Optimization of Functional Bootstraps with Large LUT and Packing Key Switching

Keyi Liu, Chungen Xu, and Bennian Dou

Nanjing University of Science and Technology, Nanjing 210094, JS, China
{liuKY,xuchung,benniandou}@njust.edu.cn

Abstract. Homomorphic encryption can perform calculations on encrypted data, which can protect the privacy of data during the usage of data. *Functional Bootstraps* algorithm proposed by I. Chillotti et al. can compute arbitrary functions represented as lookup table whilst bootstrapping, but the computational efficiency of *Functional Bootstraps* with large lookup table or highly precise functions is not high enough. To tackle this issue, we propose a new Tree-BML algorithm. Our Tree-BML algorithm accelerates the computation of *Functional Bootstraps* with large LUT by compressing more LWE ciphertexts to a TRLWE ciphertext and utilizing the PBSmanyLUT algorithm which was proposed by I. Chillotti et al. in 2021. The Tree-BML algorithm reduces the running time of LUT computation by 72.09% compared to *Tree-based* method(Antonio Guimarães et al., 2021). Additionally, we introduce a new TLWE-to-TRLWE Packing Key Switching algorithm which reduces the storage space required and communication overhead of homomorphic encryption algorithm by only generating one of those key-switching key ciphertexts of polynomials with the same non-zero coefficient values but only those values located in different slots. Our algorithm reduces the key-switching key size by 75% compared to Base-aware TLWE-to-TRLWE Key Switching algorithm. Finally, we obtained that our algorithms does not introduce new output error through theoretical and experiment result.

Keywords: Homomorphic Encryption · Lookup Table · Key Switching · TFHE

1 Introduction

Homomorphic encryption(HE) makes it possible to perform calculations on encrypted data, the security of the existing HE algorithms is based on the Learning With Errors(LWE) [1] or its ring variant(RLWE) [2, 3] problem which is to ensure the security of the message by adding an error to the ciphertext. In the process of homomorphic calculation, the error will inevitably accumulate, and in order to successfully decrypt and restore the plaintext, the error needs to be controlled. Therefore, only a limited number of homomorphic calculations can be performed, which is called Leveled homomorphic encryption(LHE). It was not until 2009 that Gentry first proposed a fully homomorphic encryption(FHE) [4], which made it possible to perform any number of homomorphic calculations on encrypted data. The Gentry's algorithm reduces the error by performing bootstrapping operations when the error reaches a given boundary, that is, performing homomorphic decryption operations using the secret key under homomorphic encrypted which called bootstrapping key. However, the bootstrapping algorithm is very expensive for time and memory.

In order to improve the efficiency and reduce memory requirements of FHE, many homomorphic encryption cryptosystem have been proposed, such as BGV [5], B/FV [6,7], CKKS [8], GSW [9], FHEW [10] etc. I. Chillotti et al. proposed the TFHE [11–13] homomorphic encryption cryptosystem based on torus, which is the most efficient homomorphic encryption cryptosystem, bootstrapping 1-bit message only takes 13ms, and can calculate arbitrary functions encoded on lookup table(LUT) while performing bootstrapping calculations, usually called *programmable bootstrapping*(PBS) [14,15] or *functional Bootstraps*(FBT) [16]. Informally, taking a ciphertext of m(called a selector), *test vector*(TV, whose coefficient is the value of the function f encoded as LUT) and bootstrapping keys as inputs, the FBT algorithm can output a ciphertext of $f(m)$, and the error contained in the output ciphertext is independent with the error contained in the input ciphertext(refreshing the error). The traditional functional bootstrapping only supports the calculation of plaintext with limited precision, generally no more than 8 bits [17], when the size or precision of m or $f(m)$ increase, the parameters of homomorphic encryption should be increased to ensure the correctness of the algorithm, which can lead to performance degradation, For example, in the experiment of Carpov et al. [18] using TFHE's FBT, 1.5 seconds is needed to calculate 6-bit-to-6-bit LUT.

In 2019, Carpov et al. proposed Multi-value Bootstrapping(MVB) algorithm [18]. For a selector input, they obtained the output of multiple LUTs by performing only one bootstrapping. In their experiments, it takes 1.6s to calculate 6-to-6-LUTs.

In 2021, Antonio Guimarães et al. introduced a *Tree-based* method to combine functional Bootstraps(TreeFB) [19] to accelerate the computational of FBT with large LUT. It takes about 378ms to compute the 6-bit-to-6-bit LUT in their experiments. Antonio Guimarães et al. also proposed Base-aware TLWE-to-TRLWE Key Switching(abbreviation Base-aware-KS in the following) to accelerate the Packing Key Switching. Moreover, through pre-computation, their algorithm can execute the Packing Key Switching by using addition only.

Subsequently, I. Chillotti et al. proposed multi-output programmable bootstrapping [20] to optimize FBT. By compressing multiple LUT results into a TRLWE ciphertext as TV, so that multiple LUTs can be computed at one bootstrapping and without adding error variance or calculation complexity, called PBSmanyLUT(BML).

Contribution In this paper, We propose two new algorithms to optimize calculation of FBT with large LUT and TLWE-to-TRLWE Packing Key Switching respectively. We analyze the error variance and rate of the new algorithms. Finally, we give the experiment results and compare them with others.

- We speed up the calculation of FBT with large LUT or highly precise functions in fully homomorphic setting by optimizing the TreeFB algorithm. We accelerate the calculation of TreeFB by compressing $2^{\vartheta} \cdot B$ instead of B TLWE ciphertext into one TRLWE ciphertext after the first layer in TreeFB. Therefore, we can extract the 2^{ϑ} instead of one samples required for the next layer in only one bootstrapping by using the BML algorithm. So we reduce the number of bootstrappings and Key Switching required for the TreeFB algorithm and without adding the error variance of output. We call it Tree-BML algorithm.
- We propose a new TLWE-to-TRLWE Packing Key Switching algorithm that reduces the key-switching key size of the Base-aware-KS. We have observed that it is necessary to generate key-switching key ciphertexts of polynomials with the same non-zero coefficient values but only those values located in different slots in the Base-aware-KS algorithm. Therefore, for these key-switching key ciphertexts, we only generate one ciphertext and perform homomorphic rotation operations when needed, which effectively reducing the ciphertext size of the key-switching key. We call it the Base-rotate TLWE-to-TRLWE Packing Key Switching(Rotate-based KS) algorithm.
- We analyze the error variance and rate of our Tree-BML algorithm and Rotate-based KS algorithm and compare with others. The Tree-BML and Rotate-based KS algorithm does not add error variance of output. We verify our conclusion about error variance and rate through experiment. We also experiment for different functions, such as LUT, 32-bit integer comparison and Relu, and compared the performance results of our algorithms with existing algorithms. Our Tree-BML algorithm reduce the running time of the TreeFB algorithm by 72.09% and Rotate-based KS algorithm reduce the key-switching key size of Base-aware-KS by 75%.

2 Background and Notations

TFHE [11–13] is a fully homomorphic encryption cryptosystem based on torus and the security based on (R)LWE [2, 3, 21]. We review related works and the necessary knowledge required for this paper in this section.

