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) and Chilotti–Gama–Georgieva–Izabachène (CGGI) 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. 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 0.72 milliseconds, which is three orders of magnitude faster than the DM/CGGI approach and 6.8x 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* [KS23, LMP22] 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, KS23, LMP22], with the method from [LMP22] 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, LMP22].

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 an 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 on single-threaded CPU [BCKS24, AP23]. Although the CKKS scheme deviates from prior exact schemes 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, as the polynomial approximations of these functions are complicated by the requirements of knowing the approximation range and achieving desired precision. For this reason, CKKS may use other schemes, e.g., DM/CGGI, via scheme switching to evaluate discontinuous functions [LHH+21, LMP22], 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 of a vector into one ciphertext (CKKS typically outperforms BGV/BFV bootstrap-

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

ping by more than one order of magnitude [AP23]). 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].

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 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 zero first derivative 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 a small increase in computational complexity.

Our second contribution is a multi-precision sign evaluation procedure for an encrypted message in $\mathbb{Z}_P$, where $P > p$, which uses the step and modular reduction functions in $\mathbb{Z}_p$ as subroutines. We also propose a homomorphic digit extraction algorithm based on the modular reduction function, and show how a multi-precision arbitrary function evaluation capability can be built using the LUT evaluation in $\mathbb{Z}_p$ and the digit extraction procedure.

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

Our fourth contribution is 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 0.72 milliseconds, which is three orders of magnitude faster than for the DM/CGGI method and 6.8x faster than a more limited functional bootstrapping functionality based on the BFV scheme.

1.1 Technical Overview

Hybrid FHE scheme with functional bootstrapping. We develop a hybrid FHE scheme that supports general functional bootstrapping following 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.

Trigonometric Hermite interpolation. 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 requirements 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 illustrate the first-order interpolation in Figure 1 for the case of $\mu \bmod 4$, where $\mu \in \mathbb{Z}_4$. One can observe that in addition to matching the function at the interpolation points, the first-order interpolation has local extrema at these points, which correspond to noise reduction in the proximity of the interpolation points.

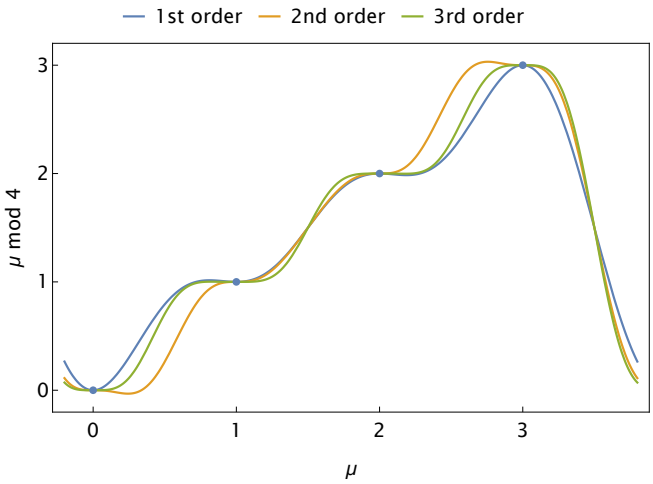

Fig. 1. Plot of $f_4(\mu) \equiv \mu \bmod 4$, where $\mu \in \mathbb{Z}_4$, and its first-order, second-order, and third-order trigonometric Hermite interpolations. The plateau near interpolation points progressively increases as the interpolation order grows.

We use results from trigonometric interpolation theory to derive an FHE-friendly analytical expression for evaluating the 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 efficiently 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. The full evaluation of the power series includes three steps: 1) the evaluation of the complex exponential function on a subinterval using a Chebyshev series interpolation, 2) the extension of the interpolation to the full interval via a recursive application of the double-angle formula, and 3) the evaluation of the power series of degree $p-1$ for the complex exponential function. Note that the first two steps are very similar to the approximate evaluation of the modular reduction function in CKKS bootstrapping.

After evaluating the power series, we perform homomorphic decoding to go back to the 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 in-

stantiations). 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 (for instance, where a long sequence of LUTs and other computations in between are evaluated), we derive expressions for second- and third-order trigonometric Hermite interpolations where the first two and three derivatives, respectively, are set to zero. The plots of second- and third-order interpolations for $\mu \bmod 4$ are illustrated in Figure 1. One can observe that the plateaus near the interpolation points progressively extend as the order is raised, achieving cubic and quartic noise reduction for second and third orders, respectively. We also derive FHE-friendly analytical polynomial expressions for these higher orders in terms of the complex exponential function. The second and third-order interpolations increase the degree of the power series from $p-1$ for the first order to $3p/2$ for second order and $2p-1$ for third order. The computational complexity increase of raising the interpolation order from first to third order is equivalent to increasing the precision, i.e., plaintext modulus, of the first-order interpolation by one bit (along with a potential increase in the scaling factor).

Noise analysis. We perform noise analysis for all three interpolation orders. The noise after functional bootstrapping is determined by the interplay of noise reduction due to a trigonometric Hermite interpolation and due to the CKKS approximation error accumulated as part of the functional bootstrapping (and other application-specific intermediate computations). The CKKS approximation error can be reduced by increasing the CKKS scaling factor, which plays a central role in fine-tuning the efficiency of functional bootstrapping. Our noise analysis shows that in scenarios where functional bootstrapping is the only computation, the first order will always be more efficient than higher orders. However, if a series of LUTs is evaluated and/or there are application-specific intermediate computations, the second order can achieve better efficiency by reducing the CKKS scaling factor. Our analysis also suggests that the noise reduction benefits of the third order are expected to be smaller than the computational complexity increase from the second to third order.

It should be highlighted that the noise reduction capability provided by trigonometric Hermite interpolations makes functional bootstrapping a "proper" bootstrapping procedure because in addition to enabling more computations (as in classical CKKS bootstrapping), this procedure also reduces the noise. Overall, our hybrid FHE scheme has the same properties (rounding during decryption, noise reduction during bootstrapping) as other exact FHE schemes, such as BGV, BFV, and DM/CGGI.

We also show that polynomial Hermite interpolations (not periodic in contrast to trigonometric interpolations) can be used to evaluate LUTs in a leveled setting and/or to reduce noise. These polynomial interpolations generalize prior results from [CKK20, DMPS24, CKKL24]. Our analysis implies that trigonometric Hermite interpolations are more efficient for LUT evaluation than polynomial Hermite interpolations in the settings where bootstrapping is needed.

Larger precision. Our method can currently efficiently support LUTs only for a limited range of p, e.g., we were able to run LUTs up to 14 bits using 64-bit words. If a larger domain 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 [LMP22], we develop multi-precision digit extraction and sign evaluation procedures. Both are based on the 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.

In the process, we 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 a cost slightly higher than a single LUT evaluation. In the case of multi-value LUT evaluation, the computation of the complex exponential function used for building the power series is performed only once for many LUTs. Our method can also be used for evaluating an encrypted LUT, which is used as a subroutine in a tree-based evaluation of large LUTs [GBA21]. In this case, the scalar coefficients in the power series are replaced with the ciphertexts resulting from the LUT evaluation. We also discuss how our method can be adapted for evaluating discontinuous functions in native CKKS, i.e., in a setting where messages are encoded using the CKKS inverse canonical embedding.

1.2 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 [LMP22] and [TCBS23]. 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- and 12-bit LUTs with another multi-precision method based on CGGI circuit bootstrapping, and our method has the amortized time two orders of magnitude smaller. The detailed results of our comparison are presented in Appendix A.2.

Bae *et al.* proposed 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 modular reduction 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 modular

reduction/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, demonstrating our method's speed-up by two orders of magnitude.

Chung *et al.* developed an 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. In summary, the evaluation of an LUT as part of bootstrapping in our method has a lower complexity than their method (the polynomial degree is reduced from $2p - 1$ to $p - 1$).

Liu and Wang devised an amortized (somewhat limited) functional bootstrapping method for DM/CGGI ciphertexts using BFV as the bootstrapping scheme [LW23, LW24]. Our method has a higher throughput (from 3.4x for 3-bit LUT to 8.4x for 12-bit LUT) and easily supports multi-precision extensions. Concrete numerical comparisons are given in Appendix A.2.

Lee et al. proposed a 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 methods in [LW23, LW24].

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 [LMP22] 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 fact that the main focus of our work is arbitrary function evaluation, we leave such comparison for future work.

