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

Shihe Ma
Tsinghua University
msh21@mails.tsinghua.edu.cn

Tairong Huang
Tsinghua University
htr19@mails.tsinghua.edu.cn

Anyu Wang
Tsinghua University
anyuwang@tsinghua.edu.cn

Xiaoyun Wang
Tsinghua University
xiaoyunwang@tsinghua.edu.cn

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%. Furthermore, our FDFB algorithms introduce an error that can be as small as 1/15 of that introduced by previous algorithms when evaluating continuous functions. 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.

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 [26, 36, 34] and privacy-preserving machine learning [29, 5, 15, 35], as well as in the construction of cryptographic protocols such as private set intersection [11, 10, 18].

Since Gentry's first construction of an FHE scheme utilizing the bootstrap technique [21], various FHE schemes have been developed [20, 6, 12, 22, 19, 14] and significant improvements have been made [31, 32, 3, 27]. Among these FHE schemes, BGV/FV, CKKS, 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, due to their expensive bootstrap process, these schemes are unsuitable for evaluating non-polynomial functions or deep circuits. On the other hand, FHEW/TFHE utilize an efficient functional bootstrap (or programable 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 [4] and later improved in PEGASUS [35], 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 [16, 28]. 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 [33, 37, 23]. 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 [17]. In addition to focusing on the construction of FDFB, a method for using FDFB to aid in evaluating CKKS ciphertexts is presented in [30]. To handle the evaluation of large-precision LUTs, Guimarães et al. [24] propose tree-based and chaining methods to combine multiple functional bootstraps in TFHE. These two methods in [24] 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. [33] 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**. **FDFB-Compress** improves Type-HalfRange to theoretical optimality, while the other three algorithms improve Type-SelectMSB but are suitable for different scenarios. When evaluating discrete LUTs, **FDFB-BFVMult** shows a speedup of 23.4% $\sim$ 39.2% compared to state-of-the-art results across all parameter settings. We have also extended our FDFB algorithms to enable the evaluation of continuous functions. In this context, the output noise introduced by **FDFB-CancelSign** and **FDFB-Select** is independent of the Lipschitz constant of the evaluated function. Additionally, our algorithms introduce an error as small as 1/15 that of previous algorithms when evaluating Sigmoid : $(-8, 8) \to (-2, 2)$ while maintaining comparable efficiency.

(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 [33], 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 [2] 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$_1$ [16] (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$_2$ [16] (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 [28] (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 [33] (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 does not exceed $\frac{p}{2}$. Then it can evaluate the LUT without being constrained by negacyclicity. We note that the **FullyFBS** of [37] and the **FDFB-C** of [23] are essentially the same as **EvalFunc**.

Comp [17] (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 [8], 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 [33] 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 message can be decomposed into a vector of 4-bit digits. However, this algorithm does not apply to extracting CKKS ciphertexts because it requires a small noise in the input ciphertext.

HomFloorAlt [33] uses three bootstraps to extract the digits of a large-precision message, allowing it to decompose the message into a vector of 5-bit digits and extract CKKS ciphertexts.

1.3 Organization

Section 2 introduces background knowledge on FHEW/TFHE and the homomorphic operations used in this paper. Section 3 and Section 4 present our FDFB and homomorphic decomposition algorithms, respectively. Section 5 gives a comprehensive noise analysis and comparison of our algorithms with previous FDFB and homomorphic decomposition algorithms. Section 6 presents the experimental evaluation of our algorithms, demonstrating their efficiency and effectiveness. Section 7 concludes the paper.

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 χ, 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 σ 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 later 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 specific details are not provided here.

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

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

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 [24, 9], which can be viewed as a specific instance of the public functional key switching method proposed in [13]. This homomorphic operator, denoted as $\mathrm{PackingKS}(c, \{\mathrm{ksk}_{i,j,k}\})$, is detailed in Algorithm 1.

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 $\frac{2N}{q}(m+e)$. 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 TV_{m+e} for $m+e \in [0, N-1]$, and equals $-\mathrm{TV}_{[m+e]_N^+}$ for $m+e \in [N, 2N-1]$,

Algorithm 1: 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 $\mathrm{RLWE}_{\mathbf{s}',n',q'}(\lfloor \frac{q'}{q}(m+e) + \frac{q'}{q_{ks}} e_{ms} \rceil (1 + X + \ldots + X^{d-1}) + \mathbf{e}_{ks})$

$(b, \vec{a}) \leftarrow \mathtt{ModSwitch}(c, q_{ks})$
$\mathrm{ct} \leftarrow (\lfloor \frac{q'}{q_{ks}} b \rceil \cdot (1 + X + \ldots + X^{d-1}), 0)$
for $i \leftarrow 0$ **to** $n-1$ **do**
  $\vec{a}_i = \sum_{k=0}^{\lceil \log_B q_{ks} \rceil - 1} a_{i,k} \cdot B^k, a_{i,k} \in [-\lfloor \frac{B}{2} \rfloor, \lfloor \frac{B-1}{2} \rfloor]$
  for $k \leftarrow 0$ **to** $\lceil \log_B q_{ks} \rceil - 1$ **do**
    $\mathrm{ct} \leftarrow \mathrm{ct} + \mathrm{ksk}_{i,a_{i,k},k}$
return ct

$\mathrm{LWE}_{\vec{s},n,q}(m+e)$ where $m = \frac{q}{p} m'$ $\xleftarrow{\{\mathrm{brk}_i^+\}}$ BlindRotate (1)
$\mathrm{RLWE}_{\mathbf{s}',N,Q}(\mathrm{TV} \cdot X^{-(m+e)} + \mathbf{e}_{acc})$ $\xleftarrow{}$ SampleExtract (2)
$\mathrm{LWE}_{\vec{s}',N,Q}(\frac{Q}{p} F(m') + e_{acc})$ $\xleftarrow{}$ ModSwitch (3)
$\mathrm{LWE}_{\vec{s}',N,q_{ks}}(\frac{q_{ks}}{p} F(m') + \frac{q_{ks}}{Q} e_{acc} + e_{ms})$ $\xleftarrow{\{\mathrm{ksk}_{i,j,k}\}}$ KeySwitch (4)
$\mathrm{LWE}_{\vec{s},n,q_{ks}}(\frac{q_{ks}}{p} F(m') + \frac{q_{ks}}{Q} e_{acc} + e_{ms} + e_{ks})$ $\xleftarrow{}$ ModSwitch (5)
$\mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p} F(m') + \frac{q}{Q} e_{acc} + \frac{q}{q_{ks}}(e_{ms} + e_{ks}) + e'_{ms})$

Figure 1: The five steps of FHEW/TFHE bootstrapping: (1) blind rotation of TV by input ciphertext; (2) extracting the constant term of 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 $\mathbb{Z}_p$.

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 (or negacyclic function) $F : \mathbb{Z}_p \to \mathbb{Z}_p$ 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 1. 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). It should be noted that 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 [7]. 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. TV_0 is first multiplied by $X^{-(m+e)}$ using blind rotation, and the resulting RLWE ciphertext is multiplied by TV_1, which also multiplies the output noise variance by $|\mathrm{TV}_1|_2^2 \le p(p-1)^2$.

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 $i = 0, 1$ and re-linearization key $\mathrm{RLWE}'_{\mathbf{s},N,Q}(\mathbf{s}^2)$, define

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

where $\mathbf{e}_{mult}$ is the noise of BFV multiplication.

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. The values of these variances are well-known in the literature and are listed in the following lemma.

Lemma 2.1. *Let σ^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* $\sigma^2_{ms}(n) = \frac{n}{18} + \frac{1}{12}$,
$\sigma^2_{ks}(n, q_{ks}, B_{ks}) = d_{ks}(1 - \frac{1}{B_{ks}}) n(\sigma^2 + \frac{1}{4})$,
$\sigma^2_{acc}(n, N, Q, B_g) = \frac{2 d_g B_g^2 n N \sigma^2}{3}$,
$\sigma^2_{boot}(n, N, Q, q, B_g, q_{ks}, B_{ks}) = (\frac{q}{q_{ks}})^2 (\sigma^2_{ms}(N) + \sigma^2_{ks}(N, q_{ks}, B_{ks})) + (\frac{q}{Q})^2 \sigma^2_{acc}(n, N, Q, B_g) + \sigma^2_{ms}(n)$.

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 1).

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 $\Pr[|x| > \mathrm{bnd} \cdot \sigma] < 2^{-32}$, where $\mathrm{bnd} = \sqrt{2} \cdot \mathrm{erfc}^{-1}(2^{-32}) \approx 6.338$. We denote the bound of bootstrapping error as $\beta = \mathrm{bnd} \cdot \sigma_{boot}$.

