General Functional Bootstrapping using CKKS*

Andreea Alexandru[1], Andrey Kim[2] and Yuriy Polyakov[1]

[1] Duality Technologies, USA
[2] Altbridge, USA

Abstract. The Ducas–Micciancio (DM/FHEW) and Chilotti–Gama–Georgieva–Izabachène (CGGI/TFHE) cryptosystems provide a general privacy-preserving computation capability. These fully homomorphic encryption (FHE) cryptosystems can evaluate an arbitrary function expressed as a general look-up table (LUT) via the method of functional bootstrapping (also known as programmable bootstrapping). The main limitation of DM/CGGI functional bootstrapping is its efficiency because this procedure has to bootstrap every encrypted number separately. A different bootstrapping approach, based on the Cheon–Kim–Kim–Song (CKKS) FHE scheme, can achieve much smaller amortized time due to its ability to bootstrap many thousands of numbers at once. However, CKKS does not currently provide a functional bootstrapping capability that can evaluate a general LUT. An open research question is whether such capability can be efficiently constructed. We give a positive answer to this question by proposing and implementing a general functional bootstrapping method based on CKKS-style bootstrapping. We devise a theoretical toolkit for evaluating an arbitrary function using the theory of trigonometric Hermite interpolations, which provides control over noise reduction during functional bootstrapping. Our experimental results for 8-bit LUT evaluation show that the proposed method achieves the amortized time of 0.75 milliseconds, which is three orders of magnitude faster than the DM/CGGI approach and 6.6x faster than (a more restrictive) amortized functional bootstrapping method based on the Brakerski/Fan-Vercauteren (BFV) FHE scheme.

* Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). This work is supported in part by DARPA through HR0011-21-9-0003. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Table of Contents

1 Introduction

Homomorphic encryption is a powerful cryptographic primitive enabling computations over encrypted data without requiring intermediate decryption. Of particular interest are *Somewhat Homomorphic Encryption* (SHE) schemes, first introduced in Gentry's PhD study [Gen09a, Gen09b], which support homomorphic evaluation of addition and multiplication or their equivalents. In all practically used SHE schemes, some *noise* is added during encryption for security reasons. This noise keeps growing as computations are performed, which eventually exhausts the computational ability of ciphertexts. To support arbitrarily deep computations, Gentry proposed a bootstrapping procedure that refreshes the noise in exhausted ciphertexts to a fixed level so that further computations can be performed on them [Gen09a]. The main idea behind bootstrapping is to homomorphically evaluate the decryption circuit for the underlying SHE scheme. The use of bootstrapping allowed Gentry to introduce the concept of Fully Homomorphic Encryption (FHE) for evaluating arbitrary circuits and formulate a concrete FHE scheme based on ideal lattices. Although Gentry's original FHE scheme was inefficient, dramatically more efficient FHE schemes and bootstrapping methods were subsequently devised [MSM+22, AP23].

A major milestone was the development of Brakerski-Gentry-Vaikuntanathan (BGV) [BGV14] and Brakerski/Fan-Vercauteren (BFV) [Bra12, FV12] leveled FHE schemes for finite field arithmetic. These schemes support efficient finite-arithmetic operations over vectors of bounded integers and include an improved bootstrapping procedure, which still follows Gentry's bootstrapping blueprint. However, the runtime of this bootstrapping procedure (even for its optimized modern variants) is not yet practical for many applications, and takes on the order of 1 minute (for roughly 1,000 encrypted integers) on a modern CPU [AP23]. Note that many practical applications of BGV and BFV typically do not use bootstrapping, i.e., run these schemes in the leveled mode.

The next major milestone was the Ducas-Micciancio (DM) FHE cryptosystem [DM15] (also called FHEW), which supports efficient bootstrapping for Boolean gates (as low as 10 milliseconds per Boolean gate for optimized DM-like schemes on modern CPUs [Zam22, AAB+22]). The DM cryptosystem deviates from Gentry's blueprint in two ways. First, it switches between multiple schemes, where the input scheme is additively homomorphic, i.e., it does not support homomorphic multiplication, and the bootstrapping accumulator scheme is somewhat homomorphic. Second, the DM cryptosystem supports a special-purpose look-up table (LUT) evaluation (outputting an encrypted bit using internal modulo 4 arithmetic) as part of bootstrapping [MP21]. For the additive homomorphic encryption scheme, the DM cryptosystem uses Regev's Learning With Errors (LWE)-based scheme [Reg09][3]. Chilotti *et al.* subsequently proposed an FHE cryptosystem using the DM blueprint but with a different

[3] Note that many LWE ciphertexts can be combined into one BFV-compatible Ring LWE ciphertext; we will refer to input ciphertexts as (R)LWE ciphertexts in the rest of the paper.

bootstrapping accumulator–and additional optimizations [MP21]—which is typically referred to as the Chilotti-Gama-Georgieva-Izabachène (CGGI) [CGGI16] (also called TFHE).

The special-purpose LUT evaluation capability of DM/CGGI (later extended to larger plaintext moduli [CJP21]), which is called *functional bootstrapping* [KS22, LMP23] or *programmable bootstrapping* [CJP21, CLOT21, Zam22],[4] was used to devise procedures for evaluating arbitrary functions over relatively small plaintext spaces, typically not higher than 3-8 bits. Recently, three different methods for arbitrary function evaluation using DM-like schemes were proposed [CLOT21, KS22, LMP23], with the method from [LMP23] having the smallest complexity and noise growth. Note that all three methods require at least two DM/CGGI functional bootstrapping operations for evaluating small-precision arbitrary functions. The evaluation of arbitrary functions over small plaintext spaces can be used for evaluating multi-precision (large-precision) functions, though this extension is generally computationally expensive (except for special cases such as sign/comparison) and often requires many functional bootstrapping operations [GBA21, LMP23].

Another major advance was the introduction of *approximate* homomorphic encryption for supporting efficient homomorphic polynomial computations over real and complex numbers [CKKS17]. The authors also proposed a concrete FHE scheme, referred to as the Cheon-Kim-Kim-Song (CKKS) scheme (also called HEAAN). The CKKS scheme provides a practical solution for many privacy-preserving machine learning applications, significantly outperforming both BGV/BFV and DM/CGGI [MSM$^+$22]. From the throughput perspective, the CKKS scheme achieves the most efficient bootstrapping operation; its optimized variants require on the order of 10 seconds for bootstrapping 32,768 encrypted real numbers with precision of roughly 15 bits [BCKS24, AP23]. Although the CKKS scheme deviates from prior exact schemes, BGV/BFV and DM/CGGI, in terms of correctness requirements, CKKS bootstrapping still conceptually uses Gentry's blueprint, i.e., it homomorphically evaluates its own decryption circuit [CHK$^+$18]. However, the CKKS scheme does not provide a robust, efficient solution for evaluating discontinuous functions, e.g., sign/comparison, as polynomial approximations of these functions are complicated by the requirements of knowing the approximation range and achieving desired precision. For this reason, CKKS often uses other schemes, e.g., DM/CGGI, via scheme switching to evaluate discontinuous functions [LHH$^+$21, LMP23], which is associated with high performance costs.

In summary, the DM/CGGI method provides the general functionality of evaluating arbitrary functions, but its efficiency is significantly lower than for both BGV/BFV and CKKS methods. The primary reason is that in DM/CGGI, bootstrapping is performed for each number independently while in the case of BGV/BFV and CKKS, one bootstrapping operation can refresh thousands of numbers at once using the Single Instruction/Multiple Data (SIMD) packing

[4] For consistency, we will use the functional bootstrapping term throughout the paper (noting that it is equivalent in meaning to programmable bootstrapping).

of a vector into one ciphertext [AP23] (note that CKKS typically outperforms BGV/BFV bootstrapping by more than one order of magnitude). As a result, several studies with a focus on amortized functional bootstrapping or leveled LUT evaluation in SIMD schemes recently appeared [LW23, LMS24, CKKL24, BCKS24, LW24] (we discuss these works in Section 1.1).

An open research question is whether arbitrary functions can be evaluated using CKKS-style bootstrapping, i.e., whether CKKS bootstrapping can be used to construct a general functional bootstrapping capability. The benefits of such an approach are improved efficiency (as CKKS bootstrapping has the best throughput among all FHE schemes) and more general functionality in CKKS (to enable direct support of discontinuous function evaluation). We give a positive answer to this question by proposing a general method of functional bootstrapping based on CKKS and provide initial experimental results to showcase its performance.

Our Contributions. Our main contribution is a general functional bootstrapping capability based on CKKS for input (R)LWE ciphertexts, which can evaluate arbitrary functions in $\mathbb{Z}_p$ for any integer $p \geq 2$. The functional bootstrapping capability uses a trigonometric Hermite interpolation that has the same values as the interpolated function and first derivative set to zero at all domain points. Setting the first derivative to zero provides a noise reduction/cleaning ability (to accommodate the approximate nature of the CKKS scheme). Using trigonometric interpolation theory, we derive an analytical expression for the general case in terms of Fourier series. We also devise an efficient "FHE-friendly" algorithm for evaluating the trigonometric series in CKKS in terms of the complex exponential function. Moreover, we derive analytical expressions for cases when higher-order derivatives are also set to zero—if further noise reduction is needed for functional bootstrapping at the expense of increased computational complexity.

Our second contribution is a homomorphic evaluation algorithm of the floor or digit extraction function for an encrypted message in $\mathbb{Z}_p$ built from a trigonometric interpolation for the step function. We devise analytical expressions for both step and floor functions and show how the latter is built from the step function. These functions are used as subroutines in building multi-precision function evaluation when it is more efficient to break down a large encrypted (R)LWE message into digits and run functional bootstrapping for each digit.

Our third contribution is the development of a multi-precision sign evaluation of (R)LWE ciphertexts (encrypting messages in $\mathbb{Z}_P$, where $P > p$) using both floor and step functions for encrypted messages in $\mathbb{Z}_p$ as subroutines. We also discuss how a multi-precision arbitrary function evaluation capability can be built using our algorithm from arbitrary function evaluation in $\mathbb{Z}_p$.

Our fourth contribution covers various extensions of the functional bootstrapping capability, including supporting the evaluation of discontinuous functions directly in CKKS (over CKKS input ciphertexts) and multi-value functional bootstrapping.

Our fifth contribution is the formulation of a general method for noise reduction in CKKS using polynomial Hermite interpolation theory. We show that prior limited noise cleaning capabilities are special cases of this general method.

We also implement our functional bootstrapping method and multi-precision functional evaluation capabilities in OpenFHE, evaluate their performance, and compare the complexity/runtime results with other state-of-the-art methods. Our experimental results suggest that for 8-bit LUT evaluation the proposed method achieves an amortized time of 0.75 milliseconds, which is three orders of magnitude faster than for the DM/CGGI method and 6.6x faster than a more limited functional bootstrapping functionality based on the BFV scheme.

Technical Overview. We develop a cryptosystem that supports general functional bootstrapping using the DM blueprint. In the classical DM/CGGI setting, the input ciphertexts are encrypted using the LWE scheme and bootstrapping is performed with the Ring-Gentry-Sahai-Waters (RGSW) scheme [DM15]. In other words, the secret key for the LWE scheme is encrypted using the RGSW scheme. Such a cryptosystem can bootstrap only one number at a time because all possible values in an LWE ciphertext get mapped to polynomial coefficients in RGSW [MP21].

We devise a "vectorized" cryptosystem, where many numbers are encrypted in an RLWE ciphertext and all these numbers are bootstrapped at once using the CKKS scheme. The RLWE scheme here is equivalent to the BFV scheme using the coefficient encoding, and the encryption can be written as $\mathrm{RLWE}(\boldsymbol{m}) = \left([-\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e} + (q/p) \cdot \boldsymbol{m}]_q, \boldsymbol{a}\right)$, where $\boldsymbol{m} \in \mathbb{Z}_p^w$ and w is the number of encrypted integers (up to the ring dimension N). For simplicity of exposition, we focus on the symmetric-key encryption case and assume that both q and p are powers of two. In this work, we specifically choose to work with RLWE because it is a more compact and practical representation. Nevertheless, an RLWE ciphertext can also be thought of as w LWE ciphertexts packed into one RLWE ciphertext (the conversions between LWE and RLWE ciphertexts in both directions are known procedures discussed and optimized elsewhere [BCK+23, BCKS24]).

The RLWE encryption of $\frac{q}{p}\boldsymbol{m}$ can be interpreted as a CKKS encryption using the coefficient encoding of $\Delta\frac{\boldsymbol{m}}{p}$, where Δ is the CKKS scaling factor (for simplicity, we focus in the description here on CKKS instantiated for a power-of-two ciphertext modulus). This CKKS ciphertext is "exhausted" and cannot support any further multiplications.

To perform functional bootstrapping, we first raise the ciphertext to a much larger modulus Q'_L (supporting L computational levels). This changes the encrypted message from $\Delta\frac{\boldsymbol{m}}{p}$ to $\Delta\frac{\boldsymbol{m}}{p} + qI(X)$, or, equivalently, $\Delta\frac{m(X)}{p} + qI(X)$ as the message $\boldsymbol{m}$ is encoded in polynomial coefficients. Our goal is to obtain an encryption of $f_p(\boldsymbol{m}) \in \mathbb{Z}_p^w$, for an arbitrary function f_p defined as a p-to-p LUT (in general, the output modulus can be different from the input one). We evaluate the LUT using a properly chosen interpolation. Prior to the LUT evaluation, we perform homomorphic encoding to place both the message and q-overflows into CKKS slots to enable CKKS-style polynomial evaluation.

The main challenge is how to choose the interpolation. This interpolation has to remove the q-overflows, i.e., it has to be a trigonometric series. As CKKS bootstrapping adds to the noise present in the RLWE message, we also want to reduce this noise during the evaluation of the trigonometric series. We observe that the approach satisfying both is the trigonometric Hermite interpolation. As our main solution, we use the first-order trigonometric Hermite interpolation, which matches the interpolated function at all p points of interest and sets first derivatives to zero, to provide quadratic noise reduction.

We use results from trigonometric interpolation theory to derive an FHE-friendly analytical expression for evaluating first-order Hermite interpolation for arbitrary p. We start with a series of shifted cosines and then transform it into a power series for the complex exponential function $e^{2\pi i m_j/p}$ for $j \in [w]$. The power series, which is the bottleneck operation for larger p, has degree $p-1$ and can be evaluated using the Paterson-Stockmeyer method, consuming roughly $\sqrt{2p}$ homomorphic multiplications. In other words, the complexity increases by $\sqrt{2}$ every time p is increased by 2.

After evaluating the power series, we perform homomorphic decoding to go back to our RLWE format. Note the result is still a BFV ciphertext and can be decrypted exactly without any approximation error. Hence, from the perspective of IND-CPA^D security [LM21, ABMP24], no flooding is needed to achieve security for shared decryption results (in contrast to the single-scheme CKKS instantiations). In other words, CKKS is used as a black box in our cryptosystem, just like RGSW is used in DM/CGGI cryptosystems.

For applications requiring additional noise reduction during functional bootstrapping, we also derive power series for second- and third-order trigonometric Hermite interpolations where the first two and three derivatives, respectively, are set to zero. Moreover, we show that polynomial Hermite interpolations (not periodic in contrast to trigonometric interpolations) can also be used to evaluate LUTs in a leveled setting and/or to reduce noise. These polynomial interpolations generalize prior results from [CKK20, DMPS24, CKKL24]. We show that trigonometric Hermite interpolations are more efficient for LUT evaluation than polynomial Hermite interpolations in the settings where bootstrapping is needed.

Our method can efficiently support LUTs only for a limited range of p, e.g., we were able to run LUTs up to 12 bits using 64-bit words. If a higher precision is required or some special functions are to be evaluated, such as sign evaluation, one can use a multi-precision approach, where LUTs for smaller p's are used to support large plaintext moduli P. Using the blueprint of [LMP23], we develop multi-precision digit extraction and sign evaluation procedures. Both are based on the LUT evaluation of the floor function, for which we derive convenient analytical expressions. The digit extraction procedure allows evaluating large-precision LUTs by working with smaller-size LUTs for individual digits. It is worth noting that evaluating the floor function using our cryptosystem requires a single functional bootstrapping operation, as compared to two bootstrapping operations in the classical DM/CGGI cryptosystems.