1.3 Concurrent Works

Bae *et al.* [BKSS24] concurrently proposed a method for bootstrapping small integers using CKKS. Their first method appears to be similar in complexity to our first-order trigonometric Hermite interpolation. However, there are several major differences. First, the authors do not consider the idea of trigonometric Hermite interpolation, which is central to our work (only polynomial Hermite interpolations are mentioned, which have higher complexity as shown in Section 3.4 and Appendix A.3 of our paper). Instead, they derive their method starting from a Lagrangian interpolation for complex roots of unity and then apply complex conjugation to achieve quadratic noise reduction. No closed-form analytical expression is provided in [BKSS24]. In contrast, our work devises analytical expressions for first-order and higher-order trigonometric Hermite interpolations. Our higher-order interpolations enable FHE computations invoking a series of LUTs and/or other intermediate computations. The analytical expressions simplify the complexity analysis, noise estimation, and implementation of the method. The experimentally observed throughput of our method is significantly higher (from 2.2x to 4.8x in the range of $\log p$ from 2 to 8). We also report runtime results for up to 14 bits (vs. maximum 10 bits in [BKSS24]). Many other extensions are also proposed in our work, e.g., we devise multi-precision procedures, allowing us to support the sign evaluation for 32-bit encrypted numbers.

Another concurrent work [KN24] extends the functional bootstrapping from [BKSS24] to modular reduction for the multi-precision setting and real numbers. The multi-precision extension appears to be similar to our multi-precision sign evaluation method presented in Section 5. The amortized runtime for 10-bit decomposition (Table 10 in [KN24]) is about 2 milliseconds, which is roughly the same as in the first row of Table 2 in our paper. Overall, our work encompasses both [BKSS24] and [KN24], while also providing several distinct capabilities/results, such as higher-order trigonometric Hermite interpolations, convenient analytical expressions, and tighter and more comprehensive noise analysis.

1.4 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, along with noise and complexity analyses. In Section 5, we describe 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. The appendices present a detailed comparison of our results with the state-of-the-art methods, additional preliminaries, detailed derivations, and auxiliary tables and graphs.

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. We map the group of residue classes modulo p, $\mathbb{Z}_p$, to the representative interval $[0, p-1] \cap \mathbb{Z}$. Additional technical 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 Encryption 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 $\mathcal{S} = \mathbb{R}[X]/\langle X^N+1\rangle$ 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 $n \le N/2$, 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 $L \ge 0$ 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 $\boldsymbol{m} \leftarrow \left\lceil \tau'^{-1}_n(\Delta\cdot\boldsymbol{x}) \right\rfloor \in \mathcal{R}$.
- $\mathsf{Decode}(\boldsymbol{m}, \Delta)$. For plaintext $\boldsymbol{m} \in \mathcal{R}$, output the vector $\boldsymbol{x} \leftarrow \tau'_n(\boldsymbol{m}/\Delta) \in \mathbb{C}^n$.

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.

2.3 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 more 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 to be used with CKKS.

3.1 Trigonometric Hermite Interpolation

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

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

for fractional $x = \frac{j}{p}$, with $j \in [p]$, and 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, a quadratic reduction of noise was also chosen.

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

$$R\left(\frac{k}{p}\right) = f(k), \quad R'\left(\frac{k}{p}\right) = 0, \quad \text{where } k \in [p]. \tag{2}$$

The problem (1) with conditions (2) represents a linear system 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 analytical 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].

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

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

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

Our case corresponds to $M = 1$, and the expression for R is written as

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

Substituting $U(x)$ into $R(x)$, 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

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

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

3.2 FHE-Friendly Expressions 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 (3.1) 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*

$$\begin{aligned} T(x) &= \alpha_0 + \sum_{k=1}^{p-1} \alpha_k \cdot e^{2\pi i k x}, \\ \alpha_0 = \frac{1}{p}\sum_{l=0}^{p-1} f(l), \quad \alpha_k &= \frac{2(p-k)}{p^2}\sum_{l=0}^{p-1} f(l)\cdot e^{-2\pi k l i/p}. \end{aligned} \tag{4}$$

Proof. The complex generalization of (3.1) can be written as

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

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

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

By writing $T(x) = \alpha_0 + \sum_{k=1}^{p-1}\alpha_k \cdot e^{2\pi i k x}$ 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 $T(x)$, $R(x) = Re(T(x))$, which is equivalent to (1), (3). □

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

To support multi-precision evaluation, we also derived analytical expressions for modular reduction and step functions (see Appendix C.1 for further insights). For $S = \{2i+1 : i \in [\frac{p}{2}]\}$, it holds that

$$\mathsf{Rmod}_p(x) = \frac{p-1}{2} + \frac{1}{p}\sum_{k=1}^{p-1}(p-k)\left(-1 + i\cot\left(\frac{\pi k}{p}\right)\right)E(x)^k, \tag{5}$$

$$\mathsf{Rstep}_p(x) = \frac{p}{4} + \frac{1}{p}\sum_{k\in S}(p-k)\left(1 - i\cot\left(\frac{\pi k}{p}\right)\right)E(x)^k. \tag{6}$$

3.3 Higher-Order Trigonometric Hermite Interpolations

So far, we have focused on the first-order trigonometric Hermite interpolation $R(x)$ with constraints (2), achieving a quadratic noise reduction. For additional noise reduction, a second-order or even third-order trigonometric Hermite interpolation can be used. We derive here expressions for the second- and third-order trigonometric Hermite interpolations, and include the proofs in Appendix C.2.

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

$$R_2\left(\frac{k}{p}\right) = f(k), \quad R_2'\left(\frac{k}{p}\right) = 0, \quad R_2''\left(\frac{k}{p}\right) = 0, \quad \text{where } k \in [p]. \tag{7}$$

Theorem 2. *The second-order trigonometric Hermite interpolation polynomial $R_2(x)$ satisfying the constraints (7) exists, is unique, and can be expressed as the real part of complex polynomial $T_2(x)$, for $E(x) := e^{2\pi i x}$:*

$$T_2(x) = \alpha_0 + \sum_{v=1}^{p-1} \alpha_v \cdot E(x)^v + \sum_{k=1}^{\lfloor p/2 \rfloor} \beta_k E(x)^k - \frac{\delta_k}{2} E(x)^{p+k} - \frac{\theta_k}{2} E(x)^{p-k}, \tag{8}$$

$$\beta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_k, \quad \delta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_{p+k}, \quad \theta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_{p-k},$$

where the coefficients are as follows: α_v are the same as in the first-order expression (4) in Theorem 1; $c_k := (2-\gamma_{p,k})k(p-k)/p^3$; $e_k := e^{-2\pi kli/p}$; $\gamma_{p,k} = 1$ 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

$$R_3\left(\frac{k}{p}\right) = f(k), R_3'\left(\frac{k}{p}\right) = 0, R_3''\left(\frac{k}{p}\right) = 0, R_3'''\left(\frac{k}{p}\right) = 0, \text{ where } k \in [p]. \tag{9}$$

Theorem 3. *The third-order trigonometric Hermite interpolation polynomial $R_3(x)$ satisfying the constraints (9) exists, is unique, and can be expressed as the real part of complex polynomial $T_3(x)$, for $E(x) := e^{2\pi i x}$:*

$$T_3(x) = \alpha_0 + \sum_{k=1}^{p-1} (\alpha_k + \beta_k) \cdot E(x)^k - \frac{\delta_k}{2} E(x)^{p+k} - \frac{\theta_k}{2} E(x)^{p-k}, \tag{10}$$

$$\beta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_k, \quad \delta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_{p+k}, \quad \theta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_{p-k},$$

where the coefficients are as follows: α_k are the same as in the first-order expression (4) in Theorem 1; $c_k := 2k(p-k)(2p-k)/(3p^4)$; $e_k := e^{-2\pi kli/p}$.

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

3.4 Polynomial Hermite 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:

$$\bar{R}\left(x_k\right) = f\left(x_k\right), \quad \bar{R}'\left(x_k\right) = 0, \quad k = 0, 1, \ldots, p-1, \tag{11}$$

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:

$$\bar{R}(x) = \sum_{k=0}^{p-1} \left[\left(1 - 2(x - x_k)\ell_k'\left(x_k\right)\right)\ell_k(x)^2\right] f(x_k), \tag{12}$$

where $\ell_k(x) = \prod_{j=0, j\neq k}^{p-1}(x - x_j)/(x_k - x_j)$ is the Lagrange basis polynomial.

Similar to Section 3.3, 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:

$$\bar{R}\left(x_k\right) = f\left(x_k\right), \quad \bar{R}'\left(x_k\right) = 0, \quad \bar{R}''\left(x_k\right) = 0, \tag{13}$$

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. 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

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 $\boldsymbol{m}$ (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 a 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, 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 $[m + q_0' I]_{q_0'} = [m]_{q_0'} \approx \frac{q_0'}{2\pi} \sin(\frac{2\pi m}{q_0'})$.

