Privacy-Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Network

Joon-Woo Lee[1,*], HyungChul Kang[2,*] , Yongwoo Lee[2], Woosuk Choi[2], Jieun Eom[2], Maxim Deryabin[2], Eunsang Lee[1], Junghyun Lee[1], Donghoon Yoo[2], Young-Sik Kim[3] and Jong-Seon No[1]

[1] Dept. of Electrical and Computer Eng., INMC, Seoul National University, Seoul, Republic of Korea,
joonwoo42@snu.ac.kr,{eslee3209,jhlee}@ccl.snu.ac.kr,jsno@snu.ac.kr

[2] Samsung Advanced Institute of Technology, Suwon, Republic of Korea,
{hc1803.hang,yw0803.lee,woosuk0.choi,jieun.eom,max.deriabin,say.yoo}@samsung.com

[3] Dept. of Information and Communication Eng., Chosun University, Republic of Korea,
iamyskim@chosun.ac.kr

Abstract. Fully homomorphic encryption (FHE) is one of the prospective tools for privacy-preserving machine learning (PPML), and several PPML models have been proposed based on various FHE schemes and approaches. Although the FHE schemes are known as suitable tools to implement PPML models, previous PPML models on FHE such as CryptoNet, SEALion, and CryptoDL are limited to only simple and non-standard types of machine learning models. These non-standard machine learning models are not proven efficient and accurate with more practical and advanced datasets. Previous PPML schemes replace non-arithmetic activation functions with simple arithmetic functions instead of adopting approximation methods and do not use bootstrapping, which enables continuous homomorphic evaluations. Thus, they could not use standard activation functions and could not employ a large number of layers. In this work, we firstly implement the standard ResNet-20 model with the RNS-CKKS FHE with bootstrapping and verify the implemented model with the CIFAR-10 dataset and the plaintext model parameters. Instead of replacing the non-arithmetic functions with the simple arithmetic function, we use state-of-the-art approximation methods to evaluate these non-arithmetic functions, such as the ReLU and softmax, with sufficient precision. Further, for the first time, we use the bootstrapping technique of the RNS-CKKS scheme in the proposed model, which enables us to evaluate an arbitrary deep learning model on the encrypted data. We numerically verify that the proposed model with the CIFAR-10 dataset shows 98.43% identical results to the original ResNet-20 model with non-encrypted data. The classification accuracy of the proposed model is 92.43%±2.65%, which is pretty close to that of the original ResNet-20 CNN model, 91.89%. It takes about 3 hours for inference on a dual Intel Xeon Platinum 8280 CPU (112 cores) with 172 GB memory. We think that it opens the possibility of applying the FHE to the advanced deep PPML model.

Keywords: Privacy-preserving machine learning · ResNet-20 · RNS-CKKS FHE scheme · SEAL library · Software implementation

*The first two authors contributed equally.

1 Introduction

The privacy-preserving issue is one of the most practical problems for machine learning recently. Fully homomorphic encryption (FHE) is the most appropriate tool for privacy-preserving machine learning (PPML) to ensure strong security in the cryptographic sense and satisfy the communication's succinctness. FHE is an encryption scheme whose ciphertexts can be processed with any deep Boolean circuits or arithmetic circuits without access to the data. The security of FHE has been usually defined with indistinguishability under chosen-plaintext attack (IND-CPA) security, which is a standard cryptographic security definition. If the client sends the public keys and the encrypted data with an FHE scheme to the PPML server, the server can perform all computation needed in the desired service before sending the encrypted output to the client. Therefore, the application of FHE to PPML has been researched much until now.

The most successful PPML model on the homomorphically encrypted data until now was constructed with the TFHE homomorphic encryption scheme by Lou and Jiang [LJ19], but it used the leveled version of the TFHE scheme without bootstrapping, which is not an FHE scheme. In other words, they chose in advance the parameters that can be used to perform the desired network without bootstrapping. If we want to design a deeper neural network with the leveled homomorphic encryption scheme, much impractically larger parameters have to be used, and it causes heavy run-time or memory overhead. Further, since the packing technique cannot be applied easily in the TFHE scheme, it can cause additional inefficiency with regard to the running time and the memory overhead if we want to process many data at once. Thus, it is desirable to use the FHE with moderate parameters and bootstrapping, which naturally supports the packing technique in the PPML model.

The applicable FHE schemes with this property are word-wise FHE schemes, such as Brakerski-Fan-Vercauteren (BFV) scheme [FV20] or Cheon-Kim-Kim-Song (CKKS) scheme [CKKS17, CHK$^+$18b]. Especially, the CKKS scheme has gained lots of interest for a suitable tool of the PPML implementation since it can deal with the encrypted real number naturally. However, these schemes support only homomorphic arithmetic operations such as the homomorphic addition and the homomorphic multiplication. Unfortunately, the popular activation functions are usually non-arithmetic functions, such as ReLU, sigmoid, leaky ReLU, and ELU. Thus, these activation functions cannot be evaluated directly using the word-wise FHE scheme. When the previous machine learning models using FHE replaced the non-arithmetic activation function with the simple polynomials, these models were not proven to show high accuracy for advanced classification tasks beyond the MNIST dataset.

Even though many machine learning models require multiple deep layers for high accuracy, there is no choice but to use a small number of layers in previous FHE-based deep learning models until FHE schemes' fast and accurate bootstrapping techniques become very recently available. The bootstrapping technique transforms a ciphertext that cannot support the homomorphic multiplication further to a fresh ciphertext by extending the levels of the ciphertext [Gen09, CHK$^+$18a]. However, the bootstrapping technique has been actively improved in regard to algorithmic time complexity [CCS19, HK20, BMTPH20], precision [LLL$^+$20a], and implementation [JKA$^+$21], which make bootstrapping more practical. The PPML model with many layers has to be implemented with the precise and efficient bootstrapping technique in FHE. In addition, since the training process is generally pretty expensive process as it requires many images and the large running time, it is more desirable to use the pre-trained parameters trained for the original standard plaintext machine learning model without any additional training process.

1.1 Our Contribution

For the first time, we implement the ResNet-20 model for the CIFAR-10 dataset [KH$^+$09] using the residue number system CKKS (RNS-CKKS) [CHK$^+$18b] FHE scheme, which is a variant of the CKKS scheme using the SEAL library 3.6.1 version [Mic21], one of the most reliable libraries implementing the RNS-CKKS scheme. In addition, we implement bootstrapping of RNS-CKKS scheme in the SEAL library according to [CHK$^+$18a, CCS19, HK20, BMTPH20, LLL$^+$20a] in order to support a large number of homomorphic operations for a deep neural network, as the SEAL library does not support the bootstrapping operation. ResNets are one of the historic convolutional neural network (CNN) models which enable a very deep neural network with high accuracy for complex datasets such as the CIFAR-10 and the ImageNet. Many high-performance works for image classification are based on ResNets since these models can reach sufficiently high classification accuracy by stacking more layers. We firstly apply the ReLU function based on the composition of minimax approximate polynomials [LLL$^+$21] to the encrypted data. Using the results, we show the possibility of applying the FHE with the bootstrapping to the standard deep machine learning model by implementing the ResNet-20 over the RNS-CKKS scheme. The implemented bootstrapping can support sufficiently high precision to successfully use the bootstrapping in the ResNet-20 with the RNS-CKKS scheme for the CIFAR-10 dataset.

Boemer et al. [BCD$^+$20] pointed out that all existing PPML models based on FHE or MPC are vulnerable to the model extraction attack. One of the reasons for this problem is that the previous PPML methods with the FHE scheme do not evaluate the softmax with the FHE scheme. It just sends the result before the softmax function, and then it is assumed that the client computes the softmax by itself. Thus, the information about the model can be extracted with lots of input-output pairs to the client. It can be desirable for the server to evaluate the softmax function with FHE. We firstly implement the softmax function in the machine learning model using the method in [CKKS17], and this is the first implementation of a privacy-preserving machine learning model based on FHE mitigating the model extraction attack.

We prepare the pre-trained model parameters by training the original ResNet-20 model with the CIFAR-10 plaintext dataset and perform the privacy-preserving ResNet-20 with these plaintext pre-trained model parameters and encrypted input images. We find that the inference result of the proposed implemented privacy-preserving ResNet-20 is 98.43% identical to that of the original ResNet-20. It achieves 92.43%$\pm$2.65% classification accuracy, which is quite close to the original accuracy of 91.89%. Thus, we verify that the proposed implemented PPML model successfully performs the ResNet-20 on the encrypted data even with the model parameters trained for the plaintext model.

