QuripfeNet: Quantum-Resistant IPFE-based Neural Network

KyungHyun Han, Wai-Kong Lee, *Member, IEEE,* Angshuman Karmakar, Myung-Kyu Yi, *Member, IEEE,* and Seong Oun Hwang, *Senior Member, IEEE*

***Abstract*—In order to protect the sensitive information in many applications involving neural networks, several privacy-preserving neural networks that operate on encrypted data have been developed. Unfortunately, existing encryption-based privacy-preserving neural networks are mainly built on classical cryptography primitives, which are not secure from the threat of quantum computing. In this paper, we propose the first quantum-resistant solution to protect neural network inferences based on an inner-product functional encryption scheme. The selected state-of-the-art functional encryption scheme based on lattice-based cryptography works with integer-type inputs, which is not directly compatible with neural network computations that operate in the floating point domain. We propose a polynomial-based secure convolution layer to allow a neural network to resolve this problem, along with a technique that reduces memory consumption. The proposed solution, named QuripfeNet, was applied in LeNet-5 and evaluated using the MNIST dataset. In a single-threaded implementation (CPU), QuripfeNet took 107.4 seconds for an inference to classify one image, achieving accuracy of 97.85%, which is very close to the unencrypted version. Additionally, the GPU-optimized QuripfeNet took 25.9 seconds to complete the same task, which is improved by 4.15 $\times$ compared to the CPU version.**

***Index Terms*—Inner-product functional encryption, Ring-learning with errors, Graphics processing units, Convolution Neural Network, privacy-preserving.**

I. Introduction

NEURAL network (NN) is widely used in many applications and has greatly revolutionized our daily lives. Commonly, these applications are hosted on a cloud server and are offered to users in the form of artificial intelligence (AI) as a service. The cloud server receives data from the users and performs the neural network inference whenever users need to use the service. Such services can greatly reduce the entry barrier to use AI technology, because the users do not need to train the AI models by themselves, which take a long time to develop and are very computationally expensive. However, such services may also expose sensitive data that users are unwilling to share with the cloud server. Due to these concerns, privacy-preserving neural network has emerged as a promising research direction in recent years.

KyungHyun Han, Myung-Kyu Yi and Seong Oun Hwang are with the Department of Computer Engineering, Gachon University, Seongnam 13120, South Korea (e-mail: sohwang@gachon.ac.kr).

Wai-Kong Lee is with the Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman.

Angshuman Karmakar is with the Indian Institute of Technology, Computer Science and Engineering, Kanpur, India.

In general, NN training requires a lot of training data to achieve a high accuracy. However, some sensitive information needs to be hidden in order to avoid misuse, so the privacy preservation approaches always try to hide as much information as possible. In the past, these two conflicting goals limit the use of NN on applications that deal with non-sensitive data only, which greatly constrained the potential of AI. To achieve these two conflicting goals simultaneously, several privacy-preserving neural networks (PPNNs) that operate on encrypted private data have been developed in recent years [1]–[5].

A. Motivations

1) Difference of HE-based PPNN and FE-based PPNN: In general, PPNNs using encryption can be constructed based on homomorphic encryption (HE) or functional encryption (FE). However, PPNNs support different application scenarios according to HE-based or FE-based.

The characteristic of HE is a computation between encrypted data and that's result is another encrypted data which can be decrypted. HE supports that all of the calculations in NN operate by encrypted data instead of plaintext data. A server can operate NN accurately using the data provided by a client in encrypted form, and the result can be decrypted by the client only, using the same encryption key [1]–[3]. It is suitable to the AI outsourcing scenario. For example, AI diagnostic service should provide the diagnosis result through a NN, but the patient data are considered sensitive. If we use HE-based PPNN, the patient can get their own diagnosis results without exposing his/her own personal information [2]. On the other hand, the server only knows the result but it cannot collect any personal information, meaning it cannot make any statistic information as well.

The characteristic of FE is a computation for encrypted data and that's result is a plaintext. FE supports that the first calculations in NN operate by encrypted data instead of plaintext data. A server can operate NN accurately using data a client encrypt, and the server can know the result without learning the related client data [4]–[8]. It is suitable to the AI scenario for public interest. For example, users want that server to filter out spam mails without server's knowing the contents in the mail. The system proposed by [5] was implemented to perform spam filtering while maintaining privacy in emails. Another notable example [6] demonstrated an electricity theft detection system that does not expose the lifestyles of consumers.

2) Quantum Resistant PPNN: HE and FE are cryptography schemes built upon the mathematical problems that are hard to be solved in polynomial time, thus offering sufficient security against adversarial attacks. However, the development of quantum computers is advancing rapidly recently [9]. This creates serious concerns on the classical cryptography schemes that relies on mathematical problems which are vulnerable to Shor's algorithm [10] such as Rivest-Shamir-Adleman (RSA) and Elliptic-Curve Digital Signature Algorithm (ECDSA). This can also be a threat to existing PPNNs that were designed based on the vulnerable classical cryptography algorithms. Most HE schemes were developed based on the hardness of lattice-based problems [11], which are resistant to threats from quantum computing. However, most of the FE-based PPNNs proposed in the recent years [4]–[6], [12] were designed based on the classical cryptography algorithms, thus susceptible to the threat from the quantum computers and Shor's algorithm. In other words, existing AI systems that employ FE schemes based on these classical cryptography algorithms could expose sensitive data. Furthermore, the original unencrypted data can be exposed after quantum computing's wide availability, even though the data were encrypted before. An attacker can store the existing encrypted data and then decrypt it later when quantum computers are widely available. This is known as *harvest-now decrypt-later* attack [13], creating a very critical issue to the community, especially for data that are not modifiable, such as bio-information. In other words, these FE-based PPNNs are already facing privacy issues now, even though the quantum computers are still not powerful enough to fully exploit these vulnerabilities yet.

3) Goal: Therefore, we need to develop a new FE-based PPNN relying on post-quantum cryptography (PQC) algorithms to avoid the quantum computer's threat. However, replacing the PPNN schemes that are developed from classical cryptography with those developed using PQC is not a trivial task. For instance, the lattice-based PPNN works with integer-type inputs, which is incompatible with the NN's data that operate in the floating point. Hence, we need new techniques to ensure compatibility between NN's data and the quantum resistant FE techniques' input, to process NN and PQC with low memory. Additionally, there is no available GPU libraries that can support PPNN schemes that combining PQC schemes and AI computations despite of AI libraries popularly use GPU platform to speed up computations. Therefore in this work we would like to address the following issues: (1) modify both sides of PQC algorithm and NN layer, (2) implement the PPNN schemes on GPU for achieving sufficient speed performance.

B. Our contributions

In this paper, we propose a quantum-resistant, FE-based neural network, named QuripfeNet, to preserve data privacy in the pre- and post-quantum computing era. Our contributions are summarized as follows.

- We propose the first quantum-resistant, FE-based PPNN with practical instantiation and implementation. Unlike previous FE-based PPNNs, the proposed QuripfeNet can provide continuous protection, even in future situations where commercial-grade quantum computers are widespread. We applied QuripfeNet on a LeNet-5 convolution neural network (CNN) trained on the MNIST dataset. Moreover, our proposal is not limited to LeNet-5, because it is designed to flexibly support various CNN models. Our implementation is available in the public domain: https://github.com/hkh112/QuripfeNet-with-GPU.
- QuripfeNet was developed based on a state-of-the-art IPFE scheme of Mera et al. [14] that is quantum-resistant. Implementing the RLWE-IPFE straightforwardly on a CNN requires huge amounts of memory, which is not practical in real-world applications. To resolve this issue, we propose a new set of parameters for RLWE-IPFE that are friendly to computations on a CNN. Using the proposed parameter set, the input length of IPFE scheme is reduced to accommodate the AI filter size, leading to a much smaller memory consumption. The time to perform encryption, key generation, and decryption is also reduced to 27 ms, 1.4 ms, and 45 ms, which is 14.1$\times$, 15.7$\times$, and 4.3$\times$ faster than the original parameter set. This allows us to infer data using a CNN with reasonable memory consumption for various AI applcations.
- The RLWE-IPFE [14] is not directly compatible with the floating-point and negative-vector operations commonly found in a CNN. To resolve this issue, we propose a polynomial-based secure convolution layer that performs the convolution operation by converting original floating-point values to polynomials. Through experiments, we found that the polynomials with more terms provide higher training accuracy, but the performance is slower. For example, the performance using 2-term polynomial is 3.12–3.23$\times$ slower than the 1-term version. This can be viewed as a trade-off between accuracy and performance, which can be chosen according to the needs in different applications. This allows us to apply RLWE-IPFE for privacy protection without sacrificing accuracy.
- The proposed QuripfeNet has been implemented on a CPU and a GPU. For a single-threaded CPU implementation, it takes 107.4 seconds to classify one image on the MNIST dataset, which can be slow for some time-sensitive applications. To improve this, we implemented a simple solution of QuripfeNet first by parallelizing it on a GPU, which takes 29.1 seconds to classify one image. We also proposed a coarse-grain approach to optimize the GPU implementation, wherein it takes only 25.9 seconds to complete the same task. The optimized solution demonstrates a 4.14$\times$ speed-up compared to the CPU version and a 1.12$\times$ speed-up compared to the simple solution on GPU.

