamond \mathrm{SK}_j$. Similarly as the $\diamond$ operation in the LUT part, note that $deg(\tilde{a}_j) = n$, the variance of e_{KS} is bounded by

$$n'/n \times nB^2_{\mathrm{KS}}\sigma^2_{\mathrm{KS}} = n'B^2_{\mathrm{KS}}\sigma^2_{\mathrm{KS}} .$$

Rounding. Suppose the rounding factor is z, simply round down the variance by a factor of z^2. To round the ciphertext to integer, $\lceil\cdot\rfloor$ results in an additional variance of $var(e_{\mathrm{RD}}) = (||s||^2_2 + 1)/12$.

Like in [22] and many other works, we assume $F(\cdot)$ is L-lipschitz and then we will have $|F(m + e_1) - F(m)| \leq L|e_1|$. Next, $|e_1|$, $|e_2|$ and $|e_{\mathrm{KS}}|$ can be bounded w.h.p. by $O(\sqrt{var(e_i)})$, $i = 1, 2, \mathrm{KS}$ under central limit heuristic[13]. To sum up, after scaling down Δ, the error between $F(m)$ and $F(m+e_1)+e_2/\Delta+e_{\mathrm{KS}}/\Delta+e_{\mathrm{RD}}$ is bounded by:

$$\begin{aligned} &O\left(L\sqrt{var(e_1)} + \frac{\sqrt{var(e_2)} + \sqrt{var(e_{\mathrm{KS}})}}{\Delta} + \sqrt{var(e_{\mathrm{RD}})}\right) \\ &= O\Bigg(L\sqrt{||W_l||^2\sigma^2 + q^2(||s||^2_2 + 2)/(48n'^2)} \\ &\quad + \frac{\sqrt{8nn'dB^2\sigma^2_{\mathrm{RGSW}}} + \sqrt{n'B^2_{\mathrm{KS}}\sigma^2_{\mathrm{KS}}}}{\Delta} + \sqrt{(||s||^2_2 + 1)/12}\Bigg) . \end{aligned}$$

Also, we give the error variance of the LWE ciphertext that output to the next layer:

$$\begin{aligned} \sigma^2_{l+1} &\leq \left(L^2 var(e_1) + \frac{var(e_2) + var(e_{\mathrm{KS}})}{\Delta^2} + var(e_{\mathrm{RD}})\right) \\ &\leq \Bigg(L^2\left(||W_l||^2\sigma^2_l + q^2(||s||^2_2 + 2)/(48n'^2)\right) \\ &\quad + \frac{8nn'dB^2\sigma^2_{\mathrm{RGSW}} + n'B^2_{\mathrm{KS}}\sigma^2_{\mathrm{KS}}}{\Delta^2} + (||s||^2_2 + 1)/12\Bigg) . \end{aligned}$$

A.3.1 Experiments on noise growing in multiple layers. After showing that our system performed well in single hidden layer neural network, we move our focus to multiple layers. The key to achieve good results in multiple layers is to ensure that the noise is always in a suitable range. In section A.3, we showed a theoretical analysis on the growing of noise. Now, we show some experimental results about it. The noise is growing in two steps. The first is in the inner-product computation. In this part, the growing of noise is linear and easy to control by choosing suitable parameters. So we focus on the second step: Look-up table (LUT), key switching (KS) and rounding. We ask the program to run one inner-product computation and then to perform LUT+KS+Rounding 5 times continuously and get the following Figure 7 (the evaluation function in LUT is ReLu). The picture shows the difference between the result of "i-th LUT+KS+Rounding" and the real value. 0 means that the result in "i-th LUT+KS+Rounding" is the same as the real value, which means no error exists. From the picture we can see that, most of points fall in $(-5, 20)$, which is relatively small compared to the inner-product range $(-1000, 1000)$.