Notations The $\mathbb{Z}_q$ is a ring $\mathbb{Z}/q\mathbb{Z}$ where $q \in \mathbb{Z}$, and $\mathbb{B} = \mathbb{Z}_2 = \{0, 1\}$. The torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. For two elements $a, b \in \mathbb{T}$, addition is defined as $(a + b)$ mod 1. $\mathbb{T}$ is not a ring because it is undefined multiplication of a with b, but it has *external multiplication*, for $k \in \mathbb{Z}, a \in \mathbb{T}$, $k \cdot a = a + \cdots + a(k$ times). We note $\mathbb{Z}_N[X] = \mathbb{Z}[X]/(X^N + 1)$ and $\mathbb{T}_N[X] = \mathbb{T}[X]/(X^N + 1)$ where N is a power of 2 ordinary. When the torus precision increases, that is, the size of bits of the plaintext increases, which would rapidly deteriorate the performance of the algorithm. Therefore, the method based on *torus base* decomposition for plaintext is generally adopted, so that the parameters of the algorithm only need to ensure that the size of *torus base* can run successfully rather than the size of the entire plaintext size. We note B for *torus base* which is a power of 2 ordinary. Since floating point values are typically stored in 32-bit or 64-bit lengths in computers, it is necessary to modulo conversion to $q = 2^{32}$ or 2^{64} in actual execution [20] although the TFHE bases in $\mathbb{T}$. We denote χ_σ to represent the *Gaussian distribution* with an mean of 0 and a standard variance of σ. We use variables with underbars to represent input variables and overbars to represent output variables.

TLWE We call $(a, b) \in \mathbb{T}^{n+1}$ is a TLWE ciphertext of the plaintext $m \in \mathbb{T}$ under the secret key $s \in \mathbb{B}^n$ which is uniformly sampled, if and only if $b - \langle a, s \rangle = m + e$, where a is a vector uniformly sampled from $\mathbb{T}^n$, $e \in \mathbb{T}$ is sampled from χ_σ and $\langle\ ,\ \rangle$ denotes the inner product.

TRLWE We call $(\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{T}_N^{k+1}[X]$ is a TRLWE ciphertext of the plaintext $\boldsymbol{m} \in \mathbb{T}_N[X]$ under the secret key $S \in \mathbb{B}_N^k[X]$, if and only if $\boldsymbol{b} - \boldsymbol{a} \cdot S = \boldsymbol{m} + e$, where $\boldsymbol{a}$ is uniformly sampled from $\mathbb{T}_N^k[X]$ and $e \in \mathbb{T}_N[X]$ is a polynomial which coefficients are sampled from χ_σ.

TRGSW A TRGSW ciphertext is a matrix where each row is a TRLWE ciphertext. Since we do not need it in this paper, please refer to TFHE [13] for more details.

Encryption To encrypt a message $m \in \mathbb{Z}_B$ into TLWE ciphertext, we should use the private key to generate ciphertext (a, b) of 0 , then calculate $(a, b) + (0, \Delta \cdot m)$, where Δ is a salar factor in order to perform modulo conversion $\mathbb{Z}_B$ to $\mathbb{T}$. The encryption method of TRLWE ciphertext is similar to it.

Decryption For a given TLWE sample $c = (a, b)$, we use secret key calculate $\phi(c) = b - \langle a, s \rangle$ and approximating the result to the closest plaintext value on $\mathbb{T}$, the decryption method for TRLWE ciphertext is similar to it. It can be seen that the result of $\phi(c) = \Delta \cdot m + e$ is a plaintext value with error. In order to correctly approximate the plaintext value m, we need to control the error to be less than half the distance between two consecutive plaintext values on $\mathbb{T}$.

Homomorphic Arithmetic We set T(R)LWE samples $c_0 = (a_0, b_0)$ of plaintext m_0 and $c_1 = (a_1, b_1)$ of plaintext m_1. Then $c_0 + c_1 = (a_0 + a_1, b_0 + b_1)$ is a T(R)LWE ciphertext of plaintext $m_0 + m_1$. We can calculate the multiplication between cleartext $k \in \mathbb{Z}(\text{or } \mathbb{Z}_\mathrm{N}[\mathrm{X}])$ and a T(R)LWE ciphertext $c = (a, b)$ just by calculating $(k \cdot a, k \cdot b)$. We need to note that the initial TFHE cryptosystem does not support the product between T(R)LWEs, but only supports the *External product*, that is, the product between TRGSW ciphertext with TRLWE ciphertext(referred to TFHE [13] for details). In the full homomorphic setting, in order to perform the product of two TRLWE ciphertexts we need to perform *circuit bootstrapping* to conversion between TRLWE and TRGSW. Although I. Chillotti et al. introduced the T(R)LWE * T(R)LWE algorithm similar to BFV in the 2021 [20], its efficiency is far less than *External product*, it requires relinearization operation similar to BFV.

CUMX For a TRGSW ciphertext C of $m' \in \{0, 1\}$ as the control line and two TRLWE ciphertext $\boldsymbol{d}_0$, $\boldsymbol{d}_1$ of two plaintext m_0, m_1 inputs, the CUMX algorithm outputs $C \cdot (\boldsymbol{d}_1 - \boldsymbol{d}_0) + \boldsymbol{d}_0$. If $m' = 0$, the output is a TRLWE ciphertext of m_0, otherwise it is a TRLWE ciphertext of m_1.

2.1 Functional Bootstraps(FBT)

Inputing a TLWE ciphertext of m(called a selector), a TRLWE ciphertext(or unencrypted) of test vector v(sometimes we call it TV, whose coefficients are the values of LUT which encodes the function f), and bootstrapping key(a set of TRGSW ciphertexts), then FBT algorithm can output a TLWE ciphertext of $f(m)$ and refresh the error in TLWE ciphertext of m. FBT is mainly composed of three algorithms: ModSwitch, BlindRotate, and SampleExtract. We have only provided a brief introduction, complete FBT algorithm see Appendices A and please refer to the TFHE [13] for more details.

2.2 Public Functional Key Switching(PublicKS)

The main purpose of key switching is to switch between different keys. I. Chillotti et al. proposed Public Functional Key Switching algorithm(abbreviation PublicKS in the following) which can calculate public linear morphisms f meanwhile switch key, see Algorithm 1.

Algorithm 1 TLWE-to-T(R)LWE Public Functional Key Switching(PublicKS)

```
Input: $B$ TLWE samples $\underline{c}^{(z)} = \left(\underline{a}^{(z)}, \underline{b}^{(z)}\right) \in \mathrm{TLWE}_{\underline{s}}(\mu_z), z \in [\![1, B]\!]$, a precision
    parameter $t \in \mathbb{Z}^*$, R-Lipschitz morphism $f : \mathbb{T}^B \to \mathbb{T}_{\underline{N}}[X]$, a list of Key Switching
    key $\mathrm{KS}_{i,j} \in \mathrm{T(R)LWE}_{\bar{S}}\left(\frac{s_i}{2^j}\right)$, for $i \in [\![1, n]\!]$, $j \in [\![1, t]\!]$
Output: a T(R)LWE sample $\bar{c} \in \mathrm{T(R)LWE}_{\bar{s}}(f(\mu_1, \ldots, \mu_B))$
for $i = 1$ to $n$ do
    $a_i \leftarrow f\left(\underline{a}_i^{(1)}, \underline{a}_i^{(2)}, \ldots, \underline{a}_i^{(B)}\right)$
    Let $\tilde{a}_i = \lceil a_i \rfloor_{\frac{1}{2^t}}$ be the closest multiple of $\frac{1}{2^t}$ to $a_i$
    Decompose each $\tilde{a}_i = \sum_{j=1}^{t} \tilde{a}_{i,j} \cdot 2^{-j}$, where $\tilde{a}_{i,j} \in \mathbb{B}_N[X]$
end for
return $\left(0, f\left(\underline{b}_i^{(1)}, \underline{b}_i^{(2)}, \ldots, \underline{b}_i^{(p)}\right)\right) - \sum_{i=1}^{n} \sum_{j=1}^{t} \tilde{a}_{i,j} \cdot \mathrm{KS}_{i,j}$
```

2.3 Multi-value Bootstrapping(MVB)

The algorithm proposed by Sergiu Carpov [18] can calculate the output of multiple values of LUTs for a selector. Inputing a selector, their MVB algorithm perform the BlindRotate algorithm on a constant polynomial instead of on the TV. Then the BlindRotate result multiply with several TVs finally performing SampleExtract algorithm. We have only provided a brief introduction, please refer to the paper [18] for more details.

