Glyph: Fast and Accurately Training Deep Neural Networks on Encrypted Data

Qian Lou
louqian@iu.edu

Bo Feng
fengbo@iu.edu

Geoffrey C. Fox
gcf@indiana.edu

Lei Jiang
jiang60@iu.edu

Indiana University Bloomington

Abstract

Big data is one of the cornerstones to enabling and training deep neural networks (DNNs). Because of the lack of expertise, to gain benefits from their data, average users have to rely on and upload their private data to big data companies they may not trust. Due to the compliance, legal, or privacy constraints, most users are willing to contribute only their encrypted data, and lack interests or resources to join the training of DNNs in cloud. To train a DNN on encrypted data in a completely non-interactive way, a recent work proposes a fully homomorphic encryption (FHE)-based technique implementing all activations in the neural network by Brakerski-Gentry-Vaikuntanathan (BGV)-based lookup tables. However, such inefficient lookup-table-based activations significantly prolong the training latency of privacy-preserving DNNs.

In this paper, we propose, Glyph, a FHE-based scheme to fast and accurately train DNNs on encrypted data by switching between TFHE (Fast Fully Homomorphic Encryption over the Torus) and BGV cryptosystems. Glyph uses logic-operation-friendly TFHE to implement nonlinear activations, while adopts vectorial-arithmetic-friendly BGV to perform multiply-accumulation (MAC) operations. Glyph further applies transfer learning on the training of DNNs to improve the test accuracy and reduce the number of MAC operations between ciphertext and ciphertext in convolutional layers. Our experimental results show Glyph obtains the state-of-the-art test accuracy, but reduces the training latency by 99% over the prior FHE-based technique on various encrypted datasets.

1. Introduction

Deep learning is one of the most dominant approaches to solving a wide variety of problems such as recommender systems, computer vision and natural language processing [12], because it has demonstrated state-of-the-art accuracy. Through only sufficient labeled data, the weights of a deep learning model can be trained to achieve high enough accuracy. Average users typically lack knowledge and expertise to build their own deep learning models to harvest benefits from their own data, so they have to depend on big data companies such as Google, Amazon and Microsoft. However, due to compliance, legal, and privacy constraints, there are many scenarios where the data required by the training of DNNs is extremely sensitive. It is risky to provide personal information, e.g., financial or healthcare records, to untrusted companies to train deep learning models. Federal privacy regulations such as the EU general data protection regulation also restrict the availability and sharing of these sensitive data.

Recent works [22, 1, 15] propose several cryptographic techniques to enable the privacy-preserving training of deep learning models. Private federated learning [15] is created to decentralize the training of deep learning models and enable users to train with their own data locally. QUOTIENT [1] takes advantage of multi-party computation (MPC) to interactively train deep learning models on both the server and the user. Both federated learning and MPC require users to stay online and heavily participate in the DNN training. However, in some cases, average users may not have strong interest, powerful hardware, or fast network connections for interactive DNN trainings [17]. To enable the private training of DNNs on encrypted data in a completely non-interactive way, a recent study presents the first fully homomorphic encryption (FHE)-based stochastic gradient descent technique [22], FHESGD. During FHESGD, a user first encrypts the private data and uploads the encrypted data to an untrusted server that performs both forward and backward propagations on the encrypted data without decryption. After uploading encrypted data, the users can simply go offline. The user privacy is preserved in the training procedure, since the input and output data, activations, losses and gradients are all encrypted.

However, FHESGD [22] is seriously limited by its long training latency, because of its BGV-lookup-table-based *sigmoid* activations. Specifically, FHESGD builds a Multi-Layer Perceptron (MLP) with 3 layers to achieve $< 98\%$ test accuracy on the encrypted MNIST dataset after 50 training epochs. A mini-batch including 60 samples takes ~ 2 hours on a 16-core CPU. FHESGD uses the BGV cryptosystem [14] to implement the stochastic gradient descent

1

algorithm, because BGV is good at performing large vectorial arithmetic operations frequently used in the MLP. However, FHESGD replaces all activations of the MLP by *sigmoid* functions, and uses BGV homomorphic table lookup operations [2] to implement the sigmoid function. A BGV homomorphic table lookup operation in the FHESGD setting is so slow that the BGV-lookup-table-based *sigmoid* activations consume $\sim 98\%$ of the training time.

In this paper, we propose a FHE-based DNN called Glyph to enable fast and accurate training over encrypted data. Our contributions can be summarized as:

- We propose a FHE-based DNN training scheme, Glyph, to adopt the logic-operation-friendly TFHE cryptosystem [8] to implement activations such as *ReLU* and *softmax* in DNN trainings. The TFHE-based activations significantly reduce the activation latency.

- We propose a cryptosystem switching technique to enable Glyph to perform activations by TFHE and switch to the vectorial-arithmetic-friendly BGV cryptosystem when processing fully-connected and convolutional layers. By switching between TFHE and BGV, Glyph can substantially improve the privacy-preserving DNN training speed on encrypted data.

- At last, we apply the method of transfer learning on Glyph to not only reduce the computing overhead of DNN trainings, but also improve its test accuracy. Compared to FHESGD, Glyph reduces the training latency by 99% and improves the test accuracy by $1 \sim 2\%$ on various encrypted datasets.

2. Background

2.1. Threat Model

Although an encryption scheme protects the data sent to external servers, untrusted servers [12] can make data leakage happen. Homomorphic Encryption is one of the most promising techniques to enable a server to perform private DNN training [22] on encrypted data. A user sends encrypted data to a server performing private DNN trainings on encrypted data. More importantly, after uploading encrypted data to the server, the user can go offline without participating time-consuming DNN trainings.

2.2. Fully Homomorphic Encryption

A cryptosystem that supports computation on ciphertexts without decryption is known as homomorphic encryption(HE) [9]. HE cryptosystem encrypts the plaintext p to the ciphertext c by a function ϵ. We have $c = \epsilon(p, k_{pub})$, where k_{pub} is the public key. Another function σ decrypts the ciphertext c back to the plaintext p. We have $p = \sigma(c, k_{pri})$, where k_{pri} is the private key. The cryptosystem is *homomorphic* in an operation $\star$, if there is another operation $\circ$ such that $\epsilon(x, k_{pub}) \circ \epsilon(y, k_{pub}) = \epsilon(x \star y, k_{pub})$,

where x and y are two plaintext operands. Modern HE cryptosystems have two modes: *leveled* HE and *fully* HE (FHE). Each HE operation introduces noise into the ciphertext. Leveled HE allows to compute HE functions of only a maximal degree by designing a set of parameters. Beyond its maximal degree, leveled HE cannot correctly decrypt the ciphertext, because the accumulated noise is too large. On the contrary, FHE can enable an unlimited number of HE operations on the ciphertext, since it uses *bootstrapping* [14, 8] to "refresh" the ciphertext and reduce its noise. However, a bootstrapping operation is computationally expensive and hence time-consuming. Because the privacy-preserving DNN training requires an impractically large maximal degree, it is impossible to train a DNN by leveled HE cryptosystems. A recent work [22] demonstrates the feasibility of using the FHE BGV cryptosystem to implement DNN training on encrypted data.

