# Fast and Accurate: Efficient Full-Domain Functional Bootstrap and Digit Decomposition for Homomorphic Computation

Shihe Ma[1], Tairong Huang[2], Anyu Wang[2,4,5(✉)], Qixian Zhou[3] and Xiaoyun Wang[2,4,5,6,7]

[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China, msh21@mails.tsinghua.edu.cn
[2] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China, htr19@mails.tsinghua.edu.cn,{anyuwang,xiaoyunwang}@tsinghua.edu.cn
[3] Ant Group, qixian.zqx@antgroup.com
[4] Zhongguancun Laboratory, Beijing, China
[5] National Financial Cryptography Research Center, Beijing, China
[6] Shandong Institute of Blockchain, Jinan, China
[7] Key Laboratory of Cryptologic Technology and Information Security (Ministry of Education), School of Cyber Science and Technology, Shandong University, China

**Abstract.** The functional bootstrap in FHEW/TFHE allows for fast table lookups on ciphertexts and is a powerful tool for privacy-preserving computations. However, the functional bootstrap suffers from two limitations: the negacyclic constraint of the lookup table (LUT) and the limited ability to evaluate large-precision LUTs. To overcome the first limitation, several full-domain functional bootstraps (FDFB) have been developed, enabling the evaluation of arbitrary LUTs. Meanwhile, algorithms based on homomorphic digit decomposition have been proposed to address the second limitation. Although these algorithms provide effective solutions, they are yet to be optimized. This paper presents four new FDFB algorithms and two new homomorphic decomposition algorithms that improve the state-of-the-art. Our FDFB algorithms reduce the output noise, thus allowing for more efficient and compact parameter selection. Across all parameter settings, our algorithms reduce the runtime by up to 39.2%. Our homomorphic decomposition algorithms also run at 2.0x and 1.5x the speed of prior algorithms. We have implemented and benchmarked all previous FDFB and homomorphic decomposition algorithms and our methods in OpenFHE. [1]

**Keywords:** Homomorphic Encryption · TFHE · FHEW · Functional Bootstrap · FDFB · Homomorphic Decomposition

## 1 Introduction

*Fully Homomorphic Encryption* (FHE) is a powerful cryptographic tool that enables computation on encrypted data without requiring access to the decryption key. It has great potential for use in computing fields where data privacy is important, such as secure cloud computing [KSK+18, PKS+19, LATV12] and privacy-preserving machine learning [LKL+22, BMMP18, CJP21, LHH+21], as well as in the construction of cryptographic protocols such as private set intersection [CLR17, CHLR18, CMdG+21].

Since Gentry's first construction of an FHE scheme utilizing the bootstrap technique [Gen09], various FHE schemes have been developed [FV12, BGV14, CKKS17,

[1]The code is available at https://github.com/msh086/FDFB-TCHES-2024.

GSW13, DM15, CGGI20] and significant improvements have been made [LW23a, LW23b, BIP+22, Klu22]. Among these FHE schemes, BGV/FV, CKKS and FHEW/TFHE have gained prominence recently because of their great efficiency. BGV/FV and CKKS have effective packing capabilities that allow for computations over vector data using *Single Instruction Multiple Data* (SIMD) instructions, making them ideal for simultaneously processing large arrays of numbers. However, these schemes are less efficient for evaluating deep circuits and inconvenient for evaluating non-polynomial functions. On the other hand, FHEW/TFHE utilize an efficient functional bootstrap (or programmable bootstrap) process that enables the evaluation of a *lookup table* (LUT) without additional cost, making these schemes ideal for evaluating boolean circuits and non-polynomial functions. Moreover, due to the switching method introduced in CHIMERA [BGGJ20] and later improved in PEGASUS [LHH+21], a CKKS ciphertext can be converted into multiple FHEW/TFHE ciphertexts to compute non-polynomial functions and then converted back to CKKS ciphertext for SIMD polynomial evaluation. This makes functional bootstrap a versatile tool for all FHE evaluation purposes.

Despite its strength, functional bootstrap still suffers from two limitations: (1) the evaluated LUT $f : \mathbb{Z}_p \to \mathbb{Z}_p$ must be negacyclic such that $f(x + \frac{p}{2}) = -f(x)$ for all $x \in \mathbb{Z}_p$, preventing some LUTs from being evaluated directly; (2) the input plaintext modulus $p$ is typically small due to efficiency constraints, limiting its ability to evaluate large precision LUTs. Numerous efforts have been made to address these two limitations. To circumvent the negacyclicity constraint, *Full Domain Functional Bootstrap* (FDFB) algorithms supporting arbitrary LUTs have been proposed. These FDFB algorithms can be categorized into Type-SelectMSB, Type-HalfRange and Type-Split. Type-SelectMSB selects between two negacyclic LUTs based on the *most significant bit* (MSB) of the encrypted message and is used in algorithms proposed by [CLOT21, KS22]. Type-HalfRange transforms the encrypted message to prevent it from exceeding $\frac{p}{2}$, thereby bypassing the negacyclic limitation. This method is adopted in algorithms proposed by [LMP22, YXS+21, GBA22]. Finally, Type-Split expresses an arbitrary LUT as the sum of a 'pseudo-odd' LUT and a 'pseudo-even' LUT, each of which can be evaluated using two functional bootstraps. This method is employed in the algorithm proposed by [CZB+22]. In addition to focusing on the construction of FDFB, a method for using FDFB to aid in evaluating CKKS ciphertexts is presented in [LY23]. To handle the evaluation of large-precision LUTs, Guimarães et al. [GBA21] propose tree-based and chaining methods to combine multiple functional bootstraps in TFHE. These two methods in [GBA21] assume that each ciphertext encrypts a digit of the original message. Therefore, when an input ciphertext has a large modulus, it must first be preprocessed with homomorphic decomposition before the methods can be applied. On the other hand, Liu et al. [LMP22] develop homomorphic digit decomposition algorithms and demonstrate how they can be used to evaluate large-precision sign functions. As a result, homomorphic decomposition is a crucial component in current techniques for evaluating large-precision LUTs.

In practice, functional bootstrap plays a critical role in many FHE applications, and thus its optimization is paramount for achieving high performance. Nevertheless, the efficiency of the FDFB and digit decomposition algorithms still requires further evaluation and optimization.

## 1.1 Our Contributions

This work presents new methods for optimizing the current FDFB and homomorphic decomposition algorithms. Our contributions can be summarized as follows.

(1) We present four novel FDFB algorithms: **FDFB-Compress**, **FDFB-CancelSign**, **FDFB-Select** and **FDFB-BFVMult** (**WoPPBS$_1$-Refine**). **FDFB-Compress** improves Type-HalfRange to theoretical optimality, while the other three algorithms improve Type-SelectMSB but are suitable for different scenarios. In our experiments, we observe

that our fastest algorithms demonstrate a significant speedup, ranging from 23.4% $\sim$ 39.2%, compared to the state-of-the-art results across various parameter settings.

(2) We present two new homomorphic decomposition algorithms **HomDecomp-Reduce** and **HomDecomp-FDFB**, whose running speed is 2x and 1.5x that of **HomFloor** and **HomFloorAlt** from [LMP22], respectively. Unlike **HomFloor**, our algorithms do not require the input ciphertext to have small noise. The speedup of our algorithms directly results in faster large-precision evaluations of functions such as sign, ReLU, max, ABS, etc.

(3) We provide a comprehensive theoretical noise analysis for our FDFB and homomorphic decomposition algorithms, as well as those developed by previous works. We have implemented and benchmarked all the algorithms in the OpenFHE library [BBB+22] to validate our results. Our implementation of all FDFB algorithms in a single library is a first-of-its-kind initiative, which provides standardized access to these algorithms.

## 1.2 Related Works

### 1.2.1 FDFB Algorithms

The current FDFB algorithms are summarized as follows.

**WoP-PBS**$_1$ [CLOT21] (Type-SelectMSB) introduces an extra MSB to the encrypted message by doubling the ciphertext modulus. The algorithm evaluates the LUT to obtain a ciphertext that possibly differs by a sign from the desired result. Then, it extracts the MSB using functional bootstrap and offsets the sign by invoking BFV multiplication. However, the rapid noise growth of BFV multiplication requires the algorithm to use inefficient parameters, thus degrading performance.

**WoP-PBS**$_2$ [CLOT21] (Type-SelectMSB) builds two sub-LUTs according to the MSB of the encrypted message. The algorithm evaluates both sub-LUTs to obtain two ciphertexts and extracts the MSB using functional bootstrap. Then BFV multiplication is invoked to select the correct ciphertext. Again, BFV multiplication still requires large parameters and degrades performance.

**FDFB-KS** [KS22] (Type-SelectMSB) builds two sub-LUTs similarly to **WoP-PBS**$_2$. The algorithm selects between the two sub-LUTs to obtain an encrypted LUT and then uses functional bootstrap to evaluate it. However, selecting the sub-LUTs requires multiple functional bootstraps and causes significant computational overhead.

**EvalFunc** [LMP22] (Type-HalfRange) introduces an extra MSB in a similar way to **WoP-PBS**$_1$. The algorithm extracts the MSB using functional bootstrap and cancels it to ensure that the message belongs to half of $\mathbb{Z}_p$. Then it can evaluate the LUT without being constrained by negacyclicity. We note that the **FullyFBS** of [YXS+21] and the **FDFB-C** of [GBA22] are essentially the same as **EvalFunc**.

**Comp** [CZB+22] (Type-Split) expresses an arbitrary LUT as the sum of a 'pseudo-odd' LUT and a 'pseudo-even' LUT. Then the algorithm evaluates each LUT using two functional bootstraps.

In [CIM19], Carpov et al. develop a multi-value bootstrap technique that allows several LUTs to be evaluated on the same input using a single functional bootstrap call. This technique can reduce the functional bootstraps required for **WoP-PBS**$_1$, **WoP-PBS**$_2$ and **Comp** when the parameters support multi-value bootstrap.

### 1.2.2 Homomorphic Decomposition Algorithms

The current homomorphic decomposition algorithms are summarized as follows.

**HomFloor** [LMP22] uses two bootstraps to clear the lower bits of a large-precision message before modulus switching, which prevents the modulus switching noise from corrupting the higher digits. By iteratively applying these operations, a large-precision

Table 1: A summary of the intuition behind our algorithms and the improvement over previous methods.

| Previous | Ours | Our Intuition/Improvement over Previous Works |
|---|---|---|
| **EvalFunc** [LMP22] | **FDFB-Compress** | Compress the coded message using a functional bootstrap and reduce the noise |
| $\textbf{WoP-PBS}_1$ [CLOT21]<br>$\textbf{WoP-PBS}_2$ [CLOT21] | **FDFB-CancelSign**<br>**FDFB-Select** | Replace BFV multiplication with LWE-to-RLWE packing and bootstrap |
| $\textbf{WoP-PBS}_1$ [CLOT21]<br>$\textbf{WoP-PBS}_2$ [CLOT21] | $\textbf{WoPPBS}_1$**-Refine**<br>**FDFB-BFVMult** | Use a refined noise analysis for BFV multiplication; use fewer BFV multiplications |
| **HomFloor** [LMP22]<br>**HomFloorAlt** [LMP22] | **HomDecomp-Reduce**<br>**HomDecomp-FDFB** | Reduce the range of the lower bits instead of clearing them and use fewer bootstraps |

message can be decomposed into a vector of 4-bit digits. However, this algorithm does not apply to extracted CKKS ciphertexts because it requires a small noise in the input ciphertext.

**HomFloorAlt** [LMP22] uses three bootstraps to extract the digits of a large-precision message, allowing it to support the decomposition into 5-bit digits and decompose extracted CKKS ciphertexts.

### 1.3 Overview of Our Algorithms

We present the intuition behind our algorithm design and explain how it leads to better performance (see Table 1 for a summary). The key advantage of our algorithms is their reduced noise growth, which enables us to choose more compact LWE and RLWE parameters (such as decomposition bases in blind rotation and RLWE dimension) for a given plaintext modulus, resulting in shorter running time.

**FDFB-Compress** is a Type-HalfRange FDFB algorithm. Our key observation is that the LWE message must be in a coded (and thus redundant) form $\frac{q}{p}m' + e \in \mathbb{Z}_q$ to prevent decryption failures due to errors, where $q$ is the ciphertext modulus. This enables us to design a compression function that can compress the coded LWE message into $[-\frac{q}{4}, \frac{q}{4} - 1]$ using one functional bootstrap. Then, we can perform another functional bootstrap on the compressed message to get the desired result. As a result, **FDFB-Compress** uses the same number of bootstraps as **EvalFunc** but reduces the error variance of the compressed message by half, resulting in a more compact parameter choice and better performance.

**FDFB-CancelSign**, **FDFB-Select** and **FDFB-BFVMult** ($\textbf{WoPPBS}_1$**-Refine**) are all Type-SelectMSB FDFB algorithms. The primary objective of **FDFB-CancelSign** and **FDFB-Select** is to replace the BFV multiplication in $\textbf{WoP-PBS}_1$ and $\textbf{WoP-PBS}_2$ with LWE-to-RLWE packing and an additional functional bootstrap. This approach prevents the multiplicative noise growth in BFV multiplication and instead achieves additive noise growth. As a result, although **FDFB-CancelSign** and **FDFB-Select** require an extra functional bootstrap compared to **WoP-PBS**, their slower noise growth allows for more compact parameter choices and better efficiency in most cases, according to our experiments. On the other hand, $\textbf{WoPPBS}_1$**-Refine** and **FDFB-BFVMult** are enhanced algorithms of $\textbf{WoP-PBS}_1$ and $\textbf{WoP-PBS}_2$, respectively. They significantly reduce the error growth in $\textbf{WoP-PBS}_1$ and $\textbf{WoP-PBS}_2$ by roughly N times, where N is the RLWE dimension. This is achieved through a refined noise analysis of the BFV

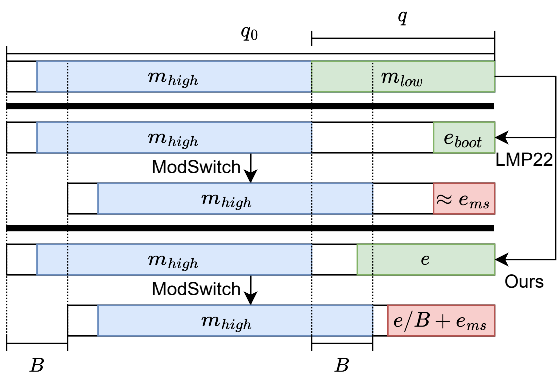


Figure 1: Comparison of our homomorphic digit decomposition approach and that of [LMP22]. The blue parts stand for higher bits, while the green and red parts stand for lower bits before and after modulus switching.

multiplication. Such an in-depth analysis allows for the choice of smaller bootstrapping parameters, resulting in enhanced efficiency. Moreover, **FDFB-BFVMult** removes one BFV multiplication in **WoP-PBS**$_2$ by combining two BFV multiplications with the sign bit into one multiplication, further reducing the noise growth by half.

The current homomorphic digit decomposition algorithms presented in [LMP22] extract digits by repeatedly clearing the lower bits $m_{low}$ of the encrypted messages (leaving a small bootstrap error) and then modulus-switching it to a smaller modulus $\frac{q_0}{B}$. We observe that this goal can also be achieved by reducing the range of the lower bits instead of clearing them. In contrast to clearing the lower bits, reducing their range consumes fewer functional bootstraps. Still, it can reserve enough room to hold the modulus switching noise, thus preventing the higher digits from being destroyed by overflowed noise. Figure 1 illustrates a comparison of these two approaches. Following this idea, we design **HomDecomp-Reduce** and **HomDecomp-FDFB**, which run 2x and 1.5x faster compared to **HomFloor** and **HomFloorAlt** in our experiments.

# 2 Preliminaries

## 2.1 Notations

The ring of integers modulo $q$ is denoted as $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. Its elements are represented as integers in either $[0, q-1]$ (positive form) or $[-\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q-1}{2} \rfloor]$ (signed form). For an integer $a$, its positive form and signed form in $\mathbb{Z}_q$ are denoted as $[a]_q^+$ and $[a]_q$, respectively.

For a power-of-2 $N$, the $2N$-th cyclotomic ring is denoted as $R = \mathbb{Z}[X]/(X^N + 1)$, and its quotient ring is denoted as $R_q = R/qR$. Polynomials are represented using bold letters, e.g., $\mathbf{a}$. For a vector $\vec{a}$ or a polynomial $\mathbf{b}$, we use $a_i$ and $\mathbf{b}_i$ respectively to denote $\vec{a}$'s $i$-th entry and $\mathbf{b}$'s coefficient of the $X^i$ term. The coefficient vector of $\mathbf{b}$ is denoted as $\vec{\mathbf{b}} = (\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{N-1})$.

For a postive interger $n$, the set $\{0, 1, \ldots, n-1\}$ is denoted as $[\![n]\!]$. We use $a \leftarrow \chi$ to represent a random variable $a$ sampled from the distribution $\chi$, and $a \leftarrow S$ to indicate that $a$ is uniformly sampled from the finite set $S$. We use $\mathcal{D}(\mathbb{Z}, \sigma)$ to denote the discrete Gaussian distribution of parameter $\sigma$ over $\mathbb{Z}$. The infinity norm and 2-norm of a vector $\vec{a}$ are denoted as $|\vec{a}|_\infty$ and $|\vec{a}|_2$ respectively. All logarithms are taken with a base of 2 unless otherwise stated.