3 IMPROVED FDFB ALGORITHMS

This section introduces the four new FDFB algorithms and demonstrates how to evaluate a continuous function with them when the input ciphertext is approximate (e.g., extracted from a CKKS ciphertext).

This section assumes that the plaintext modulus p is a power of 2 for better presentation. However, it's important to note that changing p to any even number will not affect the correctness of the algorithms presented. As assumed, the ciphertext modulus $q = 2N$ is a power of 2. In addition, we 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 $e + \frac{q}{2p} \in [0, \frac{q}{p} - 1]$. 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.

3.1 FDFB Algorithms

3.1.1 ***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 $[0, \frac{q}{2}]$ 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 2β to prevent the bootstrapping noise from interfering with the compressed message.

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 Alg. 2, and its correctness is proved in Lemma 3.1.

Algorithm 2: FDFB-Compress

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

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

Lemma 3.1. *If* $\beta < \frac{q}{4p}$ *and* $|e| < \frac{q}{2p}$, *then* ***FDFB-Compress***$(F, LWE_{\vec{s},n,q}(\frac{q}{p}m' + e)) = LWE_{\vec{s},n,q}(\frac{q}{p}F(m') + e_{boot})$.

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. □

3.1.2 ***FDFB-CancelSign.*** This algorithm employs the Type-SelectMSB strategy. Given $\mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}m' + e)$, **FDFB-CancelSign** first executes 'mod up' to obtain a ciphertext $\mathrm{LWE}_{\vec{s},n,q}(\frac{q}{2}\mathrm{MSB} + \frac{q}{2p}m' + e)$ 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 Alg. 3, and its correctness is proved in Lemma 3.2.

Algorithm 3: 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 $\mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}F(m') + e')$

1 ct $\leftarrow$ `ModUp`$((b + \frac{q}{4p}, \vec{a}), q)$
2 $\mathrm{ct}_1 \leftarrow$ `BootRaw`$[f_{cs}]$(ct)
3 $\mathrm{ct}_{pk} \leftarrow$ `PackingKS`$(\mathrm{ct}_1, \{\mathrm{ksk}'_{i,j,k}\})$
4 **return** `Boot`$[\mathrm{ct}_{pk}]$(ct)

Lemma 3.2. *If* $|e| < \frac{q}{4p}$ *and* $|e'| < \frac{q}{4p}$, *then* ***FDFB-CancelSign***$(F, LWE_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}m' + e)) = LWE_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}F(m') + e')$.

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 lemma in two steps. (1) $\mathrm{ct}_1 = \mathrm{LWE}((-1)^{\mathrm{MSB}}\lfloor \frac{Q}{p}F(m') \rceil)$; (2) line 3 to 4 multiplies the message in 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 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 ct_{pk} by ct, the constant term of ct_{pk} equals $(-1)^{\mathrm{MSB}}(m_1 + e_{pk}) + e_{acc} \approx (-1)^{\mathrm{MSB}}m_1$. □

3.1.3 ***FDFB-Select.*** This algorithm employs the Type-SelectMSB strategy but does not perform the 'mod up' 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 MSB = 0 or 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. 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 Alg. 4, and its correctness is proved in Lemma 3.3.

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 the case where multi-value bootstrap is unavailable, we develop a variant of **FDFB-Select**, called **FDFB-SelectAlt**, described in Alg. 5, which uses only three bootstraps. The correctness of **FDFB-SelectAlt** is proved in Lemma 3.4.

Remark. We actually use an improved version of the base-aware LWE-to-RLWE packing proposed by [24] to pack ct_{pos} and $-\mathrm{ct}_{neg}$ into ct_{pk}. To pack $M|N$ messages $\mathrm{LWE}(m_i)$ into $\mathrm{RLWE}(\sum_{i=0}^{M-1} m_i(1 + X + X^2 + \ldots + X^{\frac{N}{M}-1})X^{\frac{N}{M}i})$, [24] 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 [24].

Lemma 3.3. *If* $|e| < \frac{q}{2p}$, $\beta < \frac{q}{8}$ *and* $|e'| < \frac{q}{2p}$, *then* ***FDFB-Select***$(F, LWE_{\vec{s},n,q}(\frac{q}{p}m' + e)) = LWE_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$.

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 ct_{pos} when MSB = 0 and in ct_{neg} when MSB = 1. Then it only remains to prove that line 4 to line 6 selects the correct ciphertext from ct_{pos} and 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 ct_{sgn} belongs to $[0, \frac{N}{2} - 1]$ or $[-\frac{N}{2}, -1]$ when MSB = 0 or 1 respectively. Denote the values encrypted in ct_{pos} and ct_{neg} as m_{pos} and m_{neg}. Then 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 ct_{sgn}, ct_{pk} has a constant term of $m_{pos} + e_{pk} + e_{acc}$ for MSB = 0 and $m_{neg} - e_{pk} + e_{acc}$ for MSB = 1. □

Algorithm 4: 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 $\mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$

1 $\mathrm{ct} \leftarrow (b + \frac{q}{2p}, \vec{a})$
2 $\mathrm{ct}_{pos} \leftarrow \mathtt{BootRaw}[f_{pos}](\mathrm{ct})$
3 $\mathrm{ct}_{neg} \leftarrow \mathtt{BootRaw}[f_{neg}](\mathrm{ct})$
4 $\mathrm{ct}_{sgn} \leftarrow \mathtt{Boot}[f_{sgn}](\mathrm{ct})$
5 $\mathrm{ct}_{pk} \leftarrow \mathtt{PackingKS}(\mathrm{ct}_{pos}, \{\mathrm{ksk}'_{i,j,k}\}) +$
  $\mathtt{PackingKS}(-\mathrm{ct}_{neg}, \{\mathrm{ksk}'_{i,j,k}\}) \cdot X^{\frac{N}{2}}$
6 **return** $\mathtt{Boot}[\mathrm{ct}_{sgn}](\mathrm{ct}_{pk})$

Algorithm 5: 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 : $\{\mathrm{ksk}'_{i,j,k}\}$, packing keys for PackingKS with $d = N$
input : An LWE ciphertext $(b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}m' + e)$
output: An LWE ciphertext $\mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$

1 $\mathrm{ct} \leftarrow (b + \frac{q}{2p}, \vec{a})$
2 $\mathrm{ct}_{hdiff} \leftarrow \mathtt{BootRaw}[(f_{neg} - f_{pos})/2](\mathrm{ct})$
3 $\mathrm{ct}_{hsum} \leftarrow \mathtt{BootRaw}[(f_{neg} + f_{pos})/2](\mathrm{ct})$
4 $\mathrm{ct}_{pk} \leftarrow \mathtt{PackingKS}(\mathrm{ct}_{hdiff}, \{\mathrm{ksk}'_{i,j,k}\})$
5 **return** $\mathrm{ct}_{hsum} - \mathtt{Boot}[\mathrm{ct}](\mathrm{ct}_{pk})$

Lemma 3.4. *If* $|e| < \frac{q}{2p}$ *and* $e' < \frac{q}{2p}$, *then* ***FDFB-SelectAlt***$(F, LWE_{\vec{s},n,q}(\frac{q}{p}m' + e)) = LWE_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$.

Proof. $|e| < \frac{q}{2p}$ and $|e'| < \frac{q}{2p}$ ensures the validity of input and output ciphertext of **FDFB-SelectAlt**. ct_{hdiff} and 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 Lemma 3.2, $\mathrm{Boot}[\mathrm{ct}](\mathrm{ct}_{pk}) = \mathrm{LWE}((-1)^{\mathrm{MSB}} m_{hdiff})$. 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 MSB = 0 and $f_{neg}(m)$ when 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 MSB = 0, 1 respectively. □

3.1.4 ***FDFB-BFVMult***. This algorithm employs the Type-SelectMSB strategy but uses BFV multiplication to handle the MSB. It contains **FDFB-BFVMult**$_1$ and **FDFB-BFVMult**$_2$.

FDFB-BFVMult$_1$ first obtains a ciphertext that encrypts $(-1)^{\mathrm{MSB}} \lfloor \frac{Q}{p} F(m') \rceil$ in the same way as **FDFB-CancelSign**. Then it evaluates the function (7) via a functional bootstrap to acquire

the encryption of $\lfloor \frac{Q}{p}(-1)^{\mathrm{MSB}} \rceil$. Finally, it computes the product of the two LWE ciphertexts using LWE-to-RLWE packing and BFV multiplication. The algorithm is fully described in Alg. 6, and its correctness is proved in Lemma 3.5.

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

FDFB-BFVMult$_2$ 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 Alg. 7, and its correctness is proved in Lemma 3.6.

