Sparse Hermite Interpolation Method for Discrete-CKKS Functional Bootstrapping

Andreea Alexandru[1], Andrey Kim[2], Yuriy Polyakov[1], and Hongren Zheng[3](✉)

[1] Duality Technologies, USA
{aalexandru,ypolyakov}@dualitytech.com
[2] Altbridge, USA
kimandr@altbridge.ai
[3] Tsinghua University, China
zhenghr22@mails.tsinghua.edu.cn

Abstract. Discrete CKKS is a promising approach for performing high-throughput homomorphic computations over encrypted discrete data. Although it relies on CKKS, an approximate FHE scheme, as the computation engine, discrete CKKS can achieve exact correctness. The core operation of discrete CKKS is functional bootstrapping, a mechanism which enables evaluating an arbitrary function over a bounded discrete domain by representing it as a lookup table and computing it as part of bootstrapping. Simultaneously, the same procedure enables reducing the input ciphertext noise using Hermite interpolation methods. This noise reduction feature is critical for both supporting arbitrary computations and improving the efficiency of their evaluation, by providing more noise budget between bootstrapping invocations.

In this paper, we first show that both state-of-the-art Hermite interpolation noise reduction methods by Bae et al. (ASIACRYPT'24) and Alexandru et al. (CRYPTO'25) have a limited noise reduction ability for distinct structural reasons. We then propose a new method that can efficiently overcome these limitations by using a CKKS-friendly *arbitrary-order* Hermite interpolation. We call this method "sparse" trigonometric Hermite interpolation because both constraints and coefficients have convenient sparsity properties, which allow us to achieve efficiency comparable to the fastest prior method by Alexandru et al., while attaining superior noise reduction. In the process, we develop a metric that measures the noise budget between consecutive functional bootstrapping invocations, and use it to compare all methods on equal footing. We implement our new method in OpenFHE and experimentally demonstrate its noise reduction advantage over prior methods.

1 Introduction

Homomorphic encryption (HE) enables computation on encrypted data without revealing the underlying plaintexts. Since Gentry's introduction of bootstrapping and the first fully homomorphic encryption (FHE) scheme, several families of

FHE schemes—notably, Brakerski–Gentry–Vaikuntanathan (BGV)/Brakerski–Fan–Vercauteren (BFV), Ducas–Micciancio (DM)/Chillotti–Gama–Georgieva–Izabachène (CGGI) and Cheon–Kim–Kim–Song (CKKS)—have been developed to support various computation models [Gen09, BGV14, FV12, DM15, CGGI16, CKKS17]. A distinctive feature of DM and CGGI is that these FHE schemes support evaluating an arbitrary function over a discrete encrypted scalar, as part of the bootstrapping procedure. In this setting, the evaluated function is expressed as a lookup table (LUT) that maps encrypted small integers to encrypted output small integers. Such enhanced bootstrapping capability is typically called *functional*[4] or *programmable* bootstrapping [BGGJ20, CLOT21, CJP21, LMP22, KS23].

Although functional bootstrapping was viewed for many years as a unique capability of the DM/CGGI family [BP23], it was recently extended to Single Instruction Multiple Data (SIMD) schemes, including BFV [LW23, LW24] and CKKS [BCKS24, BKSS24, AKP25], resulting in a major (2-to-3-orders-of-magnitude) throughput improvement over DM/CGGI. While the original CKKS scheme was designed for *approximate complex arithmetic*, a discrete CKKS variant based on the same bootstrapping engine was introduced by encoding discrete messages into complex slots and recovering them *exactly* after decoding. The CKKS functional bootstrapping methods were inspired by the approach proposed in [DMPS24], which uses classical complex CKKS to evaluate Boolean circuits over encrypted numbers. Subsequently, multiple works enabled the LUT evaluation with larger bitwidths (e.g., 20-bit), and derived closed-form expressions for arbitrary LUTs [CKKL24, BCKS24, BKSS24, AKP25, DAP+26]. Discrete CKKS now serves as an efficient replacement of the DM/CGGI family for the workloads that deal with large amounts of encrypted small-integer data.

The discrete-CKKS functional bootstrapping capability has also been used as a building block in many recent SIMD FHE schemes for both integer arithmetic and Boolean operations over finite rings, e.g., $\mathbb{Z}_{2^{64}}$, or supporting large-bitwidth arithmetic such as multiplication in ECDSA/RSA2048 [KN25, BK25, Kim25a, Kim25b, CPL26, GZ26, Rov26]. For large-precision integer arithmetic (128 bits and higher), these FHE schemes can already achieve a lower latency for a vector of encrypted integers than the circuit-bootstrapping approaches based on DM/CGGI for a single encrypted scalar [CPL26, GZ26].

In contrast to approximate CKKS where the bootstrapping procedure does not reduce the message noise, discrete-CKKS functional bootstrapping has a *noise cleaning* capability, making it a building block for exact FHE schemes supporting arbitrary computation. All existing discrete-CKKS functional bootstrapping approaches use (trigonometric) Hermite interpolations, which are obtained by setting certain derivatives of the interpolated function at interpolation nodes to zero [AKP25]. This noise cleaning capability is conceptually similar to noise removal via rounding in the exact FHE schemes that evaluate the decryption circuit as part of bootstrapping (the idea originally proposed by Gentry [Gen09]). However, in the case of discrete-CKKS functional bootstrapping, the input error

[4] We will use the term functional bootstrapping throughout this paper.

is not fully removed, and the current methods have significant limitations in their noise reduction ability. For example, the interpolation methods in [BKSS24] and [CKKL24] are restricted to first-order noise removal, i.e., achieve only quadratic noise reduction, as illustrated in Table 1. We also observed that the method proposed in [AKP25], which provides first-, second-, and third-order Hermite interpolations for quadratic, cubic, and quartic noise removal, respectively, in real-valued function evaluation, does not reduce the *imaginary* CKKS noise added to the ciphertext in the steps before the Hermite interpolation is invoked. Hence, in addition to limited noise reduction, the actual degree of input noise reduction of current methods is not well-understood, making it difficult to estimate and control the output noise after discrete-CKKS functional bootstrapping.

In view of this, we need new practically efficient methods that can both provide desired input noise reduction (to the point of making it practically negligible) and a framework for evaluating the noise reduction efficiency of these methods. Our main research question is informally formulated as follows:

Can we construct an efficient arbitrary-order discrete-CKKS LUT interpolant that reduces input noise to theoretically and practically negligible levels?

We answer this question affirmatively, closing the gap between discrete CKKS and FHE schemes such as DM and CGGI, where the input noise is fully removed.

Our contributions. Our main contribution is a new interpolation method, which we call Sparse Trigonometric Hermite Interpolation (Sparse-THI). This method achieves arbitrary-order noise reduction for both real- and complex-valued LUTs, where the order of noise reduction n can be selected based on the requirements of a particular application. We obtain a closed-form expression for any order n and analyze its noise cleaning ability theoretically. We also develop an optimized procedure for evaluating the interpolation, which allows us to achieve almost the same computational complexity as the Alexandru–Kim–Polyakov (AKP) method [AKP25] (for the interpolation orders currently supported by AKP), which is the fastest trigonometric Hermite interpolation method available in the literature. At the same time, the noise reduction ability of Sparse-THI is asymptotically and practically better than that of AKP. We also implement Sparse-THI in OpenFHE [AAB+22], and our performance evaluation shows that the amortized latency of our method is almost as small as that of AKP for larger values of plaintext modulus p.

We examine the previously proposed interpolation methods both theoretically and practically, and improve their results. We find that the AKP method cleans only *real* noise when implemented in terms of complex exponential funcion, while maintaining or amplifying the *imaginary* noise. We prove that the AKP method and the Bae–Kim–Stehlé–Suvanto (BKSS) method [BKSS24] are equivalent when the input lies on the real line. We also propose a first-order AKP variant where the off-circle component is cleaned via a Newton method iteration at the expense of adding two levels to the multiplicative depth (this method can also be generalized to higher-order AKP). Moreover, we derive an improved analytical expression for

Table 1. Comparison of Hermite interpolation methods for a dimension-p *real-valued* LUT based on complex-exponential-function power series evaluated using CKKS. Here, $v = v_{re} + iv_{im}$ denotes the complex input noise to Hermite interpolation; v_{im}, the imaginary part of v; #HMult, the number of homomorphic multiplications (for simplicity, here the $O(n)$ and $O(1)$ terms are ignored as their contribution is relatively small); #Levels, the number of CKKS levels needed for evaluating the polynomial.

Name	**Reduced Noise**	**#HMult**	**#Levels**
CKKL[§]	$O(\|v\|^2)$	$2\sqrt{p} + \log p$	$\log p + 2$
BKSS	$O(\|v\|^2)$	$4\sqrt{p}$	$\log p + 2^{\star}$
AKP$^{(1)}$	$O(\|v_{im}\|) + O(\|v\|^2)$	$\sqrt{2p} + \log p$	$\log p + 1$
AKP$^{(2)}$	$O(\|v_{im}\|) + O(\|v\|^3)$	$\sqrt{3p} + \log p$	$\log p + 1$
AKP$^{(3)}$	$O(\|v_{im}\|) + O(\|v\|^4)$	$2\sqrt{p} + \log p$	$\log p + 2$
Sparse-THI$^{(n)}$	$O(\|v\|^{n+1})$	$\sqrt{(n+1)p} + \frac{3}{2}\log p$	$\log(p) + 1 + \lceil \log(n) \rceil$

[§] [CKKL24] only provides numerical solution for the first interpolation order. We derive an nth-order closed-form extension Full-THI$^{(n)}$ of their method.
[⋆] We provide an improved implementation of [BKSS24] using $\approx 2.75\sqrt{p}$ homomorphic multiplications with level consumption of $\log p + 1$ for real-valued LUT.

the BKSS method. Our variant has smaller computational complexity and depth requirement ($\log p + 1$ instead of $\log p + 2$ in the initial exposition [BKSS24]). We implement the improved AKP and BKSS methods in OpenFHE. Our experimental analysis shows that the original first-order AKP method is significantly faster than the improved BKSS method, with the gap increasing for larger values of p. We also extend the Chung–Kim–Kim–Lee (CKKL) first-order LUT evaluation method [CKKL24] to an arbitrary order n, providing a general closed-form expression and showing how the generalized method can be used for constructing a discrete-CKKS functional bootstrapping procedure. We denote this generalized CKKL method as Full-THI. Our complexity analysis and implementation show that Full-THI is significantly slower than Sparse-THI for real-valued LUTs, with the same asymptotic order but smaller practical noise reduction than Sparse-THI.

To examine the noise reduction of various Hermite interpolation methods, we develop a noise cleaning evaluation framework and introduce a metric called *noise capacity* for measuring the noise budget after a functional bootstrapping invocation. We also make the implementation of all Hermite interpolation methods publicly available for reproducibility and ease of reference.[5]

1.1 Technical overview

In this section, we present a simplified description of the main technical results of our work, omitting secondary results and some details related to linear transformations for switching between different encoding formats.

[5] https://github.com/openfheorg/openfhe-development/tree/sparse-thi-experiments

Our goal is to evaluate an arbitrary function $f : \mathbb{Z}_p \to \mathbb{Z}_p$ over a vector of inputs, and we aim to construct a mapping that approximates f at p equidistant points of interest on the unit circle. The AKP method was derived for evaluating real-valued LUTs by using trigonometric Hermite interpolations in the form of truncated Fourier series (for brevity, here, we focus on the first-order interpolation)

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

for fractional $x = m/p$, with $m \in \mathbb{Z}_p$, and where the coefficients $\{a_k\}_{k=0}^{p-1}$ and $\{b_k\}_{k=1}^{p-1}$ are given in analytical form.

This Fourier series can be evaluated using any FHE scheme that supports the evaluation of large-degree polynomials over reals (or properly scaled integers), e.g., BFV, CKKS, or the Gentry–Sahai–Waters scheme [GSW13]. The authors of [AKP25] used CKKS because it is the most efficient FHE scheme for evaluating such a polynomial over vectors. But since CKKS takes vectors of complex numbers as inputs, and the Fourier series can be written as a power series over complex exponential function, which can be efficiently evaluated, the authors had to extend the problem to a complex-valued setting.

Before switching to the complex function formulation, we provide basic background on CKKS arithmetic. CKKS enables computation on SIMD complex slots $\mathbb{C}^{N/2}$, where N is the ring dimension. We focus on one slot in $\mathbb{C}$ for simplicity of exposition, and note that the same applies to all slots in parallel. For a complex number $z \in \mathbb{C}$, its encoding in one CKKS slot is $\lceil \Delta z \rfloor + e$ where Δ is a scaling factor and e is the encoding noise. For an integer message $m \in \mathbb{Z}_p$, where p is a power of two, we encode it in one CKKS slot in discrete CKKS encoding as $\mathsf{D\text{-}CKKS.Ecd}(m) = \Delta m/p + e$, which is similar to Regev's RLWE [Reg09] or BFV message encoding.

The real-valued Fourier series was extended to the following complex-valued Fourier series [AKP25]:

$$T(x) = \alpha_0 + \sum_{k=1}^{p-1} \alpha_k \exp(2\pi ikx) \qquad \forall x \in [0, 1),$$

where $\{\alpha_k\}_{k=0}^{p-1}$ are complex numbers given in analytical form.

To achieve noise cleaning, the AKP method imposes the constraints $2\Re[T(m/p)] = f(m)/p$ and $\Re[T'(m/p)] = 0$ for all $m \in \mathbb{Z}_p$, which are a natural fit given the real nature of the LUT. However, this extension to the complex domain also introduces an approximate zero in the imaginary part of the input slot, which does not get reduced by the AKP method. The evaluation of $T(x)$ takes two steps: first, EvalExp moves the input data x closer to the unit circle, then EvalLUT evaluates an interpolated polynomial over the unit circle. After EvalExp, there are two types of noise: on-circle noise and off-circle noise. We remark that EvalExp itself also introduces off-circle noise due to Chebyshev approximation of $\exp(2\pi i \cdot)$. From the view point of the input x, the *off-circle* noise can be equivalently regarded as

an *imaginary* noise v_{im}, and we will use both terms interchangbly throughout this work. The imaginary noise accumulates as more CKKS operations are performed, and eventually the imaginary-part noise starts exceeding the reduced noise in the real part of the slot. To illustrate this effect, the noise reduction in the AKP method is expressed in Table 1 as $O(|v_{im}|) + O(|v|^{n+1})$, where $|v_{im}|$ is the bound on the imaginary part, and $n = 1$, 2, or 3 and corresponds to the Hermite interpolation order.

The other alternative is the BKSS method that uses a bivariate expression $h(y, \overline{y})$ such that $h(y, \overline{y}) = f(m)/p$ and $\frac{\partial}{\partial y}h = 0$ and $\frac{\partial}{\partial \overline{y}}h = 0$ at the interpolation points $y = m/p$ for $m \in \mathbb{Z}_p$. However, the bivariate expression in [BKSS24] needs a non-trivial evaluation strategy that requires a substantially higher computational complexity (as illustrated in Table 1 and our experimental results reported later in the paper), and its non-holomorphic expression makes it hard to generalize the BKSS method to an arbitrary order. As the AKP method is more efficient and easily supports higher Hermite interpolation orders, we use it as the starting point in our attempts to derive a method that solves our research problem.