II. Preliminaries

A. Computing on Encrypted Data

Due to the rapid advancement of cloud computing, many cloud-based applications have been developed in the past decade. One of the main trends in recent years is the use of AI services hosted on a cloud server. These services

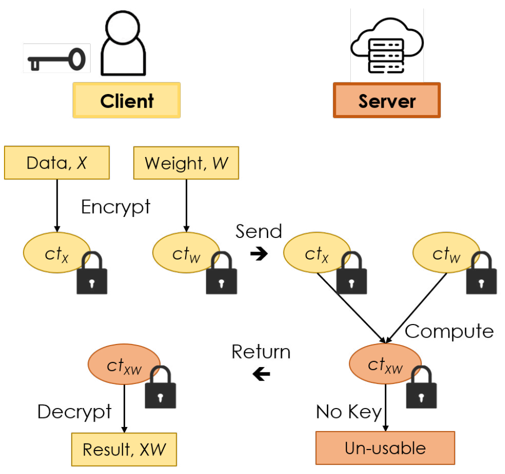

Fig. 1. The process of homomorphic encryption. The server cannot use the result of computation because it is already encrypted. Only the client that encrypted the data can decrypt and consume it.

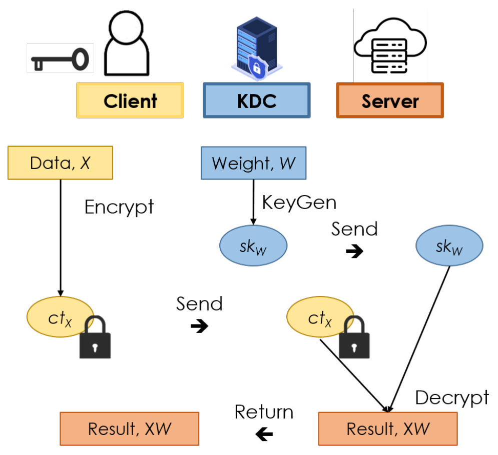

Fig. 2. The process of functional encryption. With authorization from the client, the server can utilize the output of the function (e.g., inner product) and perform subsequent computations.

process a lot of data from users, which can be sensitive due to privacy concerns. Hence, awareness of privacy issues in such applications has also increased in recent years. Although encryption algorithms such as ECDSA or RSA cryptography can be employed to preserve data privacy, they do not support arbitrary computations in the encrypted domain, which makes computation outsourcing impossible (e.g., neural network inference). Computing on Encrypted Data (COED) is a cryptographic technique that allows certain computations to be performed on the encrypted data, which is an essential technique in designing a PPNN. There are two known methods for privacy preservation:

1) HE: Figure 1 shows the process of homomorphic encryption. When HE is employed, users encrypt their data and send them to a cloud server. Under such scenarios, the cloud server performs arbitrary computations without even accessing the original data. The results of the computation are sent back to the users, who have the secret key to decrypt it. In other words, the entire computational process that takes place in the cloud server does not reveal any information related to the users' data. Hence, the cloud server cannot use the processed results to perform any additional operations. Some operations in HE (e.g., homomorphic multiplication) introduces errors, which accumulate each time the operations are performed. To allow successful decryption, we need to remove these errors by performing additional work, i.e., bootstrapping [15]. As a result, HE is slow, especially when the application is complex and when data are to be processed by many NN layers.

2) FE: Figure 2 shows the process of functional encryption. FE is similar to the case in HE, wherein users encrypt their data and send them to the cloud server. The cloud server also obtains the functional key generated by a key distribution center with permission from users. However, in the case of FE, the cloud server or another party can calculate a predefined function (e.g., inner-product or quadratic residue) with the encrypted data and functional key. Under such a scenario, the cloud server or the third party still has no access to the original data, but they obtain results of the function, which is different from HE. The cloud server can utilize the results of the function to perform subsequent computations and at the same time preserve the privacy of users' data. Considering the application to NNs, another notable difference with HE is that only the first layer of the NN needs to be performed in the encrypted domain; the remaining layers can be performed in plaintext. As a result, the FE-based PPNN is relatively faster than HE. There are several FE schemes, such as generic group-model schemes [7], [12], decisional Diffie-Hellman-based schemes [16]–[18], etc.

B. Related Works

1) HE-based PPNNs: Since the NN performs various operations such as convolution and activation functions, HE is a natural candidate for developing a PPNN. However, the performance of a HE-based PPNN is slow because it requires bootstrapping and other steps that computationally heavy. For this reason, recent work focuses on reducing the runtime of a HE-based PPNN. For example, [3] reduced bootstrapping runtime with a multiplexed technique that packs data from multiple channels into one ciphertext in a compact manner. Similarly, [1] reduced the number of HE operations by proposing a HE-friendly mobile network architecture search algorithm and merging the mask while approximating the coefficients in activation and batch normalization.

2) FE-based PPNNs: To implement an FE-based PPNN, researchers use inner-product FE (IPFE) or quadratic residue FE (QFE) in which the selection is dependent on the first layer of the NN. QFE-based PPNNs are applied to an architecture that has an activation function such as ReLU as the first layer. For example, [5], [7] applied QFE to a first layer that uses the square function as the activation function. [12] applied QFE to a first layer that uses the ReLU function as the activation function. IPFE-based PPNNs are applied to architectures that have a fully connected layer or a convolution layer as the first layer. For instance, [6] applied IPFE to the first layer that uses linear and ReLU functions, as the activation function. [4] applied IPFE to a convolution layer. [19] proposed a

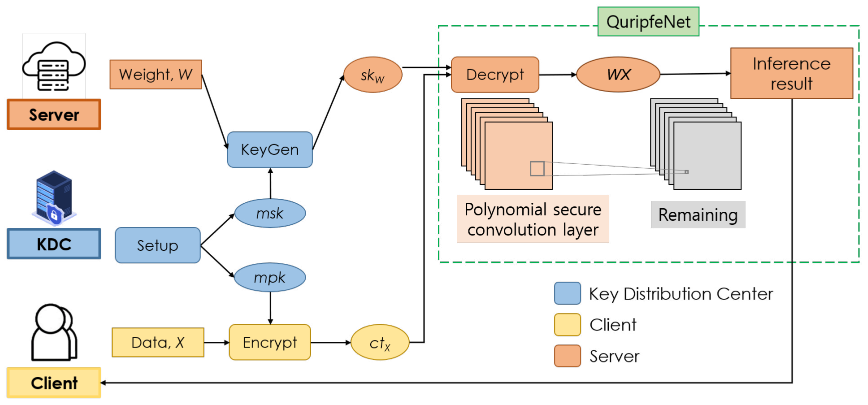

Fig. 3. The workflow of QuripfeNet inference, which involves three entities: server, key distribution center and client

framework to train a Neural Network over Encrypted Multi-sourced Datasets (NN-EMD). They proposed two protocols for horizontally/vertically partitioned computation, which are using functional encryption schemes. Also, they suggested to enhance the privacy guarantee by integrating other privacy-preserving DNN approaches such as split learning. [20] applied Function Hiding Inner Product Encryption (FHIPE) for PPNN. It is called function-hiding if the ciphertexts and keys disclose no extra information about vectors. FHIPE is more secure since it provides simulation-based security additionally.

Existing FE schemes are mostly designed based on classical cryptography with hardness assumptions, e.g., Decision Diffie-Hellman (DDH) [4], [12], [16]–[18], Matrix Decision Diffie-Hellman (MDDH) [6], and the Generic Group Model (GGM) [5], [7], [12], which are susceptible to threats from quantum computing. For this reason, PPNNs developed based on such FE schemes are also vulnerable in the quantum computing era. Recently, some post-quantum FE schemes have been proposed, which can be used to build a PPNN that is secure against quantum algorithms such as Shor's algorithm [10]. There is an existing library (CiFEr) [21] which supports some PQC FE schemes. However, prior research [20], [22] that are based on CiFEr library to build PPNNs, did not use the PQC FE schemes; they only used DDH based scheme. In other words, there is no quantum-resistant FE-based PPNN. In this paper, we propose an FE-based PPNN relying on RLWE-IPFE [14], which is more future-oriented compared to existing solutions.

