An Efficient CKKS-FHEW/TFHE Hybrid Encrypted Inference Framework

Tzu-Li Liu[1,2], Yu-Te Ku[1,2], Ming-Chien Ho[1,2], Feng-Hao Liu[3], Ming-Ching Chang[4,1], Chih-Fan Hsu[1], Wei-Chao Chen[1], and Shih-Hao Hung[2,5]

[1] Inventec Corporation, Taipei City, 111059, Taiwan
{hsu.chih-fan,chen.wei-chao}@inventec.com
[2] National Taiwan University, Taipei City, 10617, Taiwan
{r10922092,d08946006,r11944009}@ntu.edu.tw
hungsh@csie.ntu.edu.tw
[3] Washington State University, Pullman, WA, 99164, USA
feng-hao.liu@wsu.edu
[4] State University of New York, University at Albany, Albany, NY 12222, USA
mchang2@albany.edu
[5] Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi

Abstract. Machine Learning as a Service (MLaaS) is a robust platform that offers various emerging applications. Despite great convenience, user privacy has become a paramount concern, as user data may be shared or stored in outsourced environments. To address this, fully homomorphic encryption (FHE) presents a viable solution, yet the practical realization of this theoretical approach has remained a significant challenge, requiring specific optimization techniques tailored to different applications. We aim to investigate the opportunity to apply the CKKS-FHEW/TFHE hybrid approach to NNs, which inherit the advantages of both approaches. This idea has been implemented in several conventional ML approaches (PEGASUS system presented in IEEE S&P 2021), such as decision tree evaluation and K-means clustering, and demonstrated notable efficiency in specific applications. However, its effectiveness for NNs remains unknown. In this paper, we show that directly applying the PEGASUS system on encrypted NN inference would result in a significant accuracy drop, approximately 10% compared to plaintext inference. After a careful analysis, we propose a novel LUT-aware fine-tuning method to slightly adjust the NN weights and the functional bootstrapping for the ReLU function to mitigate the error accumulation throughout the NN computation. We show that by appropriately fine-tuning the model, we can largely reduce the accuracy drop, from 7.5% to 15% compared to the baseline implementation without fine-tuning, while maintaining comparable efficiency with extensive experiments.

Keywords: Homomorphic encryption · neural network · functional bootstrapping · privacy-preserving machine learning.

1 Introduction

Machine Learning as a Service (MLaaS) [30] are cloud-based platforms providing machine learning (ML) models to run services that are available anytime, anywhere. With the advance of deep learning (DL), neural networks (NN) has been widely used in various applications, *e.g.*, computer vision [20], speech processing [31], and content generation [16]. However, a major concern of MLaaS is on the difficulties regarding the security and privacy of user data. The MLaaS owner could have access to user's private data without permission. Users may hesitate to upload private data to the MLaaS platform due to confidentiality [22]. As a result, exploring a secure MLaaS is an urgent need. Ideally, a secure MLaaS in practice should provide high inference accuracy, low computational time, and minimal memory usage, meanwhile protecting the confidentiality of user data. Secure multiparty computation (SMC) [32,3] is a common solution toward secure MLaaS. SMC is typically associated with higher computational costs, but provides robust confidentiality protection.

Fully Homomorphic Encryption (FHE) [26] is a promising alternative that achieves essentially non-interactive SMC, allowing basically minimal participation (communication/computation) from the user during the protocol execution. Particularly, in the FHE pipeline for secure MLaaS, the user first encrypts the data into ciphertext which is then transmitted to a cloud server. The server then performs ML inference via homomorphic computation on the ciphertext. The encrypted result is then sent back to the user, who then decrypts to retrieve the desired output. The *FHE-based NN encrypted inference* is potentially attractive in terms of security and confidentiality, however with a main limitation of the huge computational burden induced by the FHE calculations.

The FHE-based NN encrypted inference includes three common types of encrypted computations that are executed sequentially in a NN node, namely *linear operations*, *activation functions*, and *bootstrapping*.

Linear operations are the inner product calculations between the encrypted input data array and the plaintext NN weights. There are three popular FHE schemes used in the encrypted inference: (1) CKKS [5], (2) FHEW [9]/TFHE [7], and (3) CKKS-FHEW/TFHE Hybrid [25]. The CKKS scheme can easily carryout linear operations due to its ability to support floating-point arithmetic, while FHEW/TFHE only supports lightweight bit-wise or integer operations. Note that the integer operations of FHEW/TFHE has lower computational burden over the CKKS floating point calculations.

Regarding activation functions, CKKS only supports element-wise polynomial operations, such as multiplication, addition, and rotation, and does not directly support non-polynomial operators. This is unsuitable for encrypted NN inference, as CKKS does not support ReLU, which is the commonly used activation function in NN models. A possible solution around is to approximate the activation function using polynomial approximations. Although this alternative is simple to implement, it produces deviated results that might affect NN inference accuracy. It also comes with additional computational overhead.

Bootstrapping [13,14] is originally designed to reduce the accumulated computation errors of the ciphertext in FHE, which is a crucial operation to ensure that FHE calculations can be carried out with sufficient computational depth. However, performing repeated boostrapping operations with common CKKS implementations [4] can introduce numerical errors and information loss. In contrast, FHEW/TFHE bootstrapping has no numerical errors. Besides, an advanced bootstrapping operation, the functional bootstrapping, in FHEW/TFHE scheme can achieve performing non-polynomial operations, such as ReLU, while refreshing the ciphertext simultaneously. Generally, an FHEW/TFHE functional bootstrapping supports more than one bootstrapping iteration. According to the implementation, the FHEW/TFHE functional bootstrapping can be categorized into the regular version and the large-precision version. The regular version represents the original design, while the large version was introduced by [23]. The regular version contains only one bootstrapping iteration and is efficient compared to the large-precision version [23], which is friendly for NN inference [19,2]. However, it suffers from limited numerical precision. According to the literature [21,2], regular functional bootstrapping has only been empirically evaluated on the simple MNIST dataset and has not been generalized to work with more complex datasets. In practical applications, the message domain of FHE-based NN inference architecture usually exceeds the input domain typically used in regular function bootstrapping. The large-precision version is desired for real-world applications but its computing efficiency should be further improved.