2.3. BGV and TFHE

Based on the Ring-LWE (Learning With Errors) problem, several FHE cryptosystems [8, 14], e.g., TFHE [8], BFV [6], BGV [14], HEAAN [7], have been developed. Each FHE cryptosystem can more efficiently process a specific type of homomorphic operations. For instance, TFHE [8] runs combinatorial operations on individual slots faster. BFV [6] is good at performing large vectorial arithmetic operations. Similar to BFV, BGV [8] manipulates elements in large cyclotomic rings, modulo integers with many hundreds of bits. However, BGV has less scaling operations, and thus processes vectorial multiplications of ciphertexts faster [18, 10]. At last, HEAAN [7] supports floating point computations more efficiently. A recent work [3] demonstrates the feasibility of combining and switching between three Ring-LWE-based FHE cryptosystems including TFHE, BFV and HEAAN via homomorphic operations.

2.4. Forward and Backward Propagation

The training of a DNN includes forward propagation and backward propagation. During forward propagation, as Figure 1 shows, the input data go through the layers consecutively in the forward direction. The forward propagation can be described as

$$\begin{cases} u_l = W_l d_{l-1} + b_{l-1} \\ d_l = f(u_l) \end{cases} \quad (1)$$

, where u_l is the neuron tensor of layer l; d_{l-1} is the output of layer $l - 1$ and the input of layer l; W_l is the weight tensor of layer l; b_{l-1} is the bias tensor of layer $l - 1$; and $f()$ is the forward activation function. We use y and t to indicate the output of a neural network and the standard label, respectively. An L^2 norm loss function is defined as $E(W, b) = \frac{1}{2}||y - t||_2^2$. The backward propagation can be
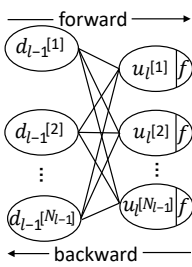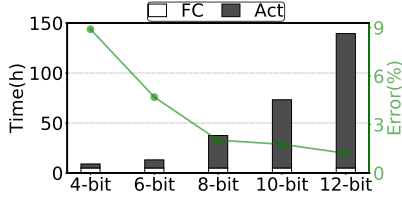
Figure 1: The forward and backward propagation.

Figure 2: The mini-batch training latency and test accuracy of a 3-layer FHESGD-based MLP on MNIST. **FC** is fully-connected layers. **Act** is activation layers.
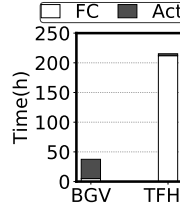

Figure 3: The mini-batch training latency of the 3-layer TFHE-based MLP on MNIST.

Operation	BFV(s)	BGV(s)	TFHE(s)
MultCC	0.043	0.012	2.121
MultCP	0.006	0.001	0.092
AddCC	0.0001	0.002	0.312
TLU	/	307.9	3.328

Table 1: The latency comparison of FHE operations. **MultCC**: ciphertext × ciphertext. **MultCP**: ciphertext × plaintext. **AddCC**: ciphertext + ciphertext. **TLU**: the table lookup.

described by

$$
\begin{cases}
\delta_{l-1} = (W_l)^T \delta_l \circ f'(u_l) \\
\nabla W_l = d_{l-1}(\delta_l)^T \\
\nabla b_l = \delta_l
\end{cases}
\tag{2}
$$

, where δ_l is the error of layer l and defined as $\frac{\partial E}{\partial b_l}$; f'() is the backward activation function; ∇W_l and ∇b_l are weight and bias gradients, respectively.

2.5. Motivation

The BGV-based FHESGD [22] trains a 3-layer MLP by substituting all activations with *sigmoid*, and implements *sigmoid* by a lookup table. However, the lookup-table-based *sigmoid* significantly increases the mini-batch training latency of FHESGD. As Figure 2 shows, with an increasing bitwidth of each entry of the BGV-based *sigmoid* lookup table, the test accuracy of FHESGD greatly improves and is approaching 98%, but its activation processing time, i.e., the *sigmoid* table lookup latency, also significantly increases and eventually occupies $> 98\%$ of the mini-batch training latency.

It is possible to fast and accurately implement the homomorphic activations including *ReLU* and *softmax* of private training by TFHE, since the TFHE cryptosystem processes combinatorial operations on individual slots more efficiently. Table 1 compares the latencies of various homomorphic operations implemented by BGV, BFV and TFHE. Compared to BGV, TFHE shortens the table lookup latency by $\sim 100\times$, and thus can implement fast activation functions. However, after we implemented the privacy-preserving DNN training by TFHE, as Figure 3 exhibits, we found although the homomorphic activations take much less time, the mini-batch training latency substantially increases, because of the slow homomorphic MAC operations implemented by TFHE. Compared to TFHE, BGV [8] demonstrates $17\times \sim 30\times$ shorter latencies for a variety of vectorial arithmetic operations such as the multiplication between a ciphertext and a ciphertext (MultCC), the multiplication between a ciphertext and a plaintext (MultCP), and the addition between a ciphertext and a ciphertext (AddCC). If we implement activation operations by TFHE, and compute vectorial MAC operations by BGV, the privacy-preserving DNN training can obtain both high test accuracy and short training latency simultaneously.

Although a recent work [3] proposes a cryptosystem switching technique to homomorphically switch between TFHE and BFV, we argue that compared to BFV, the BGV cryptosystem can implement faster privacy-preserving DNN training. First, as Table 1 shows, BGV computes multiplications between ciphertexts and plaintexts faster than BFV, because it has less scaling operations [18, 10]. Second, the state-of-the-art implementation of BFV, Microsoft SEAL [24], does not support bootstrapping. Therefore, we cannot adopt BFV for FHE-based privacy-preserving DNN training. In this paper, we propose a new cryptosystem technique to enable the homomorphic switching between BGV and TFHE.

3. Related Work

A flurry of prior works use various FHE cryptosystems including TFHE [4] and BGV [16] to implement leveled-HE-enabled privacy-preserving DNN inferences, where only the input, output data and activations are encrypted but the pre-trained weights are unencrypted. Compared to inferences, the fully-HE-enabled privacy-preserving DNN training is more computationally expensive, since it needs to compute the errors and gradients. Moreover, the inputs, activations and weights are all encrypted during the privacy-preserving DNN training. The first fully-HE-enabled DNN training technique [22] relies on the vectorial-arithmetic-friendly BGV cryptosystem.