First attempt: Evaluation of real-valued Fourier series. A natural question is whether an efficient implementation with proper n-order cleaning can be obtained from evaluating the real-valued Fourier series from [AKP25], without resorting to complex exponentiation. Briefly, we want to find a different "HE-friendly" method for computing $R(x)$. This can be achieved via a bivariate formulation with variables $\sin(2\pi x)$ and $\cos(2\pi x)$: $R(x) = \sum_{j=0}^{p-1} \alpha_j \cos(2\pi x)^j + \sin(2\pi x) \cdot \sum_{j=0}^{p-2} \beta_j \cos(2\pi x)^j$. One can efficiently homomorphically compute this expression using the depth of $\log p + 1$ (the multiplication by $\sin(2\pi x)$ can be introduced in the last steps of the Paterson-Stockmeyer algorithm for the second polynomial) and roughly $2\sqrt{p} + \log p + O(1)$ homomorphic multiplications, once $\sin(2\pi x)$ and $\cos(2\pi x)$ are evaluated. Unfortunately, we determined that this formulation is numerically unstable because it amplifies small errors introduced by evaluation of $\cos(2\pi x)$ itself around the points x satisfying $\cos(2\pi x) = 1$. We provide the derivation and analysis in Appendix A.

Second attempt: Hermite interpolation with imaginary-part constraints for both values and derivatives at interpolation nodes. Given that the real constraints seem to have inherent limitations, we imposed full complex constraints $R(m/p) = f(m)/p$ and $R^{(d)}(m/p) = 0$ for all $m \in \mathbb{Z}_p$ and all derivative orders d up to n (the derivation of a closed-form expression for an arbitrary order n is presented in Appendix E; note that this extends the CKKL method). However, this yields an interpolation polynomial of degree $(n+1)p-1$ as compared to $\approx (n+1)p/2$ in the AKP method, which adds an extra multiplicative level and increases the number of homomorphic multiplications by $\sqrt{2}$. Furthermore, the hidden constants before the $(n+1)$-order derivatives are significantly larger than for the AKP method, which leads to less effective noise cleaning.

The sweet spot: Hermite interpolation with imaginary-part constraints only for derivatives at interpolation nodes. We extend the crucial observation that

$f : \mathbb{Z}_p \to \mathbb{Z}_p$ is a real-valued function and use it to design an intermediate interpolation. The first-order constraints are simplified to $2\Re[R(m/p)] = f(m)/p$ but the derivatives are still set as $R'(m/p) = 0$ for all $m \in \mathbb{Z}_p$. One surprising consequence of this set of constraints is that the middle coefficients of R (degree $p/2+1$ to degree p) vanish to zero because imposing the real constraints results in a degree-$(p/2)$ polynomial, and the derivative constraints can be satisfied by appending additional polynomials of length $p/2$. This is immediately evident from the following expression, in which the higher-degree coefficients are designed to cancel the derivatives:[6]

$$R(x) = R_0 + \sum_{k=1}^{p/2} R_k \exp(2\pi i k x) + \exp(2\pi i p x) \sum_{k=1}^{p/2} \left(-\frac{k}{k+p} R_k \right) \exp(2\pi i k x).$$

The homomorphic evaluation of R requires computing the ciphertext for $\exp(2\pi i x)$ and two degree-$(p/2)$ polynomials over this ciphertext. Their evaluation using the Paterson-Stockmeyer algorithm [PS73] is efficient as the baby/giant steps can be reused. For higher-order interpolations, a similar conclusion holds. Concretely, for order n, the evaluation requires the depth of $\log p + \lceil \log n \rceil + 1$ and roughly $\sqrt{(n+1)p} + \frac{3}{2}\log(p)$ homomorphic multiplications.

In summary, the Sparse-THI method is born from the observation that most discrete-CKKS workloads involve real-valued LUTs. By imposing *real-valued constraints* on the points themselves but *full complex constraints* on the derivatives, we halve the required degrees of freedom compared to full complex interpolation.

Comparison of Sparse-THI *with prior art.* Figure 1 compares the noise reduction ability of Sparse-THI of orders 1, 2, 3, and 5 with the CKKL, BKSS, and AKP methods using our OpenFHE implementation of these methods. All plots are "V"-shaped, where the right branch corresponds to input noise reduction, and the left branch reflects the CKKS noise cost of computing the trigonometric Hermite interpolation. Note that the AKP interpolations have a much higher CKKS noise cost (in the left branch) because the Chebyshev approximation of $\exp(2\pi i \cdot)$ has non-negligible imaginary noise and every CKKS operation invoked also accumulates the imaginary noise–even before the Hermite interpolation. They together effectively offsets the initial noise from an almost negligible level to a significantly higher value. The V-plots for CKKL, BKSS, and first-order Sparse-THI are very close to each other as they all have the same asymptotic noise reduction (as seen from Table 1), and their magnitudes differ slightly only because of the differences in the upper bounds for the constants in these expressions. The plots for orders 1, 2, 3, and 5 of Sparse-THI illustrate a progressively increasing noise reduction, with the minimum noise level of even second-order Sparse-THI being by more than two orders of magnitude smaller than the minimum noise level for third-order AKP. For fifth-order Sparse-THI, we get about 18 bits of noise budget,

[6] For readability, in this overview we reuse $R(x)$ for the interpolant under whichever constraint set is currently being discussed; the main body assigns distinct symbols (T, R, H, P) to each method

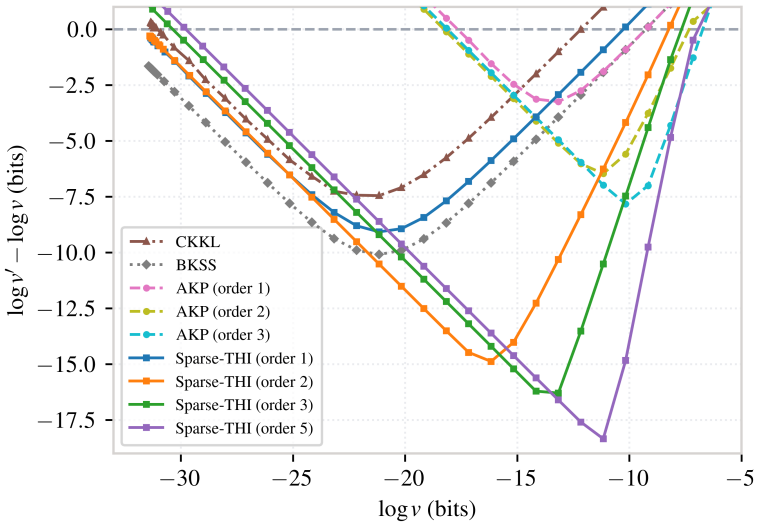


Fig. 1. Noise phase diagram depicting experimental noise reduction for various Hermite interpolation methods. We use an identity LUT of size $p = 2^4$, scaling factor $\Delta = 2^{59}$, full packing, and the FIXEDMANUAL CKKS scaling technique in OpenFHE. For third-order and fifth-order Sparse-THI, the data are obtained using ring dimension $N = 2^{17}$. Otherwise, the ring dimension is $N = 2^{16}$. The x-axis is $\log v$ (input noise), and the y-axis is $\log v' - \log v$ (noise reduction), with v and v' being the normalized input and output noises, respectively. The lower curves indicate a stronger noise reduction for a given input error. The left branch of AKP curves are much higher as they represent the amplified *off-circle* noise, whose main contribution is the Chebyshev approximation noise of the exponentiation function.

which can be used for many intermediate CKKS operations between subsequent invocations of discrete-CKKS functional bootstrapping (note that at most 1 bit of noise is consumed by one multiplication with rescaling, i.e., one multiplicative level, in CKKS [CKKS17]).

1.2 Related work

Our paper advances discrete-CKKS functional bootstrapping by proposing a new method based on trigonometric Hermite interpolation, which achieves higher noise reduction, both theoretically and practically, than the prior methods [CKKL24, BKSS24, AKP25]. The comparison is summarized in Table 1. Moreover, the computational complexity of our method is almost as low (for the same interpolation order) as that of the most efficient method proposed in [AKP25].

Dumezy *et al.* devised two discrete-CKKS functional bootstrapping methods that practically achieve a higher LUT size (20 bits vs. 14 bits in [AKP25]) by using small-precision functional bootstrapping for digit decomposition and multiplexer

trees for evaluating over the digits [DAP+26]. Their work uses AKP for small-integer functional bootstrapping. While our method does not directly compete with [DAP+26], Sparse-THI can potentially be used instead of AKP to avoid additional noise cleaning of binary values when evaluating multiplexer trees. We leave this question as a topic for future research.

A recent work by Bian *et al.* proposed a Fourier extension method to support functional bootstrapping over continuous (rather than discrete) variables in CKKS [BFS+26]. Their method can achieve a smaller function evaluation error for relatively smooth functions and attains roughly the same precision for discontinuous functions as AKP. However, their method does not reduce noise around discrete values and cannot be used in its current form to achieve exact correctness. While it may be possible to co-design noise reduction with a Fourier extension, this direction is not explored in the paper.

Other flavors of CKKS bootstrapping methods such as META-BTS [BCC+22] and EvalRound [KPK+22] achieve noise cleaning but do not provide the functional bootstrapping capability; bootstrapping methods based on scheme switching such as CHIMERA [BGGJ20] and PEGASUS [jLHH+21] offer both functional bootstrapping and noise cleaning, but their efficiency for large batch of data remains limited.

An alternative approach for functional bootstrapping is available in the context of DM and CGGI based on the technique of blind rotation. A detailed comparison between the functional bootstrapping methods in CKKS and DM/CGGI is provided elsewhere [AKP25, DAP+26].

2 Preliminaries

We use bold font $\boldsymbol{m}$ for both an element in a polynomial ring and for a vector; the context will clearly state the case. All logarithms are expressed in base 2 if not indicated otherwise. For vectors and polynomials, we use the infinity norm $\|\boldsymbol{z}\| = \max\{|z_i|\}$ unless otherwise specified.

We use $[p]$ to denote $\{0, 1, \ldots, p-1\}$. For power-of-two p, we choose representatives for $\mathbb{Z}_p$ to be $[p]$ for convenience in the paper. Centered representatives may be used in implementations.

2.1 The RLWE scheme

Let N be a power-of-two (for efficiency reasons) positive integer and let q also be a power-of-two positive integer. Let the RLWE ring be $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ with $\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1\rangle$.

For a vector of integers from $\mathbb{Z}_p$ with p being a power-of-two (e.g., 2^8), we construct a polynomial $\boldsymbol{m} \in \mathcal{R}$ such that its k-th coefficient is the k-th entry of the vector (i.e., *coefficient encoding*).

The encryption of $\boldsymbol{m} \in \mathcal{R}_q$ using a secret key $\boldsymbol{s}$ is carried out by sampling a uniformly random $\boldsymbol{a} \in \mathcal{R}_q$ and a noise $\boldsymbol{e}$, where $\boldsymbol{s}$ and $\boldsymbol{e}$ follow certain distribu-

tions [BCC+24]. The resulting ciphertext is

$$\mathsf{RLWE.Enc}_{\boldsymbol{s}}(\boldsymbol{m}) = (\boldsymbol{b}, \boldsymbol{a}) = \left(-\boldsymbol{a}\boldsymbol{s} + \frac{q}{p}\boldsymbol{m} + \boldsymbol{e}, \boldsymbol{a}\right) \in \mathcal{R}_q^2. \tag{1}$$

Its decryption can be carried out as

$$\mathsf{RLWE.Dec}_{\boldsymbol{s}}((\boldsymbol{b}, \boldsymbol{a})) = \left\lfloor \frac{p}{q}(\boldsymbol{b} + \boldsymbol{a} \cdot \boldsymbol{s}) \right\rceil_p .$$

which is an inner product followed by scaling, rounding and modulo-by-p. This maintains *exact correctness* as long as $\left\| \frac{p}{q}\boldsymbol{e} \right\| \leq \frac{1}{2}$. Note that this coincides with BFV in the coefficient encoding.

Its security is based on the RLWE problem [LPR13]. Here, we state the secret-key encryption version for simplicity. An adaptation to the public-key version is straightforward.

2.2 The CKKS scheme

CKKS [CKKS17] allows a polynomial $\boldsymbol{m} \in \mathcal{R}$ to *approximately* represent a vector of complex numbers $\boldsymbol{z} = (z_0, z_1, \ldots, z_{N/2-1}) \in \mathbb{C}^{N/2}$ through the canonical embedding (isomorphism) σ from $\mathbb{R}[X]/\langle X^N + 1\rangle$ to $\mathbb{C}^{N/2}$. Given a $(2N)$-th primitive root of unity ζ in $\mathbb{C}$, we can obtain $N/2$ primitive roots of unity $\zeta_j := \zeta^{5^j}$ with $j \in \{0, 1, \ldots, N/2 - 1\}$. The canonical embedding is defined as $\sigma(\boldsymbol{m}) = (\boldsymbol{m}(\zeta_0), \boldsymbol{m}(\zeta_1), \ldots, \boldsymbol{m}(\zeta_{N/2-1}))$. Due to the extra noise during RLWE encryption and computation, CKKS chooses a scaling factor Δ to maintain a certain precision for the message $\sigma(\Delta\boldsymbol{m}) \approx \Delta(z_0, z_1, \ldots, z_{N/2-1})$, which is referred to as *slot encoding* for a vector of messages $\boldsymbol{z} \in \mathbb{C}^{N/2}$.

The CKKS encryption of $\boldsymbol{z} \in \mathbb{C}^{N/2}$ is

$$\mathsf{CKKS.Enc}_{\Delta}(\boldsymbol{z}) = (-\boldsymbol{a}\boldsymbol{s} + \Delta\boldsymbol{m} + \boldsymbol{e}, \boldsymbol{a}) \quad \in \mathcal{R}_{Q_L}^2,$$

where $Q_L = q_0 q_1 \cdots q_L$ is composed of $(L+1)$ smaller primes q_i. As a multiplication increases the scaling factor from Δ to Δ^2, a *rescaling* operation was introduced to divide the scaling factor by q_i, transforming a ciphertext in ring $\mathcal{R}_{Q_\ell}^2$ to $\mathcal{R}_{Q_{\ell-1}}^2$. In a typical execution flow, we start from ciphertexts in $\mathcal{R}_{Q_L}^2$. After L multiplicative levels, we obtain ciphertexts in $\mathcal{R}_{q_0}^2$, which do not allow further multiplications. To support more multiplications, the technique of *CKKS bootstrapping* is used.