## 2.2 FHEW/TFHE Encryption Schemes

### 2.2.1 LWE and RLWE Ciphertexts

Throughout this paper, we use lowercase $q$ and $n$ to denote the modulus and dimension of LWE instances, while uppercase $Q$ and $N$ are used for the RLWE modulus and dimension.

The LWE ciphertext encrypting an encoded message $m \in \mathbb{Z}_q$ is defined to be

$$\mathrm{LWE}_{\vec{s},n,q}(m+e) = (-\langle \vec{a}, \vec{s} \rangle + m + e, \vec{a}) \in \mathbb{Z}_q^{n+1},$$

where $\vec{a} \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \mathcal{D}(\mathbb{Z}, \sigma)$, and the secret vector $\vec{s} \leftarrow \{0, \pm 1\}^n$.

The RLWE ciphertext encrypting an encoded message $\mathbf{m} \in R_Q$ is defined to be

$$\mathrm{RLWE}_{\mathbf{s},N,Q}(\mathbf{m}+\mathbf{e}) = (-\mathbf{a} \cdot \mathbf{s} + \mathbf{m} + \mathbf{e}, \mathbf{a}) \in R_Q^2,$$

where $\mathbf{a} \leftarrow R_Q$, $\mathbf{e}_i \leftarrow \mathcal{D}(\mathbb{Z}, \sigma)$, and the secret polynomial satisfies $\mathbf{s}_i \leftarrow \{\pm 1, 0\}$.

For simplicity, we may sometimes use the abbreviated notation $\mathrm{LWE}_{\vec{s}}(m)$ and $\mathrm{RLWE}_{\mathbf{s}}(\mathbf{m})$ (or $\mathrm{LWE}(m)$ and $\mathrm{RLWE}(\mathbf{m})$) to denote the LWE and RLWE ciphertexts respectively.

Messages in LWE and RLWE ciphertexts are typically encoded to prevent decryption failures caused by errors. For instance, in an RLWE ciphertext, $\mathbf{m}$ is often an up-scaled version of the actual message $\mathbf{m}' \in R_p$, as given by $\mathbf{m} = \lfloor \frac{Q}{p}\mathbf{m}' \rceil = \frac{Q}{p}\mathbf{m}' + \mathbf{e}_{rnd}$, where $p < Q$ is the plaintext modulus and $\mathbf{e}_{rnd}$ accounts for the rounding errors. Then an RLWE ciphertext $(\mathbf{b}, \mathbf{a}) \in R_Q^2$ decrypts to $\lfloor \frac{p}{Q}(\mathbf{b} + \mathbf{a} \cdot \mathbf{s}) \rceil = \lfloor \mathbf{m}' + \frac{p}{Q}(\mathbf{e} + \mathbf{e}_{rnd}) \rceil$, which is equal to $\mathbf{m}'$ modulo $p$ as long as $|\frac{p}{Q}(\mathbf{e} + \mathbf{e}_{rnd})|_\infty < \frac{1}{2}$.

### 2.2.2 RLWE′ and RGSW Ciphertexts

An RLWE′ ciphertext is a vector of RLWE ciphertexts encrypting the same message at different scales, i.e.,

$$\mathrm{RLWE}'_{\mathbf{s}}(\mathbf{m}) = (\mathrm{RLWE}_{\mathbf{s}}(\mathbf{m}), \mathrm{RLWE}_{\mathbf{s}}(\mathbf{m} \cdot B), \ldots, \mathrm{RLWE}_{\mathbf{s}}(\mathbf{m} \cdot B^{l-1})),$$

where $B \in \mathbb{Z}$ is the decomposition base and $l = \lceil \log_B Q \rceil$. For any $\mathbf{u} \in R_Q$, there is a decomposition $\mathbf{u} = \sum_{i=0}^{l-1} \mathbf{u}_i \cdot B^i$ such that $\mathbf{u}_i$'s coefficients are all in $[-\frac{B}{2}, \frac{B}{2}]$. Let $\mathrm{Decomp}(\mathbf{u}) = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{l-1})$. Then the product $\odot : R_q \times \mathrm{RLWE}' \to \mathrm{RLWE}$ can be defined as

$$\mathbf{u} \odot \mathrm{RLWE}'_{\mathbf{s}}(\mathbf{m}) = \langle \mathrm{Decomp}(\mathbf{u}), \mathrm{RLWE}'_{\mathbf{s}}(\mathbf{m}) \rangle = \mathrm{RLWE}_{\mathbf{s}}(\mathbf{u} \cdot \mathbf{m}).$$

The obtained RLWE ciphertext contains a noise much smaller than the regular $R_q \times \mathrm{RLWE}$ multiplication due to the small coefficients of $\mathbf{u}_i$'s. Besides, the LWE′ ciphertext can be defined similarly, but we omit the details here.

An RGSW ciphertext is defined as

$$\mathrm{RGSW}_{\mathbf{s}}(\mathbf{m}) = (\mathrm{RLWE}'_{\mathbf{s}}(\mathbf{m}), \mathrm{RLWE}'_{\mathbf{s}}(\mathbf{m} \cdot \mathbf{s})).$$

Then the external product $\diamond : \mathrm{RLWE} \times \mathrm{RGSW} \to \mathrm{RLWE}$ between $(\mathbf{b}, \mathbf{a}) = \mathrm{RLWE}_{\mathbf{s}}(\mathbf{u} + \mathbf{e})$ and $\mathrm{RGSW}_{\mathbf{s}}(\mathbf{m})$ is defined as

$$(\mathbf{b}, \mathbf{a}) \diamond \mathrm{RGSW}_{\mathbf{s}}(\mathbf{m}) = \mathbf{b} \odot \mathrm{RLWE}'_{\mathbf{s}}(\mathbf{m}) + \mathbf{a} \odot \mathrm{RLWE}'_{\mathbf{s}}(\mathbf{m} \cdot \mathbf{s})),$$

which is equal to $\mathrm{RLWE}_{\mathbf{s}}((\mathbf{b} + \mathbf{a} \cdot \mathbf{s})\mathbf{m}) = \mathrm{RLWE}_{\mathbf{s}}((\mathbf{u} + \mathbf{e})\mathbf{m})$.

## 2.3 Homomorphic Operators

We introduce some basic homomorphic operations that will be used in our constructions.

### 2.3.1 Mod Down/Up and Modulus Switching

Let $c = (b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,q}(m + e)$ be an LWE ciphertext, and let $q'$ be a positive modulus.

For $q' \mid q$, the 'mod down' is defined as

$$\mathrm{ModDown}(c, q') = ([b]_{q'}, [\vec{a}]_{q'}) = \mathrm{LWE}_{\vec{s},n,q'}([m + e]_{q'}).$$

For $q \mid q'$, the 'mod up' is defined as

$$\mathrm{ModUp}(c, q') = (b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,q'}(m + e + vq),$$

where $v \in \mathbb{Z}_{q'/q}$.

For any modulus $q'$, the 'modulus switching' is defined as

$$\mathrm{ModSwitch}(c, q') = (\lfloor \frac{q'}{q} b \rceil, \lfloor \frac{q'}{q} \vec{a} \rceil) = \mathrm{LWE}_{\vec{s},n,q'}(\frac{q'}{q}(m + e) + e_{ms}),$$

where $e_{ms}$ is the noise modulus switching introduces. The three homomorphic operators described above can also be defined for RLWE ciphertexts similarly but are omitted for brevity.

### 2.3.2 Sample Extract

Given an RLWE ciphertext $c = (\mathbf{b}, \mathbf{a}) = \mathrm{RLWE}_{\mathbf{s},N,Q}(\mathbf{m} + \mathbf{e})$ and an index $i \in [\![N]\!]$, define

$$\mathrm{SampleExtract}(c, i) = \mathrm{LWE}_{\vec{\mathbf{s}},N,Q}(\mathbf{m}_i + \mathbf{e}_i),$$

which extracts the coefficient of the $X^i$ term into an LWE ciphertext.

### 2.3.3 Key Switching

Given an LWE ciphertext $c = (b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,q_{ks}}(m + e)$, a decomposition base $B_{ks}$ and key switching keys $\mathrm{ksk}_{i,j,k} = \mathrm{LWE}_{\vec{s}',n',q'}(\lfloor \frac{q'}{q_{ks}} \vec{s}_i \cdot j \cdot B_{ks}^k \rceil)$ for $i \in [\![n]\!], j \in [\![B_{ks}]\!]$ and $k \in [\![\lceil \log_{B_{ks}}(q_{ks}) \rceil]\!]$, define

$$\mathrm{KeySwitch}(c, \{\mathrm{ksk}_{i,j,k}\}) = \mathrm{LWE}_{\vec{s}',n',q'}(\lfloor \frac{q'}{q_{ks}}(m + e) \rceil + e_{ks}),$$

where $e_{ks}$ is the error key switching introduces.

Besides LWE-to-LWE key switching, it is possible to pack LWE ciphertexts into an RLWE ciphertext with similar techniques [GBA21, CZ22], which can be viewed as a specific instance of the public functional key switching method proposed in [CGGI20]. This homomorphic operator, denoted as $\mathrm{PackingKS}(\mathrm{LWE}(m), \{\mathrm{ksk}_{i,j,k}\})$, is parameterized by a positive integer $d$ and outputs $\mathrm{RLWE}(m + mX + \ldots + mX^{d-1})$. Its full definition is detailed in the full version of the paper.

### 2.3.4 Blind Rotation and Functional Bootstrap

Blind rotation is the key step in the bootstrap of FHEW/TFHE. Given an LWE ciphertext $c = \mathrm{LWE}_{\vec{s}}(m+e)$ with modulus $q|2N$, a polynomial $\mathrm{TV} \in R_Q$ (often called the *test vector*) and blind rotation keys $\{\mathrm{brk}_i^{\pm}\}$, define

$$\mathrm{BlindRotate}(c, \mathrm{TV}, \{\mathrm{brk}_i^{\pm}\}) = \mathrm{RLWE}_{\mathbf{s}'}(\mathrm{TV} \cdot X^{-\frac{2N}{q}(m+e)} + \mathbf{e}_{acc}),$$

where $\mathbf{e}_{acc}$ is the noise that blind rotation introduces. In other words, TV is rotated left by $\frac{2N}{q}(m + e)$. $\{\mathrm{brk}_i^{\pm}\}$ are parameterized by the blind-rotation base $B_g$. A smaller $B_g$

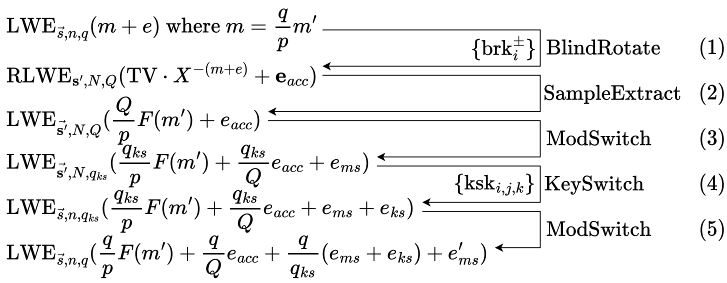


Figure 2: The five steps of FHEW/TFHE bootstrapping: (1) blind rotation of TV by the input ciphertext; (2) extracting the constant term of the rotated TV; (3) modulus switching to $q_{ks}$; (4) key switching to the original secret key; (5) modulus switching to $q$. $F$ is an LUT from $\mathbb{Z}_p$ to $\mathbb{Z}_p$.

means longer running time and smaller $e_{acc}$. Since the inner structure of blind rotation is irrelevant to the focus of this paper, we omit the details about the use of $\{\mathrm{brk}_i^{\pm}\}$. Interested readers can refer to [MP21] for more details. In this paper, we assume $q = 2N$ and omit the $\{\mathrm{brk}_i^{\pm}\}$ in notations.

Note that the constant term of the rotated TV equals $\mathrm{TV}_{m+e}$ for $m+e \in [0, N-1]$, and equals $-\mathrm{TV}_{[m+e]_N^+}$ for $m+e \in [N, 2N-1]$, then the blind rotation actually evaluates a negacyclic function $f : \mathbb{Z}_{2N} \to \mathbb{Z}_Q$ on $m+e$. To evaluate a negacyclic LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$ using blind rotation, the coefficients of TV are arranged in a redundant way to eliminate the error in input ciphertext. Specifically, by setting $\mathrm{TV}_i = \lfloor \frac{Q}{p} F(\lfloor \frac{p}{q} i \rceil) \rceil$, the constant term of $\mathrm{BlindRotate}(\mathrm{LWE}_{\vec{s}}(\lfloor \frac{q}{p} m' + e \rceil), \mathrm{TV})$ is an encryption of $\lfloor \frac{Q}{p} F(m') \rceil$.

The entire process of the functional bootstrap is illustrated in Figure 2. The noise introduced by the bootstrap process is denoted as $e_{boot}$. We use $\mathrm{Boot}[f](c)$ to represent the result of performing functional bootstrap using function $f$ on an LWE ciphertext $c$ and use $\mathrm{BootRaw}[f](c)$ to represent the freshly extracted LWE ciphertext after blind rotation (i.e., without any modulus switching or key switching). Notably, each TV uniquely corresponds to a negacyclic function $f$, so either TV or $f$ can be used to parameterize the functional bootstrap. If the plaintext polynomial TV is replaced with an RLWE ciphertext $c_{tv}$, we denote the resulting output as $\mathrm{Boot}[c_{tv}](c)$ or $\mathrm{BootRaw}[c_{tv}](c)$.

#### 2.3.5 Multi-Value Bootstrap

Multi-value bootstrap enables the evaluation of multiple LUTs on the same input LWE ciphertext with the cost of a single bootstrap [CIM19]. In this approach, the unscaled test vector is denoted as $\mathrm{TV}' \in R_p$, and the goal is to compute $\lfloor \frac{Q}{p} \mathrm{TV}' \rceil X^{-(m+e)}$, where $p$ is the plaintext modulus. To enable the computation of multiple LUTs, multi-value bootstrap decomposes $\lfloor \frac{Q}{p} \mathrm{TV}' \rceil$ approximately into $\mathrm{TV}_0 \cdot \mathrm{TV}_1$, where $\mathrm{TV}_0 = \lfloor \frac{Q}{2p} \rceil (1 + X + \ldots + X^{N-1})$ is a constant polynomial, and $\mathrm{TV}_1 = \mathrm{TV}' - \mathrm{TV}' \cdot X \in R_{2p}$ is LUT-specific. $\mathrm{TV}_0$ is first multiplied by $X^{-(m+e)}$ using blind rotation, and the resulting RLWE ciphertext is multiplied by $\mathrm{TV}_1$, which also multiplies the output error variance by $|\mathrm{TV}_1|_2^2 \leq p(p-1)^2$.

#### 2.3.6 BFV Multiplication

Let $p$ be the plaintext modulus. For two RLWE ciphertexts $c_i = \mathrm{RLWE}_{\mathbf{s},N,Q}(\frac{Q}{p}\mathbf{m}'_i + \mathbf{e}_i)$ where $i = 0, 1$, define

$$\mathrm{BFVMult}(c_0, c_1) = \mathrm{RLWE}_{\mathbf{s},N,Q}(\frac{Q}{p}\mathbf{m}'_1\mathbf{m}'_2 + \mathbf{e}_{mult}),$$

Table 2: Symbols used in our noise analysis.

| Symbol | Meaning |
|---|---|
| $\sigma^2$ | Encryption error variance |
| $\sigma^2_{ms}$ | Modulus-switching error variance |
| $\sigma^2_{ks}$ | Key-switching error variance |
| $\sigma^2_{pk}$ | PackingKS error variance, $\sigma^2_{pk} = \sigma^2_{ks}$ |
| $\sigma^2_{acc}$ | Blind-rotation error variance |
| $\sigma^2_{com}$ | Variance of noise introduced in steps (3)∼(5) in Figure 2 |
| $\sigma^2_{boot}$ | Bootstrap error variance |
| bnd, $\beta$ | $\text{bnd} = \sqrt{2} \cdot \text{erfc}^{-1}(2^{-32}) \approx 6.338$, and $\beta = \text{bnd} \cdot \sigma_{boot}$<br>For $x \sim \text{N}(0, \sigma^2_{boot})$, $\lvert x\rvert < \beta$ with high probability |
| $p$ | Plaintext modulus. $p$ is an even number |
| $q, n$ | LWE ciphertext modulus and dimension. $q$ is a power of 2 |
| $Q, N$ | RLWE ciphertext modulus and dimension. $N = 2q$ |

where $\mathbf{e}_{mult}$ is the noise of BFV multiplication. We note that re-linearization keys are required for BFV multiplication. See [KPZ21] for the detailed process.

## 2.4 Noise Introduced by the Operators

The variances of $e_{ms}, e_{ks}, e_{acc}$, $e_{boot}$ are denoted by $\sigma^2_{ms}, \sigma^2_{ks}, \sigma^2_{acc}$, $\sigma^2_{boot}$ respectively. Besides, recall that $q_{ks}$ is the key switching modulus in blind rotation. $B_{ks}$ and $B_g$ are the decomposition bases for key switching and blind rotation, respectively. The values of these variances are listed in the following lemma, and the proof can be found in [MP21].