1.2 Related Works

HE-friendly Network Some previous works re-design the machine learning model compatible with the HE scheme by replacing the standard activation functions with the simple non-linear polynomials [GBDL$^+$16, CBL$^+$18, vEPIL19, BCL$^+$20, HTG17], called the HE-friendly network. Although the highest classification accuracy of the HE-friendly CNN with the simple polynomial activation function implemented by word-wise HE is 91.5% for the CIFAR-10 dataset [HTG17], better PPML machine learning model has not been shown until now. It may suggest that these machine learning models are usually successful only for the simple dataset and cannot reach sufficiently high accuracy for the advanced dataset. Since the choice of the activation functions is sensitive in the advanced machine learning model, it may not be desirable to replace the standard and famous activation functions with simple arithmetic functions. Moreover, the additional pre-training process has to be followed before the PPML service is given. Since the training process is pretty time-consuming and needs quite a lot of data, it is preferable to use the standard model parameters of ResNets and VGGNets trained for plaintext data when the data privacy of the testing dataset has to be preserved.

Hybrid Model with FHE and MPC Some previous works evaluate the non-arithmetic activation functions with the multiparty computation technique to implement the standard well-known machine learning model preserving privacy [JVC18, RCK$^+$20, BLCW19, BCCW19, BCD$^+$20]. Although this method can evaluate even the non-arithmetic functions exactly, the privacy for the model information can be disclosed. In other words, the client should know the used activation function in the model, which is not desirable for the PPML servers. Also, since the communication with the clients is not succinct, the clients have to be involved in the computation, which is not desirable for the clients.

PPML with Leveled Homomorphic Encryption Some works use leved homomorphic encryption scheme to implement the standard machine learning model. The representative example is the work of Lou and Jiang [LJ19], which implements the ResNet-20 for the CIFAR-10 dataset or the ResNet-18 for the ImageNet dataset with leveled version of TFHE scheme. When we use leveled homomorphic encryption scheme, we should set parameters capable of the depth consumption for the desired circuit. Thus, if we want to homomorphically evaluate the deeper circuits, we have to set large parameters. This property of the leveled homomorphic encryption scheme makes difficult to evaluate more deep learning model because the required parameters may be rather impractical to the general computing environment. Further, the running time of each homomorphic encryption becomes larger, and thus the total running time could be asymptotically larger than the linear time with the circuit depth. On the other hand, FHE scheme uses practical parameters with fixed size regardless of the circuit depth, and the total running time can be just linear with the circuit depth. Therefore, for the practical deep learning models which has large circuit depths, the implementation of the deep learning model using the FHE scheme is an important research topic.

2 Preliminaries

2.1 RNS-CKKS Scheme

The CKKS scheme [CKKS17] is an FHE scheme supporting the arithmetic operations on encrypted data over real or complex numbers. Any users with the public key can process the encrypted real or complex data with the CKKS scheme without knowing any private information. The security of the CKKS scheme is based on the Ring-LWE hardness assumption. The supported homomorphic operations are the addition, the scalar multiplication, the non-scalar multiplication, the rotation, and the complex conjugation operation, and each operation except the homomorphic rotation operation is applied component-wisely. While the scalar multiplication is the multiplication with the plaintext, the non-scalar multiplication is the multiplication with the ciphertext. The rotation operation homomorphically performs a cyclic shift of the vector by some step. The non-scalar multiplication, rotation, and complex conjugation operations in the CKKS scheme need additional corresponding evaluation keys and the key-switching procedures.

Each real number data is scaled with some big integer, called the scaling factor, and then rounded to the integer before encrypting the data. When the two data encrypted with the CKKS scheme are multiplied homomorphically, the scaling factors of the two data are also multiplied. This scaling factor should be reduced to the original value using the rescaling operation for the following operations.

Since the CKKS scheme needs pretty big integers, the original CKKS scheme uses a multi-precision library, which requires higher computational complexity. To reduce the complexity, the residue number system variant of the CKKS scheme [CHK+18b], called the RNS-CKKS scheme, was also proposed. In the residue number system, the big integer is split into several small integers, and the addition and the multiplication of the original big integers are equivalent to the corresponding component-wise operations of the small integers. We use the RNS-CKKS scheme in this paper. We show some required operations in RNS-CKKS scheme in the followings.

We denote the homomorphic addition, the homomorphic scalar multiplication, and the homomorphic non-scalar multiplication as $\oplus, \odot, \otimes$ in this paper. The homomorphic rotation operation for left rotation with r steps is denoted as $\texttt{rot}(\texttt{ct}, r)$.

2.2 Baby-Step Giant-Step Polynomial Evaluation

In order to utilize homomorphic encryption, many non-arithmetic operations of ResNets and bootstrapping must be approximated using high-order polynomials. When we evaluate a polynomial for the encrypted input with RNS-CKKS scheme, it is important to reduce the number of non-scalar multiplication and the depth consumption as much as possible. A well-known polynomial evaluation method efficient in non-scalar multiplications and depth consumption is the baby-step giant-step polynomial evaluation method.

Bossuat et al. [BMTPH20] suggests a variant of the baby-step giant-step polynomial evaluation method that guarantees the optimal depth consumption with small additional non-scalar multiplication. Since the depth consumption is generally more sensitive that the number of non-scalar multiplication because of the number of the bootstrapping, we use this variant as a default method for the homomorphic polynomial evaluation. Lee et al. [LLL+20b] suggest an efficient method for polynomials with only odd degree terms. These algorithms are elaborated with the proposed implementation based on binary tree in Section 3.1.

2.3 Bootstrapping of CKKS Scheme

The rescaling operation reduces both the scaling factor and the ciphertext modulus, which is necessary for each homomorphic multiplication. After several consecutive multiplications, the ciphertext modulus cannot be reduced further at some point. The bootstrapping operation of the CKKS scheme [CHK+18a] transforms the ciphertext with too small modulus into the fresh ciphertext with large modulus without changing the message. Therefore, any arithmetic circuits with large multiplicative depth can be obtained using the bootstrapping operation.

The bootstrapping of the CKKS scheme starts with raising the modulus of the ciphertext. Since the message polynomial becomes $m + q_0 I$ where q_0 is the level-0 modulus and I is some unknown integer polynomial, the modular reduction of the coefficients of the message polynomial should be performed homomorphically to remove $q_0 I$ part.

To move the coefficients of the message polynomials into the slots, CoeffToSlot is performed to the raised ciphertext. The core part of this operation is the matrix multiplication with a kind of Vandermonde matrix. Specifically, if we have U_0 and U_1 as

$$U_0 = \begin{bmatrix} 1 & \zeta_0 & \cdots & \zeta_0^{N/2-1} \\ 1 & \zeta_1 & \cdots & \zeta_1^{N/2-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \cdots & \zeta_{N/2-1}^{N/2-1} \end{bmatrix}, U_1 = \begin{bmatrix} \zeta_0^{N/2} & \zeta_0^{N/2+1} & \cdots & \zeta_0^{N-1} \\ \zeta_1^{N/2} & \zeta_1^{N/2+1} & \cdots & \zeta_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \zeta_{N/2-1}^{N/2} & \zeta_{N/2-1}^{N/2+1} & \cdots & \zeta_{N/2-1}^{N-1} \end{bmatrix}$$

then the CoeffToSlot operation is the homomorphic evaluation of two formulas $\mathbf{z}_i' = \frac{1}{N}(\bar{U}_k^T \cdot \mathbf{z} + U_k^T \cdot \bar{\mathbf{z}})$ for $k = 0, 1$. Chen et al. [CCS19] proposed the FFT-like optimization technique for this operation. They observed these matrices are FFT-friendly matrix in that some kind of butterfly structure can be applied to the operation. To give a trade-off between the running time and the depth consumption, they also proposed the level collapsing technique, where several layers in the butterfly structure are merged to reduce the depth consumption to desired value. We use this FFT-like optimization technique and the level collapsing technique in our implementation.

Next, we perform the homomorphic modular reduction to the converted ciphertext, called ModReduction. There are several techniques and we elaborate only the techniques we used to implement. Lee et al. [LLL+20a] proposed that the modular reduction is represented by the composition of several functions $h_3 \circ h_2^\ell \circ h_1$ such that

$$h_1(x) = \cos\left(\frac{2\pi}{2^\ell}\left(x - \frac{1}{4}\right)\right), h_2(x) = 2x^2 - 1, h_3(x) = \frac{1}{2\pi}\arcsin x.$$

Then h_1 and h_3 are approximated with the minimax approximate polynomials in the approximation regions. Lee et al. also proposed the improved multi-interval Remez algorithm to obtain the minimax approximate polynomial for piecewise continuous functions. This approximation needs several parameters. K is the parameter determining the number of the approximation intervals, and ϵ is the half width of each interval. The approximation region is $\cup_{i=-(K-1)}^{K-1}[i-\epsilon, i+\epsilon]$. While the parameter ϵ is related to the range and the precision of the input message data, the parameter K is related to the bootstrapping failure. The additional parameters is the polynomial degree of the approximate polynomials of h_1 and h_3 functions. The homomorphic evaluation of the polynomials are performed by the baby-step giant-step polynomial evaluation algorithm in Section 2.2.