CKKS Bootstrapping. Due to the approximate nature of the CKKS scheme, the goal of CKKS bootstrapping is to obtain a ciphertext in $\mathcal{R}_{Q_\ell}^2$ from the ciphertext in $\mathcal{R}_{q_0}^2$, where $0 < \ell < L$, while keeping the difference of their decryption results small.

The original CKKS bootstrapping variant [CHK+18] follows the canonical template (1) ModRaise, (2) CoeffsToSlots, (3) EvalMod, and (4) SlotsToCoeffs, with a variant of (4)-(1)-(2)-(3) also possible.

After ModRaise, i.e., extending the ciphertext from $\mathcal{R}_{q_0}^2$ to $\mathcal{R}_{Q_L}^2$, its decryption result becomes $\Delta \boldsymbol{m} + \boldsymbol{e} + q_0 \boldsymbol{I} \in \mathcal{R}_{Q_L}$, where $\boldsymbol{I}$ is an integer polynomial. To remove this "side effect", first, a linear transformation (CoeffsToSlots) is applied to produce two ciphertexts (in the case of full packing) that contain the coefficient of $\boldsymbol{m}$ and $\boldsymbol{I}$ in slots. Second, an approximation of the modulo-by-q_0 operation $[\cdot]_{q_0}$ is computed, typically by evaluating the Sine function or other advanced techniques [CHKS25, BFS+26]. Then, another linear transformation (SlotsToCoeffs) is applied to move the content in slots back to the polynomial coefficients.

2.3 Discrete CKKS

Discrete CKKS [DMPS24, BKSS24, AKP25] is a variant of CKKS that operates on discrete data using functional bootstrapping with *noise cleaning*,[7] which can be used to achieve *exact correctness*. The reason for seeking exactness is that the canonical CKKS scheme is inherently approximate, which has complicated security implications [LM21, AAMP26]. Handling discrete data using discrete CKKS has many benefits such as compatibility with other FHE schemes and support of high-throughput bootstrapping.

We follow the hybrid FHE scheme specified in [AKP25] where we encrypt and decrypt in an RLWE scheme and convert the ciphertext to CKKS for functional bootstrapping because this hybrid FHE scheme natively achieves exact correctness (in contrast to the pure CKKS instantiation considered in [BKSS24, KN25]). Technically, we encrypt using the RLWE scheme, as in (1), and then we use modulus switching to reinterpret the RLWE ciphertext in $\mathcal{R}_q^2$ as a CKKS ciphertext in $\mathcal{R}_{q_0}^2$ with plaintext $\boldsymbol{m} \in \mathcal{R}_p$ in coefficient encoding and scaling factor $\Delta = q_0$.

Functional bootstrapping can evaluate a function f over a depleted ciphertext. The functional bootstrapping procedure is similar to CKKS bootstrapping and includes the following steps: (1) ModRaise, (2) CoeffsToSlots, (3) EvalExp, (4) $\mathsf{EvalLUT}_f$, and (5) SlotsToCoeffs.

The ciphertext after CoeffsToSlots is in slot encoding, and the slots of the first ciphertext (full packing produces two ciphertexts) are of the form $\Delta\left(\boldsymbol{I} + \frac{\boldsymbol{z}}{p} + \boldsymbol{v}\right)$ in $\mathbb{C}^{N/2}$, where $\boldsymbol{I}$ is a vector with integer entries introduced by ModRaise, and $\boldsymbol{z}$ is a vector with its i-th entry being the i-th coefficient of $\boldsymbol{m} \in \mathcal{R}_p$.

We call $\boldsymbol{v}$ the discrete CKKS *normalized noise*, as it is already scaled down by Δ. As long as $\boldsymbol{v}$ does not exceed $1/(2p)$ in slot encoding or in coefficient encoding, the decryption for the RLWE ciphertext succeeds with exact correctness.

Now, we examine the details of EvalExp and EvalLUT. They are defined as

$$\mathsf{EvalExp}\left(\Delta\left(\boldsymbol{I} + \frac{\boldsymbol{z}}{p}\right)\right) = \Delta\omega_p^{\boldsymbol{z}}, \quad \text{and} \quad \mathsf{EvalLUT}_f\left(\Delta\omega_p^{\boldsymbol{z}}\right) = \Delta\frac{f(\boldsymbol{z})}{p},$$

[7] There are other ways to handle discrete data with CKKS, such as [BCM+25]. There, the complex messages are decoded with fixed precision based on noise estimate, but the precision will generally decrease as the noise does not get cleaned as the computation goes on.

where $\omega_p = \exp(2\pi i/p)$ is a primitive p-th root of unity, f is an arbitrary LUT function with signature $f : \mathbb{Z}_p \to \mathbb{Z}_p$. We abuse the notation $f(\boldsymbol{z})$ to imply a vectorized application of f as $f(\boldsymbol{z}) = (f(z_i))$, and, similarly, for $\omega_p^{\boldsymbol{z}}$.

The $\mathsf{EvalLUT}$ function is obtained via an *interpolation* at the roots of unity on the unit circle. As the roots of unity are equidistant, the interpolation avoids the "Runge's phenomenon". This is a major reason for utilizing these two functions in discrete CKKS. Another reason is that $\mathsf{EvalExp}$ also removes the "side effect" of $\mathsf{ModRaise}$ when we select $\Delta = q_0$.

The noise cleaning effect of discrete-CKKS functional bootstrapping is defined as follows.

Definition 1 (Discrete CKKS Noise Cleaning). *Given a ciphertext with payload message* $\Delta\left(\boldsymbol{I} + \frac{\boldsymbol{z}}{p} + \boldsymbol{v}\right)$*, the discrete-CKKS functional bootstrapping achieves noise cleaning if*

$$\mathsf{EvalLUT}_f\left(\mathsf{EvalExp}\left(\Delta\left(\boldsymbol{I} + \frac{\boldsymbol{z}}{p} + \boldsymbol{v}\right)\right)\right) = \Delta\left(\frac{f(\boldsymbol{z})}{p} + \boldsymbol{v'}\right)$$

with $\|\boldsymbol{v'}\| \leq \|\boldsymbol{v}\|$*. For simplicity, we may use* $\log \boldsymbol{v'}$ *for denoting* $\log \|\boldsymbol{v'}\|$ *and* $\log \boldsymbol{v}$ *for denoting* $\log \|\boldsymbol{v}\|$*.*

After $\mathsf{SlotsToCoeffs}$, if we want to decrypt/decode as an RLWE ciphertext, we use $\mathsf{CKKSToRLWE}$ that switches the modulus from q_0 to q, and call $\mathsf{RLWE.Dec}$. As long as the noise is controlled properly, we obtain *exact correctness*.

3 Why existing interpolations are insufficient

In this section, we start from a baseline approach for polynomial interpolation and gradually introduce existing approaches. We underline the following metrics to pay attention to: (i) LUT type (either real-valued or complex-valued), (ii) complexity (the number of key-switching operations and number of levels consumed), and (iii) noise capacity, which we introduce below.

3.1 Noise cleaning evaluation framework

We use the same notation as in Definition 1, but for simplicity, we only focus on a single slot. Let v be the normalized input noise before $\mathsf{EvalExp}$, and v' be the normalized output noise after $\mathsf{EvalLUT}$. Both v and v' are complex values, but for simpler notation, we avoid using their magnitudes $\|v\|$ and $\|v'\|$ in the formulae below and treat them notationally as positive values in $(0, 1)$. We remark that both $\log v$ and $\log v'$ are negative numbers. Table 2 summarizes the relevant notation for our noise cleaning evaluation framework.

We assume $\mathsf{EvalLUT}$ is a *homomorphic evaluation* of an n-th order interpolation polynomial $R(\cdot) : \mathbb{C} \to \mathbb{C}$, which satisfies the following structure

$$R(m/p + v) = f(m)/p + \frac{R^{(n+1)}(m/p)}{(n+1)!}v^{n+1} + O(v^{n+2}), \tag{2}$$

Table 2. Noise cleaning evaluation concepts.

Notation	Meaning
v	Normalized input noise to the EvalExp step (complex)
v'	Normalized output noise obtained after the EvalLUT step (complex)
$\mathcal{T}$	Input noise threshold, i.e., the boundary for noise cleaning
$\mathcal{B}$	Bound for noise introduced by the CKKS evaluation
$\mathcal{I}$	Crossover point which provides maximal noise cleaning
$\mathcal{C}$	Noise capacity of the ciphertext between functional bootstrapping calls

where $m \in \mathbb{Z}_p$, and f is a function from $\mathbb{Z}_p$ to $\mathbb{Z}_p$. When the normalized noise v is small enough, we can ignore the $O(v^{n+2})$ term, and by using $C := \max_{x\in\mathbb{Z}_p}\left\{\left|\frac{R^{(n+1)}(x/p)}{(n+1)!}\right|\right\}$, we have

$$|R(m/p+v) - f(m)/p| \leq Cv^{n+1}.$$

We first study the *input noise threshold* $\mathcal{T}$, which we define as the point below which we start observing the effect of noise cleaning. Specifically, for an input noise equal to $\mathcal{T}$, it holds that $C\mathcal{T}^{n+1} = \mathcal{T}$. The solution is $\mathcal{T} = C^{-1/n}$.

Secondly, we study the *error bound* $\mathcal{B}$ corresponding to the evaluation noise for both EvalExp and EvalLUT, which includes rescaling, key switching, and other CKKS-related noise contributions. Therefore, the output noise v' can be approximated from the input noise v as follows:

$$v' \approx \mathcal{T}^{-n}v^{n+1} + \mathcal{B}. \tag{3}$$

Finally, we define a *crossover point* $\mathcal{I}$ between the above noise terms such that for an input noise $v_1 = \mathcal{I}$ and its corresponding output noise v_1', the quantity $\log v_1' - \log v_1$ is minimized. This point is where we reach *maximal noise cleaning*. The solution for $\mathcal{I}$ satisfies

$$\log \mathcal{I} = \frac{\log \mathcal{B} + n\log \mathcal{T} - \log n}{n+1}. \tag{4}$$

The expression for $\log \mathcal{I}$ is derived as follows. Let $x := \log v$, $y := \log v' - \log v$ and $r(x) = \mathcal{T}^{-n}2^{(n+1)x}$, we have $y(x) = \log(r(x) + \mathcal{B}) - x$ and $y'(x) = \frac{(n+1)r(x)}{r(x)+B} - 1$. The minimum of $y(x)$ is reached when $y'(x) = 0$, which is $nr(\log \mathcal{I}) = \mathcal{B}$. After expanding $r(\log \mathcal{I})$, we obtain $\mathcal{T}^{-n}\mathcal{I}^{n+1} = \frac{\mathcal{B}}{n}$. Taking a logarithm gives (4).

Proposition 2 (Noise Phase Diagram). *Using the above notation, the normalized output noise v' of the n-th order interpolation exhibits four regimes, determined by the normalized input noise v:*

1. Noise floor *($v \ll \mathcal{I}$):* $\log v' \approx \log \mathcal{B}$,
2. Crossover *($v \sim \mathcal{I}$):* $v' \approx \mathcal{T}^{-n}v^{n+1} + \mathcal{B}$,
3. Reduction *($\mathcal{I} \ll v \leq \mathcal{T}$):* $\log v' - \log v \approx n(\log v - \log \mathcal{T})$,
4. Amplification *($v > \mathcal{T}$):* $\log v' > \log v$.

The proof follows trivially from the definitions and equations (3) and (4).

Intuitively, the term $\mathcal{T}^{-n}v^{n+1}$ corresponds to "rounding" in classical FHE bootstrapping (this term becomes zero in Gentry-style bootstrapping because the input noise is fully removed) and the term $\mathcal{B}$ is the noise associated with evaluating the "decryption circuit".

Schemes such as the ones proposed in [Kim25a, CPL26, DAP+26], which achieve extended functionality based on discrete CKKS, introduce multiplication-like operations that increase the noise. The immediate concern is:

What is the noise budget for computations performed between consecutive functional bootstrapping invocations?

We consider the maximal budget for simplicity,[8] which is achieved when the normalized input noise v is at the *crossover point* $\mathcal{I}$. Here, the normalized output noise satisfies $v' \approx \mathcal{T}^{-n}\mathcal{I}^{n+1} + \mathcal{B} = \frac{n+1}{n}\mathcal{B}$. For the next functional bootstrapping to achieve roughly the same noise reduction, the next normalized input noise should not be larger than $\mathcal{I}$, so the maximal noise amount we can accumulate is from $\frac{n+1}{n}\mathcal{B}$ to $\mathcal{I}$. Hence, we define the *maximum noise capacity* $\mathcal{C}$ as

$$\log \mathcal{C} := \log \mathcal{I} - \log\left(\frac{n+1}{n}\mathcal{B}\right). \tag{5}$$

An illustrative demonstration of the noise concepts is presented in Figure 2. Geometrically, the point $(\log \mathcal{T}, 0)$ on the right-hand side is the input noise threshold, and the point $(\log \mathcal{B}, 0)$ is the bound for the "CKKS-induced" noise. At these two points, the input noise v is equal to the output noise v'. When the input noise is between these two points, we achieve *noise cleaning*, and the curve shows a "V"-shape. The point with x-coordinate of $\log \mathcal{I}$ is the minimum of the "V"-shape curve. Finally, as defined, $\log \mathcal{C}$ is the distance between maximal noise cleaning and the maximal normalized input noise for next functional bootstrapping.

Origin of complex noise. When we apply RLWEToCKKS and CoeffsToSlots to an input RLWE ciphertext with real noise, the real coefficients are moved into complex slots. Ideally, the imaginary part of these slots should be zero. But due to key switching and rescaling noise, the imaginary part of the slots instead contains *approximate* zeros, i.e., $\Im[v]$ is non-zero, and this imaginary-part noise should now be taken into account.

The approximation of EvalExp also adds complex noise to v. Concretely, to evaluate $R(x)$, we first evaluate complex $\exp(2\pi i x)$ using a Chebyshev series approximation. Such approximation has limited precision and introduces noise in both the $\cos(2\pi x)$ and the $\sin(2\pi x)$ directions. This noise is equivalent to a complex normalized noise v_{exp} for the normalized input noise v. In our current parameterization, the value of $\log(\Im[v_{\text{exp}}])$ is around -20. Theoretically, we can

[8] Future work using software/compiler controls can allocate the noise budget more dynamically.