The CKKS-FHEW/TFHE Hybrid scheme [25] inherits the strengths of the linear operations in CKKS and the benefits of the FHEW/TFHE functional bootstrapping. For FHE-based NN encrypted inference, the CKKS scheme can be used for linear operations, while the FHEW/TFHE functional bootstrapping scheme can be used for non-polynomial activation functions and bootstrapping without adding large computation overhead for transferring between the CKKS and FHEW/TFHE ciphertexts. Notably, the PEGASUS system [25] efficiently bridge these two schemes into a hybrid implementation. PEGASUS first *scales down* the CKKS ciphertext/plaintext, making it compatible with the input to the FHEW/TFHE regular bootstrapping. Then, only one regular functional bootstrapping is required to complete the pipeline. This would be substantially more efficient than the approach of large-precision bootstrapping [23], which would require many more core bootstrapping calls. PEGASUS has demonstrated notable efficiency in specific applications such as decision tree evaluation and K-means clustering, yet the work did not conduct experiments in the setting of NN inference [25]. It remains an interesting open question to determine whether the CKKS-FHEW/TFHE hybrid framework can be effectively extended to cover NNs. In this paper, we focus on the potential of the CKKS-FHEW/TFHE hybrid approach for FHE-based NN encrypted inference.

1.1 Our Contribution

In this work, we aim to apply the CKKS-FHEW/TFHE hybrid approach to FHE-based NN encrypted inference. However, certain numerical adjustments on

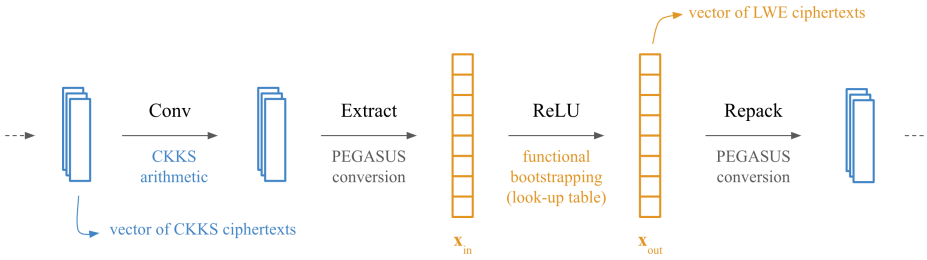

Fig. 1. The four-step process in a layer of FHE NN inference.

the model weights and the activation function are required to maintain the inference accuracy, thereby rendering the entire approach practical and usable. Below, we provide a more comprehensive explanation of our specific contributions.

First, we implement an FHE inference framework for running neural networks named CKKS-FHEW/TFHE Hybrid Encrypted Inference Framework, on top of the PEGASUS, and then notice a significant decrease in accuracy. With a careful analysis, we identify that the accuracy drop comes from the limited precision when applying regular functional bootstrapping on large-domain ciphertexts.

Next, we address this issue by proposing a novel LUT-aware model fine-tuning framework on the machine learning side. The adjustment is specifically tailored for the above hybrid approach, effectively avoiding the accuracy drop due to the precision issue while maintaining the efficiency advantage of the approach.

Finally, we conduct a comprehensive experiment to validate our LUT-aware model fine-tuning framework on a color dataset, CIFAR-10, which is more complex than the MNIST dataset. The experimental results show that our framework increases accuracy from 7.5% to 15% compared to the NN model without applying our method. Moreover, the fine-tuned NN models achieve accuracy comparable to the original ones.

We compared our approach to two state-of-the-art implementations, Shift-accumulation-based LHE- enabled Deep Neural Network (SHE) [24] and Privacy-Preserving Machine Learning (PPML) [21]. Our fine-tuned model shows competitive accuracy while maintaining high efficiency in time 2,558 s (using 8 threads) compared to PPML which takes over 3,581 s (using 8 threads) and SHE which takes more than 347,555 s (using 8 threads).

1.2 Technical Overview

Our proposed CKKS-FHEW/TFHE Hybrid Encrypted Inference framework includes four-step for each FHE-based NN layer computation, as illustrated in Fig. 1.

1. CKKS computation scheme is first employed to perform the convolution and linear operations on the CKKS ciphertexts.
2. The CKKS ciphertext is then converted into multiple large-domain LWE ciphertexts based on the PEGASUS extraction method.

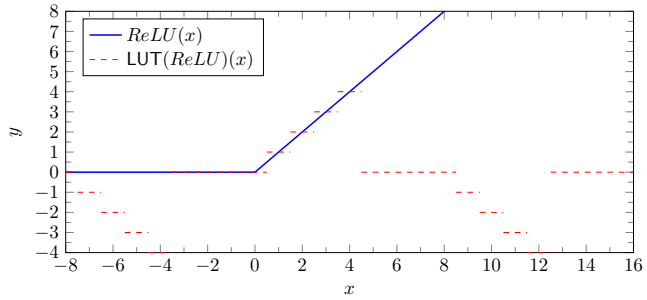

Fig. 2. The evaluation of the ReLU function using regular functional bootstrapping on large-domain ciphertexts. A step-wise numerical behavior can be observed.