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

When parameter settings allow for multi-value bootstrap, the number of bootstraps in both **FDFB-BFVMult**$_1$ and **FDFB-BFVMult**$_2$ can be decreased to one using multi-value bootstrap.

The improved efficiency of **FDFB-BFVMult** is mainly due to our fine-grained noise analysis for BFV multiplication. 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. In contrast to conventional BFV multiplication, where messages are stored in all coefficients, this results in approximately $1/N$ noise growth compared to **FDFB-BFVMult**. Refer to Appendix B for a complete analysis.

Algorithm 6: FDFB-BFVMult$_1$

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 $(b, \vec{a}) = \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}m' + e)$
output: An LWE ciphertext $\mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}F(m') + e')$

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

Lemma 3.5. *If* $|e| < \frac{q}{4p}$ *and* $e' < \frac{q}{4p}$, *then* ***FDFB-BFVMult***$_1(F, LWE_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}m' + e)) = LWE_{\vec{s},n,\frac{q}{2}}(\frac{q}{2p}F(m') + e')$.

Proof. $|e| < \frac{q}{4p}$ and $|e'| < \frac{q}{4p}$ ensures the validity of input and output ciphertext of **FDFB-BFVMult**$_1$. Similar to the proof of Lemma 3.2, $\mathrm{ct}_0 = \mathrm{RLWE}((-1)^{\mathrm{MSB}} \lfloor \frac{Q}{p} F(m') \rceil)$. 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 ct_{prod} into the desired output format. □

Algorithm 7: FDFB-BFVMult$_2$

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 $\mathrm{LWE}_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$

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

Lemma 3.6. *If* $|e| < \frac{q}{2p}$ *and* $e' < \frac{q}{2p}$, *then* ***FDFB-BFVMult***$_2(F, LWE_{\vec{s},n,q}(\frac{q}{p}m' + e)) = LWE_{\vec{s},n,q}(\frac{q}{p}F(m') + e')$.

Proof. $|e| < \frac{q}{2p}$ and $|e'| < \frac{q}{2p}$ ensures the validity of input and output ciphertext of **FDFB-BFVMult**$_2$. Denote the plaintext encrypted in ct_{diff} and 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 **FDFB-BFVMult**$_1$. □

3.2 FDFBs for Continuous Functions

This subsection describes how to evaluate a continuous function $F' : \mathbb{R} \to \mathbb{R}$ on an approximate LWE ciphertext (e.g., extracted from a CKKS ciphertext) using the modified FDFB algorithms **FDFB-Compress**, **FDFB-CancelSign** and **FDFB-Select**. **EvalFunc** and **Comp** in prior works can also be adapted for continuous purposes, whose details we leave to Appendix A. As a useful optimization, when F' is a vertically translated odd or even function (i.e., can be expressed as the sum of a constant and an odd or even function), the number of bootstraps in **Comp** can be reduced from 4 to 2. This alternative version of **Comp** is referred to as **Comp** (fast).

Evaluating a continuous function is inherently incompatible with multi-value bootstrap because the multiplication of TV_1 greatly amplifies the variance of blind rotation noise. While it is possible to select an intermediate modulus p_{mid} and discretize the continuous function F' as a $\mathbb{Z}_q \to \mathbb{Z}_{p_{mid}}$ mapping, the 2-norm of $\mathrm{TV}_1 = \mathrm{TV}'(1-X) \in R_{2p_{mid}}$ can still be large, leading to a considerable output noise. The same problem also exists for other polynomial multiplication operations (such as BFV multiplication and $R_q \times \mathrm{RLWE}'$ product), making FDFB algorithms that use these operations unsuitable for continuous function evaluation.

For continuous function evaluation, the input scaling factor Δ_{in}, output scaling factor Δ_{out} and output modulus q_{out} are introduced as new parameters. Refer to Section 5 and Appendix A for proof of the following algorithms' output noise variances.

3.2.1 ***FDFB-Compress*** *(Continuous Case).* In this algorithm, the compression function f_C in (1) is substituted with f'_C, which is defined in (9) and illustrated in Figure 2.

$$f'_C = \begin{cases} \lfloor \frac{\frac{q}{4}-2\beta}{\frac{q}{2}-1}x + \beta \rceil & x \in [0, \frac{q}{2}-1] \\ q - f'_C(x - \frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_q \tag{9}$$

The strategy adopted to construct f'_C is called 'β-padding', which creates a 2β 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 β away from $\frac{q}{4}$ and q respectively to ensure that the output message of f'_C always lies within half of $\mathbb{Z}_q$.

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

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

where $\mathrm{slope} = \frac{\frac{q}{4}-2\beta}{\frac{q}{2}-1}$. Essentially f'_{eval} aims to recover the original input to f'_C, carry out an evaluation of F' on the recovered input, and subsequently scales the result by Δ_{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'.

3.2.2 ***FDFB-CancelSign*** *(Continuous Case).* To extend **FDFB-CancelSign** to continuous functions, we replace f_{cs} with f'_{cs} defined in (10), which interprets each $x \in [0, \frac{q}{2}-1]$ as a signed fixed-point integer.

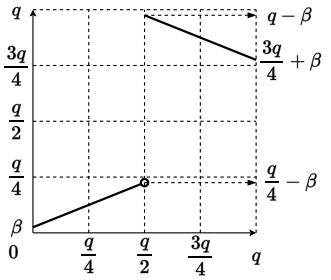


Figure 2: Compression function f'_C for continuous function evaluation.

$$f'_{cs} = \begin{cases} \lfloor F'(\frac{[x]_{\frac{q}{2}}}{\Delta_{in}})\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [0, \frac{q}{2}-1] \\ Q - f'_{cs}(x - \frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_Q \tag{10}$$

The output error of **FDFB-CancelSign** is determined by the bootstrap error and irrelevant to the Lipschitz constant of F'.

3.2.3 ***FDFB-Select*** *(Continuous Case).* The adjustments made to **FDFB-Select** and **FDFB-SelectAlt** are almost identical to those applied to **FDFB-CancelSign**, and the output error remains independent of the Lipschitz constant of F'.

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 (up to a bootstrapping error) 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.

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 (11), with different input and output ranges.

$$f_{red} = \begin{cases} \frac{q}{4} & x \in [0, \frac{q}{2}-1] \\ q_0 - \frac{q}{4} & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_{q_0} \tag{11}$$

The complete algorithm is described in Alg. 8 and its correctness is proved in Lemma 4.1. Figure 3 illustrates a comparison of the naive approach (i.e., switch the modulus without reserving any room for modulus switching noise), **HomFloor** of [33] and our **HomDecomp-Reduce**.

Lemma 4.1. *If* $bnd\sqrt{B^{-2}\sigma^2_{boot} + \sigma^2_{ms}} < \frac{q}{4B}$, ***HomDecomp-Reduce*** *outputs the decomposed digits correctly.*

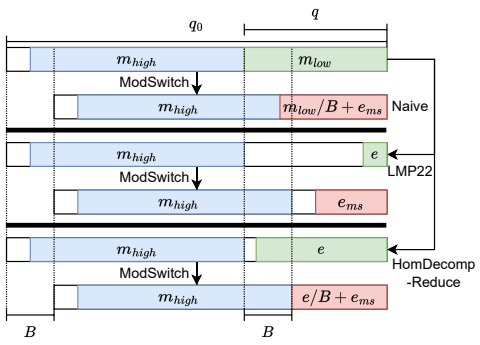


Figure 3: Comparison of the naive approach, HomFloor and our HomDecomp-Reduce. The blue parts stand for higher bits, while the green and red parts stand for lower bits before and after modulus switching.

Algorithm 8: HomDecomp-Reduce

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'

1 $i \leftarrow 0$
2 **while** $q_0 > q$ **do**
3 $\quad \mathrm{ct}_i \leftarrow \mathtt{ModDown}(\mathrm{ct}, q)$
4 $\quad \mathrm{ct} \leftarrow \mathrm{ct} + (\frac{q_0}{2p}, \vec{0})$
5 $\quad \mathrm{ct}' \leftarrow \mathtt{ModDown}(\mathrm{ct}, q)$
6 $\quad \mathrm{ct} \leftarrow \mathrm{ct} + \mathtt{Boot}[f_{red}](\mathrm{ct}') - (\frac{q}{2}, \vec{0})$
7 $\quad \mathrm{ct} \leftarrow \mathtt{ModSwitch}(\mathrm{ct}, \frac{q_0}{B})$
8 $\quad i \leftarrow i + 1$
9 $\mathrm{ct}_i \leftarrow \mathrm{ct}$
10 **return** $\{\mathrm{ct}_i\}$

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. □

4.2 HomDecomp-FDFB

In **HomDecomp-FDFB**, we use **FDFB-Compress** to evaluate the continuous identity function $f_{id}(x) = x : \mathbb{Z}_q \to \mathbb{Z}_{q_0}$ (where the input x is in the positive form), and the obtained result is used to approximately clear the lower bits in input ciphertext. See Alg. 9 for a full description of **HomDecomp-FDFB** and Lemma 4.2 for proof of its correctness. The scaling factors for both the input and output (Δ_{in} and Δ_{out}) of the **FDFB-Compress** function are set to 1, while q_{out} is set to q_0.