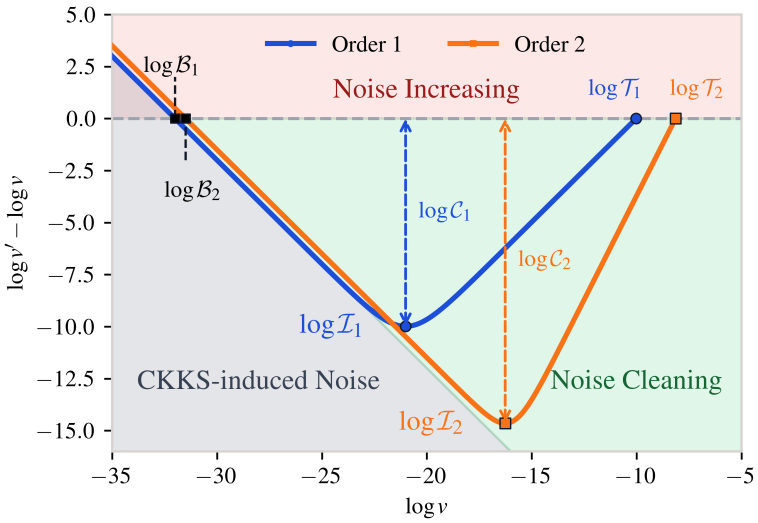


Fig. 2. Theoretical noise phase diagram for the identity LUT with $p = 2^4$ using Sparse-THI. The x-axis is $\log v$, and the y-axis is $\log v' - \log v$, with v and v' being the normalized input noise and normalized output noise. The normalized output noise v' is directly computed using (3). For illustration purposes, we use $\log \mathcal{B}_1 = -32$ and $\log \mathcal{B}_2 = -31.5$ as a higher-order interpolation generally incurs larger evaluation noise (see Table 5).

increase the precision of EvalExp by raising the Chebyshev interpolation degree, but at the cost of more key switching operations and *more levels*.

We now analyze the methods in the literature based on their efficiency and noise cleaning capacity.

3.2 Lagrange interpolation: the LUT baseline with no noise cleaning

After EvalExp, the discrete points are p-th roots of unity on the unit circle. For a complex-valued LUT defined by $f : \mathbb{Z}_p \to \mathbb{C}$, the natural baseline is Lagrange interpolation:

$$\begin{aligned} L(x) &= \sum_{k=0}^{p-1} L_k \exp(2\pi i k x) \qquad \forall x \in [0,1), \\ L_k &:= \frac{1}{p} \sum_{m=0}^{p-1} \frac{f(m)}{p} \omega_p^{-mk} = \mathsf{IDFT}\left(\frac{\boldsymbol{f}}{p}\right)_k \end{aligned} \tag{6}$$

that satisfies $L(m/p) = f(m)/p, \forall m \in \mathbb{Z}_p$. It is worth noting that the interpolation coefficients correspond exactly to the inverse[9] discrete Fourier transform of the scaled LUT values. However, the Lagrange interpolation does not generally provide noise reduction. The Lagrange interpolation gives non-zero first-order derivatives at interpolation points, and therefore a small perturbation from the interpolation points gets amplified through the non-zero derivatives.

3.3 CKKL or Full-THI: full complex Hermite constraints lead to large-degree interpolation polynomials

To achieve noise cleaning for complex-valued LUT, Chung *et al.* [CKKL24] introduced a first-order interpolation with the following conditions:

$$H(x) = H_0 + \sum_{k=1}^{2p-1} H_k \exp(2\pi i k x) \qquad \forall x \in [0,1)$$
$$\text{s.t.} \qquad H\left(\frac{m}{p}\right) = \frac{f(m)}{p}, \qquad H'\left(\frac{m}{p}\right) = 0, \qquad \forall m \in \mathbb{Z}_p,$$

which they numerically solve to find the coefficients of the interpolation polynomial.

In Appendix E, we extend this method to an arbitrary-order closed-form analytical solution. We call this interpolation method Full-THI. However, the degree of the interpolated polynomial is $(n+1)p - 1$, the highest among all methods. This leads to a larger computational complexity and larger values of the high-order derivatives at the interpolation points, which in turn lead to a higher $\mathcal{B}$ and a lower $\mathcal{T}$.

3.4 BKSS: bivariate Hermite constraints lead to non-holomorphicity and high computational complexity

Bae *et al.* [BKSS24] used the relation $z^k = \overline{z}^{p-k}$ at interpolation points to provide a non-holomorphic bivariate polynomial $h(z, \overline{z})$ with first-order noise cleaning for complex-valued LUT, leading to a maximal degree of $\max\{4, p-1\}$. However, its evaluation involves the simultaneous evaluation of *four* polynomials, which consumes $\approx 4\sqrt{p}$ HMult and $\log p + 2$ levels[10] using the Baby-Step Giant-Step (BSGS) polynomial evaluation from [HK20]. In Section 5.2, we further optimize this method to consume $\approx 3.5\sqrt{p}$ HMult ($\approx 2.75\sqrt{p}$ for real-valued LUT) and $\log p + 1$ levels using the BSGS algorithm, but the cost still remains higher than for other methods.

[9] We use $\mathsf{DFT}(\boldsymbol{L})_m = \sum_{k=0}^{p-1} L_k \omega_p^{km}$ as the forward direction because the evaluation of $L(x)$ looks like a forward pass and outputs $f(m)$ at $x = m/p$. We define the second formula in (6) as the inverse direction.

[10] We confirmed with the authors of [BKSS24] that their implementation used $\log p + 2$ and that the references to lower depths were typos.

Regarding the noise phase diagram, we did not obtain an exact formula for $\mathcal{T}$ due to their non-holomorphic expression. On the other hand, in Section 5.3 we prove the equivalence of AKP and BKSS for real inputs and real LUTs. Therefore, we apply the AKP analysis of $\mathcal{T}$ to BKSS.

3.5 AKP: real-part Hermite constraints lead to partial cleaning

Alexandru et al. [AKP25] observed that the usual LUT functions used in discrete CKKS are real-valued $f : \mathbb{Z}_p \to \mathbb{Z}_p$. Therefore, they suggested using a Hermite-Fejér interpolation that consists of sine and cosine functions. Their first-order interpolation function $R : \mathbb{R} \to \mathbb{R}$ is written as

$$R(x) = a_0 + \sum_{k=1}^{p-1} [a_k \cos(2\pi kx) + b_k \sin(2\pi kx)] \qquad \forall x \in [0, 1).$$

However, for general p and interpolation order n,[11] its evaluation is not HE-friendly,[12] and they proposed using $T : \mathbb{C} \to \mathbb{C}$ instead, which satisfies

$$\begin{aligned} T(x) &= T_0 + \sum_{k=1}^{p-1} T_k \exp(2\pi ikx) \qquad \forall x \in [0, 1) \\ \text{s.t.} \qquad 2\Re\left[T\left(\frac{m}{p}\right)\right] &= \frac{f(m)}{p}, \qquad \Re\left[T'\left(\frac{m}{p}\right)\right] = 0 \qquad \forall m \in \mathbb{Z}_p \end{aligned} \tag{7}$$

for $T_k \in \mathbb{C}$. We use $2\Re[\cdot]$ constraints as they are HE-friendly since $2\Re[z] = z + \overline{z}$, without the extra division by 2.

However, after the real values are converted to complex values to perform discrete-CKKS functional bootstrapping, non-negligible noise appears in the imaginary part due to CKKS operations, which is explained in Section 3.1.

As a result, when we evaluate $T(\cdot)$ using EvalExp and EvalLUT, by following the expansion in (2) and using $\Re\left[T'(m/p)\right] = 0$, we obtain

$$\begin{aligned} T(m/p + v) &= T(m/p) + \Im[T'(m/p)](i\Re[v] - \Im[v]) + O(|v|^2), \\ f(m)/p + v' &= 2\Re[T(m/p)] - 2\Im[T'(m/p)]\Im[v] + O(|v|^2), \\ v' &= -2\Im[T'(m/p)]\Im[v] + O(|v|^2). \end{aligned} \tag{8}$$

We remark that $\Im[T'(m/p)]$ is typically non-zero and the output noise v' is determined by CKKS noise. Therefore, we found that in the complex-valued CKKS scenario, AKP cleans the real noise $\Re[v]$, but does not reduce the imaginary noise $\Im[v]$.

In order to make (8) compatible with the analysis in Section 3.1, we use $O(|v|^2)$ to calculate $\mathcal{T}$, and treat $-2\Im[T'(m/p)]\Im[v]$ as part of $\mathcal{B}^{\mathsf{AKP}}$.

[11] The expression of $p = 2, n = 1$ only involves cosine so it is not affected by our analysis.

[12] In Appendix A, we have also explored the possibility of evaluating $R(x)$ in a Chebyshev basis with sine and cosine, but found it to be ill-conditioned.

4 Sparse trigonometric Hermite interpolation: reduced complex noise for real-valued LUT

In this section, we focus on real-valued LUTs of the form $f : \mathbb{Z}_p \to \mathbb{Z}_p$. A complex-valued LUT can be evaluated using two real-valued LUTs, and the details are presented in Appendix G.1.

Given that the normalized input noise is complex, we need to clean both real and imaginary noise. The AKP method, which has the lowest computational complexity among all prior methods, does not achieve this. Therefore, our main design goal is to devise a method that can efficiently evaluate a real-valued LUT, achieving controlled cleaning of both real and imaginary noise.

We use the EvalExp-then-EvalLUT template. The interpolation function is expressed as a univariate polynomial, with the degree d not yet fixed:

$$P(x) = P_0 + \sum_{k=1}^{d} P_k \exp(2\pi i k x) \qquad \forall x \in [0, 1). \tag{9}$$

Now, the real-valued LUT constraints can be formally stated as follows, where we prefer to use $2\Re[\cdot]$ to avoid the division by 2

$$2\Re\left[P\left(\frac{m}{p}\right)\right] = \frac{f(m)}{p} \qquad \text{for } m \in \mathbb{Z}_p. \tag{10}$$

4.1 First-order Sparse-THI

For the first-order interpolation, we set the first derivative to zero (both for real and imaginary parts), i.e., we use the constraints

$$P'\left(\frac{m}{p}\right) = 0 \qquad \text{for } m \in \mathbb{Z}_p. \tag{11}$$

Since we impose only real (as opposed to full complex as in Full-THI) constraints on the evaluated values but full complex constraints on the derivatives, we call this method Sparse Trigonometric Hermite Interpolation (Sparse-THI). The *sparsity* will also refer to zero-valued coefficients, as detailed below.

We first prove that such constrained system has a closed-form solution. The intuition for finding such a solution is that we can process the constraints layer by layer. The LUT constraints give us a degree-$(p/2)$ polynomial, while the derivative constraints force higher order terms to suppress the effect of lower-degree terms.

Theorem 3. *Given target values* $\boldsymbol{f}$ *and* $\boldsymbol{L} = \mathit{IDFT}(\boldsymbol{f}/p)$*, the first-order* Sparse-THI *method has a solution for* (9)–(11) *with* $d = \frac{3p}{2}$ *and coefficients*

$$P_0 = \frac{L_0}{2}, \qquad P_{p/2} = \frac{3}{4}L_{p/2}, \qquad P_{3p/2} = -\frac{1}{4}L_{p/2},$$
$$P_k = \frac{p+k}{p}L_k, \qquad P_{p+k} = -\frac{k}{p}L_k, \qquad \text{for } k = 1, \ldots, \frac{p}{2} - 1,$$

with other coefficients being zero.

Proof. Let $z := \exp(2\pi i x)$. We now write $P(x)$ as a polynomial with z as the variable: $P(z) = \sum_{k=0}^{d} P_k z^k$. We can thus express the derivative constraints as $P'(\omega^m) = 0$ for all m with ω being a p-th primitive root of unity. We will state and prove two useful propositions to build the proof on.

First, we achieve the LUT functionality $2\Re[P(\omega^m)] = f(m)/p$. Since $z^{\ell p+k} = z^k$ at the roots of unity for any integer ℓ, $P(z)$ is evaluation-equivalent to an effective polynomial $E(z) = \sum_{k=0}^{p-1} e_k z^k$, which can have a solution as follows.

Proposition 4. *For a real-valued LUT $f : \mathbb{Z}_p \to \mathbb{Z}_p$ and $\boldsymbol{L} = \mathsf{IDFT}(\boldsymbol{f}/p)$, the minimal-degree effective polynomial $E(z) = \sum_{k=0}^{p-1} e_k z^k$ satisfying $2\Re[E(\omega^m)] = f(m)/p, \forall m \in \mathbb{Z}_p$ has a degree (at most) $p/2$ with coefficients*

$$e_0 = L_0/2, \qquad e_{p/2} = L_{p/2}/2, \qquad e_k = L_k, \qquad \forall k \in \{1, 2, \ldots, p/2 - 1\}.$$

Proof of Proposition 4. The standard Lagrange solution for $L(\omega^m) = f(m)/p$ is $L(z) = \sum_{k=0}^{p-1} L_k z^k$. Because f is real-valued, the coefficients exhibit conjugate symmetry and we have $L_k = \overline{L_{p-k}}$. Since $\Re[L_k z^k + L_{p-k} z^{p-k}] = \Re[L_k z^k + \overline{L_k z^k}] = 2\Re[L_k z^k]$ on the unit circle for $z = \omega^m, m \in [p]$, we can fold frequencies $k > p/2$ into the lower spectrum. More specifically, a polynomial $K(z) = L_0 + L_{p/2} z^{p/2} + \sum_{k=1}^{p/2-1} 2L_k z^k$ satisfies $\Re[K(\omega^m)] = f(m)/p$. Therefore, we can set e_k as above to satisfy our LUT constraints (note the factor of $1/2$). On the other hand, $e_{p/2}$ does not fold, and thus the minimum degree is generally $p/2$.

From this proposition, the effective polynomial $E(z)$ is a degree-$(p/2)$ polynomial. To satisfy the Hermite constraints without altering the LUT values, we will decompose each effective coefficient e_k into a base term P_k and a high-frequency corrector P_{p+k}.

Proposition 5. *For $P(z)$ with degree less than $2p$ that is equal to $E(z)$ when evaluated at $\omega^m, \forall m \in [p]$, $P(z)$ has degree $d = \frac{3p}{2}$ with coefficients*

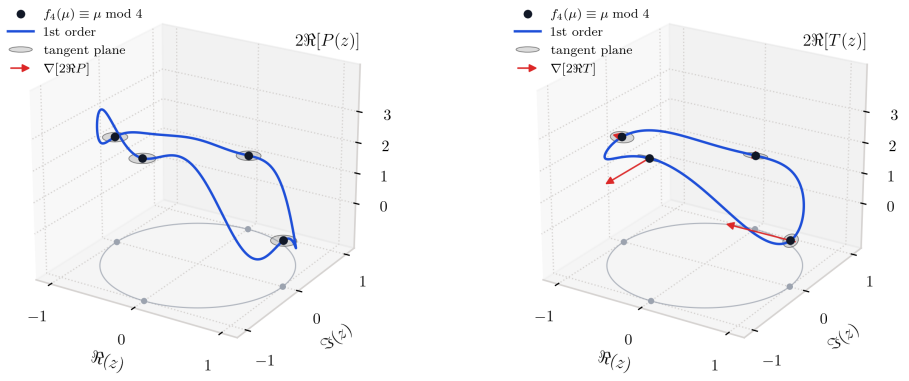
$$P_0 = e_0, \qquad P_k = \frac{p+k}{p} e_k, \qquad P_{p+k} = -\frac{k}{p} e_k, \qquad \forall k \in \{1, 2, \ldots, p/2\},$$

with all other coefficients being zero.