**Lemma 1.** *Let $\sigma^2$ be the variance of the encryption noise, and $d_g = \lceil \log_{B_g} Q \rceil$, $d_{ks} = \lceil \log_{B_{ks}}(q_{ks}) \rceil$. Then*
$\sigma^2_{ms}(n) = \frac{n}{18} + \frac{1}{12}$,
$\sigma^2_{ks}(n, q_{ks}, B_{ks}) = d_{ks}(1 - \frac{1}{B_{ks}})n(\sigma^2 + \frac{1}{4})$,
$\sigma^2_{acc}(n, N, Q, B_g) = \frac{2d_g B_g^2 nN\sigma^2}{3}$,
$\sigma^2_{boot}(n, N, Q, q, B_g, q_{ks}, B_{ks}) = (\frac{q}{q_{ks}})^2(\sigma^2_{ms}(N) + \sigma^2_{ks}(N, q_{ks}, B_{ks})) + (\frac{q}{Q})^2\sigma^2_{acc}(n, N, Q, B_g) + \sigma^2_{ms}(n)$.

PackingKS introduces the same amount of noise as KeySwitch. Besides, we denote $\sigma^2_{com} = (\frac{q}{q_{ks}})^2(\sigma^2_{ms} + \sigma^2_{ks}) + \sigma^2_{ms}$ as the variance of noise introduced by the last three steps in the functional bootstrap (Figure 2).

The literature generally assumes that error introduced by homomorphic operations follows a centered normal distribution. For a centered normal variable $x \sim N(0, \sigma^2)$, its range can be bounded by $\Pr[|x| > \text{bnd} \cdot \sigma] < 2^{-32}$, where $\text{bnd} = \sqrt{2} \cdot \text{erfc}^{-1}(2^{-32}) \approx 6.338$. We denote the bound of bootstrapping error as $\beta = \text{bnd} \cdot \sigma_{boot}$. Table 2 summarizes the symbols used in our noise analysis.

# 3 Improved FDFB Algorithms

This section introduces four new FDFB algorithms. We assume that the plaintext modulus $p$ is a power of 2 for better presentation. Notably, changing $p$ to any even number will not affect the correctness or efficiency of the algorithms presented because, as we will see later, the advantage of our algorithms comes from their slow noise growth, whose correctness is independent of the choice of $p$. We assume the ciphertext modulus $q = 2N$ is a power of 2 and view the message as an integer modulo $q$ in the positive form. For an LWE ciphertext $c$ encrypting $m = \frac{q}{p}m' + e$, we add $\frac{q}{2p}$ to $c$ before performing any operations to ensure that $e + \frac{q}{2p} \in [0, \frac{q}{p} - 1]$. This will simplify the understanding of homomorphic digit decomposition algorithms in Section 4 and is consistent with [LMP22]. To keep the description of the FDFB algorithms concise, we focus on input arguments like the LUT $F$ and the input LWE ciphertext, omitting other arguments like the bootstrap key. In our noise analysis, we assume that the input ciphertext of the FDFB algorithms has an error variance of $\sigma^2_{boot}$ as in [LMP22]. The proof of correctness and noise analysis of the FDFB algorithms is provided in the full version of the paper.

## 3.1 FDFB-Compress

This algorithm employs the Type-HalfRange strategy. Specifically, it first compresses the coded message $\frac{q}{p}m' + e \in \mathbb{Z}_q$ into the range $[-\frac{q}{4}, \frac{q}{4} - 1]$ by evaluating the negacyclic function $f_C(x) : \mathbb{Z}_q \to \mathbb{Z}_q$ via a functional bootstrap, where

$$f_C(x) = \begin{cases} \frac{q}{2p}(\lfloor \frac{p}{q}x \rfloor + \frac{1}{2}) & x \in [0, \frac{q}{2} - 1] \\ -\frac{q}{2p}(\lfloor \frac{p}{q}x \rfloor - \frac{p}{2} + \frac{1}{2}) & x \in [\frac{q}{2}, q - 1] \end{cases}. \tag{1}$$

The design of $f_C$ serves two purposes. Firstly, it maps messages encoding the same $m'$ to the same value. Secondly, it ensures that the outputs of $f_C$ for different $m'$s are at least $\frac{q}{2p}$ apart. $\frac{q}{2p}$ must be greater than $2\beta$ to prevent the bootstrapping noise from interfering with the compressed message. In other words, the plaintext modulus $p$ is upper bounded by $p < \frac{q}{4\beta}$.

After compression, it is possible to bypass the negacyclicity constraint and evaluate an arbitrary LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$ on the compressed message by using one functional bootstrap to compute $f_{eval} : \mathbb{Z}_q \to \mathbb{Z}_q$, which is defined as

$$f_{eval}(x) = \begin{cases} \lfloor \frac{q}{p} F(\lfloor \frac{2p}{q}x \rfloor) \rceil & x \in [0, \frac{q}{4} - 1] \\ \lfloor \frac{q}{p} F(\lfloor \frac{2p}{q}(q - x) \rfloor + \frac{p}{2}) \rceil & x \in [\frac{3q}{4}, q - 1] \\ -f_{eval}(x - \frac{q}{2}) & x \in [\frac{q}{4}, \frac{3q}{4} - 1] \end{cases}. \tag{2}$$

The algorithm for **FDFB-Compress** is fully described in Algorithm 1, with its parameter requirements and noise analysis provided in Theorem 1.

**Algorithm 1: FDFB-Compress**

**input** : Plaintext modulus $p$ and an LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$
**input** : An LWE ciphertext $(b, \vec{a}) = \text{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)$
**output**: An LWE ciphertext $\text{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e_{boot})$

1 ct $\leftarrow$ Boot$[f_C]((b + \frac{q}{2p}, \vec{a}))$
2 **return** Boot$[f_{eval}]$(ct)

**Theorem 1.** *Suppose $\beta < \frac{q}{4p}$ and $|e| < \frac{q}{2p}$, then **FDFB-Compress**$(F, \text{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)) = \text{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e_{boot})$ and* ct *in line 1 of Algorithm 1 has an error variance of $\sigma^2_{boot}$.*

## 3.2 FDFB-CancelSign

This algorithm employs the Type-SelectMSB strategy. Given $\text{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}m' + e)$, **FDFB-CancelSign** first executes ModUp to obtain a ciphertext $\text{LWE}_{\vec{s},n,q}(\frac{q}{2}\text{MSB} + \frac{q}{2p}m' + e)$

and then performs a raw functional bootstrap to evaluate

$$f_{cs} = \begin{cases} \lfloor \frac{Q}{p} F(\lfloor \frac{2p}{q} x \rfloor) \rceil & x \in [0, \frac{q}{2} - 1] \\ -f_{cs}(x - \frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_Q \tag{3}$$

and obtain a ciphertext encrypting $(-1)^{\mathrm{MSB}} \lfloor \frac{Q}{p} F(m') \rceil$. Finally, an LWE-to-RLWE packing key switching and another functional bootstrap cancel the extra $(-1)^{\mathrm{MSB}}$ factor. The algorithm for **FDFB-CancelSign** is fully described in Algorithm 2, and its parameter requirements and noise analysis are given in Theorem 2.

**Algorithm 2: FDFB-CancelSign**

**input** : Plaintext modulus $p$ and an LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$
**input** : Base $B_{pk}$ and modulus $q_{pk}$ for PackingKS
**input** : $\{\mathrm{ksk}'_{i,j,k}\}$, packing keys for PackingKS with $d = N$
**input** : An LWE ciphertext $(b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p} m' + e)$
**output**: An LWE ciphertext $\mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p} F(m') + e')$

```
ct ← ModUp((b + q/(4p), a⃗), q)
ct_1 ← BootRaw[f_cs](ct)
ct_pk ← PackingKS(ct_1, {ksk'_{i,j,k}})
return Boot[ct_pk](ct)
```

**Theorem 2.** *Suppose* $|e| < \frac{q}{4p}$ *and* $|e'| < \frac{q}{4p}$, *then* ***FDFB-CancelSign***$(F, \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p} m' + e)) = \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p} F(m') + e')$. *The output error* $e'$ *has a variance of* $(\frac{q}{Q})^2 (2\sigma_{acc}^2 + \sigma_{pk}^2) + (\frac{q}{q_{pk}})^2 \sigma_{ms}^2 + \sigma_{core}^2$*.ciphertext.*

## 3.3 FDFB-Select

This algorithm employs the Type-SelectMSB strategy but does not perform the ModUp operation as in **FDFB-CancelSign**. In particular, let $F : \mathbb{Z}_p \to \mathbb{Z}_p$ be an arbitrary LUT, let $\mathrm{ct} = \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p} m' + e)$ be a ciphertext encrypting $m'$, and let MSB be the most significant bit of $m'$. **FDFB-Select** first constructs two sub-LUTs from $\mathbb{Z}_{p/2}$ to $\mathbb{Z}_p$, which correspond to the LUT $F$ with $\mathrm{MSB} = 0$ or $\mathrm{MSB} = 1$ respectively. These two sub-LUTs can be extended to $F_0, F_1 : \mathbb{Z}_p \to \mathbb{Z}_p$ to fulfill the negacyclic constraint. i.e., $F_0(x) = F(x)$ and $F_1(x) = -F(x + p/2)$ for $x \in [0, p/2)$, $F_0(x) = -F(x - p/2)$ and $F_1(x) = F(x)$ for $x \in [p/2, p)$. $F_0$ and $F_1$ correspond to the functions in (4) and (5).

$$f_{pos} = \begin{cases} \lfloor \frac{Q}{p} F(\lfloor \frac{p}{q} x \rfloor) \rceil & x \in [0, \frac{q}{2} - 1] \\ -f_{pos}(x - \frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_Q, \tag{4}$$

$$f_{neg} = \begin{cases} -f_{neg}(x + \frac{q}{2}) & x \in [0, \frac{q}{2} - 1] \\ \lfloor \frac{Q}{p} F(\lfloor \frac{p}{q} x \rfloor) \rceil & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_Q. \tag{5}$$

By evaluating these two functions on $\mathrm{ct} + \frac{q}{2p}$ using a single functional bootstrap each, we can obtain two ciphertexts that encrypt $F_0(m')$ and $F_1(m')$, respectively. Additionally, we can obtain a ciphertext encrypting MSB by evaluating function (6) on $\mathrm{ct} + \frac{q}{2p}$ using a single functional bootstrap.

$$f_{sgn} = \begin{cases} \frac{q}{8} & x \in [0, \frac{q}{2} - 1] \\ -\frac{q}{8} & x \in [\frac{q}{2}, q] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_q. \tag{6}$$

Finally, we use the encryption of MSB to select $F_{\mathrm{MSB}}(m')$ from $F_i(m')$ by a single functional bootstrap. The algorithm for **FDFB-Select** is fully described in Algorithm 3, and its parameter requirements and noise analysis are given in Theorem 3.

The first three functional bootstraps have the same input ciphertext $ct$, thus can be accomplished via a single multi-value bootstrap at the cost of increased noise growth. Therefore, when the parameter settings enable multi-value bootstrap, **FDFB-Select** needs only two functional bootstraps, otherwise it requires four functional bootstraps. In case multi-value bootstrap is unavailable, we develop a variant of **FDFB-Select**, called **FDFB-SelectAlt**, described in Algorithm 4, which uses only three bootstraps. The parameter requirements and noise analysis of **FDFB-SelectAlt** are given in Theorem 4.

**Remark.** We actually use an improved version of the base-aware LWE-to-RLWE packing proposed by [GBA21] to pack $\mathrm{ct}_{pos}$ and $-\mathrm{ct}_{neg}$ into $\mathrm{ct}_{pk}$. To pack $M|N$ messages $\mathrm{LWE}(m_i)$ into $\mathrm{RLWE}(\sum_{i=0}^{M-1} m_i(1+X+X^2+\ldots+X^{\frac{N}{M}-1})X^{\frac{N}{M}i})$, [GBA21] generates $M$ key switching keys, with each key corresponding to an index $i \in [\![M]\!]$. However, we observe that generating the key switching key for $i=0$ is sufficient since the keys for $i \neq 0$ can be obtained by multiplying the key for $i=0$ by $X^{\frac{N}{M}i}$. The storage cost of this optimized version of PackingKS is only $\frac{1}{M}$ that of [GBA21].

**Algorithm 3: FDFB-Select**

**input** : Plaintext modulus $p$ and an LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$
**input** : Base $B_{pk}$ and modulus $q_{pk}$ for PackingKS
**input** : $\{\mathrm{ksk}'_{i,j,k}\}$, packing keys for PackingKS with $d = \frac{N}{2}$
**input** : An LWE ciphertext $(b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)$
**output** : An LWE ciphertext $\mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$

$\mathtt{ct} \leftarrow (b + \frac{q}{2p}, \vec{a})$
$\mathtt{ct}_{pos} \leftarrow \mathtt{BootRaw}[f_{pos}](\mathtt{ct})$
$\mathtt{ct}_{neg} \leftarrow \mathtt{BootRaw}[f_{neg}](\mathtt{ct})$
$\mathtt{ct}_{sgn} \leftarrow \mathtt{Boot}[f_{sgn}](\mathtt{ct})$
$\mathtt{ct}_{pk} \leftarrow \mathtt{PackingKS}(\mathtt{ct}_{pos}, \{\mathrm{ksk}'_{i,j,k}\}) + \mathtt{PackingKS}(-\mathtt{ct}_{neg}, \{\mathrm{ksk}'_{i,j,k}\}) \cdot X^{\frac{N}{2}}$
**return** $\mathtt{Boot}[\mathtt{ct}_{pk}](\mathtt{ct}_{sgn})$

**Theorem 3.** *Suppose $|e| < \frac{q}{2p}$, $\beta < \frac{q}{8}$ and $|e'| < \frac{q}{2p}$, then* ***FDFB-Select***$(F, \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)) = \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$. *The output error $e'$ has a variance of $(\frac{q}{Q})^2(2\sigma_{acc}^2 + 2\sigma_{pk}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2 + \sigma_{com}^2$. Additionally, when multi-value bootstrap is employed, the variance becomes $(\frac{q}{Q})^2((p(p-1)^2+1)\sigma_{acc}^2 + 2\sigma_{pk}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2 + \sigma_{com}^2$.*

**Theorem 4.** *Suppose that $|e| < \frac{q}{2p}$ and $|e'| < \frac{q}{2p}$, then* ***FDFB-SelectAlt***$(F, \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)) = \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$. *The output error $e'$ has a variance of $(\frac{q}{Q})^2(3\sigma_{acc}^2 + \sigma_{ks}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2 + \sigma_{com}^2$. Additionally, when multi-value bootstrap is employed, the variance becomes $(\frac{q}{Q})^2((6p(p-1)^2+1)\sigma_{acc}^2 + \sigma_{ks}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2 + \sigma_{com}^2$.*

### 3.4 FDFB-BFVMult ($\mathrm{WoPPBS}_1$-Refine)

This algorithm employs the Type-SelectMSB strategy but uses BFV multiplication to handle the MSB. It contains **$\mathbf{WoPPBS_1}$-Refine** and **FDFB-BFVMult**.