3. Next, regular functional bootstrapping is applied to the converted large-domain Learning With Errors (LWE) [29] ciphertexts to perform the activation operation while reducing the numerical errors.
4. Finally, the large-domain LWE ciphertexts are repacked back into the CKKS ciphertexts for the subsequent layers of operations.

We first naively applied the hybrid approach to several NN models and observed severe accuracy drops on all tested NNs. The drop is caused by the accumulated numerical errors throughout the encrypted NN inference. The numerical errors are generated by the scale-down process in the PEGASUS framework that limits the numerical precision when applying regular functional bootstrapping on the large-domain ciphertexts. More specifically, we let the domain of the message be $[-B, B)$ and n be the size of the input domain for the LWE functional bootstrapping. The LWE ciphertext of the message is scaled down to meet the requirement of the functional bootstrapping taking a look-up table of size n which is identical to the size of the input domain. In this case, the LWE functional bootstrapping can be treated as a look-up-table (LUT) operation with n entries. We note that the LWE functional bootstrapping exhibits a cyclic behavior out of the range $[-B, B)$ because of the modulus operation involved in the FHE computation. Fig. 2 illustrates an example of the ReLU function evaluated by the LWE functional bootstrapping with $n = 8$ in the range $[-4, 4)$. Observe that the ReLU output has step-wise behavior according to the LUT operation.

If the input value range of the LWE functional bootstrapping does not align with the predefined input domain of the LWE functional bootstrapping, either the input domain of the functional bootstrapping can not fulfill the range of the input values or a few LUT entries are utilized. The left-hand-side figure in Fig. 3 illustrates the example that the predefined input domain of the LWE functional bootstrapping $[-4, 4)$ is larger than the input value range $[-0.3, 1.7)$. Only three LUT entries are used to map the input value x to the output y. The range mismatching generates a large numerical error when applying functional bootstrapping and then eventually affects the NN model accuracy.

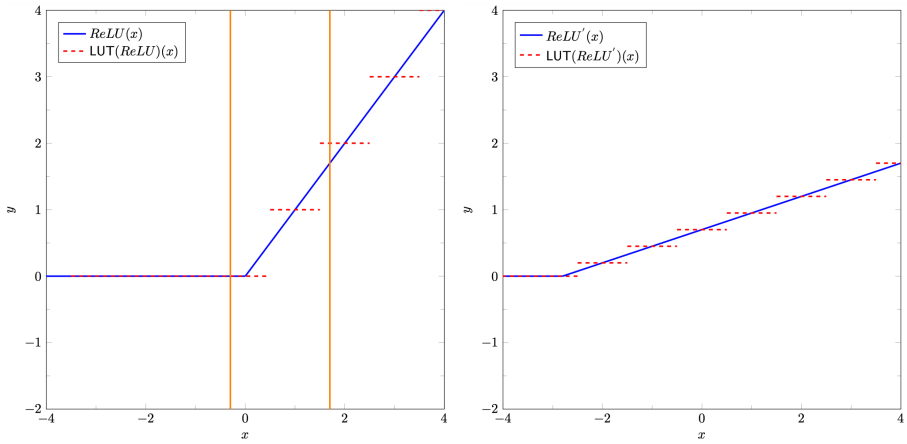

Fig. 3. An example of range alignment, where the input range $[-0.3, 1.7)$ (left, orange) is mapped to $[-4, 4)$ (right) to improve precision. It is evident that the aligned function $ReLU'$ utilizes a greater number of look-up table entries. Noted that the blue lines indicate the target functions and the red lines are the look-up tables according to the functional bootstrapping behavior.

We design a LUT-aware model fine-tuning framework to align the range of the input value with the message domain of the ciphertext. The mismatch can be mitigated by aligning the input value with the LUT entry as the right-hand-side figure in Fig. 3. More specifically, we first estimate the input range by analyzing the observed feature maps generated during the inference of plain training data on the trained model. This estimation can be used to define the range of input values. Next, we align the range with the message domain. This alignment ensures that the input values are precisely mapped to the LUT entry. The numerical optimization process greatly utilizes the data and enhances the final accuracy of the FHE NN inference. Note that the alignment process is a preprocess of our model fine-tuning framework. The alignment includes linearly transforming pre-trained NN model weights and updating the activation functions of the model. According to our design, we largely improve the accuracy of the model while maintaining the inference efficiency thanks to the better numerical alignment to the regular functional bootstrapping.

We conducted experimental evaluations by solving the image classification problem with convolutional neural networks (CNNs) on the CIFAR-10 dataset. Given the substantial time cost associated with FHE NN inference, we made the decision to train smaller-scale plain models. To achieve this, we employed knowledge distillation from a ResNet-20 model with an impressive accuracy of 92% to distill knowledge to our small models. Our experimental results provide strong validation of the effectiveness of our approach. Initially, our plain models achieved an impressive inference accuracy of 80%. However, when we performed FHE NN inference without LUT-aware fine-tuning, we observed a significant decline in accuracy ranging from 7.5% to 17.5%. Fortunately, by incorporating

LUT-aware fine-tuning into our encrypted inference process, we were able to overcome this challenge. The application of LUT-aware fine-tuning consistently yielded accuracy levels that were comparable to the plain models. This outcome underscores the robustness and reliability of our approach in maintaining high accuracy during encrypted inference.

2 Preliminaries

Our framework is built on both CKKS [6] and FHEW/TFHE [10,8] schemes. Next, we define the settings and notations used in our work.