We also introduce a number of optimizations and extensions. For instance, our functional bootstrapping method supports efficient multi-value LUT evaluation, where multiple LUTs for the same ciphertext can be evaluated at the cost slightly higher than a single LUT evaluation. We discuss how our method can be used for evaluating discontinuous functions in native CKKS.

1.1 Related Works

For the classical DM/CGGI method, we compare our results with the procedures for LUT evaluation and multi-precision sign evaluation described in [LMP23] and LUT evaluation in [TCBS23]. The results of our comparison are presented in Appendix A.2. In summary, our method achieves a better throughput than DM/CGGI functional bootstrapping as soon as the number of evaluated encrypted numbers reaches the order of one thousand/one hundred, as our method scales better with p, both asymptotically and practically. The amortized time for larger values of p, e.g., 2^8, is three orders of magnitude smaller in our method as compared to CGGI functional bootstrapping. We also compare our timing results for 8-bit LUTs with another multi-precision method based on CGGI circuit bootstrapping in Appendix A.2.

Bae *et al.* propose a method for evaluating Boolean gates using a CKKS-like bootstrapping [BCKS24]. The functionality of this method is the same as the functionality available in the original DM cryptosystem [DM15]. If our functional bootstrapping method is instantiated for the floor or step function at $p = 2$, our trigonometric series reduces to the same function as for Boolean CKKS bootstrapping in [BCKS24]. In other words, the Boolean bootstrapping in [BCKS24] can be viewed as a special case of our method for the floor/step function at $p = 2$. However, our approach supports arbitrary values of p and arbitrary functions over $\mathbb{Z}_p$. We discuss how our implementation results for 1-bit LUT compare with the implementation results of [BCKS24] in Appendix A.1. We also compare our method with the multi-precision LUT evaluation based on 1-bit LUTs in Section 7.

Chung *et al.* develop a LUT evaluation method in both CKKS and BFV using a special exponential encoding and multivariate polynomial interpolations [CKKL24]. Their results demonstrate that the method efficiently supports LUT evaluation up to $p = 2^{12}$. However, this exponential encoding does not support multiplications between ciphertexts and additional (costly) procedures need to be implemented to switch to and from the slot encoding. Moreover, their method does not refresh the ciphertexts (i.e., it is not based on functional bootstrapping) and, hence, regular CKKS or BFV bootstrapping would need to be used for deep computations. We compare their method and implementation results with ours in Appendix A.3.

Liu and Wang devise an amortized (somewhat limited) functional bootstrapping method for DM/CGGI ciphertexts using BFV as the bootstrapping scheme [LW23, LW24]. We compare our results with these works in Appendix A.2. In summary, our method has a higher throughput (from 3.2x for 3-bit LUT to 8.8x for 12-bit LUT) and easily supports multi-precision extensions.

Lee et al. propose functional bootstrapping for BFV only and BFV-to-CKKS scenarios [LMS24]. Their method works with the plaintext space $\mathbb{Z}_p^r$, i.e., builds upon regular BFV/BGV bootstrapping [GV23]. The main limitation of their method is inherited from regular BFV bootstrapping: only a small number of slots can be efficiently supported, especially when only power-of-two cyclotomic rings are available for instantiating BFV (the latter is true for all common open-source software libraries implementing BFV). As a result, the amortized time of this method is significantly (orders of magnitude) higher than both for our method and [LW23, LW24], as illustrated in Appendix A.2.

There have been many studies on large-precision sign/comparison evaluation algorithms, both using leveled computations in SIMD schemes and functional bootstrapping in DM/CGGI schemes (see [LMP23] for a summary of main methods). A highlight is the CKKS method proposed in [CKK+19], which achieves optimal complexity using leveled CKKS. The drawback of this method is that it does not include bootstrapping (hence bootstrapping has to be done separately), which may require larger lattice parameters than our method and several bootstrapping operations. Given the large number of studies specifically on sign evaluation, complexity of fair comparison of our (bootstrapping-based) method with leveled SIMD solutions, and the main focus of our work on arbitrary function evaluation, we leave such comparison for future work.

1.2 Concurrent Works

We remark that a concurrent work by Bae *et al.* [BKSS24] seems to use the same exponential encoding as [CKKL24] to allow functional bootstrapping for small integers via CKKS. At the time of submission, only the abstract of this work is available, and therefore we cannot compare our work with their results. For evaluating 8-bit LUTs on DM/CGGI ciphertexts, our method yields an amortized time of 0.81 milliseconds (with the LWE to RLWE conversion), which is 4.6 times faster than the 3.75 milliseconds mentioned in the abstract of [BKSS24].

1.3 Organization

We provide the necessary background on the DM/CGGI and CKKS schemes and methods in Section 2. In Section 3, we derive the analytical expressions for arbitrary function evaluation using CKKS-style bootstrapping and examine their properties. These expressions are used in Section 4, which presents our algorithm for general amortized functional bootstrapping of RLWE ciphertexts, and in Section 5, which describes our algorithms for digit extraction and multi-precision function evaluation. Section 6 summarizes the approach for the functional bootstrapping of CKKS ciphertexts. In Section 7 we showcase our implementation results and performance. We provide concluding remarks in Section 8. We also present a detailed comparison of our results with the state-of-the-art methods in Appendix A. The remaining appendices provide additional preliminaries and detailed derivations.

2 Preliminaries

All logarithms are expressed in base 2 if not indicated otherwise. Vectors and ring elements are indicated in bold. We choose the ring dimension N as a power of two for efficiency reasons. Additional background is provided in Appendix B.

2.1 LWE Encryption Scheme and Its Ring Variant

We recall the definition of LWE ciphertexts [Reg09].

The LWE cryptosystem [Reg09] is parametrized by a plaintext modulus p, ciphertext modulus q, and secret dimension n. The LWE encryption of a message $m \in \mathbb{Z}_p$ under (secret) key $\boldsymbol{s} \in \mathbb{Z}^n$ is a vector $(\boldsymbol{a}, b) \in \mathbb{Z}_q^{n+1}$ such that

$$b = -\langle \boldsymbol{a}, \boldsymbol{s} \rangle + (q/p) \cdot m + e \pmod q$$

where e is a small error term, $|e| < q/(2p)$. The message m is recovered by first computing the approximate LWE decryption function

$$\mathsf{Dec}_{\boldsymbol{s}}(\boldsymbol{a}, b) = b + \langle \boldsymbol{a}, \boldsymbol{s} \rangle \pmod q = (q/p) \cdot m + e$$

and then rounding the result to the closest multiple of (q/p).

The ciphertext modulus of LWE ciphertexts can be changed (at the cost of a small additional noise proportional to the secret key size) simply by scaling and rounding its entries, which is called modulus switching.

The BFV-style RLWE encryption (LWE scheme extension to rings) [Bra12, FV12] can be written as $(\boldsymbol{b}, \boldsymbol{a}) \in \mathcal{R}_q^2$ such that $\boldsymbol{a} \leftarrow \mathcal{R}_q$ and $\boldsymbol{b} = -\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e} + (q/p) \cdot \boldsymbol{m}$, where $\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1 \rangle$, $\boldsymbol{s} \leftarrow \chi_{\mathsf{key}}$, $\boldsymbol{e} \leftarrow \chi_{\mathsf{err}}$, $\boldsymbol{m} \in \mathcal{R}_p$, and χ_{key} and χ_{err} are small distributions over $\mathcal{R}$. Note that $\boldsymbol{m}$ in our case is encoded in polynomial coefficients, and, hence, we also refer to $\boldsymbol{m}$ as $m(X)$ in the paper. The decryption in this case is computed as $\left\lfloor p \cdot (\boldsymbol{b} + \boldsymbol{a} \cdot \boldsymbol{s}) / q \right\rceil_p$.

The conversion of many LWE ciphertexts to a single RLWE ciphertext is known as (base) ring packing and requires a (plaintext matrix)-(ciphertext vector) multiplication; see [BCK+23] for state-of-the-art algorithms. The conversion from one RLWE to many LWE ciphertexts, known as sample extraction, is much simpler and faster; it is performed by selecting and reordering polynomial coefficients [CGGI16].

2.2 CKKS Scheme

The original CKKS scheme is formulated for cyclotomic polynomial rings $\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1 \rangle$, where N is a ring dimension that is a power of two. With a scaling factor $\Delta = 2^\rho$ and a zero-level modulus $q_0' = 2^{\rho_0}$ (usually q_0' is larger than Δ for correct decryption), a modulus at the level ℓ is typically defined as $Q_\ell' = 2^{\rho_0 + \ell \cdot \rho} = q_0' \cdot \Delta^\ell$, i.e., the scheme works with residue rings $\mathcal{R}_{Q_\ell'} = \mathcal{R}/Q_\ell'\mathcal{R} = \mathbb{Z}_{Q_\ell'}[X]/\langle X^N + 1 \rangle$. We denote $M = 2N$, and by $\mathbb{Z}_M^* = \{x \in \mathbb{Z}_M : \mathsf{gcd}(x, M) = 1\}$ the unit multiplication group in $\mathbb{Z}_M$. The canonical embedding $\tau : \mathcal{S} \to \mathbb{C}^N$ is defined as $\tau(\boldsymbol{a}) = \left(\boldsymbol{a}(\zeta^j)\right)_{j \in \mathbb{Z}_M^*}$ for $\mathcal{S} = \mathbb{R}[X]/\left\langle X^N + 1 \right\rangle$ and $\zeta = \exp(2\pi i/M)$. Its ℓ_∞-norm is called the *canonical embedding norm* and is denoted as $\|\boldsymbol{a}\|^{\mathsf{can}} = \|\tau(\boldsymbol{a})\|_\infty$. For a power-of-two $n \leq N/2$, we also define mappings $\tau'_n : \mathcal{S} \to \mathbb{C}^n$ used to encode and decode a vector of length n in the CKKS scheme [CKKS17, CHK+18]. The setup, key generation, encryption, decryption, encoding and decoding algorithms [CKKS17, HK20] are given below, and the evaluation-related algorithms are described in Appendix B.3:

- $\mathsf{Setup}(1^\lambda)$. For an integer $L \geq 0$ corresponding to the largest ciphertext modulus level, given the security parameter λ, output the ring dimension N. Set the small distributions χ_{key}, χ_{err}, and χ_{enc} over $\mathcal{R}$ for secret, error, and encryption, respectively.
- KeyGen. Sample a secret $\boldsymbol{s} \leftarrow \chi_{\mathsf{key}}$, a random $\boldsymbol{a} \to \mathcal{R}_{Q'_L}$, and error $\boldsymbol{e} \leftarrow \chi_{\mathsf{err}}$. Set the secret key $\mathsf{sk} \leftarrow (1, \boldsymbol{s})$ and public key $\mathsf{pk} \leftarrow (\boldsymbol{b}, \boldsymbol{a}) \in \mathcal{R}^2_{Q'_L}$, where $\boldsymbol{b} \leftarrow -\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e} \pmod{Q'_L}$.
- $\mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m})$. For $\boldsymbol{m} \in \mathcal{R}$, sample $\boldsymbol{v} \leftarrow \chi_{\mathsf{enc}}$ and $\boldsymbol{e}_0, \boldsymbol{e}_1 \leftarrow \chi_{\mathsf{err}}$. Output $\mathsf{ct} \leftarrow \boldsymbol{v} \cdot \mathsf{pk} + (\boldsymbol{m} + \boldsymbol{e}_0, \boldsymbol{e}_1) \pmod{Q'_L}$.
- $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$. For $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}^2_{Q'_\ell}$, output $\tilde{\boldsymbol{m}} = \boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} \pmod{Q'_\ell}$.
- $\mathsf{Encode}(\boldsymbol{x}, \Delta)$. For $\boldsymbol{x} \in \mathbb{C}^n$, output the polynomial $\boldsymbol{m} \leftarrow \left\lceil \tau_n'^{-1}(\Delta \cdot \boldsymbol{x}) \right\rfloor \in \mathcal{R}$.
- $\mathsf{Decode}(\boldsymbol{m}, \Delta)$. For plaintext $\boldsymbol{m} \in \mathcal{R}$, output the vector $\boldsymbol{x} \leftarrow \tau'_n(\boldsymbol{m}/\Delta) \in \mathbb{C}^n$.

2.3 RNS Representation

Our CKKS implementation utilizes the Chinese Remainder Theorem (referred to as integer CRT) representation to break multi-precision integers in $\mathbb{Z}_{Q'}$ into vectors of smaller integers to perform operations efficiently using native (64-bit) integer types. The integer CRT representation is also often referred to as the Residue-Number-System (RNS) representation. We use a zero level modulus q'_0 and a chain of same-size prime moduli $q'_1, q'_2, \ldots, q'_L$ satisfying $q'_i \equiv 1 \bmod 2N$ for $i = 1, \ldots, L$. Here, the modulus Q'_ℓ is computed as $\prod_{i=0}^{\ell} q'_i$. All polynomial multiplications are performed on ring elements in the polynomial CRT representation where all integer components are represented in the integer CRT basis. Further details on the RNS implementation are given in Appendix B.3.

2.4 CKKS Bootstrapping

The CKKS bootstrapping procedure typically assumes that the input ciphertext ct is at level $L = 0$, i.e., $Q' = q'_0$. In other words, no homomorphic multiplications are allowed. The goal is to raise the ciphertext to a level L_0 so that depth-L_0 computations could be performed on it.

The high-level procedure includes the following steps [CHK+18]:

1. $\mathsf{ct}_1 \leftarrow \mathsf{ModRaise}(\mathsf{ct}, Q'_L)$: Raise the modulus from q'_0 to $Q'_L = \prod_{i=0}^{L} q'_i$, where $L > L_0$ as the bootstrapping procedure consumes some levels, namely

$L_b = L - L_0$ levels. The effect of this operation is that the new ciphertext corresponds to a decryption of $t(X) = m(X) + q_0' \cdot I(X)$, where $|I| < K$. The goal of the next steps is remove the term $q_0' \cdot I(X)$.

2. $\mathsf{ct}_2 \leftarrow \mathsf{CtS}(\mathsf{ct}_1)$: Encode $t(X)$ in the plaintext slots by homomorphically running $\tau'^{-1}(\boldsymbol{t})$. As a result, we get an encryption of a plaintext vector where each coefficient t_i is now stored in a separate slot. This allows one to apply integer-level modular reductions to all plaintext slots.
3. $\mathsf{ct}_3 \leftarrow \mathsf{EvalMod}(\mathsf{ct}_2)$: Approximate $[t_i]_{q_0'} \approx \frac{q_0'}{2\pi} \sin\left(\frac{2\pi t_i}{q_0'}\right)$, where $q_0' \gg m_i$ to achieve an adequate accuracy. The sine wave is then interpolated using a polynomial, which can be efficiently evaluated using homomorphic encryption. As a result, we get $[t_i]_{q_0'} \approx m_i$. Denote the result as $\hat{m}_i$.
4. $\mathsf{ct}_4 \leftarrow \mathsf{StC}(\mathsf{ct}_3)$: Decode $\hat{m}_i$ back to the coefficient representation to yield $\hat{m}(X)$ by running τ' homomorphically. The goal is to minimize the difference between $m(X)$ and $\hat{m}(X)$.

The total depth L_b needed for bootstrapping is $L_{enc} + L_{mod} + L_{dec}$, where L_{enc} and L_{dec} are the levels needed for encoding and decoding, respectively, and L_{mod} is the depth needed for approximate modular reduction.

3 Analytical Expressions for Arbitrary Function Evaluation

In this section we derive all intermediate and final analytical expressions for single- and multi-precision function evaluation using CKKS.

3.1 Trigonometric Hermite Interpolation for Arbitrary Function Approximation

To evaluate an arbitrary function $f : \mathbb{Z}_p \to \mathbb{Z}_p$, we aim to construct a mapping that approximates f at p equidistant points of interest. Specifically, we seek a polynomial approximation for the mapping $\frac{m}{p} + I \mapsto f(m)$, where $m \in \mathbb{Z}_p$ and I is an integer value.

