Discretization Error Reduction for High Precision Torus Fully Homomorphic Encryption[⋆]

Kang Hoon Lee and Ji Won Yoon

Korea University, School of CyberSecurity
{hoot55, jiwon_yoon}@korea.ac.kr

Abstract. In recent history of fully homomorphic encryption, bootstrapping has been actively studied throughout many HE schemes. As bootstrapping is an essential process to transform somewhat homomorphic encryption schemes into fully homomorphic, enhancing its performance is one of the key factors of improving the utility of homomorphic encryption.
In this paper, we propose an extended bootstrapping for TFHE, which we name it by EBS. One of the main drawback of TFHE bootstrapping was that the precision of bootstrapping is mainly decided by the polynomial dimension N. Thus if one wants to bootstrap with high precision, one must enlarge N, or take alternative method. Our EBS enables to use small N for parameter selection, but to bootstrap in higher dimension to keep high precision. Moreover, it can be easily parallelized for faster computation. Also, the EBS can be easily adapted to other known variants of TFHE bootstrappings based on the original bootstrapping algorithm. We implement our EBS along with the full domain bootstrapping methods known (FDFB, TOTA, Comp), and show how much our EBS can improve the precision for those bootstrapping methods. We provide experimental results and thorough analysis with our EBS, and show that EBS is capable of bootstrapping with high precision even with small N, thus small key size, and small complexity than selecting large N by birth.

Keywords: Homomorphic encryption, TFHE, Precision

1 Introduction

Fully homomorphic encryption (FHE) is a powerful cryptographic scheme that allows to compute on encrypted data with unlimited depth, without leaking any information about it. Nonetheless, performing homomorphic operations on ciphertext accumulates noise or consumes certain amount of levels, and can only evaluate circuits with bounded depth. Thus to support unlimited level of computation, these schemes come with an operation called *bootstrapping*, which follows from the blueprint of Gentry [18]. Bootstrapping refreshes a possibly

[⋆] © IACR 2023. This article is the final version submitted by the authors to the IACR and to Springer-Verlag on 17th February 2023. The version published by Springer-Verlag is available at https://doi.org/00.00000/0000000000.

noisy, or level-consumed ciphertext into a fresh ciphertext, and allows further computation.

The FHEW/TFHE [11, 16] style bootstrapping methods are still known to be one of the most efficient bootstrapping methods. These algorithms refers to the *blind rotation* algorithm, which refreshes the noisy ciphertext, and evaluates pre-computed lookup table at the same time. From its name, the blind rotate algorithm rotates a polynomial of degree N, power-of-2, by certain amount blindfolded, over $2N$-th cyclotomic ring. The LUT of a function is encoded as the coefficients of the polynomial, and the blind rotation homomorphically selects the value from the encoded LUT by rotating the polynomial. The amount of rotation is decided by the message encrypted inside the ciphertext, with rounding error added due to the scaling-and-rounding of $(n+1)$ coefficients in $\mathbb{T}$ into $\mathbb{Z}_{2N}$. Due to this rounding, bootstrapping in FHEW/TFHE style can only preserve at most $(\log_2 N + 1)$-bits of precision of the input ciphertext, and the precision is actually much smaller in practice due to the summation of those rounding errors, restricting the high precision usage of these schemes. Thus, it is believed that one should select huge N to bootstrap with high precision.

To manage real world applications with low precision ciphertexts, the most familiar, but powerful solution is to decompose the message by some base, and encrypt each of the decomposed message in a single ciphertext. The original binary logic of TFHE is a special case of this decomposition, where the base is 2. Clearly, the smaller the base is, the number of bootstrapping increases for performing arithmetic operations on decomposed ciphertexts. If larger base is used to decrease the number of bootstrapping, the bootstrapping precision works as an upper-bound of the size of the base, as the bootstrapping must preserve precision at least $\log_2(\text{base})$. This forces to use larger N, where one can gain 1 bit of precision by doubling N. Nonetheless, quasi-linear growth in bootstrapping time is accompanied, and the public key size also doubles. Thus, the most efficient usage of TFHE for large precision and corresponding parameter selection is still an open problem.

1.1 Our Contributions and Technical Overview

In this paper, we propose a large precision bootstrapping algorithm for TFHE, which we name it by EBS. Compared to the previous literature of TFHE bootstrapping with large precision, our EBS can bootstrap TFHE ciphertext without enlarging the ring dimension N. Rather, we can keep N as small as possible as long as the (bootstrapping) error doesn't damage the message in the MSB part. Working with small N has lots of advantage in TFHE literature since the time complexity of bootstrapping grows quasi-linearly by N, and the public key size grows linearly. Thus, it is recommended to use small N for efficiency, and our EBS can solve that problem, while still preserving the precision.

Our EBS inherits the idea to use larger N to hold larger information of the ciphertext. Nonetheless, rather than increasing N itself, we use the fact that N is selected as a power of 2 in TFHE and its variants. We induce a homomorphism to a larger ring from dimension N to $2^\nu N$, where we call $\nu \in \mathbb{N}$ the extension factor. The homomorphism is actually a zero padding, and does not affect the security, nor the information of the ciphertext. With the homomorphism, the bits extracted from the ciphertext increases by $(\log_2 N + 1)$ bits to $(\log_2 N + \nu + 1)$ bits, and thus we can bootstrap with additional ν bits of precision.

For efficiency reasons, we also use the fact that our induced homomorphism is actually a zero padding. Thanks to the zeros, the bootstrapping in dimension $2^\nu N$ can be converted to 2^ν times of parallel bootstrappings in dimension N. The advantage in this is that we can perform the bootstrappings simultaneously, where we can save much time with proper parallelization. Also, the asymptotic complexity decreases compared to when performing the bootstrapping in dimension $2^\nu N$.

Also, we provide a proof-of-concept implementation over the TFHE library [12] along with the three variants of the state-of-the-art full-domain functional bootstrapping algorithms. With our implementation, we provide detailed noise analysis, benchmarks with eight sets of parameters with $N = 1024, 2048$ and 4096 that achieves $\lambda = 80$ or 128 bits of security. We also evaluate functions over the torus and provide detailed precision improvements with four sets of parameters. With our EBS, we show that even with $N = 1024$ and 2048, it is possible to achieve *over* 8 bits of precision, which is known to be possible with at least $N = 2^{14} = 16386$. Thus, with our EBS, we can enhance not only the exact computation of TFHE, but also the approximate computation combined with other homomorphic encryption schemes like in [30].

1.2 Related Works

High precision bootstrapping is a common problem throughout homomorphic encryption literature. For approximate homomorphic encryption schemes, the high precision bootstrapping is required to evaluate huge depth circuits such as deep neural network training and inference [25, 26], or retrieve statistical information from a dataset. Starting from the dawn of CKKS bootstrapping of 10 to 15 bit of precision [9], many optimizations and better approximations have been studied, and reached 90 to 110 bit of precision [27], or even higher precision of 420 bits which takes 903 seconds [2]. The bootstrapping in CKKS takes much longer than FHEW/TFHE style bootstrappings, but their SIMD (Single Instruction, Multiple Data) structure enables them to bootstrap multiple messages, and usually presented in terms of amortized latency.

Nonetheless, the FHEW/TFHE style bootstrapping still has its own advantages, its significantly low latency, and its capability to evaluate even nonlinear functions with lookup table evaluation. These versatility even brought bridges to approximate homomorphic encryption schemes, to evaluate polynomial functions by approximate schemes, and bring them to TFHE to evaluate nonlinear functions [5, 30]. The enhancements in the usage of FHEW/TFHE itself has also been studied throughout many works, including the extension of binary keys to general keys (ternary, Gaussian, etc.) [20, 32], or improved FHEW bootstrapping with ring automorphisms [28]. When it come to high precision TFHE, most of the works select decomposition of plaintext message [15, 19, 22, 29, 34], with low precision TFHE bootstrapping, and no algorithm was known to bootstrap a single ciphertext with large precision except using large N. Recently, Bergerat et al. [3] proposed an algorithm called $\mathsf{WoP\text{-}PBS}$ (WithOut Padding - Programmable BootStrapping), extracting each bit of the message as a RGSW ciphertext with circuit bootstrapping [10] and evaluate function with vertically packed lookup table. Together with their algorithm, they provided a detailed noise analysis of their $\mathsf{WoP\text{-}PBS}$, and a publicly available implementation[1]. It is estimated that their $\mathsf{WoP\text{-}PBS}$ has noise variance bound larger than our EBS by factor of at least $O(\kappa N)$ when they bootstrap with κ-bits of precision. Nonetheless, their time complexity is linear to κ while our EBS is exponential. This makes their bootstrapping more efficient when κ is sufficiently large. But until certain level, our EBS is more efficient as the circuit bootstrapping is itself quite costly.

2 Preliminaries

2.1 Notations

We introduce the notations used throughout this paper. The real torus $\mathbb{T}$ denotes the real set $\mathbb{R}/\mathbb{Z}$, which is also interpreted as a half open interval $\left[-\frac{1}{2}, \frac{1}{2}\right)$. Each set $\mathbb{R}_N[X]$, $\mathbb{Z}_N[X]$, and $\mathbb{T}_N[X]$ denote the set $\mathbb{R}[X]/\left\langle X^N+1\right\rangle$, $\mathbb{Z}[X]/\left\langle X^N+1\right\rangle$, and $\mathbb{T}[X]/\left\langle X^N+1\right\rangle$.

For a set $\mathcal{S}$, $x \overset{\$}{\leftarrow} \mathcal{S}$ implies that x is sampled from $\mathcal{S}$ from uniform distribution. Also, for a distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ implies that x is sampled from a distribution $\mathcal{D}$. Next, $\mathsf{Err}\left(c\right)$ represents the error in the ciphertext c, and $\mathsf{Var}\left(\mathsf{Err}\left(c\right)\right)$ denotes the variance of error of the ciphertext c. The parentheses $[\![a, b]\!]$ for $a, b \in \mathbb{Z}$ denotes the set $\{x \in \mathbb{Z} \mid a \leq x \leq b\}$. All indices starts with 0 unless mentioned otherwise.

2.2 TFHE Ciphertext

The security of TFHE is based on the hardness of the Learning with Errors (LWE) problem [35] and its ring variant, Ring-Learning with Errors (RLWE) [31, 36]. More precisely, the generalization of those problems over the real torus $\mathbb{T}$.

TLWE Let $n \in \mathbb{N}$ be the TLWE dimension, and σ_{TLWE} be the standard deviation. Then for a discrete message space $\mathcal{M} \subset \mathbb{T}$, the TLWE encryption of a message $m \in \mathcal{M}$ under the key $\mathbf{s} \in \mathbb{B}^n$ is

$$\mathsf{TLWE}_{\mathbf{s}}(m) = (\mathbf{a}, b) \in \mathbb{T}^{n+1},$$

with $\mathbf{a} \overset{\$}{\leftarrow} \mathbb{T}^n$, and $b = \langle \mathbf{a}, \mathbf{s}\rangle + m + e$ where $e \leftarrow \mathcal{N}(0, \sigma_{\mathsf{TLWE}})$. Given arbitrarily many TLWE samples with the key $\mathbf{s}$, the (torus) LWE problem [11] assures that if

[1] `https://github.com/zama-ai/tfhe-rs/blob/main/tfhe/src/core_crypto/algorithms/lwe_wopbs.rs`

it is λ-secure, it requires at least 2^λ operations to distinguish TLWE samples from uniform distribution over $\mathbb{T}^{n+1}$, or to find $\mathbf{s}$. We now denote that the parameter achieves λ-bit of security if it is λ-secure.

The decryption of a TLWE ciphertext first begins with calculating its phase

$$\varphi_{\mathbf{s}}((\mathbf{a}, b)) = b - \langle \mathbf{a}, \mathbf{s} \rangle ,$$

and round it to its closest element in $\mathcal{M}$.

TRLWE For $N = 2^\beta, k \in \mathbb{N}$, and the standard deviation σ_{TRLWE}, the TRLWE encryption of a message $m(X)$ in a discrete message space $\mathcal{M} \subset \mathbb{T}_N[X]$ under the key $\mathcal{K} \in \mathbb{B}_N[X]^k$ is

$$\mathsf{TRLWE}_{\mathcal{K}}(m(X)) = (A_0(X), \cdots, A_{k-1}(X), B(X)) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X],$$

where $A_i(X) \xleftarrow{\$} \mathbb{T}_N[X]$, $B(X) = \langle (A_0(X), \cdots, A_{k-1}(X)), \mathcal{K} \rangle + m(X) + e(X)$, with each coefficients of $e(X)$, $e_i \leftarrow \mathcal{N}(0, \sigma_{\mathsf{TRLWE}})$. The decryption of a TRLWE ciphertext rounds its phase $\varphi_{\mathcal{K}}((A_0(X), \cdots, A_{k-1}(X), B(X)))$ to the closest element in $\mathcal{M}$.

TRGSW Given an integer base $B_{\mathsf{G}} = 2^\gamma \in \mathbb{N}$ and decomposition length $\ell_{\mathsf{G}} \in \mathbb{N}$, first define the gadget matrix $\mathbf{H}$.