CKKS Encryption. CKKS supports arithmetic operations on floating-point vectors. During the encryption process, it first encodes a message m, which is a vector of float numbers, into a plaintext polynomial. Denote the encoding operation as $\mathsf{Ecd}(\cdot)$, which encrypts the plaintext polynomial to a ciphertext. Denote the encrypting operation as $\mathsf{Enc}(\cdot)$. We use the notation of $\mathsf{CKKS}(\mathbf{m})$ to denote $\mathsf{Enc}(\mathsf{Ecd}(\mathbf{m}))$ with $\mathbf{m} \in \mathbb{R}^n$.

CKKS supports the evaluation of CKKS $\mathsf{CKKS.eval}(f, \cdot)$. The inputs are a plaintext function f and the arguments of f placed after f, such as message $\mathbf{m}$ or ciphertext $\mathsf{ctx} \in \mathsf{CKKS}(\mathbf{m})$. The operation is conducting the function with these arguments in CKKS and returns $\mathsf{ctx} \in \mathsf{CKKS}(f(\cdot))$. For example, we let $\mathsf{ctx} \in \mathsf{CKKS}(\mathbf{m})$, $\mathsf{CKKS.eval}(+, \mathsf{ctx}, \mathbf{v})$ returns $\mathsf{ctx}' \in \mathsf{CKKS}(\mathbf{m} + \mathbf{v})$.

LWE and Regular Functional Bootstrapping. LWE encrypts a plaintext, which is the encoding of a scalar $m \in \mathbb{R}$, to a ciphertext. Let $\mathsf{LWE}(\lceil m \rfloor)$ denote the LWE encryption of $m \in \mathbb{R}$, and let $\lceil \cdot \rfloor$ denote the rounding and encoding function. LWE possesses two commonly used implementations, FHEW and TFHE. Compared to FHEW, TFHE is faster in evaluating bootstrapping and requires a smaller bootstrapping key. Both implementations support bit-wise operations on LWE ciphertexts, which represent encrypted binary data $m \in \{0, 1\}$. FHEW and TFHE enable support for the NAND operation on LWE ciphertext, allowing them to support any circuit based on the completeness of the NAND circuit. Moreover, they also provide support for regular functional bootstrapping.

Regular functional bootstrapping is a type of bootstrapping used to reduce error accumulation in HE operations. It involves constructing a lookup table for a specific function, enabling efficient computation of results. While it requires fewer bootstrapping iterations and less computation, it is effective only for supporting small message domain. If the requirement is to support large domain messages, it results in significant computational overhead.

PEGASUS CKKS-FHEW Conversion. CKKS demonstrates remarkable efficiency in vector operations, while FHEW exhibits exceptional efficiency in non-polynomial functions such as ReLU, using functional bootstrapping operations. If we can switch between CKKS and FHEW efficiently, it is suitable for performing private and encrypted inference for MLaaS. Lu *et al.* [25] proposed PEGASUS, a framework that efficiently converts packed CKKS ciphertext and FHEW ciphertexts without decryption. The conversion involves these operations.

- **Extract** $\mathsf{pegasus.extract}(\mathsf{ctx})$: Given a ciphertext $\mathsf{ctx} \in \mathsf{CKKS}(\mathbf{m})$ with $\mathbf{m} \in R^n$, the operation involves transforming CKKS encryption into LWE encryption and return a set of $\{\mathsf{ctx}_i^{'}\} \in \mathsf{LWE}(\lceil m_i \rfloor)$ with $m_i \in \mathbf{m}$ and $0 \leq i < n$.
- **Repack** $\mathsf{pegasus.repack}(\{\mathsf{ctx}_i^{'}\}_{0 \leq i < n})$: Given a set $\{\mathsf{ctx}_i^{'}\}_{0 \leq i < n}$ with $\mathsf{ctx}_i^{'} \in \mathsf{LWE}(\lceil m_i \rfloor)$ with $m_i \in \mathbf{m}$, the operation involves repacking a set of LWE ciphertexts into a single CKKS ciphertext and return $\mathsf{ctx} \in \mathsf{CKKS}(\mathbf{m})$.

PEGASUS Large-domain Functional Bootstrapping. The regular functional bootstrapping method can only be applied to LWE ciphertexts with a message domain limited to the size of the look-up table, typically around 2^{10} [27]. However, when dealing with LWE ciphertexts converted from CKKS ciphertexts, a larger message domain is usually required. To address this specific issue, PEGASUS introduces a groundbreaking technique known as large-domain functional bootstrapping. This technique involves scaling down the input ciphertext to a smaller message domain during the conversion from CKKS to FHEW, enabling the extension of regular functional bootstrapping to support a larger message domain. However, this expansion comes at the cost of reduced precision. Additionally, Liu et al. [23] proposed a method that utilizes digit decomposition to divide a large message into multiple smaller chunks and performs functional bootstrapping on each chunk individually. This approach preserves the precision of the input message but requires multiple bootstrapping iterations. Therefore, considering computational efficiency, this approach is not adopted.

- **Look-up table evaluation** $\mathsf{pegasus.eval}(f, \mathsf{ctx})$: Given a plaintext f and a $\mathsf{ctx} \in \mathsf{LWE}(\lceil m \rfloor)$, the operation returns $\mathsf{ctx}' \in \mathsf{LWE}(\lceil f(m) \rfloor)$ with error.