As our goal is to approximate a periodic function (with period 1), it is natural to use a trigonometric interpolation in the form of a (truncated) Fourier series for fractional $x = \frac{j}{p}$, where $j \in [p]$,

$$R(x) = a_0 + \sum_{k=1}^{p-1} \left(a_k \cos(2\pi kx) + b_k \sin(2\pi kx)\right), \tag{1}$$

where the coefficients $\{a_k\}_{k=0}^{p-1}$ and $\{b_k\}_{k=1}^{p-1}$ are to be determined.

The messages encrypted in RLWE ciphertexts are not exact and contain small noise, which gets removed via rounding in normal RLWE decryption. When we use CKKS to perform homomorphic decryption of the messages, the noise is not automatically removed and actually increases due to the homomorphic computations performed as part of CKKS bootstrapping. To reduce the noise, we require that first derivatives of the trigonometric interpolation be set to zero at all points of interest, which results in quadratic reduction of the noise. In this case, the first-order error terms in the Taylor series expansions at all points of interest vanish. In [BCKS24], for binary bootstrapping, quadratic reduction of noise was also chosen.

Hence, our interpolation problem reduces to finding a trigonometric polynomial $R(x)$ that satisfies the conditions

$$R\left(\frac{k}{p}\right) = f(k), \quad R'\left(\frac{k}{p}\right) = 0, \tag{2}$$

for $k \in [p]$. The problem (1) with conditions (2) represents a linear systems of equations that can be numerically solved for coefficients $\{a_k\}_{k=0}^{p-1}$ and $\{b_k\}_{k=1}^{p-1}$ using a standard linear solver.

However, it is more convenient both for analysis and practical use to find general analytic expressions for the coefficients. A trigonometric polynomial $R(x)$ satisfying conditions (2) is known in trigonometric interpolation theory as a special case of the first-order trigonometric Hermite interpolation [SV65].

We summarize the result here.

Theorem 1. *The first-order trigonometric Hermite interpolation polynomial satisfying the constraints* (2) *exists, is unique and has the following expression:*

$$\begin{aligned} R\left(x\right) &= a_0 + \sum_{k=1}^{p-1} \left(a_k \cos(2\pi k x) + b_k \sin(2\pi k x)\right), \\ a_0 &= \frac{1}{p}\sum_{l=0}^{p-1} f(l), \quad a_k = \frac{2(p-k)}{p^2}\sum_{l=0}^{p-1} f(l)\cdot\cos\left(\frac{2\pi l k}{p}\right), \\ b_k &= \frac{2(p-k)}{p^2}\sum_{l=0}^{p-1} f(l)\cdot\sin\left(\frac{2\pi l k}{p}\right). \end{aligned} \tag{3}$$

Proof. For the general case of a trigonometric Hermite interpolation $(0,\, M)$, i.e., where the conditions for the function itself and M-th derivative are given, Sharma and Varma derived an explicit form of $R(x)$ and established its uniqueness (see Theorem 1 [SV65] for the details and proof).

Our case corresponds to $M = 1$, where all values of first derivative are zero, and the expression for R is written as

$$R(x) = \sum_{l=0}^{p-1} f(l)\cdot U\left(2\pi\left(x - \frac{l}{p}\right)\right) \text{ with } U(x) = \frac{1}{p}\left(1 + \frac{2}{p}\sum_{k=1}^{p-1}(p-k)\cos(kx)\right). \tag{4}$$

Substituting $U(x)$ into $R(x)$ in (4), the expression for R can be rewritten as

$$R(x) = \frac{1}{p}\sum_{l=0}^{p-1} f(l) + \frac{2}{p^2}\sum_{l=0}^{p-1}\sum_{k=1}^{p-1} f(l)(p-k)\cos\left(2\pi k\left(x - \frac{l}{p}\right)\right).$$

By applying the cosine angle subtraction identity and rearranging the order of summation, the second term can be rewritten as

$$\sum_{k=1}^{p-1} \frac{2(p-k)}{p^2} \sum_{l=0}^{p-1} f(l) \left(\cos(2\pi kx) \cos\left(\frac{2\pi lk}{p}\right) + \sin(2\pi kx) \sin\left(\frac{2\pi lk}{p}\right)\right).$$

Matching the transformed expression for $R(x)$ with equation (1) yields the sought expressions for the coefficients written in the theorem statement. $\square$

3.2 FHE-Friendly Expression using Complex Exponential Function

The Fourier series given by the expression (3) is not convenient for FHE evaluation as it contains series of both sines and cosines, which have to be separately evaluated (typically through polynomial approximations). A more FHE-friendly expression can be derived using the complex exponential function leading to the polynomial evaluation over vectors of complex numbers.

The high-level idea is to extend the expression (4) from cosines to the corresponding complex exponential functions, perform the evaluation in the complex domain, and then extract the real part of the result.

Corollary 1. *The first-order trigonometric Hermite interpolation polynomial $R(x)$ satisfying the constraints* (2) *can be expressed as the real part of complex polynomial $T(x)$ given by*

$$T(x) = \alpha_0 + \sum_{k=1}^{p-1} \alpha_k \cdot e^{2\pi ikx}, \tag{5}$$

$$\alpha_0 = \frac{1}{p}\sum_{l=0}^{p-1} f(l), \quad \alpha_k = \frac{2(p-k)}{p^2} \sum_{l=0}^{p-1} f(l) \cdot e^{-2\pi kli/p}.$$

Proof. The complex generalization of (4) can be written as

$$T(x) = \sum_{l=0}^{p-1} f(l) \cdot W\left(2\pi\left(x - \frac{l}{p}\right)\right) \text{ with } W(x) = \frac{1}{p} + \frac{2}{p^2}\sum_{k=1}^{p-1}(p-k)e^{ikx}. \tag{6}$$

Substituting $W(x)$ into $T(x)$ in (6) yields

$$T(x) = \frac{1}{p}\sum_{l=0}^{p-1} f(l) + \frac{2}{p^2}\sum_{k=1}^{p-1}\sum_{l=0}^{p-1} f(l)(p-k) \cdot e^{-2\pi kli/p} \cdot e^{2\pi ikx}.$$

By writing $T(x) = \alpha_0 + \sum_{k=1}^{p-1} \alpha_k \cdot e^{2\pi ikx}$ and identifying the coefficients $\{\alpha_k\}_{k=0}^{p-1}$, we obtain the expressions in the theorem statement.

The sought expression $R(x)$ is the real part of $T(x)$:

$$R(x) = Re(T(x)), \tag{7}$$

which is equivalent to (1), (3). $\square$

Hence, we obtain a power series of degree $p-1$ for $E(x) := e^{2\pi ix}$, which can be efficiently evaluated using the Paterson-Stockmeyer algorithm [PS73].

3.3 Analytical Expressions for Floor and Step Functions

The floor and step functions are used as subroutines in the multi-precision sign and LUT evaluation. Here, we provide an interesting relation between these functions and analytical expressions. For simplicity, we focus on the case of p being a power of two, which is the main practical scenario for using these subroutines in multi-precision evaluation.

First, we introduce a scaled step (Heaviside) function step_p as the function with period p such that for $k \in [p]$:

$$\mathsf{step}_p(k) = \begin{cases} 0, & \text{if } 0 \le k < p/2 \\ p/2 & \text{if } p/2 \le k < p. \end{cases} \tag{8}$$

The mod_p function can be recursively defined in terms of the step_p function:

$$\mathsf{mod}_p(k) = \mathsf{mod}_{\frac{p}{2}}(k) + \mathsf{step}_p(k), \quad p > 2 \tag{9}$$

$$\mathsf{mod}_2(k) = \mathsf{step}_2(k). \tag{10}$$

The R-interpolations for the mod_p and step_p functions can be expressed as

$$\mathsf{Rmod}_p\left(x\right) = \mathsf{Rmod}_{\frac{p}{2}}\left(2x\right) + \mathsf{Rstep}_p\left(x\right), \quad p > 2 \tag{11}$$

$$\mathsf{Rstep}_p\left(x\right) = \frac{p}{4} - \frac{1}{p}\sum_{k \in S}(p-k)\left(\cos(2k\pi x) + \cot\left(\frac{\pi k}{p}\right)\sin(2k\pi x)\right), \tag{12}$$

for $x \in \{0, \frac{1}{p}, \ldots, \frac{p-1}{p}\}$ and $S = \left\{2i+1 : i \in [\frac{p}{2}]\right\}$. The expression (12) was derived from (1) and (3) via a number of simplifications (similar to those in Appendix C.1 for the complex exponential expression).

Note that for $p = 2$, we have

$$\mathsf{Rmod}_2\left(x\right) = \mathsf{Rstep}_2\left(x\right) = \frac{1}{2} - \frac{1}{2}\cos\left(2\pi x\right), \tag{13}$$

which is the same as the trigonometric interpolation in [BCKS24] for binary CKKS bootstrapping. This is not surprising as the first derivative was also set to zero in [BCKS24] to achieve noise reduction during binary bootstrapping.

For evaluation with FHE, we derived analytical expressions in terms of the complex exponential function (see Appendix C.1 for the derivation details):

$$\mathsf{Rmod}_p\left(x\right) = \frac{p-1}{2} + \frac{1}{p}\sum_{k=1}^{p-1}(p-k)\left(-1 + i\cot\left(\frac{\pi k}{p}\right)\right)e^{2\pi i k x}, \tag{14}$$

$$\mathsf{Rstep}_p\left(x\right) = \frac{p}{4} + \frac{1}{p}\sum_{k \in S}(p-k)\left(1 - i\cot\left(\frac{\pi k}{p}\right)\right)e^{2\pi i k x}, \tag{15}$$

where $S = \left\{2i+1 : i \in [\frac{p}{2}]\right\}$.

3.4 Higher-Order Trigonometric Hermite Interpolations

So far, we have focused on the first-order trigonometric Hermite interpolation $R(x)$ with constraints (2), which achieves quadratic noise reduction. If additional noise reduction is needed, a second-order or even third-order trigonometric Hermite interpolation can be used. Here, we derive expressions for both second- and third-order trigonometric Hermite interpolations. The proofs are deferred to Appendix C.2.

For the second-order interpolation $R_2(x)$, the constraints are written as

$$R_2\left(\frac{k}{p}\right) = f(k), \quad R_2'\left(\frac{k}{p}\right) = 0, \quad R_2''\left(\frac{k}{p}\right) = 0, \tag{16}$$

where $k \in [p]$. In [Var69], Varma derives analytical expressions for a more general Hermite trigonometric interpolation, which we use here for the second-order interpolation $R_2(x)$.

Theorem 2. *The second-order trigonometric Hermite interpolation polynomial $R_2(x)$ satisfying the constraints* (16) *exists, is unique, and can be expressed as the real part of complex polynomial $T_2(x)$, for $E(x) := e^{2\pi i x}$:*

$$T_2(x) = \alpha_0 + \sum_{v=1}^{p-1} \alpha_v \cdot E(x)^v + \sum_{k=1}^{\lfloor p/2 \rfloor} \beta_k E(x)^k - \frac{\delta_k}{2} E(x)^{p+k} - \frac{\theta_k}{2} E(x)^{p-k}, \tag{17}$$

$$\beta_k = \frac{(2-\gamma_{p,k})k(p-k)}{p^3} \sum_{l=0}^{p-1} f(l) \cdot e^{-2\pi k l i/p},$$

$$\delta_k = \frac{(2-\gamma_{p,k})k(p-k)}{p^3} \sum_{l=0}^{p-1} f(l) \cdot e^{-2\pi (p+k) l i/p},$$

$$\theta_k = \frac{(2-\gamma_{p,k})k(p-k)}{p^3} \sum_{l=0}^{p-1} f(l) \cdot e^{-2\pi (p-k) l i/p},$$

where α_v are the same as in the first-order expression (5) *in Theorem 1; $\gamma_{p,k} = 1$ if p is even and $k = p/2$, while $\gamma_{p,k} = 0$ otherwise.*

It is easy to see that for the second-order trigonometric Hermite interpolation, one needs to evaluate a power series of degree $\frac{3p}{2}$ for $E(x)$. In other words, the computational cost of going from quadratic to cubic noise reduction is to increase the degree of the polynomial evaluated using the Patterson-Stockmeyer algorithm from $p-1$ to $\frac{3p}{2}$.

For the third-order interpolation $R_3(x)$, the constraints are written as

$$R_3\left(\frac{k}{p}\right) = f(k), R_3'\left(\frac{k}{p}\right) = 0, R_3''\left(\frac{k}{p}\right) = 0, R_3'''\left(\frac{k}{p}\right) = 0, \tag{18}$$

where $k \in [p]$.

Theorem 3. *The third-order trigonometric Hermite interpolation polynomial $R_3(x)$ satisfying the constraints* (18) *exists, is unique, and can be expressed as the real part of complex polynomial $T_3(x)$, for $E(x) := e^{2\pi i x}$:*

$$T_3(x) = \alpha_0 + \sum_{k=1}^{p-1} (\alpha_k + \beta_k) \cdot E(x)^k - \frac{\delta_k}{2} E(x)^{p+k} - \frac{\theta_k}{2} E(x)^{p-k}, \tag{19}$$

$$\beta_k = \frac{2k(p-k)(2p-k)}{3p^4} \sum_{l=0}^{p-1} f(l) \cdot e^{-2\pi k l i/p},$$
$$\delta_k = \frac{2k(p-k)(2p-k)}{3p^4} \sum_{l=0}^{p-1} f(l) \cdot e^{-2\pi (p+k) l i/p},$$
$$\theta_k = \frac{2k(p-k)(2p-k)}{3p^4} \sum_{l=0}^{p-1} f(l) \cdot e^{-2\pi (p-k) l i/p},$$

where α_k are the same as in the first-order expression (5) *in Theorem 1.*

It is easy to check that $T_3(x)$ is a power series of degree $2p-1$.

3.5 Hermite Polynomial Interpolation with Noise Cleaning

When evaluating the function f *separately* from the bootstrapping process, we can use *polynomial Hermite interpolation* to approximate f at points $\{x_k\}_{k=0}^{p}$. In this method, we encode the message in CKKS in its value representation (not in coefficient format and not scaled by p) and apply polynomial Hermite interpolation to evaluate f homomorphically.

Polynomial Hermite interpolation constructs a polynomial that satisfies function values at a set of interpolation points. To reduce the noise, we also require that the first derivatives be set to zero at the set of interpolation points. Specifically, for first-order interpolation, given the conditions:

$$\bar{R}(x_k) = f(x_k), \quad \bar{R}'(x_k) = 0, \quad k = 0, 1, \ldots, p-1, \tag{20}$$

Hermite polynomial interpolation generates a polynomial $\bar{R}(x)$ of degree $2p-1$ that passes through the points $x = x_k$ with zero-valued first derivatives at these points. The interpolation polynomial $\bar{R}(x)$ can be expressed as:

$$\bar{R}(x) = \sum_{k=0}^{p-1} \left[(1 - 2(x - x_k)\ell_k'(x_k))\, \ell_k(x)^2 \right] f(x_k), \tag{21}$$

where $\ell_k(x)$ is the Lagrange basis polynomial defined as:

$$\ell_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{p-1} \frac{x - x_j}{x_k - x_j}. \tag{22}$$

Similar to Section 3.4, we can achieve better noise cleaning, by extending polynomial Hermite interpolation to higher-order derivatives. For instance, second-order Hermite polynomial interpolation includes constraints on the function values, first derivatives, and second derivatives:

$$\bar{R}\left(x_k\right) = f\left(x_k\right), \quad \bar{R}'\left(x_k\right) = 0, \quad \bar{R}''\left(x_k\right) = 0, \tag{23}$$

where $k \in [p]$. This leads to a polynomial $\bar{R}(x)$ of degree $3p-1$. By choosing the order of the polynomial Hermite interpolation appropriately, we can balance between noise reduction and computational efficiency.