Definition 1 (Gadget Matrix). *For an integer base $B_{\mathsf{G}} = 2^\gamma \in \mathbb{N}$ and decomposition length $\ell_{\mathsf{G}} \in \mathbb{N}$, we call $\mathbf{H}$ the gadget matrix given as*

$$\mathbf{H} = \left(\begin{array}{c|c|c} 1/B_{\mathsf{G}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1/B_{\mathsf{G}}^{\ell_{\mathsf{G}}} & \cdots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \cdots & 1/B_{\mathsf{G}} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/B_{\mathsf{G}}^{\ell_{\mathsf{G}}} \end{array} \right) \in \mathbb{T}_N[X]^{(k+1)\ell_{\mathsf{G}} \times (k+1)}$$

The TRGSW ciphertext encrypts a integer polynomial $q(X) \in \mathbb{Z}_N[X]$. The TRGSW encryption of $q(X)$ under the key $\mathcal{K} \in \mathbb{B}_N[X]^k$ is

$$\mathsf{TRGSW}_{\mathcal{K}}(p(X)) = \mathbf{Z} + \mathbf{H} \cdot q(X) \in \mathbb{T}_N[X]^{(k+1)\ell_{\mathsf{G}} \times (k+1)},$$

where $\mathbf{Z}$ is a vector of $(k+1)\ell_{\mathsf{G}}$-$\mathsf{TRLWE}_{\mathcal{K}}(0)$'s. Chillotti et al. [11] defined an external product $\boxdot$ between $\mathsf{TRGSW}_{\mathcal{K}}(m_a)$ and $\mathsf{TRLWE}_{\mathcal{K}}(m_b)$ ciphertext, which gives

$$\mathsf{TRGSW}_{\mathcal{K}}(m_a) \boxdot \mathsf{TRLWE}_{\mathcal{K}}(m_b) = \mathsf{TRLWE}_{\mathcal{K}}(m_a \cdot m_b),$$

for $m_a \in \mathbb{Z}_N[X]$, and $m_b \in \mathbb{T}_N[X]$.

2.3 Bootstrapping in TFHE

The *bootstrapping* in TFHE is a homomorphic calculation of (discretized) phase of the TLWE ciphertext, and aims to reduce internal noise of the ciphertext. Moreover, it simultaneously evaluates a look-up table (LUT) of a function over the torus, and is also known as the *functional bootstrapping* [4], or *programmable bootstrapping* [13].

To bootstrap a ciphertext, one needs two kinds of public key, namely the bootstrapping key, and the keyswitch key. We will denote each of them as BSK, and KSK. For two secret keys $\mathbf{s} \in \mathbb{B}^n$ (TLWE key), $\mathcal{K} \in \mathbb{B}_N^k[X]$ (TRLWE, TRGSW key), the two public keys are defined as follows:

$$\mathsf{BSK} = \{\mathsf{TRGSW}_{\mathcal{K}}(\mathbf{s}_i)\}_{i \in [\![0,n-1]\!]},$$

$$\mathsf{KSK} = \left\{\mathsf{TLWE}_{\mathbf{s}}\left(\frac{\mathcal{K}_i}{B_{\mathsf{KS}}^j} \cdot k\right)\right\}_{i \in [\![0,N-1]\!], j \in [\![1,\ell_{\mathsf{KS}}]\!], k \in [\![0,B_{\mathsf{KS}}-1]\!]},$$

where $B_{\mathsf{KS}}, \ell_{\mathsf{KS}}$ is the decomposition base, and the length of the keyswitch key. Starting from $\mathsf{TLWE}_{\mathbf{s}}$ ciphertext $c = (\mathbf{a}, b)$, bootstrapping consists of four consecutive procedures.

- $\mathsf{ModSwitch}$: transforms $c = (\mathbf{a}, b) \in \mathbb{T}^{n+1}$ into $\bar{c} = (\bar{\mathbf{a}}, \bar{b}) \in \mathbb{Z}_{2N}^{n+1}$ by computing $\bar{\mathbf{a}}_i = \lfloor 2N\mathbf{a}_i \rceil$ for $i \in [\![0, n-1]\!]$, and $\bar{b} = \lfloor 2Nb \rceil$. From [11, 22], the variance after $\mathsf{ModSwitch}$ comes with

  $$\mathsf{Var}\left(\mathsf{Err}\left(\frac{\bar{\mathbf{a}}}{2N}, \frac{\bar{b}}{2N}\right)\right) \leq \mathsf{Var}\left(\mathsf{Err}\left(\mathbf{a}, b\right)\right) + \frac{n+1}{48N^2},$$

  and we denote

  $$V_{\mathsf{MS}} = \frac{n+1}{48N^2}.$$

- $\mathsf{BlindRotate}$: homomorphically rotates the (possibly noiseless) TRLWE encryption of the test polynomial $tv \in \mathbb{T}_N[X]$ by $-\bar{m} = \langle \bar{\mathbf{a}}, \mathbf{s} \rangle - \bar{b} \pmod{2N}$. This process be viewed as a function evaluator for a function $f : \mathbb{Z}_{2N} \to \mathbb{T}$ by setting $tv_i = f(i)$ for $i \in [\![0, N-1]\!]$. The rotation is done by computing the controlled MUX (CMux) n times:

  $$\mathsf{ACC} \leftarrow \mathsf{TRGSW}_{\mathcal{K}}(\mathbf{s}_i) \boxdot (X^{\bar{\mathbf{a}}_i} - 1) \cdot \mathsf{ACC} + \mathsf{ACC}.$$

  Each execution of the CMux multiplies $X^{\bar{\mathbf{a}}_i \mathbf{s}_i}$ to the accumulator with a certain level of noise growth. Thus, after the $\mathsf{BlindRotate}$, the accumulator is multiplied by $X^{\langle \bar{\mathbf{a}}, \mathbf{s} \rangle \pmod{2N}}$ blindfolded, and outputs the rotated TRLWE ciphertext $\mathsf{TRLWE}(X^{-\bar{m}} \cdot tv)$. After $\mathsf{BlindRotate}$, the variance of ACC is bounded by

  $$\begin{aligned} \mathsf{Var}\left(\mathsf{Err}\left(\mathsf{ACC}\right)\right) \leq{}& \mathsf{Var}\left(\mathsf{Err}\left(\mathsf{ACC}_{init}\right)\right) + \\ & n\left((k+1)N\ell_{\mathsf{BS}}\left(\frac{B_{\mathsf{BS}}}{2}\right)^2 \mathsf{Var}(\mathsf{Err}(\mathsf{BSK})) + \frac{1+kN}{4 \cdot B_{\mathsf{BS}}^{2\ell_{\mathsf{BS}}}}\right), \end{aligned}$$

where the result comes from [11]. Note that $B_{\mathsf{BS}}, \ell_{\mathsf{BS}}$ is the decomposition base, and the length of the bootstrapping key. Usually, we start off with a noiseless accumulator with $\mathsf{Var}\left(\mathsf{Err}\left(\mathsf{ACC}_{init}\right)\right) = 0$. We now denote

$$V_{\mathsf{BR}} = n\left((k+1)N\ell_{\mathsf{BS}}\left(\frac{B_{\mathsf{BS}}}{2}\right)^2 \mathsf{Var}(\mathsf{Err}(\mathsf{BSK})) + \frac{1+kN}{4\cdot B_{\mathsf{BS}}^{2\ell_{\mathsf{BS}}}}\right).$$

- $\mathsf{SampleExtract}$: extracts TLWE ciphertext from the rotated TRLWE accumulator ACC. Considering tv_i as the i-th coefficient of tv, it gives $\mathsf{TLWE}_{\mathcal{K}'}(tv_{\bar{m}}) = \mathsf{TLWE}_{\mathcal{K}'}(f(\bar{m}))$ (resp. $\mathsf{TLWE}_{\mathcal{K}'}(-tv_{\bar{m}-N}) = \mathsf{TLWE}_{\mathcal{K}'}(f(m \bar{-} N)))$ if $\bar{m} \in [\![0, N-1]\!]$ (resp. $\bar{m} \in [\![N, 2N-1]\!]$) under the key $\mathcal{K}' \in \mathbb{B}^{kN}$. The $\mathsf{SampleExtract}$ does not accumulate any noise to the ciphertext, so the variance maintains the same.

- $\mathsf{KeySwitch}$: converts the key $\mathcal{K}'$ of extracted $c' = \mathsf{TLWE}_{\mathcal{K}'}(m)$ into $\mathbf{s}$, and gives $c = \mathsf{TLWE}_{\mathbf{s}}(m)$ that encrypts the same message. Since the $\mathsf{KeySwitch}$ adds noise to the ciphertext, there have been attempts to remove the $\mathsf{KeySwitch}$ error by eliminating the need for $\mathsf{KeySwitch}$ by using $\mathbf{s} = \mathcal{K}'$ [22], or by moving around the $\mathsf{KeySwitch}$ before $\mathsf{BlindRotate}$ [3,6]. Here, we refer to the TLWE-to-TLWE $\mathsf{KeySwitch}$, and the error accumulation is given as

$$\mathsf{Var}\left(\mathsf{Err}\left(c\right)\right) \leq R^2 \mathsf{Var}\left(\mathsf{Err}\left(c'\right)\right) + kN\ell_{\mathsf{KS}}\mathsf{Var}(\mathsf{Err}(\mathsf{KSK})) + \frac{1}{12}kNB_{\mathsf{KS}}^{-2\ell_{\mathsf{KS}}},$$

where R is the Lipschitz constant for functional public keyswitching (in our work, $R = 1$). We now denote

$$V_{\mathsf{KS}} = kN\ell_{\mathsf{KS}}\mathsf{Var}(\mathsf{Err}(\mathsf{KSK})) + \frac{1}{12}kNB_{\mathsf{KS}}^{-2\ell_{\mathsf{KS}}}.$$

Algorithm 1 sums up the gate bootstrapping procedure from [11], mainly used to refresh the ciphertext after homomorphic operations (e.g. Homomorphic NAND). From Algorithm 1, the variance of the error of the output ciphertext c is given by

$$\mathsf{Var}\left(\mathsf{Err}\left(c\right)\right) \leq V_{\mathsf{BR}} + V_{\mathsf{KS}},$$

and we denote $V_{\mathsf{BS}} = V_{\mathsf{BR}} + V_{\mathsf{KS}}$.

3 Modified TFHE Bootstrapping

3.1 Functional Bootstrapping

Functional bootstrapping, which is the generalization of the gate bootstrapping in Algorithm 1, evaluates the LUT of the target function $f : \mathbb{T} \to \mathbb{T}$, and gives the $\mathsf{TLWE}_{\mathbf{s}}$ encryption of $f(\frac{\bar{m}}{2N})$ for $\bar{m} \in [0, N-1]$. The procedure is depicted in Algorithm 2. Here, we name it by the *half-domain* functional bootstrapping since it only uses the half of the torus $[0, \frac{1}{2})$ due to the negacyclic $\mathsf{BlindRotate}$

Algorithm 1: Gate Bootstrapping Algorithm (from [11])

```
  Input: TLWE ciphertext $(\mathbf{a}, b) \in \mathsf{TLWE}_{\mathbf{s}}\left(m \cdot \frac{1}{2}\right)$ with $m \in \mathbb{B}$
  Input: Bootstrapping key $\mathsf{BSK}$
  Input: Keyswitch key $\mathsf{KSK}$
  Output: Refreshed TLWE ciphertext $\mathsf{TLWE}_{\mathbf{s}}\left((-1)^m \cdot \mu\right)$
1 $\bar{\mathbf{a}}_i = \lfloor 2N\mathbf{a}_i \rceil$ for $i \in [\![0, n-1]\!]$, and $\bar{b} = \lfloor 2Nb \rceil$   ▷ $\mathsf{ModSwitch}\left((a, b), 2N\right)$
2 Let $tv = X^{\frac{N}{2}} \cdot (1 + X + \cdots + X^{N-1}) \cdot \mu$ for $\mu \in \mathbb{T}$
3 Let $\mathsf{ACC}_{init} = (\mathbf{0}, tv) \in \mathsf{TRLWE}_{\mathcal{K}}(tv)$
4 $\mathsf{ACC}_{\mathsf{BR}} \leftarrow \mathsf{BlindRotate}((\bar{\mathbf{a}}, \bar{b}), \mathsf{BSK}, \mathsf{ACC}_{init})$
5 $c' \leftarrow \mathsf{SampleExtract}(\mathsf{ACC}_{\mathsf{BR}})$   ▷ Extract $\mathsf{TLWE}_{\mathcal{K}}((-1)^m \cdot \mu)$
6 return $c = \mathsf{KeySwitch}(c', \mathsf{KSK})$   ▷ $\mathsf{TLWE}_{\mathbf{s}}((-1)^m \cdot \mu)$
```

Algorithm 2: Half-Domain Functional Bootstrapping (from [4, 19])

```
  Input: TLWE ciphertext $(\mathbf{a}, b) \in \mathsf{TLWE}_{\mathbf{s}}(m)$ with $m \in \left[0, \frac{1}{2}\right)$
  Input: A $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$
  Input: Bootstrapping key $\mathsf{BSK}$
  Input: Keyswitch key $\mathsf{KSK}$
  Output: Refreshed TLWE ciphertext $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\bar{m}}{2N}\right)\right)$
1 $(\bar{\mathbf{a}}, \bar{b}) = \mathsf{ModSwitch}\left((\mathbf{a}, b), 2N\right)$   ▷ $\bar{m} = \bar{b} - \langle \bar{\mathbf{a}}, \mathbf{s} \rangle \bmod 2N$
2 Let $tv = \Sigma_{i=0}^{N-1} f\left(\frac{i}{2N}\right) X^i$
3 Let $\mathsf{ACC}_{init} = (\mathbf{0}, tv) \in \mathsf{TRLWE}_{\mathcal{K}}(tv)$
4 $\mathsf{ACC}_{\mathsf{BR}} \leftarrow \mathsf{BlindRotate}((\bar{\mathbf{a}}, \bar{b}), \mathsf{BSK}, \mathsf{ACC}_{init})$
    ▷ $\mathsf{ACC}_{\mathsf{BR}} = \mathsf{TRLWE}(X^{-\bar{m}} \cdot tv)$
5 $c' \leftarrow \mathsf{SampleExtract}(\mathsf{ACC}_{\mathsf{BR}})$   ▷ Extract $\mathsf{TLWE}_{\mathcal{K}}\left(f\left(\frac{\bar{m}}{2N}\right)\right)$
6 return $c = \mathsf{KeySwitch}(c', \mathsf{KSK})$   ▷ $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\bar{m}}{2N}\right)\right)$
```

(i.e., it gives the encryption of $-f\left(\frac{\bar{m}-N}{2N}\right)$ if $\bar{m} \in [N, 2N-1]$). Thus, the domain can be naturally extended to the full torus for any negacyclic function $h(x) = -h\left(x + \frac{1}{2}\right)$.