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 ($\ll 1$). 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 already encoded in coefficients. Therefore, the first step in the functional bootstrapping is to apply $\mathsf{ModRaise}$, so both the message and overflows are 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 2.3 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. 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.

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; ▷ $q'_{i>0}$ 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** $\textsc{FuncBT}_{q'_0,Q'_L,\Delta}(\mathsf{ct} \in \mathcal{R}_q^2, \mathsf{LUT})$
2: $\mathsf{ct}_1 \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, q'_0)$ ▷ Switch ct from q to q'_0. The scaling of the message becomes $\frac{q'_0}{p}$.
3: $\mathsf{ct}_2 \leftarrow \frac{\Delta}{q'_0}\mathsf{ct}_1$ ▷ Adjust the scaling factor such that we obtain a CKKS ciphertext encoding $\Delta\frac{m(X)}{p} \bmod q'_0$.
4: $\mathsf{ct}_3 \leftarrow \mathsf{ModRaise}(\mathsf{ct}_2, Q'_L)$ ▷ Encoded vector becomes $\Delta\frac{m(X)}{p} + q'_0 I(X) \bmod Q'_L$
5: $\mathsf{ct}_4 \leftarrow \mathsf{CtS}(\mathsf{ct}_3)$ ▷ Homomorphic encoding operation, the encoded vector becomes $\Delta\frac{\tau(\boldsymbol{m})}{p} + q'_0\tau(\boldsymbol{I}) \bmod Q''$, for Q'' being the ciphertext modulus after the levels consumed by CtS.
6: $\mathsf{ct}_5 \leftarrow \mathsf{EvalLUT}(\mathsf{ct}_4, \mathsf{LUT})$. ▷ 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: $\mathsf{ct}_6 \leftarrow \mathsf{StC}(\mathsf{ct}_5)$ ▷ Homomorphic decoding operation with an adjusting factor of $Q'/(\Delta P)$, the encoded vector becomes $Q'\frac{m'(X)}{P} \bmod Q'$
8: $\mathsf{ct}' \leftarrow \mathsf{ModSwitch}(\mathsf{ct}_6, Q)$ ▷ Switch ct_6 from Q' to Q. The RLWE ciphertext encodes $\frac{Q}{P}m'(X)$.
9: **return** ct'

Adjusting the scaling factor in line 3 of Algorithm 1 may require another level. The homomorphic encoding CtS and decoding 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 use the polynomial (4) for the first-order trigonometric Hermite interpolation for $\mathsf{EvalLUT}$. For the second-order and third-order interpolations, we use the polynomials (8) and (10), respectively. In $\mathsf{EvalLUT}$, we first evaluate $E'(x) = e^{2\pi xi/2^r}$ 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 last step of $\mathsf{EvalLUT}$, which consists of taking the real part of the expression (4), is done via a complex conjugation.

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. For higher orders, $\cos(6\pi x)$ is required and the computation using $E(x)$ is more convenient.

Remark 2. We can achieve *multi-value bootstrapping* [CIM19], i.e., evaluating multiple LUTs over the same inputs, at a lower cost than independently evaluating each LUT. The costliest part, computing all $e^{2\pi jxi}$ for $j \in [p]$ in expression (4), can be done once for many LUTs operating on the same ciphertext. While the cost of remaining operations, scalar computations in the polynomial evaluation algorithms and evaluation of StC, is not negligible (on the order of 10% as compared to full bootstrapping), a significant reduction in the amortized runtime complexity can be achieved via this optimization.

4.1 Correctness and Noise Analysis

The correctness of the procedure depicted in Algorithm 1 follows from the correctness of the regular CKKS bootstrapping (see Section 2.3) and the correctness of trigonometric interpolations because we use EvalLUT instead of EvalMod in our functional bootstrapping algorithm. We focus here only on 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*

$$R\left(\frac{k}{p}\right) = f(k), \quad R'\left(\frac{k}{p}\right) = 0, \quad \ldots, \quad R^{(M)}\left(\frac{k}{p}\right) = 0, \tag{14}$$

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

$$\|\epsilon_{out}\|_\infty = \max_{k=0,\ldots,p-1} \left| R\left(\frac{k}{p} + \epsilon_{in}\right) - f(k) \right| \leq C_M \cdot \|\epsilon_{in}\|_\infty^{M+1}, \tag{15}$$

where C_M is a positive constant and $|\epsilon_{in}| < \frac{1}{2p}$.

Proof. If a unique interpolation $R(x)$ satisfying the constraints (14) exists, then we can apply the Taylor's theorem with the mean-value form of the remainder (R is infinitely differentiable). Concretely, we have for any k and any small ϵ_{in}:

$$\left| R\left(\frac{k}{p} + \epsilon_{in}\right) - f(k) \right| = \left| \sum_{i=M+1}^{\infty} \frac{R^{(i)}(\frac{k}{p})}{i!} (\epsilon_{in})^i \right| \leq \max_{\substack{\frac{k}{p} - |\epsilon_{in}| \leq x \\ x \leq \frac{k}{p} + |\epsilon_{in}|}} \frac{\left|R^{(M+1)}(x)\right|}{(M+1)!} |\epsilon_{in}|^{M+1}.$$

Then, the constant from the theorem statement can be obtained as

$$C_M := \max_{\substack{0 \leq k \leq p-1 \\ \frac{k}{p} - |\epsilon_{in}| \leq x \leq \frac{k}{p} + |\epsilon_{in}|}} \frac{\left|R^{(M+1)}(x)\right|}{(M+1)!}. \qquad \square$$

Corollary 2. *For some positive constants $B_1, B_2, B_3 = o\left(\frac{9}{4}p\right)$, it holds that:*

$$C_1 \leq \frac{\pi^2 B_1}{3}(p-1)p(2p-1), \quad C_2 \leq \frac{\pi^3 B_2}{48}p^2(3p+2)^2$$
$$C_3 \leq \frac{2\pi^4 B_3}{45}p(2p-1)(4p-1)(12p^2-6p-1). \tag{16}$$

Above, we assumed that $f : \mathbb{Z}_p \to \mathbb{Z}_p$. If f has a smaller codomain, the bounds can be tighter. The proof is given in Appendix C.3.

To further understand the noise growth, we focused on functions with large jumps at discontinuity points, such as modular reduction and S-box, a highly discontinuous function used in AES transciphering. We observed experimentally that the empirical bound for the constants C_M is below $(2\pi)^{M+1}p^{M+1}$, i.e., $O(B_M p^M)$. This implies that our theoretical bound $O(B_M p^{M+1})$ may be too loose even for highly discontinuous functions.

In view of the above observation, we attempted to derive a tighter theoretical bound for arbitrary functions using known relations from approximation theory. Our first-order interpolation polynomial $R(x)$ for a given p is known as the Jackson polynomial of order p [Zyg03] (Vol II, p. 21). A nice property of the Jackson polynomial is that its values are bounded in interval $[0, p-1]$ because the polynomial is expressed as a convolution (either series or integral) over the Fejer kernel $U(x)$, which can be written in closed form as $\frac{1}{p^2}\left(\frac{\sin\frac{nx}{2}}{\sin\frac{x}{2}}\right)^2$ [Var69] (Eq. 2.8). As the Fejer kernel is non-negative, a tight bound for the Jackson polynomial can be derived. However, as soon as we take the second derivative of the Fejer kernel (necessary for deriving the constant), we get a trigonometric series with both positive and negative terms. If we replace each term of the series with its absolute value (so we could then apply e.g., Abel's inequality), the bound is increased by $O(p)$, which ultimately leads to the same loose theoretical bounds as in (16). As a result, in our work we use an empirical approach, and leave the derivation of a tighter theoretical bound as an interesting research problem for researchers specializing in approximation theory.