Proof of Proposition 5. We can write $P'(z) = \sum_{k=1}^{d} k P_k z^{k-1}$. The condition $P'(\omega^m) = 0, \forall m \in \mathbb{Z}_p$ implies that $(z^p - 1) \mid P'(z)$. Using $z^{p+k-1} \equiv z^{k-1} \pmod{z^p - 1}$ and $P'(z) \equiv 0 \pmod{z^p - 1}$, we obtain $(p+k)P_{p+k} + kP_k = 0$ for each frequency k. Simultaneously, the effective polynomial equivalence in order to maintain the LUT values gives us $P_{p+k} + P_k = e_k$. The degree of P is, therefore, $d = p + p/2 = 3p/2$.

Expanding e_k in P_k gives us the result in the statement of Theorem 3. □

We plot the first-order Sparse-THI in Figure 3a, and first-order AKP in Figure 3b for comparison, with $z := \exp(2\pi i x)$ as variable. Writing $F(z) := 2\Re[P(x)]$ and viewing it as a real-valued function $F(u, v)$ on $\mathbb{R}^2$ with $z = u + iv$,



(a) Sparse-THI (b) AKP

Fig. 3. Plots of trigonometric interpolations for $f_4(\mu) = \mu \bmod 4$ on the unit circle, $\mu \in \mathbb{Z}_4$. The division by p in the LUT constraints is ignored for this exposition. The blue curves are the lifted real interpolants for the data values depicted as black dots. The grey discs depict the tangent planes at the interpolation points and the red arrows show the (planar) gradients. Sparse-THI erases the full $\nabla[2\Re P]$, so the gradient vanishes at every interpolation point and the discs are horizontal, signaling first-order stability against complex perturbations. AKP erases only the tangential component of $\nabla[2\Re T]$, leaving a radial residual, so the discs tilt radially and the gradients have non-zero radial components, signaling first-order instability under any off-circle perturbation.

we can compute its gradient $\nabla F(u,v) = \frac{1}{\pi}(u\Im[P'(x)] - v\Re[P'(x)], u\Re[P'(x)] + v\Im[P'(x)])$ using the chain rule $2\pi i z \mathrm{d}x = \mathrm{d}z$. A perturbation $\delta \in \mathbb{C}$ of an interpolation point $z_m := \exp(2\pi i m/p), m \in \mathbb{Z}_p$ shifts F by $\langle \nabla F(z_m), (\delta_u, \delta_v) \rangle$ at interpolation points to first order, so stability in direction δ requires this inner product to vanish. Sparse-THI sets $P'(m/p) = 0$ in both real and imaginary parts, making $\nabla F(z_m) = 0$, therefore, the function is locally flat to first order in every direction. However, AKP constraints only force $\Re[R'(m/p)] = 0$. We can check that the radial derivative is $\langle \nabla F(u,v), (u,v) \rangle = \Im[P'(x)]/\pi$ while the tangential derivative is $\langle \nabla F(u,v), (-v,u) \rangle = \Re[P'(x)]/\pi$. Therefore the radial component is unconstrained, and off-circle perturbation is not cleaned. We also note that the extra complexity of Sparse-THI compared to AKP, visible from the figures, illustrates that the noise cleaning does not come for free.

4.2 Arbitrary-order Sparse-THI

We now extend the Sparse-THI approach to an *arbitrary* order n to achieve $(n+1)$-th-order noise reduction (note that the maximum order supported by

prior works is 3 [AKP25]). The set of constraints becomes

$$\begin{aligned} & P(x) = P_0 + \sum_{k=1}^{d} P_k \exp(2\pi i k x) \qquad \forall x \in [0,1), \\ \text{s.t.} \quad & 2\Re\left[P\left(\frac{m}{p}\right)\right] = \frac{f(m)}{p} \qquad \text{for } m \in \mathbb{Z}_p, \\ & P^{(\ell)}\left(\frac{m}{p}\right) = 0 \qquad \text{for } m \in \mathbb{Z}_p,\ \ell \in \{1,2,\ldots,n\}. \end{aligned}$$

Theorem 6. *Using $\boldsymbol{L} = \mathit{IDFT}(\boldsymbol{f}/p)$, the order-$n$* Sparse-THI *method has a solution for $d = \frac{(2n+1)p}{2}$, with coefficients*

$$P_0 = \frac{L_0}{2}, \qquad P_{\ell p + p/2} = (-1)^\ell \binom{n}{\ell} \frac{\prod_{j=0, j\neq\ell}^{n}(2j+1)}{2^n \cdot n!} \cdot \frac{L_{p/2}}{2},$$
$$P_{\ell p + k} = (-1)^\ell \binom{n}{\ell} \frac{\prod_{j=0, j\neq\ell}^{n}(jp+k)}{n! \cdot p^n} \cdot L_k,$$

for $k = 1, 2, \ldots, p/2 - 1$ and $\ell = 0, 1, \ldots, n$, with all other coefficients being zero.

Proof. We use the effective polynomial $E(x)$ argument from the proof of Theorem 3 to satisfy the LUT constraints. To satisfy the derivative constraints, we match high-degree terms $P_{\ell p+k}$ with P_k for every order of derivatives. This yields a degree $d = (2n+1)p/2$-polynomial, and its solution using the Vandermonde matrix yields the answer. The full derivation is in Appendix C. □

4.3 Efficient evaluation of Sparse-THI

The Sparse-THI polynomial $P(z)$ has a special structure with sparse coefficients, which enables a more efficient evaluation of $P(z)$. Concretely, note that its coefficients satisfy $P_{\ell p+k} = 0$ for $\ell = 0, 1, \ldots, n-1$ and $k = p/2+1, \ldots, p$, so we can rewrite the degree $(2n+1)p/2$ polynomial as follows:

$$\begin{aligned} P(z) &= P_0 + \sum_{k=1}^{p/2} P_k z^k + z^p \sum_{k=1}^{p/2} P_{p+k} z^k + \cdots + z^{np} \sum_{k=1}^{p/2} P_{np+k} z^k \\ &= P_0 + P_{(0)}(z) + z^p P_{(1)}(z) + \cdots + z^{np} P_{(n)}(z) \\ &= P_0 + \sum_{\ell=0}^{n} z^{\ell p} P_{(\ell)}(z), \end{aligned}$$

where we use $P_{(\ell)}$ to denote the ℓ-th polynomial. The problem thus reduces to evaluating $(n+1)$ polynomials of degree-$\frac{p}{2}$ and the n monomials $z^{\ell p}$ for $\ell = 1, \ldots, n$.

Number of Homomorphic Multiplications. We first recall the Paterson-Stockmeyer evaluation of one polynomial as in the Appendix H [PS73]. For input

u and polynomial $F(x)$ with degree $p/2$, we first choose k and m such that $k(2^m-1) > p/2$. We then prepare baby-steps $\mathbf{bs} = (u, u^2, \ldots, u^k)$, giant-steps $\mathbf{gs} = (u^k, u^{2k}, \ldots, u^{2^{m-1}k})$, and an extra term $u^{k(2^m-1)}$. This preparation process consumes $(k-1)+2(m-1)$ HMult. We then use long division to recursively obtain $F(u)$. The recursion consumes $2^{m-1}-1$ HMult.

The evaluation of $(n+1)$ polynomials on the same input u can share the same preparation process. Therefore, the number of HMult consumed for recursion can be reduced to $(n+1)(2^{m-1}-1)$.

For the preparation of all $u^{\ell p}$ for $\ell = 1, \ldots, n$, as p is a power-of-two, we first pick $k' = 2^{\lfloor \log k \rfloor}$ and find $u^{k'}$ from $\mathbf{bs}$ from which we compute u^p. This consumes $\log p - \lfloor \log k \rfloor$ HMult. Then, we use u^p to prepare all $u^{\ell p}$ analogously to the baby steps, which consumes $n-1$ HMult. The multiplication of all $u^{\ell p}$ with $P_{(\ell)}(u)$ also consumes n HMult.

To minimize the number of key switching operations, we find the k that minimizes $(k-1)+2(m-1)+(n+1)(2^{m-1}-1)+(\log p - \lfloor \log k \rfloor)+(2n-1)$. The solution is $k \approx \frac{1}{2}\sqrt{(n+1)p}$ and $m \approx \frac{1}{2}\log(p/(n+1))$, and the asymptotic number of HMult is $\sqrt{(n+1)p} + \frac{3}{2}\log p + O(n)$.

Depth. The number of levels needed is $\lceil \log(n) \rceil + \log(p) + 1$, as the preparation of z^{np} takes $\lceil \log n \rceil + \log p$ levels.

In contrast, a naive evaluation of degree-$\frac{(2n+1)p}{2}$ polynomial would take $\sqrt{(2n+1)p} + \log((2n+1)p)$ HMult and consume $\left\lceil \log\left(n+\frac{1}{2}\right) \right\rceil + \log p$ levels.

4.4 Noise cleaning for Sparse-THI

In this section, we present exact formulae for analyzing the noise cleaning effect of Sparse-THI, using our closed-form solutions for an arbitrary order.

Theorem 7 (Sparse-THI input threshold). *Given a LUT function $f : \mathbb{Z}_p \to \mathbb{Z}_p$ and corresponding order-n Sparse-THI interpolation $P(\cdot)$, the LUT-specific input threshold is $\mathcal{T}_{f,n}^{\text{Sparse-THI}}$, satisfying*

$$\log \mathcal{T}_{f,n}^{\text{Sparse-THI}} = -\left(1+\frac{1}{n}\right)\log(2\pi p) - \frac{1+\log M_{f,n}}{n},$$
$$M_{f,n} = \max_{x \in \mathbb{Z}_p} \left| \sum_{k=1}^{p/2} e_k \omega_p^{kx} \frac{\Gamma(n+1+k/p)}{\Gamma(n+2)\Gamma(k/p)} \right|.$$

where e_k are the coefficients of the effective polynomial computed as in Proposition 4. Then, for input message $y = \frac{x}{p} + v$ with $x \in \mathbb{Z}_p$, we assume that $v \ll \frac{1}{2p}$ is smaller than $\mathcal{T}_{f,n}^{\text{Sparse-THI}}$ and that the tailing term $O(v^{n+2})$ from the Taylor expansion of $P(y)$ around $P(x/p)$ is negligible compared with $P^{(n+1)}(x/p)\frac{v^{n+1}}{(n+1)!}$. By using the notation $2\Re[P(y)] = \frac{f(x)}{p} + v'$, it holds that

$$\log|v'| - \log|v| \le n(\log|v| - \log \mathcal{T}_{f,n}^{\text{Sparse-THI}}). \tag{12}$$

Proof. As the interpolated polynomial satisfies $P^{(\ell)}(x/p) = 0$ for all $x \in \mathbb{Z}_p$ and $\ell \in [n]$, we can expand it using Taylor expansion at each $x \in \mathbb{Z}_p$ as $P(x/p+v) = P(x/p) + P^{(n+1)}(x/p)\frac{v^{n+1}}{(n+1)!} + O(v^{n+2})$. We ignore $O(v^{n+2})$ from the assumption. The expression of $P^{(n+1)}(x/p)$ is derived in Appendix D as

$$P^{(n+1)}(x/p) = (-1)^n (2\pi p i)^{n+1} \sum_{k=1}^{p/2} \left(e_k \omega_p^{kx} \frac{\Gamma(n+1+k/p)}{\Gamma(k/p)} \right). \tag{13}$$

We use the usual Gamma function $\Gamma(\cdot)$ with $(n+1)! = \Gamma(n+2)$. Let $C_{f,n} := \max_{x \in \mathbb{Z}_p} \left| \frac{P^{(n+1)}(x/p)}{(n+1)!} \right|$, since $v' = 2\Re[P^{(n+1)}(x/p)v^{n+1}/(n+1)!]$ when evaluated at y, we have $|v'| \leq 2C_{f,n}|v|^{n+1}$. Let $M_{f,n}$ be defined as in this theorem, then $C_{f,n}$ can be rewritten as $(2\pi p)^{n+1} M_{f,n}$.

From the definition of the input noise threshold from Section 3.1, we have that $\log \mathcal{T}_{f,n}^{\mathsf{Sparse\text{-}THI}} = -\frac{1+\log C_{f,n}}{n}$ and obtain the target formula. □

In Table 3, we present values of $\log(\mathcal{T}_{f,n})$ for different LUTs (identity, the S-box transformation in AES, extraction of the most and least significant bits) and different orders n. We can see that (i) the threshold value shifts significantly when p increases, (ii) the threshold changes by a large amount when we increase from first order to second order, but the change decelerates as the order grows, and (iii) the choice of a different LUT has a small effect on the threshold value.

Table 3. Input noise threshold $\log \mathcal{T}_{f,n}^{\mathsf{Sparse\text{-}THI}}$ of Sparse-THI for different real-valued LUTs. The MSB (and LSB) LUT outputs $\Delta \cdot \mathrm{MSB}/2$ ($\Delta \cdot \mathrm{LSB}/2$), i.e., the output is a single bit. A larger (smaller in magnitude) value of $\log \mathcal{T}$ implies tolerating a higher input noise.

LUT	p	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
ID	2^2	-5.90	-5.10	-4.87	-4.77	-4.71
ID	2^4	-10.02	-8.15	-7.56	-7.28	-7.12
ID	2^8	-18.06	-14.16	-12.90	-12.28	-11.92
AES SBox	2^8	-18.42	-14.37	-13.05	-12.40	-12.02
MSB	2^8	-17.07	-13.67	-12.57	-12.04	-11.72
LSB	2^8	-17.89	-14.14	-12.91	-12.31	-11.95
ID	2^{10}	-22.07	-17.16	-15.57	-14.79	-14.32

5 Analysis and improvements of existing methods

As part of this study, we also improved the prior methods in [BKSS24] and [AKP25], and examined their connection to each other. We describe this here.

5.1 AKPnorm: "correction" to the unit circle

We present a solution to achieve complex noise cleaning for the implementation of [AKP25] at the expense of extra multiplicative levels. We also describe a different (failed) attempt in Appendix A, which tries to evaluate the real-valued Fourier series from [AKP25] without using a complex exponential function.

The main issue with AKP is that it does not handle *off the unit circle* noise. One remedy is the Sparse-THI method that cleans the off-circle/imaginary part noise through polynomial evaluation. Another remedy is to reduce the off-circle noise right after EvalExp and before EvalLUT, as all imaginary noise corresponds to the off-circle noise at that point.

For input $m/p + v$, where $m \in \mathbb{Z}_p$ and v is the normalized input (complex) noise, after computing $\exp(2\pi i \cdot)$, we can view the result as

$$z = \exp(2\pi i(m/p + \Re[v])) \cdot (1 - 2\pi\Im[v] + O(\Im[v]^2)).$$

The EvalLUT from AKP cleans $\Re[v]$ inside the $\exp(2\pi i \cdot)$ structure. To clean the imaginary noise before AKP, we can push the result closer to the unit circle by evaluating $z/|z|$.