2.4 PBSmanyLUT(BML)

In 2021, I. Chillotti et al. proposed a multi-output functional bootstrapping algorithm [20]. For example, assuming the message $m \in \mathbb{Z}_B$, when we need to calculate the value of 2^ϑ functions $f_i, i \in [\![1, 2^\vartheta]\!]$ about m, encoding the $2^\vartheta B$ LUT's values of functions $f_i(j), j \in [\![0, B-1]\!]$ to TV

$$TV_{\left(f_1,\dots,f_{2^\vartheta}\right)} = \sum_{j=0}^{B-1} X^{j\frac{N}{B}} \sum_{k=0}^{\frac{N}{B2^\vartheta}-1} X^{k\cdot 2^\vartheta} \sum_{i=0}^{2^\vartheta-1} f_{i+1}(j)X^i$$

During ModSwitch, we need ModSwitch to the start place of each 2^ϑ code block, that is $X^0, X^{2^\vartheta}, X^{2\cdot 2^\vartheta}, \dots, X^{N-2^\vartheta}$. Therefore, we only need to continuously extract 2^ϑ TLWE ciphertexts from TV after the BlindRotation algorithm. Here, we have slightly changed the algorithm because we fix the parameter $\varkappa = 0$ in the BML algorithm, see Algorithm 2.

Algorithm 2 PBSmanyLUT(BML)

```
Input: a TLWE sample $c = (a = [a_1, a_2, \dots, a_n], b) \in \mathrm{TLWE}_s(m \cdot \Delta_{in}) \in \mathbb{Z}_q^{n+1}$, $m \in \mathbb{Z}_B$, a TRLW sample CT of $TV_{\left(f_1,\dots,f_{2^\vartheta}\right)}$, bootstrapping key BSK $= \left\{\mathrm{BSK}_i = \mathrm{TRGSW}_{\mathbf{S}'}^{(B_g,\ell)}(s_i)\right\}_{1\le i\le n}$, a integer $\vartheta \in \mathbb{N}$ such that $\frac{q2^\vartheta}{\Delta_{in}} < 2N$
Output: $\mathrm{ct}_1, \dots, \mathrm{ct}_{2^\vartheta}$ such that $\mathrm{ct}_j = \mathrm{LWE}_{\bar{\mathbf{S}}}(f_j(m) \cdot \Delta_{\mathrm{out}})$, $\bar{\mathbf{S}}$ is a vector interpretation of $\mathbf{S}'$
for $i = 1$ to $n+1$ do
    $a'_i \leftarrow \left[\left\lfloor \frac{a_i \cdot 2N}{q\cdot 2^\vartheta} \right\rceil \cdot 2^\vartheta\right]_{2N}$
end for
$\mathrm{CT} \leftarrow \mathrm{BlindRotate}\left(\mathrm{CT}, \{a'_i\}_{1\le i\le n+1}, \mathrm{BSK}\right)$
for $j = 1$ to $2^\vartheta$ do
    $\mathrm{ct}_j \leftarrow \mathrm{SampleExtract}_{j-1}(\mathrm{CT})$
end for
return $\{\mathrm{ct}_1, \dots, \mathrm{ct}_{2^\vartheta}\}$
```

2.5 TreeFB

In 2021, Antonio Guimarães et al. proposed a Tree-based method to combine FBT algorithm(TreeFB) to accelerate the calculation of FBT with large LUT [19]. For a d-bit plaintext $m = \sum_{i=0}^{d-1} m_i B^i$, where B is torus base. First, encoding function F about m as a LUT of size B^d, then the TreeFB algorithm encodes LUT into B^{d-1} TVs and executes B^{d-1} bootstrappings to extract B^{d-1} TLWE ciphertexts by using the TLWE ciphertext of m_0 as a selector. The plaintexts of these TLWE ciphertexts are values of $F(M)$ where the last bit of M is m_0. Finally, the TreeFB algorithm performs the PublicKS algorithm to package these TLWE ciphertexts to B^{d-2} TRLWE ciphertexts as TVs for next layer, see Algorithm 3.

Algorithm 3 TreeFB

```
Input: a list of TLWE samples $\underline{c}_i \in \mathrm{TLWE}_s\left(\frac{m_i}{2B}\right)$, such that $\sum_{i=0}^{d-1} m_i B^i = m$, a set
    $L$ of $B^{d-1}$ polynomials $\in \mathbb{Z}_N[X]$ encoding the LUT of an arbitrary function $F$,
    a bootstrapping key $\mathrm{BK}_i \in \mathrm{TRGSW}_S(s_i)$, for $i \in [\![1, n]\!]$, a Key Switching key
    $\mathrm{KS}_{i,j} \in \mathrm{T(R)LWE}_{\bar{s}}\left(\frac{s_i}{2^j}\right)$, for $i \in [\![1, n]\!]$ and $j \in [\![1, t]\!]$
Output: A TLWE sample $\bar{c} \in \mathrm{TLWE}_{\bar{S}}\left(\frac{F(m)}{2B}\right)$ where $\bar{S} \in \mathbb{B}^N$ is a vector (TLWE)
    interpretation of $S \in \mathbb{B}_N[X]$
1: TV ← L
2: $f : \mathbb{T}^B \mapsto \mathbb{T}_N[X] = (a_1, \ldots, a_B) \mapsto a_1 + \ldots + a_B X^{N-1}$
3: for $i = 0$ to $d - 1$ do
4:     $\bar{c} \leftarrow \mathrm{MultiValueBoostrap}(\underline{c}_i, TV, BK)$
5:     for $j = 1$ to $B^{d-i-2}$ do
6:         $\mathrm{TV}_{j-1} \leftarrow \mathrm{PublicKS}\left(\left(\bar{c}_{(j-1)\times B}, \ldots, \bar{c}_{j\times B}\right), f, \mathrm{KS}\right)$
7:     end for
8: end for
9: return $\bar{c}_0$
```

2.6 Base-aware-KS Switching

Antonio Guimarães et al. proposed Base-aware-KS Switching to accelerate the process of packing KeySwitching [19] by replacing polynomial multiplication polynomials by the inner product of a B-size vector of digits with a B-size vector of TRLWE ciphertext. Moreover, they use a larger base for decomposition to improve efficiency. Specifically, when packaging B TLWE ciphertexts, they generate key-switching key $\mathrm{KS}_{i,j,b} \in \mathrm{TRLWE}_{\bar{S}}\left(\frac{s_i}{base^j} \cdot \sum_{q=bN/B}^{(b+1)N/B-1} X^q\right)$, and replace the product of polynomial $\tilde{a}_{i,j}$ and TRLWE ciphertext $\mathrm{KS}_{i,j}$(Line 6, Algorithm 1) with the inner product of torus vector $\tilde{a}'_{i,j} = (\tilde{a}_{i,j,1}, \ldots, \tilde{a}_{i,j,B})$ and TRLWE samples vector $\mathrm{KS}'_{i,j} = (\mathrm{KS}_{i,j,0}, \mathrm{KS}_{i,j,1}, \ldots, \mathrm{KS}_{i,j,B-1})$, see in Algorithm 4.