Although not explicitly stated, several studies implicitly employ polynomial Hermite interpolation techniques for noise cleaning. For instance, papers such as [CKK20, DMPS24] utilize Hermite interpolation $\bar{R}$ with $p = 2$ and $f(x) = x$ at the points -1 and 1 or 0 and 1. These works apply different orders of interpolation to construct their respective polynomials f_i (and h_i in [DMPS24]), which are subsequently used to reduce noise in ciphertexts. Similarly, [CKKL24] adopts Hermite interpolation $\bar{R}$ of order one for power-of-two values of p, specifically using $f(x) = x$ at the roots of unity $e^{2\pi ik/p}$ for $k = 0, 1, \ldots, p-1$. These applications demonstrate that Hermite interpolation serves as the underlying mechanism for noise reduction, even when not explicitly mentioned by the authors, and is a valuable mechanism in approximate homomorphic computations.

We include a comparison between evaluating an LUT using functional CKKS bootstrapping via trigonometric Hermite interpolation and evaluating an LUT using CKKS bootstrapping and leveled computation using polynomial Hermite interpolation in Appendix A.3.

4 Amortized Functional Bootstrapping of RLWE Ciphertexts

We recall some notation here. We use $\boldsymbol{m} \in \mathbb{Z}^w$ to express a vector of input integer messages, $m(X) \in \mathcal{R}$ for the polynomial with vector of coefficients $\boldsymbol{m}$ (we will use $\mathbf{m}$ and $m(X)$ interchangeably), and $\tau(\boldsymbol{m})$ for the slot encoding (canonical embedding) of $\boldsymbol{m}$.

Assume ct is an RLWE ciphertext encrypting $\boldsymbol{m}$. (Note that ct could also be obtained by packing multiple LWE ciphertexts, each encrypting an element of $\boldsymbol{m}$.) Let $f(x)$ be a function we want to homomorphically evaluate on ciphertext encrypting $x \in \mathbb{Z}_p$, for an arbitrary p. Our goal is to build an amortized version of the DM/CGGI functional bootstrapping introduced in Appendix B.2, for ciphertexts ct satisfying $\langle \mathsf{ct}, \mathsf{sk} \rangle = \Delta \frac{\boldsymbol{m}}{p} + \boldsymbol{e}$, where $\boldsymbol{m} \in \mathbb{Z}_p^w$, with w being the number of integers bootstrapped at once.

We use the CKKS bootstrapping method as the foundation because it is currently the most efficient amortized bootstrapping method across all FHE schemes. The core idea is to remove the overflows by evaluating a polynomial approximating modular reduction over the encoded raised ciphertext, which keeps the scaled message as is but removes the scaled overflows. Since in CKKS we can scale a message $m \in \mathbb{Z}$ to satisfy $m \approx \sin(m)$, the modular reduction approximation for modulus q_0' is $[m + q_0' I]_{q_0'} = [m]_{q_0'} = \frac{q_0'}{2\pi} \sin(\frac{2\pi m}{q_0'})$.

Applying modulus raising creates overflows in the coefficient domain. Moving to the slots domain allows us to evaluate the polynomial approximating the trigonometric function (corresponding to mod 1) and remove the overflows. In the "standard" CKKS bootstrapping case discussed above, the evaluation of the approximation polynomial leaves the message in place, regardless of its encoding, since the message is scaled down (much smaller than 1). However, in the functional CKKS bootstrapping case, we also want to evaluate an interpolation polynomial that *applies* to the message. Therefore, when we apply the polynomial evaluation, the message needs to also be encoded in slots, the same way as the overflows.

Since in our case the input ciphertext is in RLWE form, the message is encoded in coefficients. Therefore, the first step in the functional bootstrapping is to apply $\mathsf{ModRaise}$, with both the message and overflows coefficient-encoded. Then, we apply the homomorphic encoding CtS, which brings both the message and the overflows in the slots domain, ready for the polynomial evaluation as the next step. Note that for full packing, i.e., to bootstrap N values in the coefficients, we have to use two CKKS ciphertexts (one representing the real part and one the imaginary part, obtained from conjugating the result of the CtS transform) and run the polynomial evaluation on both, then combine them back into one ciphertext. Finally, to return to the RLWE coefficient encoding, we run the homomorphic decoding StC. In other words, the same bootstrapping blueprint as described in Section 2.4 can be used, except for a different polynomial evaluation, which also does function evaluation in this case.

We outline the algorithm for evaluating the functional bootstrapping over ct for an LUT in Algorithm 1. We provide more technical details in the following.

Note that we do not require that the input and output RLWE ciphertexts have the same ciphertext and plaintext moduli. If they do have the same parameters, then some adjustment operations can be avoided.

Adjusting the scaling factor in line 3 of Algorithm 1 requires another level. The homomorphic encoding and decoding CtS and StC can be implemented either as linear transforms consuming a single level each, or using a collapsed FFT-like approach [CCS19], consuming multiple levels each. The latter has the advantage of a substantial decrease in both computational complexity and memory requirement (number of evaluation keys and stored plaintexts).

We evaluate $\mathsf{EvalLUT}$ using the polynomial (5) for the first-order trigonometric Hermite interpolation. For the second-order and third-order trigonometric Hermite interpolations, one can use the polynomials (17) and (19), respectively. To evaluate $\mathsf{EvalLUT}$, we first evaluate $E'(x) = e^{2\pi x i/2^r}$ on a subinterval of $[-1/2^r, 1/2^r]$ using the Chebyshev series interpolation, then use the double-angle formula to increase the interval up to [-1,1]. After that we evaluate the power series in terms of powers of $E'(x)$. Both the Chebyshev and power series are evaluated using the Paterson-Stockmeyer algorithm [PS73, CCS19]. Note that these polynomials (in their general form) have complex coefficients. The

Algorithm 1 Amortized functional bootstrapping for an RLWE ciphertext

Public parameters:
- q: input RLWE ciphertext modulus;
- q'_0: CKKS ciphertext modulus, prime, close to q; $\triangleright$ $q'_{i>0}$ can also be used
- Q'_L: raised CKKS ciphertext modulus (used during bootstrapping);
- Δ: CKKS scaling factor;
- Q: output ciphertext modulus;
- P: output plaintext modulus;
- p: input RLWE plaintext modulus;
- Q': CKKS ciphertext modulus after bootstrapping;
- LUT: coefficients of $R(x)$ for the look-up table evaluation.

1: **procedure** $\text{FuncBT}_{q'_0, Q'_L, \Delta}(\mathsf{ct} \in \mathcal{R}_q^2, \mathsf{LUT})$
2: $\mathsf{ct}_1 \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, q'_0)$ $\triangleright$ Switch ct from q to q'_0. The scaling of the message becomes $\frac{q'_0}{p}$.
3: $\mathsf{ct}_2 \leftarrow \frac{\Delta}{q'_0}\mathsf{ct}_1$ $\triangleright$ Adjust the scaling factor such that we obtain a CKKS ciphertext encoding $\Delta\frac{m(X)}{p} \bmod q'_0$.
4: $\mathsf{ct}_3 \leftarrow \mathsf{ModRaise}(\mathsf{ct}_2, Q'_L)$ $\triangleright$ Encoded vector becomes $\Delta\frac{m(X)}{p} + q'_0 I(X) \bmod Q'_L$
5: $\mathsf{ct}_4 \leftarrow \mathsf{CtS}(\mathsf{ct}_3)$ $\triangleright$ Homomorphic encoding operation, the encoded vector becomes $\Delta\frac{\tau(\boldsymbol{m})}{p} + q'_0\tau(\boldsymbol{I})$
6: $\mathsf{ct}_5 \leftarrow \mathsf{EvalLUT}(\mathsf{ct}_4, \mathsf{LUT})$. $\triangleright$ Homomorphically evaluate the trigonometric interpolation polynomial LUT. The result will encode $\Delta\tau(\boldsymbol{m}') \bmod Q'$, where $\boldsymbol{m}'$ are the coefficients corresponding to $f(\boldsymbol{m})$.
7: $\mathsf{ct}_6 \leftarrow \mathsf{StC}(\mathsf{ct}_5)$ $\triangleright$ Homomorphic decoding operation with an adjusting factor of $Q'/(\Delta QP)$, the encoded vector becomes $\frac{Q'}{Q}\frac{m'(X)}{P} \bmod Q'$
8: $\mathsf{ct}' \leftarrow \mathsf{ModSwitch}(\mathsf{ct}_6, Q')$ $\triangleright$ Switch ct_6 from Q to Q'. The RLWE ciphertext encodes $\frac{Q}{P}m'(X)$.
9: **return** ct'

last step of $\mathsf{EvalLUT}$, which consists of taking the real part of the expression (5) is done via a complex conjugation, which is a cheap homomorphic operation.

Remark 1. For $p = 2$, the first-order Hermite trigonometric interpolation can be written as $R(x) = \frac{1}{2}(f(0) + f(1)) + \frac{1}{2}(f(0) - f(1))\cos(2\pi x)$. This allows a cheaper evaluation of $\cos(2\pi x)$ instead of $E(x)$. Furthermore, using the double-angle formula, we obtain $R(x) = f(1) + (f(0) - f(1))\cos^2(\pi x)$, i.e., the coefficient of $\cos^2(\pi x)$ is integral, thus saving a level of computation.

Correctness. The correctness of the procedure depicted in Algorithm 1 follows from the correctness of the regular CKKS bootstrapping (described in Section 2.4) and the correctness of trigonometric interpolations because we use $\mathsf{EvalLUT}$ instead of $\mathsf{EvalMod}$ in our functional bootstrapping algorithm. We focus here only on $\mathsf{EvalLUT}$ as the correctness of other steps has already been studied elsewhere [CHK+18, CCS19].

Theorem 4. *For an M-th order trigonometric Hermite interpolation of $f(x)$ that satisfies the constraints*

$$R\left(\frac{k}{p}\right) = f(k), \quad R'\left(\frac{k}{p}\right) = 0, \quad \ldots, \quad R^{(M)}\left(\frac{k}{p}\right) = 0, \tag{24}$$

where $k \in [p]$, and a unique trigonometric Hermite interpolation of order M exists, the output error $\|\varepsilon_{out}\|_\infty$ after the polynomial evaluation is bounded by

$$\|\varepsilon_{out}\|_\infty = \left| R\left(\frac{k}{p} + \varepsilon\right) - f(k) \right| \leq C\varepsilon^{M+1}, \tag{25}$$

where C is a constant and $\varepsilon < \frac{1}{2p}$.

Proof. If a unique interpolation $R(x)$ satisfying the constraints (24) exists, then the Taylor series expansion of $R(x)$ at point $\frac{k}{p} + \varepsilon$ will show that the inequality (25) holds. □

Theorem 4 implies that the noise present in the encrypted message will get reduced by the interpolation to the degree determined by M. However, the homomorphic computations in functional bootstrapping contribute to the practically observed noise, too. For example, the Chebyshev series interpolation is used to approximate $E'(x)$ instead of evaluating it directly. Operations that require evaluation keys, such as homomorphic multiplications used in polynomial evaluation, also introduce some noise due to the approximate nature of CKKS. The effect of the CKKS noise depends on the chosen value of the CKKS scaling factor $\Delta = 2^\rho$ (the scaling factor can be increased to make this noise negligible). Hence, the practical noise reduction will depend on the tradeoff between the interpolation noise reduction capability and noise cost of homomorphic computations.

Complexity. Similar to the case of regular CKKS bootstrapping, the total depth L_{fb} needed for functional bootstrapping is $L_{enc} + L_{LUT} + L_{dec} + 1$ (for extra scaling), where L_{enc} and L_{dec} are the levels needed for encoding and decoding, respectively, and L_{LUT} is the depth needed for evaluating the trigonometric Hermite interpolation. For the first-order interpolation, the number of levels consumed by EvalLUT is levels for evaluating $E(x) + \log(p) + 1$. An extra level is added for the second- and third-order interpolations.

The bottleneck operation in most cases is EvalLUT as it requires a large number of homomorphic multiplications. Its complexity depends on the Paterson-Stockmeyer algorithm evaluation which is used both for Chebyshev and power series. The Paterson-Stockmeyer algorithm requires $\left\lceil \sqrt{2d} + \log d \right\rceil + \mathcal{O}(1)$ homomorphic multiplications to evaluate a degree-d polynomial [PS73, CCS19]. The power series evaluation for the first-order trigonometric Hermite interpolation deals with a degree-$(p-1)$ polynomial implying that adding another bit of precision to plaintext modulus p is expected to increase the power series evaluation roughly by a $\sqrt{2}$ factor. In a practical setting, the effect will probably be smaller as other operations in $\mathsf{EvalLUT}$ and other operations in Algorithm 1 will have a much smaller increase in complexity because they do not directly depend on p.

Note that for second- and third-order interpolations, the degree increases to $\frac{3p}{2}$ and $2p-1$, respectively, which implies that the complexity increase from first-order to third-order interpolation should not typically be more than a factor of $\sqrt{2}$. This also means that the computational cost of reducing the noise via the use of the third-order interpolation (instead of the first-order one) is comparable in complexity cost to adding an extra bit to the plaintext space.

5 Amortized Multi-Precision Function Evaluation for RLWE Ciphertexts

The polynomial degree needed for amortized functional bootstrapping using trigonometric Hermite interpolation is proportional to plaintext modulus p. This implies that higher values of p increase the complexity both due to the increased cost of Paterson-Stockmeyer polynomial evaluation (proportional to $\sqrt{p}$) and raised parameters (every doubling of p adds one more CKKS level and slightly increases the scaling factor). Hence, for larger values of p a multi-precision approach based on the blueprint of [GBA21, LMP23] can be more efficient, at least for some classes of functions. An important building block for multi-precision function evaluation is the digit extraction procedure, which can be written in terms of the floor function. In this section, we first describe a procedure for evaluating the floor function, then, show how it can be applied for the multi-precision sign evaluation of messages in $\mathbb{Z}_P^w$, and, finally, we discuss the multi-precision evaluation of an arbitrary function for messages in $\mathbb{Z}_P^w$. Here, we focus on the case of first-order trigonometric Hermite interpolation, noting that all these results easily extend to the higher-order interpolations.

5.1 Homomorphic Evaluation of Floor Function

The floor function evaluation is based on Algorithm 1 for arbitrary function evaluation. Instead of the general power series (5), we use a simpler analytic expression (14). The algorithm for the floor function is outlined in Algorithm 2 (the public parameters are the same as in Algorithm 1). The correctness of evaluating $\mathsf{HomFloor}$ follows from the correctness of FuncBT. The complexity of evaluating $\mathsf{HomFloor}$ is the same as for FuncBT because the power series (14) for $\mathsf{Rmod}_p(x)$ has the same polynomial degree as the general expression (5) for $R(x)$, and the cost of homomorphic subtraction is negligible.

5.2 Homomorphic Evaluation of Multi-Precision Sign Function

We will use the blueprint from [LMP23] that is depicted for DM/CGGI functional bootstrapping in Appendix B.2, Algorithm 5. The outline of the multi-precision sign evaluation algorithm for an input RLWE ciphertext is presented in

Algorithm 2 Homomorphic floor evaluation for an RLWE ciphertext

1: **procedure** $\mathsf{HomFloor}_p(\mathsf{ct} \in \mathcal{R}_Q^2)$
2: $\mathsf{ct}_1 \leftarrow \mathsf{ct} \mod q$ ▷ Extract the RLWE digit encrypting a digit in $\mathbb{Z}_p^w$.
3: $\mathsf{ct}_2 \leftarrow \mathsf{FuncBT}_{q',Q'_L,\Delta}(\mathsf{ct}_1, \mathsf{LUT}\left(\mathsf{Rmod}_p\left(x\right)\right))$ ▷ Perform the functional bootstrapping corresponding to the modulo p function. The returned ciphertext ct_2 encodes $\frac{Q}{P}(\boldsymbol{m} \bmod p)$.
4: **return** $\mathsf{ct} - \mathsf{ct}_2$

Algorithm 3 (the public parameters are the same as in Algorithm 1). The algorithm uses our functional bootstrapping method for the floor function and step function evaluation. Note that in the last iteration of the sign algorithm, where we want to extract the sign of the most significant digit, we use the *unscaled* step function $\frac{2}{p}\mathsf{Rstep}_p\left(x\right)$, where $\mathsf{Rstep}_p\left(x\right)$ is given by (15).