Concretely, for evaluating $\frac{1}{|z|}$ from $|z|^2 = z\overline{z}$, we approximate the function $A(x) = \frac{1}{\sqrt{x}}$ using $\frac{3-x}{2}$ at $x = 1$, which corresponds to one iteration of the Newton method.

Proposition 8. *Given $z = z_0(1 + \alpha)$ with z_0 on the unit circle and a small real number $\alpha \ll 1$, let*

$$z' := z \cdot \frac{3 - z\overline{z}}{2} = \frac{z}{\sqrt[3]{2}} \cdot \left(\frac{3}{\sqrt[3]{4}} - \frac{z}{\sqrt[3]{2}} \frac{\overline{z}}{\sqrt[3]{2}} \right).$$

It holds that $z' = z_0(1 - 3\alpha^2/2 - \alpha^3/2)$.

Proof. We have $z\overline{z} = 1 + 2\alpha + \alpha^2$ and $(3 - z\overline{z})/2 = 1 - \alpha - \alpha^2/2$. Then, from the definition of z', we obtain that $z' = z_0(1+\alpha)(1-\alpha-\alpha^2/2) = z_0(1-3\alpha^2/2-\alpha^3/2)$. □

In the noise formula (8), the use of this normalization reduces the imaginary part noise from $\Im[v]$ to $O(\Im[v]^2)$. Therefore, $2\Im[T'(m/p)]\Im[v]$ becomes $2\Im[T'(m/p)]O(\Im[v]^2)$, and we achieve quadratic noise cleaning for $\Im[v]$.

However, the homomorphic evaluation of z' steps takes two additional multiplicative levels. A higher-order imaginary noise cleaning for higher orders of AKP would consume additional levels. Therefore, this improvement over AKP is not of practical interest.

5.2 Refined BKSS

We present a corrected analytical expression for BKSS as we identified several inaccuracies in the formula for the interpolant h: (i) The coefficients for the

reflected polynomials P_{f^r} and Q_{f^r} are not exactly P_f and Q_f with L_k replaced by L_{p-k}, respectively, because their index ranges are different, and (ii) the sign for Q_f and Q_{f^r} should be positive instead of negative.

$$
\begin{aligned}
h(z,\overline{z}) &= L_0 + P_f(z) + P_{f^r}(\overline{z}) - \overline{z}z(Q_f(z) + Q_{f^r}(\overline{z})),\\
P_f(z) &= \sum_{k=1}^{p-1} \alpha_k z^k \quad \text{with} \quad \alpha_k = \begin{cases} \frac{L_k}{p}(p-k)(k+1) & \text{when } k \le p/2,\\ \frac{L_k}{p}(p-k) & \text{otherwise} \end{cases}\\
P_{f^r}(z) &= \sum_{k=1}^{p-1} \beta_k z^k \quad \text{with} \quad \beta_k = \begin{cases} \frac{L_{p-k}}{p}(p-k)(k+1) & \text{when } k \le p/2-1,\\ \frac{L_{p-k}}{p}(p-k) & \text{otherwise} \end{cases}\\
Q_f(z) &= \sum_{k=1}^{p/2} \gamma_k z^k \quad \text{with} \quad \gamma_k = \frac{L_k}{p}k(p-k),\\
Q_{f^r}(z) &= \sum_{k=1}^{p/2-1} \delta_k z^k \quad \text{with} \quad \delta_k = \frac{L_{p-k}}{p}k(p-k),
\end{aligned}
\tag{14}
$$

which requires four polynomial evaluations. The authors proposed a polynomial multi-evaluation technique to reduce the number of HMult to $O(4\sqrt{p})$ while consuming $\log p + 2$ levels [BKSS24], which takes both more HMult and more depth than [AKP25] and our Sparse-THI.

For complex-valued LUT, as the degrees for Q_f and Q_{f^r} are at most $p/2$, we can further reduce the level consumption to $\log p + 1$[13] and the number of HMult to $\approx 3.5\sqrt{p}$ using the polynomial multi-evaluation technique proposed in BKSS. The complexity analysis is presented in Appendix B. The complexity for real-valued LUT can be further reduced to $\approx 2.75\sqrt{p}$, as shown in Section 5.3.

Although it is possible to extend BKSS to higher orders, the number of constraints and the number of non-zero coefficients would grow quadratically in the resulting interpolated polynomial, because the number of partial derivatives for two variables z and $\overline{z}$ increases quadratically with the derivative order, represented in Appendix B. In Sparse-THI, this growth is linear with the derivative order. The above makes higher-order BKSS less attractive than higher-order Sparse-THI.

5.3 Equivalence of BKSS and AKP for real input and real LUT on the unit circle

Consider a real-valued LUT function $f : \mathbb{Z}_p \to \mathbb{Z}_p$. We use the same notation for the Lagrange coefficients $\boldsymbol{L} = \mathsf{IDFT}(\boldsymbol{f}/p)$. Then we have $L_k = \overline{L_{p-k}}$ and L_0 is real. Let $z := \exp(2\pi i x)$ be a point *on the unit circle* where x is a real number

[13] We note that the level consumption for first-order methods of both [BKSS24] and [AKP25] can be improved to $\log p$ using BSGS in [BMTPH21]. However, we do not take this approach as the number of HMult will substantially increase, especially for BKSS, where there are four polynomials to evaluate.

in $[0, 1)$, then we obtain

$$\frac{p-k}{p}\left(L_k z^k + \overline{L_k z^k}\right) = 2\Re[T_k z^k] \quad \text{(on the unit circle)}.$$

It holds from (14) using the derivation in Appendix B.2 that

$$h(z, \overline{z}) = 2\Re[T(x)] + (1-|z|^2)\left[\sum_{k=1}^{p/2} \frac{k(p-k)}{p} L_k z^k + \sum_{k=1}^{p/2-1} \frac{k(p-k)}{p} \overline{L_k} \overline{z}^k\right].$$

On the unit circle where $|z|^2 - 1 = 0$, $h(z, \overline{z})$ becomes equivalent to $2\Re[T(x)]$ from (7). On the other hand, if we deviate from the unit circle, there are additional terms related to $(|z|^2 - 1)$ and they are generally not equivalent.

In this scenario, the evaluation of BKSS can be carried out by evaluating the degree-p polynomial $T(x)$ and the degree-$(p/2-1)$ polynomial $\sum_{k=1}^{p/2-1} \frac{k(p-k)}{p} L_k z^k$, with further processing like extracting the real part, adding the extra $\frac{p}{4} L_{p/2} z^p$, and multiplying by $(1-|z|^2)$.[14] The number of HMult can be reduced to $\approx 2.75\sqrt{p}$ using the BSGS polynomial evaluation, with the the level consumption of $\log p + 1$.

It is the extra terms with $|z|^2 - 1$ that make the cleaning in [BKSS24] work for both real and imaginary noise of x (or on-circle and off-circle noises for z). However, it is also this term that makes their function non-holomorphic and harder to analyze and evaluate. Sparse-THI achieves this imaginary-noise cleaning capability without sacrificing holomorphicity.

6 Implementation and performance evaluation

This section compares Sparse-THI with prior (improved) interpolation methods in terms of complexity, latency, and noise capacity. For an apples-to-apples comparison, all methods start from an RLWE-encrypted input and end with the decryption of the RLWE result, going through the same procedures of converting to and from discrete CKKS. Moreover, the procedure for computing the complex exponential function is the same across all methods.

6.1 Parameter selection and implementation details

We use a large scaling factor $\Delta = 2^{59}$ to illustrate the concept of noise capacity (from (5)). We use the ring dimension of $N = 2^{16}$ for LUT size $p = 2^4$ and $N = 2^{17}$ for LUT size $p = 2^8$. We select the level budgets for CoeffsToSlots and SlotsToCoeffs to be 3 and 2. The number of levels consumed by EvalExp is 8 with the degree of the Chebyshev series approximation polynomial being 58. The number of levels consumed by EvalLUT is determined by corresponding interpolation methods. We use the HYBRID key-switching technique and FIXEDMANUAL scaling technique, the same as in [AKP25].

[14] We thank an anonymous Asiacrypt reviewer for proposing this optimization.

Table 4. Experimental results for discrete-CKKS functional bootstrapping with real-valued LUTs. Here, $\Delta = 2^{59}$; for $p = 2^4$ we use $N = 2^{16}$; for $p = 2^8$ we use $N = 2^{17}$. We use full packing (requires two ciphertexts), and the number of KeySwitch operations is given per ciphertext. EvalLUT and Total denote the corresponding latencies.

p	n	Method	# KeySwitch	# Levels	EvalLUT (s)	Total (s)
2^4	1	AKP	9	5	0.724	12.4
		BKSS(Real)†	14	5	1.04	12.6
		Sparse-THI	11	5	0.874	12.5
	2	AKP	11	5	0.887	12.2
		Sparse-THI	14	6	1.28	13.3
	3	AKP	12	6	1.08	12.7
		Sparse-THI⋆	17	7	1.85	15.2
	5	Sparse-THI⋆	23	8	2.82	16.4
2^8	1	AKP	31	9	12.7	52.6
		BKSS(Real)†	49	9	18.9	58.6
		Sparse-THI	34	9	13.4	53.7
	2	AKP	36	9	17.6	56.4
		Sparse-THI	39	10	19.1	58.4
	3	AKP	40	10	22.7	62.0
		Sparse-THI	44	11	27.1	71.9

† BKSS (Real) stands for the optimized evaluation of BKSS described in Section 5.3.
⋆ This row does not satisfy 128-bit security and is only for illustration purposes.

Regarding secret key distribution, both [BKSS24] and [AKP25] used sparse secrets according to previous security estimates [BMTPH21]. Recent results [Ogi26, HJ26] have shown that a revision of the security work factor for sparse secrets is needed, but as this research area is actively evolving, we decided to run the experiments using the same security thresholds as in [AKP25] and implemented all algorithms as such, for a fair comparison.

We implemented the new methods using OpenFHE v1.5.1 [AAB+22] with HEXL enabled. We reused the AKP implementation in OpenFHE, and we implemented the BKSS for both real-valued and complex-valued LUT using the BSGS polynomial evaluation algorithm [HK20] inside the OpenFHE codebase.

6.2 Comparison and experimental results

We conducted the experiments in a single-threaded setting with AMD RYZEN AI MAX+ 395 CPU and 128GB RAM, running Debian 13, with the clang++ 19 compiler. We benchmarked discrete-CKKS functional bootstrapping. The reported data are averaged over 5 runs, with a preceding warm-up run.

Table 5. Noise capacity $\mathcal{C}$ (in bits) of different interpolation methods for real-valued LUTs. A larger $\mathcal{C}$ is better. We use $\log \Delta = 59$ and full packing. For $p = 16$, we use ring dimension $N = 2^{16}$, with the exception of Sparse-THI order 3 and 5, where we use $N = 2^{17}$; for $p = 256$, we use $N = 2^{17}$. The threshold $\mathcal{T}$ is calculated theoretically and is in agreement with the experiments. The bound $\mathcal{B}$ is measured experimentally. Negative capacity $\mathcal{C}$ means no noise reduction in the current parameter settings, and corresponding visual plots are presented in Figure A2.

LUT Size p	**Order** n	**Method**	**Threshold** $\log \mathcal{T}$	**Bound** $\log \mathcal{B}$	**Capacity** $\log \mathcal{C}$
2^4	1	AKP	-9.21	-17.64	3.21
		BKSS	-9.21	-33.08	**10.93**
		CKKL	-12.14	-31.03	8.44
		Sparse-THI	-10.52	-31.69	9.58
2^4	2	AKP	-7.23	-18.24	6.42
		Sparse-THI	-8.39	-31.67	**14.60**
2^4	3	AKP	-6.68	-18.11	7.75
		Sparse-THI	-7.72	-30.36	**16.16**
2^4	5	Sparse-THI	-7.21	-29.77	**18.15**
2^8	1	AKP	-17.21	-13.76	-2.72
		BKSS	-17.21	-25.64	**3.21**
		CKKL	-21.31	-22.07	-0.62
		Sparse-THI	-18.56	-25.36	2.39
2^8	2	AKP	-13.30	-14.30	-0.24
		Sparse-THI	-14.41	-29.48	**9.13**

Real-valued LUT. Table 4 provides experimental results in terms of latency, number of lower-level operations, and required depth. The first-order Sparse-THI has a relatively small LUT latency increase (at most 1.3× for the Hermite interpolation evaluation in the worst case of $p = 2^4$) w.r.t. first-order AKP, as it only has 2 to 3 extra key-switching operations and the level consumption is the same. Note that the total latency of discrete-CKKS functional bootstrapping is almost the same for the first-order Sparse-THI and AKP. For larger orders, Sparse-THI requires one extra level, which makes the latency gap between Sparse-THI and AKP more pronounced due to computation being performed at a higher modulus (especially in the case of the third order). Compared to BKSS (Real), the first-order Sparse-THI has a 1.2× to 1.4× speedup as the number of key-switching operations for BKSS (Real) is significantly higher than for the first-order Sparse-THI.

Table 5 depicts a noise comparison for the same methods. BKSS achieves the highest noise capacity among the first-order Hermite interpolation methods. However, its advantage over Sparse-THI is only of 1-1.5 bits, which may not be enough to justify a higher runtime cost in many scenarios. As expected, higher-order versions of Sparse-THI achieve the highest noise capacities, e.g., 18 bits is achieved by the fifth-order version for $p = 2^4$.

In addition to the noise capacity, Table 5 also reports the input threshold $\log \mathcal{T}$ and the CKKS-induced-noise bound $\log \mathcal{B}$. First, observe that as the order n increases, for Sparse-THI $\log \mathcal{B}$ increases slowly while $\log \mathcal{T}$ increases notably. Combined with the $\frac{n}{n+1}$ factor in (4) and (5), the noise capacity also increases notably rather than decreases. Second, we see that the values of $\log \mathcal{B}$ for AKP are much higher than for other methods due to the linear effect of the imaginary-part noise (this behavior corresponds to the left branch of Figure 1). We also see that $\log \mathcal{B}$ for AKP varies significantly with LUT size because a larger LUT size leads to a larger $\Im[R'(x)]$. Third, we observe that BKSS and first-order AKP have higher input thresholds than first-order Sparse-THI, which implies that the benefit from noise reduction can be observed in these methods for a higher input noise (this behavior corresponds to the right branch in Figure 1). The theoretical $\log \mathcal{T}$ for AKP is calculated using the analytical expression of its derivatives. Since the first-order AKP is faster than both BKSS and first-order Sparse-THI, first-order AKP can still be preferred over Sparse-THI in practical scenarios where the goal is to evaluate discrete-CKKS functional bootstrapping with a small number of intermediate CKKS computations. In other words, first-order AKP is preferable when we are more concerned about having an inexpensive minimal noise cleaning effect rather than achieving the maximal noise capacity between consecutive functional bootstrapping calls.