Besides fully HE, recent works adopt multi-party computation [1] and private federated learning [15] to enable the privacy-preserving training of DNNs. Both schemes heavily involve users in the hardware-resource-demanding privacy-preserving DNN training. However, average users may not have strong motivation or powerful computing hardware to join the privacy-preserving DNN training. In this paper, we propose a fully-HE-based privacy-preserving DNN training technique that requires users to only upload their encrypted data.

Algorithm 1 The TFHE-based forward *ReLU*

Input: $u_l[i][0:n-1]$ ($u_l[i][z]$ is the z_{th} bit of $u_l[i]$)
Output: $d_l[i][0:n-1]$

1: $\underline{d_l[i][n-1]{=}0}$
2: $\overline{u_l[i][n-1]}$ = HomoNot($u_l[i][n-1]$)
3: **for** $index = 1; index < n-1; index + +$ **do**
4:     $d_l[i][index]$=HomoAND($u_l[i][index], \overline{u_l[i][n-1]}$)
    return $d_l[i][0:n-1]$

---

4. Glyph

4.1. TFHE-based Activations

To accurately train a FHE-based DNN, we propose TFHE-based homomorphic *ReLU* and *softmax* activation units. We construct a *ReLU* unit by TFHE homomorphic gates with bootstrapping, and build a *softmax* unit by TFHE homomorphic multiplexers.

ReLU: The forward *ReLU* of the i_{th} neuron in the layer l can be summarized as

$$ReLU(\mu_l[i]) = d_l[i] = \begin{cases} \mu_l[i] & \text{if } \mu_l[i] \geq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

, where $\mu_l[i]$ is the i_{th} neuron in the layer l. The backward *iReLU* for the i_{th} neuron in the layer l can be described as

$$iReLU(\mu_l[i], \delta_l[i]) = \delta_{l-1}[i] = \begin{cases} \delta_l[i] & \text{if } \mu_l[i] \geq 0. \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

, where $\delta_l[i]$ is the i_{th} error of layer l. Our TFHE-based forward *ReLU* unit can be implemented as Algorithm 1, where we first set the most significant bit of $d_l[i]$, $d_l[i][n-1]$, to 0, so that $d_l[i]$ can be always non-negative. We then get the negation of the most significant bit of $u_l[i]$, $\overline{u_l[i][n-1]}$, by a TFHE homomorphic NOT gate that even does not require bootstrapping [9]. If $u_l[i]$ is positive, $\overline{u_l[i][n-1]} = 1$; otherwise $\overline{u_l[i][n-1]} = 0$. At last, we compute $d_l[i][0:n-2]$ by homomorphically ANDing each bit of $u_l[i]$ with $\overline{u_l[i][n-1]}$. So if $u_l[i]$ is positive, $d_l[i] = \mu_l[i]$; otherwise $d_l[i] = 0$. An n-bit forward *ReLU* unit requires 1 TFHE NOT gate without bootstrapping and $n-2$ TFHE AND gates with bootstrapping.

Algorithm 2 The TFHE-based backward ReLU

Input: $\delta_l[i][0:n-1]$ and $u_l[i][n-1]$.
Output: $\delta_{l-1}[i][0:n-1]$.

1: $\overline{u_l[i][n-1]}$ = HomoNot($u_l[i][n-1]$)
2: **for** $index = 0; index < n; index + +$ **do**
3:     $\delta_{l-1}[i][index]$=HomoAND($\delta_l[i][index], \overline{u_l[i][n-1]}$)
    return $\delta_{l-1}[i][0:n-1]$

In contrast, the backward *iReLU* takes the i_{th} error of layer l, $\delta_l[i]$, and the most significant bit of $u_l[i]$, $u_l[i][n-1]$ as inputs. It generates the i_{th} error of layer $l-1$, $\delta_{l-1}[i]$. Our

TFHE-based backward *iReLU* unit can be built by Algorithm 2, where we first compute the negation of the most significant bit of $u_l[i]$, $\overline{u_l[i][n-1]}$. We then compute each bit of $\delta_{l-1}[i]$ by ANDing each bit of $\delta_l[i]$ with $\overline{u_l[i][n-1]}$. If $u_l[i][n-1] = 0$, $\delta_{l-1}[i] = \delta_l[i]$; otherwise $\delta_{l-1}[i] = 0$. An n-bit backward *iReLU* unit requires 1 TFHE NOT gate without bootstrapping and $n-1$ TFHE AND gates with bootstrapping. Our TFHE-based forward or backward *ReLU* function takes only 0.1 second, while the BGV-lookup-table-based activation consumes 307.9 seconds on our CPU baseline.
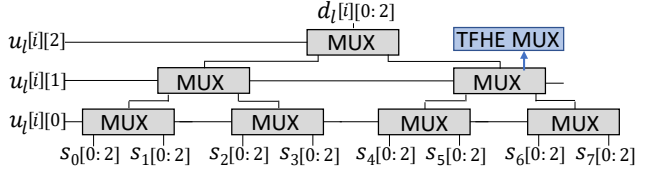

Figure 4: A 3-bit *softmax* unit.

Softmax: *softmax* takes n $u_l[i]$ as its input and normalizes them into a probability distribution consisting of n probabilities proportional to the exponentials of the inputs. The *softmax* activation can be described as

$$softmax(\mu_l[i]) = d_l[i] = \frac{e^{\mu_l[i]}}{\Sigma_i e^{\mu_l[i]}} \quad (5)$$

We use TFHE homomorphic multiplexers to implement a *softmax* unit shown in Figure 4, where we have 8 entries denoted as $S_0 \sim S_7$ for a 3-bit TFHE-lookup-table-based *softmax* unit and each entry has 3-bit. The i_{th} neuron $u_l[i]$ is used to look up one of the eight entries, and the output is $d_l[i]$. There are two TFHE gates with bootstrapping on the critical path of each TFHE homomorphic multiplexer. An n-bit *softmax* unit requires 2^n TFHE gates with bootstrapping. Compared to BGV-lookup-table-based *softmax*, our TFHE-based *softmax* unit reduces the activation latency from 307.9 seconds to only 3.3 seconds. To efficiently back-propagate the loss of *softmax*, FHESGD [22] uses a quadratic loss function to reduce the computing overhead of logarithmic operations in a cross-entropy loss function. Our experimental results show the quadratic loss function achieves the same test accuracy but costs much less computing overhead. So in this paper we also adopt the derivative of quadratic loss function described as

$$isoftmax(d_l[i], t[i]) = \delta_l[i] = d_l[i] - t[i] \quad (6)$$

, where $t[i]$ is the i_{th} ground truth. The quadratic loss function requires only homomorphic multiplications and additions. Although it is feasible to implement the quadratic loss function by TFHE, when considering the switching overhead from BGV to TFHE, we use BGV to implement the quadratic loss function.

Pooling. It is faster to adopt TFHE to implement *max pooling* operations. But considering the switching overhead from BGV to TFHE, we adopt BGV to implement *average*

pooling operations requiring only homomorphic additions and multiplications.