Algorithm 9: 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'

1 $i \leftarrow 0$
2 **while** $q_0 > q$ **do**
3 $\quad \mathrm{ct}_i \leftarrow \mathtt{ModDown}(\mathrm{ct}, q)$
4 $\quad \mathrm{ct} \leftarrow \mathrm{ct} + (\frac{q_0}{2p}, \vec{0})$
5 $\quad \mathrm{ct}' \leftarrow \mathtt{ModDown}(\mathrm{ct}, q)$
6 $\quad \mathrm{ct} \leftarrow \mathrm{ct} - \mathtt{FDFB\text{-}Compress}[f_{id}](\mathrm{ct}')$
7 $\quad \mathrm{ct} \leftarrow \mathtt{ModSwitch}(\mathrm{ct}, \frac{q_0}{B})$
8 $\quad i \leftarrow i + 1$
9 $\mathrm{ct}_i \leftarrow \mathrm{ct}$
10 **return** $\{\mathrm{ct}_i\}$

Lemma 4.2. *Let e_f be the output error of **FDFB-Compress**, and σ_f^2 be its variance (Refer to Section 5 for noise analysis of **FDFB-Compress**). If bnd$\sqrt{B^{-2}\sigma_f^2 + \sigma_{ms}^2} < \frac{q}{2B}$, **HomDecomp-FDFB** outputs the decomposed digits correctly.*

Proof. 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 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. □

5 ANALYSIS AND COMPARISON

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

The analysis shows that our noise reduction techniques, such as refined noise analysis for BFV multiplication and replacing BFV multiplications with functional bootstraps, significantly reduce the output error variance of the FDFB algorithms. As a result, we can use more compact parameters, especially larger B_g in bootstraps, which significantly improves the efficiency.

5.1 Analysis of FDFB Algorithms

Table 1 presents the (core) output variance σ_{core}^2 and the number of required bootstraps for all FDFB algorithms, where the final output variance is $\sigma_{out}^2 = \sigma_{core}^2 + \sigma_{com}^2$ and σ_{com}^2 is defined in Section 2. Note that the efficiency of an FDFB algorithm is not solely determined by the number of bootstraps it requires. The output and the intermediate noises also impact the compactness of parameters, and thus determine the final efficiency. In particular, the output error with variance σ_{out}^2 should be bounded by $\frac{q}{2p}$ (or $\frac{q}{4p}$

if the output ciphertext modulus is $\frac{q}{2}$) to ensure that the ciphertext can be correctly decrypted. Intermediate errors (if any) should also be bounded, as listed below Table 1.

The main advantage of our FDFB algorithms is the reduced output noise, which allows more compact and efficient parameters. For instance, in Table 1, we can observe that **FDFB-BFVMult**$_1$ and **FDFB-BFVMult**$_2$ use significantly tighter noise analysis for BFV multiplication than **WoP-PBS**, reducing the noise growth to $1/N$ the original value. Moreover, while performing the MSB-related selection in Type-SelectMSB, **FDFB-CancelSign** and **FDFB-Select** (**FDFB-SelectAlt**) use LWE-to-RLWE packing and blind rotation instead of BFV multiplication, thereby reducing the noise to $1/N^2p^2$ that of **WoP-PBS**$_1$ and **WoP-PBS**$_2$. Although **FDFB-CancelSign** and **FDFB-Select** (**FDFB-SelectAlt**) require an additional bootstrap to replace the BFV multiplication, we demonstrate in Section 6 that they are still faster than **WoP-PBS**$_1$ and **WoP-PBS**$_2$ due to their slow noise growth. Additionally, **FDFB-Compress** reduces the input noise variance to the second bootstrap by half, resulting in 0.5 more bits of plaintext modulus compared to **EvalFunc**. Notably, among Type-HalfRange FDFB algorithms, **FDFB-Compress** is optimal. This observation is derived from the fact that all methods in this category must obtain an encryption of $\frac{q}{2p}m'$ under modulus q using functional bootstraps. This encryption is valid only when $\beta < \frac{q}{4p}$, which is also the only requirement for **FDFB-Compress**.

Besides the efficiency in discrete LUT evaluation, the reduction in output noise offered by our FDFB algorithms also enables high-precision continuous LUT evaluation. The output noise of previous FDFB algorithms is either related to the Lipschitz constant of the evaluated function (**EvalFunc**, **Comp**) or significantly amplified by polynomial multiplication (**FDFB-KS**, **WoP-PBS**). In contrast, our **FDFB-CancelSign**, **FDFB-Select** and **FDFB-SelectAlt** avoid both defects, thus achieving the smallest output noise. In terms of **FDFB-Compress**, it has roughly the same output error as **EvalFunc** because $k_1 \approx \frac{1}{2}$ and the Δ_{in} of **FDFB-Compress** is twice that of **EvalFunc**. However, the input ciphertext modulus of **FDFB-Compress** is also twice that of **EvalFunc**, thus making **FDFB-Compress** less affected by input ciphertext noise (e.g., modulus switching noise).

5.2 Analysis of Homomorphic Decomposition

Table 2 presents the noise growth and maximum plaintext modulus of decomposed digits for all homomorphic decomposition algorithms. For the parameters used in [33], the maximum decomposition base B is 2^4 for **HomFloor** and **HomDecomp-Reduce** and 2^5 for **HomFloorAlt** and **HomDecomp-FDFB**. Our proposed homomorphic decomposition algorithms **HomDecomp-Reduce** and **HomDecomp-FDFB** can achieve the same effect as [33] for concrete parameters, but with one less bootstrap, resulting in at least 2.0x and 1.5x running speed compared to previous algorithms **HomFloor** and **HomFloorAlt** respectively.

6 IMPLEMENTATION

We implement all the FDFB algorithms and homomorphic decomposition algorithms, including both ours and previous ones, in OpenFHE [2] (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 [33]). 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.

We use three parameter sets in our LWE schemes, i.e., $\text{PARAM}_{\text{decomp}}$, $\text{PARAM}_{\text{fast}}$ and $\text{PARAM}_{\text{continuous}}$, which have been verified to meet 128-bit security using lattice-estimator [1] (commit id `48fa49b`). Table 3 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 [33]. For $\text{PARAM}_{\text{fast}}$, we focus on FDFB algorithms for discrete 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 $\text{PARAM}_{\text{fast}}$. Finally, for $\text{PARAM}_{\text{continuous}}$, we focus on FDFB algorithms for continuous LUTs. q_{ks} must be large enough to make the impact of key switching noise on the output error negligible. Therefore, we set a larger $q_{ks} = 2^{25}$ for $\text{PARAM}_{\text{continuous}}$ than for $\text{PARAM}_{\text{fast}}$.

The performance of discrete LUT evaluation with FDFB variants is tested with the plaintext modulus set to 2^4 and 2^5. The input ciphertext is error-free for continuous function evaluation because we only measure the noise introduced by the FDFB algorithms.

Figure 4 displays the benchmark results for our algorithms, where the red lines or dots represent the performance of our algorithms and the blue ones correspond to previous algorithms. 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). Refer to Appendix C for a complete list of the parameters used in the benchmark.

Performance of Discrete LUT Evaluation. Figure 4a to 4d and Table 4 shows that our FDFB variants reduce the running time of their predecessors up to 40%, and the best of our FDFB variants outperforms the best known results at least by 23.4% and up to 39.2%.

FDFB-BFVMult$_2^*$ runs the fastest (or almost the fastest) under all four scenarios since it needs only one bootstrap using $B_g = 2^{18}$. However, it cannot be applied to continuous function evaluation because the multi-value bootstrap and BFV multiplication used by it will drastically increase the output noise, thus decreasing the output precision.

For all the scenarios but 4a, **FDFB-Select** outperforms its predecessor **WoP-PBS**$_2$. Although it takes one more bootstrap than **WoP-PBS**$_2$, the small noise growth achieved by replacing BFV multiplication with blind rotation allows it to use more efficient parameters. The same reason explains why **FDFB-CancelSign** outperforms **WoP-PBS**$_1$ in 4a and 4c even though it needs one more bootstrap. **FDFB-Compress** and **EvalFunc** have the same computational cost (as shown in 4c), but since **FDFB-Compress** has a plaintext space 0.5 bits larger than the latter, it is available in scenario 4a while **EvalFunc** is not.