Intuitively, the reason why BKSS and first-order AKP have a higher input threshold can be understood by examining the smoothness of the interpolated curves Figure 3 and Figure A1. A curve with more oscillation generally has a larger coefficient before its smallest-order non-zero derivative term in the Taylor series expansion, resulting in a lower input threshold.

7 Concluding remarks

The performance evaluation suggests that our proposed method, Sparse-THI, achieves a much higher noise capacity–the noise budget between consecutive functional bootstrapping invocations–than AKP for all three orders supported by the latter. In terms of runtime complexity, Sparse-THI for the first order is almost as fast as first-order AKP, and the runtime increase of functional bootstrapping even for higher orders does not exceed 20%. Moreover, first-order Sparse-THI achieves almost the same noise capacity as BKSS, but has a significantly smaller latency for larger values of plaintext modulus p. The greatest benefit of Sparse-THI compared to both AKP and BKSS is that Sparse-THI supports an arbitrary-order noise reduction, which allows it to achieve in a controlled manner the noise capacity that is desired for a given application. A somewhat surprising result is that AKP and BKSS start achieving noise reduction for a higher input noise than same-order Sparse-THI, which implies there may still be applications where AKP and BKSS would be preferred over Sparse-THI.

In order to improve the noise reduction in the AKP method implementation, we first attempted to efficiently evaluate the real-valued Fourier series rather than its complex-valued extension. This attempt did not achieve numerical

stability and, as a result, we focused instead on developing a new and more complex-CKKS-native method. However, there is a conjugate-invariant version of CKKS that works natively with real numbers, and does not require an imaginary part [KS19]. It is an interesting question whether this CKKS variant can be used for an efficient implementation of the real-valued Hermite interpolation method proposed in [AKP25]. To the best of our knowledge, the conjugate-invariant CKKS version does not currently support a native bootstrapping capability, which the functional bootstrapping method could be built upon. We leave both problems for future work.

Acknowledgments. The authors are grateful to the anonymous reviewers for their comments and contributions. A. Alexandru and Y. Polyakov were funded, in part, by the Advanced Research Projects Agency for Health (ARPA-H). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The authors acknowledge the use of ChatGPT/Codex and Claude for proofreading and instantiating scripts for experiments/figures. The authors have reviewed and verified all generated content and assume full responsibility for the final manuscript.

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.

AAMP26. Andreea Alexandru, Ahmad Al Badawi, Daniele Micciancio, and Yuriy Polyakov. Application-aware approximate homomorphic encryption: Configuring FHE for practical use. *IACR Communications in Cryptology*, 2(4), 2026.

AKP25. Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using CKKS. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 304–337. Springer, Cham, August 2025.

BCC+22. Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. META-BTS: Bootstrapping precision beyond the limit. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 223–234. ACM Press, November 2022.

BCC+24. Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R. Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, Xianhui Lu, Carsten Maple, Alberto Pedrouzo-Ulloa, Rachel Player, Yuriy Polyakov, Luis Antonio Ruiz Lopez, Yongsoo Song, and Donggeon Yhee. Security guidelines for implementing homomorphic encryption. *CiC*, 1(4):26, 2024.

BCKS24. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 94–123. Springer, Cham, May 2024.

BCM+25. Jean-Philippe Bossuat, Anamaria Costache, Christian Mouchet, Lea Nürnberger, and Juan Ramón Troncoso-Pastoriza. Accurate and composable noise estimates for CKKS with application to exact HE computation. *IACR Communications in Cryptology*, 2(2), 2025.

BFS+26. Song Bian, Yunhao Fu, Ruiyu Shen, Haowen Pan, Anyu Wang, and Zhenyu Guan. High-precision functional bootstrapping for CKKS from Fourier extension. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 274–303, Cham, 2026. Springer Nature Switzerland.

BGGJ20. Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.

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

BK25. Dan Boneh and Jaehyung Kim. Homomorphic encryption for large integers from nested residue number systems. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 338–370. Springer, Cham, August 2025.

BKSS24. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 330–360. Springer, Singapore, December 2024.

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

BP23. Ahmad Al Badawi and Yuriy Polyakov. Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149, 2023.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 34–54. Springer, Cham, May 2019.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Berlin, Heidelberg, December 2016.

CHK+18. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 360–384. Springer, Cham, April / May 2018.

CHKS25. Jung Hee Cheon, Guillaume Hanrot, Jongmin Kim, and Damien Stehlé. SHIP: A shallow and highly parallelizable CKKS bootstrapping algorithm.

In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 398–428. Springer, Cham, May 2025.

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.

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, Report 2024/274, 2024.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017.

CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 670–699. Springer, Cham, December 2021.

CPL26. Gyeongwon Cha, Dongjin Park, and Joon-Woo Lee. Improved radix-based approximate homomorphic encryption for large integers via lightweight bootstrapped digit carry. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 243–273, Cham, 2026. Springer Nature Switzerland.

DAP+26. Jules Dumezy, Andreea Alexandru, Yuriy Polyakov, Pierre-Emmanuel Clet, Olive Chakraborty, and Aymen Boudguiga. Evaluating larger lookup tables using ckks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(1):559–591, 2026.

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.

DMPS24. Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. Bleach: Cleaning errors in discrete computations over ckks: N. drucker et al. *Journal of Cryptology*, 37(1):3, 2024.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

GZ26. Mingyu Gao and Hongren Zheng. Fhe for simd arithmetic logic units with amortized o(1) bootstrapping per ciphertext. In *Advances in Cryptology – CRYPTO 2026: 46th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–20, 2026, Proceedings, Part II*, page 299–330, Berlin, Heidelberg, 2026. Springer-Verlag.

HJ26. Jianhua Hou and Haodong Jiang. Careful with the ring: Enhanced hybrid decoding attacks against module/ring-LWE. Cryptology ePrint Archive, Paper 2026/366, 2026.

HK20. Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 364–390. Springer, Cham, February 2020.

jLHH+21. Wen jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE Computer Society Press, May 2021.

Kim25a. Jaehyung Kim. Efficient homomorphic integer computer from ckks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):873–898, 2025.

Kim25b. Jaehyung Kim. Faster homomorphic integer computer. Cryptology ePrint Archive, Report 2025/1440, 2025.

KN25. Jaehyung Kim and Taeyeong Noh. Modular reduction in CKKS. *CiC*, 2(2):17, 2025.

KPK+22. Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. EvalRound algorithm in CKKS bootstrapping. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 161–187. Springer, Cham, December 2022.

KS19. Duhyeong Kim and Yongsoo Song. Approximate homomorphic encryption over the conjugate-invariant ring. In Kwangsu Lee, editor, *Information Security and Cryptology – ICISC 2018*, pages 85–102, Cham, 2019. Springer International Publishing.

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.

LM21. Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Cham, October 2021.

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.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Berlin, Heidelberg, May 2013.

LW23. Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{\mathrm{O}}(1)$ polynomial multiplications. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 101–132. Springer, Singapore, December 2023.

LW24. Zeyu Liu and Yunhao Wang. Relaxed functional bootstrapping: A new perspective on BGV/BFV bootstrapping. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 208–240. Springer, Singapore, December 2024.

Ogi26. Tabitha Ogilvie. On the concrete hardness gap between MLWE and LWE. Cryptology ePrint Archive, Paper 2026/279, 2026.

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

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

Rov26. Lorenzo Rovida. A flexible and polynomial framework for integer arithmetic in CKKS. Cryptology ePrint Archive, Paper 2026/450, 2026.

A New HE-friendly expression for AKP order 1

As argued in [AKP25], the Fourier series expression is not convenient for FHE evaluation as it requires evaluating $\cos(2\pi kx)$ and $\sin(2\pi kx)$ for $k \in \{1, 2, \ldots, p-1\}$. More specifically, the equation (3) of [AKP25] for a function f is:

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

and the coefficients are given by:

$$a_0 = \frac{1}{p}\sum_{l=0}^{p-1} f(l), \qquad a_k = \frac{2(p-k)}{p^2}\sum_{l=0}^{p-1} f(l)\cos\left(\frac{2\pi lk}{p}\right),$$
$$b_k = \frac{2(p-k)}{p^2}\sum_{l=0}^{p-1} f(l)\sin\left(\frac{2\pi lk}{p}\right).$$

Note that we changed the convention here so that $R(m/p)$ evaluates to $f(m)$ instead of $f(m)/p$ as in the main body of the paper, for a simpler visualization of the example at the end of the section.

Therefore, we propose a different approach. Using the Chebyshev polynomials of the first kind $T_k(\cos 2\pi x) = \cos(2\pi kx)$ and the second kind $U_{k-1}(\cos 2\pi x) = \frac{\sin(2\pi kx)}{\sin(2\pi x)}$, we can rewrite $R(x)$:

$$R(x) = \underbrace{\left[a_0 + \sum_{k=1}^{p-1} a_k T_k(z)\right]}_{P(z)} + \sin(2\pi x) \cdot \underbrace{\left[\sum_{k=1}^{p-1} b_k U_{k-1}(z)\right]}_{Q(z)},$$

where $z := \cos(2\pi x)$, $P(z) = \sum \alpha_j T_j(z)$ and $Q(z) = \sum \beta_j T_j(z)$.

Then, by using recurrence relations between $T_n(x)$ and $U_n(x)$, we can rewrite the right half in terms of $T_n(x)$. The coefficients for the transformed polynomials

become

$$\begin{aligned} R(x) &= \sum_{j=0}^{p-1} \alpha_j T_j(\cos(2\pi x)) + \sin(2\pi x) \cdot \sum_{j=0}^{p-2} \beta_j T_j(\cos(2\pi x)) \\ &= P(z) + \sin(2\pi x) Q(z), \\ \alpha_0 &= a_0, \quad \alpha_j = a_j, \quad \text{for } 1 \le j \le p-1 \\ \beta_j &= (2 - \delta_{j,0}) \sum_{\substack{k=j+1 \\ k-j-1 \text{ even}}}^{p-1} b_k, \quad \text{for } 0 \le j \le p-2. \end{aligned}$$

The homomorphic evaluation process is as follows: 1) compute $\sin(2\pi x)$ and $\cos(2\pi x)$; 2) use $\cos(2\pi x)$ to evaluate the Chebyshev polynomials $P(z)$ and $Q(z)$ through Paterson-Stockmeyer; 3) multiply $Q(z)$ using $\sin(2\pi x)$ to obtain $R(x)$.

Numerical instability. However, while the Chebyshev polynomial form seems to be friendly for homomorphic evaluation of the variable, this formulation turns out to be numerically unstable at certain points.

Consider the following example, with $p = 4$, $n = 1$, $f(x) = x, \forall x \in \mathbb{Z}_4$. We have

$$\begin{aligned} P(z) &= 1.5T_0(z) - 0.75T_1(z) - 0.5T_2(z) - 0.25T_3(z) = 2 - z^2 - z^3 \\ Q(z) &= -0.5T_0(z) + 0.5T_2(z) = -1 + z^2 \\ R(x) &= P(z) + \sin(2\pi x) Q(z). \end{aligned}$$

At $x = 0$, we have $\cos(2\pi x) = 1 = z$, $\sin(2\pi x) = 0$ and $Q'(1) = 2$. Letting the Chebyshev approximation noise for $\cos(2\pi x)$ and $\sin(2\pi x)$ be e_1 and e_2, respectively, we have

$$\begin{aligned} R(0) &= 2 - (1+e_1)^2 - (1+e_1)^3 + e_2(-1 + (1+e_1)^2) && \text{(A1)} \\ &= -5e_1 + O(e_1^2) + 2e_1e_2 + e_1^2e_2 && \text{(A2)} \end{aligned}$$

Therefore, the noise e_1 is amplified by 5 times for these small parameters. We emphasize that e_1 comes from the Chebyshev approximation, and is independent of the input noise.

Note that this happens because the independent evaluation of $\cos(2\pi x)$ and $\sin(2\pi x)$ introduces different noises which do not cancel each other.

In conclusion, while the Chebyshev formulation of AKP does not suffer from imaginary noises as all evaluation happens in real domain (ignoring the CKKS artifacts), its independent evaluation of both cos and sin leads to numerical instability. We leave further investigation of this for future work.

B Details on refining BKSS

Bae et al. [BKSS24] proposed to use the relation $z^{p-k} = \overline{z}^k$ which holds at p-th roots of unity, and constructed a bivariate noise cleaning polynomial $h(z, \overline{z})$ where

$z := \exp(2\pi i x), \forall x \in [0,1)$. This function is generally non-holomorphic and only allows partial derivatives. More formally, let $f : \mathbb{Z}_p \to \mathbb{C}$ be a complex-valued LUT, the constraints for first-order interpolation are

$$h(\omega^m, \overline{\omega}^m) = \frac{f(m)}{p}, \qquad \left.\frac{\partial h}{\partial z}\right|_{z=\omega^m} = 0, \qquad \left.\frac{\partial h}{\partial \overline{z}}\right|_{z=\omega^m} = 0, \qquad \forall m \in [p].$$

To obtain a solution and make the solution friendly for homomorphic evaluation, [BKSS24] pick special "trinomial" basis

$$B_k(z) = \begin{cases} c_{0,p-k}\overline{z}^{p-k} + c_{k,0}z^k + c_{k+1,1}z^{k+1}\overline{z} & \text{when} \quad 1 \le k \le \lfloor p/2 \rfloor \\ c_{0,p-k}\overline{z}^{p-k} + c_{k,0}z^k + c_{1,p-k+1}z\overline{z}^{p-k+1} & \text{when} \quad \lfloor p/2 \rfloor < k \le p-1 \end{cases}$$

such that inside one $B_k(z)$, each monomial $z^i\overline{z}^j$ satisfies $i - j = k \pmod p$. This relation leads to an "effective polynomial" $B_k(\omega^\ell) = e_{k,\ell} \cdot \omega^{k\ell}, \forall k, \ell \in [p]$. Then, the LUT constraints can be achieved directly using the Lagrange interpolation $L(x)$ with $\boldsymbol{L} = \mathsf{IDFT}(\boldsymbol{f}/p)$.

This approach achieves quadratic noise suppression while keeping the polynomial degree being $\max\{p-1, 4\}$.

B.1 Complexity analysis for BKSS

This section focuses on re-evaluating the level consumption and HMult consumption of BKSS as outlined in (14) for different evaluation strategies.

Technically, [BKSS24] needs to evaluate four polynomials P_f, P_{f^r}, Q_f and Q_{f^r}. The former two polynomials have degree $p-1$ and Q_f has degree $p/2$. Q_{f^r} has degree $p/2-1$ thanks to our correction.

The original BSGS method. To efficiently evaluate four polynomials, the authors propose to choose $\mathbf{bs} = (u, u^2, \ldots, u^{2\sqrt{d}})$, and the overall cost becomes $\approx 7\sqrt{p}/2 + \frac{1}{2}\log p$, as $2^l \approx 2\sqrt{p}$, and $2(2^{m-l}-1) + 2(2^{m-l-1}-1) \approx 3\sqrt{p}/2$. The evaluation of P_f and P_{f^r} takes $\log p + 1$ levels. We do not employ [BMTPH21] as we have four polynomials to evaluate and that will incur $4\log p$ additional HMult. The evaluation of Q_f will consume $\log p$ no matter how we recurse, and further multiplication by $z\overline{z}$ will make the level consumption become $\log p + 1$. Therefore, the total level consumption of BKSS is $\log p + 1$ for the new expression.