**Algorithm 4: FDFB-SelectAlt**

**input** : Plaintext modulus $p$ and an LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$
**input** : Base $B_{pk}$ and modulus $q_{pk}$ for PackingKS
**input** : $\{\text{ksk}'_{i,j,k}\}$, packing keys for PackingKS with $d = N$
**input** : An LWE ciphertext $(b, \vec{a}) = \text{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)$
**output** : An LWE ciphertext $\text{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$

$\mathtt{ct} \leftarrow (b + \frac{q}{2p}, \vec{a})$
$\mathtt{ct}_{hdiff} \leftarrow \mathtt{BootRaw}[(f_{neg} - f_{pos})/2](\mathtt{ct})$
$\mathtt{ct}_{hsum} \leftarrow \mathtt{BootRaw}[(f_{neg} + f_{pos})/2](\mathtt{ct})$
$\mathtt{ct}_{pk} \leftarrow \mathtt{PackingKS}(\mathtt{ct}_{hdiff}, \{\text{ksk}'_{i,j,k}\})$
$\mathtt{ct}_{res} \leftarrow \mathtt{ct}_{hsum} - \mathtt{BootRaw}[\mathtt{ct}_{pk}](\mathtt{ct})$
$\mathtt{ct}_{res} \leftarrow \mathtt{KeySwitch}(\mathtt{ModSwitch}(\mathtt{ct}_{res}, q_{ks}), \{\text{ksk}_{i,j,k}\})$
**return** $\mathtt{ModSwitch}(\mathtt{ct}_{res}, q)$

The process of **WoPPBS$_1$-Refine** is identical to that of **WoP-PBS$_1$**, but it employs a much tighter noise analysis, as we will demonstrate later. It first obtains a ciphertext that encrypts $(-1)^{\text{MSB}}\lfloor \frac{Q}{p}F(m') \rceil$ in the same way as **FDFB-CancelSign**. Then it evaluates the function (7) via a functional bootstrap to acquire the encryption of $\lfloor \frac{Q}{p}(-1)^{\text{MSB}} \rceil$. Finally, it computes the product of the two LWE ciphertexts using LWE-to-RLWE packing and BFV multiplication. The algorithm is fully described in Algorithm 5, and its parameter requirements and noise analysis are given in Theorem 5.

$$f_{sgn1} = \begin{cases} \lfloor \frac{Q}{p} \rceil & x \in [0, \frac{q}{2} - 1] \\ Q - \lfloor \frac{Q}{p} \rceil & x \in [\frac{q}{2}, q - 1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_Q \tag{7}$$

**Algorithm 5: WoPPBS$_1$-Refine**

**input** : Plaintext modulus $p$ and an LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$
**input** : Base $B_{ks}$ and modulus $q_{ks}$ for key switching
**input** : Base $B_{pk}$ and modulus $q_{pk}$ for PackingKS
**input** : $\{\text{ksk}_{i,j,k}\}$, key switching keys
**input** : $\{\text{ksk}'_{i,j,k}\}$, packing keys for PackingKS with $d = 1$
**input** : An LWE ciphertext $(b, \vec{a}) = \text{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}m' + e)$
**output** : An LWE ciphertext $\text{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}F(m') + e')$

$\mathtt{ct} \leftarrow \mathtt{ModUp}((b + \frac{q}{4p}, \vec{a}), q)$
$\mathtt{ct}_0 \leftarrow \mathtt{PackingKS}(\mathtt{BootRaw}[f_{cs}](\mathtt{ct}), \{\text{ksk}'_{i,j,k}\})$
$\mathtt{ct}_{sgn} \leftarrow \mathtt{PackingKS}(\mathtt{BootRaw}[f_{sgn1}](\mathtt{ct}), \{\text{ksk}'_{i,j,k}\})$
$\mathtt{ct}_{prod} \leftarrow \mathtt{SampleExtract}(\mathtt{BFVMult}(\mathtt{ct}_0, \mathtt{ct}_{sgn}), 0)$
$\mathtt{ct}_{res} \leftarrow \mathtt{KeySwitch}(\mathtt{ModSwitch}(\mathtt{ct}_{prod}, q_{ks}), \{\text{ksk}_{i,j,k}\})$
**return** $\mathtt{ModSwitch}(\mathtt{ct}_{res}, \frac{q}{2})$

**FDFB-BFVMult** is an improved version of **WoP-PBS$_2$**. Unlike **WoP-PBS$_2$**, which requires the sign bit to be multiplied with both $f_{neg}(\text{ct})$ and $f_{pos}(\text{ct})$, **FDFB-BFVMult** only needs one BFV multiplication because the sign bit is multiplied with the fresh ciphertext $(f_{neg} - f_{pos})(\text{ct})$. Consequently, **FDFB-BFVMult** further halves the noise growth. Specifically, **FDFB-BFVMult** first constructs two LUTs $F_0$ and $F_1$ in the same way as **FDFB-Select**. Next, by using two functional bootstraps to evaluate $f_{pos}$ and $f_{neg} - f_{pos}$ (defined in (4) and (5)), it obtains encryptions of $m_{pos} = \lfloor \frac{Q}{p}F_0(m') \rceil$ and $m_{diff} = \lfloor \frac{Q}{p}(F_1 - F_0)(m') \rceil$. Then it evaluates the function (8) via a functional bootstrap to acquire the encryption of $m_{sgn} = \lfloor -\frac{Q}{2p}(-1)^{\mathrm{MSB}} \rceil + \lfloor \frac{Q}{2p} \rceil \approx \lfloor \frac{Q}{p}\mathrm{MSB} \rceil$ Finally, it computes $\mathrm{MSB} \cdot m_{diff} + m_{pos} \approx \lfloor \frac{Q}{p} F_{\mathrm{MSB}}(m') \rceil$ using LWE-to-RLWE packing and BFV multiplication. The algorithm is fully described in Algorithm 6, and its parameter requirements and noise analysis are given in Theorem 6.

Since the two bootstraps in **WoPPBS$_1$-Refine** (and the three bootstraps in **FDFB-BFVMult**) share the same input, they can be accelerated by employing a single multi-value bootstrap at the cost of increased noise growth.

$$f_{sgn2} = \begin{cases} Q - \lfloor \frac{Q}{2p} \rceil & x \in [0, \frac{q}{2} - 1] \\ \lfloor \frac{Q}{2p} \rceil & x \in [\frac{q}{2}, q - 1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_Q \tag{8}$$

**Algorithm 6: FDFB-BFVMult**

**input** : Plaintext modulus $p$ and an LUT $F : \mathbb{Z}_p \to \mathbb{Z}_p$
**input** : Base $B_{ks}$ and modulus $q_{ks}$ for key switching
**input** : Base $B_{pk}$ and modulus $q_{pk}$ for PackingKS
**input** : $\{\mathrm{ksk}_{i,j,k}\}$, key switching keys
**input** : $\{\mathrm{ksk}'_{i,j,k}\}$, packing keys for PackingKS with $d = 1$
**input** : An LWE ciphertext $(b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)$
**output** : An LWE ciphertext $\mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$

$\mathtt{ct} \leftarrow (b + \frac{q}{2p}, \vec{a})$
$\mathtt{ct}_{pos} \leftarrow \mathtt{BootRaw}[f_{pos}](\mathtt{ct})$
$\mathtt{ct}_{diff} \leftarrow \mathtt{PackingKS}(\mathtt{BootRaw}[f_{neg} - f_{pos}](\mathtt{ct}), \{\mathrm{ksk}'_{i,j,k}\})$
$\mathtt{ct}_{sgn} \leftarrow \mathtt{PackingKS}(\mathtt{BootRaw}[f_{sgn2}](\mathtt{ct}) + \lfloor \frac{Q}{2p} \rceil, \{\mathrm{ksk}'_{i,j,k}\})$
$\mathtt{ct}_{prod} \leftarrow \mathtt{SampleExtract}(\mathtt{BFVMult}(\mathtt{ct}_{diff}, \mathtt{ct}_{sgn}), 0)$
$\mathtt{ct}_{res} \leftarrow \mathtt{ct}_{prod} + \mathtt{ct}_{pos}$
$\mathtt{ct}_{res} \leftarrow \mathtt{KeySwitch}(\mathtt{ModSwitch}(\mathtt{ct}_{res}, q_{ks}), \{\mathrm{ksk}_{i,j,k}\})$
**return** $\mathtt{ModSwitch}(\mathtt{ct}_{res}, q)$

**Refined BFV Noise Analysis.** Next, we provide a refined noise analysis for the BFV multiplication involved in **FDFB-BFVMult** (**WoPPBS$_1$-Refine**). Our core observation is that in LWE-to-RLWE packing, only the constant term of the output polynomial message is assigned the value of the input LWE message, while the coefficients of non-constant terms are close to 0.

Lemma 2 provides a noise analysis of this kind of BFV multiplication. We note that only the dominating term of the error variance is displayed in Lemma 2 (as well as in Theorem 5 and Theorem 6) due to the complexity of the full formula. Refer to the full version of the paper for the full formula and its proof.

In **FDFB-BFVMult** (**WoPPBS$_1$-Refine**), each of the multiplicands for BFV multiplication is obtained by packing an LWE message with an error variance of $\sigma_{acc}^2$ into the constant term of an RLWE ciphertext. This means that the constant term of the encrypted polynomial has an error variance of $\sigma_{acc}^2 + \sigma_{ks}^2$, while the error variance of non-constant terms is $\sigma_{ks}^2$. Note that $\sigma_{acc}^2$ and $\sigma_{ks}^2$ correspond to $\sigma_i^2$ and $\sigma_i^{2\prime}$ in Lemma 2. In practice, $\sigma_{acc}^2$ is much larger than $N\sigma_{ks}^2$ and one of the packed LWE messages is a sign bit (i.e., in $\{0, \pm 1\}$). It then follows from Lemma 2 that the output error variance is about $2p^2\sigma_{ms}^2\sigma_{acc}^2$.

On the other hand, for ordinary BFV multiplication where all terms have an error variance of $\sigma_{acc}^2 + \sigma_{ks}^2$, the output error variance is about $2Np^2\sigma_{ms}^2\sigma_{acc}^2$. This is because the dominating noise term becomes a polynomial-polynomial multiplication and introduces an extra factor $N$ compared to scalar-polynomial multiplication (refer to the remark in the full version of the paper for details). This means the noise growth is reduced by roughly $N$ times compared to conventional BFV multiplication.

**Lemma 2.** *Let* $c_i = (\boldsymbol{b}_i, \boldsymbol{a}_i) = \mathrm{RLWE}_{\boldsymbol{s},N,Q}(\frac{Q}{p}m_i + e_i + \boldsymbol{e}_i)$ *for* $i = 0, 1$*, where* $e_i \sim N(0, \sigma_i^2)$*,* $\boldsymbol{e}_i \sim N(0, \sigma_i'^2)^N$*,* $\sigma_i^2 \gg N\sigma_i'^2$ *and* $m_0 \in \{0, \pm 1\}$*. Then SampleExtract(BFVMult*$(c_0, c_1), 0) = \frac{Q}{p}m_0 m_1 + e$ *and the variance of* $e$ *is equal to* $p^2\sigma_{ms}^2(\sigma_0^2 + \sigma_1^2)$ *approximately*[2]*.*

**Theorem 5.** *Suppose* $|e| < \frac{q}{4p}$ *and* $|e'| < \frac{q}{4p}$*, then* ***WoPPBS*$_1$*-Refine***$(F, \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}m' + e)) = \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}F(m') + e')$*. The output error* $e'$ *has a variance of* $(\frac{q}{Q})^2 \frac{N}{9} p^2 \sigma_{acc}^2 + \sigma_{com}^2$ *approximately. Additionally, when multi-value bootstrap is employed, the variance becomes* $(\frac{q}{Q})^2 \frac{N}{18} p^3 (p-1)^2 \sigma_{acc}^2 + \sigma_{com}^2$ *approximately.*

**Theorem 6.** *Suppose* $|e| < \frac{q}{2p}$ *and* $|e'| < \frac{q}{2p}$*, then* ***FDFB-BFVMult***$(F, \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)) = \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$*. The output error* $e'$ *has a variance of* $(\frac{q}{Q})^2 \frac{N}{9} p^2 \sigma_{acc}^2 + \sigma_{com}^2$ *approximately. Additionally, when multi-value bootstrap is employed, the variance becomes* $(\frac{q}{Q})^2 \frac{2N}{9} p^3 (p-1)^2 \sigma_{acc}^2 + \sigma_{com}^2$ *approximately.*

# 4 Improved Homomorphic Digit Decomposition

This section presents two algorithms **HomDecomp-Reduce** and **HomDecomp-FDFB** to decompose an LWE ciphertext with a large modulus $q_0$ into multiple LWE ciphertexts with a smaller modulus $q$, each encrypting a digit of the original message. **HomDecomp-Reduce** creates buffer space for modulus switching noise by reducing the range of lower bits by half. It can handle digits of up to 4 bits and requires one bootstrap operation per decomposed digit. In contrast, **HomDecomp-FDFB** clears the lower bits approximately and can handle digits of up to 5 bits, but it requires two bootstrap operations per digit. We still assume $q = 2N$ as in the previous section. In our noise analysis, we assume that the input ciphertext of the decomposition algorithms has an error variance of $\sigma_{boot}^2$ as in [LMP22]. Proof of theorems is left to the full version of the paper due to space limit.

## 4.1 HomDecomp-Reduce

In **HomDecomp-Reduce**, the range of lower bits is first reduced by half using one bootstrap operation to accommodate the subsequent modulus switching noise. The reduction function $f_{red} : \mathbb{Z}_q \to \mathbb{Z}_{q_0}$ is defined in (9), with different input and output ranges.

$$f_{red} = \begin{cases} \frac{q}{4} & x \in [0, \frac{q}{2} - 1] \\ q_0 - \frac{q}{4} & x \in [\frac{q}{2}, q - 1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_{q_0} \tag{9}$$

The complete algorithm is described in Algorithm 7. Its parameter requirements and noise analysis are given in Theorem 7.

**Theorem 7.** *If* $\mathrm{bnd}\sqrt{B^{-2}\sigma_{boot}^2 + \sigma_{ms}^2} < \frac{q}{4B}$*,* ***HomDecomp-Reduce*** *outputs the decomposed digits correctly.*

[2] Here, 'approximately' means that only the dominant term of the error variance is displayed, as the full formula is quite complex. For the full formula and an explanation of the approximation, please refer to the full version of the paper.

**Algorithm 7: HomDecomp-Reduce**

**input** : A base $B$ for homomorphic decomposition
**input** : An LWE ciphertext $ct = \text{LWE}_{\vec{s},n,q_0}(\frac{q_0}{p}m' + e)$
**output** : LWE ciphertexts $\{ct_i\}$ encrypting the digits of $m'$

$i \leftarrow 0$
**while** $q_0 > q$ **do**
  $\mathsf{ct}_i \leftarrow \mathtt{ModDown}(\mathsf{ct}, q)$
  $\mathsf{ct} \leftarrow \mathsf{ct} + (\frac{q_0}{2p}, \vec{0})$
  $\mathsf{ct}' \leftarrow \mathtt{ModDown}(\mathsf{ct}, q)$
  $\mathsf{ct} \leftarrow \mathsf{ct} + \mathtt{Boot}[f_{red}](\mathsf{ct}') - (\frac{q}{2}, \vec{0})$
  $\mathsf{ct} \leftarrow \mathtt{ModSwitch}(\mathsf{ct}, \frac{q_0}{B})$
  $i \leftarrow i + 1$
$\mathsf{ct}_i \leftarrow \mathsf{ct}$
**return** $\{\mathsf{ct}_i\}$

## 4.2 HomDecomp-FDFB

In **HomDecomp-FDFB**, we use **FDFB-Compress** to evaluate the continuous identity function $f_{id}(x) = x : \mathbb{Z}_q \to \mathbb{Z}_{q_0}$ (using zero extension), and the obtained result is used to approximately clear the lower bits in the input ciphertext. See Algorithm 8 for a full description of **HomDecomp-FDFB** and Theorem 9 for its parameter requirements and noise analysis.

Before beginning, we show how to evaluate a continuous function $F'$ with **FDFB-Compress**, where the input and output scaling factors are $\Delta_{in}$ and $\Delta_{out}$ respectively. First, the compression function $f_C$ in (1) is substituted with $f'_C$, which is defined in (10) and illustrated in Figure 3.

$$f'_C = \begin{cases} \lfloor \frac{\frac{q}{4}-2\beta}{\frac{q}{2}-1} x + \beta \rceil & x \in [0, \frac{q}{2}-1] \\ q - f'_C(x - \frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_q \tag{10}$$

The strategy adopted to construct $f'_C$ is called '$\beta$-padding', which creates a $2\beta$ distance between $f'_C(0)$ and $f'_C(\frac{q}{2})$ to separate the cases where the input is 0 and $\frac{q}{2}$. Otherwise, the bootstrapping error may intermix the two cases, making it impossible for $f_{eval}$ to distinguish between them. As a result, when the input is positive and near 0, **FDFB-Compress** may yield an incorrect result $F'(-\frac{q}{2\Delta_{in}})$ instead of $F'(0)$. Also, $f'_C(\frac{q}{2}-1)$ and $f'_C(q-1)$ must be $\beta$ away from $\frac{q}{4}$ and $\frac{3q}{4}$ respectively to ensure that the output message of $f'_C$ always lies within half of $\mathbb{Z}_q$.

The modified version of $f_{eval}$ in (2) (which we denote as $f'_{eval}$) is rather complicated. Intuitively $f'_{eval}$ aims to recover the original input to $f'_C$, evaluate $F'$ on the recovered input, and subsequently scales the result by $\Delta_{out}$. As the evaluation of $f'_C$ introduces a bootstrapping error, the input recovered by $f'_{eval}$ also contains a bootstrapping error (multiplied by some constant), which means that the output error of **FDFB-Compress** depends on the Lipschitz constant of $F'$. The output error variance is given in Theorem 8, and the proof can be found in the full version of the paper.

**Theorem 8.** *When evaluating a continuous function $f$ with Lipschitz constant $L$, the output error variance of **FDFB-Compress** is $(\frac{L}{k_2\Delta_{in}})^2\sigma^2_{boot} + \Delta^{-2}_{out}\sigma^2_{boot}$, where $k_2 = \frac{\frac{N}{2}-2\beta}{N-1} \approx \frac{1}{2}$.*

**HomDecomp-FDFB** sets $\Delta_{in} = \Delta_{out} = 1$ and $F' = f_{id}$, which gives the following theorem.

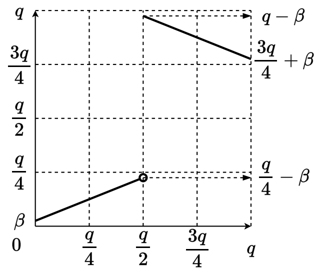


Figure 3: Compression function $f'_C$ for continuous function evaluation.

**Theorem 9.** *Let $e_f$ be the output error of **FDFB-Compress**, then its variance is $\sigma_f^2 = (1 + k_2^{-2})\sigma_{boot}^2$. If $\mathrm{bnd}\sqrt{B^{-2}\sigma_f^2 + \sigma_{ms}^2} < \frac{q}{2B}$, **HomDecomp-FDFB** outputs the decomposed digits correctly.*

---

**Algorithm 8: HomDecomp-FDFB**

**input** : A base $B$ for homomorphic decomposition
**input** : An LWE ciphertext $ct = \mathrm{LWE}_{\vec{s},n,q_0}(\frac{q_0}{p}m' + e)$
**output** : LWE ciphertexts $\{ct_i\}$ encrypting the digits of $m'$

$i \leftarrow 0$
**while** $q_0 > q$ **do**
  $\mathsf{ct}_i \leftarrow \mathtt{ModDown}(\mathsf{ct}, q)$
  $\mathsf{ct} \leftarrow \mathsf{ct} + (\frac{q_0}{2p}, \vec{0})$
  $\mathsf{ct}' \leftarrow \mathtt{ModDown}(\mathsf{ct}, q)$
  $\mathsf{ct} \leftarrow \mathsf{ct} - \mathtt{FDFB\text{-}Compress}[f_{id}](\mathsf{ct}')$
  $\mathsf{ct} \leftarrow \mathtt{ModSwitch}(\mathsf{ct}, \frac{q_0}{B})$
  $i \leftarrow i + 1$
$\mathsf{ct}_i \leftarrow \mathsf{ct}$
**return** $\{\mathsf{ct}_i\}$

---

# 5 Analysis and Comparison

This section analyzes the FDFB and the homomorphic decomposition algorithms, both previous ones and ours, concerning their noise growth and the number of required bootstraps.