Table 1: 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 σ^2_{boot} as in [33]. The final output variance $\sigma^2_{out} = \sigma^2_{core} + \sigma^2_{com}$. 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 assume the input ciphertext is error-free for continuous function evaluation because we are only interested in the noise introduced by FDFB. σ'^2_{out} is measured on the outputted real-valued message. L is the Lipschitz constant of the evaluated function.

FDFB Variant	Core Output Variance σ^2_{core}	Num of BTS	Output Variance σ'^2_{out} (Continuous Case)
EvalFunc [33][1]	$(\frac{q}{Q})^2\sigma^2_{acc}$, intermediate: $2\sigma^2_{boot}$	2	$(\frac{L}{\Delta_{in}})^2\sigma^2_{boot} + \Delta^{-2}_{out}\sigma^2_{boot}$
FDFB-KS [28][2]	$(\frac{q}{Q})^2(\sigma^2_{acc} + d_{g1}\frac{B^2_{g1}}{4}N(\sigma^2_{acc} + (\frac{Q}{q_{pk}})^2\sigma^2_{ms} + \sigma^2_{ks}))$	$d_{g1}+1$	-
Comp [17][3]	$2(\frac{q}{Q})^2\sigma^2_{acc}$, intermediate[6]: $(\frac{q}{Q})^2\sigma^2_{acc}$	4	$(\frac{L}{k_1\Delta_{in}})^2\sigma^2_{boot} + \Delta^{-2}_{out}(2(\frac{q_{out}}{Q})^2\sigma^2_{acc} + \sigma^2_{com})$
Comp [17]*[4]	$2(\frac{q}{Q})^2\sigma^2_{acc}$, intermediate[6]: $(\frac{q}{Q})^2(2p^2-4)\sigma^2_{acc}$	3	-
WoP-PBS$_1$ [16]§†	$(\frac{q}{Q})^2\frac{N^2}{9}p^2\sigma^2_{acc}$	2	-
WoP-PBS$_1$ [16]§†*	$(\frac{q}{Q})^2\frac{N^2}{18}p^3(p-1)^2\sigma^2_{acc}$	1	-
WoP-PBS$_2$ [16]†	$(\frac{q}{Q})^2\frac{2N^2}{9}p^2\sigma^2_{acc}$	3	-
WoP-PBS$_2$ [16]†*	$(\frac{q}{Q})^2\frac{2N^2}{9}p^3(p-1)^2\sigma^2_{acc}$	1	-
FDFB-BFVMult$_1$ §†	$(\frac{q}{Q})^2\frac{N}{9}p^2\sigma^2_{acc}$	2	-
FDFB-BFVMult$_1$ §†*	$(\frac{q}{Q})^2\frac{N}{18}p^3(p-1)^2\sigma^2_{acc}$	1	-
FDFB-BFVMult$_2$ †	$(\frac{q}{Q})^2\frac{N}{9}p^2\sigma^2_{acc}$	3	-
FDFB-BFVMult$_2$ †*	$(\frac{q}{Q})^2\frac{2N}{9}p^3(p-1)^2\sigma^2_{acc}$	1	-
FDFB-Compress §[5]	$(\frac{q}{Q})^2\sigma^2_{acc}$, intermediate: σ^2_{boot}	2	$(\frac{L}{k_2\Delta_{in}})^2\sigma^2_{boot} + \Delta^{-2}_{out}\sigma^2_{boot}$
FDFB-CancelSign §	$(\frac{q}{Q})^2(2\sigma^2_{acc} + \sigma^2_{ks}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms}$	2	$\frac{\sigma^2_{core}+\sigma^2_{com}}{\Delta^2_{out}}$, replace q in σ^2_{core} with q_{out}
FDFB-Select	$(\frac{q}{Q})^2(2\sigma^2_{acc} + 2\sigma^2_{ks}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms}$	4	$\frac{\sigma^2_{core}+\sigma^2_{com}}{\Delta^2_{out}}$, replace q in σ^2_{core} with q_{out}
FDFB-Select *	$(\frac{q}{Q})^2(2p(p-1)^2\sigma^2_{acc} + 2\sigma^2_{ks}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms}$	2	-
FDFB-SelectAlt	$(\frac{q}{Q})^2(3\sigma^2_{acc} + \sigma^2_{ks}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms}$	3	$\frac{\sigma^2_{core}+\sigma^2_{com}}{\Delta^2_{out}}$, replace q in σ^2_{core} with q_{out}
FDFB-SelectAlt *	$(\frac{q}{Q})^2((6p(p-1)^2+1)\sigma^2_{acc} + \sigma^2_{ks}) + (\frac{q}{q_{pk}})^2\sigma^2_{ms}$	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 Δ_{in} are halved compared to the cases where the additional MSB is not required.

† FDFB algorithms that use BFV multiplication. σ^2_{core} 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 $2\sigma^2_{boot} \le \frac{q}{4p}$ 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] $k_1 = \frac{N-2\beta}{N-1}$. A continuous function that is vertically translated can be computed using two bootstraps.

[4] For **Comp** *, the two TV_1's used in multi-value bootstrap are constant polynomials independent of the LUT. $|\mathrm{TV}_1|_2^2 \le 2p^2 - 4$.

[5] $k_2 = \frac{\frac{N}{2}-2\beta}{N-1}$. The intermediate error has a variance of σ^2_{boot} and needs to be bounded by $\frac{q}{4p}$ for correctness.

[6] In **Comp**, it is required that $\mathrm{bnd} \cdot \sqrt{\sigma^2_{inter} + \sigma^2_{com}} \le \frac{q}{2p}$ for correctness, where σ^2_{inter} is the variance of the core intermediate error.

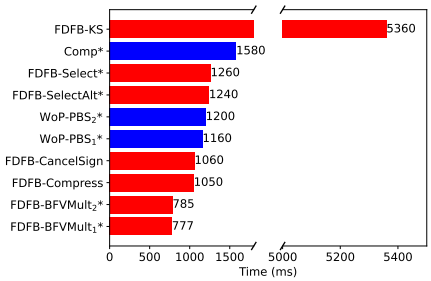
Performance of Continuous LUT Evaluation. Figure 4e illustrates the performance and output standard deviation of different FDFB algorithms applied to continuous function evaluation. The evaluated function is Sigmoid, with its input and output range set to $(-8, 8)$ and $(-2, 2)$. The output standard deviation is measured at input point $x = 0$ because Sigmoid has the largest derivative, thus having the largest output error (if the output error is Lipschitz-dependent).

The graph shows that **FDFB-Select**, **FDFB-SelectAlt** and **FDFB-CancelSign** introduce a noise $1/15 \sim 1/13$ that of previous methods because their output error is independent of the Lipschitz constant of the evaluated function. The noise introduced by **FDFB-Compress** is also smaller than its predecessor **EvalFunc** but still depends on the Lipschitz constant.

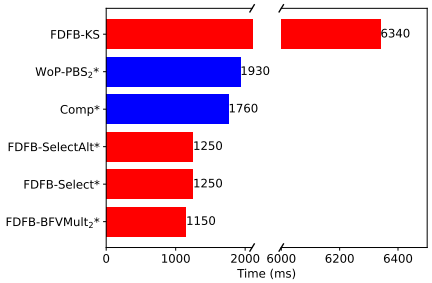
In practice, the input is not error-free and may contain modulus switching noise. As the input noise grows larger compared to the magnitude of the input message, it will dominate the output

Table 2: Maximum decomposition base and the number of bootstraps required for homomorphic decomposition algorithms.

