Functional Bootstrapping for Packed Ciphertexts via Homomorphic LUT Evaluation

Dongwon Lee, Seonhong Min, and Yongsoo Song

Seoul National University, Seoul, Republic of Korea {dongwonlee95,minsh,y.song}@snu.ac.kr

Abstract. The only known method for designing lattice-based Fully Homomorphic Encryption (FHE) involves first constructing a Somewhat Homomorphic Encryption (SHE) scheme and then applying a bootstrapping technique. In FV (Fan-Vercauteren) or CKKS (Cheon-Kim-Kim-Song) schemes, the bootstrapping technique can be viewed as a homomorphic evaluation of the decryption circuit that recovers the computational capability of a ciphertext by reducing its error or increasing its ciphertext level while preserving the encrypted plaintext. Meanwhile, TFHE scheme features programmable bootstrapping, which enables the evaluation of an arbitrary function while refreshing a ciphertext. However, it remains an open question how to achieve similar functionality with FV or CKKS schemes.

In this work, we introduce a novel functional bootstrapping framework, providing enhanced and versatile functionality for FV and CKKS. That is, our bootstrapping method can evaluate an arbitrary function on a FV or CKKS ciphertext during bootstrapping and output a FV ciphertext. Specifically, the FV-to-FV functional bootstrapping extends the functionality of FV by allowing the evaluation of functions between different input and output plaintext spaces. In addition, our CKKS-to-FV functional bootstrapping involves a natural conversion from CKKS to FV, aligned with the evaluated function. We believe that these new functionalities will enhance the usability of FHE in privacy-preserving applications.

At the heart of our functional bootstrapping framework is a homomorphic Look-Up Table (LUT) evaluation method where we represent any function using the set of operations supported by the FV scheme. We provide a proof-of-concept implementation and present benchmarks of the functional bootstrapping process. Our experiments show that the new functional bootstrapping takes about 46.5 or 171.4 seconds when evaluating the delta or sign function over a plaintext of 9 or 13 bits precision, respectively.

Keywords: Fully Homomorphic Encryption, Functional Bootstrapping, Bootstrapping, FV

1 Introduction

Fully Homomorphic Encryption (FHE) enables the evaluation of arbitrary functions on encrypted data without the need for decryption. It prevents private information from being revealed while evaluating data within an untrusted environment. The pioneering work on FHE was introduced by Gentry [28], and subsequent research has focused on leveraging the Learning With Errors (LWE) problem [52] or its ring variant, the Ring LWE (RLWE) problem [46]. Notable advancements in FHE construction such as FV [8, 25], CKKS [16], GSW [32], BGV [9], and TFHE [20] have emerged based on these foundational problems.

In these encryption schemes, a ciphertext accumulates a certain amount of noise and the noise increases while performing homomorphic operations. Since excessive noise can lead to incorrect decryption results, managing the noise is a critical issue. While there are several well-known techniques such as the special modulus technique [31] to mitigate this issue, there exists no known solution to perfectly remove the noise growth from homomorphic operations. Consequently, to construct an FHE scheme, a procedure to lower the ciphertext noise is required after a substantial number of operations. Addressing this challenge, the bootstrapping technique was introduced by Gentry [29] to effectively lower the ciphertext noise and ensure the integrity of decryption results in the face of noise accumulation in homomorphic evaluation.

Informally, the bootstrapping technique involves homomorphically evaluating the decryption circuit to refresh the ciphertext, a process known for high computational cost due to the lack of straightforward support for the decryption circuit operations within FHE schemes. In the CKKS scheme, encrypting fixedpoint numbers, the computation of the modulus operation is conducted through polynomial approximation to decrypt the message homomorphically [14]. In particular, various investigations into CKKS bootstrapping have concentrated on refining the approximation of the modulus operation [10, 42] to realize efficient bootstrapping procedures. Conversely, in the FV scheme, the evaluation of the decryption circuit follows a distinct approach compared to CKKS. Specifically, several studies, such as those by [11, 27, 35], have proposed circuit designs intending to eliminate specific least significant digits to mitigate noise and consequently enhance the overall performance of the circuit. According to these works, significant advancements in FV bootstrapping have been observed, with existing works predominantly emphasizing the reduction of ciphertext noise while preserving the integrity of the message value. In other words, previous works do not support computation of an arbitrary function during the bootstrapping. Consequently, bootstrapping is not directly involved in altering the calculation of the circuit but serves as a means to enable additional operations by minimizing the error.

In addition, both schemes lack the functionaliy to convert the plaintext space. However, in the real world, there are several functions which the plaintext domain is converted. For example, in practical applications like decision tree, it requires a comparison where input data typically comprises fixed-point numbers while the output is discrete (integer) numbers. Until recent studies [17], these functions were often calculated with a single FHE scheme, especially CKKS, to encrypt fixed-point number. However, since CKKS support operations for approximate numbers, the output includes a noise and it may causes inaccurate result in further operations. Even if the input and output are both discrete numbers, functions such as equality check can be considered as cases where the plaintext domain is changed since its output is binary. In these cases, depending on the property of the data such as data type, range, or the number of data, each data may have a suitable parameter for optimal performance. A proper parameter set tailored to the specific properties of the data can contribute to performance improvements and allow for more flexible packing within the ciphertext [9].

Unlike the FV or CKKS schemes, TFHE-like cryptosystems [2, 20, 23] allows to evaluate of an arbitrary function from any domain to any range during the bootstrapping. To the best of our knowledge, there has been no studies for the RLWE-based FHE schemes supporting these functionalities.

1.1 Our Contribution

In this study, we present a novel bootstrapping method called the functional bootstrapping. This new bootstrapping technique enables the computation of arbitrary functions during bootstrapping of any RLWE-based encryption as illustrated in Fig. 1.

As a concrete example of our method, we provide a FV-to-FV and CKKSto-FV functional bootstrapping since CKKS and FV are the most commonly used RLWE-based FHE schemes. In FV-to-FV functional bootstrapping, the plaintext modulus can be switched during the bootstrapping and consequently it extends the domains of functions that can be evaluated during the functional bootstrapping into a set of functions between different plaintext spaces. On the other hand, in CKKS-to-FV case, our functional bootstrapping converts the encryption scheme and outputs an FV ciphertext encrypting a function evaluations of the input message vector. With these features, the usability of the FHE to construct privacy-preserving applications can be further optimized and extended.

We realize the functional bootstrapping from an evaluation algorithm for a general Look-Up Table (LUT) over the commutative ring \mathbb{Z}_{p^r} for some prime p. During the bootstrapping, the message and the noise term are combined as an element in \mathbb{Z}_{p^r} , where the message is stored in the high bits and the error is stored in the low bits. In the existing bootstrapping method, the message is homomorphically recovered by iteratively eliminating the erroneous least significant bits (LSB). On the other hand, our functional bootstrapping computes an arbitrary function over the message, while removing the noise in the low bits.

To obtain such a functionality, we first construct a series of polynomials designed to selectively eliminate the LSBs. Then with these polynomials, we provide an optimized LUT evaluation method specified for a simple LUT from \mathbb{Z}_{p^r} to \mathbb{Z}_p , which has a form of Heaviside function. This simple LUT evaluation requires $\approx \frac{16}{3} \sqrt{r^3 p}$ key-switching operations and consumes $r \log p + \log(r!)$ depth asymptotically. Finally, an arbitrary LUT can be computed with a linear combination of Heaviside functions. We also generalize this result to an arbitrary LUTs from \mathbb{Z}_{p^s} to \mathbb{Z}_{p^s} , by introducing homomorphic lifting operation to the bootstrapping.

Our method takes advantage over the programmable bootstrapping of TFHElike cryptosystems with its support of Single-Instruction Multiple-Data (SIMD) operations and capability of handling large plaintext modulus. We remark that this difference mainly comes from exploiting the algebraic structure of the commutative ring \mathbb{Z}_{p^r} itself, rather than the ring structure.

Finally, we implement and provide a benchmark analysis for both functions for our new method for handling the delta and sign functions utilizing an opensource FHE library, Lattigo [48]. In addition, we outline various applications where functional bootstrapping can be effectively employed. These applications demonstrate the practical utility of our new bootstrapping method and highlight its potential impact in real-world use cases involving the evaluation of arbitrary univariate functions within the FHE framework.

Fig. 1: Concept of functional bootstrapping.

1.2 Related Works

Liu et al. [44] recently improved the amortized bootstrapping of the TFHE scheme via the FV bootstrapping. Their method includes a limited form of functional bootstrapping. In a nutshell, they utilized a temporal plaintext modulus of large size to encode a binary message in finite field \mathbb{Z}_3 to evaluate arbitrary binary gates with an addition and a bootstrapping [23]. They also generalized their result with plaintext modulus up to 12 bits. This approach was later generalized in [45] by introducing a loose definition of bootstrapping, which only guarantees correctness for a subset of the plaintext space.

Concurrently, Okada et al. suggested a TFHE-style functional bootstrapping in the FV scheme in [50] in a similar setting of leveraging a fake plaintext modulus. Their main idea is to perform a blind rotation over the exponent of the roots of unity, instead of the monomial. However, the functionality of these works is limited, since they only support a small modulus, or suffer from a high computational cost.

On the other hand, the asymptotic complexity of FV/BGV bootstrapping is recently improved in [47] and [39] concurrently. Although those two methods are conceptually different, they both leverage the message extraction technique from [38] to make the input ciphertext independent from the message. Subsequently, the bootstrapping algorithm can be adapted to rely solely on the error term. We note that while [39] employs the CKKS cryptosystem for extracting the noise term, [47] utilizes the BFV scheme to remove the noise term. As a result, the bootstrapping complexity is now only dependent on the bound of the error, rather than the size of plaintext space. However, we stress that this heuristic cannot be applied to this work, since their method does not directly work on the message itself.

The homomorphic evaluation of the sign/delta function using the FV scheme is well studied in various previous researches, to realize comparison operation or SQL query homomorphically. Cheon et al. [18, 19] utilized a bivariate polynomial interpolation to compare two (large) integers. This approach is further refined by Tan et al. [55] by leveraging the finite field structure to compare the input integers digit-wise. On the other hand, a univariate polynomial based on the interpolation approach is also proposed in [49]. Later, Kim et al. [40] improved this method by encoding a large integer into a finite field $GF(p^d)$ and leveraging the Frobenius automorphism in order to evaluate a polynomial, thus not consuming modulus from homomorphic multiplications. Iliashenko and Zucca [36] achieved a significant speedup of both approaches, by observing that we can make the coefficient vector of the interpolation polynomials sparse.

2 Background

2.1 Notation

We denote the ring of integers of the 2N-th cyclotomic field for some power-oftwo N by $R = \mathbb{Z}[X]/(X^N + 1)$, and the residue ring of R modulo an integer $Q > 0$ by $R_Q = \mathbb{Z}_Q[X]/(X^N+1)$. We also denote $[a]_Q$ as the residue of a modulo Q. Throughout the paper, we write $x \leftarrow D$ to represent that x is sampled from the distribution D . We denote the uniform distribution over a finite set S by $U(S)$. For $\sigma > 0$, D_{σ} denotes a distribution over R sampling N coefficients

independently from the discrete Gaussian distribution of variance σ^2 and χ as a key distribution over R. We also use $\overline{a_{r-1}a_{r-2} \ldots a_0}$ to denote the base-p representation of $a \in \mathbb{Z}_{p^r}$ where $a_i \in \mathbb{Z}_p$, *i.e.*, $a = \sum_{i=0}^{r-1} a_i p^i$.