C. RLWE-IPFE

RLWE-IPFE [14] is a state-of-the-art IPFE scheme designed based on hard lattice problem ring-learning with errors. Unlike classical hard problems such as integer factorization or discrete logarithm problems, there exists no quantum algorithm that can solve these hard lattice problems efficiently. Hence, lattice-based problems are very popular candidates for constructing post-quantum cryptography. Earlier quantum-resistant IPFE schemes [18], [23] are built on top of the Learning With Errors (LWE) lattice problem, which is known to be inefficient in practical implementation. Unlike these schemes, the RLWE-IPFE [14] is built on Ring-LWE lattice problem, allowing it to have smaller key sizes and faster execution. The implementation of polynomial multiplication in RLWE-IPFE can also be accelerated through asymptotically fast algorithm like number theoretic transform (NTT), which makes it efficient and practical. Similar to other IPFE schemes, there are four main operations in RLWE-IPFE: *Setup, KeyGen, Encrypt*, and *Decrypt*. *Setup* returns a master public key *mpk* and a master secret key *msk* for a given security level. The *KeyGen* function accepts the *msk* and an operand *y*; it returns a secret-key sk_y associated with the inner-product. The *Encrypt* function accepts the master public key *mpk* and a message *m*; it returns the ciphertext ct_m. Finally, the Decrypt function accepts sk_y and ct_m as inputs; it returns $f(m)$ which is an inner-product between *m* and *y*. Note that the length of both *m* and *y* are *L*. The parameter *N*, *N_mod* is dependent on other security parameters, which will be discussed in Section III-B1.

III. QuripfeNet: A Quantum-resistant IPFE-based Neural Network

A. Overview

Figure 3 shows the workflow of QuripfeNet, which involves three entities: the key distribution center (KDC), the client, and the server, which has a pre-trained model.

Initially, the KDC sets the RLWE-IPFE parameters [14]; then, it generates a master public key, mpk, and master secret key msk by running the *Setup* algorithm; mpk is published to all the entities for the follow-up processes, but msk is kept secret. Upon request from the server, the KDC generates functional key sk_W for the weights, W, by invoking the algorithm KeyGen(msk, W). Functional key sk_W will be used to produce the inner-product results. The client prepossesses data, X, and encrypts them by executing an algorithm, Enc(mpk, X), producing the ciphertext, ct_X. After receiving ct_X from the client, the server calculates the convolution operation, $a = g(WX + b)$, in which g is the activation function and b is the bias. The server asks the KDC to generate sk_W, which can be used to calculate the inner-product of W. Then, the server gets WX using Dec(mpk, ct_X, sk_W, W). In this process, the server does not have any information about X; it only has the inner-product results WX. Finally, the server calculates $a = g(WX + b)$ with a general plaintext operation. Note that the polynomial secure convolution layer is proposed to handle

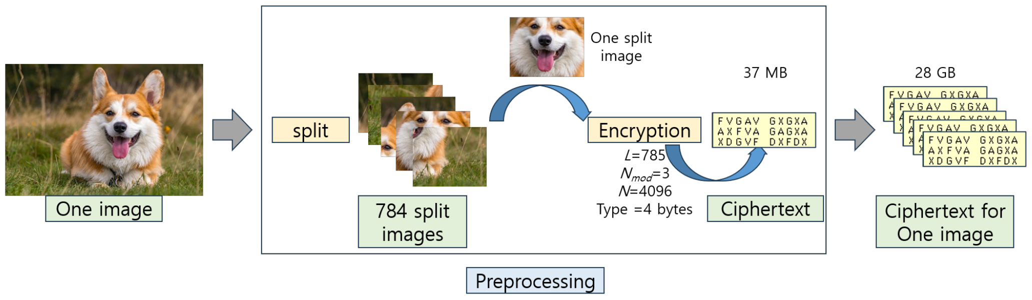

Fig. 4. The preprocessing way for FE-based PPNNs. One image is split into multiple sub-images before the encryption in order to produce the ciphertext. Here we consider the "Medium" security level of RLWE-IPFE [14].

TABLE I
PARAMETERS AND THE PERFORMANCE OF THE ORIGINAL AND THE PROPOSED RLWE-IPFE.

Security level	PQ Security	FE Bounds	Gaussian Parameters	Ring Parameters	CRT moduli	Time (ms)
Low	76.3	$B_x : 2$ $B_y : 2$ $L = 64$	$\sigma_1 : 33$ $\sigma_2 : 59473921$ $\sigma_3 : 118947840$	$N : 2048$ $\lceil log\ q \rceil : 66$	$q_1 : 2^{14} - 2^{12} + 1$ $q_2 : 2^{23} - 2^{17} + 1$ $q_3 : 2^{29} - 2^{18} + 1$	Setup : 26 Enc : 16 KG : 0.27 Dec : 1
Medium	119.2	$B_x : 4$ $B_y : 16$ $L = 785$	$\sigma_1 : 225.14$ $\sigma_2 : 258376412.19$ $\sigma_3 : 516752822.39$	$N : 4096$ $\lceil log\ q \rceil : 86$	$q_1 : 2^{24} - 2^{14} + 1$ $q_2 : 2^{31} - 2^{17} + 1$ $q_3 : 2^{31} - 2^{24} + 1$	Setup : 589 Enc : 381 KG : 22 Dec : 17
High	246.2	$B_x : 32$ $B_y : 32$ $L = 1024$	$\sigma_1 : 2049$ $\sigma_2 : 5371330561$ $\sigma_3 : 10742661120$	$N : 8192$ $\lceil log\ q \rceil : 101$	$q_1 : 2^{17} - 2^{14} + 1$ $q_2 : 2^{20} - 2^{14} + 1$ $q_3 : 2^{32} - 2^{20} + 1$ $q_4 : 2^{32} - 2^{30} + 1$	Setup : 1743 Enc : 1388 KG : 70 Dec : 45
Proposed	131.44	$B_x : 32$ $B_y : 32$ $L = 25$	$\sigma_1 : 321$ $\sigma_2 : 65740801$ $\sigma_3 : 131481600$	$N : 4096$ $\lceil log\ q \rceil : 86$	$q_1 : 2^{24} - 2^{14} + 1$ $q_2 : 2^{31} - 2^{17} + 1$ $q_3 : 2^{31} - 2^{24} + 1$	Setup : 46 Enc : 27 KG : 1.4 Dec : 3.9

issues pertaining to the implementation of RLWE-IPFE, which will be discussed in Section III-B2.

The workflow of QuripfeNet is similar to CryptoNN [4] since both PPNNs have the convolution layer as the first layer. The difference is that QuripfeNet utilizes RLWE-IPFE [14] instead of the classical FE, which is not quantum-resistant. There are several issues in employing the RLWE-IPFE scheme to construct QuripfeNet, which are discussed below.

Issue 1: Huge ciphertext size. In general, the size of the ciphertext produced by RLWE-IPFE is larger than the plaintext. This can be a problem for a PPNN because the ciphertext size may grow significantly large. In particular, the client must split the image and encrypt all of the split image data [4]. This is due to a convolution requirement wherein the kernel slides through the entire image using a fixed stride. However, the cloud server cannot perform this sliding process because the image is already encrypted, so it is impossible to know the location of image pixels, and thus, it is impossible to apply the sliding operation. The only way to overcome this is to split images and encrypt the segments separately [4]. Figure 4 shows the preprocessing way for FE-based PPNNs. As an example, in LeNet-5, the sliding policy splits one image into 784 images. On top of that, RLWE-IPFE generates almost 37 MB of ciphertext for one batch of plaintext. The ciphertext size is $L \times N_{mod} \times N \times$ double-type size, in which L=785, N_{mod}=3, N=4096, and double-type size=4 bytes. Hence, if we follow this naive way of applying RLWE-IPFE to a CNN, the ciphertext size will be 28GB for one image; the memory consumption is too huge to be practical.

Issue 2: Does not support floating-point arithmetic. NN training is typically conducted in the floating-point domain wherein the weights are updated and stored as floating-point values. However, RLWE-IPFE only accepts integer vectors as input, because the arithmetic in RLWE-IPFE is conducted in polynomial form. To solve this issue, one can convert the weights from floating-point to integer values, but this may lose important information, and affects NN classification accuracy. Therefore, we need to consider how to preserve the information in weights while employing RLWE-IPFE.

Issue 3: Small coefficient limits the range of input arguments. Three parameter sets are proposed by RLWE-IPFE (see Table. I) wherein the medium security level seems to be the most suitable for QuripfeNet. However, the range of input coefficients is limited by parameters B_x and B_y, which are rather small. In particular, input has to be $X = 0, 1, 2, 3$ and $W = 0, 1, ..., 15$. However, we found that a typical deep learning dataset (e.g., MNIST) uses 32-bit integers to represent the image data. Moreover, the trained weights are represented in 32-bit single precision floating-point. This shows that the input (NN and dataset) are clearly incompatible with the parameters at the medium security level, because they are too large to be represented correctly in RLWE-IPFE polynomials.