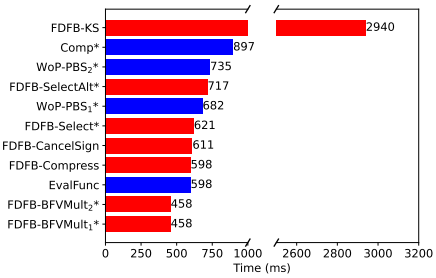
Name of Variant	Requirement for Decomposition Base B	Num of BTS	Other Requirements
HomFloor [33]	$\mathrm{bnd}\sqrt{(1+B^{-2})\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{2B}$	2	Cannot decompose extracted CKKS ciphertexts
HomFloorAlt [33]	$\mathrm{bnd}\sqrt{B^{-2}\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{2B}$	3	$q > 8\sqrt{2}\beta$
HomDecomp-Reduce	$\mathrm{bnd}\sqrt{B^{-2}\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{4B}$	1	-
HomDecomp-FDFB	$\mathrm{bnd}\sqrt{B^{-2}(1+k_2^{-2})\sigma_{boot}^2+\sigma_{ms}^2} < \frac{q}{2B}$	2	-



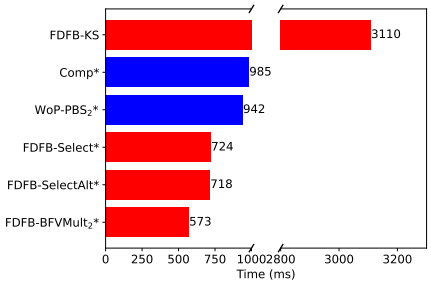
(a) PARAM$_{\mathbf{decomp}}$, FDFB with $p = 2^4$



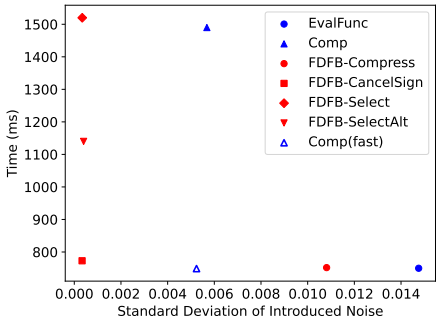
(b) PARAM$_{\mathbf{decomp}}$, FDFB with $p = 2^5$



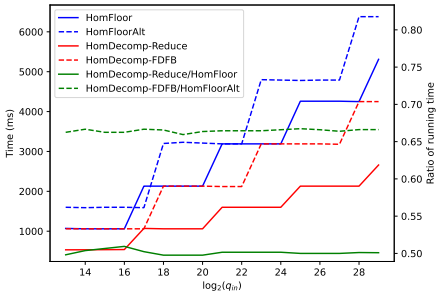
(c) PARAM$_{\mathbf{fast}}$, FDFB with $p = 2^4$



(d) PARAM$_{\mathbf{fast}}$, FDFB with $p = 2^5$



(e) PARAM$_{\mathbf{continuous}}$, $\sigma : (-8, 8) \to (-2, 2)$



(f) PARAM$_{\mathbf{decomp}}$, Decomposition

Figure 4: 4a to 4d show the performance of FDFB variants under PARAM$_{\mathbf{decomp}}$ and PARAM$_{\mathbf{fast}}$, $p = 2^4$ or $p = 2^5$. 4e shows the performance and output standard deviation near $x = 0$ of continuous Sigmoid evaluation. 4f shows the performance of homomorphic decomposition algorithms and the running time ratio between prior methods and ours.

error and weaken the advantage of our methods. Also, **Comp** (fast) only applies to vertically translated odd and even functions and is unavailable otherwise.

Performance of Homomorphic Decomposition. Figure 4f illustrates the performance of different homomorphic decomposition algorithms. Data for $B = 2^4$ is drawn in solid lines, while those for $B = 2^5$ are drawn in dashed lines. The green lines represent the running time ratio between prior decomposition methods and ours. For $B = 2^4$, **HomDecomp-Reduce** runs twice as fast as **HomFloor**, and for $B = 2^5$, **HomDecomp-FDFB** runs 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.

Table 3: Parameter sets for LWE scheme and their use cases.

LWE Param Sets	n	q_{ks}	Use Cases
PARAM$_{\mathrm{decomp}}$	1340	2^{35}	HomDecomp, Discrete FDFB
PARAM$_{\mathrm{fast}}$	760	2^{20}	Discrete FDFB
PARAM$_{\mathrm{continuous}}$	955	2^{25}	Continuous FDFB

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%. For continuous functions evaluation, our FDFB algorithms can achieve a standard deviation of output error as small as 0.008 when evaluating Sigmoid under typical parameter settings, which is 1/13 to 1/15 that

Table 4: Performance improvement of our FDFB variants.

Scenario in Figure 4	4a	4b	4c	4d
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%

of previous results. Our homomorphic decomposition algorithms run 1.5x to 2x as fast as those presented in [33], 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.

REFERENCES

[1] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9, 3, 169–203. DOI: doi:10.1515/jmc-2015-0016.

[2] Ahmad Al Badawi et al. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915. https://eprint.iacr.org/2022/915. (2022). https://eprint.iacr.org/2022/915.

[3] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder VL Pereira, and Nigel P Smart. 2023. Final: faster fhe instantiated with ntru and lwe. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*. Springer, 188–215.

[4] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. 2020. Chimera: combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14, 1, 316–338.

[5] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *Advances in Cryptology – CRYPTO 2018*. Hovav Shacham and Alexandra Boldyreva, (Eds.) Springer International Publishing, Cham, 483–512. ISBN: 978-3-319-96878-0.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory*, 6, 3, Article 13, (July 2014), 36 pages. DOI: 10.1145/2633600.

[7] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In *Topics in Cryptology – CT-RSA 2019*. Mitsuru Matsui, (Ed.) Springer International Publishing, Cham, 106–126. ISBN: 978-3-030-12612-4.

[8] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New techniques for multi-value input homomorphic evaluation and applications. In *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*. Springer, 106–126.

[9] Olive Chakraborty and Martin Zuber. 2022. Efficient and Accurate Homomorphic Comparisons. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (WAHC'22). Association for Computing Machinery, Los Angeles, CA, USA, 35–46. ISBN: 9781450398770. DOI: 10.1145/3560827.3563375.

[10] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (CCS '18). Association for Computing Machinery, Toronto, Canada, 1223–1237. ISBN: 9781450356930. DOI: 10.1145/3243734.3243836.

[11] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast Private Set Intersection from Homomorphic Encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (CCS '17). Association for Computing Machinery, Dallas, Texas, USA, 1243–1255. ISBN: 9781450349468. DOI: 10.1145/3133956.3134061.

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 409–437.

[13] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, 33, 1, 34–91. DOI: 10.1007/s00145-019-09319-x.

[14] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33, 1, 34–91.

[15] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. In *Cyber Security Cryptography and Machine Learning*. Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, (Eds.) Springer International Publishing, Cham, 1–19. ISBN: 978-3-030-78086-9.

[16] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2021. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In *Advances in Cryptology – ASIACRYPT 2021*. Mehdi Tibouchi and Huaxiong Wang, (Eds.) Springer International Publishing, Cham, 670–699. ISBN: 978-3-030-92078-4.

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

[18] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. 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). Association for Computing Machinery, Virtual Event, Republic of Korea, 1135–1150. ISBN: 9781450384544. DOI: 10.1145/3460120.3484760.

[19] Léo Ducas and Daniele Micciancio. 2015. Fhew: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*. Springer, 617–640.

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

[21] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (STOC '09). Association for Computing Machinery, Bethesda, MD, USA, 169–178. ISBN: 9781605585062. DOI: 10.1145/1536414.1536440.

[22] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*. Springer, 75–92.

[23] Antonio Guimarães, Edson Borin, and Diego F. Aranha. 2022. MOSFHET: Optimized Software for FHE over the Torus. Cryptology ePrint Archive, Paper 2022/515. https://eprint.iacr.org/2022/515. (2022). https://eprint.iacr.org/2022/515.

[24] Antonio Guimarães, Edson Borin, and Diego F. Aranha. 2021. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021, 2, (Feb. 2021), 229–253. DOI: 10.46586/tches.v2021.i2.229-253.

[25] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. 2021. Revisiting Homomorphic Encryption Schemes for Finite Fields. In *Advances in Cryptology – ASIACRYPT 2021*. Mehdi Tibouchi and Huaxiong Wang, (Eds.) Springer International Publishing, Cham, 608–639. ISBN: 978-3-030-92078-4.

[26] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic regression model training based on the approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/254. https://eprint.iacr.org/2018/254. (2018).

[27] Kamil Kluczniak. 2022. 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*, 1783–1797.

[28] Kamil Kluczniak and Leonard Schild. 2021. FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption. (2021). eprint: arXiv:2109.02731.

[29] Joon-Woo Lee et al. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access*, 10, 30039–30054. DOI: 10.1109/ACCESS.2022.3159694.

[30] Kang Hoon Lee and Ji Won Yoon. 2023. Discretization Error Reduction for Torus Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2023/402. https://eprint.iacr.org/2023/402. (2023). https://eprint.iacr.org/2023/402.

[31] Feng-Hao Liu and Han Wang. 2023. Batch bootstrapping I: a new framework for simd bootstrapping in polynomial modulus. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*. Springer, 321–352.

[32] Feng-Hao Liu and Han Wang. 2023. Batch bootstrapping II: bootstrapping in polynomial modulus only requires o˜(1) fhe multiplications in amortization. In

Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III. Springer, 353–384.