Then, the modular-reduced slots is reverted to the coefficients of the message polynomial by SlotToCoeff. The SlotToCoeff operation is the homomorphic evaluation of the formula $\mathbf{z} = U_0 \cdot z_0 + U_1 \cdot z_1$. This operation can be also optimized with the FFT-like optimization and the level collapsing technique [CCS19].

2.4 Minimax Composition of ReLU

Lee et al. [LLL+21] show that the ReLU function has to be approximated with sufficiently high precision if we use the pre-trained model parameters with the original ResNet-20 model. We need a polynomial with quite a large degree if a single minimax polynomial approximates the ReLU function, and it needs a pretty big running time to evaluate homomorphically. Instead of using the single minimax polynomial for the ReLU function, they use the formula $\mathrm{ReLU}(x) = \frac{1}{2}x(1+\mathrm{sign}(x))$ and approximate $\mathrm{sign}(x)$ by the minimax composition of the small degree polynomials [LLNK20]. It reduces the running time of the homomorphic evaluation of the ReLU function, and this approximation method makes more practical the homomorphic evaluation of the non-arithmetic functions such as the ReLU function.

Lee et al. [LLNK20] specified the method to find the optimal composite polynomials for sign function. When they find each polynomial composing the composite polynomial, they used the range of the previous polynomial as the approximation domain for the next polynomial. If each polynomial is the minimax approximate polynomial of the sign function for each domain, the range of each polynomial is always two intervals symmetric to the origin. Each degree of the element polynomial requiring minimal non-scalar multiplications for desired precision are found by dynamic programming algorithm.

3 New Consideration for ResNet-20 on RNS-CKKS Scheme

To implement the ResNet-20 model with the RNS-CKKS scheme, there are three points to be newly considered: binary tree based implementation for polynomial evaluation, natural implementation for the strided convolution, and implemenation for the softmax function.

3.1 Binary Tree Based Implementation of Polynomial Evaluation

For more intuitive and systemetic implementation, we modify the baby-step giant-step polynomial evaluation algorithm using binary tree data structure. There is a pre-computation process for recursively dividing by the division algorithm for the polynomial, which is shown in Algorithm 1. The output of `DividePoly` is a binary tree which is useful in homomorphic polynomial evaluation process.

Algorithm 2 shows the binary tree-based baby-step giant-step polynomial evaluation algorithm. For the optimal depth consumption, we may further divide the leftmost leaf node as modification of Bossuat et al.[BMTPH20] in Lines 3–13. We generalize the giant step degree as arbitrary integer, rather than power-of-two integer as in [LLL+20b].

Then, Lines 15–18 homomorphically evaluate for the polynomial in non-leaf nodes and the leaf nodes. $T_n(x)$ means the n-th Chebyshev polynomial. Chebyshev polynomials have a recursive formula as follows,

$$T_{m+n}(x) - T_{m-n}(x) = 2T_m(x)T_n(x)$$

where $m \geq n$. When we homomorphically evaluate the Chebyshev polynomials in Lines 15, 17, we use the formula with $m = n$ where $T_0(x)$ is 1. When we homomorphically evaluate other Chebyshev polynomials in Line 16, we set m as the most large power-of-two integer less than the degree and n as the difference between the degree and m.

Lines 19–26 reduce the binary tree until the binary tree has only the root node by homomorphically evaluating the polynomials for non-leaf nodes having two leaf nodes. This implementation is essentially the same as the method in [BMTPH20], but it is more easy to design the implementation for the algorithm.

Lee et al. [LLL+20b] suggest the method for polynomials with only odd degree terms. They observed if k is even, there is no need to evaluate the Chebyshev polynomials with even degree that is not power-of-two integer in Line 16. If we denote `OddPolyEval` rather than `PolyEval` in the following section, we omit these polynomial evaluation process.

Algorithm 1: `DividePoly`$(p; k)$

```
   Input   : A degree-$d$ polynomial $p$, a giant step parameter $k$
   Output  : A binary tree $P$ with leaf having polynomials
 1 if $d < k$ then
 2   | return a binary tree $P$ with a single root node having $p$
 3 else
 4   | Find $m$ such that $k \cdot 2^{m-1} < d \leq k \cdot 2^m$.
 5   | Generate a binary tree $P$ with a single root node having $T_{k \cdot 2^{m-1}}$.
 6   | Divide $p$ by $T_{k \cdot 2^{m-1}}$ to obtain the quotient $q$ and the remainder $r$.
 7   | Generate a binary tree $Q$ using DividePoly$(q; k)$.
 8   | Generate a binary tree $R$ using DividePoly$(r; k)$.
 9   | Append $Q, R$ to the left child and the right child of the root in $P$, respectively.
10   | return $P$
11 end
```

3.2 Strided Convolution

Juvekar et al. [JVC18] proposed an efficient convolution operation for the packing structure in FHE scheme. They also proposed strided convolution operation on the homomorphic encryption scheme by decomposing the strided convolution into a sum of non-strided convolutions. However, their proposed strided convolution operation is not natural for the

Algorithm 2: `PolyEval`$(\mathsf{ct}, p; k)$

```
Input  : A ciphertext $\mathsf{ct} = \mathsf{Enc}(x)$, a degree-$d$ polynomial $p$
Output : A ciphertext of $p(x)$
1  Generate a binary tree $P$ using DividePoly$(p; k)$.
2  $l \leftarrow \lceil \log k \rceil - 1$
3  if $P$ is a full binary tree and the leftmost leaf polynomial has degree more than $2^l$ then
4      $p_0 \leftarrow$ the leftmost leaf polynomial
5      $V \leftarrow$ the leftmost leaf node
6      while the polynomial in $V$ has degree more than $2^l$ do
7          Replace $p_0$ with $T_{2^l}$ in $V$.
8          Divide $p_0$ by $T_{2^l}$ to obtain the quotient $q$ and the remainder $r$.
9          Append $q, r$ to the left child and the right child of $V$.
10         $V \leftarrow$ the left child of $V$
11         $l \leftarrow l - 1$
12     end
13 end
14 $l \leftarrow \lceil \log k \rceil - 1$
15 Homomorphically evaluate $T_2(x), T_4(x), \cdots, T_{2^l}(x)$ using $T_{2n}(x) = 2T_n(x)^2 - 1$.
16 Homomorphically evaluate other $T_n(x)$ for $3 \leq n \leq k$.
17 Homomorphically evaluate $T_{2k}(x), \cdots, T_{2^{m-1} \cdot k}(x)$.
18 Evaluate all of leaf node polynomials using the pre-computed ciphertexts.
19 while $P$ has only a root node do
20     $V \leftarrow$ one of the non-leaf nodes that have two leaf child
21     $\mathsf{ct}_T \leftarrow$ ciphertext for the polynomial $(T(x))$ in $V$ (pre-computed in Line 15, 17)
22     $\mathsf{ct}_q \leftarrow$ ciphertext for the polynomial $(q(x))$ in left child of $V$
23     $\mathsf{ct}_r \leftarrow$ ciphertext for the polynomial $(r(x))$ in right child of $V$
24     $\mathsf{ct}_T \leftarrow \mathsf{ct}_q \otimes \mathsf{ct}_T \oplus \mathsf{ct}_r$
25     Replace $T(x)$ with $q(x)T(x) + r(x)$ in node $V$ and remove the childs of $V$
26 end
27 return $\mathsf{ct}_p$ for input polynomial $p(x)$
```

packing structure in the RNS-CKKS scheme. Further, the following operations after their strided convolution are difficult to be performed on the RNS-CKKS scheme.

We propose an efficient and natural method for the strided convolution on RNS-CKKS scheme. Instead of decomposing the strided convolution, we regard the output of the strided convolution as a part of the non-strided convolution. Indeed, the output data for the non-strided convolution include the output data for the strided convolution. If we perform the non-strided convolution, there are some gaps between the required output data for the strided convolution, which is not completely uniform but some regular sense. The slot structure of the output data of the strided convolution in the output slots of the non-strided convolution is shown in Fig. 1.