3 Optimization of PBS with Large LUT

TreeFB algorithm is a relatively effective PBS with large LUT calculation algorithm. The algorithm uses MVB to reduce the B^{d-1-i} bootstrappings of ith layer to one bootstrapping(Line 4, Algorithm 3). However, TV exists in clear text format only during the first layer calculation, which can directly using MVB algorithm to optimize TreeFB algorithm, in subsequent calculations, TV is in TRLWE ciphertext format. As we mentioned before, MVB algorithm bases on multiplying the results of BlindRotate with TVs but TRLWE * TRLWE is not directly supported in TFHE, only TRLWE * TRGSW is supported, an expensive circuit bootstrap must be performed(about 130ms for 1-bit) to conversion from TRLWE to TRGSW ciphertext in order to utilize MVB algorithm. Otherwise, it is necessary to perform $B^{(d-1-i)}$ bootstrapping one by one. Regardless of which method is selected, the actual efficiency of the algorithm will be

Algorithm 4 Base-aware-KS Switching

```
Input: $B$ TLWE samples $\underline{c}^{(z)} = \left(\underline{a}^{(z)}, \underline{b}^{(z)}\right) \in \mathrm{TLWE}_{\underline{s}}\left(\mu_z\right), z \in [\![1, B]\!]$, a precision parameter $t \in \mathbb{Z}$, a Key-Switching key $\mathrm{KS}_{i,j,b} \in \mathrm{TRLWE}_{\bar{S}}\left(\frac{s_i}{base^j} \cdot \sum_{q=bN/B}^{(b+1)N/B-1} X^q\right)$,
    for $i \in [\![1, n]\!]$, $j \in [\![1, t]\!]$, $b \in [\![0, B)$
Output: a TRLWE sample $\bar{c} \in \mathrm{TRLWE}_{\bar{s}}\left(f\left(\mu_z\right)\right)$, for $z \in [\![1, p]\!]$.
$f : \mathbb{T}^B \mapsto \mathbb{T}_N[X] = (a_1, \ldots, a_B) \mapsto a_1 + \ldots + a_B X^{N-1}$
for $i = 1$ to $n$ do
    for $b = 1$ to $B$ do
        $\tilde{a}_{i,b} \leftarrow \left\lceil \underline{a}_i^{(b)} \right\rfloor_{\frac{1}{2^t}}$, i.e., the closest multiple of $\frac{1}{2^t}$ to $\underline{a}_i^{(b)}$
        Decompose each $\tilde{a}_{i,b} = \sum_{j=1}^{t} \tilde{a}_{i,j,b} \cdot base^{-j}$
    end for
end for
return $\left(0, f\left(\underline{b}_i^{(1)}, \underline{b}_i^{(2)}, \ldots, \underline{b}_i^{(p)}\right)\right) - \sum_{i=1}^{n} \sum_{j=1}^{t} \left\langle \tilde{a}'_{i,j}, \mathrm{KS}'_{i,j} \right\rangle$
```

seriously affected. Antonio Guimarães et al. only use MVB algorithm in 1th layer calculation in their experiment. I. Chillotti et al. introduced the method of *vertical packing* [13] to accelerate calculation of LUT, which based on CUMX-tree. Its selector need to be a TRGSW ciphertext, this is a very efficient way in a level homomorphic setting but it is necessary to perform TRLWE to TRGSW conversion(circuit bootstrapping) in the full homomorphic encryption setting.

In 2022, Antonio et al. mentioned using the full TRGSW bootstrap algorithm to accelerate the TreeFB algorithm [22], but this method also requires a circuit bootstrapping and increases the storage space requirement during the calculation process.

Therefore, we reduce the number of bootstrapping and key switching algorithms required for each layer in TreeFB algorithmby compressing $2^{\vartheta} \cdot B$ instead of B TLWE ciphertexts in a TRLWE ciphertext and using the BML algorithm to extract 2^{ϑ} TLWE ciphertext in a single bootstrapping. Due to the fact that the BML algorithm can only extract 2^{ϑ} consecutive TLWE ciphertexts, while the TreeFB algorithm needs to extract one result from every B consecutive TLWE ciphertexts. Therefore, we divide each B consecutive TLWE ciphertexts from $2^{\vartheta}B$ TLWE ciphertexts into a group and compress the TLWE ciphertexts at the same position in each group into consecutive polynomial slot positions(see Figure 3).

The key-switching key has been determined before executing the algorithm, that is, the value of ϑ needs to be determined. Hence, we use “copy” and “filling zero” method when the number of ciphertext needs to be compressed is less than $2^{\vartheta}B$(Line 12 and Line 14, Algorithm 5), refer to Appendices B for more details.

We show the optimized TreeFB algorithm in Algorithm 5, which we call as the Tree-BML algorithm, where

$$f_{\vartheta} : \mathbb{T}^{2^{\vartheta}B} \mapsto \mathbb{T}_N[X] = (a_1, \ldots, a_{2^{\vartheta}B}) \mapsto \sum_{e=0}^{B-1} \sum_{j=0}^{\frac{N}{2^{\vartheta}B}-1} \sum_{i=0}^{2^{\vartheta}-1} a_{e+i \cdot B} X^{e \cdot \frac{N}{B} + j \cdot 2^{\vartheta} + i}$$

Algorithm 5 Tree-BML

Input: a set of TLWE samples $\underline{c}_i \in \mathrm{TLWE}_s\left(\frac{m_i}{2B}\right)$, such that $\sum_{i=0}^{d-1} m_i B^i = m$, a set L of $\frac{B^{d-1}}{2^\vartheta}$ polynomials $\in \mathbb{Z}_N[X]$ encoding the LUT of an arbitrary function F, a bootstrapping key $\mathrm{BK}_i \in \mathrm{TRGSW}_S(s_i)$, for $i \in [\![1, n]\!]$, a Key Switching key $\mathrm{KS}_{i,j} \in \mathrm{T(R)LWE}_{\bar{s}}\left(\frac{s_i}{2^j}\right)$, for $i \in [\![1, n]\!]$ and $j \in [\![1, t]\!]$