Correctness. The correctness of sign evaluation follows from the correctness of $\mathsf{HomFloor}$ and FuncBT.

Complexity. The multi-precision sign evaluation for encrypted messages in $\mathbb{Z}_P^w$ requires $\lceil \frac{\log P}{\log p} \rceil$ functional bootstrapping operations. The complexity of evaluating $\frac{2}{p}\mathsf{Rstep}_p\left(x\right)$, the last functional bootstrapping invocation, is slightly smaller as it requires the evaluation of a degree-$\frac{p}{2}$ polynomial, as compared to $p-1$ for all other functional bootstrapping invocations.

Algorithm 3 Multi-precision sign evaluation for an RLWE ciphertext

1: **procedure** $\mathsf{HomSign}(\mathsf{ct} \in \mathcal{R}_Q^2)$
2: **while** $Q > q$ **do**
3: $\mathsf{ct}_1 \leftarrow \mathsf{HomFloor}_p(\mathsf{ct})$
4: $\mathsf{ct} \leftarrow \mathsf{ModSwitch}(\mathsf{ct}_1, Q/p)$
5: $Q \leftarrow Q/p, P \leftarrow P/p$
return $\mathsf{FuncBT}_{q',Q'_L,\Delta}(\mathsf{ct}, \mathsf{LUT}\left(\frac{2}{p}\mathsf{Rstep}_p\left(x\right)\right))$

5.3 Homomorphic Evaluation of Multi-Precision Arbitrary Function

When the cost of directly computing $\mathsf{EvalLUT}$ for a large plaintext modulus is high, one can use the multi-precision LUT evaluation approach proposed in [GBA21]. The high-level idea is to decompose the RLWE ciphertexts into digits and then perform (typically different) small-size LUTs against the encrypted digits. Two methods for evaluating multi-precision LUT evaluation are available: tree-based and chain-based [GBA21]. The tree-based approach provides a general functionality, e.g., it can evaluate a random-looking LUT such as S-box [TCBS23], but has an exponential complexity. The chaining-based method provides a smaller complexity but for special (more structured) LUTs, e.g., an LUT for a parity function.

To support the multi-precision LUT evaluation using our CKKS-based method, we need to devise a digit decomposition procedure. The main idea of homomorphic digit decomposition is to decompose an RLWE ciphertext with a large plaintext (ciphertext) modulus into a vector of RLWE ciphertexts with small plaintext (ciphertext) moduli, corresponding to the digit size(s). The procedure is similar to sign evaluation in Algorithm 3, except that all intermediate encrypted digits are kept and the last iteration (step function evaluation) is not performed. The digit decomposition procedure is given in Algorithm 6 in Appendix D.

Tree-Based Evaluation of Large LUTs. We will focus here on the tree-based functionality as it can support the evaluation of an arbitrary large LUT. In the most general case, one needs $d-1+d'\sum_{k=0}^{d-1}p^k$ functional bootstrapping invocations to evaluate a message m in $\mathbb{Z}_P$ represented as $\sum_{k=0}^{d-1}m_k p^k$ [TCBS23], where d' is the number of output digits in $\mathbb{Z}_p$. Here, the term $d-1$ refers to the digit decomposition and the rest accounts for small-size LUTs.

In the case of DM/CGGI bootstrapping, the complexity can be decreased to $d-1+d'+d'\sum_{k=0}^{d-2}p^k$ bootstrapping invocations via the use of *multi-value bootstrapping* where multiple small-size LUTs for the same ciphertext can be evaluated at the cost of one bootstrapping operation [TCBS23]. Our LUT evaluation algorithm can also take advantage of multi-value bootstrapping. The most costly part, finding all $e^{2\pi jxi}$ for $j \in [p]$ in expression (5), can be done once for many LUTs operating on the same ciphertext, hence significanly reducing the amortized cost over many LUTs. While the cost of remaining operations, scalar computations in the Paterson-Stockmeyer algorithm and evaluation of StC, is not neglible (on the order of 10% as compared to full bootstrapping), a significant reduction in runtime complexity can be achieved via this optimization.

6 Functional Bootstrapping for CKKS Ciphertexts

In this section, we consider the input to the functional bootstrapping to be a CKKS ciphertext, meaning the message is encoded via the inverse canonical embedding and resides in the slots domain. We assume the input is a vector of integers $\boldsymbol{m} \in \mathbb{Z}_p^{N/2}$ that is encrypted in a CKKS ciphertext. Note that one can modify the expressions of $R(x)$ obtained from (5), (17) and (19) to address the lack of p-scaling.

Recall the discussion in Section 4 about requiring both the message and overflows to be in the slots domain in order to apply the polynomial evaluation corresponding to the desired function/LUT. Therefore, we need to first apply the homomorphic decoding StC to bring the message to the coefficients domain. Only then we raise the modulus, creating the overflows. Afterwards, we run the homomorphic encoding CtS, to prepare the ciphertext for the polynomial evaluation, which is the next step.

An important optimization in this case is that the costly polynomial evaluation is only performed on a single ciphertext, even in the case of full (real) CKKS packing. For $p = 2$, one can do full complex packing, i.e., $\boldsymbol{m} \in \mathbb{C}^{N/2}$, applying an LUT separately over the real and imaginary parts of the input, but this does not extend to larger p. For larger values of p, we deal with complex evaluations which would require evaluating the polynomials on two ciphertexts.

Algorithm 4 Amortized functional bootstrapping for a CKKS ciphertext

Public parameters:
- q': CKKS ciphertext modulus, prime; $\triangleright$ $q' > q'_{i \geq 1}$ can also be used
- Q'_L: raised CKKS ciphertext modulus (used during bootstrapping);
- Δ: CKKS scaling factor;
- Q': output CKKS ciphertext modulus after bootstrapping;
- LUT: coefficients of $R(x)$ for the look-up table evaluation.

procedure $\textsc{FuncBT'}_{q', Q'_L, \Delta}(\mathsf{ct} \in \mathcal{R}^2_{q'_1}, \mathsf{LUT})$
  $\mathsf{ct}_1 \leftarrow \mathsf{StC}(\mathsf{ct})$ $\triangleright$ Homomorphic decoding operation and potential modulus reduction, the encoded vector becomes $\Delta m(X) \bmod q'_0$
  $\mathsf{ct}_2 \leftarrow \mathsf{ModRaise}(\mathsf{ct}_1, Q'_L)$ $\triangleright$ Encoded vector becomes $\Delta m(X) + q'_0 I(X) \bmod Q'_L$
  $\mathsf{ct}_3 \leftarrow \mathsf{CtS}(\mathsf{ct}_2)$ $\triangleright$ Homomorphic encoding operation, the encoded vector becomes $\Delta\tau(\boldsymbol{m}) + q'_0\tau(\boldsymbol{I})$
  $\mathsf{ct}' \leftarrow \mathsf{EvalLUT}(\mathsf{ct}_3, \mathsf{LUT})$. $\triangleright$ Homomorphically evaluate the trigonometric interpolation polynomial LUT. The result will encode $\Delta\tau(\boldsymbol{m}') \bmod Q'$, where $\boldsymbol{m}'$ are the coefficients corresponding to $f(\boldsymbol{m})$.
  return ct'

Nevertheless, given that the output of the functional bootstrapping remains in the CKKS "approximate" form and can be subjected to further computations, additional noise cleaning procedures may be employed. These can either take the form of a higher-order trigonometric Hermite interpolation in the functional bootstrapping or of a polynomial Hermite interpolation for the modulo p functionality, as discussed in Section 3.5.

7 Implementation and Performance Evaluation

This section describes our experimental setup and provides performance results both for single- and multi-precision function evaluation.

7.1 Parameter Selection

For our experiments, we use the ring dimensions N of 2^{15} and 2^{16} to evaluate LUTs for up to 9 bits. The smaller ring dimension provides a lower latency while the larger ring dimension often results in better throughput. For larger LUTs, we use the ring dimension of 2^{17}. We use full packing, in the sense that the input RLWE ciphertext packs N integer inputs, and we make use of both real and imaginary slots in the CKKS ciphertext.

We use the sparse secret key distribution with the Hamming weight of 192, making sure the maximum CKKS modulus $Q'_L P'$ does not exceed the threshold for the 128-bit work factor. For $N = 2^{15}$, we use the threshold of 767 bits (using Table 4 of [CP19] or Table 3 of [BMTPH21]); for $N = 2^{16}$, we set the threshold to 1,553 (using Table 3 of [BMTPH21]). For $N = 2^{17}$, we used a linear interpolation fitting all values from Table 4 of [CP19] and 1,153 for $N = 2^{16}$ to estimate the threshold as 3,104. It is also possible to use smaller sparse secrets before ModRaise (as in [BCKS24]) or, on the opposite, uniform ternary secrets (as in [BMTPH21]). Note that the difference in throughput would not be significant: the number of levels could either be reduced (by 1 or 2 in the case of small sparse secrets) or increased (by up to 4 levels in the case of uniform ternary secrets). The intermediate extra levels are typically computed using the double-angle formula (just requiring a squaring for each level) and only the computation before the expensive evaluation of the power series for $e^{2\pi x i}$ becomes slower due to a higher number of RNS limbs. Our ballpark estimates suggest that the use of uniform ternary secrets (instead of the Hamming weight of 192) should not decrease the throughput of the FHE evaluation by more than 25% in all practical scenarios (with this number becoming progressively smaller as p increases).

Another important remark is related to the scaling of the messages when working with the CKKS scheme, which was observed previously [CHK+18]. In our implementation, we evaluate the second part of the trigonometric Hermite interpolation (after computing $e^{2\pi x i}$) using the Paterson-Stockmeyer method [PS73] in the power basis. Although the magnitudes of the coefficients of the Hermite interpolation polynomial (and the ratio between the largest and smallest magnitudes of the coefficients) are not too large even for larger plaintext moduli p, passing them through the large recursions in the Paterson-Stockmeyer algorithm exacerbates the magnitudes and ratios of magnitudes, causing overflows in doubles. We observed that the problem can be fully resolved by scaling down the initial coefficients and scaling back up the resulting ciphertext after the power series evaluation. This intermediate scaling also helps reduce the CKKS scaling factor, resulting in improved overall efficiency.

7.2 Implementation

All reported times are obtained via single-threaded execution on a machine with Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz and 64 GB of RAM, running Ubuntu 20.04 LTS, using OpenFHE v1.2.0 compiled with clang++ 12. For the CKKS implementation in OpenFHE, we use the FIXEDMANUAL scaling method and the hybrid key switching method (described in Appendix B.3).

7.3 Experimental Results

Table 1 shows the latency and amortized time for 1-bit to 4-bit, 8-bit, 9-bit, and 12-bit LUT evaluations, which are common LUT sizes in the related literature. Note there is an almost 2x (1.67x) increase in amortized runtime when going from 1 to 2 bits. The reason is that 1-bit evaluation requires only the evaluation of $\cos 2\pi x$ while in the 2-bit case, we need to evaluate $e^{2\pi x i}$, which is equivalent to computing both $\cos 2\pi x$ and $\sin 2\pi x$. However, as soon as we go from 2-bit to 3-bit evaluation, the amortized time increase becomes more modest (1.11x) as the only difference is the degree in the Hermite interpolation polynomial, which increases the complexity by $\sqrt{2}$ with doubling p. For smaller p, this increase is smaller than $\sqrt{2}$ because other (p-independent) parts of bootstrapping still play a significant role. But for larger p, the power series evaluation becomes dominant and the runtime increase with doubling p progressively gets closer to $\sqrt{2}$, as predicted by the complexity analysis of Section 4. Note that the increase from 9 to 12 bits is even higher than $2\sqrt{2}$ because of secondary factors, such as increased N and extra RNS limbs.

If one would use the Boolean method (as in [BCKS24]) to evaluate larger LUTs, they would need to use a multi-precison approach, such as the tree-based method discussed in Section 5.3, which incurs exponential complexity in general cases. For example, evaluating an 8-bit S-box (with multi-value functional bootstrapping) using the tree-based approach would require 1,031 1-bit LUT evaluations (see Section 5.3 for the complexity estimation expression). In our case, the cost of evaluating an 8-bit LUT for AES S-box is only 4x higher than evaluating a 1-bit LUT, implying a speed-up of about 250x over the tree-based bit-level approach.

$\log p$	$\log Q$	N	$\log(Q'_L P')$	# limbs (enc, dec)	# limbs HKS	Time (s)	Amtz. time (ms)
1	33	2^{15}	768	16 (3,3)	4	6.151	0.187
2	35	2^{16}	1105	23 (5,5)	6	20.422	0.312
3	36	2^{16}	1092	22 (4,4)	5	22.661	0.345
4	40	2^{16}	1280	23 (4,4)	6	25.874	0.395
8	47	2^{16}	1535	25 (3,3)	6	49.025	0.748
9	49	2^{16}	1514	26 (3,3)	4	69.330	1.058
12	58	2^{17}	2458	31 (4,4)	11	569.54	4.345

Table 1. Experiments for the evaluation of an LUT (floor) for an RLWE ciphertext with plaintext modulus p and ciphertext modulus Q. Here, $\log(Q) = \log(\Delta) = \log(q'_0)$ and $p = P$. By $\log(Q'_L P')$ we refer to the number of bits in the largest CKKS modulus, which includes all RNS limbs for the leveled computation (multiplicative depth + 1) and all RNS limbs used in hybrid key switching. A single RNS limb is left after the functional bootstrapping. More detailed information is provided in Table A3.

Table 2 presents the timing results for multi-precision sign evaluation for 12-bit, 21-bit, and 32-bit encrypted messages. Note that the amortized runtime improves as we increase the digit size from 1 to 6 bits for 12-bit messages and from 1 to 7 bits for 21-bit bit messages, which follows from the analysis for Table 1. This implies that our method with $p > 2$ for digits always outperforms the Boolean approach developed in [BCKS24] in both single- and multi-precision scenarios (even for special-purpose functions such as sign evaluation). We also show the timing results for evaluating the sign of 32-bit messages with 8-bit digits

$\log P$	$\log Q$	$\log p$	$\log q$	$\log_2(Q'_L P')$	Time (s)	Amtz. time (ms)
12	46	1	35	870	146.38	2.233
12	45	2	35	1105	124.13	1.894
12	45	3	36	1164	92.34	1.409
12	48	4	40	1140	77.91	1.119
12	48	6	42	1368	62.17	0.949
21	55	1	35	870	267.21	4.077
21	55	3	37	1114	159.54	2.434
21	57	7	43	1495	111.55	1.704
32	71	8	47	1535	197.93	3.022

Table 2. Experiments for the multi-precision evaluation of the sign function on an RLWE ciphertext with plaintext modulus P and ciphertext modulus Q. The digit plaintext modulus is p and the digit ciphertext modulus size is q. The ring dimension, equal to the number of RLWE slots, is $N = 2^{16}$. $\log(q) = \log(\Delta) = \log(q'_0)$.

as this scenario is useful for practical applications dealing with 32-bit arithmetic. Note that the amortized runtime for sign evaluation is roughly the product of the number of digits encrypting messages in $\mathbb{Z}_p^w$ by the runtime of the $\log p$-bit LUT in Table 1, as expected from our complexity analysis.

For simplicity, we only implemented the case when there is a single digit of size p whose bit-size $\log p$ divides the plaintext modulus bit-size $\log P$, but we remark that this is not necessary, and multi-precision evaluations with different digit sizes (that do not divide $\log P$) can be implemented.

To port our results from RLWE inputs to DM/CGGI inputs, we need to add the time for the ring packing procedure, which yields an amortized time of 0.056 ms; see Appendix A.2.

8 Concluding Remarks

Our performance evaluation suggests that the general functional bootstrapping method developed in this work starts outperforming the conventional DM/CGGI method when the number of slots reaches the order of thousands or even hundreds for LUTs of size larger than 8 bits. For many practical scenarios that require the simultaneous evaluation of hundreds/thousands of slots, our proposed RLWE-based method can replace the DM/CGGI solution. Moreover, our method based on CKKS-style bootstrapping achieves significantly better complexity and concrete amortized time than all prior methods based on BFV-style bootstrapping.