2.2 The Fan-Vercauteren (FV) Scheme

The FV scheme [25] supports arithmetic operations such as addition and multiplication over integers. In FV, the plaintext space is defined by $\mathbb{Z}_{p^r}[X]/\Phi_m(X)$ where p is a prime, r and m are positive integers, and $\Phi_m(X)$ denotes the m-th cyclotomic polynomial. When $p \nmid m$, there is an isomorphism $\sigma : \mathbb{Z}_{p^r}[X]/\Phi_m(X) \to$ $\prod_{i=1}^k \mathbb{Z}_{p^r}[X]/F_i$ for some irreducible polynomials F_i $(1 \leq i \leq k)$ of degree d, where d is multiplicative order of p in \mathbb{Z}_m^{\times} , and $k = \phi(m)/d$. From this isomorphism, a vector $\mathbf{m} \in \mathbb{Z}_{p^r}^k$ can be encoded into a single plaintext $\sigma^{-1} \circ \varphi(\mathbf{m})$ where φ is a natural embedding $\mathbb{Z}_{p^r}^k \hookrightarrow \prod_{i=1}^k \mathbb{Z}_{p^r}[X]/F_i$. (We will omit the natural embedding φ in the later sections for better readability.) Employing this encoding strategy enables the performance of component-wise addition and multiplication operations on vectors in $\mathbb{Z}_{p^r}^k$ by adding and multiplying the ring elements in $\mathbb{Z}_{p^r}[X]/\Phi_m(X)$. In later sections, we will refer to each component of the vector as the *slot*. In this work, we choose $m = 2N$ for some power-of-two N for simplicity of the description.

As an independent interest, a technique known as the *gadget decompositon* is applied to reduce noise growth during homomorphic operations. It is a commonly used technique in lattice-based FHE cryptosystems [5, 15, 34]. We call $h: R_Q \rightarrow$ R^{α} and $\mathbf{g} = (g_0, g_1, \dots, g_{\alpha-1}) \in R_Q^{\alpha}$ a decomposition function and the gadget vector if they satisfy the equation:

$$
\sum_{0 \le i < \alpha} b_i \cdot g_i = a \pmod{Q} \quad \text{and} \quad \|\mathbf{b}\|_{\infty} \le B
$$

where $B > 0$ is a real constant, Q and α are positive integers, a is an arbitrary ring element over R_Q , and $\mathbf{b} = (b_0, b_1, \ldots, b_{\alpha-1})$ is generated from $h(a)$.

A detailed description of the FV scheme is given below.

• FV. Setup(1^{λ}): Set the ring degree N, the plaintext modulus p^{r} , the ciphertext modulus Q, the key distribution χ over R, and the error parameter σ . Here, p is a prime and $r > 0$ is an integer. Choose a gadget decomposition $h : R_Q \to R^{\alpha}$ with a gadget vector $\mathbf{g} \in R_Q^{\alpha}$. Output the parameter set $\mathsf{pp} = (N, p^r, Q, \chi, \sigma, h, \mathbf{g})$.

• FV.KeyGen(pp): Sample $s \leftarrow \chi$, $a \leftarrow \mathcal{U}(R_Q)$ and $e \leftarrow D_{\sigma}$. Set the secret and public keys as $sk = s$ and $pk = (b, a) \in R_Q^2$ where $b = -sk \cdot a + e \pmod{Q}$. Sample $\mathbf{k}_1 \leftarrow \mathcal{U}(R_Q^{\alpha})$ and $\mathbf{e} \leftarrow D_{\sigma}^{\alpha}$, and set the relinearization key as rlk = $(k_0, k_1) \in R_Q^{\alpha \times 2}$ where $k_0 = -\mathsf{sk} \cdot k_1 + \mathsf{e} + \mathsf{sk}^2 \cdot \mathsf{g} \pmod{Q}$. Output sk, pk and rlk.

• FV.Encode(m): Let k be the biggest power-of-two two such that $2k | p - 1$. Then, given a message vector $\mathbf{m} \in \mathbb{Z}_{p^r}^k$, return a plaintext $\mu = \sigma^{-1}(\mathbf{m}) \in R_{p^r}$ where σ is the isomorphism defined at the beginning of the section.

• FV.Decode(μ): Given $\mu \in R_{p^r}$, return $\mathbf{m} = \sigma(\mu)$.

• FV.Enc(pk; μ): Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_{\sigma}$. Given an encoding $\mu \in R_{p^r}$, output the ciphertext $ct = w \cdot pk + (\Delta \cdot \mu + e_0, e_1) \pmod{Q}$, for $\Delta = \lfloor Q/p^r \rceil$.

• FV.Dec(sk; ct): Given a ciphertext $ct = (c_0, c_1) \in R_Q^2$ and associated secret key sk, return $\mu = \lfloor (p^r/Q) \cdot (c_0 + c_1 \cdot$ sk) $\rfloor \pmod{p^r}$

• FV.Add(ct, ct'): Given two ciphertexts ct , $ct' \in R_Q^2$, output $ct_{add} = ct + ct'$ $\pmod{\overline{Q}}$.

• FV.Mult(rlk; ct, ct'): Given two ciphertexts $ct = (c_0, c_1)$, $ct' = (c'_0, c'_1) \in R_Q^2$ and the relinearization key rlk $\in R_Q^{\alpha}$, let (d_0, d_1, d_2) such that $d_0 = \lfloor (p^r/Q) \cdot c_0 c'_0 \rfloor$ $(\text{mod } Q), d_1 = \lfloor (p^r/Q) \cdot (c_0 c'_1 + c'_0 c_1) \rceil \pmod{Q}, \text{and } d_2 = \lfloor (p^r/Q) \cdot c_1 c'_1 \rceil \pmod{Q}.$ Output the ciphertext

$$
\mathsf{ct}_{mul} = (d_0, d_1) + (\lfloor \langle h(d_2), \mathbf{k}_0 \rangle \rfloor, \lfloor \langle h(d_2), \mathbf{k}_1 \rangle \rfloor) \pmod{Q}.
$$

2.3 The Cheon-Kim-Kim-Song (CKKS) Scheme

The CKKS scheme [16] provides a framework for FHE tailored to fixed-point arithmetic in an approximate manner. In the CKKS scheme, the message space is defined as $\mathbb{C}^{N/2}$ and the plaintext space is $R = \mathbb{Z}[X]/(X^N + 1)$ for a power-of-two N. Given a complex vector $\mathbf{m} \in \mathbb{C}^{N/2}$, the encoding process proceeds as follows: first, find an N-degree polynomial $p(X)$ such that $p(\zeta_i) = m_i$ and $p(\overline{\zeta_i}) = \overline{m_i} =$ $\overline{p(\zeta_i)}$ $(1 \leq i \leq N/2)$ where ζ_i 's are 2N-th roots of unity, *i.e.*, roots of $X^N + 1$. This polynomial $p(X)$ is real since it is invariant under the conjugate operation, enabling point-wise computations on the vector **. Now, for some sufficiently** large real parameter $\Delta > 0$, we use $\mu(X) := |\Delta \cdot p| \in \mathbb{Z}[X]/(X^N + 1)$ as the encoding of **m**. Naturally, the decoding can be computed as $(\Delta^{-1} \cdot \mu(\zeta_i))_{1 \leq i \leq N/2}$. Observe that this encoding strategy does not preserve the exact values of the elements in **m**, but introduces an error bounded by $O(\Delta^{-1})$.

Note that a common choice of $\{\zeta_i\}_{1 \leq i \leq N/2}$ is setting $\zeta_i = \exp(-5^i \pi/N)$ to give a cyclic slot structure, which can be rotated with a simple Frobenius map $X \mapsto X^5$ in practice.

• CKKS.KeyGen(pp): Sample $s \leftarrow \chi$, $a \leftarrow \mathcal{U}(R_Q)$ and $e \leftarrow D_{\sigma}$. Set the secret and public keys as $\mathsf{sk} = s$ and $\mathsf{pk} = (b, a) \in R^2_Q$ where $b = -s \cdot a + e \pmod{Q}$.

• CKKS.Encode $(\Delta; m)$: Given a scaling factor Δ and a message vector $m =$ $(m_1, \ldots, m_{N/2}) \in \mathbb{C}^{N/2}$, return a plaintext $\mu = \lfloor \Delta \cdot p \rfloor \in R$ where $p \in \mathbb{R}[X]/(X^N +$ 1) is a real polynomial such that $p(\zeta_i) = m_i$ and $p(\overline{\zeta_i}) = \overline{m_i}$ for $1 \leq i \leq N/2$.

• CKKS.Decode $(\Delta; \mu)$: Given $\mu \in R$, return $\mathbf{m} = (\Delta^{-1} \cdot \mu(\zeta_1), \dots, \Delta^{-1} \cdot \mu(\zeta_{N/2})).$

• CKKS.Enc(pk; μ): Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_{\sigma}$. Given an encoding $\mu \in R$, output the ciphertext $ct = w \cdot pk + (\mu + e_0, e_1) \pmod{Q}$.

• CKKS.Dec(sk; ct): Given a ciphertext $ct = (c_0, c_1) \in R_Q^2$ and associated secret key sk = s, return $\mu = c_0 + c_1 \cdot s \pmod{Q}$.

Functional bootstrapping in our context is based on the existing FV bootstrapping method. Therefore, we omit CKKS arithmetic operations (and relinearization key generation) as they are not used in our paper. We also stress that the key structures of FV and CKKS are identical, allowing us to share the key while performing functional bootstrapping from CKKS to FV.

2.4 Bootstrapping of FV

Bootstrapping is a technique employed to reduce the noise of a ciphertext, allowing a user to perform an unlimited number of computations without compromising the privacy of the underlying message. Essentially, bootstrapping involves homomorphically evaluating the decryption circuit. In the context of the FV scheme, the decryption circuit is given by $\lfloor p^r/q \cdot (b + a \cdot \mathsf{sk}) \rceil$ where $(b, a) \in R_q^2$ is the input ciphertext and p^r is the plaintext modulus for a prime p. The basic pipeline of the bootstrapping for the FV scheme consists of four steps, namely, ModSwitch, Coeffs2Slots, DigitExtract and Slots2Coeffs [11, 35].