[33] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. 2022. Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping. In *Advances in Cryptology – ASIACRYPT 2022*. Shweta Agrawal and Dongdai Lin, (Eds.) Springer Nature Switzerland, Cham, 130–160. ISBN: 978-3-031-22966-4.

[34] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2012. 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*, 1219–1234.

[35] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. 2021. PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, 1057–1073. DOI: 10.1109/SP40001.2021.00043.

[36] Sungjoon Park, Minsu Kim, Seokjun Seo, Seungwan Hong, Kyoohyung Han, Keewoo Lee, Jung Hee Cheon, and Sun Kim. 2019. A secure snp panel scheme using homomorphically encrypted k-mers without snp calling on the user side. *BMC genomics*, 20, 163–174.

[37] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. 2021. Tota: fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Paper 2021/1347. https://eprint.iacr.org/2021/1347. (2021). https://eprint.iacr.org/2021/1347.

A FDFB FOR CONTINUOUS FUNCTION EVALUATION

A.1 Algorithms

When porting **FDFB-Compress**, **FDFB-CancelSign**, **FDFB-Select** and **FDFB-SelectAlt** to the case of continuous function evaluation, the applied modifications are well explained in the paper. Here we only describe our changes to **EvalFunc** and **Comp**. They have a similar problem to **FDFB-Compress** and need to use the β-padding technique to avoid outputting completely wrong results. Although [30] has already used these two methods to evaluate continuous functions, the mentioned issue is not taken into account.

A.1.1 ***EvalFunc.*** The input ciphertext is $c = \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(m)$, where $m \in [0, \frac{q}{4}-1-\beta] \bigcup [\frac{q}{4}+\beta, \frac{q}{4}-1]$ by our assumption, which represents a two's complement fixed-point number in $(-(\frac{q}{4}-\beta)\Delta_{in}^{-1}, (\frac{q}{4}-\beta)\Delta_{in}^{-1})$. This assumption that the fixed-point value is β away from its lower and upper bounds is quite reasonable in practice.

First, c is added with $(\frac{q}{4}, \vec{0})$, resulting in $c_1 = \mathrm{LWE}_{\vec{s},n,\frac{q}{2}}(m_1)$, where $m_1 \in [\beta, \frac{q}{2}-1-\beta]$. The range of m_1 corresponds linearly to the range of the input fixed-point value. Then c_1 is raised to modulus q, giving $c_2 = \mathrm{LWE}_{\vec{s},n,q}(m_1 + \frac{q}{2}\mathrm{MSB})$. The first bootstrap computes an encryption of $\frac{q}{2}\mathrm{MSB} + e_{boot}$. After subtracting it from c_2, the result is $c_3 = \mathrm{LWE}_{\vec{s},n,q}(m_1 - e_{boot})$. Since $|e_{boot}| = \beta$, $m_1 - e_{boot}$ lies within $[0, \frac{q}{2}-1]$. Finally, we can compute $\mathrm{BootRaw}[f_a](c_3)$ to obtain the desired result.

$$f_a(x) = \begin{cases} \lfloor F'([x-\frac{q}{4}]_{\frac{q}{2}}\Delta_{in}^{-1})\frac{Q\Delta_{out}}{q_{out}} \rceil & x \in [0, \frac{q}{2}-1] \\ Q - f_a(x-\frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases} : \mathbb{Z}_q \to \mathbb{Z}_Q$$

A.1.2 ***Comp.*** Before using **Comp** to evaluate continuous functions, we introduce the concept of pseudo-odd and pseudo-even LUTs (defined in [17]).

For an LUT $f : \mathbb{Z}_q \to \mathbb{Z}_Q$, we call it as pseudo-odd if $f(i) = -f(q-1-i)$ and call it as pseudo-even if $f(i) = f(q-1-i)$. Informally speaking, a pseudo-odd LUT is anti-symmetric about $\frac{q-1}{2}$ while a pseudo-even LUT is symmetric about $\frac{q-1}{2}$. An arbitrary LUT f can be expressed as the sum of a pseudo-odd LUT f_{pso} and a pseudo-even LUT f_{pse}. When evaluating a continuous function $F' : \mathbb{R} \to \mathbb{R}$, we can also treat $f(x) = F'(\frac{[x]_q}{\Delta_{in}})\frac{Q\Delta_{out}}{q_{out}}$ as continuous. Then its corresponding f_{pso} and f_{pse} are defined as follows.

$$f_{pso}(x) = \frac{f(x) - f(q-1-x)}{2} : \mathbb{R} \to \mathbb{R}$$

$$f_{pse}(x) = \frac{f(x) + f(q-1-x)}{2} : \mathbb{R} \to \mathbb{R}$$

To compute f_{pso}, the ciphertext is first preprocessed by bootstrapping it using the function $f_0 : \mathbb{Z}_q \to \mathbb{Z}_q$, which is independent of F'. Then we evaluate the LUT-specific function $f'_{pso} : \mathbb{Z}_q \to \mathbb{Z}_Q$ on the obtained result. Similarly, we evaluate f_1 and f'_{pse} consecutively to compute f_{pse}. The functions used in bootstraps are defined as follows.

$$f_0(x) = \begin{cases} \lfloor \beta + \frac{q/2-2\beta-1}{q/2-1}x \rceil & x \in [0, \frac{q}{2}-1] \\ q - f_0(x-\frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases}$$

$$f'_{pso}(x) = \begin{cases} \lfloor f_{pso}(0) \rceil & x \in [0, \beta] \\ \lfloor f_{pso}(\frac{q/2-1}{q/2-2\beta-1}(x-\beta)) \rceil & x \in [\beta+1, \frac{q}{2}-\beta-1] \\ \lfloor f_{pso}(\frac{q}{2}-1) \rceil & x \in [\frac{q}{2}-\beta, \frac{q}{2}-1] \\ Q - f_{pso}(x-\frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases}$$

$$f_1(x) = \begin{cases} \lfloor \beta + \frac{q}{4} + \frac{q/2-2\beta-1}{q/2-1}x \rceil & x \in [0, \frac{q}{2}-1] \\ q - f_1(x-\frac{q}{2}) & x \in [\frac{q}{2}, q-1] \end{cases}$$

$$f'_{pse}(x) = \begin{cases} \lfloor f_{pse}(0) \rceil & x \in [\frac{q}{4}, \frac{q}{4}+\beta] \\ \lfloor f_{pse}(\frac{q/2-1}{q/2-2\beta-1}(x-\frac{q}{4}-\beta)) \rceil & x \in [\frac{q}{4}+\beta+1, \frac{3q}{4}-\beta-1] \\ \lfloor f_{pse}(\frac{q}{2}-1) \rceil & x \in [\frac{3q}{4}-\beta, \frac{3q}{4}-1] \\ Q - f_{pse}(x-\frac{q}{2}) & x \in [0, \frac{q}{4}-1] \bigcup [\frac{3q}{4}, q-1] \end{cases}$$

A.1.3 ***Comp** (fast).* When F' is an odd function, it can be treated approximated as a pseudo-odd function to reduce the number of bootstraps from four to two (or from three to two when multi-value bootstrap is used). Similarly, when F' can be treated as pseudo-even when it is even.

We focus on the case when F' is odd for brevity since the two cases are very similar. To make the pseudo-even part disappear (i.e., $f_{pse} = 0$), the trick is to compute $f(x+0.5)$ instead of $f(x)$.

A.2 Output Noise Variances

We briefly prove the output variances of **FDFB-Compress**, **EvalFunc** and **Comp**. They differ from other FDFB methods when evaluating continuous functions because their preprocessing of the input ciphertext introduces an input error to F'. This makes their output variances dependent on the Lipschitz constant of F'. i.e., an input error of F' with a variance of σ_{in}^2 can increase the output error variance by at most $\frac{L^2}{\Delta_{in}^2}\sigma_{in}^2$, where L is the Lipschitz constant of F'. The proof for other FDFB methods is omitted because the output noise variance is the same as the discrete case.

A.2.1 ***FDFB-Compress***. The input to F' in **FDFB-Compress** can be roughly expressed as $\lfloor f_C'^{-1}(\lfloor f_C'(x)\rceil + e_{boot})\rceil$, where f_C' is a two-piece linear function, and the slope of both pieces have the same absolute value $k_2 = \frac{q/4-2\beta}{q/2-1}$. Then the input error to F' is $k_2^{-1}(e_{boot} + e_{rnd,0}) + e_{rnd,1}$, where $e_{rnd,i}$ is the rounding error of the i-th rounding operator and has a variance of $\frac{1}{4}$. The input error to F' has a variance of $k_2^{-2}(\sigma_{boot}^2 + \frac{1}{4}) + \frac{1}{4} \approx k_2^{-2}\sigma_{boot}^2$. The final output error is $\frac{L^2}{\Delta_{in}^2 k_2^2}\sigma_{boot}^2 + \sigma_{boot}^2$.