Next, we describe our empirical approach for the case of modular reduction, both to show how to examine errors in practice and to build an intuition for choosing the interpolation order for a specific application. Figure 2 illustrates the noise reduction for different interpolation orders for the modular reduction function $\mathrm{mod}p$ at $p = 256$. Here, we initially ignore the effect of CKKS approximation error (we discuss it later). To find $\|\epsilon_{out}\|_\infty$, we analytically compute the first non-zero Taylor series coefficients at each point for each interpolation order, and then use the infinity norm over all points for a given interpolation as the value of $\|\epsilon_{out}\|_\infty$ (note that the effect of higher-order Taylor series terms is negligible for the selected range of $\|\epsilon_{in}\|_\infty$). The line $\|\epsilon_{out}\|_\infty / \|\epsilon_{in}\|_\infty = 1$ is of special interest, as it indicates the points at which we start observing noise reduction. Figure 2 implies that the third-order interpolation starts reducing the error when the gap between the error and message reaches roughly 6.5 bits (note that the first 8 bits are used for the message) while the first-order requires the

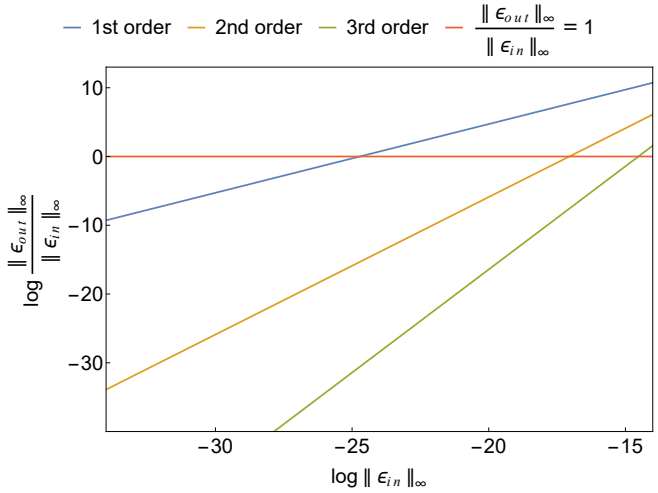

Fig. 2. Noise reduction for different interpolation orders when evaluating the modular reduction function at $p = 256$. ϵ_{in} is the input deviation from the interpolation points and ϵ_{out} is the output error after functional bootstrapping. The maximum error that can provide correctness is $\frac{1}{2p} = 2^{-9}$. The line $\|\epsilon_{out}\|_\infty / \|\epsilon_{in}\|_\infty = 1$ corresponds to zero noise reduction (anything above this line corresponds to noise increase). The zero noise reduction points in terms of $\log \|\epsilon_{in}\|_\infty$ are -24.7, -17.1, and -14.5 for the first, second, and third orders, respectively.

gap to be at least 16.7 bits. We performed the same analysis for the S-box function. The results summarized in Appendix E are very similar to the case $\bmod p$, with points of zero noise reduction being within 1 bit of the values for $\bmod p$.

In practice, we need to also take into account the CKKS approximation error, which itself is composed of multiple different approximation errors (see the introduction of [KPP22] for an overview of these errors). These errors also depend on how the rescaling operation is performed for various RNS variants. The main parameter that can compensate for the approximation error is the CKKS scaling factor 2^ρ. The larger 2^ρ is, the larger is the gap between the message and error that can be supported by CKKS computations in functional bootstrapping. The analysis illustrated in Figure 2 assumes that the CKKS scaling factor there is large enough for the CKKS approximation error to be ignored. When the CKKS approximation error is significantly large, its effect on noise needs to be considered. The analysis of Table 7 in [KPP22] implies that the approximation error in nontrivial polynomial CKKS computations is at least 20 bits (it is even larger for bootstrapping because of higher polynomial degrees). This suggests that the minimum CKKS scaling factor that can achieve correctness even for $p = 2$ should be of the order of 30 bits (which is supported by our experimental results presented in Table 1; note that minimum approximation error in Table 7 of [KPP22] for the CKKS RNS variant we used in our implementation is 25 bits). Another important consideration is whether a single LUT needs to be computed or we

deal with a series of LUTs (with potentially other computations in between). In the case of a series of LUTs, $\|\epsilon_{in}\|_\infty$ before the second functional bootstrapping can be much larger than before the first functional bootstrapping as it includes the CKKS approximation error from the first functional bootstrapping.

Now we come to the main practical question. What interpolation order should one use for a given application? Theoretically speaking, the first-order interpolation can always work as it achieves noise reduction as long as there is a sufficient gap between the message and error, i.e., the CKKS scaling factor is large enough. However, when the CKKS scaling factor needs to be restricted, e.g., in order to use a lower ring dimension or to fit into a 64-bit machine word size, a higher interpolation order should be used in scenarios with a series of LUTs or when some computations are performed before a single LUT.[5] There is no benefit in higher orders if a single LUT over a fresh ciphertext (where $\|\epsilon_{in}\|_\infty$ is very small) is performed before decrypting, as only the noise of functional bootstrapping operations determines the minimum CKKS scaling factor (and the noise is higher for higher orders). Figure 2 implies that the second-order interpolation should be (practically) sufficient in cases requiring higher-order interpolations as the difference in bits between the second and third orders is much smaller than between the first and second orders. A similar observation is expected for higher orders (which is why we did not consider a fourth-order interpolation in our work).

Note that in our proposed hybrid scheme, the decryption happens in BFV, i.e., the noise is removed during decryption via rounding. In scenarios where the IND-CPA^D security model needs to be satisfied, the decryption failure probability must be made negligible [LM21, ABMP24]. To do so, the parameters have to be chosen using the accumulated noise from BFV encryption, modulus switching when switching to CKKS, CKKS approximation errors, the modulus switching when going back to BFV, and rounding during decryption.

4.2 Complexity Analysis

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 $\mathsf{levels\text{-}for\text{-}evaluating}(E(x)) + \log(p) + 1$.[6] 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

[5] This discussion is for general LUTs; for LUTs with small output space, first-order interpolation followed by a cleaning polynomial [DMPS24] can be more efficient.

[6] Higher precision requires a better approximation. In practice, for $p > 2^{10}$, we have to increase the degree (and depth) of the Chebyshev series interpolation for $E(x)$.

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 is typically smaller as other operations in EvalLUT and Algorithm 1 have a much smaller increase in complexity because they do not directly depend on p.

For second- and third-order interpolations, the degree increases to $3p/2$ 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 to the cost of adding an extra bit to the plaintext space.

5 Amortized Multi-Precision Function Evaluation

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, LMP22] 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

For P and p the multi-precision and single-precision plaintext moduli, and Q and q are the corresponding ciphertext moduli, the floor function for digit decomposition we want to obtain is $f(m) = m - (m \bmod p)$, where $m \in \mathbb{Z}_P$. The floor function evaluation is based on Algorithm 1 for evaluating $\bmod p$, where instead of the general power series (4), one can use a simpler analytic expression (5).

The algorithm for the floor function is outlined in Algorithm 2 (same public parameters as in Algorithm 1). The correctness of evaluating HomFloor follows from the correctness of FuncBT. The complexity of evaluating HomFloor is the same as for FuncBT because the power series (5) for $\mathsf{Rmod}_p(x)$ has the same polynomial degree as the general expression (4) for $R(x)$, and the cost of homomorphic subtraction is negligible.

Algorithm 2 Homomorphic floor evaluation for an RLWE ciphertext

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

5.2 Homomorphic Evaluation of Multi-Precision Sign Function

We use the blueprint from [LMP22] (included in Appendix B.2 Algorithm 5, for DM/CGGI functional bootstrapping). The outline of the multi-precision sign evaluation algorithm for an input RLWE ciphertext is presented in Algorithm 3 (same public parameters as in Algorithm 1). The algorithm uses our functional bootstrapping method for the mod$p$ function (inside $\mathsf{HomFloor}$) 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 $\frac{2}{p}\mathsf{Rstep}_p(x)$, where $\mathsf{Rstep}_p(x)$ is given by (6).

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 $\lceil \log P / \log p \rceil$ functional bootstrapping operations. The complexity of evaluating the last functional bootstrapping invocation for step is slightly smaller as the even terms in the series are zero.

Algorithm 3 Multi-precision sign evaluation for an RLWE ciphertext

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

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 devised 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 the 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, although at a higher cost. 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. Following [GBA21], the small-size LUTs on the first level of tree ($d'p^{d-1}$ LUTs) have plaintext coefficients and can be evaluated with our functional bootstrapping described in Algorithm 1. The small-size LUTs on the following levels $i+1$ have encrypted coefficients obtained as the result of the LUTs on level i.

Our method can be adapted to evaluating LUTs with encrypted coefficients as well, by replacing the Paterson-Stockmeyer polynomial evaluation (which requires long division of polynomials, i.e., division by encrypted coefficients) with a version of the "baby-step-giant-step" polynomial evaluation algorithm (which recursively evaluates smaller-degree polynomials with the same coefficients as in the initial polynomial). For plaintext coefficients, the operation complexity of the two polynomial evaluation algorithms is similar [HK20]; however, evaluating a degree-p polynomial with encrypted coefficients requires additional $p-1$ ciphertext-ciphertext multiplications. Moreover, the recursion depth of the polynomial should be chosen such that the results of LUT evaluation have a sufficient number of RNS limbs, to support the use of the results as encrypted input coefficients for further recursive evaluation.