Let us briefly explain the functionality of each step. In ModSwitch, the input ciphertext modulus is changed into a smaller modulus p^e , where $e > r$ is the smallest integer possible as long as the decryption does not fail. This allows us to reduce the bootstrapping depth and perform bootstrapping more efficiently in the further steps. Let the output of the ModSwitch be $(b, a) \in R_{p^e}^2$. In the rest of the steps, our goal is to homomorphically decrypt this ciphertext itself, i.e., computing $\left\vert \left[b+a\cdot\mathsf{sk}\right]_{p^{e}}/p^{e-r}\right\rvert$. To achieve this, we first generate a ciphertext that encrypts $b + a \cdot$ sk with plaintext modulus p^e . Now, note that the rounding function $|\cdot|$ is applied to the coefficients in the decryption circuit, the coefficients are required to be moved to the slots for the sake of a SIMD computation. This can be achieved with a simple linear transformation, in Coeffs2Slots step. Then, in the following DigitExtract step, the division and rounding are evaluated on the coefficients homomorphically. Finally, the rounded coefficients in slots are brought back to the coefficients with the inverse linear transformation in the Slots2Coeffs step.

In the FV/BGV bootstrapping process, DigitExtract is the main bottleneck. The functionality of DigitExtract is essentially removing the lower $e - r$ digits in base p . The significant challenge when achieving this functionality is that there is no direct polynomial representation of this operation unless only the LSB is removed. To address this challenge, an iterative approach to remove the LSB and change to a smaller plaintext modulus is employed. In [11], the authors showed that there always exists a polynomial G_i such that $G_i(x) = [x]_p \pmod{p^i}$, called the i-th digit extraction polynomial. Assume that a single input of DigitExtract is given as follows:

$$
w = \overline{w_{e-1}w_{e-2} \dots w_0} = \sum_{i=0}^{e-1} w_i p^i \in \mathbb{Z}_{p^e}
$$
, where $w_i \in \mathbb{Z}_p$.

Then, we obtain $w - [w]_p = \overline{w_{e-1}w_{e-2} \ldots w_10}$ using the digit extraction polynomial. This can be understood as encryption of $\overline{w_{e-1} \dots w_1}$ with plaintext modulus

 p^{e-1} , and the LSB is removed successfully. We remark that this plaintext modulus changing operation is called the *homomorphic division by p*, and written as x/p for input x. Then, it directly follows that the decryption circuit can be computed by repeating this iteratively.

In practice, the computation of $[w_0]_{p^e}$, $[w_1]_{p^{e-1}}$, ..., $[w_{e-r-1}]_{p^{r+1}}$ are conducted differently for the minimization of the depth consumption. The fundamental idea is that the computation of $[w_i]_{p^{e-i}}$ does not necessarily require $\overline{w_{e-1} \dots w_i}$ as an input, but any number with the form $\times \cdots \times w_i$ is sufficient. Let us elaborate on how one can obtain $[w_1]_{p^{e-1}}$ with the minimum depth cost as an example. Firstly, observe that he second bit of $G_2(w)$ is zero, *i.e.*, $G_2(w)$ = $\overline{\times \cdots \times 0 w_0}$. Then, for $w^{(1)} := (w - G_2(w)) / p \in \mathbb{Z}_{p^{e-1}}$, the LSB of $w^{(1)}$ is w_1 . Therefore, for the computation of $[w_1]_{p^{e-1}}$, it is sufficient to evaluate $G_{e-1}(w^{(1)})$ instead of the direct computation. Likewise, $[w_2]_{p^{e-2}}$ can be computed from $G_3(w) = \overline{\times \cdots \times 00w_0}$ and $G_2(w^{(1)}) = \overline{\times \cdots \times 0w_1}$, since $G_{e-2}(w^{(2)}) = [w_2]_{p^{e-2}}$ for $w^{(2)} := ((w - G_3(w)) / p - w^{(1)}) / p$. The rest of the computations of $[w_i]_{p^{e-i}}$ can be conducted in a similar way, by computing a number with the form $\overline{\times \cdots \times w_i} \in \mathbb{Z}_{p^{e-i}}$. We note that this approach requires $\approx (e-r)^2/2$ polynomial evaluations and $O((e-r) \log p + \log e)$ multiplicative depth, compared to e – r polynomial evaluations and $O((e-r) \log p + \log(e!/r!))$ depth of the naïve approach.

As an independent interest, Chen and Han [11] showed that one can enhance the bootstrapping speed by rearranging the order of each step. In their construction, Slots2Coeffs comes the first, and is followed by ModSwitch, Coeffs2Slots and DigitExtract. The underlying idea of their construction is to make the error term augmented to the message vector, instead of the coefficients. As a result, it only needs to perform digit extraction on a single ciphertext compared to the previous method, which requires digit extraction for d ciphertexts where d is the multiplicative order of p in \mathbb{Z}_m^{\times} .

We remark that a homomorphic trace evaluation is required after ModSwitch to remove the obsolete noise terms when the number of the slots is smaller than the ring degree. After ModSwitch, the augmented error polynomial does not have a valid encoding structure, and therefore a (partial) trace function should be evaluated for further computation. We will not discuss this operation deeply in this paper since this is an orthogonal issue to this work.

3 Our Functional Bootstrapping Framework

Bootstrapping is typically utilized as a black-box technique to reduce the noise bound, not directly influencing the performance of the circuit evaluation. In this section, we discuss how one can give extra functionality to the bootstrapping by integrating the bootstrapping with circuit evaluation. In the literature, such functionality was supported only in TFHE-like cryptosystems [2, 20, 23]. While the TFHE scheme allows us to compute any univariate function with programmable bootstrapping, it comes with some drawbacks. Firstly, it cannot bootstrap multiple messages at once since it is designed to operate binary data without SIMD operations. Secondly, the precision afforded by TFHE bootstrapping is limited as the bootstrapping precision $(i.e.,$ plaintext space) linearly grows with respect to the ring degree. Consequently, it is not suitable for functions which demand high precision. Finally, the noise level after the TFHE bootstrapping tends to be quite high. As the parameters are set to be as tight as possible for the sake of optimization in TFHE, there is technically no remaining level (for multiplication) after the bootstrapping.

In this section, we provide a new general functional bootstrapping that can address all the aforementioned issues. To be precise, our new innovative bootstrapping framework supports the SIMD arithmetic, covers a large plaintext space, and the noise introduced during the bootstrapping is small. In Sec. 3.1, we first describe the overall pipeline of the functional bootstrapping. Our new functional bootstrapping takes any RLWE-based ciphertext as input and outputs as the FV ciphertext. In Sec. 3.2 and Sec. 3.3, we describe a detailed process for our method when inputs are FV and CKKS ciphertext, respectively.

3.1 Pipeline

In the following, we describe the overall pipeline of the functional bootstrapping. Note that the polynomial evaluation in the ordinary bootstrapping acts on the coefficients, not the message itself. Therefore, our procedure should follow a similar pipeline to the slim bootstrapping, which consists of four steps as shown in Alg. 1: Slots2Coeffs, ModSwitch, Coeffs2Slots and EvalLUT.

Algorithm 1 Pipeline of Functional Bootstrapping

Input: Ciphertext $ct \in R_{Q_{in}}^2$, Look-Up Table $F : \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$ **Output:** Ciphertext **ct**^{*} ∈ R_Q^2
1: **ct**['] = (c'_0, c'_1) ← Slots2Coeffs(**ct**) ∈ $R_{Q_{in}}^2$ 2: ct $'' \leftarrow (\frac{Q}{p^r} \cdot \left| \frac{p^r}{Q_{ii}} \right|$ $\left. \frac{p^r}{Q_{in}} \cdot c_0' \right], \frac{Q}{p^r} \cdot \left. \left. \frac{p^r}{Q_{in}} \right. \right.$ $\left\{\frac{p^r}{Q_{in}}\cdot c'_1\right\})\in R_Q^2 \quad \text{ //ModSwitch}$ $\mathrm{3}\colon$ ct $'''\leftarrow \mathsf{Coeffs2S}$ lots $(\mathsf{ct}'')\in\bar{R}_Q^2$ 4: ct^* ← EvalLUT(ct''' , F)

Compared to the ordinary bootstrapping, some distinctions arise in two steps, ModSwitch and EvalLUT. Our new method extends the functionality compared to the previous bootstrapping approach from these phases. During the ModSwitch phase, the input ciphertext is scaled so that the ciphetext modulus becomes p^r . Then, we generate a ciphertext with a large ciphertext modulus Q encrypting the 'phase' of the input ciphertext. After this step, we obtain an encryption of 'noisy' messages of which lower bits are rounding error. Therefore, multiple adjacent values in the new message space are associated with a single message. Hence, these multiple values should be converted into the same value for the correct bootstrapping functionality. For example, the existing bootstrapping sends

the multiple values into the original message by removing the noise homomorphically. In our functional bootstrapping pipeline, we aim to evaluate an arbitrary function during the bootstrapping and therefore we need to evaluate some LUT which sends the noisy messages into the evaluation point of the noiseless message.

This LUT evaluation is performed in the EvalLUT step, which generalizes the functionality of DigitExtract. Given an LUT $F: \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$ is evaluated on the input message residing in the commutative ring \mathbb{Z}_{p^r} via homomorphic operations. It is usually a challenging task since only a few functions have polynomial representations in the commutative ring. To overcome this structural difficulty, we adopt a similar approach to the digit extraction algorithm. Roughly speaking, we iteratively evaluate polynomials while performing homomorphic division by p in between.

In this work, we realize the LUT evaluation with a certain family of polynomials. Unlike digit extraction polynomials, these polynomials can 'selectively' remove the LSB. Combining the evaluation of these polynomials with the homomorphic division by p , we can construct a homomorphic selector which iteratively operates on the LSB. A comprehensive analysis of these polynomials and a detailed description of our algorithm for LUT evaluation are provided in Sec. 4.

Let Q_{in} and Q be the ciphertext modulus of input and output, respectively, and p^s be the plaintext modulus of the output ciphertext for some prime p . We also denote $F : \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$ as a LUT corresponding to the function that is evaluated during the functional bootstrapping. Here, $r > s$ is a constant which will be determined in the later subsections. For simplicity, we deliberately choose the parameters so that p^r divides Q . Then each step of our functional bootstrapping can be described as follows.

• Slots2Coeffs(ct): Given an RLWE-based ciphertext $ct \in R^2_{Q_{in}}$, homomorphically move the messages in slots to the coefficients. *i.e.*, Return $ct' \in R^2_{Q_{in}}$, whose coefficients are the messages in slots of the ciphertext ct.

• ModSwitch(ct; p^r): For the input RLWE ciphertext $ct = (c_0, c_1) \in R^2_{Q_{in}}$, return $ct' = (Q/p^r \cdot [p^r/Q_{in} \cdot c_0], Q/p^r \cdot [p^r/Q_{in} \cdot c_1]) \in R_Q^2$. We remark that ct' is essentially an FV encryption whose plaintext is $M(X) = m_0 + m_1 X^d + \cdots$ $m_{k-1}X^{N-d}$ for some $m_0, \ldots, m_{k-1} \in \mathbb{Z}_{p^r}.$

• Coeffs2Slots(ct): Given an FV encryption $ct \in R_Q^2$ of the plaintext $M(X) =$ $m_0 + m_1 X^d + \cdots + m_{k-1} X^{N-d}$, homomorphically move the coefficients to the slots. *i.e.*, Return $ct' \in R_Q^2$, an FV encryption of a vector $(m_0, \ldots, m_{k-1}) \in \mathbb{Z}_{p^r}^k$.