In Proposition 1, we analyze the error of the functionally bootstrapped ciphertext with $\mathcal{L}$-Lipschitz function f evaluated on an arbitrary message $m \in [0, \frac{1}{2})$. From the result, we observe that the rounding error from the $\mathsf{ModSwitch}$ affects the value of the function itself by directly changing the message of the original ciphertext. The rounding error is thus highly related to the maximal precision a ciphertext can have, and has been pointed out to be the major reason for the severe precision loss in TFHE based applications [13, 22].

Nonetheless, this rounding error was not a serious problem for the *gate* bootstrapping since it only needed 1-bit of precision after the $\mathsf{ModSwitch}$ to assure its correctness. However, functional bootstrapping works on larger plaintext space

$\mathcal{M}$, which usually has the size of power of 2 (i.e., $|\mathcal{M}| = 2^{\pi}$). Thus, to correctly bootstrap a ciphertext $\mathsf{TLWE}_{\mathbf{s}}(m)$ with full precision π (with high probability), the errors should satisfy

$$\epsilon_{pre}, \epsilon_{\mathsf{BS}} \leq \frac{1}{2|\mathcal{M}|},$$

with high probability for pre-BootStrap and BootStrap errors $\epsilon_{pre}, \epsilon_{\mathsf{BS}}$.

Proposition 1 (Functional Bootstrapping Error) *Let c be the output of functional bootstrapping with a $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$. Then the variance of the error between $\varphi_{\mathbf{s}}(c) = f(m + \epsilon_r) + \epsilon_{\mathsf{BS}}$ and $f(m)$ is bounded by*

$$\mathsf{Var}\left(\varphi_{\mathbf{s}}(c) - f(m)\right) \leq \mathcal{L}^2 V_{\mathsf{MS}} + V_{\mathsf{BS}}.$$

Proof. During the ModSwitch, the message m is rounded into $\frac{\bar{m}}{2N} = m + \epsilon_{\mathsf{MS}}$, where ϵ_{MS} is the rounding error. Then during the BlindRotate, the function f is evaluated on $\frac{\bar{m}}{2N}$, with the BlindRotate error ϵ_{BR}. Thus, after the KeySwitch, the phase of the output ciphertext c from line 6 of Algorithm 2 will be

$$\varphi_{\mathbf{s}}(c) = f(m + \epsilon_{\mathsf{MS}}) + \epsilon_{\mathsf{BR}} + \epsilon_{\mathsf{KS}},$$

where ϵ_{KS} is the error from the KeySwitch. Then by the $\mathcal{L}$-Lipschitz condition, we have

$$\begin{aligned}
\mathsf{Var}\left(\varphi_{\mathbf{s}}(c) - f(m)\right) &\leq \mathsf{Var}\left(f(m + \epsilon_{\mathsf{MS}}) - f(m)\right) + \mathsf{Var}\left(\epsilon_{\mathsf{BR}}\right) + \mathsf{Var}\left(\epsilon_{\mathsf{KS}}\right) \\
&\leq \mathsf{Var}\left(\mathcal{L}\epsilon_{\mathsf{MS}}\right) + V_{\mathsf{BR}} + V_{\mathsf{KS}} \\
&\leq \mathcal{L}^2 \mathsf{Var}\left(\epsilon_{\mathsf{MS}}\right) + V_{\mathsf{BS}} \\
&\leq \mathcal{L}^2 V_{\mathsf{MS}} + V_{\mathsf{BS}}.
\end{aligned}$$

3.2 Large precision TFHE with functional bootstrapping

We revise the major branches of TFHE based applications which attempts to operate with large precision. The functional bootstrapping plays an essential role in all of these works, and sometimes appropriately modified to fulfill their required functionality.

Radix-based decomposition with multiple ciphertexts In this branch, the plaintext m is decomposed into several digits $(m_0, m_1, \cdots, m_d)$ of certain base(s) B, and each m_i's are encrypted as a single TLWE ciphertext. Usually, small power-of-2 integers $(2^{\pi} = 2^1, 2^2)$ are used as a base [3, 7, 11, 15, 19, 37], or decomposed by co-prime integer bases for the CRT representation [3, 24].

To collaborate with the vector of ciphertexts (i.e., addition, multiplication, function evaluation), *tree-based* [19] and *chaining* [7] method are known as two major solutions. Both methods lookup to huge LUT (encoded in multiple TRLWE ciphertexts) by applying the functional bootstrapping consecutively. The *chaining* method is known to have lower complexity and output noise compared to the *tree-based* method, but can only evaluate restricted types of function. Recently, Clet et al. [15] generalized the *chaining* method and enabled to evaluate any function by the cost of larger plaintext modulus (i.e., $3 \cdot \lceil \log_2 B \rceil + 1$ bits, or $2 \cdot \lceil \log_2 B \rceil + 1$ bits with additional bootstrapping) to work with base B.

Using larger N As the variance of the rounding error was bounded by $\mathsf{Var}\left(\epsilon_{\mathsf{MS}}\right) \leq \frac{n+1}{48N^2}$, if we double N, the bound halves [7, 22, 24, 30]. However, using large N brings superlinear growth in $\mathsf{BlindRotate}$ due to the expensive polynomial multiplication with complexity $O(N \log N)$. Moreover by doubling N, the size of the public key (e.g., BSK, KSK, etc) exactly doubles which can be a burden for both the client and the server using TFHE based applications.

Small Hamming weight $\mathsf{Ham}(\mathbf{s})$ Similar to the case of using large N, by restricting the hamming weight of $h = \mathsf{Ham}(\mathbf{s})$, the rounding error gets bounded by $\epsilon_r \leq \frac{h+1}{4N}$ [8, 24]. Nonetheless, large TLWE dimension n is required to achieve the same security level with small hamming weight compared to when using uniform binary key, worsening the performance of bootstrapping as well as its output noise.

VP-LUT evaluation and Circuit Bootstrapping Recent approach of Bergerat et al. [3] employed the TLWE-to-TRGSW *circuit bootstrapping* from Chillotti et al. [10]. For ciphertext(s) with total κ bit of precision, their method extracts a single bit TLWE encryption with κ functional bootstrappings (i.e., $m = \sum_{i=0}^{\kappa-1} m_i \cdot 2^i$ extracted into $\mathsf{TLWE}_{\mathbf{s}}\left(\frac{m_i}{2^{\kappa-i}}\right)$'s). Then the circuit bootstrapping transforms each ciphertexts into $\mathsf{TRGSW}_{\mathcal{K}}(m_i)$'s. These TRGSW ciphertexts are used to evaluate the VP-LUT (Vertical Packing LUT), which costs $\frac{2^\kappa}{N} + \log_2 N - 1$ CMux evaluations. Note that the circuit bootstrapping before the VP-LUT is a costly operation that contains multiple functional bootstrappings, and the output error of the VP-LUT evaluation can be larger than works featured above.

Aforementioned approaches can be combined together for further improvements if needed (e.g., selecting larger N to attain larger plaintext modulus for the *chaining* method). However, this can reduce the overall usability of TFHE, and should be selected with care.

3.3 Extended $\mathsf{BlindRotate}$ for Larger Precision

In this section, we introduce a strategy that can be adapted during the $\mathsf{BlindRotate}$ that allows to attain full precision a single ciphertext can have, even when using small N. In other words, our method can make the error from the $\mathsf{ModSwitch}$ quite negligible without using larger N, and enlarge the precision as long as it is affected the after-bootstrap error, ϵ_{BS}.

The main idea of our algorithm is to *crank up* the $\mathsf{BlindRotate}$ into a larger auxiliary ring dimension of $2^\nu N$ using a homomorphism, which is actually sent back to 2^ν-rings of dimension N for efficient calculation. We first investigate on how to *crank up* the $\mathsf{BlindRotate}$ into a larger space. Note that here, we use an uppercase to clarify the polynomial dimension of TRLWE ciphertext. To be specific, $\mathsf{TRLWE}^{N}_{\mathcal{K}}$ implies that each of the polynomial elements in this TRLWE ciphertext is an element of $\mathbb{T}_N[X]$, while $\mathsf{TRLWE}^{2^\nu N}_{\hat{\mathcal{K}}}$ comes from $\mathbb{T}_{2^\nu N}[X]$.

BlindRotate in larger dimension Recall that the main precision drop comes from the $\mathsf{ModSwitch}$, where the elements of $\mathsf{TLWE}_{\mathbf{s}}(m) = (\mathbf{a}, b)$ are rounded into $\mathbb{Z}_{2N}$. A simple intuition is to *pretend* we are using $2^\nu N$ instead of N for $\nu \in \mathbb{N} \cup \{0\}$, and round the coefficients into $\mathbb{Z}_{2^{\nu+1}N}$. The variance of error from the $\mathsf{ModSwitch}$ can now be written as

$$\mathsf{Var}\left(\mathsf{Err}\left(\frac{\bar{\mathbf{a}}}{2^{\nu+1}N}, \frac{\bar{b}}{2^{\nu+1}N}\right)\right) \leq \mathsf{Var}\left(\mathsf{Err}\left(\mathbf{a}, b\right)\right) + \frac{n+1}{48 \cdot 2^{2\nu} N^2},$$

where the variance decreased by $V_{\mathsf{MS}}/2^{2\nu}$. What is left is on how to evaluate the $\mathsf{BlindRotate}$ on dimension $2^\nu N$. Thus, we induce a module homomorphism $\iota : \mathbb{T}_N[X] \to \mathbb{T}_{2^\nu N}[X]$ by

$$\begin{aligned} \iota : \mathbb{T}_N[X] &\longrightarrow \mathbb{T}_{2^\nu N}[X], \\ p(x) = \sum_{i=0}^{N-1} p_i X^i &\longmapsto p_{\mathsf{ext}}(X) = \sum_{i=0}^{N-1} p_i X^{2^\nu i}, \end{aligned}$$

which is actually a zero padding. We now write the undercase ext to denote the polynomials zero-padded in a similar way. By applying ι on each torus polynomials of ciphertext, which we denote by $\iota\left(\mathsf{TRLWE}_{\mathcal{K}}(p(X))\right)$ and $\iota\left(\mathsf{TRGSW}_{\mathcal{K}}(q(X))\right)$, are also extended to

$$\iota\left(\mathsf{TRLWE}^{N}_{\mathcal{K}}(p(X))\right) = \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(p_{\mathsf{ext}}(X)) \text{ for } p(X) \in \mathbb{T}_N[X],$$

$$\iota\left(\mathsf{TRGSW}^{N}_{\mathcal{K}}(q(X))\right) = \mathsf{TRGSW}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(q_{\mathsf{ext}}(X)) \text{ for } q(X) \in \mathbb{Z}_N[X],$$

for extended key $\mathcal{K}_{\mathsf{ext}} \in \mathbb{B}_{2^\nu N}[X]$. Also, since ι does not add any noise, the noise variance of the ciphertext stays the same. The external product $\boxdot$ follows naturally

$$\begin{aligned} &\mathsf{TRGSW}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(q_{\mathsf{ext}}(X)) \boxdot \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(p(X)) \\ &\qquad = \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(p(X) \cdot q_{\mathsf{ext}}(X)). \end{aligned}$$

Thanks to the zero padding in TRGSW ciphertext, the error propagation of the external product is exactly the same with computing the external product in dimension N whether the TRLWE message $p(X)$ is an extended polynomial or not. Thus, by extending the bootstrapping key BSK with ι, we can evaluate the $\mathsf{BlindRotate}$ with reduced $\mathsf{ModSwitch}$ error with exactly same error propagation, i.e., V_{BR}. Note that the test vector of the accumulator should be generated in $\mathbb{T}_{2^\nu N}[X]$, so the accumulator is a TRLWE encryption under the key $\mathcal{K}_{\mathsf{ext}}$, but the message is not an extended torus polynomial.