We also find that this slot structure with the regular gaps is compatible with the following ReLU functions, non-strided convolution operations, and even strided convolution operations. Since the ReLU function is evaluated component-wisely, this slot structure does not count for the ReLU function. The non-strided convolution to the slot structure after the proposed strided convolution can be performed with the Gazelle's convolution method with all rotation steps doubled. The additional strided convolution to the slot structure after the strided convolution can be performed with the non-strided convolution for this slot structure followed by additional filtering. With these convolution methods,

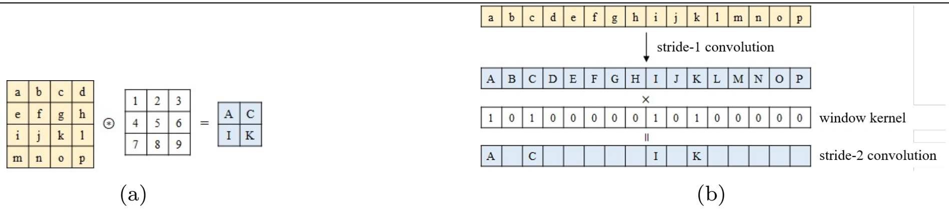

Figure 1: Stride-2 convolution (a) plaintext (b) ciphertext.

we can perform non-strided and strided convolution operations even after several strided convolutions.

In the ResNet-20, we only use the convolution with stride one or two, and thus we assume that the strided convolution is the convolution with stride two. Each convolution operation should be given an additional parameter, slotstr, that represents the slot structure for the meaningful data in the input ciphertext of each convolution. The parameter slotstr is stored in each ciphertext for each channel and initialized with zero, and it is added by one only when the strided convolution is performed. If the non-strided convolution is performed, we perform Gazelle's convolution method with the steps multiplied by 2^{slotstr}. If the strided convolution is performed, we perform the same procedure as the non-strided convolution except the following filtering. Specific algorithm for the strided convolution will be suggested in Section 5.3.

3.3 Approximation for Softmax

We implement the softmax function in our privacy-preserving ResNet-20 implementation for the security against the model extraction attack. The softmax function is $e^{x_i}/\sum_{j=0}^{T-1} e^{x_j}$ for each $i = 0, \cdots, T-1$ where T is the number of the classification types. Since the softmax function was not implemented in the previous works for the PPML with homomorphic encryption, the approximation method for the softmax function should be newly designed. There are two non-arithmetic operations in the process of the softmax function, the exponential function and the inverse function.

We use different approximation techniques to these non-arithmetic function because of the difference in the characteristic of the input values. The absolute input values of the exponential function are dozens, but the output values of that function are rather unstable. On the other hand, the input values of the inverse function are unstable in that each scale of the input value is different from each other's input value. With these characteristics, we choose the following approximation methods, and whole algorithm is suggested in Section 5.7.

Exponential Function If we simply approximate the exponential functions on a desired interval, the approximation may not be so accurate in that the scales of the output for the exponential function can be too various. Assume that we have to approximate the exponential function e^x in $[-B, B]$. Then, we can regard the function as $(e^{x/B})^B$. Note that we may approximate e^y in $[-1, 1]$ when we set $y = x/B$, and the exponential function in this interval is easy to approximate. Thus, we approximate the exponential function in $[-1, 1]$ with the least square method, and we find that the approximate polynomial with degree 12 can approximate sufficiently precisely. Then, when we homomorphically evaluate the exponential function in $[-B, B]$, we divide the input by B, evaluate the approximate polynomial for the exponential function, and exponentiate with B. If we set B as power-of-two integer, exponentiation with B can be implemented by repeated squaring.

Inverse Function While the exponential function has a range with various scales, the inverse function in the softmax function has a domain with various scales. This characteristic makes the approximation of the inverse function difficult with some ordinary polynomial approximation, even with some scaling of the input. In this case, Goldschmidt division method is appropriate for the evaluation of the inverse function on the input with various scale [Gol64, CKK+19]. In Goldschmidt division method, the following formula is used,

$$\frac{1-x^{2^n}}{1-x}=\prod_{i=0}^{n-1}(1+x^{2^i}).$$

If $|x|< 1$, the left term of the above formula converges to $1/(1-x)$ very fast even with small n. When we substitute $y=1-x$, the inverse function $1/y$ can be approximated as $\prod_{i=0}^{n-1}(1+(1-y)^{2^i})$ when $0<y<2$. Note that even if y is close to zero, the approximated inverse function value amplified to very large number. This characteristic cannot be satisfied with the ordinary polynomial approximation methods. The characteristic can be used to reserve the role of the softmax function for generating one-hot vector even when we approximate the softmax function.

When the range of the input is $(0, 2R]$, we regard the inverse function on the range as $1/R \cdot 1/(y/R)$. In other words, the input value is multiplied with $1/R$, evaluated by the inverse function with Goldschmidt division method, and multiplied with $1/R$ again. Note that R will be very large number, and the input may be far less than R. Even if y/R is very close to zero, the Goldschmidt method stably evaluates the inverse function as mentioned above.

Gumbel Softmax Function If the input value of the softmax function is somewhat large, the bound B for the range of the exponential function should be extremely large. If the value R is set to a fixed value for a sufficient precision of the inverse function, the input value of the inverse function can be larger than $2R$. In this case, we may use the Gumbel softmax technique, which evaluates the following function instead of the softmax function,

$$\frac{e^{x_i/\lambda}}{\sum_{j=0}^{T-1} e^{x_j/\lambda}}$$

where λ is an additional parameter. If we use the Gumbel softmax function, the output vector is still similar to one-hot vector, and thus the model extraction attack can be mitigated enough. Further, the range of the exponential function is reduced from B to $B^{1/\lambda}$, the input of the inverse function will be included in $(0, 2R]$.

4 Position of Bootstrapping

As we first use the bootstrapping operation in a machine learning model, we should consider when we perform the bootstrapping operation in the middle of the ResNet-20 model. In this section, we analyze several factors about the position of the bootstrapping operation affecting the efficiency.

The key-switching operation is the heavist operation in the homomorphic operations in RNS-CKKS scheme, and thus the non-scalar multiplication, the rotation, and the complex conjutation requiring the key-switching operation are far heavier than the addition and the scalar multiplication. For this reason, the number of the key-switching operations roughly determines the total amount of operations.

There is a major additional factor affecting the total amount of operations, the level of the ciphertext for the key-switching operation. The key-switching operation includes the decomposing operation, the mult-sum operation, and the mod-down operation. While the mod-down operation is linear with the level of the input ciphertext, the decomposing operation and the mult-sum operation are quadratic with the level of the input ciphertext. Since the most time-consuming operation among the three procedures is the decomposing operation, the key-switching operation is quadratic function with the level. This quadratic property shows the large effect of the level of the ciphertext to the key-switching operation and the total amount of the operations.

This quadratic property is numerically confirmed with SEAL library as in Fig. 2. Fig. 2 shows the running time of the rotation operation for the various levels of the input ciphertext, where the most part of the rotation operation is the key-switching operation, and it also shows the graph of the square root of the running time to represent the quadratic property more clearly. The square root of the running time is almost the linear function with the level, which confirms the quadratic property. Thus, the sum of the squared level of each input ciphertext of each key-switching operation can simulate much more closely than the number of the key-switching operations.

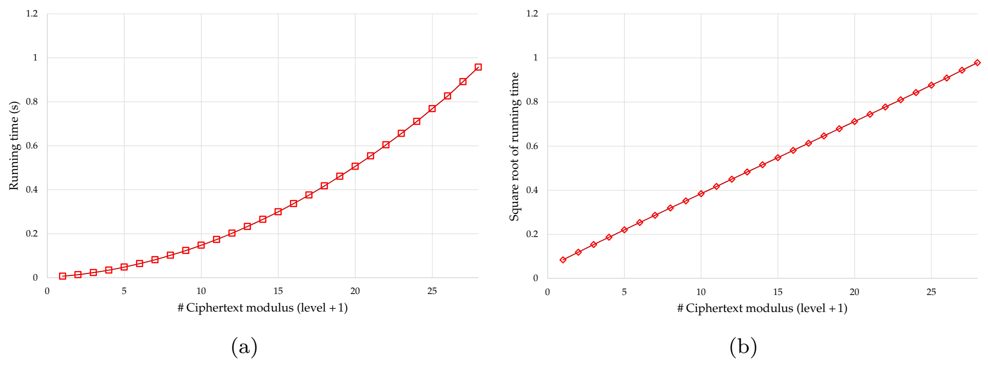

Figure 2: Running time for the rotation operation for various number of ciphertext modulus with $N = 2^{16}$ (a) T_{rot}-$(\ell + 1)$ graph (b) $\sqrt{T_{\text{rot}}}$-$(\ell + 1)$ graph.

Although the most time-consuming operation in the ResNet-20 is the bootstrapping operation, the level of the ciphertexts for each key-switching operation in the bootstrapping is rather fixed regardless of the structure of the ResNet-20. Thus, it is desirable to compare the number of the key-switching operations of the convolution operation and the ReLU function. We note that the number of the key-switching operations in the convolution operation is far more than that in the ReLU function because of many rotation operation in the convolution operation.