• EvalLUT(ct; F): For the input encryption $ct \in R_Q^2$ of a vector $(m_0, \ldots, m_{k-1}) \in$ $\mathbb{Z}_{p^r}^k$ and an LUT $F: \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$, homomorphically compute F on the ciphertext ct. *i.e.*, Return an FV encryption $ct' \in R_Q^2$ of a vector $(F(m_0), \ldots, F(m_{k-1}))$.

As the inputs of Slots2Coeffs and Coeffs2Slots have different ciphertext modulus Q_{in} and Q , both proceed in a similar way as previous bootstrapping. In particular, they can be performed via a linear transformation and we will not discuss the details of the linear transformation since it is an orthogonal issue to our contribution.

Depending on the chosen parameters, the input ciphertext may contain more or fewer slots than the desired output ciphertext. If the input ciphertext has fewer slots, it is straightforward to pack fewer messages into the output ciphertext. However, if the input ciphertext has more slots than the output, multiple ciphertexts need to be generated from the Coeffs2Slots step, similar to ordinary bootstrapping. We simply assume that the input and output ciphertexts have the same number of slots.

3.2 FV-to-FV Functional Bootstrapping

In this subsection, we discuss functional bootstrapping where the input ciphertext is an FV ciphertext. Recall that our framework takes any RLWE ciphertext as an input, the plaintext modulus of the input ciphertext does not necessarily need to be the same as the plaintext modulus of the output ciphertext. For generality, we designate the input plaintext modulus as t , distinct from the output plaintext modulus p^s .

As mentioned in Sec. 3.1, two key steps of our functional bootstrapping are ModSwitch and EvalLUT, while Slots2Coeffs, Coeffs2Slots can be computed with simple linear transformations. In ModSwitch, the ciphertext modulus of the input is adjusted to a smaller modulus to make the decryption circuit compact. Let $ct = (c_0, c_1) \in R^2_{Q_{in}}$ represent an encryption of a plaintext $\mu(X) \in R_t$ with a ciphertext modulus Q_{in} . Then, $(c'_0, c'_1) = (\lfloor p^r/Q_{in} \cdot c_0 \rfloor, \lfloor p^r/Q_{in} \cdot c_1 \rfloor)$ is a valid encryption of $\mu(X)$ as long as p^r is sufficiently large to accommodate the rounding error. More precisely, it satisfies that $c'_0 + c'_1 \cdot \mathsf{sk} = \lfloor p^r/t \rfloor \cdot \mu + e$ (mod p^r) where $e \in R$ is an error polynomial such that $||e||_{\infty} < p^r/2t$. Therefore, generating an LUT involves mapping input values from \mathbb{Z}_t to \mathbb{Z}_{p^s} based on a specified function $f : \mathbb{Z}_t \to \mathbb{Z}_{p^s}$. For any $m \in \mathbb{Z}_t$, the LUT should output $f(m)$ for the input values $|p^r/t| \cdot m + e$ where $-p^r/2t < e < p^r/2t$. In other words, the LUT has the same values for intervals with length p^r/t . Consequently, we obtain an output LUT with a form of step function regardless of the target function, as shown in Figure 2.

In practice, it is important to choose the parameter r as small as possible to optimize the performance of the bootstrapping. Hence, a thorough investigation of the probabilistic bound of e is required. The noise e comprises two components: e_{init} , the initial noise, and e_{rnd} , the rounding error introduced during the dividing-and-rounding. Analogous to the prior work [11], we assume that the initial noise has at least two bits of noise margin, *i.e.*, $||e_{init}||_{\infty} < p^{r}/4t$. Consequently, it is sufficient to bound the rounding error by $p^r/4t$. The exact form of rounding error is given as $e_{rnd} = e_{rnd}^0 + e_{rnd}^1 \cdot$ sk where

$$
e_{rnd}^i = \frac{p^r}{Q} \cdot c_i - \left\lfloor \frac{p^r}{Q} \cdot c_i \right\rfloor \ (i = 0, 1).
$$

Under the RLWE assumption, the coefficients of c_0 and c_1 are indistinguishable from uniformly sampled random numbers over \mathbb{Z}_Q . In practical scenarios,

Fig. 2: The shape of LUT in functional bootstrapping. The dots denote the function values at each integer point.

it is common to assume that the secret key sk is uniformly sampled from the ternary set $\{-1, 0, 1\}$ with a hamming weight $h := ||\textbf{s}||_1$. Hence, each coefficient of the error polynomial e can be regarded as a sum of $h+1$ uniformly distributed variables over $[-0.5, 0.5]$. In prior works, the worst case bound $h+1$ is leveraged to estimate the rounding error bound [11].

We tighten this bound using a probabilistic bound as proposed in the works on CKKS bootstrapping [6, 42], which establishes that the sum of $h + 1$ uniformly distributed variables follows the Irwin-Hall distribution [42], and it can formly distributed variables follows the frwin-Hall distribution [42], and it can
be bounded by 1.81√h with failure probability less than 2^{-15} [6]. Therefore, in a heuristic approach, we can refine the bound of the size of the coefficients of e_{rnd} as follows:

$$
||e_{rnd}||_{\infty} \le 1.81\sqrt{h}.
$$

Therefore, substituting this back to the error bound $||e_{rnd}||_{\infty} \leq p^{r}/4t$, we can obtain the following bound for the plaintext modulus p^r for LUT evaluation.

$$
p^r > 7.24t\sqrt{h} \tag{1}
$$

Observe that once p and t are settled, r depends solely on the norm of the secret, we want to keep the secret key as sparse as possible. Therefore, we propose to use the sparse key encapsulation technique introduced in [7]. Their method consists of two key-switching keys, where one of them is used to key-switch from the dense key to an ephemeral sparse key at the lowest level, and the other is used to key-switch from the sparse key to the original dense key after ModSwitch. By doing so, we can benefit from the sparsity of the ephemeral sparse key during ModSwitch without compromising security.

Furthermore, we stress that selecting an appropriate value for p^r can facilitate bootstrapping within the FV scheme with a large plaintext modulus. Previous works indicate that the complexity of FV bootstrapping is mainly influenced by the size of the prime factor of the plaintext modulus employed during bootstrapping. Therefore, bootstrapping ciphertexts with a large prime modulus have been considered nearly impractical. However, our novel bootstrapping technique allows for the modification of the plaintext modulus to the power of a smaller plaintext modulus p. This adjustment suggests a potential reduction in the complexity associated with the bootstrapping process.

3.3 CKKS-to-FV Functional Bootstrapping

In the functional bootstrapping from CKKS to FV, Slots2Coeffs and ModSwitch are the only distinguished algorithms compared to the case of FV-to-FV. The reason is that the computation of Slots2Coeffs is carried out approximately unlike FV, and ModSwitch affects the bootstrapping precision instead of the decryption failure.

Assuming that Slots2Coeffs is computed with sufficiently large precision, the overall bootstrapping precision will be dominated by the noise introduced from the scaling and rounding in ModSwitch step. Therefore, we will only focus ModSwitch as in Sec. 3.2 to optimize the parameter r . For simplicity, the input messages are confined in the interval $[-1, 1]$ in the following.

Let $(b, a) \in R^2_Q$ be the CKKS ciphertext after the Slots2Coeffs in our pipeline, with scaling factor Δ . Then, after the modulus switching to p^r , the scaling factor becomes $p^r \cdot \Delta/Q$. On the other hand, the statistical bound of the rounding noise becomes $p' \cdot \Delta/Q$. On the other hand, the statistical bound of the rounding noise introduced by the modulus switching is $1.81\sqrt{h}$ where h denotes the hamming weight of the secret key, as discussed in Sec. 3.2. Thus, we obtain the bit precision after the modulus switching as follows:

$$
\log(p^r \cdot \Delta/Q) - \log(1.81\sqrt{h}).
$$

From the analysis of the bit precision, we propose to leverage the sparse From the analysis of the bit precision, we propose to leverage the sparse
key encapsulation technique [7] to minimize $\log(1.81\sqrt{h})$ term as in Sec. 3.2, and set $\Delta/Q < 1$ as large as possible in order to maximize the bootstrapping precision while keeping r small. By doing so, we obtain a better upper bound of precision while keeping r small. By doing so, we obtain a better upper bound of $r \log(p) - \log(1.81\sqrt{h})$ for the bit precision. Note that this is the opposite situation to the CKKS bootstrapping. In a common setting for bootstrappable CKKS parameters, the ratio between the scaling factor Δ and the modulus Q is set to be small (smaller than 2^{-5}) to achieve a better efficiency in bootstrapping [6, 10]. Instead, our construction requires Δ/Q to be as large as possible, for the sake of the bootstrapping performance.

In CKKS-to-FV functional bootstrapping, the form of the LUT does not necessarily need to be a step function, unlike in the FV-to-FV case. However, it is important to note that in practical scenarios, most of the functions evaluated during bootstrapping tend to have the form of a step function. While it is known that CKKS is generally more efficient when computing continuous functions, it is challenging to evaluate discontinuous functions such as sign function [17, 41] or modular reduction function [43]. Naturally, our functional bootstrapping approach is expected to be particularly beneficial for computing discontinuous functions, as these functions typically yield an LUT with a step-function-like form.

We stress that the CKKS-to-FV functional bootstrapping can be understood as a scheme conversion. Considering that the scheme conversion between SIMD FHE schemes is a task that is almost as hard as the bootstrapping [24], we believe that our construction is asymptotically optimal.

4 Evaluation of Look-Up Table

In this section, we describe how to homomorphically evaluate an arbitrary LUT from \mathbb{Z}_{p^r} to \mathbb{Z}_{p^s} . Before we delve into the construction of our algorithm, we first discuss the homomorphic division by p . Generally, a division by p is not a welldefined operation over \mathbb{Z}_{p^r} since it is not a unit. However, it can be carried out when the message is a multiple of p by simply changing the plaintext modulus to p^{r-1} . The reason is that the message is stored in the most significant bits in the FV cryptosystem. In the later sections, we will abuse the notation of regular division for this operation. *i.e.*, for $x \in \mathbb{Z}_{p^r}$ such that $p | x, x/p$ denotes the ring element $x/p \in \mathbb{Z}_{p^{r-1}}$.

Now, suppose that an arbitrary LUT $F : \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$ is given for a prime p and some positive integers r and s . If there is a polynomial which directly computes F, *i.e.*, a polynomial representation of $p^{r-s} \cdot F(x) \in \mathbb{Z}_{p^r}$, it can be evaluated directly through a single polynomial evaluation. However, only a small number of LUTs can be evaluated in such a way, since most of the functions defined over the commutative ring \mathbb{Z}_{p^r} are not functions with polynomial representations, so-called the *polyfunctions*. For example, even LUT for digit extraction does not have an explicit polynomial representation. Therefore, we utilize the homomorphic division by p between the polynomial evaluations to obtain the desired functionality, similar to the conventional bootstrapping method. By adopting such an approach, we can finally evaluate an arbitrary LUT.