After the extended $\mathsf{BlindRotate}$, the $\mathsf{SampleExtract}$ follows. However, due to the extension, the $\mathsf{SampleExtract}$ now gives $\mathsf{TLWE}_{\mathcal{K}'_{\mathsf{ext}}} = (\mathbf{a}, b) \in \mathbb{T}^{2^\nu kN} \times \mathbb{T}$. Notice that the key $\mathcal{K}'_{\mathsf{ext}} \in \mathbb{B}^{2^\nu kN}$ is just a TLWE representation of the extended key $\mathcal{K}_{\mathsf{ext}}$, and are all 0 except for the indices multiple of 2^ν. Thus we extract only the $2^\nu i$-th coefficients from $\mathbf{a}$ for $i \in [\![0, kN-1]\!]$ and attain $\mathsf{TLWE}_{\mathcal{K}'}$, which can now be keyswitched. The full Algorithm is depicted in Algorithm 3.

Algorithm 3: Large Precision Bootstrapping (without parallelization)

```
  Input: TLWE ciphertext $(\mathbf{a}, b) \in \mathsf{TLWE}_{\mathbf{s}}(m)$ with $m \in \left[0, \frac{1}{2}\right)$
  Input: extension factor $\nu \in \mathbb{N} \cup \{0\}$
  Input: A $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$
  Input: Extended Bootstrapping key $\mathsf{BSK}_{\mathsf{ext}} = \left\{\mathsf{TRGSW}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(\mathbf{s}_i)\right\}_{i \in [\![0, n-1]\!]}$
  Input: Keyswitch key KSK
  Output: Refreshed TLWE ciphertext $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\bar{m}}{2^{\nu+1}N}\right)\right)$
1 $(\bar{\mathbf{a}}, \bar{b}) = \mathsf{ModSwitch}\left((\mathbf{a}, b), 2^{\nu+1}N\right)$      ▷ $\bar{m} = \bar{b} - \langle \bar{\mathbf{a}}, \mathbf{s} \rangle \bmod 2^{\nu+1}N$
2 Let $tv = \Sigma_{i=0}^{2^\nu N-1} f\left(\frac{i}{2^{\nu+1}N}\right) X^i$
3 Let $\mathsf{ACC} = (\mathbf{0}, tv) \in \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(tv)$
4 $\mathsf{ACC}_{\mathsf{BR}} \leftarrow \mathsf{BlindRotate}((\bar{\mathbf{a}}, \bar{b}), \mathsf{BSK}_{\mathsf{ext}}, \mathsf{ACC})$      ▷ $\mathsf{ACC}_{\mathsf{BR}} = \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(X^{-\bar{m}} \cdot tv)$
5 $c' \leftarrow \mathsf{SampleExtract}(\mathsf{ACC}_{\mathsf{BR}})$
6 return $c = \mathsf{KeySwitch}(c', \mathsf{KSK})$      ▷ $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\bar{m}}{2^{\nu+1}N}\right)\right)$
```

Parallelization of extended BlindRotate Still, the extended $\mathsf{BlindRotate}$ contains lots of polynomial multiplications in dimension $2^\nu N$, which is quite costly. Therefore, we bring back the calculation of $\mathsf{BlindRotate}$ into multiple polynomial multiplications in dimension N, which can be easily parallelized. First, we introduce a module isomorphism $\tau : \mathbb{T}_{2^\nu N}[X] \to \mathbb{T}_N^{2^\nu}[X]$ defined by

$$\begin{aligned} \tau : \mathbb{T}_{2^\nu N}[X] &\to \mathbb{T}_N^{2^\nu}[X] \\ p(x) = \sum_{i=0}^{2^\nu N-1} p_i X^i &\longmapsto \left(p^{(0)}(X), \cdots, p^{(2^\nu-1)}(X)\right) \\ &\qquad = \left(\sum_{i=0}^{N-1} p_{2^\nu i} X^i, \cdots, \sum_{i=0}^{N-1} p_{2^\nu i + 2^\nu - 1} X^i\right). \end{aligned}$$

Then for $\mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}$ ciphertext encrypted under extended key $\mathcal{K}_{\mathsf{ext}}$, we apply τ on each torus polynomial elements in $\mathbb{T}_{2^\nu N}[X]$, creating $(k+1)$ vectors of torus polynomials in $\mathbb{T}_N^{2^\nu}[X]$. Due to the zero padding in $\mathcal{K}_{\mathsf{ext}}$, collecting i-th entries from the $(k+1)$ vectors naturally induces a $\mathsf{TRLWE}^N_{\mathcal{K}}$ ciphertext for $i \in [\![0, 2^\nu - 1]\!]$. We denote the whole process by $\tau\left(\mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(m)\right)$:

$$\tau\left(\mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(m)\right) = \left(\mathsf{TRLWE}^N_{\mathcal{K}}(m_0), \cdots, \mathsf{TRLWE}^N_{\mathcal{K}}(m_{2^\nu-1})\right).$$

for $m \in \mathbb{T}_{2^\nu N}[X]$ and $\tau(m) = (m_0, \cdots, m_{2^\nu - 1})$. Since τ is rearrangement of coefficients, the noise variance of TRLWE ciphertexts generated by τ is at most the noise variance of original ciphertext $\mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(m)$.

Then with two constraints that the $\mathsf{TRGSW}^N_{\mathcal{K}}(z)$ encrypts an integer $z \in \mathbb{Z}$ (which is quite common in TFHE literature) and that the $\mathsf{TRGSW}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(z)$ is extended from $\mathsf{TRGSW}^N_{\mathcal{K}}(z)$, we can perform the parallel external product on the decomposed TRLWE ciphertext by

$$\begin{aligned}
&\mathsf{TRGSW}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(z) \boxdot \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(m) \\
&\qquad \cong \Big(\mathsf{TRGSW}^N_{\mathcal{K}}(z) \boxdot \mathsf{TRLWE}^N_{\mathcal{K}}(m_0), \cdots, \\
&\qquad\qquad\qquad \mathsf{TRGSW}^N_{\mathcal{K}}(z) \boxdot \mathsf{TRLWE}^N_{\mathcal{K}}(m_{2^\nu - 1})\Big) \\
&\qquad \cong \Big(\mathsf{TRLWE}^N_{\mathcal{K}}(z \cdot m_0), \cdots, \mathsf{TRLWE}^N_{\mathcal{K}}(z \cdot m_{2^\nu - 1})\Big),
\end{aligned}$$

where each external product in dimension N can be all performed in parallel. Moreover, with the inverse mapping τ^{-1}, the output exactly maps to

$$\tau^{-1}\Big(\mathsf{TRLWE}^N_{\mathcal{K}}(z \cdot m_0), \cdots, \mathsf{TRLWE}^N_{\mathcal{K}}(z \cdot m_{2^\nu - 1})\Big) = \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(z \cdot m),$$

for $z \in \mathbb{Z}$ and $m \in \mathbb{T}_{2^\nu N}[X]$. Keeping this in mind, we now suggest a parallelized extended $\mathsf{BlindRotate}$, which we now denote as $\mathsf{ExtBlindRotate}$ in Algorithm 4. From our construction, the 2^k external products in line 6 used to compute the CMux gate can be computed in parallel.

Remark 1. In Algorithm 4, the rotation of the accumulator was represented by rotating in large dimension, and sending back to its vector of dimension N with the isomorphism τ. We used this representation for simplification, which in practice can actually be *rotated* by changing the order of polynomial vector, and rotating the polynomials.

With the parallel $\mathsf{ExtBlindRotate}$, we now present our final extended bootstrapping algorithm EBS in Algorithm 5.

Proposition 2 (EBS) *The* EBS *in Algorithm 5 with the extension factor* $\nu \in \mathbb{N} \cup \{0\}$ *allows to bootstrap a ciphertext with reduced* $\mathsf{ModSwitch}$ *error with variance* $\frac{1}{2^{2\nu}} V_{\mathsf{MS}}$. *The variance of error of the bootstrapped ciphertext is exactly same as* V_{BS}.

Proof. From line 1 and 2 of Algorithm 5, the reduced $\mathsf{ModSwitch}$ error variance naturally follows

$$\begin{aligned}
\mathsf{Var}\left(\mathsf{Err}\left(\frac{\bar{\mathbf{a}}}{2^{\nu+1}N}, \frac{\bar{b}}{2^{\nu+1}N}\right)\right) &\le \mathsf{Var}\left(\mathsf{Err}\left(\mathbf{a}, b\right)\right) + \frac{n+1}{48 \cdot 2^{2\nu} N^2} \\
&\le \mathsf{Var}\left(\mathsf{Err}\left(\mathbf{a}, b\right)\right) + \frac{V_{\mathsf{MS}}}{2^{2\nu}}.
\end{aligned}$$

```
Algorithm 4: Parallel ExtBlindRotate
  Input: $(\bar{\mathbf{a}}, \bar{b}) \in \mathbb{Z}^n_{2^{\nu+1}N} \times \mathbb{Z}_{2^{\nu+1}N}$
  Input: extension factor $\nu \in \mathbb{N} \cup \{0\}$
  Input: A $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$
  Input: Bootstrapping key $\mathsf{BSK}$
  Output: $\overrightarrow{\mathsf{ACC}}$ with $\tau^{-1}\left(\overrightarrow{\mathsf{ACC}}\right) = \mathsf{TRLWE}^{2^\nu N}_{\mathcal{K}_{\mathsf{ext}}}(X^{-\bar{b}+\langle \bar{\mathbf{a}},\mathbf{s}\rangle \bmod 2^{\nu+1}N} \cdot tv)$
1 Let $tv = \Sigma_{i=0}^{2^\nu N-1} f\left(\frac{i}{2^{\nu+1}N}\right) X^i$
2 $\overrightarrow{\mathsf{ACC}} \leftarrow \tau\left(\mathsf{TRLWE}^{2^k N}_{\mathcal{K}_{\mathsf{ext}}}(X^{-\bar{b}} \cdot tv)\right)$
3 for $i \in [\![0, n-1]\!]$ do
4     $\overrightarrow{\mathsf{RotACC}} \leftarrow \tau\left(X^{\bar{\mathbf{a}}_i} \cdot \tau^{-1}\left(\overrightarrow{\mathsf{ACC}}\right)\right)$
5     for $j \in [\![0, 2^\nu - 1]\!]$ do
6         $\overrightarrow{\mathsf{ACC}}_j \leftarrow \mathsf{BSK}_i \boxdot \left(\overrightarrow{\mathsf{RotACC}}_j - \overrightarrow{\mathsf{ACC}}_j\right) + \overrightarrow{\mathsf{ACC}}_j$   ▷ Parallel comp.
7     end
8 end
9 return $\overrightarrow{\mathsf{ACC}}$
```

Now we show the correctness of ExtBlindRotate in Algorithm 4 and analyze its error propagation. Starting from line 4 of Algorithm 4, we see that it is a rotation of $\overrightarrow{\mathsf{ACC}}$ by $\bar{\mathbf{a}}_i$ in $\mathbb{T}_{2^\nu N}[X]$ and does not add any noise since it only rearranges the coefficients.

In line 6 of Algorithm 4, the CMux gate is evaluated on each row of the accumulator using the external product. Thus if BSK_i encrypts 1, the $\bar{\mathbf{a}}_i$-rotated accumulator $\overrightarrow{\mathsf{RotACC}}$ is selected for the next accumulator. If not (i.e., if BSK_i encrypts 0), $\overrightarrow{\mathsf{ACC}}$ is selected. Thus after each i-th loop, the merged accumulator $\tau^{-1}\left(\overrightarrow{\mathsf{ACC}}\right)$ is rotated by $X^{\bar{\mathbf{a}}_i \mathbf{s}_i}$, and hence encrypts $X^{-\bar{b}+\sum_{p=0}^{i} \bar{\mathbf{a}}_i \cdot \mathbf{s}_i} \cdot tv$ for $i \in [\![0, n-1]\!]$.

For the error propagation of ExtBlindRotate, the errors only comes from the CMux evaluation. More specifically, from the decomposition of the TRLWE ciphertext for the external product, and the external product itself. Thus the error propagation for a single CMux evaluation is exactly the same as the BlindRotate error V_{BR}. After the ExtBlindRotate, the error is once more accumulated from the KeySwitch in line 5 of Algorithm 5, whose variance is same as V_{KS}. Thus the variance of error after bootstrapping is exactly bounded by V_{BS}.

Remark 2. The homomorphism ι and isomorphism τ was defined on module since $\mathbb{T}_N[X]$, $\mathbb{T}_{2^\nu N}[X]$ are not rings. Nonetheless, this can be easily associated to rings, using the isomorphism between $\mathbb{Z}_q$ and $\frac{1}{q}\mathbb{Z}_q \subset \mathbb{T}$. Thus, our method can naturally be used in ring-based TFHE bootstrapping implementations like in [24, 30, 37].