Although we focus here on a vectorized instantiation of a DM/CGGI-style cryptosystem, where the input scheme is RLWE and CKKS is only used for functional bootstrapping, we envision that this method can also be used to develop an end-to-end CKKS capability for fixed-precision arithmetic, which can evaluate both polynomial and discontinuous functions, providing exact (rather than approximate) results. The Hermite interpolations (both trigonometric and polynomial) can be used to evaluate functions and control noise (keeping it below a certain threshold). This exact CKKS could benefit from the ideas proposed in [DMPS24] and [BCM+24], and use our functional bootstrapping method and Hermite interpolations as essential building blocks. Finally, a BFV-style decryption can be employed in this exact CKKS formulation to achieve IND-CPA^D security without expensive noise flooding. We see the design and efficient instantiation of such exact CKKS FHE scheme as an interesting research problem.

An interesting generalization was presented by Chung et al. [CKKL24], which allows for the slotwise evaluation of multiple independent LUTs, including multivariable LUTs, within the same ciphertext. One can adapt their method to our case for evaluating different multivariable LUTs in a slotwise manner, via matrix multiplication, which is another topic for future research.

The main limitation of our method is the high computational complexity of functional bootstrapping for large p (though this complexity is lower than in all prior methods). For instance, our single-precision experimental results are limited to 12-bit LUTs as the scaling factor approaches the limit for 64-bit modular operations and the complexity of polynomial evaluation becomes high. We view our implementation as a proof of concept, with potential for further improvement to efficiently support the evaluation of larger LUTs. We leave the development of such optimizations as a research problem for future studies.

Acknowledgements. The authors would like to thank Zeyu Liu and Yunhao Wang for helpful discussions on multi-precision sign evaluation.

References

AAB+22. Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. `https://eprint.iacr.org/2022/915`.

ABMP24. Andreea Alexandru, Ahmad Al Badawi, Daniele Micciancio, and Yuriy Polyakov. Application-aware approximate homomorphic encryption: Configuring FHE for practical use. Cryptology ePrint Archive, Paper 2024/203, 2024.

AP23. Ahmad Al Badawi and Yuriy Polyakov. Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149, 2023.

BBB+23. Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (T)FHE. *J. Cryptol.*, 36(3):28, 2023.

BCK+23. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. Hermes: Efficient ring packing using mlwe ciphertexts and application to transciphering. In *Advances in Cryptology – CRYPTO*

2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part IV, page 37–69, Berlin, Heidelberg, 2023. Springer-Verlag.

BCKS24. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 94–123, Cham, 2024. Springer Nature Switzerland.

BCM+24. Jean-Philippe Bossuat, Anamaria Costache, Christian Mouchet, Lea Nürnberger, and Juan Ramón Troncoso-Pastoriza. Practical q-IND-CPA-d-secure approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2024/853, 2024.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

BKSS24. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In *ASIACRYPT 2024*. Springer-Verlag, 2024.

BMTPH21. Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617, Cham, 2021. Springer International Publishing.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 868–886. Springer, 2012.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 34–54. Springer, 2019.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachéne. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 3–33. Springer, 2016.

CHK+18. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 360–384, Cham, 2018. Springer International Publishing.

CJP21. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*, pages 1–19. Springer, 2021.

CKK+19. Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 415–445, Cham, 2019. Springer International Publishing.

CKK20. Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 221–256, Cham, 2020. Springer International Publishing.

CKKL24. Heewon Chung, Hyojun Kim, Young-Sik Kim, and Yongwoo Lee. Amortized large look-up table evaluation with multivariate polynomials for homomorphic encryption. Cryptology ePrint Archive, Paper 2024/274, 2024.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 670–699, Cham, 2021. Springer International Publishing.

CP19. Benjamin R. Curtis and Rachel Player. On the feasibility and impact of standardising sparse-secret LWE parameter sets for homomorphic encryption. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'19, page 1–10, New York, NY, USA, 2019. Association for Computing Machinery.

DM15. Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015.

DMPS24. Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. BLEACH: cleaning errors in discrete computations over CKKS. *J. Cryptol.*, 37(1):3, 2024.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

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

Gen09a. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

Gen09b. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

GV23. Robin Geelen and Frederik Vercauteren. Bootstrapping for BGV and BFV revisited. *J. Cryptol.*, 36(2):12, 2023.

HK20. Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 364–390. Springer, 2020.

HL24. Jincheol Ha and Jooyoung Lee. Patching and extending the WWL+ circuit bootstrapping method to FFT domains. Cryptology ePrint Archive, Paper 2024/1318, 2024.

KPP22. Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. In *Topics in Cryptology – CT-RSA 2022*, page 120–144, Berlin, Heidelberg, 2022. Springer-Verlag.

KS22. Kamil Kluczniak and Leonard Schild. FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):501–537, Nov. 2022.

LHH+21. Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1057–1073. IEEE, 2021.

LM21. Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 648–677. Springer, 2021.

LMP23. Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*, page 130–160, Berlin, Heidelberg, 2023. Springer-Verlag.

LMS24. Dongwon Lee, Seonhong Min, and Yongsoo Song. Functional bootstrapping for packed ciphertexts via homomorphic LUT evaluation. Cryptology ePrint Archive, Paper 2024/181, 2024.

LW23. Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{o}(1)$ polynomial multiplications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part VI*, volume 14443 of *Lecture Notes in Computer Science*, pages 101–132. Springer, 2023.

LW24. Zeyu Liu and Yunhao Wang. Relaxed functional bootstrapping: A new perspective on BGV/BFV bootstrapping. Cryptology ePrint Archive, Paper 2024/172, 2024.

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

MSM+22. Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H. P. Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609, 2022.

PS73. Michael S Paterson and Larry J Stockmeyer. On the number of non-scalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

SV65. A. Sharma and A. K. Varma. Trigonometric interpolation. *Duke Mathematical Journal*, 32(2):341 – 357, 1965.

TCBS23. Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. A homomorphic aes evaluation in less than 30 seconds by means of tfhe. In *WAHC '23*, page 79–90, New York, NY, USA, 2023. Association for Computing Machinery.

Var69. A.K. Varma. Trigonometric interpolation. *Journal of Mathematical Analysis and Applications*, 28(3):652–659, 1969.

Var73. A. K. Varma. Hermite-Birkhoff trigonometric interpolation in the (0, 1, 2, m) case. *Journal of the Australian Mathematical Society*, 15(2):228–242, 1973.

Zam22. Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. `https://github.com/zama-ai/tfhe-rs`.

A Comparison with Other Methods

This section compares our results with prior work. There are two main directions of obtaining the homomorphic evaluation of arbitrary LUTs. The first one is performing functional bootstrapping, and the second one is performing leveled computations and potentially bootstrapping.

A.1 Comparison with the Boolean CKKS method

Our 1-bit LUT evaluation parallels the bit-level CKKS bootstrapping developed in [BCKS24] for bootstrapping DM/CGGI ciphertexts, although with a couple of differences. First, their implementation uses smaller sparse secrets when adding overflows during $\mathsf{ModRaise}$. Second, their implementation uses the base ring packing in HERMES to transform from LWE to RLWE and the functional bootstrapping without the initial CtS transform (which seems to be performing the costly polynomial evaluation on a single ciphertext even when using full packing, instead of on two ciphertexts as in our case). Their reported time for their full-slot complex functional bootstrapping (without the ring packing) is 1.54 seconds for 2^{14} gates. The resulting amortized time is 0.094 ms (this only includes a single linear transform) versus our amortized time of 0.187 ms, which implies that both implementations have similar efficiency.

A.2 Comparison with Other Methods for Functional Bootstrapping

There are two main methods in the literature for performing functional bootstrapping: DM/CGGI-based, which bootstraps one number at a time, and BFV-based, which supports amortized functional bootstrapping.

For the scenarios where we compare our runtimes with the runtimes for the functional bootstrapping of LWE ciphertexts, we add the amortized base ring packing time to our amortized time to account for the conversion of N LWE ciphertexts to an RLWE ciphertext, as the latter was used as an input in our experiments. Note that the amortized ring packing time is estimated using the runtimes from [BCK$^+$23], which are more conservative than the ring packing runtimes in the later work of the authors [BCKS24]. For $N = 2^{15}$, the ring packing time is 1.85 s, which is computed as the runtime for the ring dimension of 2^{12} (0.231 s) multiplied by 2^3. For $N = 2^{16}$, the ring packing time is twice larger, i.e., 3.7 s. The corresponding amortized time for both is 0.056 ms.

Comparison with DM/CGGI functional bootstrapping. Table A1 compares our experimental results with the CGGI-based results in [LMP23] for single- and multi-precision LUT evaluation (note that uniform ternary secret key distribution was used in [LMP23]). For the fairness of comparison, we reran the experiments from [LMP23] using OpenFHE v1.2.0 and clang++ 12. For multi-precision sign evaluation, we observe a speed-up of three orders of magnitude for our method due to the SIMD capability of CKKS, enabling it to bootstrap 2^{16} numbers at once.

We also include the results from [TCBS23] in Table A1, which evaluate LUTs using CGGI functional bootstrapping (with uniform binary secret key distribution) augmented with multi-value bootstrapping. In particular, for LUTs such as S-box or XOR (for the AES algorithm), they provide runtimes for both a direct 8-to-8 bit LUT evaluation and tree-based multiprecision LUT evaluation. For the 8-to-8 bit LUT, with the method of Trama *et al.* it is more efficient to decompose the LUT into smaller LUTs, while our approach is still very efficient for a direct evaluation of an 8-to-8 bit LUT. The amortization makes our results 400x times more efficient than the evaluation of S-box in [TCBS23]. Note that our amortized runtime for 8-bit LUT evaluation is 1,900x smaller than the direct evaluation of the 8-bit LUT using the CGGI approach. It is worth pointing out that the runtime for our method increases by roughly a factor of $\sqrt{2}$ when doubling p while the DM/CGGI method scales exponentially (the ring dimension doubles every time p is doubled).

Comparison with multi-precision method based on CGGI circuit bootstrapping. Another multi-precision approach for evaluating large LUTs using the CGGI/TFHE method is via circuit bootstrapping [BBB$^+$23]. This method is implemented in the tfhe-rs library [Zam22]. The high-level idea is to (1) decompose a large-precision LWE ciphertext into LWE ciphertexts for each encrypted bit of the message using homomorphic digit extraction, (2) convert the LWE ciphertexts into RGSW ciphertexts using circuit bootstrapping to enable leveled multiplications, and (3) evaluate a CMUX tree involving (many) multiplications. This approach performs better for the CGGI cryptosystem than the tree-based

[5] Corresponds to using the multiprecision approach with 4-bit LUTs

Table A1. Comparison of our single- and multi-precision results with DM/CGGI functional bootstrapping.

Function	$\log P$	[TCBS23] runtime (ms)	[LMP23] runtime (ms)	Our amtz. runtime (ms)
EvalLUT	2	7	92	0.368
EvalLUT	3	15	243	0.401
EvalLUT	4	29	–	0.451
EvalLUT	8	1,500/300[5]	–	0.804
HomSign	8	–	671	0.804
HomSign	12	–	1,367	1.175
HomSign	21	–	3,451	2.490

approach when the precision reaches 10-11 bits [BBB$^+$23]. For the ranges of $\log P$ considered in our paper, the circuit bootstrapping is often the bottleneck operation. It is hard to compare this approach with our method directly because the cost of the CGGI circuit bootstrapping method depends on the complexity of the evaluated LUT, but we can at least compare the cost of circuit bootstrapping with our LUT evaluation runtime to get a ballpark estimate. To the best of our knowledge, the state-of-the-art results for circuit bootstrapping compatible with 8-bit LUTs are presented in [HL24]. In Table 5, the authors report the timing results for WOPBS_4_4, which is the tfhe-rs parameter set supporting 8-bit LUTs. The runtime they report for tfhe-rs is 625 ms while their improved circuit bootstrapping takes 87 ms [HL24]. Note that 87 ms is still two orders of magnitude slower than the amortized 8-bit LUT evaluation using our method. We also want to highlight that a circuit bootstrapping capability could potentially be built based on our functional bootstrapping method and leveled computations in CKKS, which in some settings may result in more efficient LUT evaluation than using functional bootstrapping directly. But we leave the development of such circuit bootstrapping capability as a topic for future research, as it is not directly related to functional bootstrapping.

Comparison with BFV-based functional bootstrapping. Liu and Wang [LW23] proposed a method of batch evaluating an LUT over a number of LWE ciphertexts by switching to BFV and evaluating a polynomial of degree q, the BFV plaintext modulus (or, equivalently, LWE ciphertext modulus). For LWE plaintext moduli p of up to 9 bits, the corresponding q is 65,537. For p up to 12 bits, the corresponding q is 786,433. Note that the BFV scheme requires special moduli for q, which complicates its use for multi-precision LUT and sign evaluation (we are not aware of any multi-precision extensions of this method). Our method via CKKS involves evaluating a polynomial of a much smaller degree: for LWE plaintext modulus p, the trigonometric Hermite interpolation requires evaluating a polynomial of degree $p-1$ over the approximation of $e^{2\pi x i}$, with the latter achieved by evaluating a polynomial of degree 58 followed by 2 to 4 double-angle-formula iterations. Specifically, in our implementation, for $p = 2^9$, we evaluate a polynomial of degree 58, two squarings, and a polynomial of degree 511, while for $p = 2^{12}$, we evaluate a polynomial of degree 58, two squarings, and a polynomial of degree 4095.

Liu and Wang [LW24] propose an optimization of [LW23] by relaxing the correctness notion for the values outside of the points of interest, which reduces the degree of the polynomial for the values of p smaller than 2^9, i.e., the maximum p for a given value of q. The degree in this case becomes roughly $p{\cdot}r$, where r is the error bound, which is equal to 128 for $q = 65,537$ for the secret key distribution choice in [LW24] (sparse secrets with the Hamming weight of 512). Effectively, this replaces the degree q with $p \cdot r$ and allows one to choose optimal parameters for a given value of p rather than special modulus q. In terms of complexity, our method has the same advantage as w.r.t. [LW23], but the concrete benefit of our method becomes less significant for smaller values of p, when the contribution of power series evaluation is smaller.

We also mention the work of Lee et al [LMS24], which homomorphically evaluates an arbitrary LUT over a BFV ciphertext by using a method based on conventional BFV/BGV bootstrapping working with plaintext space $\mathbb{Z}_p^s$, where p is prime. Their experiments focus on input BFV plaintext moduli of $p < 700$ (≈ 9.5 bits) and $p < 17000$ (≈ 14 bits) and output plaintext moduli of 17^4 (≈ 16 bits) and 17^5 (≈ 20.5 bits) for the delta or sign functions. Due to the plaintext algebra restrictions for $p = 17$, their number of slots for the reported results is only 16. However, the number of levels remaining for computation in BFV is 11, whereas we leave only 1 level (though we could also add CKKS levels to our bootstrapping, paying only a modest price, i.e., below 2x, in complexity).

Table A2 compares the online amortized runtimes of our method for general functional bootstrapping via CKKS with the methods of [LW23] and [LW24] via BFV. The times reported for [LW23] are from Tables 3 and 4 in their paper. We note that we implemented their method in OpenFHE to check any differences in runtime due to the underlying library and we obtained similar results (for the 9-bit LUT, 6.7 ms in SEAL compared to 5.8 ms in OpenFHE). The times reported for [LW24] are taken from Figure 4 from [LW24][6]. The times reported for [LMS24] are taken from Table 3 in their paper.

The relevant comparison is with [LW23] and [LW24], as the amortization is done over a number of slots of similar magnitude. Our method exhibits improvements in throughput ranging from 3.2x to 8.8x, with the speed-up increasing with p (for $\log p \geq 3$).