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, see Remark 2.

6 Functional Bootstrapping for CKKS Ciphertexts

Here, 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

$\boldsymbol{m} \in \mathbb{Z}_p^w$ that is encrypted in a CKKS ciphertext. One can modify the expressions of $R(x)$ obtained from (4), (8) and (10) 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. 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 require evaluating the polynomials on two ciphertexts.

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 3.4.

Algorithm 4 Amortized functional bootstrapping for a CKKS ciphertext

Public parameters:
- q': CKKS ciphertext modulus, prime; ▷ $q' > q'_{i\geq 1}$ 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.

1: **procedure** FuncBT'$_{q',Q'_L,\Delta}(\mathsf{ct} \in \mathcal{R}^2_{q'}, \mathsf{LUT})$
2: $\quad \mathsf{ct}_1 \leftarrow \mathsf{StC}(\mathsf{ct})$ ▷ Homomorphic decoding operation and potential modulus reduction, the encoded vector becomes $\Delta m(X) \bmod q'_0$
3: $\quad \mathsf{ct}_2 \leftarrow \mathsf{ModRaise}(\mathsf{ct}_1, Q'_L)$ ▷ Encoded vector becomes $\Delta m(X) + q'_0 I(X) \bmod Q'_L$
4: $\quad \mathsf{ct}_3 \leftarrow \mathsf{CtS}(\mathsf{ct}_2)$ ▷ Homomorphic encoding operation, the encoded vector becomes $\Delta\tau(\boldsymbol{m}) + q'_0\tau(\boldsymbol{I}) \bmod Q''$, for Q'' being the ciphertext modulus after the levels consumed by CtS.
5: $\quad \mathsf{ct}' \leftarrow \mathsf{EvalLUT}(\mathsf{ct}_3, \mathsf{LUT})$. ▷ 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 $f(\boldsymbol{m})$.
6: $\quad$ **return** ct'

7 Implementation and Performance Evaluation

7.1 Parameter Selection and Implementation Details

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 [BMTH21]); for $N = 2^{16}$, we set the threshold to 1,553 (using Table 3 of [BMTH21]). For $N = 2^{17}$, we used a linear interpolation fitting all values from Table 4 of [CP19] and 1,553 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 [BMTH21]). 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 $e^{2\pi xi}$ 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).

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.

7.2 Experimental Results

Table 1 shows the latency and amortized time for 1-bit to 4-bit, 8-bit, 9-bit, 12-bit and 14-bit LUT evaluations, which are common LUT sizes in the related literature. Note there is an almost 2x (1.6x) 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 $e^{2\pi xi}$, 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 (1.09x) 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

Table 1. Experiments for the isolated evaluation of a single LUT (modular reduction) 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.

Interp. order	$\log p$	$\log Q$	N	$\log(Q'_L P')$	Time (s)	Amtz. time (ms)
1	1	33	2^{15}	768	6.159	0.188
2, 3	1	33	2^{15}	747	10.25	0.315[a]
1	2	35	2^{16}	1035	19.718	0.3
2, 3	2	35	2^{16}	1070	21.383	0.326
1	3	37	2^{16}	1114	21.464	0.327
2	3	37	2^{16}	1114	22.134	0.338
1	4	38	2^{16}	1234	24.575	0.375
2	4	38	2^{16}	1234	25.447	0.388
1	8	47	2^{16}	1535	47.322	0.722
2	8	47	2^{16}	1535	55.062	0.84
1	9	48	2^{16}	1548	63.958	0.976
2	9	48	2^{16}	1548	76.288	1.164
1	12	55	2^{17}	2420	599.9	4.577
2	12	55	2^{17}	2420	893.78	6.819
1	14	58	2^{17}	2692	2130.1	16.25
2	14	58	2^{17}	2692	3370.9	25.72

[a] Using the ring dimension 2^{16}, we can get the same runtime as for $p = 4$ first-order.

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. We remark 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. More detailed information (including runtimes for the third order) is provided in Table A3.

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 generally incurs exponential complexity. 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 expression). In our case, the cost of evaluating an 8-bit LUT for AES S-box is only 3.8x higher than evaluating a 1-bit LUT, implying a speed-up of about 250x over the tree-based bit-level approach.

For higher orders, we notice the same expected increase in runtime with the increase of the polynomial degree. For $p = 2$, going from the first-order to second-order (and, equivalently, to third-order interpolation, since $p + p/2 = 2p - 1$) yields the same increase as between $p = 2$ and $p = 4$ for first-order interpolation, requiring the evaluation of $e^{2\pi i x}$ instead of only $\cos 2\pi x$ (see Remark 1).

Table 2. Experiments for multi-precision sign evaluation 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 $N = 2^{16}$. $\log(q) = \log(\Delta) = \log(q'_0)$. The interpolation order used is 1.

$\log P$	$\log Q$	$\log p$	$\log q$	$\log_2(Q'_L P')$	Time (s)	Amtz. time (ms)
12	46	1	35	870	146.97	2.242
12	45	2	35	1035	119.59	1.825
12	46	3	37	1114	86.5	1.32
12	48	4	40	1280	73.869	1.127
12	48	6	42	1470	63.81	0.974
21	56	1	36[a]	888	268.04	4.09
21	55	3	37	1114	150.16	2.293
21	57	7	43	1538	112.25	1.712
32	71	8	47	1535	191.604	2.924

[a] As an example of where a higher-order interpolation can achieve correctness with a smaller scaling factor, here, $\log q$ can be 35 if we use order 2. However, for this particular case, the runtime becomes 1.48x higher, see Remark 1.

Finally, barring runtime increases due to a larger scaling factor, the complexity of evaluating the third-order interpolation for p is equivalent to the complexity of evaluating the first-order approximation for $2p$ (as illustrated in Table A3).

Table 2 presents the timing results for multi-precision sign evaluation for 12-bit, 21-bit, and 32-bit encrypted messages. 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 our method with $p > 2$ for digits always outperforms the Boolean approach of [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, which is a useful practical scenario. As expected from complexity analysis, 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.

To port our results from RLWE to DM/CGGI inputs, we need to add the ring packing procedure time (the amortized time is 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 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.

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) and the increase in the scaling factor. Further optimizations can be performed over our proof of concept implementation to reduce the overhead of the polynomial evaluation for a large degree p. In the CGGI setting, circuit bootstrapping is more efficient for evaluating large-precision LUTs than the tree-based method discussed in our work [BBB+23]. We expect that a circuit bootstrapping method can also be developed based on our scheme and leave it as an interesting problem for future research.

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. URL: `https://eprint.iacr.org/2022/915`, `doi:10.1145/3560827.3563379`.

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. URL: `https://eprint.iacr.org/2024/203`.

AP23. Ahmad Al Badawi and Yuriy Polyakov. Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149, 2023. URL: `https://eprint.iacr.org/2023/149`.

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. `doi:10.1007/S00145-023-09463-5`.

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 *CRYPTO (4)*, volume 14084 of *Lecture Notes in Computer Science*, pages 37–69. Springer, 2023. `doi:10.1007/978-3-031-38551-3_2`.

BCKS24. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In *EUROCRYPT (2)*, volume 14652 of *Lecture Notes in Computer Science*, pages 94–123. Springer, 2024. `doi:10.1007/978-3-031-58723-8\_4`.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014. `doi:10.1145/2633600`.

BKSS24. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In *ASIACRYPT (1)*, volume 15484

of *Lecture Notes in Computer Science*, pages 330–360. Springer, 2024. doi:10.1007/978-981-96-0875-1_11.

BMTH21. Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *EUROCRYPT (1)*, volume 12696 of *Lecture Notes in Computer Science*, pages 587–617. Springer, 2021. doi:10.1007/978-3-030-77870-5_21.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012. doi:10.1007/978-3-642-32009-5_50.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *EUROCRYPT (2)*, volume 11477 of *Lecture Notes in Computer Science*, pages 34–54. Springer, 2019. doi:10.1007/978-3-030-17656-3_2.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, 2016. doi:10.1007/978-3-662-53887-6_1.

CHK+18. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT (1)*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2018. doi:10.1007/978-3-319-78381-9_14.

CIM19. Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In *CT-RSA*, volume 11405 of *Lecture Notes in Computer Science*, pages 106–126. Springer, 2019. doi:10.1007/978-3-030-12612-4_6.