We first provide useful lemmas on the polyfunctions and investigate the structure of the polynomials in $\mathbb{Z}_{p^{\ell}}$ for some positive integer ℓ in Sec. 4.1. These results are exploited to construct an evaluation method for an arbitrary LUT $F: \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$ in the following sections. We first cover the basic case of $s = 1$ and generalize the result to the case $s > 1$ in Sec. 4.2, and Sec. 4.3, respectively. Finally, concrete examples of our functional bootstrapping method are given in Sec. 4.4. We select widely-used functions, the delta and sign functions, as examples. For better readability, we will use the unsigned representation for the integers over the commutative ring in the following subsections.

4.1 Polyfunctions over $\mathbb{Z}_{p^{\ell}}$

We introduce a necessary and sufficient condition for polyfunctions over $\mathbb{Z}_{p^{\ell}}$, established by Guha and Dukkipati [33].

Proposition 1 ([33]). A function f over $\mathbb{Z}_{p^{\ell}}$ is a polyfunction if and only if the function f can be represented with a linear combination of the following functions.

1.
$$
u_0^{\ell}(x) = \begin{cases} 0 & \text{for } p \nmid x \\ x & \text{for } p \mid x \end{cases}
$$

\n2. $u_i^{\ell}(x)$, *i*-th shift of $u_0^{\ell}(x)$, *i.e.*, $u_i^{\ell}(x) = u_0^{\ell}(x - i) \ (0 \le i < p)$
\n3. *j*-th powers of $u_i^{\ell}(x)$, *i.e.*, $(u_i^{\ell}(x))^j = \begin{cases} 0 & \text{for } p \nmid x \\ x^j & \text{for } p \mid x \end{cases} \ (0 \le i < p, 0 \le j < \ell)$

Note that in our notation $u_i^{\ell}(x)$, *i* is an index for shifting while it was the exponent related to the output in [33]. We substitute it as an exponent j described in the third item. We also unify the expression of $u_i(x)$ (in [33]) which is divided into two cases where $i = 0$ and $1 \leq i \leq \ell - 1$ by changing the bound of the exponent j.

The implication of proposition 1 is that if a function f is a polyfunction, then restricting its domain to the congruence class of i modulo p is a polynomial with degree at most ℓ for any $0 \leq i < p$. A simple example is the digit extraction polynomials ${G_i}$ which are commonly used in the state-of-the-art bootstrapping methods for the FV/BGV scheme. In a nutshell, *i*-th digit extraction polynomial G_i is a polynomial which satisfies $G_i(x) = [x]_p \pmod{p^i}$ for any $x \in \mathbb{Z}$. Observe that G_i over \mathbb{Z}_{p^i} is essentially a constant function at any congruence class of j modulo p, since $G_i(j + p \cdot x) = j$ regardless the value of x.

Now let us discuss the properties of the polynomial representation of polyfunctions and how they can be obtained. We first introduce the definition of the Smarandache function.

Definition 1 (Smarandache function). The Smarandache function $\mu(\cdot)$ is defined as $\mu(x) = \min\{i \in \mathbb{N} : x \mid i!\}.$

It is easy to show that $\mu(p^{\ell}) \leq p\ell$ since the number of multiples of p is equal to or more than ℓ in successive p ℓ integers. Therefore, we use p ℓ as the upper bound of $\mu(p^{\ell})$ in the later sections. *i.e.*, $\mu(p^{\ell}) = O(p\ell)$. Interestingly, it is known that any polyfunctions over $\mathbb{Z}_{p^{\ell}}$ can be represented with a polynomial with degree less than $\mu(p^{\ell})$. We state it more formally below in Lem. 1.

Lemma 1 ([37]). If $f : \mathbb{Z}_{p^{\ell}} \to \mathbb{Z}_{p^{\ell}}$ is a polyfunction, there exists a polynomial representation of f with degree smaller than $\mu(p^{\ell})$.

In [27], the authors mentioned an efficient method to find such a 'compact' polynomial representation using the Newton interpolation. It is essentially a direct adoption of the divided difference method, which is a common interpolation technique in numerical analysis.

4.2 LUT Evaluation for Arbitrary Functions

In this subsection, we investigate the details of our new evaluation technique for LUTs of the form $F : \mathbb{Z}_{p^r} \to \mathbb{Z}_p$. In our LUT evaluation algorithm, the basis polynomials u_j^i are utilized. Observe that u_j^i is always a multiple of p, and we can

homomorphically divide the output by p. For the input x, $u_j^i(x)/p$ is $(x - j)/p$ only if $x = j \pmod{p}$ and zero if $x \neq j \pmod{p}$. We can observe that $(x - j)/p$ is essentially 'upper $i - 1$ digits of x', and it can be regarded as a homomorphic extractor of the upper digit based on the last digit, in base-p representation. From this observation, we deduce that $(u_{j_{r-1}}^1)^0 u_{j_{r-2}}^2 (\ldots u_{j_1}^{r-1} (u_{j_0}^r(x)/p) / p \ldots) / p$ is a function which is zero except for $x = \overline{j_{r-1} \dots j_0}$ where $j_{r-1} \neq 0$. (Note that j_{r-1} should not be zero, otherwise it will always return one regardless of the input. To cover the case $j_{r-1} = 0$, we can simply add a constant to the input to make the most significant bit nonzero.) Therefore, an arbitrary LUT can be homomorphically computed by evaluating this polynomial for every possible combination of j_{r-1}, \ldots, j_0 , multiply the LUT value, and add them altogether.

However, this idea suffers from a fatiguing evaluation of exponentially many polynomials which makes functional bootstrapping almost infeasible. In this work, we focus on the construction of an optimized algorithm for a specific type of LUT. More specifically, a step-function like LUT will be considered in the following, since LUT evaluated in most of the scenarios will be a step function as discussed in Sec. 3.2.

Evaluation of Heaviside function We commence by covering the simplest non-trivial step-function, called the Heaviside function. Let the Heaviside LUT $F: \mathbb{Z}_{p^r} \to \mathbb{Z}_p$ is defined as follows,

$$
F(x) = \begin{cases} 0 & \text{if } x < B \\ 1 & \text{otherwise} \end{cases}
$$

for some bound $B \in \mathbb{Z}_{p^r}$. Without loss of generality, we suppose that $B \geq p^{r-1}$ since $F'(x) := 1 - F(x - B)$ is essentially another Heaviside function with bound $p^{r} - B \geq p^{r-1}.$

Now, let $\overline{b_{r-1}b_{r-2} \ldots b_0}$ be the base p representation of B. Observe that if the LSB of the input x is smaller than b_0 , the LUT F returns 0 if and only if its upper $r-1$ bits are less than $b_{r-1} \ldots b_2(b_1 + 1)$ and returns 1 otherwise. On the other hand, if the LSB of the input x is equal or bigger than b_0 , the LUT F returns 0 if and only if its upper $r-1$ bits are less than $b_{r-1} \dots b_2 b_1$, and 1 otherwise. Therefore, we can evaluate the LUT F by dividing it into two sub-LUT's $F_1^{r-1}, F_2^{r-1} : \mathbb{Z}_{p^{r-1}} \to \mathbb{Z}_p$, defined as follows:

$$
F_1^{r-1} = \begin{cases} 0 & \text{if } x < \overline{b_{r-1} \dots (b_1 + 1)} \\ 1 & \text{otherwise} \end{cases},
$$

$$
F_2^{r-1} = \begin{cases} 0 & \text{if } x < \overline{b_{r-1} \dots b_1} \\ 1 & \text{otherwise} \end{cases}.
$$

From this relation, we stress that the LUT F can be computed utilizing the polynomials u_i^r and evaluation of sub-LUTs F_1^{r-1} and F_2^{r-1} . Observe that $u_i^r(x)/p$ outputs the upper $r-1$ bits if the LSB of x is i, and zero otherwise for any input x. Subsequently, for any value x with LSB i, we can obtain $F(x)$ by

evaluating $F_1^{r-1}(u_i^r/p)$ if $i < b_0$ and $F_2^{r-1}(u_i^r/p)$ if $b_0 \leq i$. Recall that we assumed that $B \geq p^{r-1}$, new bounds $\overline{b_{r-1} \ldots (b_1 + 1)}$ and $\overline{b_{r-1} \ldots b_1}$ are strictly bigger than zero, and thus $F_1^{r-1}(0) = F_2^{r-1}(0) = 0$. Hence, the LUT F can be evaluated by computing the sum of $F_1^{r-1}(\bar{u}_i^r(x)/p)$ $(0 \le i < b_0)$ and $F_2^{r-1}(u_i^r(x)/p)$ $(b_0 \le j < b_0)$ $i < p$). Since we essentially evaluate the two identical LUTs for the cases $0 \leq i <$ b_0 and $b_0 \leq i < p$ respectively, the sum of LUTs can be integrated as follows.

$$
F(x) = F_1^{r-1} \left(\sum_{0 \le i < b_0} u_i^r(x)/p \right) + F_2^{r-1} \left(\sum_{b_0 \le i < p} u_i^r(x)/p \right)
$$

Thus, it remains to evaluate two sub-LUTs F_1^{r-1} and F_2^{r-1} . We remark that F_1^{r-1} and F_2^{r-1} are again Heaviside function over $\mathbb{Z}_{p^{r-1}}$ with bounds $B_1 :=$ $b_{r-1} \ldots b_2(b_1 + 1)$ and $B_2 := b_{r-1} \ldots b_2 b_1$. Hence, they can also be divided into sub-LUTs analogously. In a similar way that F_1^{r-1} and F_2^{r-1} do not contain any information on the LSB b_0 in them, the sub-LUTs of F_1^{r-1} and F_2^{r-1} are also independent from the LSB $b_1 + 1$ and b_1 , of the bounds B_1 and B_2 , respectively. As B_1 and B_2 only differ by the LSB, the sub-LUTs of F_1^{r-1} and F_2^{r-1} should be identical. Let us denote them by F_1^{r-2} and F_2^{r-2} . Then, they are defined as follows:

$$
F_1^{r-2} = \begin{cases} 0 & \text{if } x < b_{r-1} \dots (b_2 + 1) \\ 1 & \text{otherwise} \end{cases}
$$

$$
F_2^{r-2} = \begin{cases} 0 & \text{if } x < b_{r-1} \dots b_2 \\ 1 & \text{otherwise} \end{cases}.
$$

Analogous to the evaluation of F, F_1^{r-1} and F_2^{r-1} can be evaluated in a recursive manner. For a better readability, let us denote by $x_1 := \sum_{0 \leq i < b_0} u_i^r(x)/p$ and $x_2 := \sum_{b_0 \leq i < p} u_i^r(x)/p$. Then, it follows that

$$
F(x) = F_1^{r-2} \left(\sum_{0 \le i \le b_1} u_i^{r-1}(x_1)/p + \sum_{0 \le i < b_1} u_i^{r-1}(x_2)/p \right) + F_2^{r-2} \left(\sum_{b_1 < i < p} u_i^{r-1}(x_1)/p + \sum_{b_1 \le i < p} u_i^{r-1}(x_2)/p \right)
$$

since $F_1^{r-2}(0) = F_2^{r-2}(0) = 0$ due to the condition $B \ge p^{r-1}$.