Output: A TLWE sample $\bar{c} \in \mathrm{TLWE}_{\bar{S}}\left(\frac{F(m)}{2B}\right)$ where $\bar{S} \in \mathbb{B}^N$ is a vector (TLWE) interpretation of $S \in \mathbb{B}_N[X]$

```
TV ← L
$\bar{c} \leftarrow \mathrm{MultiValueBoostrap}(\underline{c}_0, TV, BK)$
for $i = 1$ to $d - 1$ do
    $Fu \leftarrow \lfloor \frac{B^{d-i-1}}{2^\vartheta} \rfloor$
    $R \leftarrow B^{d-i} \bmod 2^\vartheta B$
    for $j = 0$ to $Fu - 1$ do
        $\mathrm{TV}_j \leftarrow \mathrm{PublicKS}(\bar{c}_{j2^\vartheta B}, \ldots, \bar{c}_{(j+1)2^\vartheta B-1}, f_\vartheta, \mathrm{KS})$
        $(\bar{c}_{j2^\vartheta}, \ldots, \bar{c}_{(j+1)2^\vartheta-1}) \leftarrow \mathrm{PBSmanyLUT}(\underline{c}_i, \mathrm{TV}_j, \mathrm{BK}, \vartheta)$
    end for
    if $R \neq 0$ then
        if $R \mid 2^\vartheta B$ then
            $\mathrm{TV}_{Fu} \leftarrow \mathrm{PublicKS}((\bar{c}_{Fu\cdot 2^\theta B}, \ldots, \bar{c}_{Fu\cdot 2^\theta B+R-1}), f_{\log(\frac{R}{B})}, \mathrm{KS})$
        else
            $\mathrm{TV}_{Fu} \leftarrow \mathrm{PublicKS}((\bar{c}_{Fu\cdot 2^\theta B}, \ldots, \bar{c}_{Fu\cdot 2^\theta B+R-1}, 0, \ldots, 0), f_{\lceil\log(\frac{R}{B})\rceil}, \mathrm{KS})$
        end if
        $(\bar{c}_{Fu2^\vartheta}, \ldots, \bar{c}_{Fu2^\vartheta+\lceil\log(\frac{R}{B})\rceil-1}) \leftarrow \mathrm{PBSmanyLUT}(\underline{c}_i, \mathrm{TV}_{Fu}, \mathrm{BK}, \lceil\log(\frac{R}{B})\rceil)$
    end if
end for
return $\bar{c}_0$
```

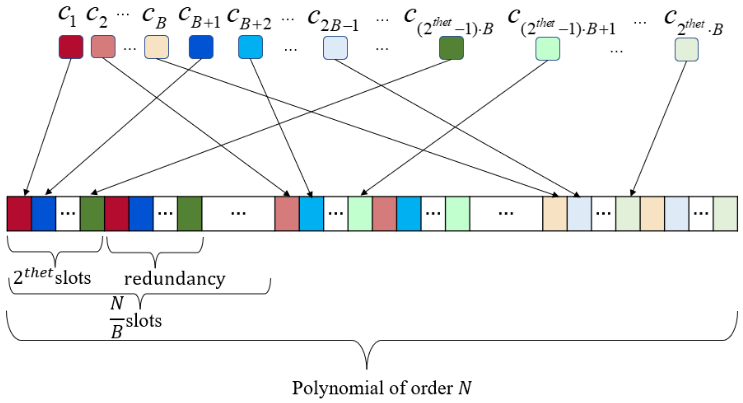

Fig. 1. compression of $2^\vartheta B$ TLWE ciphertexts

Remark: for convenience, our Tree-BML and the TreeFB algorithm does not mention that the LUT values of $F(m)$ is stored based on B decomposition because the parameter of the algorithm supports the plaintext space of size B to successfully perform. Therefore, for example, set $m \in [\![0, B^d - 1]\!]$ and $\max\lceil \log_B f(m) \rceil = l$, the actual LUT size is $l \cdot B^d$. Then the algorithm outputs l TLWE ciphertexts corresponding to the lth bit(based B) of result.

4 Optimization of Base-aware-KS

Although the Base-aware-KS algorithm can effectively reduce the running time of the packing key switching process, it significantly increases the size of the key-switching key. Moreover, if we package TLWE ciphertexts by using the Base-aware-KS algorithm, the more TLWE ciphertexts packaged the larger size of key-switching key will expand. To avoid this expansion, we observe that in these Key-Switching key ciphertexts $\mathrm{KS}_{i,j,b} \in \mathrm{TRLWE}_{\bar{S}}\left(\frac{s_i}{base^j} \cdot \sum_{q=bN/B}^{(b+1)N/B-1} X^q\right)$, if we need to pack B TLWE samples then $b \in [\![1, B]\!]$, which encrypted polynomial coefficient values are the same except these values slot positions. Therefore, we only generate the key-switching key $\mathrm{KS}_{i,j,0}$. We reduce the storage space consumed by rotating the key-switching key to the appropriate position in subsequent calculations. Therefore we replace ith inner product(Line 8, Algorithm 4) by

$$\left\langle \tilde{a}'_{i,j}, (\mathrm{KS}_{i,j,0}, X^{\frac{N}{B}} \cdot \mathrm{KS}_{i,j,0}, \dots, X^{(B-1)\cdot\frac{N}{B}} \cdot \mathrm{KS}_{i,j,0}) \right\rangle$$

However, in our Tree-BML algorithm, the $2^\vartheta B$ ciphertexts to be compressed are not continuously stored, which mean that there is only one redundant value every 2^ϑ slots(see Figure 3). Therefore, we generate key-switching key which inserts s_i every 2^ϑ slots in the first $\frac{N}{B}$ slots in the polynomial of N order, that is, $\mathrm{KS}_{i,j} \in \mathrm{TRLWE}_{\bar{S}}\left(\frac{s_i}{base^j} \cdot \sum_{q=0}^{\frac{N}{2^\vartheta B}-1} X^{q\cdot 2^\vartheta}\right)$.

We show the optimized Base-aware-KS algorithm in Algorithm 6, we call it Rotate-based Packing KS, where $\tilde{a}'_{i,j} = (\tilde{a}_{i,j,1}, \dots, \tilde{a}_{i,j,2^\vartheta B})$ and

$$\begin{aligned}\mathrm{KS}'_{i,j} = \mathrm{KS}_{i,j} \cdot (1, X^{\frac{N}{B}}, \dots, X^{(B-1)\frac{N}{B}}, X, X^{\frac{N}{B}+1}, \dots, X^{(B-1)\frac{N}{B}+1},\\ \dots, X^{2^\vartheta-1}, X^{\frac{N}{B}+2^\vartheta-1}, \dots, X^{\frac{N}{B}+2^\vartheta-1})\end{aligned}$$

5 Error analysis

In this section, we analyze the error variance and error rate of our algorithms.

Algorithm 6 Rotate-based Packing KS

```
Input: $2^{\vartheta}B$ TLWE samples $\underline{c}^{(z)} = \left(\underline{a}^{(z)}, \underline{b}^{(z)}\right) \in \mathrm{TLWE}_{\underline{s}}\left(\mu_z\right), z \in [\![1, 2^{\vartheta}B]\!]$, a precision parameter $t \in \mathbb{Z}$, a Key Switching key $\mathrm{KS}_{i,j} \in \mathrm{TRLWE}_{\bar{S}}\left(\frac{s_i}{2^j} \cdot \sum_{q=0}^{\frac{N}{2^{\vartheta}B}-1} X^{q \cdot 2^{\vartheta}B}\right)$, for $i \in [\![1, n]\!]$, $j \in [\![1, t]\!]$
Output: a TRLWE sample $\bar{c} \in \mathrm{TRLWE}_{\bar{s}}\left(f\left(\mu_1, \mu_2, \ldots, \mu_{2^{\vartheta}B}\right)\right)$
1: $f_k : \mathbb{T}^{2^k B} \mapsto \mathbb{T}_N[X] = (a_1, \ldots, a_{2^k B}) \mapsto \sum_{e=0}^{B-1} \sum_{j=0}^{\frac{N}{B2^k}-1} \sum_{i=0}^{2^k-1} a_{e+i \cdot B} X^{e\frac{N}{B}+j2^{\vartheta}+i}$
2: for $i = 1$ to $n$ do
3:     for $b = 1$ to $2^{\vartheta}B$ do
4:         $\tilde{a}_{i,b} \leftarrow \left\lceil \underline{a}_i^{(b)} \right\rfloor_{\frac{1}{2^t}}$, i.e., the closest multiple of $\frac{1}{2^t}$ to $\underline{a}_i^{(b)}$
5:         Decompose each $\tilde{a}_{i,b} = \sum_{j=1}^{t} \tilde{a}_{i,j,b} \cdot 2^{-j}$
6:     end for
7: end for
8: return $\left(0, f_k\left(\underline{b}_i^{(1)}, \underline{b}_i^{(2)}, \ldots, \underline{b}_i^{(2^{\vartheta}B)}\right)\right) - \sum_{i=1}^{n} \sum_{j=1}^{t} \left\langle \tilde{a}'_{i,j}, \mathrm{KS}'_{i,j} \right\rangle$
```