Algorithm 5: EBS

```
Input: TLWE ciphertext $(\mathbf{a}, b) \in \mathsf{TLWE}_{\mathbf{s}}(m)$ with $m \in \left[0, \frac{1}{2}\right)$
Input: extension factor $\nu \in \mathbb{N} \cup \{0\}$
Input: A $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$
Input: Bootstrapping key BSK
Input: Keyswitch key KSK
Output: Refreshed TLWE ciphertext $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\bar{m}}{2^{\nu+1}N}\right)\right)$
1 $(\bar{\mathbf{a}}, \bar{b}) = \mathsf{ModSwitch}\left((\mathbf{a}, b), 2^{\nu+1}N\right)$    ▷ $\bar{m} = \bar{b} - \langle \bar{\mathbf{a}}, \mathbf{s} \rangle \bmod 2^{\nu+1}N$
2 $\overrightarrow{\mathsf{ACC}} \leftarrow \mathsf{ExtBlindRotate}((\bar{\mathbf{a}}, \bar{b}), \nu, f, \mathsf{BSK})$
3 $c' \leftarrow \mathsf{SampleExtract}(\overrightarrow{\mathsf{ACC}}_0)$    ▷ Extract $\mathsf{TLWE}_{\mathcal{K}'}\left(f\left(\frac{\bar{m}}{2^{\nu+1}N}\right)\right)$
4 return $c = \mathsf{KeySwitch}(c', \mathsf{KSK})$    ▷ $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\bar{m}}{2^{\nu+1}N}\right)\right)$
```

3.4 Large Precision in Full Domain TFHE Bootstrapping

Recall that the aforementioned functional bootstrapping algorithms (including our EBS) only works with half domain of the torus $\left[0, \frac{1}{2}\right)$ to evaluate arbitrary function $f : \mathbb{T} \to \mathbb{T}$. These algorithms consumes 1 additional bit in front of the MSB of the message, and bootstrapping requires $p + 1$ bits of precision to successfully bootstrap messages of p bit precision.

Luckily, it is always possible to evaluate arbitrary function $f : \mathbb{T} \to \mathbb{T}$ in the *full* domain of the torus with extra operations. From the state-of-the-art full domain functional bootstrapping algorithms, we observed that our EBS can cooperate with most of these algorithms [14, 15, 24, 37], as they all contain the ModSwitch to $2N$ or N. The WoP-PBS from [3] uses the functional bootstrapping for extracting bits from the ciphertext and during the circuit bootstrapping. Nonetheless, neither the bit extraction nor the the circuit bootstrapping require high precision. As a result, even if our EBS is adaptable, there would be no need to adapt it to their method. Thus, we first briefly explain and compare three full domain bootstrapping algorithms from [15, 24, 37].

FDFB The idea of full domain bootstrapping of FDFB [24] is to select between two test vectors $p_+, p_- \in \mathbb{T}_N[X]$ based on the sign of message ct encrypts. The selection is done by public Mux evaluation, PubMux, with the external product. First, the sign of ct is first encrypted in a TRGSW-like ciphertext with the circuit bootstrapping [11] (like) procedure. We refer to it as a (T)RLev ciphertext [14], which equals to the last ℓ_{PM} rows of the TRGSW ciphertext. Note that the multiplication between a torus polynomial $p(X)$ and a TRLev encryption of $q(X) \in \mathbb{Z}_N[X]$ outputs a TRLWE encryption of $q(X) \cdot p(X)$.

The transformation to TRLev starts with ℓ_{PM}-functional bootstrappings to extract the sign from ct. Then each ciphertexts are TLWE-to-TRLWE keyswitched, which we denote it as RS. For specific information about the algorithm, refer to Algorithm 2 of [11]. This ends the conversion to TRLev, and the error in TRLev is bounded by

$$\mathsf{Var}\left(\mathsf{Err}\left(\mathsf{TRLev}(\mathsf{sign}(\mathsf{ct}))\right)\right) \leq V_{\mathsf{BS}} + V_{\mathsf{RS}},$$

and where V_{RS} denotes

$$V_{\mathsf{RS}} = n\ell_{\mathsf{RS}}\mathsf{Var}(\mathsf{Err}(\mathsf{RSK})) + \frac{1}{12}nB_{\mathsf{RS}}^{-2\ell_{\mathsf{RS}}}.$$

The TRGSW' encrypts 1 (resp. 0) if the sign of ct is positive (resp. negative). Then the evaluation of the PubMux follows by

$$\mathsf{ACC} = (\mathbf{0}, p_+ - p_-) \boxdot' \mathsf{TRLev}(\mathsf{sign}(\mathsf{ct})) + (\mathbf{0}, p_-).$$

Then ACC is initialized as a $\mathsf{TRLWE}_{\mathcal{K}}$ encryption of p_+ or p_- according to the sign of the input ciphertext ct. The error of ACC is given as

$$\mathsf{Var}\left(\mathsf{Err}\left(\mathsf{ACC}\right)\right) \leq N\ell_{\mathsf{PM}}\left(\frac{B_{\mathsf{PM}}}{2}\right)^2 (V_{\mathsf{BS}} + V_{\mathsf{RS}}) + \frac{1+kN}{4 \cdot B_{\mathsf{PM}}^{2\ell_{\mathsf{PM}}}},$$

which we will now denote it as $V_{\mathsf{FDFB-ACC}}$. What is left is to bootstrap ct with the accumulator, and the final error after bootstrap is bounded by

$$V_{\mathsf{FDFB-ACC}} + V_{\mathsf{BS}}.$$

The full algorithm of FDFB(-EBS) is shown in Algorithm 6.

TOTA The main intuition for TOTA [37] bootstrapping is the ModSwitch to $\mathbb{Z}_N$, since N is the maximal number of coefficients a test vector $tv \in \mathbb{T}_N[X]$ can hold. This makes the quadruple growth to the variance of ModSwitch (i.e., $4V_{\mathsf{MS}}$). Also, the decryption in $\mathbb{Z}_{2N}$ during BlindRotate with elements that lied in $\mathbb{Z}_N$ adds unwanted term pN to the message, for $p \in \{0,1\}$. Thus TOTA computes the pN with the sign bootstrapping. The term pN is removed by subtracting the ModSwitch-ed sign bootstrapped ciphertext, which adds additional ModSwitch error V_{MS} and the bootstrapping error V_{BS} before the final bootstrapping. After removing the pN, the ciphertext is then finally bootstrapped with the test vector encoding the function f.

To sum up, TOTA involves two bootstrappings. First bootstrapping to calculate pN with pre-bootstrapping error variance

$$\leq V_{\mathsf{ct}} + 4V_{\mathsf{MS}},$$

followed by the second bootstrapping to evaluate the function f, with pre-bootstrapping error variance

$$\leq V_{\mathsf{BS}} + V_{\mathsf{ct}} + 5V_{\mathsf{MS}}.$$

The variance of error of the output ciphertext is bounded by V_{BS}. The full algorithm of TOTA-EBS is shown in Algorithm 7. For the EBS in line 3 of Algorithm 7, the ciphertext $(\bar{\mathbf{a}}, \bar{b})$ is already in $Z_{2^{\nu+1}N}$, and we assume that ModSwitch is skipped during the EBS in Algorithm 5.

Algorithm 6: FDFB-EBS

```
   Input: TLWE ciphertext $(\mathbf{a}, b) \in \mathsf{TLWE}_{\mathbf{s}}(m)$
   Input: extension factor $\nu \in \mathbb{N} \cup \{0\}$
   Input: A $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$
   Input: Bootstrapping key $\mathsf{BSK}$
   Input: Keyswitch key $\mathsf{KSK}$
   Input: TLWE-to-TRLWE Keyswitch key $\mathsf{RSK}$
   Input: PubMux parameter $\ell_{\mathsf{PM}}, B_{\mathsf{PM}}$
   Output: Refreshed TLWE ciphertext $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\widehat{m}}{2^{\nu+1}N}\right)\right)$
 1 for $i \in [\![1, \ell_{\mathsf{PM}}]\!]$ do
 2     $(\mathbf{a}^i, b^i) \leftarrow \mathsf{EBS}\left((\mathbf{a}, b), \nu, \frac{1}{2B_{\mathsf{PM}}^i} f_{\mathsf{sign}}, \mathsf{BSK}, \mathsf{KSK}\right) + \left(\mathbf{0}, \frac{1}{2B_{\mathsf{PM}}^i}\right)$
                                   ▷ $(\mathbf{a}^i, b^i) = \mathsf{TLWE}_{\mathbf{s}}\left(\frac{\mathsf{sign}(ct)}{B_{\mathsf{PM}}^i}\right)$
 3     $\mathsf{PubACC}_i \leftarrow \mathsf{RS}_{\mathbf{s}\to\mathcal{K}}\left((\mathbf{a}^i, b^i), \mathsf{RSK}\right)$   ▷ $\mathsf{PubACC} = \mathsf{TRLev}_{\mathcal{K}}(\mathsf{sign}(\mathsf{ct}))$
 4 end
 5 $\overrightarrow{\mathsf{ACC}} \leftarrow \mathsf{PubMux}\left(\mathsf{PubACC}, f(x), -f\left(x - \frac{1}{2}\right)\right)$
                                   ▷ $\tau^{-1}\left(\overrightarrow{\mathsf{ACC}}\right) = \mathsf{TRLWE}_{\mathcal{K}_{\mathsf{ext}}}^{2^\nu N}(tv_{\mathsf{sign}(\mathsf{ct})})$
 6 for $i \in [\![0, n-1]\!]$ do
 7     $\overrightarrow{\mathsf{RotACC}} \leftarrow \tau\left(X^{\bar{\mathbf{a}}_i} \cdot \tau^{-1}\left(\overrightarrow{\mathsf{ACC}}\right)\right)$
 8     for $j \in [\![0, 2^\nu - 1]\!]$ do
 9         $\overrightarrow{\mathsf{ACC}}_j \leftarrow \mathsf{BSK}_i \boxdot \left(\overrightarrow{\mathsf{RotACC}}_j - \overrightarrow{\mathsf{ACC}}_j\right) + \overrightarrow{\mathsf{ACC}}_j$
10     end
11 end
12 $c' \leftarrow \mathsf{SampleExtract}(\overrightarrow{\mathsf{ACC}}_0)$
13 return $c = \mathsf{KeySwitch}(c', \mathsf{KSK})$
```

Comp Using the fact that every function f can be written as the sum of (pseudo) odd and even functions, the Comp method [15] decomposes a function f as the sum of odd and even functions f_g, and f_h. They aim to compute odd/even functions within 2 bootstrappings each, which can be performed in parallel, and combine them together with simple addition. In total, they can compute any function with 4 functional bootstrappings. The first bootstrapping contains pre-bootstrapping error bounded by

$$\leq V_{\mathsf{ct}} + V_{\mathsf{MS}},$$

and the second bootstrapping bounded by

$$\leq V_{\mathsf{BS}} + V_{\mathsf{MS}}.$$

Due to the addition of two ciphertexts encrypting the value of the odd/even function, the final error of the Comp method is bounded by $2V_{\mathsf{BS}}$. The full algorithm of Comp(-EBS) is shown in Algorithm 8.

```
Algorithm 7: TOTA-EBS
    Input: TLWE ciphertext $(\mathbf{a}, b) \in \mathsf{TLWE}_{\mathbf{s}}(m)$
    Input: extension factor $\nu \in \mathbb{N} \cup \{0\}$
    Input: A $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$
    Input: Bootstrapping key BSK
    Input: Keyswitch key KSK
    Output: Refreshed TLWE ciphertext $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\widehat{m}}{2^\nu N}\right)\right)$
1   $(\bar{\mathbf{a}}', \bar{b}') = \mathsf{ModSwitch}\left((\mathbf{a}, b), 2^\nu N\right)$    ▷ $\widehat{m} = \bar{b}' - \langle \bar{\mathbf{a}}', \mathbf{s} \rangle \bmod 2^\nu N$
2   $(\bar{\mathbf{a}}, \bar{b}) = \mathsf{ModRaise}_{2^\nu N \to 2^{\nu+1} N}\left((\bar{\mathbf{a}}', \bar{b}')\right)$    ▷ $\bar{m} = \widehat{m} + p2^\nu N$  (in $\mathbb{Z}_{2^{\nu+1}N}$)
3   $\mathsf{ct}_{\mathsf{sgn}} \leftarrow \mathsf{EBS}\left((\bar{\mathbf{a}}, \bar{b}), \nu, \frac{1}{4} f_{\mathsf{sign}}, \mathsf{BSK}, \mathsf{KSK}\right) + (\mathbf{0}, \frac{1}{4})$    ▷ $\mathsf{ct}_{\mathsf{sgn}} = \mathsf{TLWE}_{\mathbf{s}}\left(\frac{p}{2}\right)$
4   $(\mathbf{a}', b') \leftarrow \mathsf{ModSwitch}\left(\mathsf{ct}_{\mathsf{sgn}}, 2^{\nu+1} N\right)$    ▷ $b' - \langle \mathbf{a}', \mathbf{s} \rangle \bmod 2^{\nu+1} N = p2^\nu N$
5   $(\mathfrak{a}, \mathfrak{b}) = (\mathbf{a}', b') + (\bar{\mathbf{a}}, \bar{b})$    ▷ $\mathfrak{b} - \langle \mathfrak{a}, \mathbf{s} \rangle \bmod 2^{\nu+1} N = \widehat{m}$
6   return $c = \mathsf{EBS}\left((\mathfrak{a}, \mathfrak{b}), \nu, f, \mathsf{BSK}, \mathsf{KSK}\right)$
```

The whole comparison of three full domain bootstrapping algorithms is shown in Table 1, and the functional bootstrapping is denoted as BS, and the keyswitch as KS. Among the three full domain bootstrapping algorithms, TOTA [37] outperforms other two works in terms of number of operations needed, and also the error after the bootstrapping. However, the variance of error from the ModSwitch nearly quadraples, and quintuples compared to other two. This can be effectively mitigated by our EBS, without changing any of the structure of TOTA. Still, our EBS can also be adapted to other two methods as they all inevitably bootstrap with ModSwitch error added.