This fact suggests that it is desirable to perform the bootstrapping just after the convolution operation. Then, the convolution operation is performed in the lowest level of the ciphertext, and lots of rotation operations in the convolution operation will be significantly reduced. Numerical comparison for this analysis will be dealt with in Section 6.

5 Implementation Details of ResNet-20 on RNS-CKKS

5.1 Structure

Fig. 3 shows the structure of the ResNet-20 model and Table 1 shows the specification of the ResNet-20. With this structure, We design our implemented structure for the ResNet-20 with the RNS-CKKS scheme as shown in Fig. 4, where it consists of convolution (Conv), batch normalization (BN), ReLU, bootstrapping (Boot), average pooling (AP), fully connected layer (FC), and softmax. This model is virtually identical to the original ResNet-20 model except that the bootstrapping procedures are added. All of these procedures will be specified in the following subsections.

Table 1: The specification of the ResNet-20 (CIFAR-10)

Layer		Input Size	#Inputs	Filter Size	#Filters	Output Size	#Outputs
Conv1		32×32	3	3×3	16	32×32	16
Conv2	2-1	32×32	16	3×3	16	32×32	16
	2-2	32×32	16	3×3	16	32×32	16
	2-3	32×32	16	3×3	16	32×32	16
Conv3	3-1-1	32×32	16	3×3	32	16×16	32
	3-1-2	16×16	32	3×3	32	16×16	32
	3-1-s	32×32	16	1×1	32	16×16	32
	3-2	16×16	32	3×3	32	16×16	32
	3-3	16×16	32	3×3	32	16×16	32
Conv4	4-1-1	16×16	32	3×3	64	8×8	64
	4-1-2	8×8	64	3×3	64	8×8	64
	4-1-s	16×16	32	1×1	64	8×8	64
	4-2	8×8	64	3×3	64	8×8	64
	4-3	8×8	64	3×3	64	8×8	64
Average Pooling		8×8	64	8×8	64	-	64
Fully Connected		64×1	1	-	-	-	10

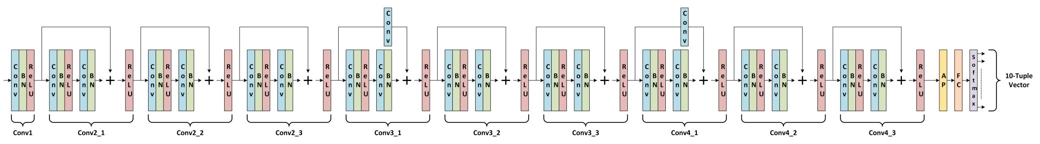

Figure 3: Structure of ResNet-20.

5.2 General Setting for RNS-CKKS Scheme

5.2.1 Parameters

We set the ciphertext polynomial degree as 2^{16} and the secret key Hamming weight as 64. The bit length of base modulus (q_0), special modulus, and default modulus are set as 60, 60, and 50, respectively. The bit length of modulus in the bootstrapping range is the same as that of q_0. The numbers of levels for the general homomorphic operations and the bootstrapping are set as 11 and 13, respectively. The maximum bit length of modulus is 1450, which satisfies 111.6-bit security. The security level λ is computed based on Cheon et al.'s hybrid dual attack [CHHS19], which is the fastest attack for the LWE with the sparse key. Table 2 lists the above parameters.

Table 2: RNS-CKKS parameter settings

λ	Hamming Weight	Degree	Modulus Q	q_0	Special Prime	Scaling Factor	Evaluation Level	Bootstrapping Level
111.6	64	2^{16}	1450 bits	60 bits	60 bits	50 bits	11	13

5.2.2 Data Packing

The message is a $32 \times 32 \times 3$ CIFAR-10 RGB image, and a single image is processed at a time. We can use 2^{15} message slots in one ciphertext with our parameters, which is the half polynomial degree. Rather than using the full slots of the ciphertext, we employ the sparse packing method [CHK+18a] to pack a channel of a CIFAR-10 image in one ciphertext using only 2^{10} sparse slots. This is because the bootstrapping of sparsely packed ciphertext takes much less time than that of fully packed ciphertext and the convolution operations can be performed more smoothly with the minimal rotation operations.

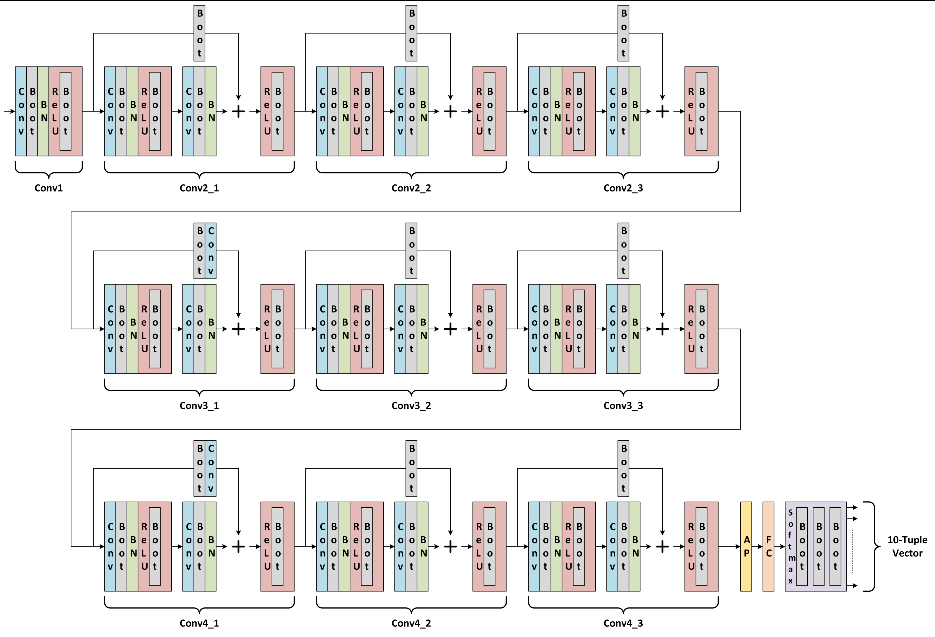

Figure 4: Proposed structure of ResNet-20 over RNS-CKKS scheme.

We construct the structure type for the encrypted tensor. In our implementation, it is not sufficient only to have ciphertexts composing the encrypted data, but we have to store slot structure parameter generated by the strided convolution. For the ease of understanding, we also store the dimension of the tensor to the encrypted tensor. An encrypted tensor $\mathsf{Tensorct}$ for a tensor in $\mathbb{R}^{\ell\times\ell\times h}$ is the form of $(\{\mathsf{ct}_k\}_k, \ell, \mathsf{slotstr}, h)$, where $\{\mathsf{ct}_k\}_k$ is an array of the ciphertexts composing the encrypted tensor and $\mathsf{slotstr}$ is the slot structure parameter. Algorithm 3 shows the detailed algorithm to encrypt image tensors.

Algorithm 3: $\texttt{EncTensor}(A \in \mathbb{R}^{L\times L\times H})$

```
   Input  : A tensor $A \in \mathbb{R}^{L\times L\times H}$
   Output : An encrypted tensor $\mathsf{Tensorct}$
1  for $k = 0$ to $H-1$ do
2      $v_k \leftarrow \mathbf{0} \in \mathbb{R}^{L^2}$
3      for $i = 0$ to $L-1$ do
4          for $j = 0$ to $L-1$ do
5              $x \leftarrow i \cdot L + j$
6              $v_k[x] \leftarrow A[i,j,k]$
7          end
8      end
9      $\mathsf{ct}_k \leftarrow \mathsf{Enc}(v_k; N, L^2)$
10 end
11 return $(\{\mathsf{ct}_k\}_{k=0,\cdots,H-1}, L, 0, H)$
```

5.2.3 Data Range and Precision

Any polynomials can approximate continuous functions only in some bounded set. If even one value in the message slots exceeds this bounded set, the absolute value of output diverges to a pretty big number, leading to complete classification failure. Since FHE can only handle arithmetic operations, polynomial approximation should be used for non-arithmetic operations such as the ReLU function, the bootstrapping, and the softmax function. Thus, the inputs for these procedures should be in the bounded approximation region. We analyze the absolute input values for the ReLU, the bootstrapping, and the softmax when performing the ResNet-20 with several images. Since the observed maximum absolute input value for these procedures is 37.1, we conjecture that the absolute input values for these procedures are less than 40 with a very high probability. We use this observation in the implementation of each procedure. We also empirically find that the precision of the approximate polynomial or the function should be at least 16-bit below the decimal point, and thus we approximate each non-arithmetic function with 16-bit average precision.