Note that using the parameters from a high security level does not solve this problem because B_x and B_y are still small. Moreover, a higher security level requires more computations due to the larger polynomial length ($N = 8192$).

Issue 4: Unable to process negative vectors. After the training process, weights in the NN may contain negative values. Some classical IPFEs [18] support calculating negative vectors in the encrypted domain, but that is not supported in RLWE-IPFE. For this reason, we cannot correctly perform the inner-product functional encryption scheme using RLWE-IPFE, which eventually affects NN accuracy adversely. A solution is required to avoid this issue.

B. QuripfeNet in Detail

In this paper, we propose several techniques to solve the aforementioned problems, which are described in detail here.

1) The RLWE-IPFE parameter set:

Solution to Issue 1: Reducing the ciphertext size. After analyzing the factors that affect ciphertext size, we found that the CNN convolution needs to execute the sliding policy and produce 784 split images, which cannot be omitted. Hence, we turn our focus to reducing the ciphertext size per image segment, which is related to parameters L, N_{mod}, and N. Among them, L is related to the length of input to be processed under RLWE-IPFE, which can be modified according to different application scenarios. Since convolution in the CNN uses a small filter size (e.g., 5×5 was used in LeNet-5), we changed parameter L according to the target CNN. In this paper, we set $L = 25$ and evaluated QuripfeNet in LeNet-5; the resulting ciphertext was reduced by 97% to 0.9 GB.

Solution to Issue 3: Small coefficient size for input. The maximum value of inner-product results in RLWE-IPFE is bounded by $P = B_x \times B_y \times L$. Taking the medium security level as an example, $P = 4 \times 16 \times (785+1) = 50241$. Since we reduced L to 25, we can allow a higher range for B_x and B_y. In this paper, we set $B_x = 32$ and $B_y = 32$, so $P = 26,624$, which is less than the maximum allowable value (i.e., 50241). In this way, the representation range for data and weight increases by eight times and two times, respectively. Note that one can also configure other parameter sets according to application-specific requirements for input and weight. For instance, one can define a larger P to accommodate larger inner-product results, which eventually affects other parameters (i.e., N, CRT moduli, and Gaussian parameters).

The proposed RLWE-IPFE parameters for QuripfeNet and its performance are shown in Table. I. This new parameter set has security performance similar to the medium-level parameters, with 4× to 15× faster performance owing to the smaller L.

Note that, some CNN models use a bigger filter (i.e., 7×7). In this case, we can set L to 49, while P is still not over the maximum allowable value. In other words, the proposed parameter set can be used for the most of the CNN models which have a filter size within 7×7 of first layer. Even if the CNN model has a larger filter size, one can derive a new parameter set (including smaller B_x and B_y) by referring to the above consideration.

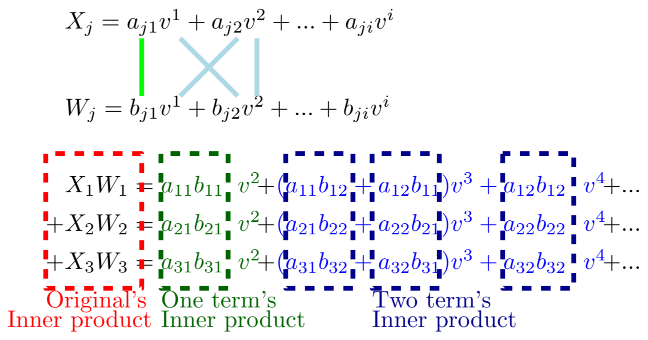
$X_j = a_{j1}v^1 + a_{j2}v^2 + ... + a_{ji}v^i$
$W_j = b_{j1}v^1 + b_{j2}v^2 + ... + b_{ji}v^i$
$X_1W_1 = a_{11}b_{11}\ v^2 + (a_{11}b_{12} + a_{12}b_{11})v^3 + a_{12}b_{12}\ v^4 + ...$
$+X_2W_2 = a_{21}b_{21}\ v^2 + (a_{21}b_{22} + a_{22}b_{21})v^3 + a_{22}b_{22}\ v^4 + ...$
$+X_3W_3 = a_{31}b_{31}\ v^2 + (a_{31}b_{32} + a_{32}b_{31})v^3 + a_{32}b_{32}\ v^4 + ...$
Original's Inner product
One term's Inner product
Two term's Inner product

Fig. 5. Polynomial representation: convert floating point weights to the proposed polynomial integer form

2) Polynomial secure convolution layer:

Solution to Issue 2: Polynomial representation. In previous research, this problem was usually solved through two methods:

1) Keep the i-th decimal places and discard the rest. This is equivalent to approximation of the original value [4].
2) Use a quantized model [2].

The first method is easy to implement, but it discards a lot of important information. The second method requires the NN (hosted on the cloud server) to be trained in the quantized form, which may not always be possible. In this paper, we propose a new solution to this problem: convert floating point weights to polynomial integer form. Referring to Figure 5, data X_j and weight W_j can be represented in polynomial form. Note that, in the case where NN has a 5×5 size filter, $j = 0, 1, ...25$. The convolution layer is computing the sum of X_jW_j (the red box in Figure 5), which involves inner-product. We convert all data and weights to polynomial form. Then X_jW_j is represented like Figure 5. We can calculate the whole inner-product by summing the inner-products (the green and blue box in Figure 5) of each coefficient considering radix v (the green and blue box in Figure 5). In this way, we can control the precision of the polynomial multiplication result by changing the number of terms and radix v. Note that we should keep the number of terms to the minimum, because each term introduces additional computations. For example, the one-term case requires one calculation (the vertical green line), but the two-term case requires four calculations (the green line and three brown lines). These extra computations can have a detrimental effect on performance speed because the inner-product operations are repeated many times in a typical NN architecture.

To decide on a suitable radix and number of terms to be used, we performed a micro-benchmark and checked the corresponding accuracy. Table. II shows the accuracy without encryption to illustrate the effect of the polynomial convolution layer. Referring to Table. II, accuracy increases as radix v increases. If the polynomial has more terms, it can represent data and weight more precisely. However, v must be smaller than parameter B_x in Table. I so that the value does not overflow. Fortunately, the accuracy of the combination of one term and $v = 32$ is close to the peak accuracy. Hence, we used one term and $v = 32$ in our implementation.

TABLE II
ACCURACY ACCORDING TO THE NUMBER OF TERMS TESTED ON MNIST [24] AND LENET-5 [25].

	Accuracy (%)		
v	one term	2-term	3-term
2	65.26	96.02	97.23
4	96.02	97.81	97.87
8	97.23	97.87	97.86
16	97.81	97.86	97.86
32	97.85	97.86	97.86
64	97.87	97.86	97.86

Solution to Issue 4: Use a redundant negative vector. To calculate negative vectors in RLWE-IPFE, we make the following proposition.

Proposition 1. Consider X_P that contains only the positive elements of vector X, and consider X_N that contains only the negative elements of X. For vectors X and Y, if X is divided by X_P and X_N, $X \cdot Y = X_P \cdot Y - (-X_N) \cdot Y$.

Proof. The inner-product is the sum of the products of the corresponding entries from two sequences of numbers. Hence, it has the following properties as expressed in III-B2 and III-B2.

Lemma 2. For any vector X and Y, $X \cdot Y = -(-X) \cdot Y$ is satisfied.

Lemma 3. For any vector X and Y, if X is divided into X_1, ... X_n, $X \cdot Y = X_1 \cdot Y + ... + X_n \cdot Y$ is satisfied.

We can conclude that Proposition 1 is true using Lemma 2 and Lemma 3. Therefore, there is no problem if we utilize Proposition 1.

Note that $(-X_N)$ becomes a positive vector, so we have no issue in computing it using the RLWE-IPFE [14]. Therefore, we can calculate inner-product WX by invoking the algorithm Dec(mpk, ct_X, sk_{PW}, PW) - Dec(mpk, ct_X, sk_{-NW}, $-NW$), in which PW contains the positive elements from W, and NW contains the negative elements from W.

By combining these two solutions, we propose the polynomial secure convolution layer that is used in QuripfeNet. Referring to Figure 3, the server decomposes weight W into PW and NW, then converts them to polynomial forms $PolyPW$ and $PolyNW$. After that, the server receives functional keys sk_{PolyPW} and sk_{PolyNW} by requesting them from the KDC. With this information, the polynomial secure convolution layer can compute the inner-product in a CNN successfully. The details are shown in Algorithm 1.