4.2. The Switching between BGV and TFHE

BGV can efficiently process vectorized arithmetic operations, while TFHE runs logic operations faster. During the training of Glyph, we plan to use BGV for convolutional, fully-connected, average pooling, and batch normalization layers, and adopt TFHE for activation operations. To use both BGV and TFHE, we propose a cryptosystem switching technique switching Glyph between BGV and TFHE cryptosystems.

Both BGV and TFHE are built upon the the Ring-LWE problem [8, 14], but they cannot naïvely switch between each other. Because BGV and TFHE work on different plaintext spaces. The plaintext space of BGV is the ring $\mathcal{R}_p = \mathbb{Z}[X]/(X^N + 1) \mod p^r$, where p is a prime and r is an integer. We denote the BGV plaintext space as $\mathbb{Z}_N[X] \mod p^r$. TFHE has three plaintext spaces [9] including *TLWE*, *TRLWE* and *TRGSW*. TLWE encodes individual continuous plaintexts over the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z} \mod 1$. TRLWE encodes continuous plaintexts over $\mathbb{R}[X] \mod (X^N + 1) \mod 1$. We denote the TRLWE plaintext space as $\mathbb{T}_N[X] \mod 1$, which can be viewed as the packing of N individual coefficients. TRGSW encodes integer polynomials in $\mathbb{Z}_N[X]$ with bounded norm. Through key-switching, TFHE can switch between these three plaintext spaces. Our cryptosystem switching scheme maps the plaintext spaces of BGV and TFHE to a common algebraic structure using natural algebraic homomorphisms. The cryptosystem switching can then happen in the common algebraic structure.
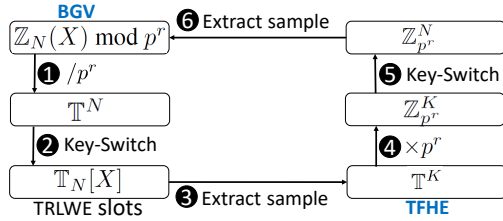

Figure 5: The switching between TFHE and BGV. Steps from ❶ to ❸ are switching from BGV to TFHE; Steps from ❹ to ❻ are switching from TFHE to BGV.

Our cryptosystem can enable Glyph to use both TFHE and BGV cryptosystems by homomorphically switching between different plaintext spaces, as shown in Figure 5.

- **From BGV to TFHE**. The switch from BGV to TFHE homomorphically transforms the ciphertext of N BGV slots encrypting N plaintexts over $\mathbb{Z}_N[X] \mod p^r$ to K TLWE-mode TFHE ciphertexts, each of which encrypts plaintexts over $\mathbb{T} = \mathbb{R}/\mathbb{Z} \mod 1$. The switch from BGV to TFHE includes three steps. ❶ Based on Lemma 1 in [3], $\mathbb{Z}_N[X] \mod p^r$ homomorphically

multiplying p^{-r} is a $\mathbb{Z}_N[X]$-module isomorphism from $\mathcal{R}_p = \mathbb{Z}_N[X] \mod p^r$ to the submodule of $\mathbb{T}_N[X]$ generated by p^{-r}. Via multiplying p^{-r}, we can convert integer coefficients in the plaintext space of BGV into a subset of torus $\mathbb{T}$ consisting of multiples of p^{-r}. In this way, we extract N coefficients from the BGV plaintexts over $\mathbb{Z}_N[X] \mod p^r$ to form $\mathbb{T}^N$. ❷ Based on Theorem 2 in [3], we use the functional key-switching to homomorphically convert $\mathbb{T}^N$ into $\mathbb{T}_N[X]$, which is the plaintext space of the TRLWE-mode of TFHE. ❸ We adopt the SampleExtract function [3] of TFHE to homomorphically achieve K individual TLWE ciphertexts from $\mathbb{T}_N[X]$. Given a TRLWE ciphertext c of a plaintext μ, *SampleExtract(c)* extracts from c the TLWE sample that encrypts the i_{th} coefficient μ_i with at most the same noise variance or amplitude as c.

- **From TFHE to BGV**. The switch from TFHE to BGV is to homomorphically transform K TFHE ciphertexts in the TLWE-mode $(m_0, m_1, \ldots, m_{K-1})$ in $\mathbb{T}^K$ to a BGV N-slot ciphertext whose plaintexts are over $\mathbb{Z}_N[X] \mod p^r$. ❹ Based on Theorem 3 in [3], we can use the functional gate bootstrapping of TFHE to restrict the plaintext space of TFHE in the TLWE-mode to an integer domain $\mathbb{Z}_{p_r}^K$ consisting of multiples of p^{-r}. ❺ The plaintext space transformation from $\mathbb{Z}_{p_r}^K$ to $\mathbb{Z}_{p^r}^N$ is a $\mathbb{Z}_N[X]$-module isomorphism, so we also can use the key-switching to implement it. ❻ At last, we can use the SampleExtract function of TFHE to homomorphically obtain the BGV N-slot ciphertext whose plaintexts are over $\mathbb{Z}_N[X] \mod p^r$.

4.3. Transfer Learning for Private DNN Training

Although FHESGD [22] shows that it is feasible to homomorphically train a 3-layer MLP, it is still very challenging to homomorphically train a deep convolutional neural network (CNN), because of the huge computing overhead of homomorphic convolutions. We propose to use transfer learning to reduce the computing overhead of homomorphic convolutions in privacy-preserving CNN trainings. Although several prior works [5, 21] adopt transfer learning in privacy-preserving inferences, to our best knowledge, this is the first work to use transfer learning in privacy-preserving CNN trainings.

Transfer learning [28, 23, 13] can be used to reuse knowledge among different datasets in the same CNN architecture, since the first several convolutional layers of the CNN extracts general features independent of datasets. Applying transfer learning in privacy-preserving CNN trainings brings two benefits. First, transfer learning reduces the number of trainable layers, i.e., the weights in the convolutional layers are fixed, so that the training latency can be greatly reduced. Second, we can convert computationally expensive convolutions between ciphertext and ciphertext

to cheaper convolutions between ciphertext and plaintext, because the fixed weights in the convolutional layers are not updated by encrypted weight gradients. Moreover, transfer learning does not hurt the security of the FHE-based DNN training, since the input, activations, losses and gradients are still encrypted.
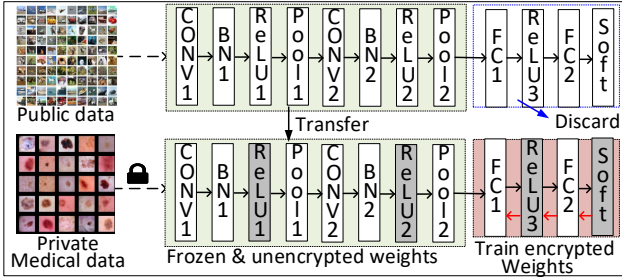

Figure 6: An example of transfer learning in the privacy-preserving CNN training.