5.2.4 Optimization for Precision of Homomorphic Operations

We apply several methods to reduce the rescaling error and relinearization error and ensure the precision of the resultant message, such as the scaling factor management in [KPP20], lazy rescaling, and lazy relinearization [BGP+20, LLK+20]. The lazy rescaling and relinearization can also be applied to reduce the computation time as it requires much computation due to the number-theoretic transformation (NTT) and gadget decomposition.

5.3 Convolution and Batch Normalization

Most of the operations in the ResNet-20 are convolutions with zero-padded input to maintain their size. We use the packed single input single output (SISO) convolution with stride 1 used in Gazelle [JVC18]. The strided convolution with stride 2 is also required to perform down-sampling, and it is performed by the proposed method in Section 3.2. Algorithm 4 shows the detailed algorithm for convolution, and it includes both the non-strided convolution and the strided convolution. The non-strided convolution is performed when str is 1, and the strided convolution is performed when str is 2. Each rotation steps are multiplied by the slotstr value as discussed in Section 3.2.

Since the batch normalization procedure is a simple linear function with constant coefficients, it can be implemented with the homomorphic addition and the homomorphic scalar multiplication.

5.4 ReLU

For the first time, we implement the ReLU function in the ResNet-20 with the RNS-CKKS scheme using the composition of the minimax polynomial approximation by Lee et al.[LLL+21]. To find an appropriate precision value, we repeatedly perform the ResNet-20 simulation over the RNS-CKKS scheme while changing the precision, and as a result, we find that the minimum 16-bit precision shows good performance on average.

To synthesize the sign function for the ReLU approximation, we generate the composition of the small minimax approximate polynomials with precision parameter $\alpha = 12$ using the three minimax approximate polynomials with degrees 7, 15, and 27. Algorithm 5 shows the algorithm generating the composite polynomials approximating the sign function [LLNK20]. `GenMinimax`(f, d, D) in Algorithm 5 is an algorithm generating the minimax approximate polynoimal with degree d for function f on the domain D, and we implement this algorithm by the multi-interval Remez algorithm [LLL+20a]. Range(f, D) means the range of f on the domain D.

Algorithm 4: $\texttt{Conv}(\mathsf{Tensorct}, W \in \mathbb{R}^{c\times c\times t\times t'}, \mathsf{stride})$

```
Input  : An encrypted tensor $\mathsf{Tensorct} = (\{\mathsf{ct}_k\}_{k=0,\cdots,t-1}, \ell, \mathsf{slotstr}, t)$, weight
         parameters $W \in \mathbb{R}^{c\times c\times t\times t'}$ ($c$ is an odd integer), and the stride of the
         convolution operation $\mathsf{str}$
Output : An output encrypted tensor $\mathsf{Tensorct}'$
1  $L \leftarrow \ell \cdot \mathsf{slotstr}$
2  for $h = 0$ to $t' - 1$ do
3      $\mathsf{ct}'_h \leftarrow \mathbf{0}$
4      for $k = 0$ to $t - 1$ do
5          for $(i, j) = (0, 0)$ to $(c - 1, c - 1)$ do
6              $w \leftarrow \mathbf{0} \in \mathbb{R}^{L^2}$
7              for $(i', j') = (0, 0)$ to $(\ell - 1, \ell - 1)$ do
8                  if $0 \le i' + i - \lfloor c/2\rfloor \le \ell - 1$ and $i', j' = 0 \mod \mathsf{slotstr} \cdot \mathsf{str}$ then
9                      $w[i' \cdot \ell + j'] \leftarrow W[i, j, k, h]$
10                 end
11             end
12             $r \leftarrow (i - \lfloor c/2\rfloor) \cdot \ell + (j - \lfloor c/2\rfloor)$
13             $\mathsf{ct}'_h \leftarrow \mathsf{ct}'_h \oplus (w \odot \texttt{rot}(\mathsf{ct}_k, r \cdot \mathsf{slotstr}))$
14         end
15     end
16 end
17 return $(\{\mathsf{ct}'_h\}_{h=0,\cdots,t'-1}, \ell/\mathsf{str}, \mathsf{slotstr} \cdot \mathsf{str}, t')$
```

Algorithm 6 shows the homomorphic evaluation method for ReLU function using the composite polynomials generated by Algorithm 5 as input. After homomorphically evaluating p_i's in order, we homomorphically evaluate $x(1 + \mathrm{sign}(x))/2$.

This composition of polynomials ensures that the average approximation precision is about 16-bit precision. The homomorphic evaluation of the polynomials is carried out using the odd baby-giant method in [LLL+20b] and the optimal level consumption method in [BMTPH20]. Since the homomorphic evaluation of polynomial compositions consumes many depths, it is impossible to finish it without bootstrapping. Thus, we use bootstrapping twice in a layer, once in the middle and once at the end of evaluating the ReLU function.

5.5 Bootstrapping

Since we have to consume many depths to implement the ResNet-20 on the RNS-CKKS scheme, many bootstrapping procedures are required to ensure enough homomorphic multiplications. For the first time, we apply the bootstrapping technique to perform the deep neural network such as the ResNet-20 on the encrypted data and prove that the FHE scheme with the state-of-the-art bootstrapping can be successfully applied for privacy-preserving deep neural networks. Since the `SEAL` library does not support any bootstrapping technique, we implement the most advanced bootstrapping with the `SEAL` library [BMTPH20, LLL+20a, JKA+21]. The CoeffToSlot and the SlotToCoeff are implemented using collapsed FFT structure [CCS19] with depth 2. The ModReduction is implemented using the composition of the cosine function, two double angle formulas, and the inverse sine function [HK20, LLL+20a], where the cosine function and the inverse sine function are approximated with the multi-interval Remez algorithm as in [LLL+20a].

The most crucial issue when using the bootstrapping of the RNS-CKKS scheme is the bootstrapping failure. More than a thousand bootstrapping procedures are required in our

Algorithm 5: `GenSignPoly`$(\alpha, \{d_i\}_i)$

```
Input  : Precision parameter of sign function $\alpha$, sequence of composite polynomial
         degrees $\{d_i\}_{i=0,\cdots,s-1}$
Output : Sequence of composite polynomials for sign function $\{p_i\}_{i=0,\cdots,s-1}$ where
         $p_{s-1} \circ \cdots \circ p_0(x) \approx \mathrm{sign}(x)$
1 for $i = 0$ to $s-1$ do
2     if $i = 0$ then
3         $D_0 \leftarrow [-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$
4     else
5         $D_i \leftarrow \mathsf{Range}(p_{i-1}, D_{i-1})$
6     end
7     $p_i \leftarrow \mathsf{GenMinimax}(\mathrm{sign}, d_i, D_i)$
8 end
```

Algorithm 6: `ReLU`$(\mathsf{Tensorct}, \{p_i\}_i)$

```
Input  : An encrypted tensor $\mathsf{Tensorct} = (\{\mathsf{ct}_k\}_{k=0,\cdots,t-1}, \ell, \mathsf{slotstr}, t)$, sequence of
         composite polynomials for sign function $\{p_i\}_{i=0,\cdots,s-1}$
Output : An activated encrypted tensor with ReLU $\mathsf{Tensorct}'$
1 for $k = 0$ to $t-1$ do
2     $\mathsf{ct}'_k \leftarrow \mathsf{ct}_k$
3     for $i = 0$ to $s-1$ do
4         $\mathsf{ct}'_k \leftarrow$ OddPolyEval$(\mathsf{ct}'_k, p_i)$
5     end
6     $\mathsf{ct}'_k \leftarrow (0.5 \odot \mathsf{ct}_k) \otimes (1 + \mathsf{ct}'_k)$
7 end
```

model, and the result of the whole neural network can be largely distorted if even one of the bootstrapping procedures fails. The bootstrapping failure occurs when one of the slots in the input ciphertext of the ModReduction procedure is not on the approximation region. The approximation interval can be controlled by the bootstrapping parameters (K, ϵ), where the approximation region is $\cup_{i=-(K-1)}^{K-1}[i-\epsilon, i+\epsilon]$ [CHK+18a]. While the parameter ϵ is related to the range and the precision of the input message data, the parameter K is related to the values composing the ciphertext and not related to the input data. Since the values contained in the ciphertext are not predictable, we have to investigate the relation between the bootstrapping failure probability and the parameter K.

Table 3: Boundary of approximation region given key Hamming weight and failure probability of modular reduction

$\Pr(\lvert I_i\rvert \geq K)$	$h = 64$	$h = 128$	$h = 192$
2^{-23} [BMTPH20]	12	17	21
2^{-30}	14	20	24
2^{-40}	16	23	28

We describe how the bootstrapping failure affects the whole ResNet evaluation and propose a method to reduce the bootstrapping failure probability. As the CKKS bootstrapping is based on the sparsity of the secret key, there is a failure probability of bootstrapping.