PEGASUS evaluation. PEGASUS employs a fine-grained look-up table approximation for evaluating non-polynomial functions such as sigmoid and ReLU. Inputs of the evaluation is a plaintext function f and a ciphertext $\mathsf{ctx} \in \mathsf{CKKS}(\mathbf{m})$ and output is $\mathsf{ctx}^{'} \in \mathsf{CKKS}(f(\mathbf{m}))$ with error. The evaluation involves three steps. First, as functional bootstrapping is faster under the FHEW scheme than CKKS, PEGASUS extracts a CKKS ciphertext ctx to a set FHEW ciphertext $\{\mathsf{ctx}_i\} \in \mathsf{pegasus.extract}(\mathsf{ctx})$. Second, it evaluates the look-up table $\mathcal{T}$ according to f on each $\{\mathsf{ctx}_i\}$ and gets a set $\{\mathsf{ctx}_i^{'}\} \in \mathsf{pegasus.eval}(f, \mathsf{ctx}_i)$. Finally, it repacks $\{\mathsf{ctx}_i^{'}\}$ to $\mathsf{ctx}^{'} \in \mathsf{pegasus.repack}(\{\mathsf{ctx}_i^{'}\})$.

3 Methodology

We propose the LUT-aware model fine-tuning framework to mitigate the impact caused by the limited precision of regular functional bootstrapping. Recall our CKKS/FHEW-TFHE Hybrid Secure DNN implementation, we next describe the mathematical formulation of the four-step process in § 1.2:

- The **Convolution Layer** $\mathsf{CKKS.eval}(f_{\mathbf{CONV}}, \{\mathsf{ctx}_0, \mathsf{ctx}_1, ...\})$ performs the convolution operation with the input feature map that is encrypted in multiple CKKS ciphertexts: $f_{\mathrm{CONV}}(\mathbf{x}) = \mathbf{W}(\mathbf{x}) + \mathbf{b}$. The calculation is carried

Algorithm 1: Secure inference on layer i of LUT-aware fine-tuned model.

input : the encrypted feature maps $\mathbf{ctx}_{n_0} \in \{\mathsf{CKKS}(\mathbf{m}_i)\}_{0\le i<n_0}$,
the folded convolution operation $f_{\mathrm{CONV}}{}'_i$,
the aligned activation function $f_{\mathrm{ACT}}{}'_i$
output: $\mathbf{ctx}_{\mathsf{pack}} \in \{\mathsf{CKKS}(\mathbf{m}_i)\}_{0\le i<n_{\mathsf{pack}}}$

1 $\mathbf{ctx}_{\mathsf{conv}} = \mathsf{CKKS.eval}(f_{\mathrm{CONV}}{}', \mathbf{ctx}_{n_0})$; // $\mathbf{ctx}_{\mathsf{conv}} \in \{\mathsf{CKKS}(\mathbf{m}_i)\}_{0\le i<n_{\mathsf{conv}}}$
2 $\mathbf{ctx}_{\mathsf{ext}} = \mathsf{pegasus.extract}(\mathbf{ctx}_{\mathsf{conv}})$; // $\mathbf{ctx}_{\mathsf{ext}} \in \{\mathsf{LWE}(m_i)\}_{0\le i<n_{\mathsf{ext}}}$
3 $\mathbf{ctx}_{\mathsf{act}} = \mathsf{pegasus.eval}(f_{\mathrm{ACT}}{}', \mathbf{ctx}_{\mathsf{ext}})$; // $\mathbf{ctx}_{\mathsf{act}} \in \{\mathsf{LWE}(m_i)\}_{0\le i<n_{\mathsf{act}}}$
4 $\mathbf{ctx}_{\mathsf{pack}} = \mathsf{pegasus.repack}(\mathbf{ctx}_{\mathsf{act}})$; // $\mathbf{ctx}_{\mathsf{pack}} \in \{\mathsf{CKKS}(\mathbf{m}_i)\}_{0\le i<n_{\mathsf{pack}}}$
5 Output $\mathbf{ctx}_{\mathsf{pack}}$

out in the encrypted domain, and a list of CKKS ciphertexts $\{\mathsf{ctx}'_0, \mathsf{ctx}'_1, ...\}$ are produced as the output.

- The **Extraction** $\mathsf{pegasus.extract}(\{\mathsf{ctx}_0, \mathsf{ctx}_1, ...\})$**:** takes a list of CKKS ciphertexts as input and extracts each CKKS ciphertext to gather the resulting LWE ciphertexts. The output is these gathered LWE ciphertexts.
- The **Activation Layer** $\mathsf{pegasus.eval}(f_{\mathbf{ACT}}, \{\mathsf{ctx}_0^{\mathsf{LWE}}, \mathsf{ctx}_1^{\mathsf{LWE}}, ...\})$ evaluates the activation function f_{ACT} on each input LWE ciphertext. The output is a list of the resulting LWE ciphertexts.
- The **Repacking** $\mathsf{pegasus.repack}(\{\mathsf{ctx}_0^{\mathsf{LWE}}, \mathsf{ctx}_1^{\mathsf{LWE}}, ...\})$ step repacks the multiple LWE ciphertexts into several CKKS ciphertexts, where each CKKS ciphertext encrypts a specific number of values (message).

To overcome the mismatch issue introduced in § 1.2, we propose the precise alignment of the input range with the message domain. To achieve this alignment, we employ a fine-tuning approach that focuses on optimizing the convolution operations and the activation functions. Through fine-tuning, we can effectively align the input range with the desired message domain, ensuring optimal precision and accuracy in our computations.

3.1 LUT-aware Model Fine-tuning

Our Look-Up Table (LUT) aware model fine-tuning involves two main steps.