First, the client encrypts the private data using the function pre-process (X) (line 5). For that, the client needs to decide on the padding and splitting strategy and the filter size, and then divides the data into the image segments (line 7). After that, the client decomposes the image into positive elements and negative elements (lines 9 and 10), converts it to polynomial form (lines 11 and 12), and encrypts it by using mpk (lines 13 and 14). These steps (lines 8-15) are repeated for all the split image segments. The client sends encrypted data CP and CN to the server (line 16).

In the second step, the cloud server needs to perform convolution of the client's CP and CN with the server's weight, W. For that, the server prepares $PolyPW$, $PolyNW$,

Algorithm 1 The polynomial secure convolution scheme

```
1: Input: master public key $mpk$, test image data $X$, polynomial weights $PolyPW$ and $PolyNW$, functional keys $sk_{PolyPW}$ and $sk_{PolyNW}$
2: Output: result $Z$
3: Server and client decide a suitable interval $term$.
4:
5: function pre-process($X$)
6: initialize an empty window list CP, CN
7: $X' \leftarrow X$ with padding and splitting
8: repeat
9:     $X_P \leftarrow$ the positive elements of vector $X'$
10:    $X_N \leftarrow$ the negative elements of vector $X'$
11:    $Poly_{XP} \leftarrow$ convert $X_P$ to polynomial form using $terms$
12:    $Poly_{XN} \leftarrow$ convert $X_N$ to polynomial form using $terms$
13:    $c_p \leftarrow$Enc$(mpk, Poly_{XP})$ into CP
14:    $c_n \leftarrow$Enc$(mpk, -Poly_{XN})$ into CN
15: until slide window finished;
16: return CP, CN
17:
18: function polynomial secure convolution(CP, CN)
19: initialize an empty matrix S, Z
20: repeat
21:    $s_1 \leftarrow$ Dec$(mpk, c_p, sk_{PolyPW}, PolyPW)$
22:    $s_2 \leftarrow$ - Dec$(mpk, c_n, sk_{PolyPW}, PolyPW)$
23:    $s_3 \leftarrow$ - Dec$(mpk, c_p, sk_{-PolyNW}, -PolyNW)$
24:    $s_4 \leftarrow$ Dec$(mpk, c_n, sk_{-PolyNW}, -PolyNW)$
25:    $u \leftarrow$ inverse from polynomial form using $s_1, s_2, s_3, s_4, term$
26:    $z \leftarrow g(u) + b$ into Z // activation function
27: until slide window finished;
28: return Z
```

sk_{PolyPW}, and sk_{PolyNW}. The server decrypts CP and CN using $PolyPW$ and sk_{PolyPW}, $PolyNW$ and sk_{PolyNW} sequentially (lines 21-24). Note that the results s_1, s_2, s_3, s_4 are coefficient values. The server must convert the polynomial results back to floating point by substituting the $term$ into each variable (line 25). Thus, this step needs to use the formula in a different way based on the chosen number of terms. Finally, the server calculates convolution result z using activation function g and bias b (line 26). These steps (lines 20-27) are repeated for all image segments.

C. GPU implementation of QuripfeNet

GPU is a well-known parallel platform to speed up computations. It was initially developed for graphics applications, but now becomes a very popular choice in accelerating artificial intelligence (AI) and machine learning applications. However, due to the complexity involved, there is no available GPU libraries that can support both PPNN schemes and AI computations. This sub-section describes the GPU implementation of QuripfeNet based on a recent work that parallelized the RLWE-IPFE scheme [26]. Specific techniques are also proposed to optimize the performance of QuripfeNet on a GPU.

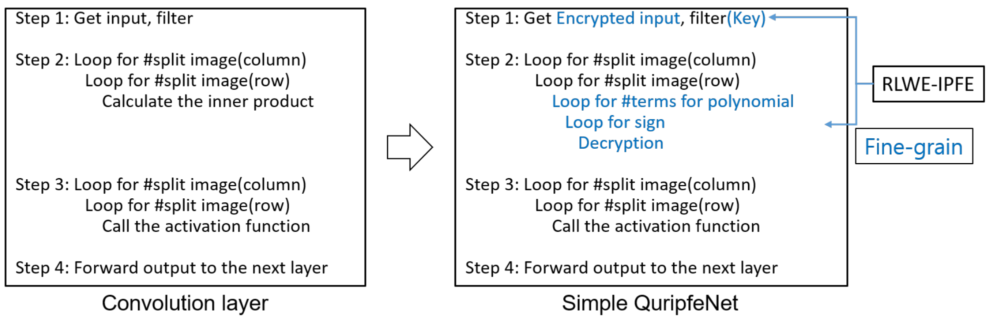
Step 1: Get input, filter
Step 2: Loop for #split image(column)
Loop for #split image(row)
Calculate the inner product
Step 3: Loop for #split image(column)
Loop for #split image(row)
Call the activation function
Step 4: Forward output to the next layer
Convolution layer
Step 1: Get Encrypted input, filter(Key)
Step 2: Loop for #split image(column)
Loop for #split image(row)
Loop for #terms for polynomial
Loop for sign
Decryption
Step 3: Loop for #split image(column)
Loop for #split image(row)
Call the activation function
Step 4: Forward output to the next layer
Simple QuripfeNet
RLWE-IPFE
Fine-grain

Fig. 6. The overview of major convolution and simple solution: in Step 2, the simple solution replaces the inner product with *Decrypt* function, which is repeated in the loop for all the polynomial terms and the sign.

Algorithm 2 The original convolution algorithm

```
1:  Input: split image data $x[column][row]$, filter $w$
2:  initialize $output[column][row]$, $t$
3:
4:  for $i = 0$; $i < column$ do
5:      for $j = 0$; $j < row$ do
6:          $t \leftarrow$ Inner-product($x[i][j]$, $w$)
7:          $output[i][j] \leftarrow$ Activation($t$)
8:      end for
9:  end for
10: return $output$
```

1) Simple GPU Implementation: Algorithm 2 shows the original convolution operation in CNN. It first gets the split image data and a filter (weight) as the parameter (line 1). Next, it calculates the inner product of one split image and the filter (line 4) and calls an activation function (line 5). This calculation is repeated during the loop for column and row (lines 2-7). The algorithm outputs the result (line 8).

Figure 6 shows a straightforward way to apply the CNN convolution to QuripfeNet. Only Step 1 and 2 are implemented on the GPU, the other steps are comparatively lightweight and thus executed on the CPU. In Step 1, the split image data is encrypted by *Encrypt*, while the filter is corresponding to a secret-key, generated by *KeyGen*. After that, both results containing arrays of polynomial terms and the sign for positive/negative vectors, are passed as inputs to Step 2, The computation of inner product (original convolution) is now replaced by *Decrypt*, because a *Decrypt* operation in QuripfeNet correspond to the decryption of inner-product results from IPFE-RLWE [14]. This process is repeated for all the terms and the signs inside the loop. Step 3 and 4 correspond to the activation function and convolution output. Note that the implementation of Step 2 is actually exploiting the fine-grain parallelism, which is explained in detail in Algorithm 3.

The simple solution shown in Algorithm 3 only exploits the inner parallelism that exists in RLWE-IPFE [14]. It receives encrypted data array ct_x and secret-key array sk as input (line 1), then assigns a temp array to save many results for the terms and the sign (line 3). There are four additional **for** loops because the arrays for polynomial terms and sign

Algorithm 3 The polynomial secure convolution algorithm in the simple solution

```
1:  Input: encrypted data $ct_x[column][row][terms][sign]$,
    filter(Key) $sk[terms][sign]$
2:  initialize $output[column][row]$, $t$
3:  initialize $temp[terms^2][sign^2]$
4:
5:  for $i = 0$; $i < column$ do
6:      for $j = 0$; $j < row$ do
7:          for $k = 0$; $k < terms$ do
8:              for $l = 0$; $l < terms$ do
9:                  for $m = 0$; $m < sign$ do
10:                     for $n = 0$; $n < sign$ do
11:                         $temp[k \times terms + l][m \times sign + n] \leftarrow$
                            Dec($ct_x[i][j][k][m]$, $sk[l][n]$)
12:                     end for
13:                 end for
14:                 $temp[k \times terms + l][0] \leftarrow temp[k \times terms + l][0]$
                    - $temp[k \times terms + l][1]$ - $temp[k \times terms + l][2]$
                    + $temp[k \times terms + l][3]$
15:             end for
16:         end for
17:         $t \leftarrow$ Merge $temp$ according to $terms$
18:         $output[i][j] \leftarrow$ Activation($t$)
19:     end for
20: end for
21: return $output$
```

exist in both the encrypted data and the filter (lines 6-15). In these loop, we utilized the GPU implementation of *Decrypt* provided in [26] and the results of *Decrypt* are saved to a temp array (line 10). Following this, lines 13 and 16 are the operations to merge results of the sign and the polynomial terms. Finally the algorithm calls an activation function (line 17) and outputs the result (line 20). This approach exploits the inner parallelism within all the RLWE-IPFE functions. However, it does not fully utilize the capability of a GPU, because fine-grain parallelism itself only able to create small amount of workload, which is insufficient to fully utilize the computational resources available in a GPU.