A.3 Discussion on Leveled Methods for LUT Evaluation

We presented the method of using *trigonometric Hermite interpolation* integrated into the functional bootstrapping process, allowing efficient evaluation of arbitrary functions with noise reduction. We also mentioned the *polynomial Hermite interpolation* approach to evaluate the function f separately from the

[6] At the moment of writing, Figure 4 from [LW24] contained some typos and we included the timings communicated by the authors.

Table A2. Comparison of our functional bootstrapping runtimes with BFV-based methods. For our implementation, we report the base runtimes for BFV-input-ciphertext functional bootstrapping as well as the runtimes with the ring packing for DM/CGGI input ciphertexts.

$\log p$	[LW23] runtime (ms)	[LW24] runtime (ms)	[LMS24] runtime (ms)	Our amtz. runtime (ms)
1	4.7	1.3	–	0.254/0.310
3	6.7	1.3	–	0.345/0.401
4	6.7	1.5	–	0.395/0.451
8	6.7	5.3	–	0.748/0.804
9	6.7	6.7	2,960	1.058/1.114
12	39.1	–	10,760	4.345/4.401

bootstrapping process. In this section, we compare these two methods in terms of computational efficiency and precision, and discuss other leveled methods for Look-Up Table (LUT) evaluation.

When the function f is evaluated separately from the bootstrapping process, a polynomial $\bar{R}(x)$ of degree $2p-1$ (for first-order interpolation) is used to approximate f. This method requires a higher-degree polynomial than $R(x)$ (with the basis $E(x)$), which increases computational complexity and noise accumulation. Moreover, it has to invoke regular CKKS bootstrapping for deep computations, which requires a larger scaling factor than the functional bootstrapping method (see the discussion in Section 3.4 of [BCKS24] for $p=2$; the gap gets higher as p is increased). However, this method is independent of the bootstrapping process and can be suitable for shallow computations without bootstrapping.

Another approach is presented in [CKKL24], where Chung *et al.* proposed a technique for evaluating LUTs using the CKKS scheme with custom encoding. This method uses polynomial Hermite approximations to evaluate functions directly on encrypted data, leveraging the homomorphic properties of CKKS. However, the custom encoding complicates the application of multiplications, making it less straightforward when handling more complex computations. We mention that the results reported in Table 2 in [CKKL24] are obtained using a GPU, which is typically faster than a CPU by at least an order of magnitude, and yet the speed-up they obtain for an 8-to-8 bit S-box evaluation is only five times faster than our method (0.15 ms versus 0.75 ms). Another drawback is that large parameters (ring dimension N and ciphertext modulus Q) need to be used to support both leveled LUT computation and subsequent bootstrapping. The advantage of our method is that potentially smaller parameters can be used as our method provides simultaneous arbitrary function evaluation and bootstrapping.

In contrast to both of these methods, integrating trigonometric Hermite interpolation directly into the bootstrapping process allows for functional bootstrapping, where both noise reduction and function evaluation are performed simultaneously. In this method, we evaluate a polynomial of degree $p-1$ on top of evaluating $e^{2\pi i x}$ (which can be thought of as part of the original bootstrapping process). The polynomial degree is reduced from $2p-1$ to $p-1$, resulting in faster computation and less noise accumulation.

B More Preliminaries

B.1 LWE Modulus Switching

Lemma 1 (Modulus Switching). *Let $(\boldsymbol{a}, b) \in \mathbb{Z}_q^{n+1}$ be an LWE encryption of a message $m \in \mathbb{Z}_p$ under secret key $\boldsymbol{s} \in \mathbb{Z}^n$ with ciphertext modulus q and noise bound $|\mathsf{Dec}_{\boldsymbol{s}}(\boldsymbol{a}, b) - (q/p)m| < \beta$. Then, for any modulus q', the rounded ciphertext $(\boldsymbol{a}', b') = \lceil (q'/q) \cdot (\boldsymbol{a}, b) \rfloor$ is an encryption of the same message m under $\boldsymbol{s}$ with ciphertext modulus q' and noise bound $|\mathsf{Dec}_{\boldsymbol{s}}(\boldsymbol{a}', b') - (q'/p)m| < (q'/q)\beta + \beta''$, where $\beta'' = \frac{1}{2}(\|\boldsymbol{s}\|_1 + 1)$.*

In practice, when the input ciphertext is sufficiently random, or when modulus switching is performed by *randomized* rounding, it is possible to replace the additive term β'' with a smaller probabilistic bound $O(\|\boldsymbol{s}\|_2)$. For uniformly random ternary keys $\boldsymbol{s} \in \{0, 1, -1\}^n$, this is $\beta'' \approx O(\sqrt{n})$. For sparse secret keys with a hamming weight h, it is $\beta'' \approx O(\sqrt{h})$.

B.2 Functional Bootstrapping and Multi-Precision Sign Evaluation using DM/CGGI Cryptosystems

A key feature of a DM/CGGI cryptosystem is that it allows to perform certain homomorphic computations (described by an LUT) on ciphertexts during bootstrapping at no additional cost. We will use the generalization of the DM/CGGI bootstrapping procedure presented in [LMP23]. The functional bootstrapping algorithm is parameterized by

- a dimension n and (input ciphertext) modulus q, where q is a power of 2,
- a secret key $\boldsymbol{s} \in \mathbb{Z}^n$, which must be a short vector.
- a large ciphertext modulus Q' used internally to the bootstrapping procedure, and which is not required to be a power of 2,
- an output ciphertext modulus Q, which we set to a power of 2 possibly different from q, and
- an LUT function $f \colon \mathbb{Z}_q \to \mathbb{Z}$ which must satisfy the negacyclic constraint

$$f(x + q/2) = -f(x). \tag{A1}$$

The bootstrapping procedure also uses a bootstrapping key, which is computed from $\boldsymbol{s}$, but can be made public. Since this bootstrapping key is only used internally by the bootstrapping procedure, we omit it from the notation.

On input an LWE ciphertext $(\boldsymbol{a}, b) \in \mathbb{Z}_q^{n+1}$, the DM/CGGI bootstrapping procedure first computes an LWE ciphertext $(\boldsymbol{c}', d') \in \mathbb{Z}_{Q'}^{n+1}$ such that

$$\mathsf{Dec}_{\boldsymbol{s}}(\boldsymbol{c}', d') = f'(\mathsf{Dec}_{\boldsymbol{s}}(\boldsymbol{a}, b)) + e' \pmod{Q'},$$

where the noise bound $|e'| \leq \beta'$ depends only on the computation performed during bootstrapping (and not the input ciphertext), and

$$f'(x) = \left\lceil \frac{Q'}{Q} \cdot f(x) \right\rfloor$$

is a scaled version of f still satisfying the negacyclic condition (A1). Then, modulus switching is applied to $(\boldsymbol{c}', d')$ to obtain a ciphertext $(\boldsymbol{c}, d) = \left\lceil \frac{Q}{Q'} (\boldsymbol{c}', d') \right\rfloor \in \mathbb{Z}_Q^{n+1}$ modulo Q such that

$$\mathsf{Dec}_{\boldsymbol{s}}(\boldsymbol{c}, d) = f(\mathsf{Dec}_{\boldsymbol{s}}(\boldsymbol{a}, b)) + e \pmod{Q}$$

where $|e| < \beta = (Q/Q')\beta + \beta''$ is the noise bound from Lemma 1.

Similarly to [LMP23], we express the bootstrapping invocation for a given function f as $\mathsf{Boot}[f](\boldsymbol{a}, b)$.

Liu *et al.* and similar works show how this functional programming capability for negacyclic functions can be used to build arbitrary function evaluation in $\mathbb{Z}_p$ [CLOT21, KS22, LMP23]. The cost is at least two functional bootstrapping operations (the first one is needed to handle the negacyclic requirement). Further, a multi-precision approach based on digit extraction (floor function) and arbitrary function evaluation in $\mathbb{Z}_p$ was derived to evaluate large arbitrary functions in $\mathbb{Z}_P$, where P is the large plaintext modulus P required for a given application [GBA21, LMP23].

Of special practical interest is the multi-precision sign evaluation capability due to its linear increase of complexity with $\log P$ [LMP23]. The high-level algorithm for evaluating the multi-precision sign function is depicted in Algorithm 5. Here, HomFloor is an LUT evaluation for the floor/digit decomposition function (requires two functional bootstrapping operations) and $\mathsf{Boot}[f_{MSB}]$ is the regular MSB function evaluation (only one DM/CGGI bootstrapping is needed).

Algorithm 5 Algorithm for Multi-precision Homomorphic Sign Computation [LMP23]

1: **procedure** $\mathsf{HomSign}(Q, (\boldsymbol{c}, d))$
2:     **while** $Q > q$ **do**
3:         $(\boldsymbol{c}, d) \leftarrow \mathsf{HomFloor}(Q, (\boldsymbol{c}, d))$
4:         $(\boldsymbol{c}, d) \leftarrow \left\lceil \frac{\alpha}{q} \cdot (\boldsymbol{c}, d) \right\rfloor$ ▷ $\alpha = q/p$, for p the plaintext modulus of the digit
5:         $Q \leftarrow \alpha Q / q$
6:     $d \leftarrow d + \beta$
7:     $(\boldsymbol{a}, b) \leftarrow (q/Q) \cdot (\boldsymbol{c}, d)$
8:     $(\boldsymbol{c}, d) \leftarrow (-\mathsf{Boot}[f_{MSB}](\boldsymbol{a}, b)) \pmod{Q}$
9:     **return** $(\boldsymbol{c}, d)$

B.3 CKKS Scheme in RNS

We first provide the CKKS algorithms related to evaluation (we will introduce the details specific to the RNS instantiation later in this section):

- $\mathsf{KeySwitchGen}_{\mathsf{sk}}(\boldsymbol{s}')$. For a power-of-two P' that corresponds to the auxiliary modulus, sample a random $\boldsymbol{a}'_k \leftarrow \mathcal{R}_{P'Q'_L}$ and error $\boldsymbol{e}'_k \leftarrow \chi_{\mathsf{err}}$. For a predefined power-of-two base ω, output the switching key as

$$\mathsf{swk} = (\mathsf{swk}_0, \mathsf{swk}_1) = \left(\{\boldsymbol{b}'_k\}_{k=0}^{\mathsf{dnum}-1}, \{\boldsymbol{a}'_k\}_{k=0}^{\mathsf{dnum}-1}\right) \in \mathcal{R}_{P'Q'_L}^{2\times\mathsf{dnum}},$$

  where $\boldsymbol{b}'_k \leftarrow -\boldsymbol{a}'_k \cdot \boldsymbol{s} + \boldsymbol{e}'_k + P' \cdot \mathcal{PW}_L(\boldsymbol{s}')_k \pmod{P'Q'_L}$ and $\mathsf{dnum} = \lceil \log_\omega(Q'_L) \rceil$. Set $\mathsf{evk} \leftarrow \mathsf{KeySwitchGen}_{\mathsf{sk}}(\boldsymbol{s}^2)$. Set $\mathsf{rk}^{(\kappa)} \leftarrow \mathsf{KeySwitchGen}_{\mathsf{sk}}(\boldsymbol{s}^{(\kappa)})$.
- $\mathsf{KeySwitch}_{\mathsf{swk}}(\mathsf{ct})$. For $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}^2_{Q'_\ell}$, $\mathsf{swk} = (\mathsf{swk}_0, \mathsf{swk}_1)$ [7] output

$$\left(\boldsymbol{c}_0 + \left\lceil \frac{\langle \mathcal{WD}_\ell(\boldsymbol{c}_1), \mathsf{swk}_0 \rangle}{P'} \right\rfloor, \left\lceil \frac{\langle \mathcal{WD}_\ell(\boldsymbol{c}_1), \mathsf{swk}_1 \rangle}{P'} \right\rfloor\right) \pmod{Q'_\ell}.$$

  To keep the noise from key switching small, we can take $P' \approx \omega$.
- $\mathsf{CAdd}(\mathsf{ct}, x)$. For $\mathsf{ct} = (\boldsymbol{b}, \boldsymbol{a}) \in \mathcal{R}^2_{Q'_\ell}$ with scaling factor $\Delta^{\ell'}$ and scalar $x \in \mathbb{C}^n$, first encode x with same scaling factor $\boldsymbol{m} = \mathsf{Encode}(x, \Delta^{\ell'})$, and output $\mathsf{ct}_{\mathsf{cadd}} \leftarrow (\boldsymbol{b} + \boldsymbol{m}, \boldsymbol{a}) \pmod{Q'_\ell}$.
- $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$. For $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}^2_{Q'_\ell}$, output $\mathsf{ct}_{\mathsf{add}} \leftarrow \mathsf{ct}_1 + \mathsf{ct}_2 \pmod{Q'_\ell}$.
- $\mathsf{CMult}(\mathsf{ct}, x)$. For $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}^2_{Q'_\ell}$ and scalar $x \in \mathbb{C}^n$, first encode x, $\boldsymbol{m} = \mathsf{Encode}(x, \Delta)$ and output $\mathsf{ct}_{\mathsf{cmult}} \leftarrow (\boldsymbol{c}_0 \cdot \boldsymbol{m}, \boldsymbol{c}_1 \cdot \boldsymbol{m}) \pmod{Q'_\ell}$.
- $\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$. For $\mathsf{ct}_i = (\boldsymbol{b}_i, \boldsymbol{a}_i) \in \mathcal{R}^2_{Q'_\ell}$, let $(\boldsymbol{d}_0, \boldsymbol{d}_1, \boldsymbol{d}_2) = (\boldsymbol{b}_1 \cdot \boldsymbol{b}_2, \boldsymbol{a}_1 \cdot \boldsymbol{b}_2 + \boldsymbol{a}_2 \cdot \boldsymbol{b}_1, \boldsymbol{a}_1 \cdot \boldsymbol{a}_2) \pmod{Q'_\ell}$. Output

$$\mathsf{ct}_{\mathsf{mult}} \leftarrow (\boldsymbol{d}_0, \boldsymbol{d}_1) + \mathsf{KeySwitch}_{\mathsf{evk}}(0, \boldsymbol{d}_2) \pmod{Q'_\ell}.$$

- $\mathsf{Rot}_{\mathsf{rk}^{(5^\kappa)}}(\mathsf{ct}, \kappa)$. For $\mathsf{ct} = (\boldsymbol{b}, \boldsymbol{a}) \in \mathcal{R}^2_{Q'_\ell}$ and rotation index κ, output

$$\mathsf{ct}_{\mathsf{rot}} \leftarrow (\boldsymbol{b}^{(5^\kappa)}, 0) + \mathsf{KeySwitch}_{\mathsf{rk}^{(5^\kappa)}}(0, \boldsymbol{a}^{(5^\kappa)}) \pmod{Q'_\ell}.$$

- $\mathsf{Rescale}(\mathsf{ct}, \Delta^{\ell'})$. For a ciphertext $\mathsf{ct} \in \mathcal{R}^2_{Q'_\ell}$ and a rescaling factor $\Delta^{\ell'}$, output $\mathsf{ct}' \leftarrow \left\lceil \Delta^{-\ell'} \cdot \mathsf{ct} \right\rfloor \pmod{Q'_{\ell-\ell'}}$.
  Typically rescaling operation is done after multiplication and by one level.

RNS CKKS variants perform all operations in RNS. In other words, the power-of-two modulus $Q'_\ell = 2^{\rho_0 + \ell\cdot\rho}$ is replaced with $\prod_{i=0}^{\ell} q'_i$, where q'_i's are chosen as described above to support efficient number theoretic transforms (NTT) for converting native-integer polynomials w.r.t. each CRT modulus from coefficient representation to the evaluation one, and vice versa. The primes q'_i for $i = 1, \ldots, \ell$ are chosen to be as close to 2^ρ as possible to minimize the error introduced by rescaling.

The two major changes in the RNS instantiation compared to the CKKS scheme deal with rescaling and key switching.

[7] We can adapt swk to perform key switching for level $\ell < L$.