A.2.2 ***EvalFunc***. The input error to F' in **EvalFunc** is e_{boot}. Consequently, the final output variance is $\frac{L^2}{\Delta_{in}^2}\sigma_{boot}^2 + \sigma_{boot}^2$.

A.2.3 ***Comp***. Denote the derivative of $F'(x), f_{pso}' = f_{pso}(\frac{x}{\Delta_{in}})$ and $f_{pse}' = f_{pse}(\frac{x}{\Delta_{in}})$ as $d(x), d_o(x)$ and $d_e(x)$. Note that $F' = f_{pso}' + f_{pse}'$. The input errors to f_{pso}' and f_{pse}' are both $k_1^{-1}e_{boot} = \frac{q/2-1}{q/2-2\beta}e_{boot}$ (rounding errors are omitted). Then the final output error variance is $\frac{d_o^2+d_e^2}{k_1^2\Delta_{in}^2}\sigma_{boot}^2 + 2\sigma_{acc}^2 + \sigma_{com}^2$. Note that $d = d_o + d_e$ and $|d_o|, |d_e| \le |d| = L$, we have $d_o^2 + d_e^2 \le L^2$. i.e., the output error variance is $\frac{L^2}{k_1^2\Delta_{in}^2}\sigma_{boot}^2 + 2\sigma_{acc}^2 + \sigma_{com}^2$.

A.2.4 ***Comp (fast).*** **Comp** (fast) has nearly the same output variance as **Comp**. The only difference is that the input to **Comp** (fast) is added by $\frac{1}{2\Delta_{in}}$ at the beginning, leading to an extra noise of $\frac{L}{2\Delta_{in}}$ at most. In practice, this extra noise is very small compared to the output variance, and we omit it in our analysis.

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 [25] for more discussion on BFV multiplication.

Theorem B.1. *Let $c_i = (\boldsymbol{b}_i, \boldsymbol{a}_i)RLWE_{\mathbf{s},N,Q}(\frac{Q}{p}m_i + e_i + \boldsymbol{e}_i)$ 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 for re-linearization B_{rl}, SampleExtract(BFVMult$(c_0, c_1)) = \frac{Q}{p}m_0m_1 + e$, where the variance of e is*

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

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.

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

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 σ_{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 σ^2 is the encryption error variance.

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

Remark. For ordinary BFV multiplication where m_0 and m_1 are polynomials instead of scalars, the terms in line $1 \sim 3$ in the expression of σ_m^2 needs to be multiplied by N. The overflow of m_0m_1 modulo p also introduces an additional term. In **FDFB-BFVMult**, the bootstrapping error e_i is magnitudes larger than the key switching error $\mathbf{e}_i$. This means the dominating term of σ_m^2 is $p^2\sigma_{ms}^2(\sigma_0^2 + \sigma_1^2)$. In contrast, **WoP-PBS** estimates this term to be N times larger, which leads to inefficient parameters. We state this observation as Lemma B.2.

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

B.2 Simple Proofs of Table 1

B.2.1 ***FDFB-BFVMult.*** Applying Lemma B.2 and the noise growth formula for multi-value bootstraps gives the output variances of **FDFB-BFVMult**. For non-multi-value versions of **FDFB-BFVMult**, replacing σ_{ms}^2 with $\frac{N}{18}$ and σ_i^2 with σ_{acc}^2 gives the expression listed in Table 1.

For the multi-value version of **FDFB-BFVMult**$_1$, the TV_1 for f_{sgn1} has an L2 norm of 4. Thus, σ_0^2 is close to $4\sigma_{acc}^2$ instead of $p(p-1)^2\sigma_{acc}^2$. Note that for the multi-value version of **FDFB-BFVMult**$_2$, 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 TV_1 for f_{sgn2} still has an L2 norm of 4, while the TV_1 for f_{diff} has an L2 norm of $4p(p-1)^2$ at most.

B.2.2 ***FDFB-Compress.*** Since **FDFB-Compress** computes two consecutive functional bootstraps, the output and intermediate errors are the same as bootstrapping errors. The requirements for these two errors are well explained in Lemma 3.1.

B.2.3 ***FDFB-CancelSign.*** The output of the first functional bootstrap has an error variance of σ_{acc}^2. After modulus switching to q_{pk} and LWE-to-RLWE packing, the variance becomes $\sigma_{acc}^2 + (\frac{Q}{q_{pk}})^2\sigma_{ms}^2 + \sigma_{pk}^2$. The second bootstrap adds σ_{acc}^2 to the error variance.

B.2.4 ***FDFB-Select.*** 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 $\sigma_{acc}^2+(\frac{Q}{q_{pk}})^2\sigma_{ms}^2+2\sigma_{pk}^2$. As in **FDFB-CancelSign**, the final bootstrap adds another σ_{acc}^2 to the error variance.

For the multi-value version, $\mathrm{ct}_{pos}, \mathrm{ct}_{neg}$ and ct_{sgn} have error variances of $p(p-1)^2\sigma_{acc}^2$. However, the error in 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 $p(p-1)^2\sigma_{acc}^2 + (\frac{Q}{q_{pk}})^2\sigma_{ms}^2 + 2\sigma_{pk}^2$.

B.2.5 ***FDFB-SelectAlt.*** 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 ct_{hsum} in the last step of **FDFB-SelectAlt** adds σ_{acc}^2 to this variance.

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.

C TABLES

This section presents the full table of parameters used in the benchmarks and the results of the benchmarks. B_{g0} is the base of RLWE′ ciphertexts used in **FDFB-KS**. P is the auxiliary prime used in BFV multiplication [25]. 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 $n35 = 1340$, $n25 = 955$, and $n20 = 760$ correspond to the parameter sets in Table 3. $P53$ 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.

Table 5: Running Time of Homomorphic Digit Decomposition Methods under $\mathbf{PARAM_{decomp}}$.

$\log_2(q)$	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 6: 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}
FDFB-BFVMult$_1$*	777	16	$n35$	2^{11}	2^{11}	$Q53$	2^{20}	2^5	2^{18}		2^{25}	2^5		+	-	-	$P53$	2^{27}
FDFB-BFVMult$_2$*	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$_2$*	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}	-	+	-		
FDFB-BFVMult$_1$	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$_2$	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 7: 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}
FDFB-BFVMult$_1$ *	458	16	$n20$	2^{11}	2^{11}	$Q53$	2^{20}	2^5	2^{18}		2^{20}	2^5		+	-	-	$P53$	2^{27}
FDFB-BFVMult$_2$ *	458	16	$n20$	2^{11}	2^{12}	$Q53$	2^{20}	2^5	2^{18}		2^{20}	2^5		+	-	-	$P53$	2^{27}
FDFB-BFVMult$_2$ *	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		+	-	-		

Table 8: Parameters, Performance and Precision for Benchmark under $\text{PARAM}_{\text{continuous}}$, Evaluating Sigmoid$(-8, 8) \to (-2, 2)$.

Name	Time(ms)	σ_{out}	p	n	N	q	Q	q_ks	B_ks	B_g	q_pk	B_pk	d			Δ_{in}	Δ_{out}	q_{out}
EvalFunc	750	1.476e-2	0	$n25$	2^{11}	2^{11}	$Q53$	2^{25}	2^5	2^{27}			-	-	-	2^7	2^{23}	2^{25}
Comp	1490	5.682e-3	0	$n25$	2^{11}	2^{12}	$Q53$	2^{25}	2^5	2^{27}			-	-	-	2^8	2^{23}	2^{25}
FDFB-Compress	752	1.081e-2	0	$n25$	2^{11}	2^{12}	$Q53$	2^{25}	2^5	2^{27}			-	-	-	2^8	2^{23}	2^{25}
FDFB-CancelSign	773	3.338e-4	0	$n25$	2^{11}	2^{11}	$Q53$	2^{25}	2^5	2^{27}	2^{25}	2^5	-	-	+	2^7	2^{23}	2^{25}
FDFB-Select	1520	3.403e-4	0	$n25$	2^{11}	2^{12}	$Q53$	2^{25}	2^5	2^{27}	2^{25}	2^5	-	+	-	2^8	2^{23}	2^{25}
FDFB-SelectAlt	1140	4.000e-4	0	$n25$	2^{11}	2^{12}	$Q53$	2^{25}	2^5	2^{27}	2^{25}	2^5	-	-	+	2^8	2^{23}	2^{25}
Comp (fast)	749	5.237e-3	0	$n25$	2^{11}	2^{11}	$Q53$	2^{25}	2^5	2^{27}			-	-	-	2^7	2^{23}	2^{25}