5.1 Error Variance of the Rotate-based Packing KS

We first provide the error variance analysis result of the PublicKS and Base-aware-KS algorithm, as shown in Equation 1. ϑ_{KS} represent the error variance of key-switching keys and R is related with public function($R^2 = 1$ in TreeFB because we only use TLWE-to-TRLWE packing function).

$$\mathrm{Var}(\mathrm{Err}(c)) \leq R^2\,\mathrm{Var}(\mathrm{Err}(\underline{c})) + \underline{n}tN\vartheta_{KS} + \frac{1}{12}\underline{n}base^{-2t} \tag{1}$$

The differences of our Rotate-based Packing KS compare to Base-aware-KS algorithm are mainly that the error of each TRLWE samples in $\mathrm{KS}'_{i,j}$ is not independent and additionally calculated $\cdot X^i$(Line 8, Algorithm 6). In Equation 1, the second term represents the error variance of TRLWE samples $\mathrm{KS}_{i,j}$ sums, so we need to change the second term.

First, $\cdot X^i$ does not increase the norm of error, see in Equation 2.

$$\left\|X^i A\right\|_2 \leq \left\|X^i\right\|_2 \left\|A\right\|_2 = \left\|A\right\|_2 \tag{2}$$

Then, according to the relevant knowledge of statistical probability theory, when variables $\mathrm{E}(a) = 0$ and $\mathrm{E}(b) = 0$ then $\mathrm{Var}(a+b) = \mathrm{Var}(a)+\mathrm{Var}(b)$ although variables a and b are not mutually independent. Due to $\mathrm{E}(error(\mathrm{KS}_{i,j})) = 0$, then thier sum(Line 8, Algorithm 6) do not introduce new error.

So, our Rotate-based Packing KS algorithm did not introduce new error.

5.2 Error Variance of the Tree-BML

We can obtain that the int. coefficients input $a_1, \ldots, a_p, b \in \mathbb{Z}/(2N\mathbb{Z})$ do not affect the output error variance of the BlindRotate algorithm through the analysis process of I. Chillotti et al. [20], which means the ModSwitch process do not affect the output error variance of the bootstrapping algorithm. Because the differences between the BML algorithm and the FBT algorithm lies in the ModSwitch process and the SampleExtract process(the SampleExtract process does not add new error), the output error variance of the BML algorithm is the same as the FBT algorithm. So compared to TreeFB, our Tree-BML algorithm does not introduce additional output error. We show the out error variance of Tree-BML in Equation 3, refer Appendices C for more detials.

$$\begin{aligned}\mathrm{Var}(\mathrm{TreeFB}) \leq (d-1+\|TV_f\|_2^2)(\underline{n}(k+1)\ell N\left(\frac{B_g}{2}\right)^2\vartheta_{BK}+\underline{n}(1+kN)\epsilon^2)+\\ (d-1)(\underline{n}tN\vartheta_{KS}+\frac{1}{12}\underline{n}base^{-2t})\end{aligned} \tag{3}$$

5.3 Eerror rate

In this section, we analyze the error rate of Tree-BML. We need the BlindRotation algorithm to output appropriate values during the bootstrapping process. So it is necessary to ensure the correctness of the ModSwitch results. We refer the analysis method and result of I. Chillotti et al. [20], for the correctness of the ModSwitch probability $P = \mathrm{erf}\left(\frac{\Gamma}{\sqrt{2}}\right)$, where erf is the Gaussian error function, σ_{in} denote error variance of the input LWE ciphertext and $w = 2N \cdot 2^{-\vartheta}$.

Although our Tree-BML algorithm does not increase the error variance of the output, for the same successful probability of ModSwitch, our Tree-BML algorithm has stricter error requirements for the TLWEs input than TreeFB algorithm, see Equation 4.

$$\sigma_{\mathrm{in}}^2 < \frac{q^2}{16B^2\Gamma^2} - \frac{q^2}{12w^2} + \frac{1}{12} - \frac{nq^2}{24w^2} - \frac{n}{48} \tag{4}$$

Alternatively, for the same security level i.e. the same input error variance, our Tree-BML algorithm has a slightly lower probability of ModSwitch success than TreeFB algorithm, see Equation 5.

$$\Gamma < \frac{q}{B\sqrt{(16\sigma_{\mathrm{in}}^2 + \frac{4q^2}{3w^2} - \frac{4}{3} + \frac{2nq^2}{3w^2} + \frac{n}{3})}} \tag{5}$$

We will validate our results about error variance and rate by experiments in the next section.

6 Experimental results

We conducted experiments on the Ubuntu 20.0 virtual machine on a computer with i7-10700F @ 2.90GHZ, and comparisons with others on LUT, integer comparison, and Relu function. Fortunately, the code for the TreeFB algorithm was completely open source, and we fully implemented the author's code.

For the code that has not been implemented, we will make appropriate scaling based on the proportion of the implemented parts. In order to compare with the TreeFB algorithm, we adopted the parameters set by Antonio Guimarães et al. as shown in Table 1. We take the average time after running 10 times for each function. However, due to different hardware conditions, the experimental results are slightly different such as 6-bit-to-6-bit LUT in the author's paper were 378.2ms, 409.9ms in our environment. It should be noted that there may be slight differences in the results of each running(about 10%) even in the same environment.

Table 1. Experiment parameters

Security Level	B	LWE		RLWE			Bootstrap		Key Switch	
		n	σ	N	k	σ	l	$\log_2(B_g)$	Base	t
127	4	630	2^{-15}	1024	1	2^{-25}	5	5	64	2

6.1 Calculation of LUT

We conducted experiment about 6-bit, 8-bit, 10-bit, and 12-bit LUT calculations by useing Tree-BML algorithm with parameter $\vartheta = 1, 2, 3$ and TreeFB algorithm, see Figure 2, where Tree-BML algorithm utilizes Rotate-based Packing KS and TreeFB algorithm utilizes Base-aware-KS for reducing key-switching key size. The detailed data results are shown in the Table 2. We can abtain that as ϑ increases, the algorithm consumes less time. However, in the calculation of 6-bit LUT, the speed increase is not significant when ϑ increases from 2 to 3. This is because even if we can output more results in one calculation, we still only need one LUT value in the end.