3.5 Probability of correct bootstrapping with EBS

As formerly mentioned, TFHE based applications usually works on plaintext space of $\mathbb{Z}_p$, with p an integer [15, 23, 33]. Using the isomorphism between $\mathbb{Z}_p$ and $\frac{1}{p}\mathbb{Z}_p$, the elements $m \in \mathbb{Z}_p$ is encoded as $\frac{m}{p} \in \mathbb{T}$ ($\frac{m}{2p}$ for half domain). Then, to correctly bootstrap a ciphertext, both pre-bootstrap error and the error after bootstrapping must be smaller than $\frac{1}{2p}$. For a ciphertext whose error variance is V and plaintext space $\mathbb{Z}_p$, the probability of correct decryption is estimated by

$$p\left(|\mathsf{Err}(\mathsf{ct})| \leq \frac{1}{2p}\right) = \mathtt{erf}\left(\frac{1}{2p\sqrt{2V}}\right),$$

where `erf` is the error function. Thus, starting from the half-domain EBS with extension factor ν, the probability of correct bootstrapping is given as

$$p(\mathsf{HDEBS}) \geq \mathtt{erf}\left(\frac{1}{4p\sqrt{2V_{\mathsf{ct}} + \frac{1}{2^{2\nu-1}} V_{\mathsf{MS}}}}\right) \cdot \mathtt{erf}\left(\frac{1}{4p\sqrt{2V_{\mathsf{BS}}}}\right),$$

Table 1: Comparison of 3 full-domain functional bootstrapping algorithms. Here, $V_{\mathsf{MS}} = \frac{n+1}{48N^2}$, V_{ct} is the variance of error of input ciphertext ct. The BS denotes bootstrapping, RS denotes the TLWE-to-TRLWE KeySwitch.

	FDFB [24]	**TOTA** [37]	**Comp** [15]
Pre-bootstrap error	$\boldsymbol{V_{\mathsf{ct}} + V_{\mathsf{MS}}}$	$V_{\mathsf{ct}} + 4V_{\mathsf{MS}}$ $V_{\mathsf{ct}} + V_{\mathsf{BS}} + 5V_{\mathsf{MS}}$	$\boldsymbol{V_{\mathsf{ct}} + V_{\mathsf{MS}}}$ $\boldsymbol{V_{\mathsf{BS}} + V_{\mathsf{MS}}}$
# of operations	$(\ell_{\mathsf{PM}}+1)$-BS $+\ell_{\mathsf{PM}}$-RS +1-PubMux	**2-BS**	4-BS
Error after Bootstrap	$V_{\mathsf{FDFB-ACC}} + V_{\mathsf{BS}}$	$\boldsymbol{V_{\mathsf{BS}}}$	$2V_{\mathsf{BS}}$
Parallel Computing	Partial	✗	✓
Compatible with EBS	✓	✓	✓

as for successful bootstrapping, both the pre-bootstrap error, and the after-bootstrap error must both be smaller than $\frac{1}{2p}$. Thus, for other three bootstrappings, FDFB-EBS, TOTA-EBS, Comp-EBS, we have

$$p(\mathsf{FDFB\text{-}EBS}) \geq \mathtt{erf}\left(\frac{1}{2p\sqrt{2V_{\mathsf{ct}} + \frac{1}{2^{2\nu-1}}V_{\mathsf{MS}}}}\right) \cdot \mathtt{erf}\left(\frac{1}{2p\sqrt{2V_{\mathsf{FDFB-ACC}} + 2V_{\mathsf{BS}}}}\right),$$

$$p(\mathsf{TOTA\text{-}EBS}) \geq \mathtt{erf}\left(\frac{1}{2p\sqrt{2V_{\mathsf{ct}} + \frac{1}{2^{2\nu-3}}V_{\mathsf{MS}}}}\right) \cdot \mathtt{erf}\left(\frac{1}{2p\sqrt{2V_{\mathsf{ct}} + 2V_{\mathsf{BS}} + \frac{5}{2^{2\nu-1}}V_{\mathsf{MS}}}}\right) \cdot \mathtt{erf}\left(\frac{1}{2p\sqrt{2V_{\mathsf{BS}}}}\right),$$

$$p(\mathsf{Comp\text{-}EBS}) \geq \mathtt{erf}\left(\frac{1}{2p\sqrt{2V_{\mathsf{ct}} + \frac{1}{2^{2\nu-1}}V_{\mathsf{MS}}}}\right) \cdot \mathtt{erf}\left(\frac{1}{2p\sqrt{2V_{\mathsf{BS}} + \frac{1}{2^{2\nu-1}}V_{\mathsf{MS}}}}\right)^2 \cdot \mathtt{erf}\left(\frac{1}{2p\sqrt{4V_{\mathsf{BS}}}}\right).$$

Then, the probability of failure is calculated by subtracting the success rate from 1, e.g., $p_{err}(\mathsf{HDEBS}) \leq 1 - p(\mathsf{HDEBS})$.

Remark 3. The above estimation is for when fixed-point arithmetic is used (in which is $\mathsf{IND\text{-}CPA^D}$ secure), using $\mathbb{Z}_p$ as a plaintext space. Thus, the functions are encoded in a staircase-like manner, i.e., $\sum_i f\left(\lfloor \frac{p}{2^\nu N} \cdot i \rceil\right)$ for $f : \mathbb{Z}_p \to \mathbb{Z}_p$, that works like a breakwater to prevent pre-bootstrap noise flooding out.

4 Experimental Results

We implemented our (HD)EBS along with the adaptation of EBS to three full-domain bootstrappings, FDFB-EBS, TOTA-EBS, Comp-EBS. Our implementations were built upon TFHE library [12], where the torus elements $\mathbb{T}$ are represented as 32-bit integer, $\mathbb{Z}_{2^{32}}$. Our experiments were executed with Intel i9-13900K running at 5.80GHz with 24 cores (8 performance cores, 16 efficient

```
Algorithm 8: Comp-EBS (Modular)
    Input: TLWE ciphertext $(\mathbf{a}, b) \in \mathsf{TLWE}_{\mathbf{s}}(m)$
    Input: extension factor $\nu \in \mathbb{N} \cup \{0\}$
    Input: A $\mathcal{L}$-Lipschitz morphism $f : \mathbb{T} \to \mathbb{T}$
    Input: Bootstrapping key BSK
    Input: Keyswitch key KSK
    Input: Plaintext modulus $\mathcal{P}$
    Output: Refreshed TLWE ciphertext $\mathsf{TLWE}_{\mathbf{s}}\left(f\left(\frac{\widehat{m}}{2^\nu N}\right)\right)$
1   $c_1 = \mathsf{EBS}\left((\mathbf{a}, b), \nu, \frac{1}{2\mathcal{P}} + \frac{1}{\mathcal{P}}\lfloor 2^\nu N \mathcal{P} x \rceil, \mathsf{BSK}, \mathsf{KSK}\right) - \left(\mathbf{0}, \frac{1}{2\mathcal{P}}\right)$
2   $c_2 = \mathsf{EBS}\left((\mathbf{a}, b), \nu, \frac{1}{2\mathcal{P}} + \frac{1}{4} + \frac{1}{\mathcal{P}}\lfloor 2^\nu N \mathcal{P} x \rceil, \mathsf{BSK}, \mathsf{KSK}\right) - \left(\mathbf{0}, \frac{1}{2\mathcal{P}} + \frac{1}{4}\right)$
3   $c_3 = \mathsf{EBS}\left(c_1, \nu, \frac{f(x) - f(-x - \frac{1}{\mathcal{P}})}{2}, \mathsf{BSK}, \mathsf{KSK}\right)$
4   $c_4 = \mathsf{EBS}\left(c_2, \nu, \frac{f(x) + f(-x - \frac{1}{\mathcal{P}})}{2}, \mathsf{BSK}, \mathsf{KSK}\right)$
5   return $c_3 + c_4$
```

cores) and 32 threads, 128GB RAM, and with 64-bit Ubuntu 22.04 environment. We compiled our experiment with g++ 11.3.0 with flags `-ltfhe-spqlios-fma -fopenmp -lquadmath`, using `spqlios` FFT in TFHE for fast polynomial multiplication, and multi-threading for our parallel EBS. The code we used for experiment is publicly available at `https://github.com/Stirling75/Extended-BootStrapping`.

4.1 TFHE Parameters

As the security of TFHE scheme has its roots in the hardness of (R)LWE problem, its security level is decided by the dimension of ciphertext (i.e., n, kN), and its corresponding standard deviation of errors added during encryption (i.e., $\sigma_{\mathsf{TLWE}}, \sigma_{\mathsf{TRLWE}}$). The security of TRGSW is guaranteed by the security of TRLWE, as it is a vector of TRLWE ciphertexts. We estimated the cost of attack models for various instances $(n, \sigma_{\mathsf{TLWE}})$, and $(N, \sigma_{\mathsf{TRLWE}})$ with the lattice estimator [1]. For most of our instances, the cost of the dual-hybrid attack [17] were estimated to be the cheapest.

In Table 2, we present eight TFHE parameter sets satisfying $\lambda = 80, 128$ bits of security, which implies it requires at least 2^λ operations for the attack models to succeed their attacks. As can been seen from parameter set I_1 and III_1, T(R)LWE ciphertexts with same dimension, decreasing the security parameter λ enables to use small standard deviation for the error which decreases the error after bootstrapping. Notice that for parameter sets $(\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3)$ and $(\mathrm{III}_1, \mathrm{III}_2, \mathrm{III}_3)$, we used exactly the same parameters except for the ring dimension N, to observe the effect of using larger N. Nonetheless, the native TFHE library only supports FFT of dimension 1024, and we made slight changes in their library to enable FFT on dimension 2048 and 4096 to support fast polynomial multiplication.

Table 2: TFHE parameter sets. λ indicates the security level of given parameter set.

Param Set	λ	TLWE		TRLWE			KSK		BSK	
		n	σ_{TLWE} ($\log_2$)	N	k	σ_{TRLWE} ($\log_2$)	ℓ_{KS}	B_{KS}	ℓ_{BS}	B_{BS}
I_1	80	750	-21.2	1024	1	-29.3	3	2^8	7	2^4
I_2	80	750	-21.2	2048	1	-32	3	2^8	7	2^4
I_3	80	750	-21.2	4096	1	-32	3	2^8	7	2^4
II	80	900	-25.7	2048	1	-32	5	2^6	7	2^4
III_1	128	670	-12.4	1024	1	-20.1	3	2^5	8	2^3
III_2	128	670	-12.4	2048	1	-32	3	2^5	8	2^3
III_3	128	670	-12.4	4096	1	-32	3	2^5	8	2^3
IV	128	1300	-26.1	2048	1	-32	5	2^6	7	2^4

However, this modified FFT still uses 64-bit `double` with 53 bits of precision, accumulating non-negligible noise during polynomial multiplication when $N \geq 2048$. We found this inhibits exact noise analysis for cases where polynomial multiplication over dimension $N \geq 2048$ is used. Further details on noise accumulation during FFT and polynomial multiplication can be found in Proposition 1 of [21].

We also describe FDFB parameters in Table 3. These parameters are only used for FDFB and has no effect on other bootstrapping methods. Using these parameters, FDFB runs with $\ell_{\mathsf{PM}} + 1 = 6$ (functional) bootstrappings, $\ell_{\mathsf{PM}} = 5$ RS and 1 PubMux operations.

Table 3: Parameters for RSK and PubMux for FDFB.

Parameter Set	RSK		PubMux	
	ℓ_{RS}	B_{RS}	ℓ_{PM}	B_{PM}
$\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3$	6	2^5	5	2^5
II	4	2^6		
$\mathrm{III}_1, \mathrm{III}_2, \mathrm{III}_3$	4	2^6		
IV	6	2^5		

4.2 Performance Results

With the parameters we suggested in Section 4.1, we make a thorough analysis in terms of public key size, latency, and noise.

Public Key Size Following the footsteps of [11], we measure the size of the public keys (BSK, KSK, RSK) published for homomorphic operations. First we measure the size of TLWE, TRLWE, TRGSW ciphertexts and then calculate the size of each public keys. For example, the size of TLWE ciphertext of parameter set I_1 is $(n+1) \times 32 = 24\,032$ bits ≈ 3.004 KB. Also, since KSK is composed of $N \times \ell_{\mathsf{KS}} \times B_{\mathsf{KS}}$ TLWE ciphertexts, the size of the KSK is 2.36 GB.