2) Optimized Solution: To fully utilize the GPU resources, a coarse-grain parallel approach is proposed, wherein multiple

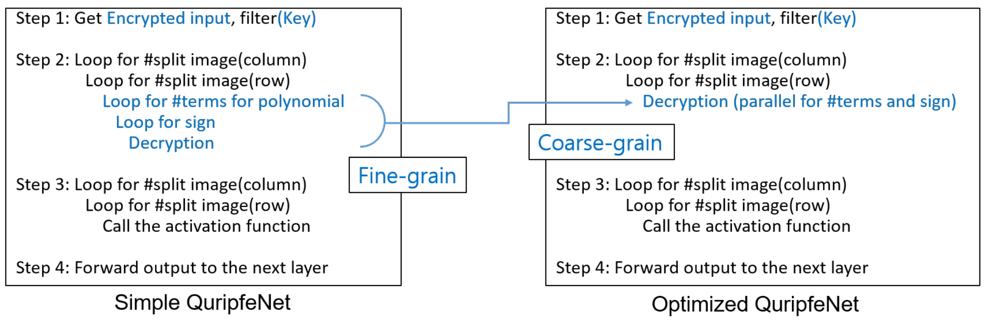
Step 1: Get Encrypted input, filter(Key)
Step 2: Loop for #split image(column)
Loop for #split image(row)
Loop for #terms for polynomial
Loop for sign
Decryption
Step 3: Loop for #split image(column)
Loop for #split image(row)
Call the activation function
Step 4: Forward output to the next layer
Fine-grain
Simple QuripfeNet
Step 1: Get Encrypted input, filter(Key)
Step 2: Loop for #split image(column)
Loop for #split image(row)
Decryption (parallel for #terms and sign)
Coarse-grain
Step 3: Loop for #split image(column)
Loop for #split image(row)
Call the activation function
Step 4: Forward output to the next layer
Optimized QuripfeNet

Fig. 7. The overview of simple solution and optimized solution: The optimized solution replaces the inner product to *Dec_parallel* which is Executed one time for all the terms and the sign.

input data are processed in parallel instead of one at a time. Figure 7 shows an the modification from the simple GPU implementation to the optimized version. In the simple solution, *Decrypt* was performed repeatedly in the loops for the polynomial terms and the sign. In other words, *Decrypt* is invoked many times for one split image data. To allow more parallelism in the optimized version, *Dec_parallel* takes in a larger array for the polynomial terms and the sign as inputs. Then, it assigns the inputs to several blocks. In such a way, *Dec_parallel* is only executed once for each split image data, and the parallelism is increased compared to the simple version. This proposed strategy combined both the coarse-grain parallelism at input data level and the fine-grain parallelism at the RLWE-IPFE function level.

Algorithm 4 shows the proposed optimized GPU implementation in detail. This algorithm receives encrypted data array ct_x and secret-key array sk as input (line 1). It assigns a temporary array to store the results for the polynomial terms and the sign (line 3). The proposed *Dec_parallel* processes all terms and sign, for the same split image data (line 6), at once. For this, *Dec_parallel* also take one more parameter, which is the number of operations, calculated as $terms^2 \times sign^2$. The results of *Dec_parallel* are stored on a temp array (line 6) before proceed to the next stop. Then, the operations to merge results from the polynomial terms and sign are performed (lines 7-12). Finally, the algorithm calls an activation function (line 13) and outputs the results (line 16). In this approach, two **for** loops for the polynomial terms and signs that are not used are removed. In addition, two **for** loops are added to merge the results of the polynomial terms and signs, which is a lightweight computation compared to the entire algorithm. Through this proposed solution, sufficient workload is generated to fully utilizing the computational resources of a GPU.

Algorithm 4 The polynomial secure convolution algorithm in the optimized solution

```
1:  Input: encrypted data $ct_x[column][row][terms][sign]$,
    filter(Key) $sk[terms][sign]$
2:  initialize $output[column][row]$, $t$
3:  initialize $temp[terms^2][sign^2]$
4:
5:  for $i = 0$; $i < column$ do
6:      for $j = 0$; $j < row$ do
7:          $temp \leftarrow$ Dec_parallel($ct_x[i][j]$, $sk$, $terms^2 \times sign^2$)
8:          for $k = 0$; $k < terms$ do
9:              for $l = 0$; $l < terms$ do
10:                 $temp[k \times terms+l][0] \leftarrow temp[k \times terms+l][0]$
                    - $temp[k \times terms+l][1]$ - $temp[k \times terms+l][2]$
                    + $temp[k \times terms + l][3]$
11:             end for
12:         end for
13:         $t \leftarrow$ Merge $temp$ according to $terms$
14:         $output[i][j] \leftarrow$ Activation($t$)
15:     end for
16: end for
17: return $output$
```

IV. Experimental Methodology

The proposed QuripfeNet was evaluated on a LeNet-5 CNN that was trained to classify images from the MNIST dataset [24], consists of 10,000 test examples with size 32 × 32 from 10 classes. LeNet-5 is a CNN architecture that includes three convolution layers and two sub-sampling layers. We use a model provided by https://github.com/fan-wenjie/LeNet-5, which was pre-trained with 32-bit floating point precision at 98.2% accuracy. The experiments were carried out based on the RLWE-IPFE library [14] on a CPU, and another library [26] on a GPU. These experiments were conducted on a workstation with a 3.0GHz eight-core Intel i7-9700F CPU and 16GB RAM, and an NVIDIA RTX 2060 GPU with 1920 cores and 6GB RAM.

A. Ablation Studies

This subsection presents the ablation studies to evaluate the improvements brought about by each of the proposed techniques. The experimental conditions are as follows:

1) Set 1: The original LeNet-5 model trained on floating point without any encryption.
2) Set 2: RLWE-IPFE with the original parameter set (B_x, B_y = 4) and one term polynomial is applied.
3) Set 3: RLWE-IPFE with the proposed polynomial secure convolution layer (2-term polynomial) and the original parameter set (B_x, B_y = 4) is applied.

TABLE III
CLASSIFICATION PERFORMANCE OF THE PROPOSED QURIPFENET (FOR ONE DATA)

Condition	Accuracy	Time (No Encryption)	Time (Encryption)
Set 1	98.20%	0.10 ms	-
Set 2	96.02%	-	105.2 seconds
Set 3	97.81%	-	327.8 seconds
Set 4	97.85%	-	107.4 seconds
Set 5	97.86%	-	346.7 seconds

4) Set 4: RLWE-IPFE with one term polynomial and the original parameter set (B_x, B_y = 32) is applied.
5) Set 5: RLWE-IPFE with the proposed polynomial secure convolution layer (2-term polynomial) and the original parameter set (B_x, B_y = 32) is applied.

Referring to Table. III, the inference time taken by LeNet-5 to classify one image is 0.1 ms, and the accuracy is 98.2% (Set 1). When RLWE-IPFE with the original parameter set is applied (Set 2), the accuracy dropped to 96.02% and it takes 105.2 secondsto complete. Note that encoding the floating point values of an image to integers and merging the results introduce overhead, which adds on to the encryption overhead. Moreover, the encoding process maps floating point values to integer values 0-3 (B_x, B_y = 4 of the original RLWE-IPFE), this causes information loss and reduces the accuracy by 2.18%.

To improve the accuracy, the proposed polynomial secure convolution (Set 3) is applied, wherein 2-term polynomial was used. Compared to Set 1, Set 3 loses only 0.39% of accuracy, which is better than Set 1. However, the inference time Set 3 is 327.8 seconds, $\approx 3.12\times$ slower than Set 2, due to more operations involved after adding the polynomial terms (see Section III-B for details). This is due to the fact that the 2-term solution requires two encryption, two key generations, and four decryption. In other words, 2-term polynomial is estimated to be $\approx 4\times$ slower than 1-term polynomial, while 3-term polynomial could be $\approx 9\times$ slower.

By applying the new proposed parameter set (Set 4), the accuracy lost compared to Set 1 is only 0.35%. This shows that the new parameter set can avoid losing too much information during the encoding process. Furthermore, the inference time of Set 4 is 107.4 seconds, which is very closed to Set 2, but the accuracy is 1.83% higher. This shows that by adopting the new parameter set, we can keep a good accuracy with only a small timing overhead. The timing performance is similar to the classification time of Set 1 because the number of encryption and decryption operation does not changed.