Note that the inputs for the same LUTs are integrated and hence it only requires the evaluation of four polynomials $\sum_{0 \le i \le b_1} u_i^{r-1}(x_1)/p$, $\sum_{0 \le i < b_1} u_i^{r-1}(x_2)/p$, $\sum_{b_1 \leq i \leq p} u_i^{r-1}(x_1)/p$ and $\sum_{b_1 \leq i \leq p} u_i^{r-1}(x_2)/p$, and LUTs F_1^{r-2} and F_2^{r-2} over $\mathbb{Z}_{p^{r-2}}$. These LUTs F_1^{r-2} and $\overline{F_2^{r-2}}$ can be iteratively computed via four polynomial evaluations and two LUT evaluations in the smaller dimension, in a similar manner. (Note that the condition $B \geq p^{r-1}$ plays an important role here, making the value of the LUT zero for the zero input.) Subsequently, at the end of

the iteration, it remains to evaluate two LUTs F_1^1 and F_2^1 over \mathbb{Z}_p , defined as follows.

$$
F_1^1 = \begin{cases} 0 & \text{if } x < b_{r-1} + 1 \\ 1 & \text{otherwise} \end{cases}
$$
\n
$$
F_2^1 = \begin{cases} 0 & \text{if } x < b_{r-1} \\ 1 & \text{otherwise} \end{cases}.
$$

We stress that any function over \mathbb{Z}_p is a polyfunction, they have polynomial representations with integer coefficients and each LUT can be evaluated with one polynomial evaluation. From these recurrence relations, we can devise an algorithm for Heaviside function evaluation. The exact algorithm based on this approach is presented in Alg. 2.

Algorithm 2 Polynomial evaluation for Heaviside function

Input: An input $x \in \mathbb{Z}_{p^r}$, LUT F of a Heaviside function with bound B. **Output:** $F(x) \in \mathbb{Z}_p$ 1: parse $B = b_{r-1}b_{r-2} \ldots b_0$ in digit p representation. 2: $x_1, x_2 \leftarrow 0 \in \mathbb{Z}_{p^r}, x \in \mathbb{Z}_{p^r}$ 3: for $i = 0; i < r - 1; i+=1$ do 4: $x_1 \leftarrow \sum_{0 \leq j \leq b_i} u_j^{r-i}(x_1) + \sum_{0 \leq j < b_i} u_j^{r-i}(x_2) \pmod{p^{r-i}}$ 5: $x_2 \leftarrow \sum_{b_i < j < p} u_j^{r-i}(x_1) + \sum_{b_i \le j < p} u_j^{r-i}(x_2) \pmod{p^{r-i}}$ 6: $x_1, x_2 \leftarrow x_1/p \in \mathbb{Z}_{p^{r-i-1}}, x_2/p \in \mathbb{Z}_{p^{r-i-1}}$ 7: end for 8: Return $F_1^1(x_1) + F_2^1(x_2) \in \mathbb{Z}_p$

Depth and Time complexity Analysis We analyze the depth consumption and the time complexity of our method. For the time complexity, we consider the number of multiplication (key-switching) which takes a large portion of the computations time in the functional bootstrapping. We remark that they are solely dependent on the degree of the polynomials utilized during the evaluation. It is known that any polynomial with degree d can be evaluated with $2\n\sqrt{d}$ non-scalar multiplications while consuming $\lceil \log d \rceil$ levels, applying the Paterson-Stockmeyer algorithm [30, 51]. Based on this analysis, we conduct the time complexity and the depth consumption analysis for our method.

In the *i*-th iteration, four polynomials over $\mathbb{Z}_{p^{r-i}}$ are evaluated. As every polynomial over $\mathbb{Z}_{p^{r-i}}$ has degree at most $\mu(p^{r-i}) \approx p(r-i)$ by Lemma 1, their evaluation requires $\approx 4 \cdot 2\sqrt{p(r-i)} = 8\sqrt{p(r-i)}$ key-switching operations and $\approx \log(pr)$ levels of depth consumption. Hence, during r iterations of the algorithm, we perform $\sum_{i=0}^{r-1} 8\sqrt{p(r-i)} \approx \frac{16}{3}\sqrt{r^3p}$ key-switching operations and consume $\sum_{i=0}^{r-1} \log(p(r-i)) = r \log p + \log(r!)$ multiplicative depths.

Optimization Our method can be optimized under certain cases. If $b_i = 0$ for some $0 \leq i < r$, the polynomial $\sum_{0 \leq j < b_i} u_j^{r-i}(x_1)$ is essentially zero and thus its evaluation can be skipped. Moreover, if the bound of the Heaviside function B has consecutive ℓ zeros in the LSB, *i.e.*, $b_0 = b_1 = \cdots = b_{\ell-1} = 0$, x_1 is essentially zero for ℓ iterations. In this case, only one polynomial evaluation and two polynomial evaluations are necessitated at the beginning and in each iteration of the algorithm, respectively.

We also remark that there exists a trade-off between time and depth complexity for our LUT evaluation. Our method essentially compares each digit of the bound and the input, it can be realized as a multivariate function over \mathbb{Z}_p with digits of the input message as variables. Note that each digit can be obtained while consuming $i \cdot \log p$ during the digit extraction algorithm of the conventional FV bootstrapping, the multivariate polynomial can be evaluated with output depth $(r + 1) \log p$. This depth asymptotically improves the depth consumption of the aforementioned method, while its asymptotic time complexity is increased.

Evaluation of arbitrary LUT The evaluation of the Heaviside function can be naturally extended to the case of an arbitrary LUT. Naïvely, we can decompose a LUT with k intervals into a linear combination of k Heaviside functions. In particular, let the LUT with k intervals is defined as follows:

$$
F(x) = \begin{cases} \alpha_1 & \text{if } x < B_1 \\ \alpha_2 & \text{if } B_1 \le x < B_2 \\ \dots & \\ \alpha_{k-1} & \text{if } B_{k-2} \le x < B_{k-1} \\ \alpha_k & \text{otherwise} \end{cases}
$$

Then, we can represent it as $F(x) = \alpha_1 + \sum_{i=1}^{k-1} (\alpha_{i+1} - \alpha_i) F_i(x)$ such that Heaviside function $F_i(x)$ is defined as follows:

$$
F_i(x) = \begin{cases} 0 & \text{if } x < B_i \\ 1 & \text{otherwise} \end{cases} \quad \text{for } 1 \le i < k
$$

Hence, we have the capability to evaluate an arbitrary step function during functional bootstrapping. However, employing naïve approach entails performing k time-consuming evaluation of k Heaviside functions. We remark that the time complexity can be mitigated by constructing a recurrence relation akin to the Heaviside function case. Specifically, we categorize the cases based on the LSB of the given bounds B_1, \ldots, B_{k-1} . Nonetheless, this approach involves an exhaustive classification of edge cases, given the existence of $k \cdot (k-1)! = k!$ possible orderings of the LSB of the input value and $k - 1$ bounds. While it reduces the number of polynomials evaluated throughout the LUT evaluation, this classification should be considered carefully. Therefore, for better scalability, it is advisable to use the naïve method except for certain use-cases demanding optimization.

4.3 Generalization

In this section, we establish an evaluation method for an arbitrary LUT F : $\mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$. This can be more challenging than the basic case of $s = 1$, which is covered in Sec. 4.2. In the basic case, the existence of a polynomial representation of arbitrary function over the finite field \mathbb{Z}_p played a crucial role. On the contrary, only a small number of functions over the commutative ring \mathbb{Z}_{p^s} have a polynomial representation. Due to this reason, our method does not naturally extend to the prime power case. We resolve this problem by introducing the homomorphic lifting to the LUT evaluation, *i.e.*, homomorphically computing $x \in \mathbb{Z}_{p^i}$ for some $i > 1$ from $x \in \mathbb{Z}_p$. Roughly speaking, we evaluate all s digits of the given LUT $F: \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$ utilizing our basic LUT evaluation algorithm and use the homomorphic lifting to merge the results to obtain the evaluation of F with a simple linear combination.

In detail, for an arbitrary LUT $F : \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^s}$, we decompose F into s sub-LUTs $F_i: \mathbb{Z}_{p^r} \to \mathbb{Z}_p$ by taking the *i*-th digit of the result of the result of F, *i.e.*, $F_i(x) := [[F(x)/p^i]]_p \in \mathbb{Z}_p$. Such sub-LUTs $\{F_i\}_{0 \leq i < s}$ can be evaluated simultaneously leveraging our LUT evaluation algorithm, and consequently s FV encryptions of $F_i(\mathbf{m}) := (F_i(m_1), \ldots, F_i(m_k))$ $(0 \leq i \leq s)$ can be obtained where $\mathbf{m} = (m_1, \ldots, m_k) \in \mathbb{Z}_{p^r}^k$ is the input message. Then, the encryption of $F(\mathbf{m})$ can be computed from these encryptions of $F_i(\mathbf{m})$ $(0 \leq i \leq s)$ using the following relation:

$$
F(x) = [F_0(x)]_{p^s} + p[F_1(x)]_{p^s} + \dots + p^{s-1}[F_{s-1}(x)]_{p^s} = \sum_{i=0}^{s-1} p^i [F_i(x)]_{p^s} \pmod{p^s}.
$$

To evaluate $F(x)$ using this relation, a homomorphic evaluation of lifting function $[\cdot]_{p^s}: \mathbb{Z}_p \to \mathbb{Z}_{p^s}$ is necessitated. This operation is nontrivial in the context of FV, however, we stress that the lifting function can be homomorphically computed via the modulus switching operation and polynomial evaluation. Suppose that we are given an FV encryption $(b, a) \in R_Q^2$ of $\mu(X) := \mu_0 + \mu_1 X^d +$ $\mu_2 X^{2d} + \cdots + \mu_{k-1} X^{N-d} \in R_p$, which is an encoding of $\mathbf{m} = (m_1, \ldots, m_k) \in \mathbb{Z}_p^k$. Firstly, we compute $(b', a') = (\lfloor b/p^{s-1} \rfloor, \lfloor a/p^{s-1} \rfloor) \in R_Q^2$, and change the plaintext modulus to p^s . Then, it is easy to show that (b', a') is an encryption of $\mu + pI \in \mathbb{Z}_{p^s}$ for some random polynomial $I(X) \in R_{p^{s-1}}$. Since all the coefficients of I can be nonzero, the (partial) trace map can be evaluated to remove the redundant coefficients of I analogous to the bootstrapping. As a result, an encryption of $\mu + p\tilde{I} \in \mathbb{Z}_{p^s}$ is obtained, where *i*-th slot's value is $m_i + pI_i \in \mathbb{Z}_{p^s}$ for some I_i . Now, by homomorphically evaluating the s-th digit extraction polynomial G_s , we can obtain an encryption of $\tilde{\mu} \in R_{p^s}$, an encoding of the lifting $\tilde{\mathbf{m}} = (m_1, \dots, m_k) \in \mathbb{Z}_{p^s}^k$.