Likewise, we evaluate the key size for every parameter set and present the result in Figure 1. As we can observe from the result of parameter sets $(\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3)$ and $(\mathrm{III}_1, \mathrm{III}_2, \mathrm{III}_3)$, the key size doubles as N doubles. Still, with proper adjustment of parameters, we can make the public key size '*sufficiently small*' (see BSK and KSK of parameter I_1 and II) even with large N.

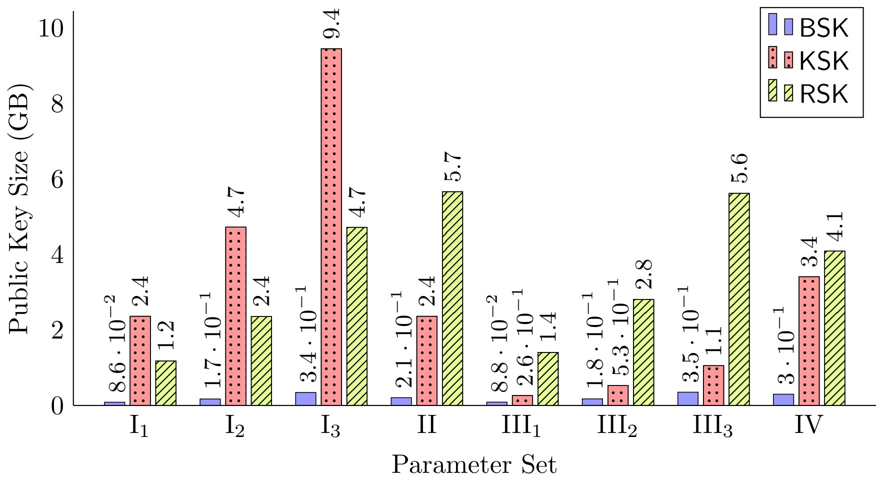

Fig. 1: Public key (BSK, KSK, RSK) sizes for our parameters.

Noise Analysis Next, we examine the variance of noise for our parameters. As our work heavily relies on the noise estimates of variety of algorithms, we found it necessary to show experimental validation of our estimations proposed in previous sections. We present our results in Table 4. We calculated (with label $^{(c)}$) the standard deviations with our variance estimations, and experimentally validated it (with label $^{(E)}$) by observing 2^{15} samples for each case. For bootstrappings, we set the test vector as identity function and calculated the standard deviations.

For ModSwitch, we calculated two standard deviations with the conventional V_{MS} formula $\frac{n+1}{48N^2}$, and with the hamming weight of the TLWE key $\mathbf{s}$, $\frac{\mathsf{Ham}(\mathbf{s})+1}{48N^2}$. Our result shows that the experimental error nearly corresponds to the standard deviation calculated with the hamming weight for all cases. For bootstrapping (including TOTA and Comp), we observe that for parameter sets I_2, I_3, II and IV, their experimental bootstrapping error are larger than expected due to the non-negligible FFT error accumulated during polynomial multiplication. Nonetheless, for parameter set III_2 and III_3, their main error standard deviations are dominated by the keyswitching error and seems to follow the estimation well. We provide detailed error analysis of BlindRotate and KeySwitch in appendix A.1, Table 5.

To only observe the output noise of each bootstrapping method, we first pre-computed the rotated amount (during BlindRotate) for given ciphertext. Then we bootstrapped the ciphertext with each bootstrapping methods, and then subtracted the rotated test vector from the accumulator, thereby eliminating the effect of the ModSwitch. From the result, FDFB shows larger noise standard deviation compared to other bootstrapping methods due to the PubMux operation. Moreover, from the result of Proposition 1, we claim that until $\mathcal{L}\sigma_{\mathsf{MS}}$, where $\mathcal{L}$ is the Lipschitz constant of the function f, is larger than the output bootstrapping noise standard deviation, there will be room for improvement with our EBS.

Table 4: Estimated noise standard deviation (with label $^{(c)}$) and experimental noise standard deviation (with label $^{(E)}$) of ModSwitch and four bootstrapping methods. The standard deviations are presented in the form of $\log_2$.

		I_1	I_2	I_3	II	III_1	III_2	III_3	IV
ModSwitch	$\sigma^{(c)}_{\mathsf{MS}}$	-8.016	-9.016	-10.016	-8.885	-8.097	-9.097	-10.097	-8.620
	$\mathsf{Ham}(\mathbf{s})$	379	379	379	448	330	330	330	645
	$\sigma^{(c,\mathsf{Ham})}_{\mathsf{MS}}$	-8.508	-9.508	-10.508	-9.387	-8.607	-9.607	-10.607	-9.125
	$\boldsymbol{\sigma}^{(E)}_{\mathsf{MS}}$	**-8.509**	**-9.506**	**-10.507**	**-9.385**	**-8.601**	**-9.591**	**-10.544**	**-9.116**
Bootstrap	$\sigma^{(c)}_{\mathsf{BS}}$	-14.411	-14.855	-14.355	-16.614	-6.000	-6.107	-5.607	-16.364
	$\boldsymbol{\sigma}^{(E,\mathsf{id})}_{\mathsf{BS}}$	**-14.882**	**-14.663**	**-14.033**	**-15.293**	**-6.369**	**-6.157**	**-5.654**	**-15.128**
FDFB	$\sigma^{(c)}_{\mathsf{FDFB}}$	-4.250	-4.194	-3.193	-5.951	4.161	4.554	5.554	-5.703
	$\boldsymbol{\sigma}^{(E,\mathsf{id})}_{\mathsf{FDFB}}$	**-10.196**	**-10.010**	**-8.524**	**-9.882**	**-2.152**	**-1.830**	**-1.750**	**-9.753**
TOTA	$\sigma^{(c)}_{\mathsf{TOTA}}$	-14.411	-14.855	-14.355	-16.614	-6.000	-6.107	-5.607	-16.364
	$\boldsymbol{\sigma}^{(E,\mathsf{id})}_{\mathsf{TOTA}}$	**-14.879**	**-14.639**	**-14.063**	**-15.628**	**-6.371**	**-6.160**	**-5.649**	**-14.992**
Comp	$\sigma^{(c)}_{\mathsf{Comp}}$	-13.911	-14.355	-13.855	-16.114	-5.500	-5.607	-5.107	-15.864
	$\boldsymbol{\sigma}^{(E,\mathsf{id})}_{\mathsf{Comp}}$	**-14.581**	**-14.460**	**-13.611**	**-14.796**	**-6.163**	**-6.149**	**-5.645**	**-14.533**

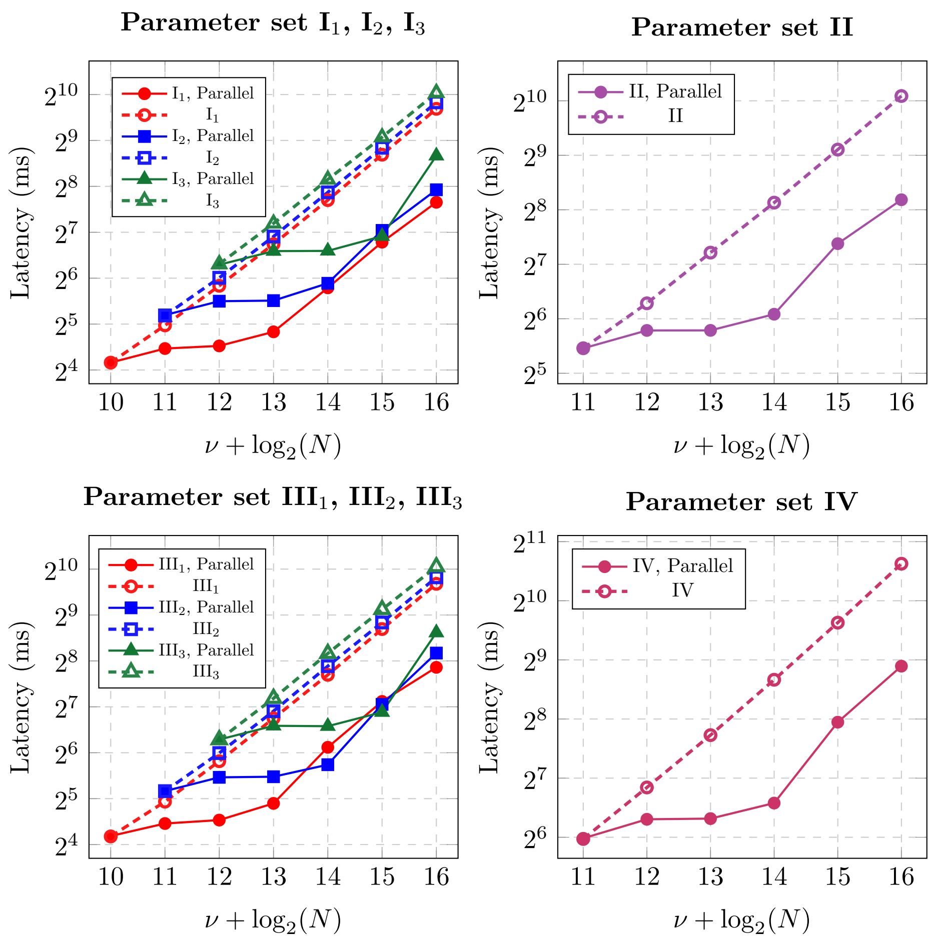

Fig. 2: Latency for EBS with eight sets of parameters given in Table 2. The x-axis represents $\nu + \log_2(N)$ for each parameter set, and the y-axis represents the latency in milliseconds in logarithmic scale of base 2.

Benchmarks We now present benchmarks for EBS, and precision growth when adapted to full-domain bootstrapping methods. We first measure the latency for computing a single conventional TFHE bootstrapping with EBS. Note that when the extension factor $\nu = 0$, the EBS becomes exactly same as original bootstrapping. From the results, we can see that for non-parallelized settings (depicted in dashed line), using $N' = 2^\nu N$ with $\nu > 0$ is slower than using EBS with dimension N and extension factor ν. Also, since the EBS increases the number of external products by the factor of 2^ν, it is easy to see the exponential increase in latency with respect to the extension factor ν. For parallelized version of EBS (depicted in solid line), we found that we lose full parallelization from

$\nu = 3$ due to computational limitations. Thus, our results show exponential growth after $\nu = 3$.

Precision We finally turn to our major contribution of EBS, the precision enhancement. For three full-domain bootstrapping methods we introduced, we measured the output noise standard deviations for two functions, the identity function (with Lipschitz constant $\mathcal{L}_1 = 1$) and $f(x) = 43\sin(\pi x/32)$ (with Lipschitz constant $\mathcal{L}_2 \approx 33.772$) by matching the torus $\left[-\frac{1}{2}, \frac{1}{2}\right)$ to $[-64, 64)$. With the three sigma rule, we measured the bit precision of the results and presented them in Figure 3 for four parameter sets I_1, II, III_1, IV. With the experimental results from Table 4 and some additional experiments, we also calculated the (ideal) maximum precision of each parameter set and depicted it as a dash-dot line in all of the figures. We noticed that the change of function (which changes the test vector) introduces noticeable changes to the maximum precision to FDFB due to their PubMux, but is quite negligible to other two full-domain bootstrapping methods.

From our results, we can see for both parameter sets I_1, II and IV, the precision improvement is significantly clear for TOTA and Comp. Nonetheless, due to the PubMux in FDFB, the maximum precision for their method is lower than other two methods. For parameter set III_1, due to its large noise standard deviation ($\sigma_{\mathsf{TRLWE}} = 2^{-20.1}$) for TRGSW ciphertext (since they achieve 128 bits of security), their output precision is quite lower than other parameter sets. In this case, it is suggested to increase the size of N and use smaller standard deviation, like in parameter set IV ($N = 2048, \sigma_{\mathsf{TRLWE}} = 2^{-32}$).

5 Conclusion

In this paper, we suggested a high precision TFHE bootstrapping algorithm EBS, which can almost remove the affect of ModSwitch during bootstrapping. The biggest advantage in our scheme is that it allows to bootstrap with large precision with small public key size compared to when enlarging N. Also, EBS can be naturally parallelized for fast computation, where no known algorithm is known to parallelize TFHE bootstrapping. Thus, we can even bootstrap much faster than previous literature of using large N. We show that our EBS is compatible with both modular, and approximate arithmetic, as well as previously known full domain bootstrapping algorithms. We also believe EBS can be one of the solution for bridging other homomorphic encryption schemes with TFHE, by allowing high precision, nonlinear function bootstrapping with small cost.

Acknowledgements This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.2022R1F1A1074291).