Finally, Set 5 shows the effect of the RLWE-IPFE parameter set and the polynomial secure convolution layer. In this case, only 0.34% of accuracy is lost compared to Set 1, which is slightly better than the accuracy in Set 4. However, the inference time of this version is 346.7 seconds on CPU, which is $3.23\times$ more than Set 4. Note that using the Set 4 configuration can provide a higher accuracy compared to Set 4, but it is also significant slower compared to the other versions.

In summary, the two proposed techniques can keep accuracy of NNs very closed to the original NN trained in floating point. They can be used according to the specific requirements in different applications. For instance, one can apply the new RLWE-IPFE parameter set only (Set 4), if timing performance is of utmost importance. This is because Set 4 already able to provide sufficiently accurate inference results with fast timing performance. Alternatively, one can also use both techniques (Set 5) to improve the accuracy further, in expense of slower timing performance.

B. Evaluation of the CPU and GPU Implementation Performance

This subsection reports the timing performance of CPU and GPU implementation, including the proposed optimized GPU techniques. The experiments were carried out using Set 4 and Set 5, which were giving high accuracy.

Table. IV shows the classification time taken by QuripfeNet implemented on CPU and GPU respectively. In the case of Set 4, the CPU version of QuripfeNet takes 107.4 seconds to classify one image. In detail, the *Setup* time is 45.9ms followed by the loading of images and encryption that takes 41.5 seconds, while *KeyGen*, decryption and prediction take 65.8 seconds. When QuripfeNet was implemented on an RTX 2080 GPU with the simple solution, it only takes 29.1 seconds to complete the same task. In detail, the *Setup* time is 86.0ms followed by the loading of images and encryption that takes 4.7 seconds, while *KeyGen*, decryption and prediction take 24.2 seconds. With the proposed optimized solution, the GPU implementation can classify one image in only 25.9 seconds, wherein the *Setup* time is 86.5ms, the loading of images and encryption time is 3.0 seconds, and the remaining parts (*KeyGen*, decryption and prediction) take 22.8 seconds. This is also $4.14\times$ faster than the CPU implementation. Moreover, the proposed optimized GPU implementation is 12% faster than the simple implementation.

In the case of Set 5, the CPU version of QuripfeNet takes 346.7 seconds to classify one image. This long classification time can be reduced to 108.2 seconds by applying the simple GPU implementation, which is a $3.2\times$ reduction. An optimized GPU implementation can further reduce this to 81.8 seconds. This means that the GPU version of QuripfeNet is $4.23\times$ faster than the CPU version for Set 5.

C. Comparison with Other Candidates

Table. V shows the proposed QuripfeNet with several recently proposed PPNNs. We would like to point out that comparison with existing works is not a trivial task, due to the different dataset, cryptography algorithm and neural network used in the experimentation. For instance, the inference result of HE-based PPNN is only known by the user. In contrast, FE-based PPNN allows the user and also the server to know the inference result. Both HE- and FE-based PPNN are used in different scenarios, so it is hard to decide their superiority solely based on the inference time. However, we included both schemes into Table. V for completeness, because HE-based PPNN is also a popular in for some applications.

Firstly, we present the results of a notable HE-based PPNN from [27], which can complete the inference in 297.5 seconds

TABLE IV
CLASSIFICATION TIME OF THE PROPOSED QURIPFENET (FOR ONE DATA)

Condition	Set 4 Total = (*Setup* + *Encrypt* + *Decrypt*)	Times	Set 5 Total = (*Setup* + *Encrypt* + *Decrypt*)	Times
CPU	107.4 seconds (45.9 ms + 41.5 seconds + 65.8 seconds)	-	346.7 seconds (43.9 ms + 83.7 seconds + 262.9 seconds)	-
GPU_Simple	29.1 seconds (86.0 ms + 4.7 seconds + 24.2 seconds)	3.69×	108.2 seconds (87.1 ms + 9.4 seconds + 98.6 seconds)	3.20×
GPU_Optimized	25.9 seconds (86.5 ms + 3.0 seconds + 22.8 seconds)	4.14×	81.8 seconds (86.7 ms + 4.2 seconds + 77.5 seconds)	4.23×

TABLE V
PPNN CLASSIFICATION PERFORMANCE

Research	Enc Type	Inference	Accuracy	NN model	Dataset	Quantum resistance	Environment
[27]	HE	297.5 seconds	99%	5-layer NN	MNIST	O	3.5GHz*
[7]	QFE	not specified	97%	2-layer NN	MNIST	X (GGM)	2.6GHz*
[12]	QFE	3 seconds	97.70%	4-layer NN	MNIST	X (GGM)	2.7GHz*
[5]	QFE	33.9 seconds	not specified	2-layer NN	TREC07p CEAS08-1 ENRON	X (GGM)	2.5GHz*
[4]	IPFE	not specified	95.49%	LeNet-5	MNIST	X (DDH)	2.3GHz**
[6]	IPFE	1.9 seconds	not specified	15-layer NN	Energy Regulation dataset	X (MDDH)	1.2GH*
[19]	IPFE	1.5 seconds (e)	not specified	5-layer NN	MNIST	X (DDH)	2.3GHz*
[20]	IPFE	1.0 seconds	94.87%	4-layer NN	MNIST	X (DDH)	2.1GHz**
[20]	FHIPE	2.0 seconds	94.50%	4-layer NN	MNIST	X (DDH)	2.1GHz**
QuripfeNet_CPU (Ours)	IPFE	107.4 seconds	97.85%	LeNet-5	MNIST	O (LWE)	3.0GHz*
QuripfeNet_GPU (Ours)	IPFE	25.9 seconds	97.85%	LeNet-5	MNIST	O (LWE)	RTX 2060***

(e): the expected value by dividing the training time by the number of samples.
*: unoptimized CPU implementation, **: CPU optimized, ***: GPU optimized)

to classify an image from the MNIST dataset by using a five-layer NN. Our LeNet-5 implementation with QuripfeNet on a CPU and a GPU can complete the same task within 107.4 seconds and 25.9 seconds, which are 2.79× and 11.5× faster than [27], respectively.

Next, we compare the proposed QuripfeNet with other FE-based PPNNs. PPNNs developed based on quadratic residue FE schemes are generally fast. For instance, [12] proposed a scheme that can classify an image from the MNIST dataset in only 3 seconds, which is almost 100× faster than the HE-based PPNN by [27]. Several PPNNs developed based on IPFE schemes [4], [6] offer similar performance. On the other hand, PPNNs developed based on IPFE schemes are generally faster than PPNNs developed based on quadratic residue FE schemes. Among that, the works from Xu et al. [19] and Panzade et al. [20] improved their security considering several attacks. However, all of these FE-based PPNNs are not quantum-resistant, rendering them inadequate in the quantum computing era.

Referring to Table. V, DDH, GGM, and MDDH are hardness assumptions that are vulnerable to Shor's algorithm. Although the proposed QuripfeNet is slower than all the existing FE-based PPNNs, it is built upon the LWE lattice problem, which is quantum-resistant, and thus offering higher security protection considering the threat from quantum computers. This is a very critical aspect for existing PPNN systems because the *harvest-now decrypt-later* attack [13] is considered as an immediate threat, even though the technology to build quantum computers is still not matured. Adopting a quantum-resistant PPNN like QuripfeNet can prevent such attacks, which is also the main motivation of our work.

On the other hand, QuripfeNet achieved the highest accuracy among all FE-based PPNN schemes. Consider CryptoNN by [4], which implemented the same LeNet-5 trained on the MNIST dataset. They achieved 95.49% accuracy, which is lower than the 97.85% achieved by QuripfeNet. This is because CryptoNN discards parts of its weight by keeping only two decimal places. In contrast, the accuracy achieved by QuripfeNet is closed with that achieved in unencrypted floating-point form. This shows that the proposed polynomial conversion technique can retain most of the precision in weights, eventually improving classification accuracy. Since CryptoNN does not provide the timing performance of inference, we are unable to compare with it.

D. Security Analysis

FE-based PPNN exposes the inner product of W and X to the server due to the characteristics of FE. This may raise concerns about various attacks that exploit such intermediate results and infringed the data privacy, including membership inference attacks, which is a common issue with FE-based PPNN. To react for these attacks, [19] was integrated with SplitNN to control the layer which expose intermediate result. According to the analysis in [28], we can also obtain a higher privacy guarantee, if more layers are computed in the client. Therefore, FE-based PPNN can be secure from privacy related attacks, if we combine our work with these two solutions. Note that, since the computational burden of the client increases, each FE-based PPNN research should conduct an efficiency-security trade-off analysis after combining with split learning. This can be interesting independent research for future work.

In addition, FE-based PPNN must receive a key through KDC to operate. However, various attacks by KDC can occur. For example, if KDC generates a key for a one-hot vector or colludes with an AI server, client data can be exposed through simple inversion. Therefore, FE-based PPNN usually assumes that there exists a trustworthy KDC between the client and the server.