## 5.1 Analysis of FDFB Algorithms

Table 3 presents the error variance ratio between our and previous FDFB algorithms and the number of bootstraps required. For Type-HalfRange FDFB algorithms (**FDFB-Compress** and **EvalFunc**), the coded message must first be compressed into half of $\mathbb{Z}_q$. Thus the error of the compressed message (e.g., the error in ct of line 1 of Algorithm 1) plays a major role in the selection of parameters. For Type-SelectMSB FDFB algorithms (other algorithms in Table 3), the output error plays a major role in the selection of parameters. The dominant term of the output error variance is the $\sigma_{acc}^2$-term for most algorithms (refer to the full version of the paper for the formula of the output error variance of all algorithms). Thus, in the table, the first row of the ratio column represents the ratio of the error variances of the compressed message. The remaining rows of the

Table 3: Comparison of previous and our FDFB algorithms regarding their noise growth and the number of bootstraps required.

| Ours | Previous | Error Var Ratio (Ours/Prev) | Num of BTS (Prev→Ours) |
|---|---|---|---|
| **FDFB-Compress** | **EvalFunc** [LMP22] | $1/2$ | $2 \to 2$ |
| **FDFB-CancelSign** | **WoP-PBS$_1$** [CLOT21] | $18/N^2p^2$ | $2 \to 2$ |
| **WoPPBS$_1$-Refine** | | $1/N$ | $2 \to 2$ |
| **FDFB-Select** | **WoP-PBS$_2$** [CLOT21] | $9/N^2p^2$ | $3 \to 4$ |
| **FDFB-SelectAlt** | | $27/2N^2p^2$ | $3 \to 3$ |
| **FDFB-BFVMult** | | $1/N$ | $3 \to 3$ |
| **WoPPBS$_1$-Refine*** | **WoP-PBS$_1^*$** [CLOT21] | $1/N$ | $1 \to 1$ |
| **FDFB-Select*** | **WoP-PBS$_2^*$** [CLOT21] | $9/2N^2p^2$ | $1 \to 2$ |
| **FDFB-SelectAlt*** | | $27/N^2p^2$ | $1 \to 2$ |
| **FDFB-BFVMult*** | | $1/N$ | $1 \to 1$ |

* FDFB algorithms that use multi-value bootstrap.

ratio column represent the ratios of the $\sigma_{acc}^2$-terms of the output error variance. For **FDFB-CancelSign**, **FDFB-Select** and **FDFB-SelectAlt**, the ratios of the output error variance can be a small multiple of the displayed ones. For other algorithms, the output error variance ratios are very close to the displayed ones since the $\sigma_{acc}^2$-term is dominant.

As stated earlier, the efficiency of an FDFB algorithm is not solely determined by the number of bootstraps it requires. The error variances also impact the compactness of parameters and thus affect the final efficiency. As shown in Table 3, the main advantage of our FDFB algorithms is their reduced noise growth. This allows for the selection of larger decomposition bases during blind rotation, resulting in a reduction in the decomposition dimension (denoted by $l$ as described in Section 2.2.2). Since the number of NTTs required for a blind rotation is proportional to $(l+1)$, our algorithms achieve better performance. To be more specific:

- **FDFB-Compress** reduces the error variance of the compressed message by half, resulting in a more relaxed parameter choice than **EvalFunc**.
- **FDFB-CancelSign**, **FDFB-Select**, **FDFB-SelectAlt** and their multi-value bootstrap variants use LWE-to-RLWE packing and blind rotation instead of BFV multiplication. This reduces the noise to $O(1/N^2p^2)$ that of **WoP-PBS**. Although our algorithms require an additional bootstrap to replace the BFV multiplication, we demonstrate in Section 6 that they are still faster than **WoP-PBS** in most cases due to their slower noise growth.
- **WoPPBS$_1$-Refine** and **FDFB-BFVMult** use significantly tighter noise analysis for BFV multiplication than **WoP-PBS$_1$** and **WoP-PBS$_2$**, reducing the noise growth to $1/N$ the original value.

**The Optimality of FDFB-Compress.** We observe that **FDFB-Compress** achieves optimality among Type-HalfRange algorithms. Recall that Type-HalfRange first uses functional bootstraps to transform the coded message $\frac{q}{p}m'+e \in \mathbb{Z}_q$ into $\phi(m')+\tilde{e} \in U \subseteq \mathbb{Z}_q$ and then evaluate the LUT with another functional bootstrap, where $\phi$ is an arbitrary map, $U$ satisfies $U \cap (U+\frac{q}{2}) = \emptyset$ to bypass the negacyclic constraint, and $\tilde{e}$ has a variance of at least $\sigma_{\tilde{e}}^2 \geq \sigma_{boot}^2$. Additionally, to ensure the correctness of evaluation, $m'$ must be reconstructible from $\tilde{m}+\tilde{e}$, i.e., there is a map $\lambda$ from $U$ to $\mathbb{Z}_p$ such that $\lambda(\phi(m')+\tilde{e}) = m'$ for any $m' \in \mathbb{Z}_p$ and any $|\tilde{e}| < \text{bnd} \cdot \sigma_{\tilde{e}}$.

Thus, on the one hand, **FDFB-Compress** achieves the minimum number of bootstraps required for Type-HalfRange (i.e., 2). On the other hand, since $\phi(m')+\tilde{e} \in \lambda^{-1}(m')$, by the pigeonhole principle there exists an $m' \in \mathbb{Z}_p$ such that $|\lambda^{-1}(m')| \leq \frac{|U|}{p} \leq \frac{q}{2p}$,

Table 4: Comparison of previous and our homomorphic decomposition algorithms.

| Ours | **HomDecomp-Reduce** | **HomDecomp-FDFB** |
|---|---|---|
| Previous | **HomFloor** [LMP22] | **HomFloorAlt** [LMP22] |
| Number of BTS (Previous→Ours) | $2 \to 1$ | $3 \to 2$ |
| Constraints of Previous Methods | Cannot decompose extracted CKKS ciphertexts | $q > 8\sqrt{2}\beta$ |

implying $\frac{q}{2p} > 2 \cdot \text{bnd} \cdot \sigma_{\tilde{e}} \geq 2\beta$. This requires $\beta < \frac{q}{4p}$, which is also the *only* requirement for **FDFB-Compress**. This means that **FDFB-Compress** achieves the most compact parameter choice among Type-HalfRange algorithms, thus achieving optimality.

### 5.2 Analysis of Homomorphic Decomposition

Table 4 compares the number of bootstraps needed for previous and our homomorphic digit decomposition algorithms. Algorithms in the same row of the table share the same digit decomposition base $B$ (i.e., their decomposed digits have the same plaintext modulus). According to the table, our algorithms need one less bootstrap than previous algorithms in [LMP22]. **HomFloor** requires that the input ciphertext encodes a discrete plaintext with small noise, which ensures a gap between two adjacent encoded messages to accommodate the noise introduced by subsequent bootstraps. Since an extracted CKKS ciphertext encodes messages continuously without any gaps, **HomFloor** cannot be applied to decompose it. Also, **HomFloorAlt** has an extra constraint for the ciphertext modulus. In contrast, our methods are free of these constraints, making them more flexible than previous methods. The full version of the paper provides a theoretical analysis of the noise growth and parameter choice.

## 6 Implementation

We implement all the FDFB algorithms and homomorphic decomposition algorithms, including both previous ones and ours, in OpenFHE [BBB+22] (commit id `745a492`). We disable multi-threading, except during key generation. We build OpenFHE using the g++ compiler of version 12.2.1 with flag `WITH_NATIVEOPT=ON` (as the authors did in [LMP22]). The performance of algorithms is tested on a machine with Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz and 125G of RAM, running Fedora Release 36.

**Parameter Setting.** We use two parameter sets in our LWE schemes, i.e., $\text{PARAM}_{\text{decomp}}$ and $\text{PARAM}_{\text{fast}}$, which have been verified to meet 128-bit security using lattice-estimator [APS15] (commit id `48fa49b`). Table 5 presents the details of these parameter sets, and we briefly explain the selection criteria of $q_{ks}$ below since $n$ can be determined from $q_{ks}$. For $\text{PARAM}_{\text{decomp}}$, the maximum ciphertext modulus is set to $2^{35}$ such that the ciphertext to be digit-decomposed has a large modulus. This choice for $q_{ks}$ is also consistent with [LMP22]. For $\text{PARAM}_{\text{fast}}$, we focus on FDFB algorithms for discrete

Table 5: Parameter sets for LWE scheme and their use cases.

| LWE Param Sets | $n$ | $q_{ks}$ | Use Cases |
|---|---|---|---|
| $\text{PARAM}_{\text{decomp}}$ | 1340 | $2^{35}$ | HomDecomp, Discrete FDFB |
| $\text{PARAM}_{\text{fast}}$ | 760 | $2^{20}$ | Discrete FDFB |

Table 6: Running time of previous and our FDFB algorithms under four scenarios (A to D). Each running time is obtained by averaging over 100 tests and is measured in milliseconds (ms). For each scenario, the best algorithms from previous works and this paper are marked in blue and red, respectively. A '/' indicates that the algorithm is unavailable in that scenario because the plaintext modulus $p$ exceeds its parameter requirements.

| Algorithm | $\text{PARAM}_{\text{decomp}}$ A: $p = 2^4$ | $\text{PARAM}_{\text{decomp}}$ B: $p = 2^5$ | $\text{PARAM}_{\text{fast}}$ C: $p = 2^4$ | $\text{PARAM}_{\text{fast}}$ D: $p = 2^5$ |
|---|---|---|---|---|
| **EvalFunc** | / | / | 598 | / |
| **WoP-PBS**$_1^*$ | 1160 | / | 682 | / |
| **WoP-PBS**$_2^*$ | 1200 | 1930 | 735 | 942 |
| **FDFB-KS** | 5360 | 6340 | 2940 | 3110 |
| **Comp**$^*$ | 1580 | 1760 | 897 | 985 |
| **FDFB-Compress** | 1050 | / | 598 | / |
| **FDFB-CancelSign** | 1060 | / | 611 | / |
| **FDFB-Select**$^*$ | 1260 | 1250 | 621 | 724 |
| **FDFB-SelectAlt**$^*$ | 1240 | 1250 | 717 | 718 |
| **WoPPBS**$_1$**-Refine**$^*$ | **777** | / | **458** | / |
| **FDFB-BFVMult**$^*$ | 785 | **1150** | **458** | **573** |

LUTs. Thus $q_{ks}$ can be set to a smaller value to accelerate FDFB. However, if $q_{ks}$ is too small, it may lead to large key switching noise, corrupting the correctness of FDFB. Therefore, we set $q_{ks} = 2^{20}$ in $\text{PARAM}_{\text{fast}}$.

The performance of discrete LUT evaluation with FDFB variants is tested with the plaintext modulus set to $2^4$ and $2^5$. To ensure fair comparisons, we have only recorded the best performance among the parameters for FDFB variants with multiple parameter choices (e.g., multi-value or not). In our experiments, the multi-value versions usually run faster than the non-multi-value ones. Thus, the multi-value versions of most algorithms are recorded.

Please refer to the full version of the paper for a complete list of the parameters used in the benchmarks.

**Performance of FDFB Algorithms.** Table 6 shows the running time of previous and our FDFB algorithms under four scenarios (two parameter sets $\times$ two choices of $p$). We can draw the following conclusions from the benchmark data.

First, the experiment data validate our algorithms' advantage over their predecessors, as suggested theoretically in Section 5. To be more specific:

• **FDFB-Compress** can support $p = 2^4$ in scenario A while **EvalFunc** cannot because the former benefits from a reduced error variance of the compressed message. In fact, **EvalFunc** would need to double the RLWE dimension $N$ to support $p = 2^4$, which leads to worse efficiency.

• **FDFB-CancelSign** shows a speedup of 8.6%∼10.4% compared to **WoP-PBS**$_1^*$, even though it requires one additional bootstrap and does not use multi-value bootstrap for acceleration. This is due to the slower noise growth of **FDFB-CancelSign**, which

Table 7: Performance improvement of our FDFB algorithms.

| Scenario in Table 6 | A | B | C | D |
|---|---|---|---|---|
| Best Running Time (Old, ms) | 1160 | 1760 | 598 | 942 |
| Best Running Time (New, ms) | 777 | 1150 | 458 | 573 |
| Reduction in Running Time | 33.0% | 34.7% | 23.4% | 39.2% |

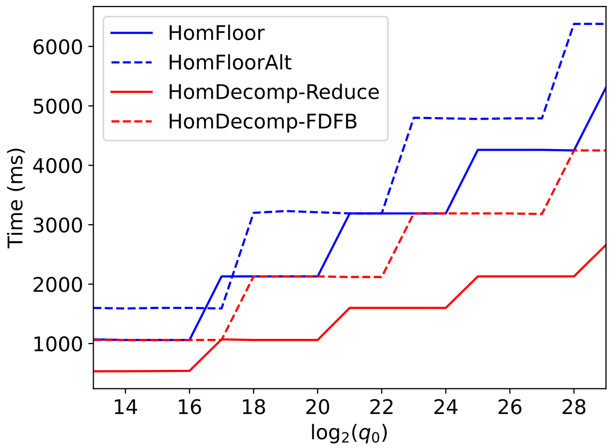


Figure 4: The "running time"-"input precision" graph for previous (blue) and our (red) homomorphic digit decomposition algorithms under $\mathrm{PARAM}_{\mathrm{decomp}}$.

allows for the choice of a larger decomposition base $B_g$ in blind rotation, resulting in improved performance. On the other hand, **FDFB-Select*** and **FDFB-SelectAlt*** have similar running time to $\mathbf{WoP\text{-}PBS}_2^*$ in scenarios A & C and are 23.1%∼35.2% faster than $\mathbf{WoP\text{-}PBS}_2^*$ in scenarios B & D. This advantage grows with $p$, as a larger $p$ results in less tolerance for homomorphic noise and forces prior methods to use smaller $B_g$, degrading their performance.

• $\mathbf{WoPPBS_1\text{-}Refine}^*$ is 32.8%∼33.0% faster than $\mathbf{WoP\text{-}PBS}_1^*$ and **FDFB-BFVMult*** is 37.7%∼40.4% faster than $\mathbf{WoP\text{-}PBS}_2^*$. Again, such performance improvement benefits from the choice of a larger $B_g$, which is possible due to the algorithms' reduced noise growth.

Second, when comparing the fastest algorithms from previous works and our algorithms, we observe a 23.4%∼39.2% reduction in running time across all four scenarios (see Table 7). Among our algorithms, **FDFB-BFVMult*** is the fastest or very close to the fastest in all the scenarios. However, it does not render our other algorithms obsolete because (1) they support the addition of more bootstrapped ciphertexts since they have smaller output error than **FDFB-BFVMult** ($\mathbf{WoPPBS_1\text{-}Refine}$); (2) they are useful for smaller RLWE dimensions, where BFV-based FDFB methods might be unavailable.

**Performance of Homomorphic Digit Decomposition.** Figure 4 illustrates the performance of different homomorphic decomposition algorithms (the raw data can be found in the full version of the paper). Data for $B = 2^4$ are drawn in solid lines, while data for $B = 2^5$ are drawn in dashed lines. For all choices of $\log_2(q_0)$, **HomDecomp-Reduce** runs roughly twice as fast as **HomFloor**, and **HomDecomp-FDFB** runs roughly at 1.5 times the speed of **HomFloorAlt**. Such speedup in homomorphic decomposition directly leads to speedup in the large-precision sign/ReLU/max/ABS evaluation, as they all require extracting the MSB of the input message.

+

## 7 Conclusion

This paper develops four FDFB algorithms and two homomorphic decomposition algorithms. Our FDFB algorithms achieve a running time shorter than the best known results by up to 39.2%. Our homomorphic decomposition algorithms run 1.5x to 2x as fast as those presented in [LMP22], leading to speedup in large-precision ReLU, sign, max and ABS evaluation. We give a thorough theoretical noise analysis for FDFB and homomorphic decomposition algorithms, both in prior works and ours. We also implement all the algorithms in OpenFHE for a fair comparison between them.

## Acknowledgments

This work is supported by the National Key R&D Program of China (2018YFA0704701, 2020YFA0309705), Shandong Key Research and Development Program (2020ZLYS09), the Major Scientific and Technological Innovation Project of Shandong, China (2019JZZY010133), the Major Program of Guangdong Basic and Applied Research (2019B030302008), and Tsinghua University Dushi Program.

## References

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[BBB+22] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[BGGJ20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory*, 6(3), July 2014.

[BIP+22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE Instantiated with NTRU and LWE. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 188–215, Cham, 2022. Springer Nature Switzerland.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 483–512, Cham, 2018. Springer International Publishing.