With this homomorphic lifting technique, we can finally evaluate the LUT F. Let us recall the relation between the sub-LUTs F_i and the original LUT F. As explained above, $F(x) = \sum_{i=0}^{s-1} p^i [F_i(x)]_{p^s}$ and thus s times of homomorphic lifting is required. Naïvely, we can lift each $F_i(\mathbf{m})$ into \mathbb{Z}_{p^s} and compute the

linear combination. However, observe that $p^{i}[x]_{p^{s}} = [x]_{p^{s-i}}$ for any $x \in \mathbb{Z}_{p^{s-i}}$ in the context of FV cryptosystem, the relation can be re-written as follows:

$$
F(x) = \sum_{i=0}^{s-1} p^i [F_i(x)]_{p^s} = \sum_{i=0}^{s-1} [F_i(x)]_{p^{s-i}}.
$$

Hence, it is sufficient to perform the homomorphic lifting of $[\cdot]_{p^{s-i}}$ for each F_i $(0 \leq i \leq s)$ in total. By doing this, we can save a few relinearizations while the depth consumption remains the same.

Time and Space Complexity Now, let us analyze the time complexity and the depth consumption of this general LUT evaluation. Analogous to the basic case of $s = 1$, we will only consider the step function case. The computation of F_i ($0 \leq i \leq s$) requires $\frac{16}{3}\sqrt{r^3p}$ key-switching operations, and consumes $r \log p + \log(s!)$ depths as discussed in Sec.4.2. In the following homomorphic lifting, a series of digit extraction polynomials G_1, \ldots, G_s are evaluated. Since the degree of *i*-th digit extraction polynomial is $(p-1)(i-1)$, its computation requires $\approx 2\sqrt{(p-1)(i-1)} \approx 2\sqrt{pi}$ key-switching and log(*pi*) depths. To sum up it all, $2\sqrt{p}+2\sqrt{2p}+\cdots+2\sqrt{ps} \approx \frac{2}{3}\sqrt{s^3p}$ key-switching operations are performed, and $log((p-1)(s-1)) \approx log p + log s$ multiplicative depths are needed. Therefore, this general LUT evaluation method requires $\frac{2\sqrt{p}}{3}$ e depuis a $\frac{\sqrt{p}}{3}(8\sqrt{r^3} +$ √ s ³) relinearizations and consumes $(r + 1) \log p + \log(s!)$ depths in total.

4.4 Concrete Examples: Delta & Sign Functions

In this section, we apply our functional bootstrapping technique to some selected functions, namely delta and sign function. The delta function is a special function that returns 1 when the input is zero and 0 for other cases. To put it in a functional form with discrete input x ,

$$
Delta(x) = \begin{cases} 1 & \text{if } x = 0\\ 0 & \text{otherwise} \end{cases}
$$

After modulus switching from p^r to Q in the modulus switching phase, the ciphertext can be regarded as an encryption of $|p^r/t| \cdot m + e$ where $-p^r/2t \leq$ $e < p^r/2t$, as discussed in the beginning of this section. Therefore, the evaluated LUT during the functional bootstrapping of the delta function should be

$$
F(x) = \begin{cases} 0 & \text{if } x < -p^r/2t \\ 1 & \text{if } -p^r/2t \le x < p^r/2t \\ 0 & \text{otherwise.} \end{cases}
$$

Even though the function F seems like a LUT with multiple steps, we can convert it into a step function case by shifting a domain. Since the message space of FV is $(-t/2, t/2]$ for the plaintext modulus t, the delta function is converted to the following function when we substitute the input x to $x - \lfloor t/2 \rfloor$:

$$
Delta(x) = \begin{cases} 0 & \text{if } x < t - 1 \\ 1 & \text{otherwise} \end{cases}
$$

Then, the LUT F is also changed as follows:

$$
F(x) = \begin{cases} 0 & \text{if } x < p^r - p^r / 2t \\ 1 & \text{otherwise} \end{cases}
$$

As evaluating a functional bootstrapping with a LUT with three intervals takes more complexity than a step function, we can reduce the complexity by transforming the delta function as we mentioned above. We can exploit the delta function to extract items with the same attribute as the target value. We will describe more detailed scenarios in the next section.

Note that the LUT may have different intervals depending on the precision when the input is the CKKS ciphertext. As mentioned in the Sec. 3.3, the scaling factor after ModSwitch becomes $p^r \cdot \Delta/Q$. Therefore, the delta function can be represented as follows for continuous input x :

$$
Delta(x) = \begin{cases} 0 & \text{if } -\frac{Q}{p^r \cdot \Delta} < x < \frac{Q}{p^r \cdot \Delta} \\ 1 & \text{otherwise} \end{cases}
$$

While evaluating an LUT about this delta function also requires high computational cost because it has three intervals, we can decrease the number of intervals for performance improvement by shifting the input data, similar to the aforementioned example.

Another useful function is a sign function which returns -1 when the input is negative and 1 for other cases. It also has various ways to utilize this function. For example, The sign function is the main building block of comparison which is a sign value of subtraction of two inputs and the comparison is a valuable function used in several applications such as decision trees, sorting algorithms, or SQL queries in the privacy-preserving database.

The sign function is a typical step function that can be written as follows:

$$
Sign(x) = \begin{cases} -1 & \text{if } x < 0\\ 1 & \text{otherwise} \end{cases}
$$

Similar to the delta function, the LUT F from \mathbb{Z}_{p^r} to \mathbb{Z}_{p^s} is as following:

$$
F(x) = \begin{cases} -1 & \text{if } x < -p^r/2t \\ 1 & \text{otherwise} \end{cases}
$$

As both LUTs have the same form as the Heaviside function, we can easily compute them during the functional bootstrapping according to the Algorithm 2.

Note that the depth and time complexity are asymptotically the same with analysis in Section 4.2 because the evaluation process is identical to the Algorithm 2. As a result, it requires $\sum_{i=0}^{r-1} 8\sqrt{p(r-i)} \approx \frac{16}{3}\sqrt{r^3p}$ key-switching operations and consumes $\sum_{i=0}^{r-1} \log(p(r-i)) = r \log p + \log(r!)$ depths. In the following section, concrete performances of these two functions are provided in the next section with an appropriate parameter set.

5 Experiments & Applications

Employing the functional bootstrapping algorithm, specifically tailored for the delta and sign functions, we present a proof-of-concept level implementation within the context of the FV scheme. We apply functional bootstrapping to address specific functions within practical applications, notably in the domains of circuit Private Set Intersection (circuit PSI), and tree based classification model. Remark that the delta and sign functions described in Section 4 have extensive utility in real-world scenarios.

The following subsections present a comprehensive benchmark analysis of the performance of functional bootstrapping and introduce its efficiency in addressing the specified functions within the FV scheme. Additionally, we present practical use cases and demonstrate the utilization of functional bootstrapping.

5.1 Implementation

We have conducted a proof-of-concept implementation of our functional bootstrapping approach for the delta and sign functions, transitioning from FV with a plaintext modulus of t to FV with a plaintext modulus of p^r . The implementation was carried out using Lattigo v5 [48]. All experiments were performed on a machine with Intel(R) Xeon(R) Platinum 8268 @ 2.90GHz CPU and 192GB RAM running Ubuntu 20.04.2 LTS.

We only present the experimental results for FV-to-FV functional bootstrapping, since the only difference between FV-to-FV case and CKKS-to-FV case is Slots2Coeffs, which is essentially identical in both cases except for the encoding process. We utilized two ring dimensions $N = 2^{15}$ and $N = 2^{16}$, which are commonly used for circuit evaluation with a sufficiently large depth such as bootstrapping. Each parameter set (t, p, r, h) satisfies the condition of equation 1 to guarantee the low bootstrapping failure probability and achieves an estimated security level of \geq 128-bits, where h denotes the Hamming weight of the secret. The parameters that are used in the implementation are summarized in Table. 1.

Additionally, in the implementation, we apply a special modulus method [31], a well-known optimization technique in FHE, to mitigate the noise growth from homomorphic operations. In this variant, public keys are generated in R_{QP} for an integer P known as the special modulus. During multiplication, the computation result is scaled down by P to recover the ciphertext modulus Q while decreasing the noise growth.

	\mathbf{r}		$\,n$	$\log PQ$
< 700		$ 17 4 256 2^4 2^{15}$		840
≤ 12000 17 5 256 2 ⁴ 2 ¹⁶				1700

Table 1: Parameter set for the implementation. t, p and n denote the input and output plaintext modulus and the number of slots, respectively.

We remark that the number of slots may not be optimal in our implementation. In the context of FV/BGV style cryptosystems, non-power-of-two cyclotomic rings are commonly used as the base ring in order to use more slots. However, since the Lattigo library only supports the power-of-two cyclotomic ring, it was unable to provide a benchmark result with a large number of slots. It is worth emphasizing that the number of slots only affects the linear transform part, not the polynomial evaluation part. We expect that our implementation can be further optimized, utilizing well-known optimization techniques for the maximum slots. In [4], the authors showed that one can encode more messages than the number of the factoring polynomials, by interpolating the messages over the field. Note that this approach requires 're-encoding' after several multiplications. On the other hand, Arita and Handa [3] proposed the usage of the subring of a prime degree cyclotomic polynomial in order to encode the same number of the data to the ring dimension.

The limitations imposed by the choice of plaintext modulus in Lattigo, which must be a power-of-two NTT-friendly prime number of moderate size, can indeed affect the behavior of existing baseline algorithms. To address this, we implemented several (unoptimized) algorithms such as the Paterson-Stockmeyer polynomial evaluation algorithm, matrix multiplication, encoding and decoding. Due to such a reason, we believe that there is still room for further optimization in our code. An illustrative example is the application of the lazy polynomial Baby-Step Giant-Step (BSGS) algorithm [43], which holds the promise of minimizing the relinearization operations during polynomial evaluation. In addition, we recognize the opportunity to enhance the time complexity of polynomial evaluation from a sublinear to a logarithmic scale through the utilization of the polynomial evaluation technique proposed by Okada et al. [50]. We expect that these potential optimizations will refine the efficiency and overall performance of our code.

5.2 Benchmarks

We present experimental results derived from our implementation of the functional bootstrapping method for the FV scheme. The execution time associated with functional bootstrapping is presented in Table 2. As described in Section 4.4, the LUTs employed for the computation of delta and sign functions exhibit identical characteristics, sharing the same number of intervals and ranges within those intervals. The only difference is the output values of each interval. Consequently, the evaluation time for the functional bootstrapping with both functions remains consistent. During the measurement of elapsed time, we constrained the dataset size to ensure that the data could be efficiently packed within a single ciphertext. Note that the initial plaintext modulus t , correlated with the number of messages packable in the ciphertext, does not affect the execution time within the scope of our experimental setup.