We show an example of applying transfer learning in the privacy-preserving CNN training in Figure 6. We reuse the first two convolutional layers trained by unencrypted CIFAR-10, discard the last two fully-connected layers, and add two randomly initialized fully-connected layers, when homomorphically training of the same CNN architecture on an encrypted skin cancer dataset [26]. During the privacy-preserving training on the skin cancer dataset, we update the weights only in the last two fully-connected layers. In this way, the privacy-preserving model can reuse the general features learned from public unencrypted datasets. Meanwhile, in the privacy-preserving training, the computations on the first several convolutional and batch normalization layers are computationally cheap, since their weights are fixed and unencrypted.

5. Experimental Methodology

5.1. The Setting of Cryptosystems

For BGV, we used the same parameter setting rule as [11], and the HElib [2] library to implement all related algorithms. We adopted the m_{th} cyclotomic ring with $m = 2^{10} - 1$, corresponding to lattices of dimension $\psi(m) = 600$. This native plaintext space has 60 plaintext slots which can pack 60 input ciphertexts. The BGV setting parameters yield a security level of > 80 bits. Both BGV and TFHE implement bootstrapping operations and support fully homomorphic encryption. We set the parameters of TFHE to the same security level as BGV, and used the TFHE [9] library to implement all related algorithms. TFHE is a three-level scheme. For first-level TLWE, we set the minimal noise standard variation to $\underline{\alpha} = 6.10 \cdot 10^{-5}$ and the count of coefficients to $\underline{n} = 280$ to achieve the security level of $\underline{\lambda} = 80$. The second level TRLWE configures the minimal noise standard variation to $\alpha = 3.29 \cdot 10^{-10}$, the

count of coefficients to $n = 800$, and the security degree to $\lambda = 128$. The third-level TRGSW sets the minimal noise standard variation to $\overline{\alpha} = 1.42 \cdot 10^{-10}$, the count of coefficients to $\overline{n} = 1024$, the security degree to $\overline{\lambda} = 156$. We adopted the same key-switching and extract-sample parameters of TFHE as [3].

5.2. Simulation, Dataset and Network Architecture

We evaluated all schemes on an Intel Xeon E7-8890 v4 2.2GHz CPU with 256GB DRAM. It has two sockets, each of which owns 12 cores and supports 24 threads. Our encrypted datasets include MNIST [20] and Skin-Cancer-MNIST [26]. Skin-Cancer-MNIST consists of 10015 dermatoscopic images and includes a representative collection of 7 important diagnostic categories in the realm of pigmented lesions. We grouped it into a 8K training dataset and a 2K test dataset. We also used SVHN [25] and CIFAR-10 [19] to pre-train our models which are for transfer learning on encrypted datasets. We adopted two network architectures, a 3-layer MLP [22] and a 4-layer CNN shown in Figure 6. The 3-layer MLP has a 28×28 input layer, a 128-neuron hidden layer and a 32-neuron hidden layer. The CNN includes two convolutional layers, two batch normalization layers, two pooling layers, three *ReLU* layers and two fully-connected layers. The CNN architectures are different for MNIST and Skin-Cancer-MNIST. For MNIST, the input size is 28×28. There are $6 \times 3 \times 3$ and $16 \times 3 \times 3$ weight kernels, respectively, in two convolutional layers. Two fully connected layers have 84 neurons and 10 neurons respectively. For Skin-Cancer-MNIST, the input size is $28 \times 28 \times 3$. There are $64 \times 3 \times 3 \times 3$ and $96 \times 64 \times 3 \times 3$ weight kernels in two convolutional layers, respectively. Two fully-connected layers are 128 neurons and 7 neurons, respectively. We quantized the inputs, weights and activations of two network architectures with 8-bit by the training quantization technique in SWALP [27].

6. Results and Analysis

6.1. MNIST

FHESGD. The mini-batch training latency breakdown of a 3-layer FHESGD-based MLP [22] on a single CPU core is shown in Table 2. During a mini-batch, the MLP is trained with 60 MNIST images. Each BGV lookup-table operation consumes 307.9 seconds, while a single BGV MAC operation costs only 0.012 seconds. Although the activation layers of FHESGD require only a small number of BGV lookup-table operations, they consumes 98% of the total training latency. The FHESGD-based MLP makes all homomorphic multiplications happen between ciphertext and ciphertext, though the homomorphic multiplications between ciphertext and plaintext is computationally cheaper. The total training latency of the 3-layer FHESGD-based MLP for a mini-batch is 118K seconds, which is

about 1.35 days [22].

Table 2: The mini-batch training latency of the FH-ESGD [22]-based MLP. **HOP** includes the number of homomorphic operations. **MultCC** indicates the number of multiplications between ciphertext and ciphertext. **AddCC** is the number of additions between ciphertext and ciphertext. **TLU** means the number of table-lookup operations. **FC** is a fully-connected layer. **Act** is an activation layer.

Layers	Time(s)	HOP	MultCC	AddCC	TLU
FC1-forward	1357	201K	100K	100K	0
Act1-forward	44.8K	128	0	0	128
FC2-forward	54.4	8.2K	4196	4.2K	0
Act2-forward	11.7K	32	0	0	32
FC3-forward	4.32	640	320	320	0
Act3-forward	1.98K	10	0	0	10
Act3-error	0.1	10	0	10	0
FC3-error	4.32	640	320	320	0
FC3-gradient	4.32	640	320	320	0
Act2-error	11.7K	32	0	0	32
FC2-error	55.4	8.2K	4.2K	4.2K	0
FC2-gradient	55.4	8.2K	4.2K	4.2K	0
Act1-error	44.8K	128	0	0	128
FC1-gradient	1356	201K	100K	100K	0
Total	**118K**	**429K**	**213K**	**213K**	**330**

Table 3: The mini-batch training latency of the Glyph-based MLP with TFHE activations and cryptosystem switching. **HOP** includes the number of homomorphic operations. **MultCC** indicates the number of multiplications between ciphertext and ciphertext. **AddCC** is the number of additions between ciphertext and ciphertext. **Switch** means the cryptosystem switching. **FC** is a fully-connected layer. **Act** denotes an activation layer.