Step 1. Update the model weights $f_{\mathrm{CONV}}{}'(\mathbf{x}) = \mathcal{F}_{\mathrm{LM}}^{(a,b,B)}(f_{\mathrm{CONV}}(\mathbf{x}))$: We apply a linear map to the input to ensure that the input interval is aligned with the message domain. This linear map function can be incorporated into the convolution weights and bias, thereby avoiding any additional computational costs or memory usage. The incorporation details will be introduced in § 3.3. Function $\mathcal{F}_{\mathrm{LM}}^{(a,b,B)}$ linearly map the input $\mathbf{x} \in \mathbb{R}^n, x_i \in [a, b)$ to the range $[-B, B)$.

Step 2. Update the activation functions $f_{\mathrm{ACT}}{}' = f_{\mathrm{ACT}} \circ \mathcal{F}_{\mathrm{LM}}^{(a,b,B)^{-1}}$: We update all the activation functions with inverse linear mapping functions such that $f_{\mathrm{ACT}}{}'(\mathcal{F}_{\mathrm{LM}}^{(a,b,B)}(x)) = f_{\mathrm{ACT}}(x)$. The modification ensures that the new activation functions produce identical results when applied to the linearly mapped

inputs, which greatly reduces the error caused by the limited precision of the regular functional bootstrapping.

Noted that the fine-tunings are performed on the plain neural network models, before we turn them into the encrypted inference version. Algorithm 1 describes the encrypted inference calculation steps.

3.2 Estimate the Numerical Ranges of the Input for Activation Functions

Recall that aligning the input value range with the message domain $[-B, B)$ enhances the utilization of LUT entries. The range of the activation function inputs, denoted as $[a_i, b_i)$ for each layer i, plays a crucial role in aligning them with the desired interval $[-B, B)$. These parameters determine the valid range of inputs for the activation functions. If the input values fall outside the range $[a_i, b_i)$, applying the activation function with an inverse linear mapping will result in a significant mismatch or error in the output. On the other hand, if the range $[a_i, b_i)$ is too large, only a small portion of the precision within that interval will be effectively utilized. Hence, determining the appropriate range of activation function inputs is essential to ensure accurate and efficient computation while maximizing the utility of precision.

Since the inputs to the activation functions are encrypted, also the ranges of activation inputs are not predetermined and vary based on the input values fed into the neural network. We assume that the training/testing distribution are similar. Therefore, to estimate these input ranges, we conduct inference on the trained model using plain (non-encrypted) data from the training set. During this process, we observe and record the minimum and maximum input values, denoted as a_i and b_i respectively, that occur at the activation function of each layer i. By performing inference on the plain trained model with training data, we are able to capture the dynamic input ranges that are possibly encountered during actual usage. These minimum and maximum values provide insights into the range of values the activation inputs can take, allowing us to align the input ranges with proper parameters a_i and b_i.

3.3 Incorporate the Linear Mapping into Model Weights

In Step 1, inspired by batch normalization folding [17], we also incorporated the linear mapping function into the convolution layers to migrate the additional multiplication depth and computational cost caused by linear mapping function. Specifically, we fold the linear map $\mathcal{F}_{\mathrm{LM}}^{(a,b,B)}$ into convolution layer f_{CONV}.

In addition, due to the independence of the computation of each output channel in the convolution operation, it is viable to estimate the ranges for all output channels and perform activation function evaluation using the corresponding look-up table. Fig. 4 illustrates that the ranges $[a_i, b_i), 0 \leq i < 8$, vary

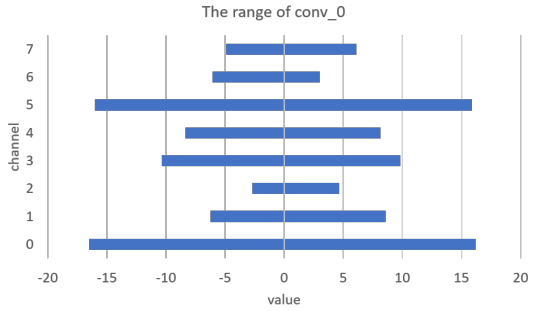

Fig. 4. Visualization of the output ranges $[a_i, b_i)$ of each convolution channel from layer 0 of the $D5L1$ model.

Table 1. Crypto parameters used in PEGASUS in our experiments. We follow the notations of [25]. RLWE is CKKS encryption. $\widetilde{\mathsf{RLWE}}$ and RGSW are encryption schemes used in TFHE/FHEW conversion and look-up table evaluation. ns denote the ring dimensions, qs and Q denote the modulus and σs denote the noise parameters. Descriptions of these symbols can be found in the Preliminaries of the PEGASUS paper [25].

Encryption	Parameters
$\widetilde{\mathsf{RLWE}}^{\underline{n},q_0}(\cdot)$	$\underline{n} = 2^{11}, q_0 \approx 2^{45}, \sigma_{ks} = 2^{10}, B_{ks} = 2^7, d_{ks} = 7$
$\mathsf{RGSW}^{n,q'}(\cdot)$	$n = 2^{12}, q' \approx 2^{60} \times q_0, q_0 \approx 2^{45}, \sigma_{lut} = 2^{10}$
$\mathsf{RLWE}^{\overline{n},\overline{Q}}(\cdot)$	$\overline{n} = 2^{16}, q_i \approx 2^{45}, \sigma_{ckks} = 3.19, \log \overline{Q} = \mathbf{689}$

across different output channels. Suppose all channels are appropriately linearly mapped to the functional bootstrapping input domain with their own ranges. In that case, the look-up table evaluation can be performed with the same level of resolution for all channels.

4 Experiments

We conducted extensive experiments to evaluate the effectiveness of our fine-tuning scheme using real neural network models on the CIFAR-10 dataset. Through the implementation and testing of our approach, we assess the encrypted ML model performance and compare it with two state-of-the-art implementations, namely SHE [24] and PPML [21]. These comparisons provided valuable insights into the strengths and advantages of our method.