CJP21. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *CSCML*, volume 12716 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2021. doi:10.1007/978-3-030-78086-9_1.

CKK+19. Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun-Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In *ASIACRYPT (2)*, volume 11922 of *Lecture Notes in Computer Science*, pages 415–445. Springer, 2019. doi:10.1007/978-3-030-34621-8_15.

CKK20. Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *ASIACRYPT (2)*, volume 12492 of *Lecture Notes in Computer Science*, pages 221–256. Springer, 2020. doi:10.1007/978-3-030-64834-3_8.

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. URL: https://eprint.iacr.org/2024/274.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT (1)*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017. doi:10.1007/978-3-319-70694-8_15.

CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient

arithmetic circuits for TFHE. In *ASIACRYPT (3)*, volume 13092 of *Lecture Notes in Computer Science*, pages 670–699. Springer, 2021. doi:10.1007/978-3-030-92078-4_23.

CP19. Benjamin R. Curtis and Rachel Player. On the feasibility and impact of standardising sparse-secret LWE parameter sets for homomorphic encryption. In *WAHC@CCS*, pages 1–10. ACM, 2019. doi:10.1145/3338469.3358940.

DM15. Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015. doi:10.1007/978-3-662-46800-5_24.

DMPS24. Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. BLEACH: cleaning errors in discrete computations over CKKS. *J. Cryptol.*, 37(1):3, 2024. doi:10.1007/S00145-023-09483-1.

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

GBA21. Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in TFHE. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):229–253, 2021. doi:10.46586/tches.v2021.i2.229-253.

Gen09a. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. URL: https://crypto.stanford.edu/craig/craig-thesis.pdf.

Gen09b. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009. doi:10.1145/1536414.1536440.

GV23. Robin Geelen and Frederik Vercauteren. Bootstrapping for BGV and BFV revisited. *J. Cryptol.*, 36(2):12, 2023. doi:10.1007/S00145-023-09454-6.

HK20. Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *CT-RSA*, volume 12006 of *Lecture Notes in Computer Science*, pages 364–390. Springer, 2020. doi:10.1007/978-3-030-40186-3_16.

KN24. Jaehyung Kim and Taeyeong Noh. Modular reduction in CKKS. Cryptology ePrint Archive, Paper 2024/1638, 2024. URL: https://eprint.iacr.org/2024/1638.

KPP22. Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. In *CT-RSA*, volume 13161 of *Lecture Notes in Computer Science*, pages 120–144. Springer, 2022. doi:10.1007/978-3-030-95312-6_6.

KS23. Kamil Kluczniak and Leonard Schild. FDFB: full domain functional bootstrapping towards practical fully homomorphic encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):501–537, 2023. doi:10.46586/tches.v2023.i1.501-537.

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 *SP*, pages 1057–1073. IEEE, 2021. doi:10.1109/SP40001.2021.00043.

LM21. Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In *EUROCRYPT (1)*, volume 12696 of *Lecture Notes in Computer Science*, pages 648–677. Springer, 2021. doi:10.1007/978-3-030-77870-5_23.

LMP22. Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In *ASIACRYPT (2)*, volume 13792 of *Lecture Notes in Computer Science*, pages 130–160. Springer, 2022. doi:10.1007/978-3-031-22966-4_5.

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

LW23. Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{o}(1)$ polynomial multiplications. In *ASIACRYPT (6)*, volume 14443 of *Lecture Notes in Computer Science*, pages 101–132. Springer, 2023. doi:10.1007/978-981-99-8736-8_4.

LW24. Zeyu Liu and Yunhao Wang. Relaxed functional bootstrapping: A new perspective on BGV/BFV bootstrapping. In *ASIACRYPT (1)*, volume 15484 of *Lecture Notes in Computer Science*, pages 208–240. Springer, 2024. doi:10.1007/978-981-96-0875-1_7.

MP21. Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In *WAHC@CCS*, pages 17–28. WAHC@ACM, 2021. doi:10.1145/3474366.3486924.

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. *Proc. IEEE*, 110(10):1572–1609, 2022. doi:10.1109/JPROC.2022.3205665.

PS73. Mike Paterson and Larry J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973. doi:10.1137/0202007.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009. doi:10.1145/1568318.1568324.

SV65. Ambikeshwar Sharma and Arun K. Varma. Trigonometric interpolation. *Duke Mathematical J.*, 32(2):341 – 357, 1965. doi:10.1215/S0012-7094-65-03235-7.

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@CCS*, pages 79–90. ACM, 2023. doi:10.1145/3605759.3625260.

Var69. Arun K. Varma. Trigonometric interpolation. *J. Mathematical Analysis and Applications*, 28(3):652–659, 1969. doi:10.1016/0022-247X(69)90018-3.

Var73. Arun K. Varma. Hermite-Birkhoff trigonometric interpolation in the (0, 1, 2, m) case. *J. Australian Mathematical Society*, 15(2):228–242, 1973. doi:10.1017/S1446788700012994.

WHS+24. Ruida Wang, Jincheol Ha, Xuan Shen, Xianhui Lu, Chunling Chen, Kunpeng Wang, and Jooyoung Lee. Refined TFHE leveled homomorphic evaluation and its application. Cryptology ePrint Archive, Paper 2024/1318, 2024. URL: https://eprint.iacr.org/2024/1318.

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.

Zyg03. A. Zygmund. *Trigonometric Series*. Cambridge Mathematical Library. Cambridge University Press, 3 edition, 2003. doi:10.1017/CBO9781316036587.

A Comparison with Other Methods

This section compares our results (for the first-order interpolation) 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 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 0.188 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 [LMP22] for single- and multi-precision LUT evaluation (note that uniform ternary secret key distribution was used in [LMP22]). For the fairness of comparison, we reran the experiments from [LMP22] 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 multi-precision 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).

Finally, Table A1 also shows the comparison with the CPU benchmarks for single-precision programmable bootstrapping (PBS) with failure probability of 2^{-40} in TFHE-rs.[7] We observe the increase of the speed-up of our method compared to the single-precision PBS in TFHE-rs from two to three orders of magnitude as the LUT size increases from 4 to 8 bits. For the homomorphic sign evaluation, we include the multi-precision CPU benchmarks for the comparison between an encrypted and unencrypted input, with failure probability of 2^{-64}.[8] The speed-up of our method in this case ranges from 18x to 46x.

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 starts performing better for the CGGI cryptosystem than the tree-based approach when the precision reaches 10-11 bits [BBB+23]. To the best of our knowledge, the state-of-the-art results for general LUT evaluation using the circuit bootstrapping approach are presented in [WHS+24]. In Table 10, the authors report the runtimes of 112.74 and 170.62 ms for 8-bit and 12-bit

[7] The CPU benchmarks are taken from `https://docs.zama.ai/tfhe-rs/get-started/benchmarks/cpu/cpu_programmable_bootstrapping`. Note that the machine used for this benchmark is different than ours (AWS hpc7a.96xlarge instance with a 96-core AMD EPYC 9R14 CPU @ 2.60GHz and 740GB of RAM) and uses AVX-512 extensions.

[8] The CPU benchmarks are taken from `https://docs.zama.ai/tfhe-rs/get-started/benchmarks/cpu/cpu_integer_operations`, run on the same machine as mentioned above.

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

Function	$\log P$	[TCBS23] runtime (ms)	[LMP22] runtime (ms)	[Zam22] PBS runtime (ms)	Our amtz. runtime (ms)
EvalLUT	2	7	92	6.04	0.356
EvalLUT	3	15	243	–	0.383
EvalLUT	4	29	–	11.3	0.431
EvalLUT	8	1,500/300[a]	–	458	0.778
HomSign	8	–	671	36.4	0.778
HomSign	12	–	1,367	35.2[b]	1.030
HomSign	21	–	3,451	53.7[c]	1.768
HomSign	32	–	–	53.7	2.98

[a] Corresponds to using the multi-precision approach with 4-bit LUTs.
[b] Corresponds to the multi-precision benchmark for 16 bits.
[c] Corresponds to the multi-precision benchmark for 32 bits.

LUTs, respectively [WHS+24].[9] Our amortized runtimes in Table 1 are 0.722 and 4.577 ms, respectively, implying that our method has an amortized time that is two orders of magnitude smaller.

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 $e^{2\pi xi}$, 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 118, two squarings, and a polynomial of degree 4095.

[9] In [WHS+24], the experiments were run on a CPU system with i9-11900K @ 3.50 GHz and 32 GB RAM, with AVX-512 support.