ID Slots2Coeffs ModSwitch Coeffs2Slots EvalLUT Total				
0.03s	2.5s	0.6s	44.2s	47.3s
0.2s	4.0s	$1.5\mathrm{s}$	166.5s	172.1s

Table 2: Evaluation time of each step of the functional bootstrapping for delta/sign function

ID Noise Consumption Remaining Levels			
\approx 595 bits	\approx 11 levels		
≈ 840 bits	≈ 20 levels		

Table 3: Noise consumption of functional bootstrapping and remaining level after functional bootstrapping for delta/sign function

Table 3 provides insights into the noise consumption and the resulting remaining noise level after the functional bootstrapping process. The ring dimension Nplays a crucial role in determining the magnitude of noise generated during homomorphic evaluation. In our experiments, we observe that parameter sets with a larger ring dimension, such as $N = 2^{16}$, tend to result in higher noise consumption during the bootstrapping process. This is attributed to the increased computational complexity associated with larger ring dimensions. Furthermore, we calculate the remaining level in scenarios where the plaintext modulus q is smaller than t . In such cases, the remaining noise level is higher in comparison to situations where the modulus remains t. This difference arises since homomorphic operations require less noise consumption with a smaller plaintext modulus.

Note that the elapsed times of homomorphic DFT and iDFT, and modulus switching are almost negligible in our experiment. This results from the small number of plaintext slots due to the property of the power-of-two cyclotomic ring structure. Still, the bootstrapping time is heavily dependent on the LUT evaluation time which accounts for about 93 − 96% of the end-to-end functional bootstrapping process.

5.3 Applications

The application of functional bootstrapping with delta or sign functions holds significant potential in various real-world scenarios. One prominent example is circuit PSI which is a generalization of the plain PSI that evaluates a function over the intersection $X \cap Y$ of private sets X and Y and outputs the evaluation result in a secret shared form. In [54], the authors realizes the circuit PSI using the FV scheme. During the protocol, it applies two-party computation after FHE-based plain PSI protocol [12, 13, 22] for the sake of function evaluation, sacrificing additional communication cost. It is to avoid a cumbersome large depth polynomial evaluation with FHE, which is almost infeasible due to a large plaintext modulus. However, we stress that our functional bootstrapping can be applied to address such a problem and consequently optimizes the communication cost.

A tree-based machine learning model can also benefit from our functional bootstrapping, e.g., decision tree, random forest. Many studies [21, 26] utilizes TFHE scheme since the decision tree model requires computation of nonpolynomial operations such as comparison. However, since TFHE scheme does not support SIMD operations for multiple input data, it provides low throughput compared to the decision tree model constructed by FV or CKKS scheme [1, 53], which causes high computational costs. We also stress that our functional bootstrapping can reduce this cost since it can evaluate any functions with a single bootstrapping in a SIMD manner.

Moreover, the inputs of the decision tree or other classification models are usually fixed-point numbers. Therefore, encrypting these features using the CKKS encryption scheme appears to be a more intuitive choice. However, the outputs of these models are often classification or regression results, which are discrete data. Therefore, the result my contain noise since CKKS additional noise is introduced during the computation. We emphasize that our functional bootstrapping can prevent it with scheme conversion from CKKS to FV for noiseless result.

6 Conclusion & Future Works

In this paper, we introduce a new bootstrapping method called functional bootstrapping which generalizes the existing bootstrapping techniques for the FV scheme. It extends the functionality of bootstrapping by allowing us to bootstrap any RLWE-based ciphertexts while evaluating an arbitrary function. As a result, we can instantiate the scheme conversion and plaintext change with no additional cost. As a result, the consumed depth and the computational complexity of a large-depth circuit can be mitigated asymptotically. It also gives more flexibility in the parameter selection by allowing us the plaintext modulus conversion.

To realize this, a new algorithm to evaluate the arbitrary LUTs from the commutative ring \mathbb{Z}_{p^r} to \mathbb{Z}_{p^s} is developed. We first propose an optimized LUT evaluation algorithm for a Heaviside function from \mathbb{Z}_{p^r} to \mathbb{Z}_p and generalize it to an arbitrary LUT over \mathbb{Z}_{p^r} to \mathbb{Z}_{p^s} by introducing the homomorphic lifting operation. In addition, we demonstrate its application in handling special functions like delta and sign functions. We implement this approach using the open-source FHE library, Lattigo [48], and provide benchmark analyses.

We note that our method can also be applied to the bootstrapping for the BGV cryptosystem. We expect that the functional bootstrapping for BGV ciphertext will be asymptotically worse than the functional bootstrapping for FV ciphertext due to the rounding error bound and limited choice of t and p similar to the discussion given in [11]. We believe that optimizing our functional bootstrapping framework for these cases can be an interesting research topic in the future.

References

- 1. Akhavan Mahdavi, R., Ni, H., Linkov, D., Kerschbaum, F.: Level up: Private noninteractive decision tree evaluation using levelled homomorphic encryption. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 2945–2958 (2023)
- 2. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34. pp. 297–314. Springer (2014)
- 3. Arita, S., Handa, S.: Subring homomorphic encryption. In: International Conference on Information Security and Cryptology. pp. 112–136. Springer (2017)
- 4. Aung, K.M.M., Lim, E., Sim, J.J., Tan, B.H.M., Wang, H., Yeo, S.L.: Field instruction multiple data. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 611–641. Springer (2022)
- 5. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full rns variant of fv like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer (2016)
- 6. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 587–617. Springer (2021)
- 7. Bossuat, J.P., Troncoso-Pastoriza, J., Hubaux, J.P.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In: International Conference on Applied Cryptography and Network Security. pp. 521–541. Springer (2022)
- 8. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
- 9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) 6(3), 1–36 (2014)
- 10. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 34–54. Springer (2019)
- 11. Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 315–337. Springer (2018)
- 12. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1223–1237 (2018)
- 13. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255 (2017)
- 14. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37. pp. 360– 384. Springer (2018)
- 15. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full rns variant of approximate homomorphic encryption. In: International Conference on Selected Areas in Cryptography. pp. 347–368. Springer (2018)
- 16. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
- 17. Cheon, J.H., Kim, D., Kim, D., Lee, H.H., Lee, K.: Numerical method for comparison on homomorphically encrypted numbers. In: International conference on the theory and application of cryptology and information security. pp. 415–445. Springer (2019)
- 18. Cheon, J.H., Kim, M., Kim, M.: Optimized search-and-compute circuits and their application to query evaluation on encrypted data. IEEE Transactions on Information Forensics and Security $11(1)$, 188–199 (2015)
- 19. Cheon, J.H., Kim, M., Kim, M.: Search-and-compute on encrypted data. In: Financial Cryptography and Data Security: FC 2015 International Workshops, BIT-COIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers. pp. 142–159. Springer (2015)
- 20. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: international conference on the theory and application of cryptology and information security. pp. 3–33. Springer (2016)
- 21. Cong, K., Das, D., Park, J., Pereira, H.V.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 563–577 (2022)
- 22. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled psi from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 1135–1150 (2021)
- 23. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 617–640. Springer (2015)
- 24. Eldefrawy, K., Genise, N., Manohar, N.: On the hardness of scheme-switching between simd fhe schemes. In: International Conference on Post-Quantum Cryptography. pp. 196–224. Springer (2023)
- 25. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. 2012, 144 (2012)
- 26. Frery, J., Stoian, A., Bredehoft, R., Montero, L., Kherfallah, C., Chevallier-Mames, B., Meyre, A.: Privacy-preserving tree-based inference with fully homomorphic encryption. Cryptology ePrint Archive (2023)
- 27. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo pe and faster bootstrapping for homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 257–286. Springer (2023)
- 28. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
- 29. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
- 30. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 129–148. Springer (2011)
- 31. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
- 32. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92. Springer (2013)
- 33. Guha, A., Dukkipati, A.: An algorithmic characterization of polynomial functions over F_{p^n} . Algorithmica 71(1), 201–218 (Jun 2013). https://doi.org/10.1007/s00453-013-9799-7, http://dx.doi.org/10.1007/ s00453-013-9799-7
- 34. Halevi, S., Polyakov, Y., Shoup, V.: An improved rns variant of the bfv homomorphic encryption scheme. In: Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings. pp. 83–105. Springer (2019)
- 35. Halevi, S., Shoup, V.: Bootstrapping for helib. Journal of Cryptology 34(1), 7 (2021)
- 36. Iliashenko, I., Zucca, V.: Faster homomorphic comparison operations for bgv and bfv. Proceedings on Privacy Enhancing Technologies 2021(3), 246–264 (2021)
- 37. Keller, G., Olson, F.R.: Counting polynomial functions (mod pn) (1968)
- 38. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General bootstrapping approach for rlwe-based homomorphic encryption. IEEE Transactions on Computers (2023)
- 39. Kim, J., Seo, J., Song, Y.: Simpler and faster bfv bootstrapping for arbitrary plaintext modulus from ckks. Cryptology ePrint Archive (2024)
- 40. Kim, M., Lee, H.T., Ling, S., Wang, H.: On the efficiency of fhe-based private queries. IEEE Transactions on Dependable and Secure Computing 15(2), 357–363 (2016)
- 41. Lee, E., Lee, J.W., No, J.S., Kim, Y.S.: Minimax approximation of sign function by composite polynomial for homomorphic comparison. IEEE Transactions on Dependable and Secure Computing 19(6), 3711–3727 (2021)
- 42. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40. pp. 618–647. Springer (2021)
- 43. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimiza-

tion. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 551–580. Springer (2022)

- 44. Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7ms, with $O(1)$ polynomial multiplications. Cryptology ePrint Archive, Paper 2023/910 (2023), https://eprint.iacr.org/2023/910, https://eprint.iacr.org/2023/910
- 45. Liu, Z., Wang, Y.: Relaxed functional bootstrapping: A new perspective on bgv/bfv bootstrapping. Cryptology ePrint Archive, Paper 2024/172 (2024), https: //eprint.iacr.org/2024/172, https://eprint.iacr.org/2024/172
- 46. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) $60(6)$, 1–35 (2013)
- 47. Ma, S., Huang, T., Wang, A., Wang, X.: Accelerating bgv bootstrapping for large p using null polynomials over \mathbb{Z}_{p^e} . Cryptology ePrint Archive (2024)
- 48. Mouchet, C.V., Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Lattigo: A multiparty homomorphic encryption library in go. In: Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography. pp. 64–70. No. CONF (2020)
- 49. Narumanchi, H., Goyal, D., Emmadi, N., Gauravaram, P.: Performance analysis of sorting of fhe data: integer-wise comparison vs bit-wise comparison. In: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA). pp. 902–908. IEEE (2017)
- 50. Okada, H., Player, R., Pohmann, S.: Homomorphic polynomial evaluation using galois structure and applications to bfv bootstrapping. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 69– 100. Springer (2023)
- 51. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing $2(1)$, 60–66 (1973)
- 52. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) $56(6)$, 1–40 (2009)
- 53. Shin, H., Choi, J., Lee, D., Kim, K., Lee, Y.: Fully homomorphic training and inference on binary decision tree and random forest. Cryptology ePrint Archive (2024)
- 54. Son, Y., Jeong, J.: Psi with computation or circuit-psi for unbalanced sets from homomorphic encryption. In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security. pp. 342–356 (2023)
- 55. Tan, B.H.M., Lee, H.T., Wang, H., Ren, S., Aung, K.M.M.: Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. IEEE Transactions on Dependable and Secure Computing 18(6), 2861– 2874 (2020)