[CGGI20] 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.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1223–1237, New York, NY, USA, 2018. Association for Computing Machinery.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 106–126, Cham, 2019. Springer International Publishing.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning*, pages 1–19, Cham, 2021. Springer International Publishing.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 670–699, Cham, 2021. Springer International Publishing.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast Private Set Intersection from Homomorphic Encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1243–1255, New York, NY, USA, 2017. Association for Computing Machinery.

[CMdG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1135–1150, New York, NY, USA, 2021. Association for Computing Machinery.

[CZ22] Olive Chakraborty and Martin Zuber. Efficient and Accurate Homomorphic Comparisons. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 35–46, New York, NY, USA, 2022. Association for Computing Machinery.

[CZB+22] Pierre-Emmanuel Clet, Martin Zuber, Aymen Boudguiga, Renaud Sirdey, and Cédric Gouy-Pailler. Putting up the swiss army knife of homomorphic calculations by means of TFHE functional bootstrapping. Cryptology ePrint Archive, Paper 2022/149, 2022. https://eprint.iacr.org/2022/149.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):229–253, February 2021.

[GBA22] 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.

[Gen09] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Klu22] Kamil Kluczniak. NTRU-v-Um: Secure Fully Homomorphic Encryption from NTRU with Small Modulus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1783–1797, New York, NY, USA, 2022. Association for Computing Machinery.

[KPZ21] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting Homomorphic Encryption Schemes for Finite Fields. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 608–639, Cham, 2021. Springer International Publishing.

[KS22] Kamil Kluczniak and Leonard Schild. FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):501–537, Nov. 2022.

[KSK+18] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics*, 11(4):83, October 2018.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1219–1234, New York, NY, USA, 2012. Association for Computing Machinery.

[LHH+21] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073, 2021.

[LKL+22] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access*, 10:30039–30054, 2022.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 130–160, Cham, 2022. Springer Nature Switzerland.

[LW23a] Feng-Hao Liu and Han Wang. Batch Bootstrapping I: A New Framework for SIMD Bootstrapping in Polynomial Modulus. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 321–352, Cham, 2023. Springer Nature Switzerland.

[LW23b] Feng-Hao Liu and Han Wang. Batch Bootstrapping II: Bootstrapping in Polynomial Modulus only Requires $\tilde{O}(1)$ FHE Multiplications in Amortization. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 353–384, Cham, 2023. Springer Nature Switzerland.

[LY23] Kang Hoon Lee and Ji Won Yoon. Discretization Error Reduction for High Precision Torus Fully Homomorphic Encryption. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography – PKC 2023*, pages 33–62, Cham, 2023. Springer Nature Switzerland.

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like Cryptosystems. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '21, page 17–28, New York, NY, USA, 2021. Association for Computing Machinery.

[PKS+19] Sungjoon Park, Minsu Kim, Seokjun Seo, Seungwan Hong, Kyoohyung Han, Keewoo Lee, Jung Hee Cheon, and Sun Kim. A secure SNP panel scheme using homomorphically encrypted K-mers without SNP calling on the user side. *BMC Genomics*, 20(2):188, April 2019.

[YXS+21] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. TOTA: Fully Homomorphic Encryption with Smaller Parameters and Stronger Security. Cryptology ePrint Archive, Paper 2021/1347, 2021. https://eprint.iacr.org/2021/1347.

# Supplementary Material

## A Evaluating a Continuous Function with FDFB-Compress

This section presents the details of evaluating a continuous function with **FDFB-Compress**, including the definition of $f'_{eval}$ and the proof of Theorem 8.

The function $f_{eval}$ in (2) is replaced by $f'_{eval} : \mathbb{Z}q \to \mathbb{Z}_Q$, which is defined as

$$f'_{eval}(x) = \begin{cases} \lfloor F'(\frac{x-\beta}{\text{slope}\Delta_{in}})\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [\beta, \frac{q}{4} - \beta] \\ \lfloor F'((\frac{q-x-\beta}{\text{slope}} - \frac{q}{2})\Delta_{in}^{-1})\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [\frac{3q}{4} + \beta, q - \beta] \\ \lfloor F'(0)\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [0, \beta - 1] \\ \lfloor F'((\frac{q}{2} - 1)\Delta_{in}^{-1})\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [\frac{q}{4} - \beta + 1, \frac{q}{4} - 1] \\ \lfloor F'(-\frac{q}{2\Delta_{in}})\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [\frac{3q}{4}, \frac{3q}{4} + \beta - 1] \\ \lfloor F'(-\frac{1}{\Delta_{in}})\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [q - \beta + 1, q - 1] \\ Q - f'_{eval}(x + \frac{q}{2}) & x \in [\frac{q}{4}, \frac{3q}{4} - 1] \end{cases},$$

where slope $= \frac{\frac{q}{4} - 2\beta}{\frac{q}{2} - 1}$.

As said previously, the first two cases compute the inverse of $f'_C$, evaluate $F'$ on the recovered input, and scale the result to $\Delta_{out}$. The third to sixth cases handle the boundary cases. The last case exists only to ensure the negacyclicity of $f'_{eval}$.

**Proof of Theorem 8.**

*Proof.* We denote the input value of $f'_C$ as $x_0 \in \mathbb{Z}_q$. As discussed in the paper, $f'_{eval}$ first tries to recover $x_0$ and then evaluate $F'$ on $x_0\Delta_{in}^{-1}$. Let us first consider the case of $x \in [\beta, \frac{q}{4} - \beta] \bigcup [\frac{3q}{4} + \beta, q - \beta]$. In this case, the input to $F'$ in **FDFB-Compress** is exactly

$$\lfloor f_C'^{-1}(\lfloor f'_C(x_0) \rceil + e_{boot}) \rceil \Delta_{in}^{-1}$$

, where $f'_C$ is a two-piece linear function, and the slope of both pieces have the same absolute value $k_2 = \frac{q/4 - 2\beta}{q/2 - 1}$. Thus, the input error to $F'$ is

$$(k_2^{-1}(e_{boot} + e_{rnd,0}) + e_{rnd,1})\Delta_{in}^{-1}$$

, where $e_{rnd,i}$ is the rounding error of the $i$-th rounding operator and has a variance of $\frac{1}{4}$. Thus, the input error to $F'$ has a variance of $(k_2^{-2}(\sigma_{boot}^2 + \frac{1}{4}) + \frac{1}{4})\Delta_{in}^{-2} \approx k_2^{-2}\Delta_{in}^{-2}\sigma_{boot}^2$. Let $L$ be the Lipschitz constant of $F'$, then an input error of $F'$ with a variance of $\sigma_{in}^2$ can lead to an output error variance of at most $L^2\sigma_{in}^2$. Taking $\sigma_{in}^2 = k_2^{-2}\Delta_{in}^{-2}\sigma_{boot}^2$ and counting in the error introduced by the final bootstrap itself, we get the final output error variance

$$\frac{L^2}{\Delta_{in}^2 k_2^2}\sigma_{boot}^2 + \sigma_{boot}^2.$$

Now we consider the cases when the distance between $x$ and $0, q/4$ or $3q/4$ is less than $\beta$, which means the bootstrapping error when evaluating $f'_C$ 'pushes' its output value out of $[\beta, q/4 - \beta] \bigcup [3q/4 + \beta, q - \beta]$. In these cases, we use $f_C'^{-1}(\beta), f_C'^{-1}(q/4 - \beta), f_C'^{-1}(3q/4 + \beta)$ and $f_C'^{-1}(q - \beta)$ as the recovered values of $x_0$, corresponding to the 3rd $\sim$ 6th cases in the expression of $f'_{eval}$. Since we 'align' $x$ to the nearest value in $[\beta, q/4 - \beta] \bigcup [3q/4 + \beta, q - \beta]$ before computing $f_C'^{-1}$ on it, part of the bootstrapping noise is corrected and the recovered value for $x_0$ has smaller error variance than the case of $x \in [\beta, \frac{q}{4} - \beta] \bigcup [\frac{3q}{4} + \beta, q - \beta]$. Thus, the final output error variance in these cases is also no more than $\frac{L^2}{\Delta_{in}^2 k_2^2}\sigma_{boot}^2 + \sigma_{boot}^2$.

Finally, the case of $x \in [q/4, 3q/4-1]$ will never happen when evaluating $f'_{eval}$ because the design of $f'_C$ has ensured that $x \in [0, q/4-1] \bigcup [3q/4, q-1]$. This case exists only to satisfy the negacyclicity constraint and to make the expression of $f'_{eval}$ complete. □

# B Noise Analysis for FDFB Methods

## B.1 Refined Noise Analysis of BFV Multiplication

Below is our refined noise analysis of BFV multiplication. In our analysis, only the constant terms of encrypted polynomials are used to store messages, while non-constant coefficients are close to zero. See [KPZ21] for more details about BFV multiplication.

**Theorem 10.** *Let $c_i = (\boldsymbol{b}_i, \boldsymbol{a}_i) = \mathrm{RLWE}_{\boldsymbol{s},N,Q}(\frac{Q}{p}m_i + e_i + \boldsymbol{e}_i)$ for $i = 0, 1$, where $e_i \sim N(0, \sigma_i^2)$ and $\boldsymbol{e}_i \sim N(0, \sigma_i'^2)^N$ and $m_0 \in \{0, \pm 1\}$. Given $c_i$, an auxiliary modulus $P$ and a base $B_{rl}$ for re-linearization, SampleExtract(BFVMult$(c_0, c_1), 0) = \frac{Q}{p}m_0m_1 + e$, where the variance of $e$ is*

$$\sigma_1^2 + \frac{p^2}{4}\sigma_0^2 + \frac{p^2}{Q^2}\sigma_0^2\sigma_1^2 + \sigma_1'^2 + \frac{Q^2}{P^2}\sigma_{ms}^2 + \frac{p^2}{4}\sigma_0'^2 + \frac{p^2}{Q^2}\sigma_0^2\sigma_1'^2 + \frac{p^2}{P^2}\sigma_0^2\sigma_{ms}^2 + \frac{p^2}{Q^2}\sigma_1^2\sigma_0'^2 + p^2\sigma_{ms}^2(\sigma_0^2 + \sigma_1^2) + N(\frac{p^2}{Q^2}\sigma_0'^2\sigma_1'^2 + \frac{p^2}{P^2}\sigma_0'^2\sigma_{ms}^2 + p^2\sigma_{ms}^2(\sigma_0'^2 + \sigma_1'^2) + \frac{p^2Q^2}{P^2}\sigma_{ms}^2\sigma_0'^2) + \sigma_{ms}^2 + \frac{N^2}{27} + d_{rl}\frac{B_{rl}^2}{4}N\sigma^2$$

*Proof.* First $c_1$ is modulus-switched to $\mathbb{Z}_P$, producing $c_1 = \mathrm{RLWE}_{\mathbf{s},N,P}(\frac{P}{p}m_1 + \frac{P}{Q}e_1 + \frac{P}{Q}\mathbf{e}_1 + \mathbf{e}_{ms})$. Then $c_0$ and $c_1$ are modded up to $\mathbb{Z}_{PQ}$. The messages encrypted in them are added by $\mathbf{u}_0Q$ and $\mathbf{u}_1P$ respectively, where $\mathbf{u}_0Q = \mathbf{b}_0 + \mathbf{a}_0\mathbf{s} - [\mathbf{b}_0 + \mathbf{a}_0\mathbf{s}]_Q$ and $\mathbf{u}_1P = \mathbf{b}_1 + \mathbf{a}_1\mathbf{s} - [\mathbf{b}_1 + \mathbf{a}_1\mathbf{s}]_P$. Then a tensor product between $c_0$ and $c_1$ outputs an RLWE encryption of $m_{prod} = (\frac{Q}{p}m_0 + e_0 + \mathbf{e}_0 + \mathbf{u}_0Q)(\frac{P}{p}m_1 + \frac{P}{Q}e_1 + \frac{P}{Q}\mathbf{e}_1 + \mathbf{e}_{ms} + \mathbf{u}_1P) \in \mathbb{Z}_{PQ}$ under extended secret keys. Finally, the ciphertext is multiplied by $p$, modulus-switched to $\mathbb{Z}_Q$ and re-linearized using the re-linearization keys. The output ciphertext is an RLWE encryption of $\frac{p}{P}m_{prod} + \mathbf{e}'_{ms} + \mathbf{e}_{rl}$, where $e_{rl}$ is the re-linearization error. The constant term of the encrypted polynomial is extracted as the output LWE ciphertext. Expanding $\frac{p}{P}m_{prod}$ gives the following.

$$\begin{aligned}\frac{p}{P}m_{prod} =& \frac{Q}{p}m_0m_1 \\ &+ m_0e_1 + m_1e_0 + \frac{p}{Q}e_0e_1 \\ &+ m_0\mathbf{e}_1 + \frac{Q}{P}m_0\mathbf{e}_{ms} + m_1\mathbf{e}_0 \\ &+ \frac{p}{Q}e_0\mathbf{e}_1 + \frac{p}{P}e_0\mathbf{e}_{ms} + \frac{p}{Q}e_1\mathbf{e}_0 + p(e_0\mathbf{u}_1 + e_1\mathbf{u}_0) \\ &+ \frac{p}{Q}\mathbf{e}_0\mathbf{e}_1 + \frac{p}{P}\mathbf{e}_0\mathbf{e}_{ms} + p(\mathbf{u}_0\mathbf{e}_1 + \mathbf{u}_1\mathbf{e}_0) + \frac{pQ}{P}\mathbf{u}_0\mathbf{e}_{ms}\end{aligned}$$

The first line of RHS is the desired message. Terms on line 2 are products between scalar values; those on lines 3 and 4 are products between scalars and polynomials; those in the last line are products between polynomials.

Each coefficient of $\mathbf{u}_i$ can be viewed as an inner product between the coefficients of $\mathbf{s}$ and $N$ random variables sampled from $U(-0.5, 0.5)$, which means $\mathbf{u}_i \sim N(0, \sigma_{ms}^2)^N$. Then the variance of the constant term of $\frac{p}{P}m_{prod}$ is

$$
\begin{aligned}
&\sigma_m^2 \\
=&\sigma_1^2 + \frac{p^2}{4}\sigma_0^2 + \frac{p^2}{Q^2}\sigma_0^2\sigma_1^2 \\
+&\sigma_1'^2 + \frac{Q^2}{P^2}\sigma_{ms}^2 + \frac{p^2}{4}\sigma_0'^2 \\
+&\frac{p^2}{Q^2}\sigma_0^2\sigma_1'^2 + \frac{p^2}{P^2}\sigma_0^2\sigma_{ms}^2 + \frac{p^2}{Q^2}\sigma_1^2\sigma_0'^2 + p^2\sigma_{ms}^2(\sigma_0^2+\sigma_1^2) \\
+&N(\frac{p^2}{Q^2}\sigma_0'^2\sigma_1'^2 + \frac{p^2}{P^2}\sigma_0'^2\sigma_{ms}^2 + p^2\sigma_{ms}^2(\sigma_0'^2+\sigma_1'^2) + \frac{p^2Q^2}{P^2}\sigma_{ms}^2\sigma_0'^2)
\end{aligned}
$$

The modulus switching from $\mathbb{Z}_{PQ}$ to $\mathbb{Z}_Q$ is slightly different from a regular one since a part of the rounding error is multiplied with $\mathbf{s}^2$. Heuristically we can estimate its error variance as $\sigma_{ms}'^2 = \sigma_{ms}^2 + \frac{N^2}{27}$.

The re-linearization process of BFV is essentially a $R_Q \times \mathrm{RLWE}'(\mathbf{s}^2)$ multiplication. Its error variance $\sigma_{rl}^2$ is given by $d_{rl}\frac{B_{rl}^2}{4}N\sigma^2$, where $d_{rl} = \lceil \log_{B_{rl}}(Q) \rceil$ and $\sigma^2$ is the encryption error variance.

Summing up $\sigma_m^2$, $\sigma_{ms}'^2$ and $\sigma_{rl}^2$ gives the variance of $e$ in the output ciphertext. □

**Remark.** For ordinary BFV multiplication where $m_i$ and $e_i$ are polynomials instead of scalars, the terms in line $1 \sim 3$ in the expression of $\sigma_m^2$ need to be multiplied by $N$. The overflow of $m_0 m_1$ modulo $p$ also introduces an additional term. In **FDFB-BFVMult** (**WoPPBS$_1$-Refine**), the bootstrapping error $e_i$ is magnitudes larger than the key switching error $\mathbf{e}_i$. This means the dominating term of $\sigma_m^2$ is $p^2\sigma_{ms}^2(\sigma_0^2+\sigma_1^2)$ (corresponding to $p(e_0\mathbf{u}_1 + e_1\mathbf{u}_0)$). In contrast, **WoP-PBS** estimates this term to be $N$ times larger by treating the multiplication between $e_i$ and $\mathbf{u}_i$ as polynomial-polynomial multiplication, which leads to inefficient parameters. We state this observation as Proposition 1.