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]. 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 3.4x to 8.4x, with the speed-up increasing with p (for $\log p \geq 3$).

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	–	0.188/0.244
3	6.7	1.3	–	0.327/0.383
4	6.7	1.5	–	0.375/0.431
8	6.7	5.3	–	0.722/0.778
9	6.7	6.7	2,960	0.976/1.032
12	39.1	–	10,760	4.577/4.633

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 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 grows). 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 0.72 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.

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 (which can lead to potentially smaller parameters). 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 [LMP22]. 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 [LMP22], 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, KS23, LMP22]. 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, LMP22].

Of special practical interest is the multi-precision sign evaluation capability due to its linear increase of complexity with $\log P$ [LMP22]. The high-level algorithm for evaluating the multi-precision sign function is depicted in Algorithm 5. Here, $\mathsf{HomFloor}$ 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 [LMP22]

procedure $\mathsf{HomSign}(Q, (\boldsymbol{c}, d))$
  while $Q > q$ **do**
    $(\boldsymbol{c}, d) \leftarrow \mathsf{HomFloor}(Q, (\boldsymbol{c}, d))$
    $(\boldsymbol{c}, d) \leftarrow \left\lceil \frac{\alpha}{q} \cdot (\boldsymbol{c}, d) \right\rfloor$ ▷ $\alpha = q/p$, for p the plaintext modulus of the digit
    $Q \leftarrow \alpha Q / q$
  $d \leftarrow d + \beta$
  $(\boldsymbol{a}, b) \leftarrow (q/Q) \cdot (\boldsymbol{c}, d)$
  $(\boldsymbol{c}, d) \leftarrow (-\mathsf{Boot}[f_{MSB}](\boldsymbol{a}, b)) \pmod{Q}$
  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 $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}^2_{Q'_\ell}$, $\mathsf{swk} = (\mathsf{swk}_0, \mathsf{swk}_1)$ [10] output
$$\left(\boldsymbol{c}_0 + \left\lceil \frac{\langle \mathcal{WD}_\ell(\boldsymbol{c}_1), \mathsf{swk}_0 \rangle}{P'} \right\rfloor, \left\lceil \frac{\langle \mathcal{WD}_\ell(\boldsymbol{c}_1), \mathsf{swk}_1 \rangle}{P'} \right\rfloor \right) \pmod{Q'_\ell}.$$

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

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.

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 $\mathsf{ct} \in \mathcal{R}^2_\ell$, 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 $\{\tilde{Q'}_j\}_{0\le j<\mathsf{dnum}} = \{\prod_{i=j\alpha}^{(j+1)\alpha-1} q'_i\}_{0\le j<\mathsf{dnum}}$, 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

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

- $\mathsf{KeySwitch}_{\mathsf{swk}}(\mathsf{ct})$. For $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_{Q'_\ell}^2$, $\mathsf{swk} = (\mathsf{swk}_0, \mathsf{swk}_1)$ [11] output

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

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 Sections 3 and 4

C.1 Analytical Expressions for Modular Reduction and Step Functions

The modular reduction 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.

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

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

$$\mathsf{step}_p(k) = \begin{cases} 0, & \text{if } 0 \le k < p/2 \\ p/2 & \text{if } p/2 \le k < p. \end{cases} \tag{A2}$$

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

$$\mathsf{mod}_p(k) = \mathsf{mod}_{\frac{p}{2}}(k) + \mathsf{step}_p(k), \quad p > 2 \tag{A3}$$

$$\mathsf{mod}_2(k) = \mathsf{step}_2(k). \tag{A4}$$

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

$$\mathsf{Rmod}_p(x) = \mathsf{Rmod}_{\frac{p}{2}}(2x) + \mathsf{Rstep}_p(x), \quad p > 2 \tag{A5}$$

$$\mathsf{Rstep}_p(x) = \frac{p}{4} - \frac{1}{p}\sum_{k\in S}(p-k)\left(\cos(2k\pi x) + \cot\left(\frac{\pi k}{p}\right)\sin(2k\pi x)\right), \tag{A6}$$

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 (A6) was derived from (1) and (3) via a number of simplifications.

Note that for $p = 2$, we have

$$\mathsf{Rmod}_2(x) = \mathsf{Rstep}_2(x) = \frac{1}{2} - \frac{1}{2}\cos(2\pi x), \tag{A7}$$

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 also derived analytical expressions in terms of the complex exponential function (see the proof below for derivation details):

$$\begin{aligned} \mathsf{Rmod}_p(x) &= \frac{p-1}{2} + \frac{1}{p}\sum_{k=1}^{p-1}(p-k)\left(-1 + i\cot\left(\frac{\pi k}{p}\right)\right)e^{2\pi i k x}, \\ \mathsf{Rstep}_p(x) &= \frac{p}{4} + \frac{1}{p}\sum_{k\in S}(p-k)\left(1 - i\cot\left(\frac{\pi k}{p}\right)\right)e^{2\pi i k x}, \end{aligned} \tag{A8}$$

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

Proof. For the function $\mathsf{Rstep}_x(p)$ approximating

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

we can evaluate the closed formulae for α_i from (4):

$$\begin{aligned} \alpha_0 &= \frac{1}{p}\left(\sum_{l=p/2}^{p-1}\frac{p}{2}\right) = \frac{p}{4}, \\ \alpha_k &= \frac{p-k}{p}\sum_{l=p/2}^{p-1} e^{-2\pi i k l/p} = \frac{(p-k)}{p}e^{-\pi i k}\left(\frac{1-e^{-\pi i k}}{1-e^{-2\pi i k/p}}\right) = \end{aligned} \tag{A9}$$

$$= \frac{p-k}{p} e^{-\pi i k} \left(\frac{1-(-1)^k}{2i \sin\left(\frac{\pi k}{p}\right) e^{-i\pi k/p}} \right). \tag{A10}$$

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

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

□

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

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 (7), the expression for $R_2(x)$ can be written as[12]

$$\begin{aligned}
R_2(x) &= \sum_{l=0}^{p-1} f(l) \cdot U_2\left(2\pi\left(x - \frac{l}{p}\right)\right), \text{ where} \\
U_2(x) &= U(x) + \frac{1-\cos(px)}{p^3} \sum_{k=1}^{\lfloor p/2 \rfloor} (2 - \gamma_{p,k}) k(p-k) \cos(kx),
\end{aligned} \tag{A11}$$

$\gamma_{p,k} = 1$ if p is even and $k = p/2$, while $\gamma_{p,k} = 0$ otherwise. Here, $U(x)$ is borrowed from (3.1).

We now derive the complex exponential expression for (A11). We transform the second summand of $U_2(x)$:

$$\begin{aligned}
U_2'(x) &= \frac{1-\cos(px)}{p^3} \sum_{k=1}^{\lfloor p/2 \rfloor} (2 - \gamma_{p,k}) k(p-k) \cos(kx) \\
&= \frac{1}{p^3} \left(\sum_{k=1}^{\lfloor p/2 \rfloor} (2 - \gamma_{p,k}) k(p-k) \left(\cos(kx) - \frac{1}{2} \left(\cos((p+k)x) + \cos((p-k)x) \right) \right) \right).
\end{aligned} \tag{A12}$$

Next we switch to the complex exponential formulation:

[12] 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 $k = p/2$.

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

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

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):

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

where $c_k := 2k(p-k)(2p-k)/(3p^4)$ and $e_k := e^{-2\pi kli/p}$:

$$\beta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_k, \quad \delta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_{p+k}, \quad \theta_k = c_k \cdot \sum_{l=0}^{p-1} f(l) \cdot e_{p-k}. \qquad \square$$

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 (9), the expression for $R_3(x)$ can be written as

$$\begin{aligned} R_3(x) &= \sum_{l=0}^{p-1} f(l) \cdot U_3\left(2\pi\left(x - \frac{l}{p}\right)\right), \text{ where} \\ U_3(x) &= U(x) + \frac{2\left(1 - \cos(px)\right)}{3p^4}\sum_{k=1}^{p-1} k(p-k)(2p-k)\cos(kx). \end{aligned}$$

Here, $U(x)$ is as in (3.1).