Here is the reason why the approximated modular reduction in the previous CKKS bootstrapping has a certain failure probability. The decryption formula for a ciphertext (a, b) of the CKKS scheme is given as $a \cdot s + b = m + e \pmod{\mathcal{R}_q}$ for the secret key s; hence, $a \cdot s + b \approx m + q \cdot I \pmod{\mathcal{R}_Q}$, where the Hamming weight of s is h. As the coefficients of

a and b are in $[-\frac{q_0}{2}, \frac{q_0}{2})$, the coefficients of $a \cdot s + b$ have an absolute value less than $\frac{q_0(h+1)}{2}$. However, by the Ring-LWE assumption, the coefficients of $a \cdot s + b$ follow a scaled Irwin-Hall distribution, and it is assumed that the coefficients of $I < K = O(\sqrt{h})$ [LLK+20]. As the modular reduction function is approximated in the domain $\cup_{i=-(K-1)}^{K-1}[i-\epsilon, i+\epsilon]$, if a coefficient of I has a value greater than or equal to K, the modular reduction returns a useless value, and thus, failed.

Even though $O(\sqrt{h})$ is a reasonable upper bound for a single bootstrapping, it is not enough when the number of slots is large, and there are many bootstrappings. Let p be the probability of modular reduction failure, $\Pr(|I_i| \geq K)$. If there are n slots in the ciphertext, there are $2n$ coefficients to perform the modular reduction. Hence, the failure probability of single bootstrapping is $1-(1-p)^{2n} \approx 2n \cdot p$. Similarly, when there are N_b bootstrappings in the evaluation of the whole network, the failure probability of the whole network is $2N_b \cdot n \cdot p$. As there are many slots in our ciphertext, and thousands of bootstrapping are performed, the failure probability is very high when using previous approximate polynomials.

In Table 3, we show several bounds of the input message and its failure probability. A larger bound means that a higher degree of the approximate polynomial is required; hence, more computation is required. Using the new bound for approximation in Table 3, we can offer a trade-off between the evaluation time and failure probability of the whole network. Following [BMTPH20, LLK+20], the approximated modular reduction in the CKKS bootstrapping so far has failure probability $\approx 2^{-23}$, but it is not sufficiently small since we have to perform pretty many bootstrapping procedures for the ResNet-20. Thus, the bootstrapping failure probability is set to be less than 2^{-40} in our implementation. The Hamming weight of the secret key is set to be 64, and $(K, \epsilon) = (17, 2^{-10})$. The corresponding degree for the minimax polynomial for the cosine function is 45, and that of the inverse sine function is 1, which is obtained by the multi-interval Remez algorithm [LLL+20a]. The number of the double-angle formula ℓ is set to be 2.

5.6 Average Pooling and Fully Connected Layer

The size of the tensor after all convolutional layers are performed is $8 \times 8 \times 64$. We perform the average pooling to each channel to obtain a vector of length 64 and the fully connectned layer to obtain a vector of length 10. The form of the output for the average pooling is an array of the ciphertexts with length 64, and each element of ciphertext array has corresponding data in the first slot. Since all data are seperated in other ciphertexts, no rotation operation is needed when we perform the fully connected layer. Algorithm 7 shows the detailed procedures for average pooling and the fully connected layer.

5.7 Softmax

We use the softmax method proposed in Section 3.3. The bound parameters B and R are set as 64 and 10^4, respectively, and the Gumbel softmax parameter λ is set to be 4. The approximation parameter in Goldschmidt's division algorithm is set to be 16. Although parameter B greater than 40 is sufficient, as we discussed in Section 3.3, we use 64 since a power-of-two B is better for implementation. T is the number of classification types, which is 10 in the CIFAR-10 dataset. Algorithm 8 shows the detailed procedures for the softmax function.

Since the softmax function consumes many depths, several bootstrapping operations are used in the middle of the process. The bootstrapping is performed for each ciphertext just before the beginning of the softmax function, just before the inverse function, and after 8 iterations of Goldschmidt division process.

Algorithm 7: AvgPoolFullCon(Tensorct, W)

```
Input  : An encrypted tensor $\mathsf{Tensorct} = (\{\mathsf{ct}_k\}_{k=0,\cdots,t-1}, \ell, \mathsf{slotstr}, t)$, weight
         parameters for fully connected layer $W \in \mathbb{R}^{T \times t}$
Output : An array of ciphertexts $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
1  for $k = 0$ to $t-1$ do
2      $\bar{\mathsf{ct}}_k \leftarrow \mathsf{ct}_k$
3      for $i = 0$ to $\log \ell - 1$ do
4          $\mathsf{tmpct} \leftarrow \mathtt{rot}(\bar{\mathsf{ct}}_k, \mathsf{slotstr} \cdot 2^i)$
5          $\bar{\mathsf{ct}}_k \leftarrow \bar{\mathsf{ct}}_k \oplus \mathsf{tmpct}$
6      end
7      for $j = 0$ to $\log \ell - 1$ do
8          $\mathsf{tmpct} \leftarrow \mathtt{rot}(\bar{\mathsf{ct}}_k, \mathsf{slotstr} \cdot \ell \cdot 2^i)$
9          $\bar{\mathsf{ct}}_k \leftarrow \bar{\mathsf{ct}}_k \oplus \mathsf{tmpct}$
10     end
11 end
12 for $u = 0$ to $T-1$ do
13     $\mathsf{ct}'_u \leftarrow \mathbf{0}$
14     for $k = 0$ to $t-1$ do
15         $\mathsf{ct}'_u \leftarrow \mathsf{ct}'_u \oplus (W[u,k] \odot \bar{\mathsf{ct}}_k)$
16     end
17 end
18 return $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
```

6 Simulation Result

6.1 Simulation Setting and Model Parameters

We simulate the proposed model by the SEAL library [Mic21] released by Microsoft equipped with our implementation of RNS-CKKS bootstrapping. Our simulation environment is a dual Intel Xeon Platinum 8280 CPU (112 cores) with 512GB memory. We allocate one thread per one channel of each layer by using the OpenMP library to improve the execution speed of the ResNet-20. The required memory for this simulation is 172GB.

The model parameters are prepared by the following training method. We use 32×32 RGB images, subtract the mean of the pixels in the training dataset, and adopt a data argumentation method such as shifting and mirroring horizontally for training. We adopt the He initialization [HZRS15] as the weight initialization and no dropout. We train the model with 32×32 mini-batches and cross-entropy loss function. The learning rate starts with a 0.001 learning rate divided by 10 after 80 epochs and 100 after 120 epochs during training. The classification accuracy with the trained model parameters is 91.89%, which is tested with 10,000 images.

6.2 Performance

Table 4 shows the agreement ratio between the classification results of the implemented privacy-preserving ResNet-20 and that of the original ResNet-20, which shows almost the same results. We test the inference on 383 encrypted images, and the 95% confidence interval is suggested for each result. The classification accuracy of the ResNet-20 for the encrypted data is 92.43%$\pm$ 2.65%, while that of the original ResNet-20 for the corresponding plaintext image is 92.95%$\pm$ 2.56%. We also measure the agreement ratio, which is the percentage of the case when the output of the classification in the proposed PPML model is the same as the that in the original ResNet-20 model. Our agreement ratio is 98.43%$\pm$

```
Algorithm 8: Softmax(Tensorct, $B, R, \lambda$)
   Input  : An array of ciphertext $\{\mathsf{ct}_k\}_{k=0,\cdots,T-1}$, bound parameter $B, R$ ($B$ is a
            power-of-two integer), power-of-two Gumbel parameter $\lambda$, Goldschmidt
            approximation parameter $d$
   Output : An encrypted one-hot vector $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
 1 $p_{\exp} \leftarrow$ GenApproxPoly($e^x, [-1, 1]$)
 2 $\tilde{\mathsf{ct}} \leftarrow \mathbf{0}$
 3 for $k = 0$ to $T - 1$ do
 4     $\mathsf{ct}'_k \leftarrow 1/B \odot \mathsf{ct}_k$
 5     $\mathsf{ct}'_k \leftarrow$ polyeval($\mathsf{ct}'_k, p_{\exp}$)
 6     for $i = 0$ to $\log B - \log \lambda$ do
 7         $\mathsf{ct}'_k \leftarrow \mathsf{ct}'_k \otimes \mathsf{ct}'_k$
 8     end
 9     $\tilde{\mathsf{ct}} \leftarrow \tilde{\mathsf{ct}} \oplus \mathsf{ct}'_k$
10 end
11 $\tilde{\mathsf{ct}} \leftarrow 2 \ominus 1/R \odot \tilde{\mathsf{ct}}$
12 $\mathsf{tmpct} \leftarrow \tilde{\mathsf{ct}} \ominus 1$
13 for $j = 0$ to $d - 1$ do
14     $\mathsf{tmpct} \leftarrow \mathsf{tmpct} \otimes \mathsf{tmpct}$
15     $\tilde{\mathsf{ct}} \leftarrow \tilde{\mathsf{ct}} \otimes (1 \oplus \mathsf{tmpct})$
16 end
17 for $k = 0$ to $T - 1$ do
18     $\mathsf{ct}'_k \leftarrow \mathsf{ct}_k \otimes \tilde{\mathsf{ct}}$
19 end
20 return $\{\mathsf{ct}'_k\}_{k=0,\cdots,T-1}$
```

1.25%, which is a sufficiently high agreement result. Thus, we verify that the ResNet-20 can be successfully carried out using the RNS-CKKS scheme with sufficient accuracy for classification and the proper bootstrapping operation.