Table 2. Time consumption of LUT calculation

Method	Security	6-bit	8-bit	10-bit	12-bit	Reduced time
TreeFB	127	0.409s	2.203s	23.667s	51.085s	1
$\vartheta = 1$	127	0.215s	1.026s	5.013s	24.291s	51.20%
$\vartheta = 2$	127	0.135s	0.586s	2.805s	13.181s	72.09%
$\vartheta = 3$	127	0.124s	0.409s	1.779s	8.092s	79.66%

6.2 32-bit integer comparison

We compared the Rotate-based Packing KS with Base-aware-KS algorithm in the integer comparison function experiment. The experimental results are shown

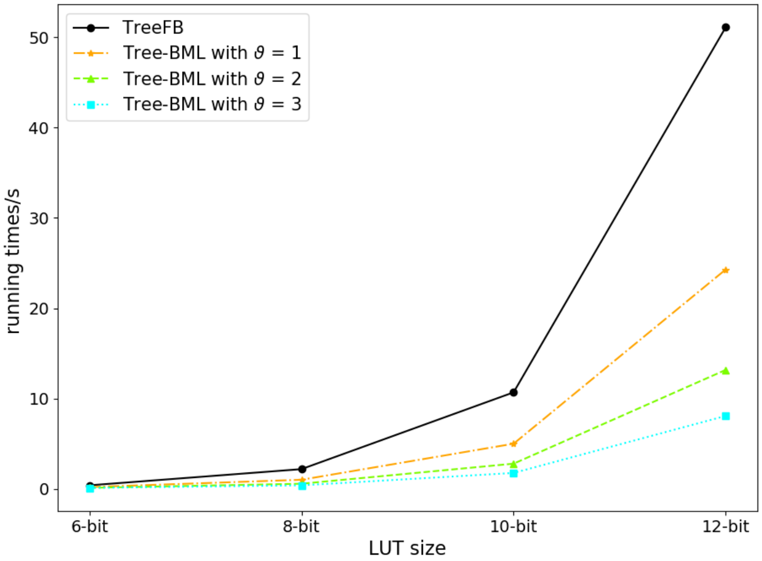

Fig. 2. Calculation of LUT

in Table 3. It can be obtained that our Rotate-based Packing KS algorithm only requires approximately 700MB for storage key-switch key, while the Base-aware-KS algorithm requires approximately 2GB and our Rotate-based Packing KS algorithm almost no increasing runtime compare with the Base-aware-KS algorithm.

Table 3. Experimental result of 32-bit integer comparison

KeySwitching Method	Security	time	key-switching key size	Reduced key Size
Base-aware-KS	127	343.5ms	$\approx$ 2.113GB	1
Ours	127	343.7ms	$\approx$ 0.528GB	75.01%

6.3 8-bit Relu

One of the activation function widely used in neural networks is the Rectified Linear Unit(Relu). Due to $B = 4$, the LUT results are also decomposed based on B(as mentioned earlier), we need to output 4 result bits in the calculation of the 8-bit Relu function. Therefore, when $\vartheta = 2$, the efficiency of the algorithm is the highest. The results are shown in the Table 4.

Table 4. Experimental result of 8-bit Relu

Method	Security	time consumption	Reduced time
[LJ19] [23]	127	603.10ms	No
[ZLPL20] [24]	127	103.10ms	No
TreeFB	127	89.03ms	1
Ours	127	25.74ms	71.09%

6.4 Error result

We counted the values of output error of using the TreeFB and our Tree-BML(when $\vartheta = 2$) algorithms to calculate the 6-bit-to-6-bit LUT 3000 times respectively, then conducted interval statistics(step length 100000), see Figure 3. The probability distribution of the output error of the two algorithms is similar, which proof our conclusion that the output error variance of Tree-BML and TreeFB algorithm is the same.

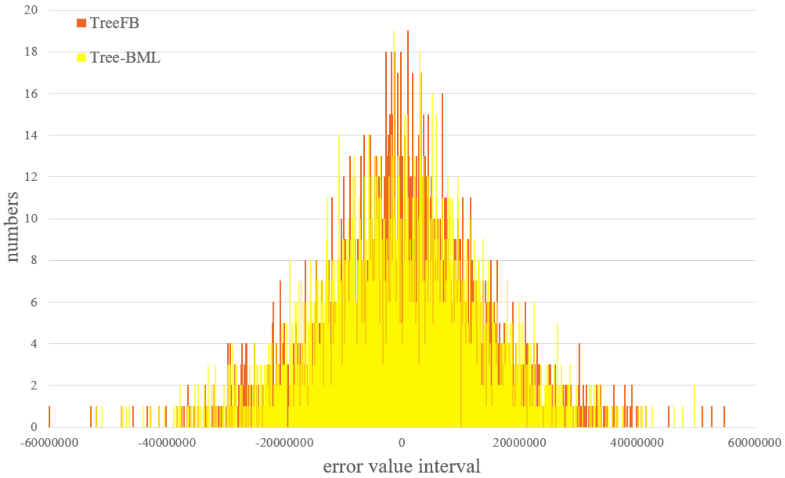

Fig. 3. interval statistics of values of output error

We calculated the successful probability of ModSwitch of the Tree-BML and TreeFB algorithm under different input error variances using Equation 5, see Picture 4. When $\vartheta = 1$ or 2, our Tree-BML algorithm almost no reduces the successful probability compared to TreeFB algorithm. However, when $\vartheta = 3$, our Tree-BML algorithm's probability decreases significantly, but also reaches 99.8%.

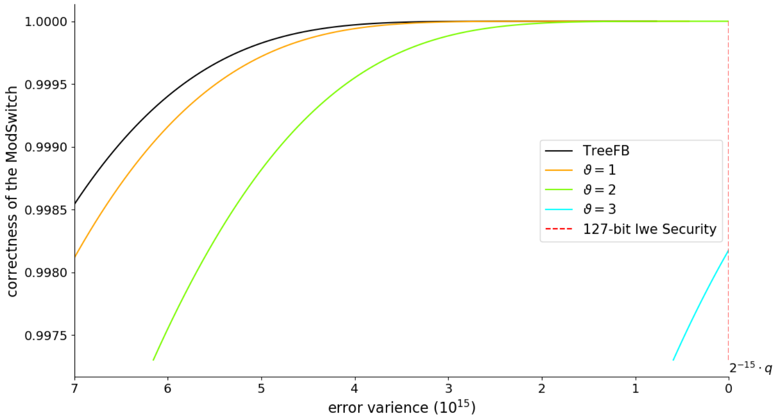

Fig. 4. error rate of Tree-BML and TreeFB algorithm

7 Conclusion

In this paper, we accelerated the calculation of TreeFB algorithm by compressing $2^{\vartheta}B$ TLWE ciphertexts into one TRLWE ciphertext and utilizing the BML algorithm, which reduces the number of functional bootstrapping and key switching times required by the algorithm. When $\vartheta = 1, 2,$ and 3, our Tree-BML algorithm reduce the runtime of TreeFB algorithm by 51.20%, 72.09% and 79.66% respectively. Plus, we proposed Rotate-based Packing KS algorithm that reduces the size of the key-switching key compare with the Base-aware-KS algorithm and almost no increasing the runtime. When $B = 4$, our algorithm reduce the storage requirement of the key-switching key by 75%. We analyzed the output error variance and error rate of our algorithms and obtained that the Rotate-based Packing KS and Tree-BML algorithm does not introduce new error. We conducted experiments on 6-bit, 8-bit, 10-bit, 12-bit LUTs, 32-bit integer comparisons, 8-bit Relu functions, error variance and error rate about our algorithms and compared our results with others. It can be obtained that as the ϑ value increases the efficiency of the algorithm increases, while the efficiency improvement rate gradually slows down. This is because that although we reduced the number of Bootstraps required in each **for** loop of the TreeFB algorithm, we did not reduce the depth of the algorithm. An interesting open problem is to reduce the depth of the TreeFB algorithm.