V. Conclusion

QuripfeNet is the first quantum-resistant IPFE-based PPNN with a practical implementation. To achieve this, several techniques were proposed to enable RLWE-IPFE by [14] to be used in a CNN. Experiments were conducted to evaluate its accuracy and inference speed. Our implementation on a single-threaded CPU takes 107.4 seconds to classify one image. Furthermore, our implementation on a GPU takes just 25.9 seconds to classify one image. Unlike other existing IPFE-based PPNNs, QuripfeNet provides a privacy-preserving neural network even for the quantum computing era.

In future, we wish to extend our work to other more complex neural networks targeting various applications, such as public peace, crime prevention, spam filtering, an electricity theft detection system, etc. Some state-of-the-art (SOTA) neural networks like RNN and Transformer may introduce new modules like attention mechanism, which has a very different computational requirements. Adapting existing FE schemes on these SOTA neural network can be a very interesting research direction.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (RS-2024-00340882, RS-2024-00451320). This work was also conducted by funding from The Circle Foundation (Republic of Korea) for 1 year since December 2023 as Quantum Security Research Center selected as the '2023 The Circle Foundation Innovative Science Technology Center'. (Grant Number: 2023 TCF Innovative Science Project-05).

References

[1] Q. Lou and L. Jiang, "Hemet: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture," in *International conference on machine learning*. PMLR, 2021, pp. 7102–7110.

[2] A. Madi, "Secure machine learning by means of homomorphic encryption and verifiable computing," Ph.D. dissertation, Université Paris-Saclay, 2022.

[3] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *International Conference on Machine Learning*. PMLR, 2022, pp. 12 403–12 422.

[4] R. Xu, J. B. Joshi, and C. Li, "Cryptonn: Training neural networks over encrypted data," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1199–1209.

[5] T. Nguyen, N. Karunanayake, S. Wang, S. Seneviratne, and P. Hu, "Privacy-preserving spam filtering using homomorphic and functional encryption," *Computer Communications*, vol. 197, pp. 230–241, 2023.

[6] M. I. Ibrahem, M. Nabil, M. M. Fouda, M. M. Mahmoud, W. Alasmary, and F. Alsolami, "Efficient privacy-preserving electricity theft detection with dynamic billing and load monitoring for ami networks," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 1243–1258, 2020.

[7] E. Dufour-Sans, R. Gay, and D. Pointcheval, "Reading in the dark: Classifying encrypted digits with functional encryption," *Cryptology ePrint Archive*, 2018.

[8] C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay, "Practical functional encryption for quadratic functions with applications to predicate encryption," in *Annual International Cryptology Conference*. Springer, 2017, pp. 67–98.

[9] J. Gambetta, "Expanding the ibm quantum roadmap to anticipate the future of quantum-centric supercomputing," 2022. [Online]. Available: https://research.ibm.com/blog/ibm-quantum-roadmap-2025

[10] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[11] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International conference on the theory and application of cryptology and information security*. Springer, 2017, pp. 409–437.

[12] T. Ryffel, E. Dufour-Sans, R. Gay, F. Bach, and D. Pointcheval, "Partially encrypted machine learning using functional encryption," *arXiv preprint arXiv:1905.10214*, 2019.

[13] J. Y. Cho, "Securing optical networks by modern cryptographic techniques," in *Nordic Conference on Secure IT Systems*. Springer, 2019, pp. 120–133.

[14] J. M. B. Mera, A. Karmakar, T. Marc, and A. Soleimanian, "Efficient lattice-based inner-product functional encryption," *The International Conference on Practice and Theory of Public-Key Cryptography (PKC)*, 2022.

[15] Y. Lee, J.-W. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and H. Kang, "High-precision bootstrapping for approximate homomorphic encryption by error variance minimization," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 551–580.

[16] M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu, "Multi-input functional encryption for inner products: function-hiding realizations and constructions without pairings," in *Annual International Cryptology Conference*. Springer, 2018, pp. 597–627.

[17] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," in *International Conference on Security and Cryptography for Networks*. Springer, 2018, pp. 544–562.

[18] M. Abdalla, F. Bourse, A. D. Caro, and D. Pointcheval, "Simple functional encryption schemes for inner products," in *IACR International Workshop on Public Key Cryptography*. Springer, 2015, pp. 733–751.

[19] R. Xu, J. Joshi, and C. Li, "Nn-emd: Efficiently training neural networks using encrypted multi-sourced datasets," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2807–2820, 2021.

[20] P. Panzade and D. Takabi, "Fenet: Privacy-preserving neural network training with functional encryption," in *Proceedings of the 9th ACM International Workshop on Security and Privacy Analytics*, 2023, pp. 33–43.

[21] T. Marc, M. Stopar, J. Hartman, M. Bizjak, and J. Modic, "Privacy-enhanced machine learning with functional encryption," in *Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*. Springer, 2019, pp. 3–21.

[22] P. Panzade and D. Takabi, "Towards faster functional encryption for privacy-preserving machine learning," in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, 2021, pp. 21–30.

[23] S. Agrawal, B. Libert, and D. Stehlé, "Fully secure functional encryption for inner products, from standard assumptions," in *Annual International Cryptology Conference*. Springer, 2016, pp. 333–362.

[24] Y. LeCun and C. Cortes, "Mnist handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] K. Han, W.-K. Lee, A. Karmakar, J. M. B. Mera, and S. O. Hwang, "cufe: High performance privacy preserving support vector machine with inner-product functional encryption," *Cryptology ePrint Archive*, 2022.

[27] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data

with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[28] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

KyungHyun Han received a B.Sc degree in computer engineering, an M.S. degree in electronics and computer engineering, and a Ph.D. degree in electronics and computer engineering from Hongik University, South Korea, in 2015, 2017, and 2023 respectively. He is currently a Researcher with the Information Security and Machine Learning Laboratory at Gachon University in South Korea. His research interests include cyber security, GPU computing, machine learning, and blockchain.

Wai-Kong Lee (Member, IEEE) received the B.Eng. degree in electronics and the M.Sc. degree from Multimedia University in 2006 and 2009, respectively, and the Ph.D. degree in engineering from Universiti Tunku Abdul Rahman (UTAR), Malaysia, in 2018. He was a Visiting Scholar with Carleton University, Canada, in 2017, Feng Chia University, Taiwan, in 2016 and 2018, and OTH Regensburg, Germany, in 2015, 2018 and 2019. Prior to joining academia, he worked in several multi-national companies including Agilent Technologies (Malaysia) as R&D engineer. His research interests are in the areas of cryptography, numerical algorithms, GPU computing, Internet of Things, and energy harvesting. He is currently an associate professor in Faculty of Information and Communication Technology, UTAR. Prior to joining UTAR, he was a post-doctoral researcher in Gachon University, South Korea.

Angshuman Karmakar received the BE degree in computer science and engineering from Jadavpur University, Kolkata, and the MTech degree in computer science and engineering from the Indian Institute of Technology, Kharagpur. He received his doctorate from Katholieke Universiteit Leuven, Belgium for his dissertation titled "Design and implementation aspects of post-quantum cryptography". He is one of the primary designers of the post-quantum Saber key-encapsulation mechanism scheme which is one of the finalists in the National Institute of Standards and Technology's post-quantum standardization procedure. Currently, he is an Assistant Professor in the Department of Computer science and engineering at the Indian Institute of Technology Kanpur, India. He was an FWO post-doctoral fellow in the COSIC research group of KU Leuven before joining IITP Kanpur. Earlier, he worked as an engineer in Citrix R&D India Ltd, Bangalore, and as a research intern at Microsoft Research, Redmond, USA. His research interest spans different aspects of lattice-based post-quantum cryptography and computation on encrypted data.

Myung-Kyu Yi (Member, IEEE) received a Ph.D. in Computer Science and Engineering from Korea University in 2005. He is currently a Research Professor with Gachon University, and is a member of the personal health information standardization task force of the TTA U-Health project group. His research interests include healthcare, security, machine learning, deep learning, and human activity recognition.

Seong Oun Hwang (Senior Member, IEEE) received the B.S. degree in mathematics from Seoul National University, in 1993, the M.S. degree in information and communications engineering from the Pohang University of Science and Technology, in 1998, and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology, in 2004, South Korea. He worked as a Software Engineer with LG-CNS Systems, Inc., from 1994 to 1996. He worked as a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), from 1998 to 2007. He worked as a Professor with the Department of Software and Communications Engineering, Hongik University, from 2008 to 2019. He is currently a Professor with the Department of Computer Engineering, Gachon University. His research interests include cryptography, cybersecurity, and artificial intelligence. He is an Editor of *ETRI Journal*.