Layers	Time(s)	HOP	MultCC	AddCC	Act	Switch
FC1-forward	1370	201K	100K	100K	0	BGV-TFHE
Act1-forward	19.2	128	0	0	128	TFHE-BGV
FC2-forward	57.1	8.2K	4.1K	4.1K	0	BGV-TFHE
Act2-forward	4.82	32	0	0	32	TFHE-BGV
FC3-forward	6.02	640	320	320	0	BGV-TFHE
Act3-forward	34.76	10	0	0	10	TFHE-BGV
Act3-error	0.1	10	0	0	0	-
FC3-error	4.32	640	320	320	0	-
FC3-gradient	6.02	640	320	320	0	BGV-TFHE
Act2-error	4.82	32	0	0	32	TFHE-BGV
FC2-error	55.4	8.2K	4.1K	4.1K	0	-
FC2-gradient	62.1	8.2K	4.1K	4.1K	0	BGV-TFHE
Act1-error	19.2	128	0	0	128	TFHE-BGV
FC1-gradient	1356	201K	100K	100K	0	-
Total	2991	429K	213K	21K	330	-

TFHE Activation and Cryptosystem Switching. We replace all activations of the 3-layer FHESGD-based MLP by our TFHE-based *ReLU* and *softmax* activations, and build it as the Glyph-based MLP. We also integrate the cryptosystem switchings into the Glyph-based MLP to perform homomorphic MAC operations by BGV, and conduct activations by TFHE. The mini-batch training latency break-down of the 3-layer Glyph-based MLP on a single CPU core is shown in Table 3. Because of the logic-operation-friendly TFHE, the processing latency of activation layers of Glyph significantly decreases. The cryptosystem switchings introduce only small computing overhead. For instance, compared to the counterpart in the FHESGD-based MLP, *FC1-forward* increases the processing latency by only 0.96%, due to the cryptosystem switching overhead. Because of fast activations, compared to the FHESGD-based MLP, our Glyph-based MLP reduces the mini-batch training latency by 97.4% but maintains the same test accuracy.

Transfer Learning on CNN. We use our TFHE-based activations and cryptosystem switching technique to build a Glyph-based CNN, whose detailed architecture is explained in Section 5.2. We implement transfer learning in the Glyph-based CNN by fixing the convolutional layers trained by SVHN and training only two fully-connected layers. The mini-batch training latency breakdown of the Glyph-based CNN with transfer learning on a single CPU core is shown in Table 4. Because the weights of the convolutional layers are unencrypted and fixed, our Glyph-based CNN significantly reduces the number of multiplications between ciphertext and ciphertext (MultCC), and adds only computationally cheap multiplications between ciphertext and plaintext (MultCP). The Glyph-based CNN decreases the training latency by 56.7%, but improves the test accuracy by ∼2% over the Glyph-based MLP.

Table 4: The mini-batch training latency of the Glyph-based CNN with TFHE activations, cryptosystem switching, and transfer learning. **HOP** includes the number of homomorphic operations. **MultCC** indicates the number of multiplications between ciphertext and ciphertext. **MultCP** means the number of multiplications between ciphertext and plaintext. **AddCC** is the number of additions between ciphertext and ciphertext. **Switch** means the cryptosystem switching. **Conv** means a convolutional layer. **FC** is a fully-connected layer, while **Act** denotes an activation layer. **BN** is a batch normalization layer. **Pool** denotes an average pooling layer.

| Layers	Time(s)	HOP	**MultCP**	MultCC	AddCC	Act	Switch
Conv1-forward	69	73K	37K	0	37K	0	-
BN1-forward	61	15K	8K	0	8K	0	BGV-TFHE
Act1-forward	321	4.1K	0	0	0	4.1K	TFHE-BGV
Pool1-forward	17	18K	9.1K	0	9.1K	0	-
Conv2-forward	33	35K	17K	0	17K	0	-
BN2-forward	27	7K	3K	0	3K	0	BGV-TFHE
Act2-forward	151	1.9K	0	0	0	1.9K	TFHE-BGV
Pool2-forward	7	7.2K	3.6K	0	3.6K	84	-
FC1-forward	228	67K	0	34K	34K	0	BGV-TFHE
Act3-forward	8.2	84	0	0	0	84	TFHE-BGV
FC2-forward	6.1	1.68K	0	840	840	0	BGV-TFHE
Act4-forward	68	10	0	0	0	10	TFHE-BGV
Act4-error	0.1	10	0	0	10	0	-
FC2-error	6	1.68K	0	840	840	0	-
FC2-gradient	31	1.68K	0	840	840	0	BGV-TFHE
Act3-error	32	84	0	0	0	84	TFHE-BGV
FC1-gradient	227	67K	0	34K	34K	0	-
Total	3.5K	1716K	746K	106K	852K	14K	-
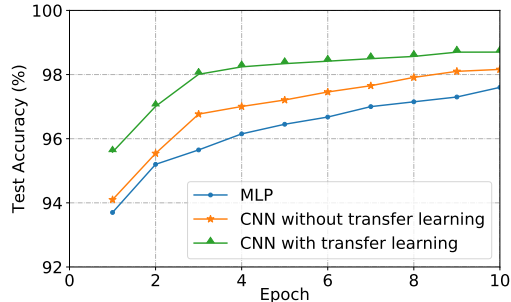
Figure 7: The accuracy comparison on MNIST.

Test Accuracy. The test accuracy comparison of the FHESGD-based MLP and the Glyph-based CNN is shown in Figure 7, where all networks are trained in the plaintext domain. It takes 5 epochs for the FHESGD-based MLP to reach 96.4% test accuracy on MNIST. After 5 epochs, the Glyph-based CNN can achieve 97.1% test accuracy even without transfer learning. By reusing low-level features of the SVHN dataset, the Glyph-based CNN with transferring learning obtains 98.6% test accuracy. The CNN architecture and transferring learning particularly can help the FHE-based privacy-preserving DNN training to achieve higher test accuracy when we do not have long time for trainings.
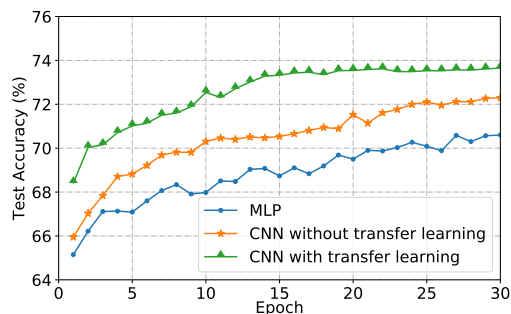

Figure 8: The accuracy comparison on Skin-Cancer.

6.2. Skin-Cancer-MNIST

We built the Glyph-based MLP and CNN architectures for Skin-Cancer-MNIST by our TFHE-based activations, cryptosystem switching and transfer learning. Since the network on Skin-Cancer-MNIST (Cancer) is larger than MNIST, the training latency is the larger as the Table 5 shows. The mini-batch training latency changes on Skin-Cancer-MNIST are similar to those on MNIST. The test accuracy comparison of the FHESGD-based MLP and the Glyph-based CNN is shown in Figure 8. For transferring learning, we first train the Glyph-based CNN with CIFAR-10, fix its convolutional layers, and then train its fully-connected layers with Skin-Cancer-MNIST. On such a more complex dataset, compared to the FHESGD-based MLP, the Glyph-based CNN without transferring learning increases the training accuracy by 2% at the 15_{th} epoch. The transferring learning further improves the test accuracy of the Glyph-based CNN to 73.2%, i.e., a 4% test accuracy boost. Our TFHE-based activations, cryptosystem switch-

ing and transfer learning makes Glyph efficiently support deep CNN architecture.