**Proposition 1.** *For **WoP-PBS** and **FDFB-BFVMult** (**WoPPBS$_1$-Refine**), the dominating terms of output variance are $Np^2\sigma_{ms}^2(\sigma_0^2+\sigma_1^2)$ and $p^2\sigma_{ms}^2(\sigma_0^2+\sigma_1^2)$ respectively, where $\sigma_i^2$ are the error variances in LWE ciphertexts being multiplied.*

## B.2 Proofs of FDFB Algorithms' Correctness and Output Noise

### B.2.1 FDFB-Compress

**Proof of Theorem 1.**

*Proof.* If $m + \frac{q}{2p} \in [0, \frac{q}{2} - 1]$, $f_{eval}(f_C(\frac{q}{p}m' + e + \frac{q}{2p}) + e_{boot}) = f_{eval}(\frac{q}{2p}(\lfloor \frac{p}{q}(\frac{q}{p}m' + e + \frac{q}{2p}) \rfloor + \frac{1}{2}) + e_{boot}) = f_{eval}(\frac{q}{2p}(m' + \frac{1}{2}) + e_{boot}) = \lfloor \frac{q}{p}F(\lfloor \frac{2p}{q}(\frac{q}{2p}(m' + \frac{1}{2}) + e_{boot}) \rfloor) \rceil = \lfloor \frac{q}{p}F(m') \rceil$. The first and third equations simply substitute $f_C$ for (1) and $f_{eval}$ for (2); the second equation uses the condition that $|e| < \frac{q}{2p}$; the last equation follows from $|e_{boot}| = \beta < \frac{q}{4p}$. The case of $m + \frac{q}{2p} \in [\frac{q}{2}, q-1]$ can be proven in a similar way. Thus the final output of **FDFB-Compress** is $\mathrm{LWE}(\frac{q}{p}F(m') + e_{boot})$. Again, since $|e_{boot}| = \beta < \frac{q}{4p}$, the output is a valid LWE ciphertext. The ct in line 1 of Algorithm 1 has an error variance of $\sigma_{boot}^2$ because it is the output of a functional bootstrap. □

### B.2.2 FDFB-CancelSign

**Proof of Theorem 2.**

*Proof.* $|e| < \frac{q}{4p}$ and $|e'| < \frac{q}{4p}$ ensures the validity of the input and output ciphertext of **FDFB-CancelSign**. We prove the theorem in two steps. (1) $\mathrm{ct}_1 = \mathrm{LWE}((-1)^{\mathrm{MSB}}\lfloor\frac{Q}{p}F(m')\rceil)$; (2) line 3 to line 4 multiplies the message in $\mathrm{ct}_1$ by $(-1)^{\mathrm{MSB}}$.

For the first step, let $\mathrm{ct} = \mathrm{LWE}(\frac{q}{2p}m' + e + \frac{q}{4p} + \mathrm{MSB}\frac{q}{2})$. Then when $\mathrm{MSB} = 0$, $\mathrm{ct}_1 = \mathrm{LWE}(\lfloor\frac{Q}{p}F(\lfloor\frac{2p}{q}(\frac{q}{2p}m' + e + \frac{q}{4p})\rfloor)\rceil + e_{boot}) = \mathrm{LWE}(\lfloor\frac{Q}{p}F(m')\rceil + e_{boot})$, where the second equation follows from $|e| < \frac{q}{4p}$. When $\mathrm{MSB} = 1$, $\mathrm{ct}_1 = \mathrm{LWE}(-\lfloor\frac{Q}{p}F(m')\rceil + e_{boot})$ as defined by (3), which finishes the proof.

For the second step, suppose $\mathrm{ct}_1 = \mathrm{LWE}_{\vec{s},n,q}(m_1)$ at line 3, then $\mathrm{ct}_{pk}$ encrypts a polynomial whose coefficients are $m_1 + e_{pk}$. Since the value encrypted in ct lies within $[\mathrm{MSB}\frac{q}{2}, \mathrm{MSB}\frac{q}{2} + \frac{q}{2} - 1]$, after blind rotating $\mathrm{ct}_{pk}$ by ct, the constant term of $\mathrm{ct}_{pk}$ equals $(-1)^{\mathrm{MSB}}(m_1 + e_{pk}) + e_{acc} \approx (-1)^{\mathrm{MSB}}m_1$.

Now we prove the output error variance of **FDFB-CancelSign**. The output of the first functional bootstrap has an error variance of $\sigma^2_{acc}$. After modulus switching to $q_{pk}$ and LWE-to-RLWE packing, the variance becomes $\sigma^2_{acc} + (\frac{Q}{q_{pk}})^2\sigma^2_{ms} + \sigma^2_{pk}$. The second bootstrap adds $\sigma^2_{acc}$ to the error variance. Finally, modulus-switching the ciphertext gives $(\frac{q}{Q})^2(2\sigma^2_{acc} + (\frac{Q}{q_{pk}})^2\sigma^2_{ms} + \sigma^2_{pk}) + \sigma^2_{com} = (\frac{q}{Q})^2(2\sigma^2_{acc} + \sigma^2_{pk}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms} + \sigma^2_{com}$ □

#### B.2.3 FDFB-Select

**Proof of Theorem 3.**

*Proof.* $|e| < \frac{q}{2p}$ and $|e'| < \frac{q}{2p}$ ensures the validity of input and output ciphertext of **FDFB-Select**. The desired value is encrypted in $\mathrm{ct}_{pos}$ when $\mathrm{MSB} = 0$ and in $\mathrm{ct}_{neg}$ when $\mathrm{MSB} = 1$. Then it only remains to prove that line 4 to line 6 selects the correct ciphertext from $\mathrm{ct}_{pos}$ and $\mathrm{ct}_{neg}$. Since $|e| < \frac{q}{2p}$, $\mathrm{ct}_{sgn} = \mathrm{LWE}(f_{sgn}(\frac{q}{p}m' + e + \frac{q}{2p}) + e_{boot})$ lies in $[\frac{q}{8}(-1)^{\mathrm{MSB}} - \beta, \frac{q}{8}(-1)^{\mathrm{MSB}} + \beta]$. Applying $\beta < \frac{q}{8}$ and $q = 2N$, the value encrypted in $\mathrm{ct}_{sgn}$ belongs to $[0, \frac{N}{2} - 1]$ or $[-\frac{N}{2}, -1]$ when $\mathrm{MSB} = 0$ or 1 respectively. Denote the values encrypted in $\mathrm{ct}_{pos}$ and $\mathrm{ct}_{neg}$ as $m_{pos}$ and $m_{neg}$. Then $\mathrm{ct}_{pk}$ encrypts a polynomial whose $i$-th coefficient is $m_{pos} + e_{pk}$ for $i \in [0, \frac{N}{2} - 1]$ and $-m_{neg} + e_{pk}$ for $i \in [\frac{N}{2}, N - 1]$. After blind-rotated by $\mathrm{ct}_{sgn}$, $\mathrm{ct}_{pk}$ has a constant term of $m_{pos} + e_{pk} + e_{acc}$ for $\mathrm{MSB} = 0$ and $m_{neg} - e_{pk} + e_{acc}$ for $\mathrm{MSB} = 1$.

In the non-multi-value version of **FDFB-Select**, two bootstrap results are modulus-switched to $q_{pk}$ and packed into an RLWE ciphertext. Hence, each coefficient of the packed polynomial has an error variance of $\sigma^2_{acc} + (\frac{Q}{q_{pk}})^2\sigma^2_{ms} + 2\sigma^2_{pk}$. As in **FDFB-CancelSign**, the final bootstrap adds another $\sigma^2_{acc}$ to the error variance. Then after performing the last three steps of FHEW/TFHE bootstrap, the output error variance is $(\frac{q}{Q})^2(2\sigma^2_{acc} + (\frac{Q}{q_{pk}})^2\sigma^2_{ms} + 2\sigma^2_{pk}) + \sigma^2_{com} = (\frac{q}{Q})^2(2\sigma^2_{acc} + 2\sigma^2_{pk}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms} + \sigma^2_{com}$.

For the multi-value version, $\mathrm{ct}_{pos}, \mathrm{ct}_{neg}$ and $\mathrm{ct}_{sgn}$ have error variances of $p(p-1)^2\sigma^2_{acc}$ (the $p(p-1)^2$ term comes from multi-value bootstrap). However, the error in $\mathrm{ct}_{sgn}$ will not affect the error in the output ciphertext as long as it is bounded by $\frac{N}{4}$. Then the packed polynomial has an error variance of $p(p-1)^2\sigma^2_{acc} + (\frac{Q}{q_{pk}})^2\sigma^2_{ms} + 2\sigma^2_{pk}$. Again, the final blind rotation adds $\sigma^2_{acc}$ to the error variance, and after the last three steps of FHEW/TFHE bootstrap, the output error variance is $(\frac{q}{Q})^2((p(p-1)^2 + 1)\sigma^2_{acc} + 2\sigma^2_{pk}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms} + \sigma^2_{com}$.

□

#### B.2.4 FDFB-SelectAlt

**Proof of Theorem 4.**

*Proof.* $|e| < \frac{q}{2p}$ and $|e'| < \frac{q}{2p}$ ensures the validity of input and output ciphertext of **FDFB-SelectAlt**. $\mathrm{ct}_{hdiff}$ and $\mathrm{ct}_{hsum}$ encrypt $m_{hdiff} = \frac{f_{neg}(m)-f_{pos}(m)}{2}$ and $m_{hsum} = \frac{f_{neg}(m)+f_{pos}(m)}{2}$ respectively. Similar to the proof of Theorem 2, $\mathrm{Boot}[\mathrm{ct}](\mathrm{ct}_{pk}) = \mathrm{LWE}((-1)^{\mathrm{MSB}} m_{hdiff})$. Then the returned result encrypts $\frac{f_{neg}(m)+f_{pos}(m)}{2} - (-1)^{\mathrm{MSB}} \frac{f_{neg}(m)-f_{pos}(m)}{2}$, which equals $f_{pos}(m)$ when $\mathrm{MSB} = 0$ and $f_{neg}(m)$ when $\mathrm{MSB} = 1$. Applying $m = \frac{q}{p}m' + e + \frac{q}{2p}$ and $|e| < \frac{q}{2p}$, we have $f_{neg}(m) = f_{pos}(m) = \lfloor \frac{Q}{p} F(m') \rceil$ when $\mathrm{MSB} = 0, 1$ respectively.

For the non-multi-value version of **FDFB-SelectAlt**, the error variance of the blind-rotated packed polynomial is the same as that in **FDFB-CancelSign**. The addition by $\mathrm{ct}_{hsum}$ in the last step of **FDFB-SelectAlt** adds $\sigma_{acc}^2$ to this variance. Thus, the final output variance is $(\frac{q}{Q})^2 \sigma_{acc}^2$ greater than that of **FDFB-CancelSign** and equals to $(\frac{q}{Q})^2(3\sigma_{acc}^2 + 2\sigma_{pk}^2) + (\frac{q}{q_{pk}})^2 \sigma_{ms}^2 + \sigma_{com}^2$.

The multi-value version of **FDFB-SelectAlt** has a different workflow from the non-multi-value version. It computes $\frac{1-\mathrm{SGN}}{2}(f_{neg} - f_{pos}) + f_{pos}$ instead of $\frac{f_{neg}+f_{pos}}{2} - \mathrm{SGN}\frac{f_{neg}-f_{pos}}{2}$. In this way, the error variance of $\frac{f_{neg}-f_{pos}}{2}$ is $p(p-1)^2\sigma_{acc}^2$ and that of $f_{pos}$ is $4p(p-1)^2\sigma_{acc}^2$. Along with the error in $\mathrm{SGN}\frac{f_{neg}-f_{pos}}{2}$, the total error is $6p(p-1)^2\sigma_{acc}^2$. In contrast, if we follow the original workflow, the error variance will be $8p(p-1)^2\sigma_{acc}^2$, which is slightly larger. Taking into account the last three steps of bootstrap, the final output error is $(\frac{q}{Q})^2((6p(p-1)^2 + 1)\sigma_{acc}^2 + \sigma_{pk}^2) + (\frac{q}{q_{pk}})^2 \sigma_{ms}^2 + \sigma_{com}^2$.

□

### B.2.5 FDFB-BFVMult (WoPPBS$_1$-Refine)

#### Proof of Theorem 5.

*Proof.* $|e| < \frac{q}{4p}$ and $|e'| < \frac{q}{4p}$ ensures the validity of input and output ciphertext of **WoPPBS$_1$-Refine**. Similar to the proof of Theorem 2, $\mathrm{ct}_0 = \mathrm{RLWE}((-1)^{\mathrm{MSB}} \lfloor \frac{Q}{p} F(m') \rceil)$. Moreover, $\mathrm{ct}_{sgn} = \mathrm{RLWE}(f_{sgn1}(\frac{q}{2p}m' + e + \frac{q}{4p})) = \mathrm{RLWE}((-1)^{\mathrm{MSB}} \lfloor \frac{Q}{p} \rceil)$. Computing their BFV product gives $\mathrm{ct}_{prod} = \mathrm{RLWE}(\lfloor \frac{Q}{p} F(m') \rceil)$. Finally, sample extraction, key switching and modulus switching convert $\mathrm{ct}_{prod}$ into the desired output format.

Recall that the dominating term of BFV multiplication used in **WoPPBS$_1$-Refine** is $p^2\sigma_{ms}^2(\sigma_0^2 + \sigma_1^2)$, where $\sigma_i^2$ is the error variance of the LWE ciphertexts to pack. For the non-multi-value version of **WoPPBS$_1$-Refine**, $\sigma_i^2 = \sigma_{acc}^2$ because the LWE ciphertexts to pack are freshly blind-rotated. By replacing $\sigma_{ms}^2 \approx \frac{N}{18}$, we known the error variance of $\mathrm{ct}_{prod}$ is $\frac{N}{9}p^2\sigma_{acc}^2$. The output error variance is given by applying the last three steps of bootstrap, which is equal to $(\frac{q}{Q})^2 \frac{N}{9} p^2 \sigma_{acc}^2 + \sigma_{com}^2$.

For the multi-value version of **WoPPBS$_1$-Refine**, $\sigma_1^2$ is equal to $p(p-1)^2\sigma_{acc}^2$ while $\sigma_0^2 = 4\sigma_{acc}^2 \ll \sigma_1^2$ because the L2 norm of $\mathrm{TV}_1$ for $f_{sgn1}$ is 4. Following a similar analysis of the non-multi-value version, the final output variance is approximately $(\frac{q}{Q})^2 \frac{N}{18} p^3 (p-1)^2 \sigma_{acc}^2 + \sigma_{com}^2$. □

#### Proof of Theorem 6.

*Proof.* $|e| < \frac{q}{2p}$ and $|e'| < \frac{q}{2p}$ ensures the validity of input and output ciphertext of **FDFB-BFVMult**. Denote the plaintext encrypted in $\mathrm{ct}_{diff}$ and $\mathrm{ct}_{pos}$ as $m_{diff} = f_{neg}(m) - f_{pos}(m)$ and $m_{pos} = f_{pos}(m)$ respectively. Also, $\mathrm{ct}_{sgn} = \mathrm{RLWE}(f'_{sgn}(m) + \lfloor \frac{Q}{2p} \rceil) = \mathrm{RLWE}(\lfloor \frac{Q}{2p} \rceil (1 - (-1)^{\mathrm{MSB}})) = \mathrm{RLWE}(\mathrm{MSB}\lfloor \frac{Q}{p} \rceil)$. Then $\mathrm{BFVMult}(\mathrm{ct}_{sgn}, \mathrm{ct}_{diff}) + \mathrm{ct}_{pos}$ encrypts $\mathrm{MSB}(f_{neg}(m) - f_{pos}(m)) + f_{pos}(m)$, which equals $f_{neg}(m)$ when $\mathrm{MSB} = 1$ and $f_{pos}(m)$ when $\mathrm{MSB} = 0$. Finally, the result is converted to the output format as in **WoPPBS$_1$-Refine**.

The output error analysis for **FDFB-BFVMult** is similar to **WoPPBS$_1$-Refine**. Note that for the multi-value version of **FDFB-BFVMult**, we need to compute an encryption of $\frac{Q}{2p}\mathrm{SGN}$ to obtain $\frac{Q}{p}\mathrm{MSB}$ (where $\mathrm{SGN} = 1-2\mathrm{MSB}$), which means the output plaintext space for multi-value bootstrap is $2p$ instead of $p$. The $\mathrm{TV}_1$ for $f_{sgn2}$ still has an L2 norm of 4, while the $\mathrm{TV}_1$ for $f_{diff}$ has an L2 norm of $4p(p-1)^2$ at most. This explains where the coefficient of $\sigma^2_{acc}$ for **FDFB-BFVMult** is $\frac{2N}{9}p^3(p-1)^2$ instead of $\frac{N}{18}p^3(p-1)^2$ in **WoPPBS$_1$-Refine**. □

## B.3 Correctness Proofs of Homormophic Digit Decomposition Algorithms

### B.3.1 HomDecomp-Reduce

**Proof of Theorem 7.**

*Proof.* It suffices to prove that the modulus switching noise will not cause an overflow to the higher digits. Denote the message encrypted in ct at line 4 as $m_{high}q + m_{low}(m_{low} \in [0, q-1])$. Note that $m_{low}$ is the message encrypted in ct′. Then ct encrypts $m_1 = m_{high}q + m_{low} + f_{red}(m_{low}) + e_{boot} - \frac{q}{2}$ at line 6. By the definition of $f_{red}$, $m_{low} + f_{red}(m_{low}) \in [\frac{q}{4}, \frac{3q}{4} - 1], \forall m_{low} \in \mathbb{Z}_q$, meaning $m_{err} = m_1 - (m_{high}q + e_{boot}) \in [-\frac{q}{4}, \frac{q}{4} - 1]$. Modulus switching ct down by $B$ will produce an encryption of $m_{high}\frac{q}{B} + \frac{e_{boot}}{B} + e_{ms} + \frac{m_{err}}{B}$. $\frac{e_{boot}}{B} + e_{ms} + \frac{m_{err}}{B}$ needs to be bounded by $\frac{q}{2B}$ to prevent $m_{high}$ from being destroyed by an overflow. Applying $|m_{err}| \le \frac{q}{4}$ gives the desired result. □

### B.3.2 HomDecomp-FDFB

**Proof of Theorem 9.**

*Proof.* $\sigma^2_f$ can be obtained by assigning $L = \Delta_{in} = \Delta_{out} = 1$ in Theorem 8. Denote the message encrypted in ct at line 4 as $m_{high}q_0 + m_{low}(m_{low} \in [0, q-1])$. Note that $m_{low}$ is also the message encrypted in ct′. Then ct encrypts $m_1 = m_{high}q + m_{low} - (m_{low} + e_f) = m_{high}q - e_f$ at line line 6. Modulus switching ct down by $B$ will produce an encryption of $m_{high}\frac{q}{B} - \frac{e_f}{B} + e_{ms}$. $-\frac{e_f}{B} + e_{ms}$ needs to be bounded by $\frac{q}{2B}$ to prevent $m_{high}$ from being destroyed by an overflow, which leads to our conclusion directly. □

# C Miscellaneous

## C.1 Full Algorithms of PackingKS

# D Tables

This section presents (1) the full tables of noise analysis for all previous and our FDFB algorithms (Table 8), as well as homomorphic digit decomposition algorithms (Table 9); (2) the full table of parameters used in the benchmarks and the results of the benchmarks (Table 11, Table 12, Table 10).