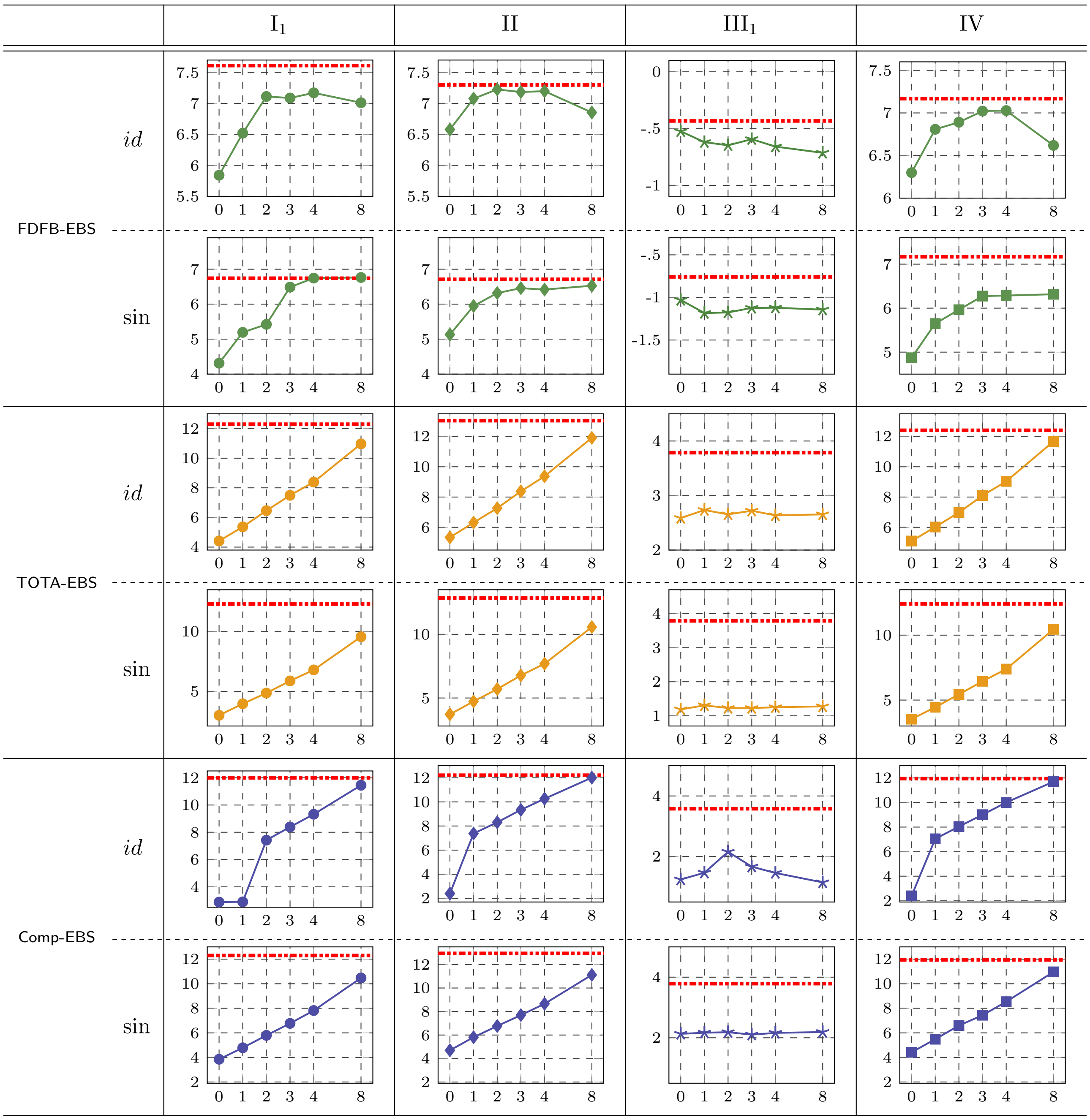

Fig. 3: Experimental output precision of three full-domain bootstrapping methods (FDFB, TOTA, Comp) with EBS evaluated with the four parameter sets I$_1$, II, III$_1$, and IV. The *id* and sin stands for the identity function and the function $f(x) = 43\sin(\pi x/32)$, by mapping the torus to $[-64, 64)$. The x-axis represents the extension factor ν, and the y-axis represents the oupput bit precision. The red dash-dot line represents the maximum precision for each parameter and bootstrapping method.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Cryptology ePrint Archive, Paper 2015/046 (2015), `https://eprint.iacr.org/2015/046`, `https://eprint.iacr.org/2015/046`
2. Bae, Y., Cheon, J.H., Cho, W., Kim, J., Kim, T.: Meta-bts: Bootstrapping precision beyond the limit. Cryptology ePrint Archive (2022)
3. Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Parameter optimization & larger precision for (t) fhe. Cryptology ePrint Archive (2022)
4. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Simulating homomorphic evaluation of deep learning predictions. In: International Symposium on Cyber Security Cryptography and Machine Learning. pp. 212–230. Springer (2019)
5. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology **14**(1), 316–338 (2020)
6. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Annual International Cryptology Conference. pp. 483–512. Springer (2018)
7. Bourse, F., Sanders, O., Traoré, J.: Improved secure integer comparison via homomorphic encryption. In: Cryptographers' Track at the RSA Conference. pp. 391–416. Springer (2020)
8. Carpov, S., Izabachène, M., Mollimard, V.: New techniques for multi-value input homomorphic evaluation and applications. In: Cryptographers' Track at the RSA Conference. pp. 106–126. Springer (2019)
9. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018)
10. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 377–408. Springer (2017)
11. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption library (August 2016), https://tfhe.github.io/tfhe/
13. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: International Symposium on Cyber Security Cryptography and Machine Learning. pp. 1–19. Springer (2021)
14. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 670–699. Springer (2021)
15. Clet, P.E., Zuber, M., Boudguiga, A., Sirdey, R., Gouy-Pailler, C.: Putting up the swiss army knife of homomorphic calculations by means of tfhe functional bootstrapping (2022), `https://eprint.iacr.org/2022/149`, `https://eprint.iacr.org/2022/149`
16. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 617–640. Springer (2015)

17. Espitau, T., Joux, A., Kharchenko, N.: On a dual/hybrid approach to small secret lwe. In: International Conference on Cryptology in India. pp. 440–462. Springer (2020)
18. Gentry, C.: A fully homomorphic encryption scheme. Stanford university (2009)
19. Guimaraes, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in tfhe. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 229–253 (2021)
20. Joye, M., Paillier, P.: Blind rotation in fully homomorphic encryption with extended keys. In: International Symposium on Cyber Security, Cryptology, and Machine Learning. pp. 1–18. Springer (2022)
21. Klemsa, J.: Fast and error-free negacyclic integer convolution using extended fourier transform. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings. pp. 282–300. Springer (2021)
22. Klemsa, J.: Setting up efficient tfhe parameters for multivalue plaintexts and multiple additions. Cryptology ePrint Archive (2021)
23. Klemsa, J., Önen, M.: Parallel operations over tfhe-encrypted multi-digit integers. In: Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy. pp. 288–299 (2022)
24. Kluczniak, K., Schild, L.: Fdfb: Full domain functional bootstrapping towards practical fully homomorphic encryption. arXiv preprint arXiv:2109.02731 (2021)
25. Lee, E., Lee, J.W., Lee, J., Kim, Y.S., Kim, Y., No, J.S., Choi, W.: Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In: International Conference on Machine Learning. pp. 12403–12422. PMLR (2022)
26. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., et al.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10**, 30039–30054 (2022)
27. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 551–580. Springer (2022)
28. Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. Cryptology ePrint Archive (2022)
29. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using fhew/tfhe bootstrapping. Cryptology ePrint Archive (2021)
30. Lu, W.j., Huang, Z., Hong, C., Ma, Y., Qu, H.: Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1057–1073. IEEE (2021)
31. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) **60**(6), 1–35 (2013)
32. Micciancio, D., Polyakov, Y.: Bootstrapping in fhew-like cryptosystems. In: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 17–28 (2021)
33. Okada, H., Kiyomoto, S., Cid, C.: Integer-wise functional bootstrapping on tfhe: Applications in secure integer arithmetics. Information **12**(8), 297 (2021)
34. Paul, J., Tan, B.H.M., Veeravalli, B., Aung, K.M.M.: Non-interactive decision trees and applications with multi-bit tfhe. Algorithms **15**(9), 333 (2022)
35. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)

36. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: 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. pp. 617–635. Springer (2009)
37. Yang, Z., Xie, X., Shen, H., Chen, S., Zhou, J.: Tota: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive (2021)

A Appendix

A.1 Noise Analysis

We present detailed analysis of noise for the BlindRotate, KeySwitch, and FDFB-ACC in Table 5. Due to the error added during polynomial multiplication (with FFT), the experimental error standard deviation for $N \geq 2048$ is larger than estimated results.

Table 5: Estimated noise standard deviation (with label $^{(c)}$) and experimental noise standard deviation (with label $^{(E)}$). The standard deviations are presented in the form of $\log_2$.

		I_1	I_2	I_3	II	III_1	III_2	III_3	IV
Bootstrap	$\sigma_{\mathsf{BR}}^{(c)}$	−14.620	−16.771	−16.271	−16.640	−6.406	−14.794	−14.295	−16.374
	$\sigma_{\mathsf{BR}}^{(E,\mathsf{id})}$	**−15.355**	**−15.539**	**−14.675**	**−15.297**	**−7.181**	**−11.539**	**−10.682**	**−15.129**
	$\sigma_{\mathsf{KS}}^{(c)}$	−15.407	−14.907	−14.407	−19.039	−6.607	−6.107	−5.607	−19.439
	$\sigma_{\mathsf{KS}}^{(E)}$	**−15.413**	**−14.910**	**−14.419**	**−19.068**	**−6.657**	**−6.157**	**−5.655**	**−19.472**
	$\sigma_{\mathsf{BS}}^{(c)}$	−14.411	−14.855	−14.355	−16.614	−6.000	−6.107	−5.607	−16.364
	$\boldsymbol{\sigma}_{\mathsf{BS}}^{(E,\mathsf{id})}$	**−14.882**	**−14.663**	**−14.033**	**−15.293**	**−6.369**	**−6.157**	**−5.654**	**−15.128**
FDFB	$\sigma_{\mathsf{ACC}}^{(c)}$	−4.250	−4.194	−3.194	−5.951	4.161	4.554	5.554	−5.703
	$\sigma_{\mathsf{ACC}}^{(E,\mathsf{id})}$	**−10.207**	**−10.004**	**−8.546**	**−9.892**	**−2.155**	**−1.838**	**−1.751**	**−9.764**
	$\sigma_{\mathsf{FDFB}}^{(c)}$	−4.250	−4.194	−3.193	−5.951	4.161	4.554	5.554	−5.703
	$\boldsymbol{\sigma}_{\mathsf{FDFB}}^{(E,\mathsf{id})}$	**−10.196**	**−10.010**	**−8.524**	**−9.882**	**−2.152**	**−1.830**	**−1.750**	**−9.753**

A.2 Benchmarks

In this section, we present the benchmark results for our parallelized and non-parallelized EBS along with the benchmarks for three full-domain bootstrapping methods. Note that none of the operations except the EBS were parallelized for fair comparison.

Table 6: Benchmark results for EBS and three full-domain bootstrapping methods. The **NP** is for non-parallelized, and **P** denotes parallelized results. All results are presented in milliseconds (ms).

		ν	I_1	I_2	I_3	II	III_1	III_2	III_3	IV
HDEBS	**NP**	0	17.88	36.44	79.28	43.98	18.08	35.84	77.88	62.72
		1	31.38	64.4	146.24	77.88	30.58	63.7	145.5	114.74
		2	57.04	120.02	284	148.42	56.32	119.88	286.42	211.9
		3	107	233.82	537.98	280.32	107.52	236.54	556.74	405.54
		4	207.98	458.46	1050.92	550.52	207.48	457.56	1061.12	792.28
		5	412.86	910.26	-	1088.24	414.1	904.76	-	1580.92
		6	825.16	-	-	-	819.8	-	-	-
	P	0	17.895	36.67	78.4	43.94	18.205	35.85	78.1	63.21
		1	22.12	45.205	96.28	55.16	21.98	44.15	96.12	79.015
		2	23.015	45.6	96.62	55.2	23.15	44.55	95.53	79.64
		3	28.465	59.235	120.6	67.8	29.79	53.42	118.055	95.545
		4	55.305	131.64	406.12	166.36	69.47	132.765	392.385	246.675
		5	110.225	243.07	-	290.92	138.865	288.21	-	475.665
		6	201.275	-	-	-	232.34	-	-	-
FDFB-EBS	**P**	0	130.22	253.78	543.57	303.67	123.02	243.8	529.18	457.89
		1	154.42	298.65	605.84	355.07	146.28	295.92	600.24	520.72
		2	157.94	307.18	639.79	365.77	150.76	294.27	626.65	532.28
		3	239.23	432.24	1016.78	478.13	193.35	443.41	957.79	821.15
		4	410.85	854.58	2561.31	1012.06	438.65	833.93	2522.98	1484.04
TOTA-EBS	**P**	0	37.98	75.1	162.64	92.84	37.15	74.9	166.33	134.13
		1	44.06	88.56	191.68	113.51	45.33	91.87	193.88	157.56
		2	46.77	91.86	196.31	113.15	46.52	91.47	196.84	159.61
		3	83.1	134.62	275.54	155.91	59.89	131.12	307.76	258.35
		4	153.99	273.6	827	330.39	139.47	268.45	822.79	474.7
Comp-EBS	**P**	0	74.67	150.25	325.02	184.9	74.12	149.96	332.31	267.24
		1	89.31	181.11	384.38	226.16	89.9	181.79	394.74	317.37
		2	93.24	184.26	391.46	225.2	93.5	182.38	392.45	318.85
		3	171.22	308.43	561.69	321.24	152.34	271.53	595.7	460.4
		4	303.01	544.15	1650.39	657.29	281.76	541.95	1669.71	964.52