Table 5: The comparison of overall training latency.

Dataset	Network	Thread #	Mini-batch	Epoch #	Time	Acc(%)
MNIST	MLP	1	33 hours	50	187 years	97.8
		48	2.3 hours	50	13.4 years	97.8
	CNN	1	0.44 hours	5	2.46 months	98.6
		48	0.04 hours	5	8 days	98.6
Cancer	MLP	1	34.1 hours	30	15.6 years	70.2
		48	2.4 hours	30	1.1 years	70.2
	CNN	1	0.93 hours	15	0.21 years	73.2
		48	0.08 hours	15	7 days	73.2

6.3. Overall Training Latency and Scalability

The overall training latency of multiple threads on our CPU baseline is shown in Table 5. We measured the mini-batch training latency by running various FHE-based training for a mini-batch. We estimated the total training latency via the product of the mini-batch training latency and the total mini-batch number for a training. For MNIST, the FHESGD-based MLP requires 50 epochs, each of which includes 1000 mini-batches (60 images), to obtain 97.8% test accuracy. On a single CPU core, the training of the FHESGD-based MLP needs 187-year, which is impractical. On the contrary, our Glyph-based CNN requires only 5 epochs to achieve 98.6% test accuracy. The training of the Glyph-based CNN needs 2.46 months. If we use 48 threads to train the Glyph-based CNN, the overall training latency can be reduced to 8 days. Multi-threading can effectively increase the training parallelism, since the weight updates in Stochastic Gradient Decent (SGD) are independent. With 48 threads, we observed a $9.3\times$ training speedup, because the memory bandwidth has become the performance scaling bottleneck. For Skin-Cancer-MNIST, it takes 30 epochs, each of which includes 134 mini-batches, for the FHESGD-based MLP to achieve 70.2% test accuracy. In contrast, our Glyph-based CNN requires only 15 epochs to obtain 73.2% test accuracy. By 48 threads, the training of the Glyph-based CNN can be completed within 7 days.

7. Conclusion

In this paper, we propose, Glyph, a FHE-based privacy-preserving technique to fast and accurately train DNNs on encrypted data. Glyph performs *ReLU* and *softmax* by logic-operation-friendly TFHE, while conducts MAC operations by vectorial-arithmetic-friendly BGV. We create a cryptosystem switching technique to switch Glyph between TFHE and BGV. We further apply the method of transfer learning on Glyph to support CNN architectures and reduce the number of homomorphic multiplications between ciphertext and ciphertext. Our experimental results show Glyph obtains the state-of-the-art test accuracy, but reduces the training latency by 99% over the prior FHE-based technique on multiple encrypted datasets.

References

[1] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In *ACM SIGSAC Conference on Computer and Communications Security*, 2019. 1, 3

[2] Flavio Bergamaschi. HElib: an Implementation of homomorphic encryption. https://github.com/homenc/HElib, 2019. 2, 6

[3] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Cryptology ePrint Archive, Report 2018/758, 2018. https://eprint.iacr.org/2018/758. 2, 3, 5, 6

[4] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *Advances in Cryptology*, 2018. 3

[5] Alon Brutzkus et al. Low Latency Privacy Preserving Inference. In *International Conference on Machine Learning*, 2019. 5

[6] Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library-SEAL v2.1. In *International Conference on Financial Cryptography and Data Security*. Springer, 2017. 2

[7] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for Approximate Homomorphic Encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, 2018. 2

[8] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption over the Torus. *Journal of Cryptology*, 2018. 2, 3, 5

[9] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption Library, August 2016. https://tfhe.github.io/tfhe/. 2, 4, 5, 6

[10] Anamaria Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? Cryptology ePrint Archive, Report 2015/889, 2015. https://eprint.iacr.org/2015/889. 2, 3

[11] Jack L. H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. Doing Real Work with FHE: The Case of Logistic Regression. In *the Workshop on Encrypted Computing Applied Homomorphic Cryptography*, 2018. 6

[12] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *International Conference on Machine Learning*, 2016. 1, 2

[13] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018. 5

[14] Shai Halevi and Victor Shoup. Algorithms in HElib. In *Advances in Cryptology*, 2014. 1, 2, 5

[15] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *CoRR*, abs/1711.10677, 2017. 1, 3

[16] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Deep Neural Networks Classification over Encrypted Data. In *ACM Conference on Data and Application Security and Privacy*, 2019. 3

[17] Ryan Karl, Timothy Burchfield, Jonathan Takeshita, and Taeho Jung. Non-interactive mpc with trusted hardware secure against residual function attacks. Cryptology ePrint Archive, Report 2019/454, 2019. https://eprint.iacr.org/2019/454. 1

[18] Miran Kim and Kristin Lauter. Private Genome Analysis Through Homomorphic Encryption. *BMC Medical Informatics and Decision Making*, 15, 2015. 2, 3

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset, 2014. http://www.cs.toronto.edu/kriz/cifar.html. 6

[20] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST Handwritten Digit Database. *AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2010. 6

[21] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. Epic: Efficient private image classification (or: Learning from the masters). Cryptology ePrint Archive, Report 2017/1190, 2017. https://eprint.iacr.org/2017/1190. 5

[22] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards Deep Neural Network Training on Encrypted Data. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019. 1, 2, 3, 4, 5, 6, 7

[23] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. 5

[24] Microsoft SEAL (release 3.4). https://github.com/Microsoft/SEAL, Oct. 2019. Microsoft Research, Redmond, WA. 3

[25] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. *arXiv preprint arXiv:1204.3968*, 2012. 6

[26] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5:180161, 2018. 6

[27] Guandao Yang, Tianyi Zhang, Polina Kirichenko, Junwen Bai, Andrew Gordon Wilson, and Chris De Sa. SWALP : Stochastic weight averaging in low precision training. In *International Conference on Machine Learning*, 2019. 6

[28] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014. 5

8. Supplementary Results and Analysis

In this supplementary material, we further explain how we achieve the overall training latency on Skin-Cancer-MNIST (Cancer) in Table 5. In the Table 5, we listed the comparison of FHESGD-based MLP and our Glyph-based CNN. Here we explain more on how we built the Glyph-based MLP and CNN architectures for Skin-Cancer-MNIST by our TFHE-based activations, cryptosystem switching and transfer learning.