$B_{g0}$ is the base of RLWE′ ciphertexts used in **FDFB-KS**. $P$ is the auxiliary prime used in BFV multiplication [KPZ21]. $B_{rl}$ is the base of re-linearization keys for BFV multiplication. The non-multi-value bootstraps in FDFB methods that use the multi-value bootstrap are accelerated with larger $B_g$. This extra $B_g$ is listed as $B'_g$ in the table. The LWE dimensions $n35 = 1340$ and $n20 = 760$ correspond to the parameter sets in Table 5. $P53$ and $Q53$ are primes that approximately equal to $2^{53}$. The "$d = 1, \frac{N}{2}, N$" columns represent whether packing keys for the given value of $d$ are generated.

**Algorithm 9:** PackingKS

**input :** Base $B$ and modulus $q_{ks}$ for key switching
**input :** $d$, the number of coefficients to pack
**input :** An LWE ciphertext $c = \mathrm{LWE}_{\vec{s},n,q}(m+e)$
**input :** Packing keys
$\mathrm{ksk}_{i,j,k} = \mathrm{RLWE}_{\mathbf{s}',n',q'}(\lfloor \frac{q'}{q_{ks}} \vec{s}_i \cdot j \cdot B^k \cdot (1 + X + \ldots + X^{d-1}) \rceil)$,
$i \in [\![n]\!], j \in [\![B]\!], k \in [\![\lceil \log_B(q_{ks}) \rceil]\!]$
**output :** An RLWE ciphertext
$\mathrm{RLWE}_{\mathbf{s}',n',q'}(\lfloor \frac{q'}{q}(m+e) + \frac{q'}{q_{ks}} e_{ms} \rceil (1 + X + \ldots + X^{d-1}) + \mathbf{e}_{ks})$

```
(b, a⃗) ← ModSwitch(c, q_ks)
ct ← (⌊q'/q_ks · b⌉ · (1 + X + ... + X^{d−1}), 0)
for i ← 0 to n − 1 do
    a⃗_i = Σ_{k=0}^{⌈log_B q_ks⌉−1} a_{i,k} · B^k, a_{i,k} ∈ [−⌊B/2⌋, ⌊(B−1)/2⌋]
    for k ← 0 to ⌈log_B q_ks⌉ − 1 do
        ct ← ct + ksk_{i,a_{i,k},k}
return ct
```

Table 8: Output error variance and the number of bootstraps required for all FDFB algorithms. Those below the mid-line are proposed in this paper. For discrete LUT evaluation, we assume the error variance of input ciphertext is equal to $\sigma_{boot}^2$ as in [LMP22]. The final output variance $\sigma_{out}^2 = \sigma_{core}^2 + \sigma_{com}^2$. FDFB algorithms that cannot reach the maximum plaintext modulus $p_{max} = \frac{q}{2\beta} = \frac{N}{\beta}$ are explicitly footnoted, and those unmentioned can reach $p_{max}$. We only measure the noise introduced by FDFB. $L$ is the Lipschitz constant of the evaluated function.

| FDFB Variant | Core Output Variance $\sigma_{core}^2$ | Num of BTS |
|---|---|---|
| **EvalFunc** [LMP22][1] | $(\frac{q}{Q})^2\sigma_{acc}^2$, intermediate: $2\sigma_{boot}^2$ | 2 |
| **FDFB-KS** [KS22][2] | $(\frac{q}{Q})^2(\sigma_{acc}^2 + d_{g1}\frac{B_{g1}^2}{4}N(\sigma_{acc}^2 + (\frac{Q}{q_{pk}})^2\sigma_{ms}^2 + \sigma_{ks}^2))$ | $d_{g1}+1$ |
| **Comp** [CZB+22] | $2(\frac{q}{Q})^2\sigma_{acc}^2$, intermediate[5]: $(\frac{q}{Q})^2\sigma_{acc}^2$ | 4 |
| **Comp** [CZB+22][*3] | $2(\frac{q}{Q})^2\sigma_{acc}^2$, intermediate[5]: $(\frac{q}{Q})^2(2p^2-4)\sigma_{acc}^2$ | 3 |
| **WoP-PBS$_1$** [CLOT21][§†] | $(\frac{q}{Q})^2\frac{N^2}{9}p^2\sigma_{acc}^2$ | 2 |
| **WoP-PBS$_1$** [CLOT21][§ † *] | $(\frac{q}{Q})^2\frac{N^2}{18}p^3(p-1)^2\sigma_{acc}^2$ | 1 |
| **WoP-PBS$_2$** [CLOT21][†] | $(\frac{q}{Q})^2\frac{2N^2}{9}p^2\sigma_{acc}^2$ | 3 |
| **WoP-PBS$_2$** [CLOT21][†*] | $(\frac{q}{Q})^2\frac{2N^2}{9}p^3(p-1)^2\sigma_{acc}^2$ | 1 |
| **WoPPBS$_1$-Refine**[§†] | $(\frac{q}{Q})^2\frac{N}{9}p^2\sigma_{acc}^2$ | 2 |
| **WoPPBS$_1$-Refine**[§ † *] | $(\frac{q}{Q})^2\frac{N}{18}p^3(p-1)^2\sigma_{acc}^2$ | 1 |
| **FDFB-BFVMult**[†] | $(\frac{q}{Q})^2\frac{N}{9}p^2\sigma_{acc}^2$ | 3 |
| **FDFB-BFVMult**[†*] | $(\frac{q}{Q})^2\frac{2N}{9}p^3(p-1)^2\sigma_{acc}^2$ | 1 |
| **FDFB-Compress** [§4] | $(\frac{q}{Q})^2\sigma_{acc}^2$, intermediate: $\sigma_{boot}^2$ | 2 |
| **FDFB-CancelSign** [§] | $(\frac{q}{Q})^2(2\sigma_{acc}^2 + \sigma_{ks}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2$ | 2 |
| **FDFB-Select** | $(\frac{q}{Q})^2(2\sigma_{acc}^2 + 2\sigma_{ks}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2$ | 4 |
| **FDFB-Select** [*] | $(\frac{q}{Q})^2((p(p-1)^2+1)\sigma_{acc}^2 + 2\sigma_{ks}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2$ | 2 |
| **FDFB-SelectAlt** | $(\frac{q}{Q})^2(3\sigma_{acc}^2 + \sigma_{ks}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2$ | 3 |
| **FDFB-SelectAlt** [*] | $(\frac{q}{Q})^2((6p(p-1)^2+1)\sigma_{acc}^2 + \sigma_{ks}^2) + (\frac{q}{q_{pk}})^2\sigma_{ms}^2$ | 2 |

§ FDFB algorithms that introduce an additional MSB. The maximum plaintext modulus $p$ is 1 bit less than $p_{max}$, and the parameters $q$ and $\Delta_{in}$ are halved compared to the cases where the additional MSB is not required.

† FDFB algorithms that use BFV multiplication. $\sigma_{core}^2$ is merely given a simplified version for brevity (see Appendix B for the details).

* FDFB algorithms that make use of multi-value bootstrap.

1 For **EvalFunc**, besides doubling the input modulus, it is also required that the intermediate error after preprocessing is bounded by $2\sigma_{boot}^2 \leq \frac{q}{4p}$ to ensure correctness. Consequently, the maximum $p$ of **EvalFunc** is 1.5 bits less than $p_{max}$.

2 $d_{g1} = \lceil \log_{B_{g1}}(Q) \rceil$, $B_{g1}$ is the decomposition base for RLWE′.

3 For **Comp** *, the two $\mathrm{TV}_1$'s used in multi-value bootstrap are constant polynomials independent of the LUT. $|\mathrm{TV}_1|_2^2 \leq 2p^2 - 4$.

4 The intermediate error has a variance of $\sigma_{boot}^2$ and needs to be bounded by $\frac{q}{4p}$ for correctness.

5 In **Comp**, it is required that $\mathrm{bnd} \cdot \sqrt{\sigma_{inter}^2 + \sigma_{com}^2} \leq \frac{q}{2p}$ for correctness, where $\sigma_{inter}^2$ is the variance of the core intermediate error.

Table 9: Maximum decomposition base and the number of bootstraps required for homomorphic decomposition algorithms.

| Name of Variant | Requirement for Decomposition Base $B$ | Num of BTS | Other Requirements |
|---|---|---|---|
| **HomFloor** [LMP22] | $\text{bnd}\sqrt{(1+B^{-2})\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{2B}$ | 2 | Cannot decompose extracted CKKS ciphertexts |
| **HomFloorAlt** [LMP22] | $\text{bnd}\sqrt{B^{-2}\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{2B}$ | 3 | $q > 8\sqrt{2}\beta$ |
| **HomDecomp-Reduce** | $\text{bnd}\sqrt{B^{-2}\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{4B}$ | 1 | - |
| **HomDecomp-FDFB** | $\text{bnd}\sqrt{B^{-2}(1+k_2^{-2})\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{2B}$ | 2 | - |

Table 10: Running Time of Homomorphic Digit Decomposition Methods under $\mathrm{PARAM}_{\mathrm{decomp}}$.

| $\log_2(q)$ | Running Time (ms) | | | |
|---|---|---|---|---|
| | **HomFloor** | **HomFloorAlt** | **HomDecomp-Reduce** | **HomDecomp-FDFB** |
| 13 | 1070 | 1600 | 533 | 1060 |
| 14 | 1060 | 1590 | 534 | 1060 |
| 15 | 1060 | 1600 | 537 | 1060 |
| 16 | 1060 | 1600 | 540 | 1060 |
| 17 | 2130 | 1590 | 1070 | 1060 |
| 18 | 2130 | 3200 | 1060 | 2130 |
| 19 | 2130 | 3230 | 1060 | 2130 |
| 20 | 2130 | 3210 | 1060 | 2130 |
| 21 | 3190 | 3190 | 1600 | 2120 |
| 22 | 3190 | 3190 | 1600 | 2120 |
| 23 | 3190 | 4800 | 1600 | 3190 |
| 24 | 3190 | 4790 | 1600 | 3190 |
| 25 | 4260 | 4780 | 2130 | 3190 |
| 26 | 4260 | 4790 | 2130 | 3190 |
| 27 | 4260 | 4790 | 2130 | 3180 |
| 28 | 4250 | 6380 | 2130 | 4250 |
| 29 | 5310 | 6380 | 2660 | 4250 |

Table 11: Parameters and Performance for Benchmark under $\text{PARAM}_{\text{decomp}}$, Sorted in Increasing Order by Running Time.

| Name | Time(ms) | $p$ | $n$ | $N$ | $q$ | $Q$ | $q_{ks}$ | $B_{ks}$ | $B_g$ | $B_{g0}$ | $q_{pk}$ | $B_{pk}$ | $B'_g$ | $d = 1,$ | $\frac{N}{2},$ | $N$ | $P$ | $B_{rl}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **WoPPBS$_1$-Refine*** | 777 | 16 | $n35$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{25}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **FDFB-BFVMult*** | 785 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{25}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **FDFB-Compress** | 1050 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **EvalFunc** | 1060 | 8 | $n35$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **FDFB-CancelSign** | 1060 | 16 | $n35$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | $2^{15}$ | $2^5$ | | - | - | + | | |
| **FDFB-BFVMult*** | 1150 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{11}$ | | $2^{25}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **WoP-PBS$_1$*** | 1160 | 16 | $n35$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{11}$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **WoP-PBS$_2$*** | 1200 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{11}$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **FDFB-SelectAlt** * | 1240 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{15}$ | $2^5$ | $2^{27}$ | - | - | + | | |
| **FDFB-Select** * | 1250 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{15}$ | $2^5$ | $2^{27}$ | - | + | - | | |
| **FDFB-SelectAlt** * | 1250 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{15}$ | $2^5$ | $2^{27}$ | - | - | + | | |
| **FDFB-Select** * | 1260 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{15}$ | $2^5$ | $2^{27}$ | - | + | - | | |
| **WoPPBS$_1$-Refine** | 1470 | 16 | $n35$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{25}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **WoP-PBS$_1$** | 1490 | 16 | $n35$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **Comp** * | 1580 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{25}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **FDFB-SelectAlt** | 1580 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | $2^{15}$ | $2^5$ | | - | - | + | | |
| **FDFB-SelectAlt** | 1600 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | $2^{15}$ | $2^5$ | | - | - | + | | |
| **Comp** * | 1760 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | | | $2^{27}$ | - | - | - | | |
| **WoP-PBS$_2$*** | 1930 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^6$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **Comp** | 2090 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **Comp** | 2090 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **FDFB-Select** | 2100 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | $2^{15}$ | $2^5$ | | - | + | - | | |
| **FDFB-Select** | 2100 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | $2^{15}$ | $2^5$ | | - | + | - | | |
| **FDFB-BFVMult** | 2180 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{25}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **WoP-PBS$_2$** | 2230 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **WoP-PBS$_2$** | 2780 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{14}$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **FDFB-KS** | 5360 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{11}$ | $2^{15}$ | $2^{35}$ | $2^5$ | | + | - | - | | |
| **FDFB-KS** | 5440 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{14}$ | $2^{12}$ | $2^{35}$ | $2^5$ | | + | - | - | | |
| **FDFB-KS** | 5770 | 16 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | $2^8$ | $2^{30}$ | $2^5$ | | + | - | - | | |
| **FDFB-KS** | 6340 | 32 | $n35$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{14}$ | $2^{10}$ | $2^{35}$ | $2^5$ | | + | - | - | | |

Table 12: Parameters and Performance for Benchmark under $\text{PARAM}_{\text{fast}}$, Sorted in Increasing Order by Running Time.

| Name | Time(ms) | $p$ | $n$ | $N$ | $q$ | $Q$ | $q_{ks}$ | $B_{ks}$ | $B_g$ | $B_{g0}$ | $q_{pk}$ | $B_{pk}$ | $B'_g$ | $d = 1, \frac{N}{2}, N$ | | | $P$ | $B_{rl}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **WoPPBS$_1$-Refine***| 458 | 16 | $n20$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{20}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **FDFB-BFVMult*** | 458 | 16 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{20}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **FDFB-BFVMult*** | 573 | 32 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{14}$ | | $2^{25}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **EvalFunc** | 598 | 16 | $n20$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **FDFB-Compress** | 598 | 16 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **FDFB-CancelSign** | 611 | 16 | $n20$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | $2^{15}$ | $2^5$ | | - | - | + | | |
| **FDFB-Select** * | 621 | 16 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | $2^{15}$ | $2^5$ | | - | + | - | | |
| **WoP-PBS$_1$***| 682 | 16 | $n20$ | $2^{11}$ | $2^{11}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{11}$ | | $2^{25}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **FDFB-SelectAlt** * | 717 | 16 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{15}$ | $2^5$ | $2^{27}$ | - | - | + | | |
| **FDFB-SelectAlt** * | 718 | 32 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{15}$ | $2^5$ | $2^{27}$ | - | - | + | | |
| **FDFB-Select** * | 724 | 32 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | $2^{15}$ | $2^5$ | $2^{27}$ | - | + | - | | |
| **WoP-PBS$_2$***| 735 | 16 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{11}$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **Comp** * | 897 | 16 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{27}$ | | | | | - | - | - | | |
| **WoP-PBS$_2$***| 942 | 32 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^8$ | | $2^{30}$ | $2^5$ | | + | - | - | $P53$ | $2^{27}$ |
| **Comp** * | 985 | 32 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | | | | $2^{27}$ | - | - | - | | |
| **FDFB-KS** | 2940 | 16 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{18}$ | $2^9$ | $2^{30}$ | $2^5$ | | + | - | - | | |
| **FDFB-KS** | 3110 | 32 | $n20$ | $2^{11}$ | $2^{12}$ | $Q53$ | $2^{20}$ | $2^5$ | $2^{11}$ | $2^{14}$ | $2^{35}$ | $2^5$ | | + | - | - | | |