In our experiments, we utilized a server equipped with an Ubuntu 20.04.6 LTS operating system. The server featured 8 cores of an AMD EPYC 7763 64-Core Processor, providing 16 threads and 126GB memory. We evaluated the proposed model fine-tuning and then compare the model performance against other methods in terms of the image classification accuracy and the computational speed.

Table 2. Experimental results for various model structures, where D denotes network depth and L denotes the number of FC layers. See texts for explanation.

Model	$D5L1$	$D8L1$	$D11L1$	$D7L3$
Structure	2×2 Conv, 8, /2 ReLU 2×2 Conv, 16, /2 ReLU 2×2 Conv, 16, /2 ReLU 2×2 Conv, 16, /2 ReLU Linear, 64, 10	5×5 Conv, 8, /2 ReLU 3×3 Conv, 8 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU Linear, 64, 10	5×5 Conv, 8 ReLU 3×3 Conv, 8 ReLU 3×3 Conv, 8 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16, /2 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU Linear, 16, 10	5×5 Conv, 8, /2 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU 3×3 Conv, 16 ReLU Linear, 1024, 256 ReLU Linear, 256, 256 ReLU Linear, 256, 10
Plain DNN Acc.	60%	72.5%	77.5%	80%
Secure DNN Acc. wo LUT-Aware	47.5%	55%	62.5%	72.5%
Secure DNN Acc. w/ LUT-Aware (Ours)	**57.5%**	**70%**	**75%**	**80%**
Total Time	881 s	2,092 s	9,458 s	2,558 s

To ensure sufficient security for data encryption, we set the crypto parameters of PEGASUS to be at least 119-bit security. Table 1 shows detailed parameters used in our implementation.

4.1 Performance

In our experiments, the accuracy obtained from the original plaintext NN inference is treated as an upper bound of those obtained from the encrypted NN that are under evaluation. We evaluated four CNN models with varying architectures and network depths. Let D denote the network depth and L denote the number of fully connected (FC) layers in the network. Specifically, we have evaluated the networks of $D5L1$, $D8L1$, $D11L1$, $D7L3$. Table 2 presents the results, including the hyper-parameters, obtained accuracy, and computational

time. 'Plain DNN Acc' shows the obtained accuracy from the given plain model. 'Secure DNN Acc. wo LUT-Aware' shows the accuracy of models prior to applying LUT-aware model fine-tuning. 'Secure DNN Acc. w/ LUT-Aware' shows the accuracy after fine-tuning. '$k \times k$ Conv, c, /s' denotes a convolution layer with kernel size $k \times k$, output channel c and stride s. 'Linear, d_i, d_o' indicates a fully connected layer with input dimension d_i and output dimension d_o. Results in Table 2 indicate that our models after fine-tuning outperforms the original PEGASUS-based secure DNN models by 7.5% to 15% of accuracy. The $D7L3$ architecture achieved the best accuracy and execution time. Specifically, our $D7L3$ model finished the encrypted DNN classification of a CIFAR-10 image in 43 minutes using only 8 threads of CPU and < 50 GB of runtime memory.

Regardless of the model architecture or depth, our method consistently achieves accuracy scores that are comparable to the target performance of "Secure DNN Acc. wo LUT-Aware". These findings showcase the effectiveness of our approach in maintaining high accuracy while ensuring the security of the inference process.

4.2 Comparison with Two State-of-the-art Methods

We performed a comparative analysis between our method and two state-of-the-art implementations that support Fully Homomorphic Encryption (FHE)-based Deep Neural Network (DNN) inference, namely, Shift-accumulation-based LHE-enabled Deep Neural Network (SHE) [24] and Privacy-Preserving Machine Learning (PPML) [21]. SHE employed the FHEW/TFHE Bitwise scheme, whereas PPML utilized the CKKS scheme. Our secure $D7L3$ model after fine-tuning has almost no loss in accuracy compared to the plain DNN inference, as shown in Table 2. In comparison, our other models yield about 2.5% loss.

We next compare the execution time with SHE and PPML. We report the computational time required for the $D7L3$ model that are implemented on SHE and PPML and compare with our approach. It is stated in the SHE [24] paper that the level-TFHE inference scheme has the ability to perform homomorphic AND operations up to a depth of 32K in LHE mode, all without requiring bootstrapping. However, it is important to note that this claim overlooks a significant detail. The computation relies on a specific branching program structure [3], which is noticeably absent in the computation structure presented in [24]. Therefore, it is crucial to reassess and possibly reconsider their claims regarding the utilization of level-TFHE.

Compare the execution time with SHE [24]. We next report the runtime for executing a DNN using the FHEW/TFHE Bitwise scheme [27], employing the same security level as our approach. It should be noted that a logical operation takes 75 ms when executed on a single thread. Based on this, the estimation indicates that an 8-bit addition operation would take approximately 0.3744 seconds (8 thread), while an 8-bit ReLU computation would take around 0.126 seconds (8 thread). The multiplication operation time of SHE-DNN is incredibly fast when compared to the time needed for addition and ReLU. In fact, it's so efficient that it can practically be disregarded. We estimate that running

the SHE-DNN technology on the $D7L3$ architecture with an 8-bit model would require at least 347,555 seconds, including 346,672 seconds for 925,942 addition instructions and 883 seconds for 7,008 ReLU operations.Our approach requires less time to execute the $D7L3$ model compared to the SHE-DNN method.