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

C.3 Bounds on Constants

Proof of Corollary 2. Using Corollary 1, Theorem 2 and 3, we can write the M-th order trigonometric Hermite Interpolation as

$$R(x) = Re\left(\sum_{k=0}^{d} \eta_k \cdot e^{2\pi ikx}\right),$$

where for all $0 \leq k \leq d$, $|\eta_k| \leq B_M$, and

$$M = 1: \quad d = p-1, \quad B_1 \leq 2\frac{(p-1)^2}{p}$$
$$M = 2: \quad d = \frac{3p}{2}, \quad B_2 \leq \frac{9}{4}(p-1)$$
$$M = 3: \quad d = 2p-1, \quad B_3 \leq \frac{(p-1)^2(p+1)(2p-1)}{p^3}.$$

Note that all bounds on B_M are smaller than $\frac{9}{4}p$ for all $p \geq 2$. These bounds were obtained by taking the maximum over all terms in the corresponding series:

$$B_1 \leq \max\left\{p-1, 2\frac{(p-1)^2}{p}\right\},$$
$$B_2 \leq \left\{\max\{p-1, 2\frac{(p-1)^2(p+1)}{p^2}, \frac{9}{4}(p-1), \frac{|p^2-3p+1|(p-1)}{p^2}\right\}, \tag{A13}$$
$$B_3 \leq \max\left\{p-1, \frac{(p-1)^2(p+1)(2p-1)}{p^3}, \frac{4\sqrt{3}}{27}(p-1)\right\}. \tag{A14}$$

Writing the $M+1$-th derivative, we obtain:

$$\left|R^{M+1}(x)\right| = \left|Re\left((2\pi i)^{M+1}\sum_{k=0}^{d}\eta_k k^{M+1}e^{2\pi ikx}\right)\right| \leq (2\pi)^{M+1}B_M\sum_{k=0}^{M+1}k^{M+1}.$$

Writing it explicitly for $M = 1, 2, 3$ and dividing by $(M+1)!$, we obtain:

$$C_1 \leq \frac{4\pi^2}{2}B_1\frac{(p-1)p(2p-1)}{6} = \frac{\pi^2 B_1}{3}(p-1)p(2p-1) \leq \frac{2\pi^2}{3}(p-1)^3(2p-1).$$
$$C_2 \leq \frac{8\pi^3}{6}B_2\frac{(3p/2)^2(3p/2+1)^2}{4} = \frac{3\pi^3 B_2}{16}p^2(3p-2)^2$$
$$\leq \frac{27\pi^3}{64}(p-1)p^2(3p-2)^2.$$
$$C_3 \leq \frac{16\pi^4}{24}B_3\frac{(2p-1)2p(4p-1)(3(2p-1)^2+3(2p-1)-1)}{30}$$
$$\leq \frac{2\pi^4}{45}B_3 p(2p-1)(4p-1)(12p^2-6p-1)$$
$$\leq \frac{2\pi^4}{45}\frac{(p-1)^2(p+1)(2p+1)^2(4p-1)(12p^2-6p-1)}{p^2}.$$

We thus obtain the bounds in Corollary 2 and replace also the values of B_M. Note that these are very loose bounds. □

D Homomorphic Digit Decomposition

Lines 4 through 6 in Algorithm 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 [LMP22].

Algorithm 6 Homomorphic digit decomposition for an RLWE ciphertext

procedure $\mathsf{HomDigitDecomp}(\mathsf{ct} \in \mathcal{R}_Q^2)$
  $k \leftarrow 0$
  while $Q > q$ **do**
    $\mathsf{ct}_k \leftarrow \mathsf{ct} \mod q$ ▷ Extract the RLWE digit encrypting a digit in $\mathbb{Z}_p^w$.
    $\mathsf{ct}_d \leftarrow \mathsf{FuncBT}_{q',Q'_L,\Delta}(\mathsf{ct}_k, \mathsf{LUT}(\mathsf{Rmod}_p(x)))$
    $\mathsf{ct} \leftarrow \mathsf{ct} - \mathsf{ct}_d$
    $\mathsf{ct} \leftarrow \mathsf{ModSwitch}(\mathsf{ct}, Q/p)$
    $Q \leftarrow Q/p, P \leftarrow P/p$
    $k \leftarrow k + 1$
  return $(\{\mathsf{ct}_k\}_{i\in[k]}, \mathsf{ct})$

E Noise Analysis for S-box

The noise reduction for S-box is illustrated in Fig. A3. For the first-order interpolation, the constant C is bounded by $4.31 \times 10^7 < 4\pi^2 B_1 p^2$. For the second-order interpolation, the constant C is bounded by $3.36 \times 10^{10} < 8\pi^3 B_2 p^3$. For the third-order interpolation, the constant C is bounded by $2.1 \times 10^{13} < 16\pi^4 B_3 p^4$.

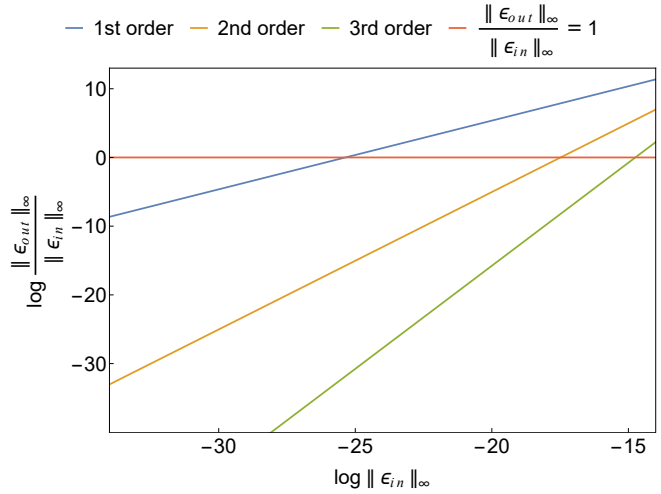

Fig. A3. Noise reduction for different interpolation orders when evaluating the S-box (at $p = 256$). ϵ_{in} is the input error and ϵ_{out} is the output error after functional bootstrapping. The maximum error ϵ that provides correctness is $\frac{1}{2p} = 2^{-9}$. The line $\|\epsilon_{out}\|_\infty / \|\epsilon_{in}\|_\infty = 1$ corresponds to zero noise reduction (anything above this line corresponds to noise increase). The zero-noise reduction points in terms of $\log \|\epsilon_{in}\|_\infty$ are -25.4, -17.5, and -14.7 for the first, second, and third orders, respectively.

F More Implementation Details and Experimental Results

An 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 $e^{2\pi xi}$) 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.

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

For simplicity, in our multi-precision sign evaluation from Table 2, 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.

Table A3. Experiments for the isolated evaluation of a single LUT (modular reduction) 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. 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.

Interp. order	$\log p$	$\log Q$	N	$\log(Q'_L P')$	# limbs (enc, dec)	# limbs HKS	Online time (s)	Amtz. on. time (ms)	Offline time (s)
1	1	33	2^{15}	768	16 (3,3)	4	6.159	0.188	24.038
2, 3	1	33	2^{15}	747	19 (3,3)	2	10.25	0.315	45.934
1	2	35	2^{16}	1035	21 (4,4)	5	19.718	0.3	55.211
2, 3	2	35	2^{16}	1070	22 (4,4)	5	21.383	0.326	57.841
1	3	37	2^{16}	1114	22 (4,4)	5	21.464	0.327	57.047
2	3	37	2^{16}	1114	22 (4,4)	5	22.134	0.338	58.486
3	3	37	2^{16}	1151	23 (4,4)	5	23.663	0.363	60.586
1	4	38	2^{16}	1234	23 (4,4)	6	24.575	0.375	62.49
2	4	38	2^{16}	1234	23 (4,4)	6	25.447	0.388	63.439
3	4	38	2^{16}	1272	24 (4,4)	6	27.703	0.422	67.005
1	8	47	2^{16}	1535	25 (3,3)	6	47.322	0.722	89.609
2	8	47	2^{16}	1535	25 (3,3)	6	55.062	0.84	97.48
3	8	47	2^{16}	1522	26 (3,3)	5	63.054	0.962	111.129
1	9	48	2^{16}	1548	26 (3,3)	5	63.958	0.976	112.873
2	9	48	2^{16}	1548	26 (3,3)	5	76.288	1.164	125.13
3	9	48	2^{16}	1536	27 (3,3)	4	97.396	1.486	158.457
1	12	55	2^{17}	2420	32 (4,4)	11	599.9	4.577	698.705
2	12	55	2^{17}	2420	32 (4,4)	11	893.78	6.819	993.618
3	12	55	2^{17}	2475	33 (4,4)	11	1148.61	8.763	1251.85
1	14	58	2^{17}	2692	34 (4,4)	12	2130.1	16.25	2242.16
2	14	58	2^{17}	2692	34 (4,4)	12	3370.9	25.72	3493.46
3	14	58	2^{17}	2750	35 (4,4)	12	4407.3	33.62	4544.8