Table 5 shows the running time for the whole ResNet-20 and the portion for each component in the model. Note that we include the running time of the bootstrapping operation in BN or ReLU when the bootstrapping operation is performed in the middle of each operation. In other words, the regular bootstrapping for each layer is counted for the running time of the bootstrapping. The proposed model takes about 3 hours to infer one image, and the most time-consuming components are the convolution, the ReLU, and the bootstrapping.

Table 6 shows the running time of ResNet-20 when the bootstrapping operation is performed after convolution operation and after ReLU function, respectively. The running time of the case when the bootstrapping is performed after convolution operation is reduced by 27.8% compared to the case when the bootstrapping is performed after ReLU function. This supports the analysis of the bootstrapping position in Section 4.

Table 4: Classification accuracy of the ResNet-20 for plaintext and ciphertext and agreement ratio

Model	ResNet-20[1]	ResNet-20[2]	PPML ResNet-20	Agreement
Accuracy	91.89%	92.95% ± 2.56%	92.43% ± 2.65%	98.43%± 1.25%

[1] Classification accuracy verified with 10,000 images.

[2] Classification accuracy verified with 383 images which are used to test ResNet-20 on encrypted images.

Table 5: The running time of the ResNet-20 and the percentage of time spent in each component relative to total time

Layer	Conv	BN	ReLU	Boot	AP + FC	Softmax	Total time (s)
Time ratio	17.44%	13.55%	34.61%	31.55%	0.04%	2.81%	10,602

Table 6: Comparison of the running time of ResNet-20 for two positions of the bootstrapping

Bootstrapping position	After conv	After ReLU
Total Time (s)	10,602	14,694

6.3 Discussion

Running Time The running time for the proposed model, which is about 3 hours, is somewhat large for practical use. This work firstly shows the possibility of applying the FHE to standard deep learning models with high accuracy, but it has to be optimized and improved in various ways to reduce the running time. Therefore, the essential future work is the advanced implementation of the ResNets with the RNS-CKKS scheme with various accelerators realized using GPU, FPGA, or ASIC. Since research on implementing the state-of-the-art FHE scheme is advancing rapidly, the ResNet-20 over the encrypted data will be made more practically soon. Also, we implement the PPML model for only one image, and the running time per image can be much improved if we properly use the packing method of the RNS-CKKS scheme. We leave this optimization for many images as future work.

Security Level The security level of the proposed model is 111.6-bit security, which is a marginal security level that can be considered secure. Since the standard security level in most applications is 128 bit, someone can regard this security level as insecure, and we may want to raise the security level. However, this 111.6-bit security is not a hard limit of our implementation; we can easily raise the security level by changing the parameters of the RNS-CKKS scheme. This just makes the trade-off between the security and the running time, and thus we can reach the higher security level at the cost of longer running time.

Classification Accuracy Even if ML models are trained with the same hyper-parameters, the ML models have different performances because weights are initialized to random values for each training. Thus, the ML model performance, such as accuracy, is shown as the average values obtained by training several times. However, since we focus on implementing the ResNet-20 for homomorphically encrypted data, we train this model only once, not many times. Nevertheless, we have shown that the encrypted ResNet-20 operation is possible with almost the same accuracy as the original ResNet-20 paper [HZRS16]. Furthermore, since it is implemented in the FHE with bootstrapping, it can be expected that the same result will be obtained for a deeper network than the Resnet-20.

7 Conclusion

For the first time, we applied the RNS-CKKS scheme, one of the state-of-the-art FHE schemes, to the standard deep neural network ResNet-20 to implement the PPML. Since the precise approximation of the ReLU function, the bootstrapping, and the softmax function had not been applied to the PPML models until now, we applied these techniques with fine-tuned various parameters. Then, we showed that the implemented ResNet-20 with the RNS-CKKS scheme achieves almost the same result as the original ResNet-20 and reaches the highest classification accuracy among the PPML models with the word-wise

FHE scheme introduced so far. This work firstly suggested that the word-wise FHE with the most advanced techniques can be applied to the state-of-the-art machine learning model without re-training it.

References

[BCCW19] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 45–56, 2019.

[BCD$^+$20] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: A mixed-protocol machine learning framework for private inference. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020.

[BCL$^+$20] Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Sim Jun Jie, Benjamin Hong Meng Tan, Xiao Nan, Aung Mi Mi Khin, and Vijay Chandrasekhar. Towards the Alexnet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs. *IEEE Transactions on Emerging Topics in Computing*, 2020.

[BGP$^+$20] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan. Optimized homomorphic encryption solution for secure genome-wide association studies. *BMC Medical Genomics*, 13(7):1–13, 2020.

[BLCW19] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 3–13, 2019.

[BMTPH20] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. Cryptology ePrint Archive, Report 2020/1203, 2020. https://eprint.iacr.org/2020/1203, accepted to *EUROCRYPT 2021*.

[CBL$^+$18] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. arXiv preprint, abs/1811.09953, 2018. http://arxiv.org/abs/1811.09953.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2019*, pages 34–54, 2019.

[CHHS19] Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son. A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret lwe. *IEEE Access*, 7:89497–89506, 2019.

[CHK$^+$18a] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2018*, pages 360–384, 2018.

[CHK+18b] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *Proceedings of International Conference on Selected Areas in Cryptography (SAC)*, pages 347–368, Calgary, Canada, 2018.

[CKK+19] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 415–445. Springer, 2019.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017*, pages 409–437, 2017.

[FV20] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2020. https://eprint.iacr.org/2012/144.

[GBDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 201–210, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing (STOC)*, pages 169–178, 2009.

[Gol64] Robert E Goldschmidt. *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.

[HK20] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Proceedings of Cryptographers' Track at the RSA Conference*, pages 364–390, 2020.

[HTG17] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JKA+21] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. Cryptology ePrint Archive, Report 2021/508, 2021. https://eprint.iacr.org/2021/508.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium*, pages 1651–1669, 2018.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KPP20] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. Cryptology ePrint Archive, Report 2020/1118, 2020. https://eprint.iacr.org/2020/1118.

[LJ19] Qian Lou and Lei Jiang. SHE: A fast and accurate deep neural network for encrypted data. *Advances in Neural Information Processing systems (NeurIPS)*, 2019.

[LLK$^+$20] Yongwoo Lee, Joonwoo Lee, Young-Sik Kim, HyungChul Kang, and Jong-Seon No. High-precision and low-complexity approximate homomorphic encryption by error variance minimization. Cryptology ePrint Archive, Report 2020/1549, 2020. https://eprint.iacr.org/2020/1549.

[LLL$^+$20a] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approxiamtion and inverse sine function. Cryptology ePrint Archive, Report 2020/552, 2020. https://eprint.iacr.org/2020/552, accepted to *EUROCRYPT 2021*.

[LLL$^+$20b] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. Optimal minimax polynomial approximation of modular reduction for bootstrapping of approximate homomorphic encryption. Cryptology ePrint Archive, Report 2020/552, Second Version, 2020. https://eprint.iacr.org/2020/552/20200803:084202.

[LLL$^+$21] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. Precise approximation of convolutional neural networks for homomorphically encrypted data. arXiv preprint, abs/2105.10879, 2021. http://arxiv.org/abs/2105.10879.

[LLNK20] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. Minimax approximation of sign function by composite polynomial for homomorphic comparison. Cryptology ePrint Archive, Report 2020/834, 2020. https://eprint.iacr.org/2020/834.

[Mic21] Microsoft. Microsoft SEAL. https://github.com/microsoft/SEAL, 2021.

[RCK$^+$20] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T Lee, Hsien-Hsin S Lee, Gu-Yeon Wei, and David Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 26–39. IEEE, 2020.

[vEPIL19] Tim van Elsloo, Giorgio Patrini, and Hamish Ivey-Law. SEALion: A framework for neural network inference on encrypted data. arXiv preprint, abs/1904.12840, 2019. http://arxiv.org/abs/1904.12840.