Compare the execution time with PPML [21]. PPML adopts the CKKS scheme, in which bootstrapping and ReLU are performance bottlenecks. We estimate that the total execution time for a single bootstrap operation in the PPML method with 8 threads is 0.231 seconds, whereas performing a single ReLU operation takes 0.28 seconds. $D7L3$ consists of 7,008 ReLU operations and bootstrappings. Therefore, we estimate that executing the $D7L3$ model in PPML will take at least 3,581 seconds. Our method is slightly faster than that.

5 Other Related Works

Secure Multiparty Computation (SMC) methods achieve data confidentiality by using cryptography tools, where the security level can be evaluated with mathematical derivations. Generally, SMC methods can be categorized into *interactive* schemes [32,15,18] and *non-interactive* schemes [2]. Interactive schemes require the active participation of all involved parties during the computation. Participants must possess sufficient computation power and a reliable network to maintain their online presence and manage the communication overheads. In contrast, non-interactive SMC implementation, such as the *Homomorphic Encryption* (HE), can perform the entire NN encrypted inference locally, without the need of communication nor client participation during the encrypted computation. The DNN inference method using the FHEW/TFHE scheme in [2] belongs to this category. It achieves 96% accuracy on the MNIST dataset. However, direct adaptation of the method to CIFAR-10 and other complex datasets is not yet investigated.

Differential Privacy (DP) is a mainstream method for privacy protection, which achieves data anonymization by introducing noise to the data or the model [12]. DP methods are primarily used to avoid the leakage of personal information in a dataset [1,28]. In practical, DP methods need to select suitable parameters according to the characteristic of a problem. However, no general method exists to define the parameter to achieve privacy-preserving [11].

6 Conclusion

We presented a encrypted ML inference approach leveraging the CKKS-FHEW / TFHE hybrid crypto system based on the PEGASUS implementation. We showed that direct incorporation of PEGASUS would result in a 10% accuracy drop, when compared with plain text inference. We then proposed a LUT-aware model fine-tuning framework that significantly improves the accuracy by 7.5% to 15% accuracy over the original FHE inference without fine-tuning. We compared

our method with two state-of-the-art implementations, namely PPML and SHE, and demonstrated competitive performance of ours in accuracy and efficiency.

Future work includes further explorations of design and methods that can effectively address the computational overhead associated with the encrypted FHE NN inference. Furthermore, how best to facilitate a feasible and scalable deployment of secure ML applications continues to be an active area of research.

Acknowledgements This work is supported in part by the National Science and Technology Council, Taiwan, under the grants NSTC 112-2119-M-002-017 and MOST-111-2221-E-006 -116 -MY3.

References

1. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: ACM SIGSAC conference on computer and communications security. pp. 308–318 (2016)
2. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 483–512. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_17
3. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) ITCS 2014. pp. 1–12. ACM (Jan 2014). https://doi.org/10.1145/2554797.2554799
4. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37. pp. 360–384. Springer (2018)
5. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Heidelberg (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
6. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: ASIACRYPT. pp. 409–437. Springer (2017)
7. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Report 2018/421 (2018), https://eprint.iacr.org/2018/421
8. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
9. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_24
10. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: EUROCRYPT. pp. 617–640. Springer (2015)
11. Dwork, C.: Differential privacy: A survey of results. In: Theory and Applications of Models of Computation (TAMC). pp. 1–19. Springer (2008)

12. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science **9**(3–4), 211–407 (2014)
13. Gentry, C.: A fully homomorphic encryption scheme. Stanford university (2009)
14. Gentry, C.: Computing arbitrary functions of encrypted data. Communications of the ACM **53**(3), 97–105 (2010)
15. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 307–328 (2019)
16. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial nets. In: Neural Information Processing Systems. pp. 2672–2680 (2014)
17. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: IEEE CVPR (June 2018)
18. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: USENIX Security Symposium. pp. 1651–1669 (2018)
19. Kluczniak, K., Schild, L.: FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2021/1135 (2021), https://eprint.iacr.org/2021/1135
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Neural Information Processing Systems. pp. 1106–1114 (2012), https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
21. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., et al.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10**, 30039–30054 (2022)
22. Liu, X., Xie, L., Wang, Y., Zou, J., Xiong, J., Ying, Z., Vasilakos, A.V.: Privacy and security issues in deep learning: A survey. IEEE Access **9**, 4566–4593 (2021). https://doi.org/10.1109/ACCESS.2020.3045078, https://doi.org/10.1109/ACCESS.2020.3045078
23. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 130–160. Springer, Heidelberg (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_5
24. Lou, Q., Jiang, L.: SHE: A fast and accurate deep neural network for encrypted data. In: Neural Information Processing Systems. pp. 10035–10043 (2019)
25. jie Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy. pp. 1057–1073. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00043
26. Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F.H.P., Aaraj, N.: Survey on fully homomorphic encryption, theory, and applications. IEEE **110**(10), 1572–1609 (2022)
27. Micciancio, D., Polyakov, Y.: Bootstrapping in FHEW-like cryptosystems. In: WAHC. pp. 17–28 (2021)
28. Papernot, N., Song, S., Mironov, I., Raghunathan, A., Talwar, K., Erlingsson, Ú.: Scalable private learning with pate. arXiv preprint arXiv:1802.08908 (2018)

29. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
30. Ribeiro, M., Grolinger, K., Capretz, M.A.: Mlaas: Machine learning as a service. In: IEEE ICMLA. pp. 896–902 (2015)
31. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Neural Information Processing Systems. pp. 3104–3112 (2014)
32. Yao, A.C.: Protocols for secure computations. In: SFCS. pp. 160–164. IEEE (1982)