8.1. Skin-Cancer-MNIST

FHESGD. The mini-batch training latency breakdown of a 3-layer FHESGD-based MLP on a single CPU core is shown in Table 6. During a mini-batch, the MLP is trained with 60 Skin-Cancer-MNIST images. Each BGV lookup-table operation consumes 307.9 seconds, while a single BGV MAC operation costs only 0.012 seconds. Although the activation layers of FHESGD require only a small number of BGV lookup-table operations, they consumes 98% of the total training latency. The FHESGD-based MLP makes all homomorphic multiplications happen between ciphertext and ciphertext, though the homomorphic multiplications between ciphertext and plaintext is computationally cheaper. The total training latency of the 3-layer FHESGD-based MLP for a mini-batch is 123K seconds, which is about 1.42 days.

Table 6: The mini-batch training latency of the FHESGD-based MLP. **HOP** includes the number of homomorphic operations. **MultCC** indicates the number of multiplications between ciphertext and ciphertext. **AddCC** is the number of additions between ciphertext and ciphertext. **TLU** means the number of table-lookup operations. **FC** is a fully-connected layer. **Act** is an activation layer.

Layers	Time(s)	HOP	MultCC	AddCC	TLU
FC1-forward	4068	603K	302K	302K	0
Act1-forward	44.8K	128	0	0	128
FC2-forward	55.4	8.2K	4096	4.1K	0
Act2-forward	11.7K	32	0	0	32
FC3-forward	4.32	640	320	320	0
Act3-forward	1.45K	7	0	0	7
Act3-error	0.1	7	0	7	0
FC3-error	4.32	640	320	320	0
FC3-gradient	4.32	640	320	320	0
Act2-error	11.7K	32	0	0	32
FC2-error	55.4	8.2K	4.2K	4.2K	0
FC2-gradient	55.4	8.2K	4.2K	4.2K	0
Act1-error	44.8K	128	0	0	128
FC1-gradient	4068	603K	302K	302K	0
Total	**123K**	1233K	613K	613K	327

TFHE Activation and Cryptosystem Switching. We replace all activations of the 3-layer FHESGD-based MLP by our TFHE-based *ReLU* and *softmax* activations, and

Table 7: The mini-batch training latency of the Glyph-based MLP with TFHE activations and cryptosystem switching. **HOP** includes the number of homomorphic operations. **MultCC** indicates the number of multiplications between ciphertext and ciphertext. **AddCC** is the number of additions between ciphertext and ciphertext. **Switch** means the cryptosystem switching. **FC** is a fully-connected layer. **Act** denotes an activation layer.

Layers	Time(s)	HOP	MultCC	AddCC	Act	Switch
FC1-forward	4075	603K	301K	301K	0	BGV-TFHE
Act1-forward	19.2	128	0	0	128	TFHE-BGV
FC2-forward	57.1	8.2K	4.1K	4.1K	0	BGV-TFHE
Act2-forward	4.82	32	0	0	32	TFHE-BGV
FC3-forward	6.02	640	320	320	0	BGV-TFHE
Act3-forward	34.76	7	0	0	7	TFHE-BGV
Act3-error	0.1	7	0	0	0	-
FC3-error	4.32	640	320	320	0	-
FC3-gradient	6.02	640	320	320	0	BGV-TFHE
Act2-error	4.82	32	0	0	32	TFHE-BGV
FC2-error	55.4	8.2K	4.1K	4.1K	0	-
FC2-gradient	62.1	8.2K	4.1K	4.1K	0	BGV-TFHE
Act1-error	19.2	128	0	0	128	TFHE-BGV
FC1-gradient	4068	201K	100K	100K	0	-
Total	10.6K	1233K	613K	613K	327	-

build it as the Glyph-based MLP. We also integrate the cryptosystem switchings into the Glyph-based MLP to perform homomorphic MAC operations by BGV, and conduct activations by TFHE. The mini-batch training latency breakdown of the 3-layer Glyph-based MLP on a single CPU core is shown in Table 7. Because of the logic-operation-friendly TFHE, the processing latency of activation layers of Glyph significantly decreases.

The cryptosystem switchings introduce only small computing overhead. For instance, compared to the counterpart in the FHESGD-based MLP, *FC1-forward* increases the processing latency by only 0.17%, due to the cryptosystem switching overhead. Because of fast activations, compared to the FHESGD-based MLP, our Glyph-based MLP reduces the mini-batch training latency by 91.4% but maintains the same test accuracy.

Transfer Learning on CNN. We use our TFHE-based activations and cryptosystem switching technique to build a Glyph-based CNN, whose detailed architecture is explained in Section 5.2. We implement transfer learning in the Glyph-based CNN by fixing the convolutional layers trained by CIFAR-10 and training only two fully-connected layers. The mini-batch training latency breakdown of the Glyph-based CNN with transfer learning on a single CPU core is shown in Table 8. Because the weights of the convolutional layers are unencrypted and fixed, our Glyph-based CNN significantly reduces the number of multiplications between ciphertext and ciphertext (MultCC), and adds only computationally cheap multiplications between ciphertext and plaintext (MultCP). The Glyph-based CNN decreases

the training latency by 67.2%, but improves the test accuracy by ∼4% over the Glyph-based MLP.

Table 8: The mini-batch training latency of the Glyph-based CNN with TFHE activations, cryptosystem switching, and transfer learning. **HOP** includes the number of homomorphic operations. **MultCC** indicates the number of multiplications between ciphertext and ciphertext. **MultCP** means the number of multiplications between ciphertext and plaintext. **AddCC** is the number of additions between ciphertext and ciphertext. **Switch** means the cryptosystem switching. **Conv** means a convolutional layer. **FC** is a fully-connected layer, while **Act** denotes an activation layer. **BN** is a batch normalization layer. **Pool** denotes an average pooling layer.

Layers	Time(s)	HOP	**MultCP**	MultCC	AddCC	Act	Switch
Conv1-forward	552	584K	296K	0	296K	0	-
BN1-forward	162	40K	21.3K	0	21.3K	0	BGV-TFHE
Act1-forward	856	10.8K	0	0	0	10.8K	TFHE-BGV
Pool1-forward	272	288K	146K	0	146K	0	-
Conv2-forward	528	560K	272K	0	272K	0	-
BN2-forward	41	10.5K	5.3K	0	5.3K	0	BGV-TFHE
Act2-forward	227	29K	0	0	0	29K	TFHE-BGV
Pool2-forward	10.5	11K	5.4K	0	5.4K	0	-
FC1-forward	342	101K	0	51K	51K	0	BGV-TFHE
Act3-forward	12.3	128	0	0	0	128	TFHE-BGV
FC2-forward	9.3	2560	0	1280	1280	0	BGV-TFHE
Act4-forward	50.1	7	0	0	0	7	TFHE-BGV
Act4-error	0.1	7	0	0	7	0	-
FC2-error	9.1	2560	0	1280	1280	0	-
FC2-gradient	34.1	2560	0	1280	1280	0	BGV-TFHE
Act3-error	36	128	0	0	0	128	TFHE-BGV
FC1-gradient	341	101K	0	51K	51K	0	-
Total	3481	1716K	746K	106K	852K	14K	-