References

1. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Symposium on the Theory of Computing*, 2005.
2. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15*, pages 617–635. Springer, 2009.
3. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.
4. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.
5. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.
6. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 868–886. Springer, 2012.
7. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.
8. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.
9. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013.
10. Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*, pages 617–640. Springer, 2015.
11. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 3–33. Springer, 2016.
12. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 377–408. Springer, 2017.
13. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

14. Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Concrete: Concrete operates on ciphertexts rapidly by extending tfhe. In *WAHC 2020-8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2020.
15. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*, pages 1–19. Springer, 2021.
16. Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Simulating homomorphic evaluation of deep learning predictions. In *Cyber Security Cryptography and Machine Learning: Third International Symposium, CSCML 2019, Beer-Sheva, Israel, June 27–28, 2019, Proceedings*, pages 212–230. Springer, 2019.
17. Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization & larger precision for (t) fhe. *Cryptology ePrint Archive*, 2022.
18. Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*, pages 106–126. Springer, 2019.
19. Antonio Guimarães, Edson Borin, and Diego F Aranha. Revisiting the functional bootstrap in tfhe. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 229–253, 2021.
20. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*, pages 670–699. Springer, 2021.
21. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.
22. Antonio Guimarães, Edson Borin, and Diego F. Aranha. Mosfhet: Optimized software for fhe over the torus. Cryptology ePrint Archive, Paper 2022/515, 2022. `https://eprint.iacr.org/2022/515`.
23. Qian Lou and Lei Jiang. She: A fast and accurate deep neural network for encrypted data. *Advances in neural information processing systems*, 32, 2019.
24. Junwei Zhou, Junjiong Li, Emmanouil Panaousis, and Kaitai Liang. Deep binarized convolutional neural network inferences over encrypted data. In *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 160–167. IEEE, 2020.

Appendices

A Functional Bootstraps(FBT) Algorithm

```
Algorithm 7 Functional Bootstraps(FBT)
Input: a TLWE sample $\underline{c} = (\underline{a}, \underline{b}) \in \mathrm{TLWE}_s(\frac{m}{2B}) \in \mathbb{Z}_q^{n+1}$, for $m \in \mathbb{Z}_B$, an integer
    LUT $L = [l_0, l_1, \ldots, l_{B-1}] \in \mathbb{Z}_B^B$, a set of bootstrapping key $\mathrm{BK}_i \in \mathrm{TRGSW}_S(s_i)$,
    for $i \in [\![1, n]\!]$
Output: $\bar{c} \in \mathrm{TLWE}_{\bar{S}}(\frac{L[m]}{2B})$, where $\bar{S} \in \mathbb{B}^N$ is a vector (TLWE) interpretation of
    $S \in \mathbb{B}_N[X]$
$b \leftarrow \lfloor \frac{2N\underline{b}}{q} \rceil$ and $a_i \leftarrow \left\lfloor \frac{2N\underline{a_i}}{q} \right\rceil \in \mathbb{Z}_{2N}$ for each $i \in [\![1, n]\!]$
$v \leftarrow \sum_{i=0}^{N-1} \frac{1}{2B} \cdot l_{\lfloor \frac{iB}{N} \rfloor} X^i \in \mathbb{T}_N[X]$
$\mathrm{ACC} \leftarrow \mathrm{BlindRotate}\left((0, v), (a_1, \ldots, a_n, b + \frac{1}{4B}), (\mathrm{BK}_1, \ldots, \mathrm{BK}_n)\right)$
return $\mathrm{SampleExtract}_0(\mathrm{ACC})$
```

B Method of Compressing Ciphertexts in Tree-BML

Assuming the number of TLWE ciphertexts we need to compress is $K \cdot B$.

1. When $K \mid 2^{\vartheta}$, we copy these ciphertexts to achieve $2^{\vartheta}B$. Since we only need to extract K TLWE ciphertexts in the corresponding BML algorithm, we need to change the corresponding ModSwitch and SampleExtract process parameter $\vartheta = \log(k)$.
2. When K cannot divide 2^{ϑ}, we fill $K \cdot B$ TLWEs with $(2^{\lceil \log k \rceil} - k) \cdot B$ zeros in the high position. Then execute the algorithm for the "divisible" method mentioned above. Moreover, filling the high position with 0 does not affect the correctness of the result because the extracted ciphertext at the backer of the position corresponds to the higher position of the result after decryption.

C Error Variance of Tree-BML

We first introduce two equations from paper [19], Equation 6 is the error variance of FBT algorithm and Equation 7 is the error variance of MVB algorithm, where ϑ_{BK} represent the error variance of bootstrapping key and $\epsilon = \frac{1}{2B_g^l}$.

$$\mathrm{Var}(\mathrm{Err}(c)) \leq \mathrm{Var}(\mathrm{Err}(TV)) + \underline{n}(k+1)\ell N \left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \underline{n}(1+kN)\epsilon^2 \tag{6}$$

$$\mathrm{Var}(\mathrm{Err}(c)) \leq \|TV_f\|_2^2 \left(\underline{n}(k+1)\ell N \left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \underline{n}(1+kN)\epsilon^2\right) \tag{7}$$

For a $m \in B^d$, the TreeFB algorithm requires to perform d layers FBT and $d-1$ layers PublicKS algorithms(Line 3 and Line 5, Algorithm 3). The output of FBT is the input of PublicKS, that is, the output error of FBT algorithm are the input error $\mathrm{Var}(\mathrm{Err}(\underline{c}))$ in Equation 1 of PublicKS algorithm. so after the first layer of PublicKS algorithm, the output error variance is

$$\|TV_f\|_2^2 \left(\underline{n}(k+1)\ell N \left(\frac{B_g}{2}\right)^2 \vartheta_{BK} + \underline{n}(1+kN)\epsilon^2\right) + \underline{n}tN\vartheta_{KS} + \frac{1}{12}\underline{n}base^{-2t}$$

Then, the results enter the next layer of FBT, which mean that the above equation is the new Var(Err(TV)) in Equation 6. Finally, the output error variance of the TreeFB algorithm as shown in Equation 8.

$$\begin{aligned} \text{Var}(\text{TreeFB}) \leq (d-1+\|TV_f\|_2^2)(\underline{n}(k+1)\ell N\left(\frac{B_g}{2}\right)^2\vartheta_{BK}+\underline{n}(1+kN)\epsilon^2)+ \\ (d-1)(\underline{n}tN\vartheta_{KS}+\frac{1}{12}\underline{n}base^{-2t}) \end{aligned} \tag{8}$$

Due to our Rotate-based Packing KS do not introduce new error, the main difference between our Tree-BML algorithm and TreeFB algorithm lies in the difference between FBT and BML algorithms. So we only need to analyze the difference of error variance between FBT and the BML algorithm. It can be obtained that the values of int. coefficients input do not affect the output error variance of FBT or BML algorithm form Equation 6, hence the output error variance of our Tree-BML algorithm is the same as the TreeFB algorithm.