Rescaling in RNS. To efficiently perform rescaling in RNS from Q'_ℓ to $Q'_{\ell-1}$, the scaling down by 2^ρ is replaced with scaling down by q'_ℓ. For $i \in [L]$, q'_i are chosen, such that $2^\rho/q'_i$ is in the range $(1-2^{-\epsilon}, 1+2^{-\epsilon})$, where ϵ is kept as small as possible. The new rescaling operation to scale down by one level is defined as

– $\mathsf{Rescale}(\mathsf{ct}, {q'}_\ell)$. For a ciphertext $\mathsf{ct} \in \mathcal{R}_\ell^2$, output $\mathsf{ct}' \leftarrow \left\lceil {q'}_\ell^{-1} \cdot \mathsf{ct} \right\rfloor \pmod{Q'_{\ell-1}}$.

The maximum approximation error introduced by rescaling from ℓ to $\ell-1$ is

$$\left| {q'}_\ell^{-1} \cdot \boldsymbol{m} - 2^{-\rho} \cdot \boldsymbol{m} \right| \leq 2^{-\epsilon} \cdot \left| 2^{-\rho} \cdot \boldsymbol{m} \right|.$$

To minimize the cumulative approximation error growth in deeper computations, one can also alternate q'_i w.r.t. 2^ρ. For instance, if $q'_1 < 2^\rho$, then $q'_2 > 2^\rho$ and $q'_3 < 2^\rho$, etc. [KPP22].

Key Switching in RNS. To take advantage of RNS, we have to modify certain operations, such as base ω decomposition, to make them RNS-friendly. We use the hybrid key switching method described in [HK20]. Instead of the base ω decomposition, RNS digit decomposition is used. First, we use the partial products $\{\tilde{Q}'_j\}_{0 \leq j < \mathsf{dnum}} = \{\prod_{i=j\alpha}^{(j+1)\alpha-1} q'_i\}_{0 \leq j < \mathsf{dnum}}$, where $\alpha = (L+1)/\mathsf{dnum}$ for a pre-fixed parameter dnum. For level ℓ and $\mathsf{dnum}' = \lceil (\ell+1)/\alpha \rceil$ we then have:

$$\mathcal{WD}'_\ell(\boldsymbol{a}) = \left(\left[\boldsymbol{a} \frac{\tilde{Q}'_0}{Q'_\ell} \right]_{\tilde{Q}'_0}, \ldots, \left[\boldsymbol{a} \frac{\tilde{Q}'_{\mathsf{dnum}'-1}}{Q'_\ell} \right]_{\tilde{Q}'_{\mathsf{dnum}'-1}} \right) \in \mathcal{R}^{\mathsf{dnum}'},$$

$$\mathcal{PW}'_\ell(\boldsymbol{a}) = \left(\left[\boldsymbol{a} \frac{Q'_\ell}{\tilde{Q}'_0} \right]_{Q'_\ell}, \ldots, \left[\boldsymbol{a} \frac{Q'_\ell}{\tilde{Q}'_{\mathsf{dnum}'-1}} \right]_{Q'_\ell} \right) \in \mathcal{R}_{Q'_\ell}^{\mathsf{dnum}'}.$$

For any $(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}_\ell^2$, $\mathcal{WD}'_\ell$ and $\mathcal{PW}'_\ell$ satisfy the following congruence relation:

$$\left\langle \mathcal{WD}'_\ell(\boldsymbol{a}), \mathcal{PW}'_\ell(\boldsymbol{b}) \right\rangle \equiv \boldsymbol{a} \cdot \boldsymbol{b} \pmod{Q'_\ell}.$$

This key switching procedure is similar to the one used in CKKS with the only difference in the decomposition method.

– $\mathsf{KeySwitchGen}_{\mathsf{sk}}(\boldsymbol{s}')$. For auxiliary modulus $P' = \prod_{i=0}^{k} p_i$, sample a random $\boldsymbol{a}'_k \leftarrow \mathcal{R}_{P'Q'_L}$ and error $\boldsymbol{e}'_k \leftarrow \chi_{\mathsf{err}}$. For a pre-fixed parameter dnum, output the switching key as

$$\mathsf{swk} = (\mathsf{swk}_0, \mathsf{swk}_1) = \left(\{\boldsymbol{b}'_k\}_{k=0}^{\mathsf{dnum}-1}, \{\boldsymbol{a}'_k\}_{k=0}^{\mathsf{dnum}-1} \right) \in \mathcal{R}_{P'Q'_L}^{2 \times \mathsf{dnum}},$$

where

$$\boldsymbol{b}'_k \leftarrow -\boldsymbol{a}'_k \cdot \boldsymbol{s} + \boldsymbol{e}'_k + P' \cdot \mathcal{PW}'(\boldsymbol{s}')_k \pmod{P'Q'_L}.$$

– $\mathsf{KeySwitch}_{\mathsf{swk}}(\mathsf{ct})$. For $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}^2_{Q'_\ell}$, $\mathsf{swk} = (\mathsf{swk}_0, \mathsf{swk}_1)$ [8] output

$$\left(\boldsymbol{c}_0 + \left\lceil \frac{\left\langle \mathcal{WD}'_\ell\left(\boldsymbol{c}_1\right), \mathsf{swk}_0\right\rangle}{P'} \right\rfloor, \left\lceil \frac{\left\langle \mathcal{WD}'_\ell\left(\boldsymbol{c}_1\right), \mathsf{swk}_1\right\rangle}{P'} \right\rfloor\right) \pmod{Q'_\ell}.$$

To keep the noise from key switching small, we can take $P' \approx \max_j(\tilde{Q'}_j)$.

C Derivations and Proofs of Results in Section 3

C.1 Floor and Step Functions in terms of Complex Exponential Function

For the function $\mathsf{Rstep}_x\left(p\right)$ approximating

$$\mathsf{step}_p(k) = \begin{cases} 0, & \text{if } 0 \leq k < p/2 \\ p/2 & \text{if } p/2 \leq k < p. \end{cases}$$

we can evaluate the closed formulae for α_i from (5):

$$\begin{aligned} \alpha_0 &= \frac{1}{p}\left(\sum_{l=p/2}^{p-1} \frac{p}{2}\right) = \frac{p}{4}, \\ \alpha_k &= \frac{p-k}{p}\sum_{l=p/2}^{p-1} e^{-2\pi ikl/p} = \frac{(p-k)}{p}e^{-\pi ik}\left(\frac{1-e^{-\pi ik}}{1-e^{-2\pi ik/p}}\right) = \\ &= \frac{p-k}{p}e^{-\pi ik}\left(\frac{1-(-1)^k}{2i\sin\left(\frac{\pi k}{p}\right)e^{-i\pi k/p}}\right). \end{aligned}$$

For even k, $\alpha_k = 0$, and for odd k,

$$\alpha_k = \frac{p-k}{p}\left(\frac{\cos\left(\frac{\pi k}{p}\right) + i\sin\left(\frac{\pi k}{p}\right)}{i\sin\left(\frac{\pi k}{p}\right)}\right) = \frac{p-k}{p}\left(1 - i\cot\left(\frac{\pi k}{p}\right)\right).$$

For $\mathsf{Rmod}_p\left(x\right)$, the closed expressions can be written as

$$\begin{aligned} \alpha_0 &= \frac{1}{p}\sum_{l=0}^{p-1}\frac{l}{p} = \frac{p-1}{2}, \\ \alpha_k &= \frac{2(p-k)}{p^2}\sum_{l=0}^{p-1} le^{-i\pi kl/p} = \frac{2(p-k)}{p^2}\cdot\frac{(-p)}{1-e^{-2i\pi k/p}} \\ &= \frac{p-k}{p}\left(-1 + i\cot\left(\frac{\pi k}{p}\right)\right). \end{aligned}$$

[8] We can adapt swk to perform key switching for level $\ell < L$.

C.2 Second- and Third-Order Trigonometric Hermite Interpolation

Proof of Theorem 2. As the starting point we use Theorem 2.1 of [Var69], which proves uniqueness and provides a solution in terms of cosine series in explicit form for the (0,1,M) trigonometric interpolation where M is even. For $M = 2$ and conditions (16), the expression for $R_2(x)$ can be written as[9]

$$\begin{aligned} R_2(x) &= \sum_{l=0}^{p-1} f(l) \cdot U_2\left(2\pi\left(x - \frac{l}{p}\right)\right), \text{ where} \\ U_2(x) &= U(x) + \frac{1-\cos(px)}{p^3} \sum_{k=1}^{\lfloor p/2 \rfloor} (2 - \gamma_{p,k}) k(p-k)\cos(kx), \end{aligned} \tag{A2}$$

$\gamma_{p,k} = 1$ if p is even and $k = p/2$, while $\gamma_{p,k} = 0$ otherwise. Here, $U(x)$ is borrowed from (4).

We now derive the complex exponential expression for (A2). We transform the second summand of $U_2(x)$:

$$\begin{aligned} U_2'(x) &= \frac{1-\cos(px)}{p^3} \sum_{k=1}^{\lfloor p/2 \rfloor} (2 - \gamma_{p,k}) k(p-k)\cos(kx) \\ &= \frac{1}{p^3}\left(\sum_{k=1}^{\lfloor p/2 \rfloor} (2 - \gamma_{p,k}) k(p-k)\left(\cos(kx) - \frac{1}{2}\left(\cos((p+k)x) + \cos((p-k)x)\right)\right)\right) \end{aligned}$$

Next we switch to the complex exponential formulation:

$$T_2(x) = \sum_{l=0}^{p-1} f(l) \cdot W\left(2\pi\left(x - \frac{l}{p}\right)\right),$$

$$W_2(x) = W(x) + \frac{1}{p^3}\left(\sum_{k=1}^{\lfloor p/2 \rfloor} (2 - \gamma_{p,k}) k(p-k)\left(e^{ikx} - \frac{1}{2}\left(e^{i(p+k)x} + e^{i(p-k)x}\right)\right)\right).$$

We transform the second summand (here, $T_2(x) = T(x) + T_2'(x)$ and $T(x)$ is the same as for the first-order interpolation):

$$\begin{aligned} T_2'(x) &= \frac{1}{p^3} \sum_{l=0}^{p-1} \sum_{k=1}^{\lfloor p/2 \rfloor} f(l)(2 - \gamma_{p,k}) k(p-k) \times \\ &\quad \left(e^{-2\pi k l i/p} E(x)^k - \frac{e^{-2\pi(p+k)li/p}}{2} E(x)^{p+k} - \frac{e^{-2\pi(p-k)li/p}}{2} E(x)^{p-k}\right) \\ &= \sum_{k=1}^{\lfloor p/2 \rfloor} \beta_k E(x)^k - \frac{\delta_k}{2} E(x)^{p+k} - \frac{\theta_k}{2} E(x)^{p-k}, \end{aligned}$$

[9] We noticed a typo in the expression for $U_2(x)$ in [Var69], which is corrected in our expression; the correction is to have a different term for even p at $k = p/2$.

where

$$\beta_k = \frac{(2-\gamma_{p,k})k(p-k)}{p^3}\sum_{l=0}^{p-1} f(l)\cdot e^{-2\pi kli/p},$$

$$\delta_k = \frac{(2-\gamma_{p,k})k(p-k)}{p^3}\sum_{l=0}^{p-1} f(l)\cdot e^{-2\pi (p+k)li/p},$$

$$\theta_k = \frac{(2-\gamma_{p,k})k(p-k)}{p^3}\sum_{l=0}^{p-1} f(l)\cdot e^{-2\pi (p-k)li/p}.$$

□

Proof of Theorem 3. We use Theorem 7 of [Var73], which formulates in explicit form the (0,1,2,M) trigonometric interpolation. For $M = 3$ and conditions (18), the expression for $R_3(x)$ can be written as

$$R_3(x) = \sum_{l=0}^{p-1} f(l)\cdot U_3\left(2\pi\left(x-\frac{l}{p}\right)\right), \text{ where}$$

$$U_3(x) = U(x) + \frac{2\left(1-\cos(px)\right)}{3p^4}\sum_{k=1}^{p-1} k(p-k)(2p-k)\cos(kx).$$

Here, $U(x)$ is borrowed from (4).

The complex exponential expression $T_3(x)$ from the theorem's statement can be derived similarly to the derivation for $T_2(x)$. □

D Homomorphic Digit Decomposition

Algorithm 6 Homomorphic digit decomposition for an RLWE ciphertext

```
1: procedure HomDigitDecomp(ct ∈ R_Q^2)
2:     k ← 0
3:     while Q > q do
4:         ct_k ← ct mod q          ▷ Extract the RLWE digit encrypting a digit in Z_p^w.
5:         ct_d ← FuncBT_{q',Q'_L,Δ}(ct_k, LUT(Rmod_p(x)))
6:         ct ← ct − ct_d
7:         ct ← ModSwitch(ct, Q/p)
8:         Q ← Q/p, P ← P/p
9:         k ← k + 1
       return ({ct_k}_{i∈[k]}, ct)
```

Note that lines 4 through 6 directly correspond to the homomorphic floor evaluation algorithm. The correctness of HomDigitDecomp directly follows from the correctness of HomSign and HomFloor. The complexity is $\lceil \frac{\log P}{\log p} \rceil - 1$ functional bootstrapping invocations. The digit decomposition algorithm can also be extended to varying-size digits for different p_i, q_i via an approach analogous to the DM/CGGI case [LMP23].

E Experimental Results

Table A3 provides more detailed results on the runtimes presented in Table 1.

$\log p$	$\log Q$	N	$\log(Q'_L P')$	# limbs (enc, dec)	# limbs HKS	Online time (s)	Amtz. on. time (ms)	Offline time (s)
1	33	2^{15}	768	16 (3,3)	4	6.151	0.187	17.734
1	35	2^{16}	800	16 (3,3)	4	14.04	0.214	24.567
1	35	2^{16}	870	18 (4,4)	4	13.262	0.202	30.026
1	35	2^{16}	1000	20 (5,5)	4	13.712	0.209	18.278
2	33	2^{15}	747	19 (3,3)	2	12.432	0.379	49.009
2	35	2^{16}	965	19 (3,3)	4	21.804	0.331	28.961
2	35	2^{16}	1035	21 (4,4)	5	20.541	0.313	35.299
2	35	2^{16}	1105	23 (5,5)	6	20.422	0.312	20.489
3	35	2^{15}	750	18 (2,2)	2	17.332	0.529	39.037
3	36	2^{16}	1020	20 (3,3)	5	23.756	0.362	29.653
3	36	2^{16}	1092	22 (4,4)	5	22.661	0.345	35.653
3	36	2^{16}	1164	24 (5,5)	5	23.055	0.351	20.846
4	37	2^{15}	763	19 (2,2)	1	65.733	0.784	91.427
4	40	2^{16}	1140	21 (3,3)	5	25.934	0.396	30.412
4	40	2^{16}	1280	23 (4,4)	6	25.874	0.395	37.758
4	40	2^{16}	1420	25 (5,5)	7	26.644	0.406	22.51
8	47	2^{16}	1535	25 (3,3)	6	42.026	0.748	91.051
8	47	2^{16}	1509	27 (4,4)	4	76.918	0.834	131.61
8	47	2^{16}	1543	29 (5,5)	3	60.894	0.929	55.936
9	49	2^{16}	1514	26 (3,3)	4	69.33	1.058	59.43
9	49	2^{16}	1552	28 (4,4)	3	77.144	1.177	103.66
12	58	2^{17}	2458	31 (4,4)	11	569.54	4.345	97.08
12	58	2^{17}	2274	33 (5,5)	6	613.52	4.682	148.78
12	58	2^{17}	2574	33 (5,5)	11	610.96	4.661	98.39
12	58	2^{17}	2934	33 (5,5)	17	621.96	4.745	86.96

Table A3. Experiments for the evaluation of an LUT (floor) for an RLWE ciphertext with plaintext modulus p and ciphertext modulus Q. Here, $\log(Q) = \log(\Delta) = \log(q'_0)$ and $p = P$. By $\log(Q'_L P')$ we refer to the number of bits in the largest CKKS modulus, which includes all RNS limbs for the leveled computation (depth + 1) and all RNS limbs used in Hybrid Key Switching. A single RNS limb is left after the functional bootstrapping. Online time refers to the time for the evaluation of the functional bootstrapping. The offline time refers to the the setup time (evaluation keys generation), encryption of the messages, and precomputations. Best amortized times are marked in blue.