When the LUT is real-valued, BKSS requires the evaluation of a polynomial with degree $p-1$ and $p/2-1$. By using the same baby-step basis as above, the overall cost becomes $(2+3/4)\sqrt{p} = 11\sqrt{p}/4$.

Attempt for Paterson-Stockmeyer. We can evaluate $h(z, \overline{z})$ using Paterson-Stockmeyer, and reuse the baby and giant steps (monomials) across multiple polynomials. By choosing $k = 2\sqrt{p}$, the cost is $k - 1 + 2(m-1) + 4(2^{m-1}-1) = 3\sqrt{p} + \log(p/4) + O(1)$ HMult.

However, if we reuse the Paterson-Stockmeyer steps for all four polynomials, the evaluation of Q_f and Q_{f^r} will take $\log(p) + 2$ levels. For example, when $p = 2^8$, we have $k = 18$ and $m = 4$ and evaluating P takes 9 levels. For Q, however, we have $k(2^{m-1}-1) = 126$ which is less than $p/2$. Therefore evaluating

Q also takes $\log(p)+1$ levels. After multiplication of the extra $z\overline{z}$, the level consumption is $\log(p)+2$.

Another issue is numerical instability. When we were experimenting AES S-box using BKSS, the long division step makes the coefficients of the resulting polynomial large, and the noise will blow up.

In summary, Paterson-Stockmeyer for BKSS does not offer advantages because of extra level consumption (without significant improvement for the number of HMults) and its numerical instability for highly discontinuous functions.

B.2 Equivalence of BKSS and AKP for real LUT on the real line

Using the Lagrange interpolation coefficients, the closed-form solution for $T(z)$ for AKP is as follows:

$$T_0 = \frac{L_0}{2}, \qquad T_k = \frac{p-k}{p} L_k.$$

For the BKSS $h(z,\overline{z})$, we can rewrite it as

$$\begin{aligned}
h(z,\overline{z}) &= L_0 + \sum_{k=1}^{p/2} \frac{L_k}{p}\left(k\overline{z}^{p-k} + (p-k)(k+1)z^k - k(p-k)z^k|z|^2\right) \\
&\quad + \sum_{k=p/2+1}^{p-1} \frac{L_k}{p}\left((p-k)z^k + k(p-k+1)\overline{z}^{p-k} - k(p-k)\overline{z}^{p-k}|z|^2\right) \\
&= L_0 + \sum_{k=1}^{p/2}\left[\frac{L_k}{p}\left(k\overline{z}^{p-k} + (p-k)z^k\right) + \frac{L_k}{p}k(p-k)z^k(1-|z|^2)\right] \\
&\quad + \sum_{k=p/2+1}^{p-1}\left[\frac{L_k}{p}\left((p-k)z^k + k\overline{z}^{p-k}\right) + \frac{L_k}{p}k(p-k)\overline{z}^{p-k}(1-|z|^2)\right] \\
&= 2\cdot\frac{L_0}{2} + \sum_{k=1}^{p-1}\frac{p-k}{p}\left(L_k z^k + \overline{L_k}\overline{z}^k\right) \\
&\quad + (1-|z|^2)\left[\sum_{k=1}^{p/2}\frac{k(p-k)}{p}L_k z^k + \sum_{k=1}^{p/2-1}\frac{k(p-k)}{p}\overline{L_k}\overline{z}^k\right].
\end{aligned}$$

and we have $2\cdot\frac{L_0}{2} + \sum_{k=1}^{p-1}\frac{p-k}{p}\left(L_k z^k + \overline{L_k}\overline{z}^k\right) = 2\Re[T(z)]$.

C Proof of Theorem 6: higher order Sparse-THI

Let $z := \exp(2\pi i x)$. Note that $P'(x) = 0$ is equivalent to $P'(z) = 0$ as $\frac{\mathrm{d}z}{\mathrm{d}x} = 2\pi i z$ is non-zero for z in unit circle. Similar conclusion holds for higher-order derivatives. The m-th derivative of $P(z)$ is $P^{(m)}(z) = \sum_{k=m}^{d} k^{\underline{m}} P_k z^{k-m}$, where $k^{\underline{m}} = k(k-1)\cdots(k-m+1)$ is the falling factorial.

For each frequency $k \in \{1, \ldots, p/2\}$, we have $n+1$ coefficients: P_k, P_{p+k} until P_{np+k}, as there are n derivative constraints. If we parameterize them as $P_{\ell p+k} = \gamma_{\ell,k} \cdot e_k$. The effective polynomial equivalence with $E(x)$ gives

$$\sum_{\ell=0}^{n} \gamma_{\ell,k} = 1.$$

Each m-th derivative constraint yields

$$\sum_{\ell=0}^{n} (\ell p + k)^{\underline{m}} \cdot \gamma_{\ell,k} = 0 \quad \text{for } m = 1, \ldots, n.$$

Therefore, we can solve $\gamma_{\ell,k}$ in matrix form.

Proposition A1 (Matrix Form of the γ System). *For order n, the γ coefficients satisfy the linear system $\boldsymbol{\gamma}^T A = \boldsymbol{b}^T$ where*

$$A = \begin{bmatrix} 1 & k^{\underline{1}} & \cdots & k^{\underline{n}} \\ 1 & (p+k)^{\underline{1}} & \cdots & (p+k)^{\underline{n}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & (np+k)^{\underline{1}} & \cdots & (np+k)^{\underline{n}} \end{bmatrix}, \quad \boldsymbol{\gamma} = \begin{pmatrix} \gamma_{0,k} \\ \gamma_{1,k} \\ \vdots \\ \gamma_{n,k} \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

The solution is

$$\gamma_{\ell,k} = (-1)^{\ell} \binom{n}{\ell} \frac{\prod_{j=0, j\neq\ell}^{n} (jp+k)}{n! p^n}.$$

Proof of Proposition A1. Note that A is a Vandermonde matrix with falling factorial. $\boldsymbol{\gamma}^T$ is the first row of its inverse, i.e., $\gamma_{\ell,k} = A^{-1}_{0,\ell}$. Now, we consider a regular Vandermonde matrix B where falling factorial is replaced by powers. The first row of the inverse of B is

$$B^{-1}_{0,\ell} = \frac{\prod_{j=0, j\neq\ell}^{n} (-(jp+k))}{\prod_{j=0, j\neq\ell}^{n} (\ell - j)p} = (-1)^{\ell} \frac{\prod_{j=0, j\neq\ell}^{n} (jp+k)}{p^n \ell!(n-\ell)!}.$$

The falling factorial satisfies $x^{\underline{m}} = \sum_{k=0}^{m} s(m,k) x^k$ where $s(m,k)$ are Stirling numbers of the first kind. So we have $A = BC$ where C is an upper-triangular matrix with entries $C_{k,m} = s(m,k)$ and $C_{0,0} = 1$. Therefore, the first row of C^{-1} is $(1, 0, \ldots, 0)$. Since $A^{-1} = C^{-1}B^{-1}$, we have $(A^{-1})_{0,\ell} = (B^{-1})_{0,\ell}$.

This completes the proof of Theorem 6. □

D Expression for higher order Sparse-THI polynomial

In the proof of Theorem 7, we need the $(n+1)$-th derivative at point x/p for $x \in \mathbb{Z}_p$:

$$\begin{aligned}
P^{(n+1)}(x/p) &= (2\pi i)^{n+1} \sum_{k=1}^{p/2} \left(\sum_{\ell=0}^{n} (\ell p + k)^{n+1} P_{\ell p+k} \right) \omega_p^{kx} \\
&= (2\pi i)^{n+1} \sum_{k=1}^{p/2} \left(e_k \sum_{\ell=0}^{n} (\ell p + k)^{n+1} (-1)^{\ell} \binom{n}{\ell} \frac{\prod_{j=0, j\neq\ell}^{n} (jp+k)}{n! \cdot p^n} \right) \omega_p^{kx} \\
&= (2\pi i)^{n+1} \sum_{k=1}^{p/2} \left(e_k \frac{\sum_{\ell=0}^{n} (\ell p + k)^{n} (-1)^{\ell} \binom{n}{\ell}}{n! \cdot p^n} \prod_{j=0}^{n} (jp+k) \right) \omega_p^{kx} \\
&= (-1)^n (2\pi i)^{n+1} p^{n+1} \sum_{k=1}^{p/2} \left(e_k \omega_p^{kx} \prod_{j=0}^{n} \left(j + \frac{k}{p} \right) \right) \\
&= (-1)^n (2\pi p i)^{n+1} \sum_{k=1}^{p/2} \left(e_k \omega_p^{kx} \frac{\Gamma(n+1+k/p)}{\Gamma(k/p)} \right).
\end{aligned}$$

Here, we used the finite-difference identity $\sum_{\ell=0}^{n} (-1)^{\ell} \binom{n}{\ell} (\ell p + k)^n = (-1)^n n! p^n$.

E Full trigonometric Hermite interpolation: reduced complex noise for complex-valued LUT

We extend the Hermite interpolation in [CKKL24] to arbitrary order with closed-form solution, and we call it Full-THI. The derivation is similar to Sparse-THI.

The constraints for order-n interpolation are

$$\begin{aligned}
& H(x) = H_0 + \sum_{k=1}^{d} H_k \exp(2\pi i k x) \qquad \forall x \in [0,1) \\
\text{s.t.} \quad & H\left(\frac{m}{p}\right) = \frac{f(m)}{p} \qquad \text{for } m \in \mathbb{Z}_p \\
& H^{(\ell)}\left(\frac{m}{p}\right) = 0 \qquad \text{for } m \in \mathbb{Z}_p, \ell \in \{1, 2, \ldots, n\}.
\end{aligned}$$

Theorem A2. *Using $\boldsymbol{L} = \mathit{IDFT}(\boldsymbol{f}/p)$, the interpolation order n* Full-THI *method has a solution for $d = (n+1)p - 1$ and the coefficients are*

$$H_{\ell p+k} = (-1)^{\ell} \binom{n}{\ell} \cdot \frac{\prod_{j=0, j\neq\ell}^{n} (jp+k)}{n! \cdot p^n} \cdot L_k, \qquad \forall k \in [p], \ell \in \{0, 1, \ldots, n\}.$$

Proof. As the LUT constraints are full, the effective polynomial is the Lagrange interpolated polynomial. Then we use the argument for derivative constraints in

the proof of Theorem 6 to derive all higher order terms. When $k = 0$, we have $H_0 = L_0$ and all other $H_{\ell p} = 0$. □

We illustrate the Full-THI interpolation in Figure A1 (again, without the extra division by p for simplicity).

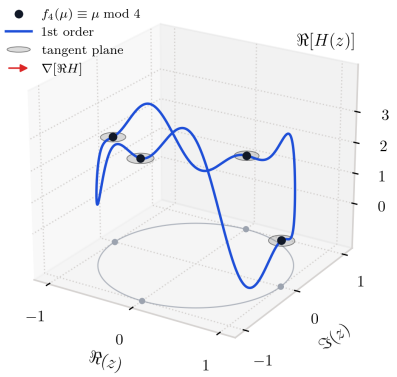


Fig. A1. Plot of Full-THI interpolation for $f_4(\mu) = \mu \mod 4$, where $\mu \in \mathbb{Z}_4$. The explanation is similar to the one in Figure 3.

E.1 Noise cleaning for Full-THI

Theorem A3. *Given a LUT function $f : \mathbb{Z}_p \to \mathbb{C}$ and corresponding order-n* Full-THI *interpolation $H(\cdot)$, the LUT-specific input threshold is $\mathcal{T}_{f,n}^{\text{Full-THI}}$, satisfying*

$$\log \mathcal{T}_{f,n}^{\text{Full-THI}} = -\left(1 + \frac{1}{n}\right) \log(2\pi p) - \frac{\log M_{f,n}}{n}$$
$$M_{f,n} = \max_{x \in \mathbb{Z}_p} \left| \sum_{k=1}^{p-1} L_k \omega_p^{kx} \frac{\Gamma(n+1+k/p)}{\Gamma(n+2)\Gamma(k/p)} \right|,$$

where Γ is the Gamma-function. Then, for input message $y = \frac{x}{p} + v$ with $x \in \mathbb{Z}_p$, we assume that $|v| \ll \frac{1}{2p}$ is smaller than $\mathcal{T}_{f,n}^{\text{Full-THI}}$ and that the tailing term $O(v^{n+2})$ from the Taylor expansion of $H(y)$ around $H(x/p)$ is negligible compared with $H^{(n+1)}(x/p)\frac{v^{n+1}}{(n+1)!}$. By using the notation $H(y) = \frac{f(x)}{p} + v'$, it holds that

$$\log |v'| - \log |v| \leq n(\log |v| - \log \mathcal{T}_{f,n}^{\text{Full-THI}}).$$

The proof follows the same structure as that of Theorem 7, except Full-THI does not have the extra 2 in the $2\Re[\cdot]$ structure.

E.2 Noise capacity for Full-THI for real-valued LUTs

Theoretical analysis shows worse performance for Full-THI compared to Sparse-THI. The results for first order (called CKKL) were shown in Table 5, below in Table A1 we show results for second order.

Table A1. Noise capacity $\mathcal{C}$ (in bits) for Full-THI of order 2 for real-valued LUTs. We use $\log \Delta = 59$ and full packing. For $p = 16$, we use ring dimension $N = 2^{16}$; for $p = 256$, we use ring dimension $N = 2^{17}$. The bound $\mathcal{B}$ is measured experimentally.

LUT Size p	**Threshold** $\log \mathcal{T}$	**Bound** $\log \mathcal{B}$	**Capacity** $\log \mathcal{C}$
2^4	-9.36	-32.22	14.32
2^8	-15.97	-29.38	8.02

F Additional experimental results for real-valued LUTs

We plot the noise phase diagrams for the Full-THI, BKSS, AKP and Sparse-THI methods for $p = 2^8$ in Figure A2. Then, in Table A2, we show experimental results for p^{10} for all methods excluding Full-THI.

Table A2. Additional experimental results of discrete-CKKS functional bootstrapping for real-valued LUT. We use scaling factor of $\Delta = 2^{59}$. For $p = 2^{10}$, we use ring dimension of $N = 2^{17}$. We use full packing. The number of KeySwitch for LUT accounts for evaluation of the LUT for *one* ciphertext, while the LUT latency column counts the evaluation of EvalLUT for two ciphertexts, due to full packing.

p	n	**Method**	**# KeySwitch**	**# Levels**	**EvalLUT (s)**	**Total (s)**
2^{10}	1	AKP	57	11	40.6	79.8
		BKSS	115	11	105.7	147.7
		Sparse-THI	61	11	42.2	85.1
	2	AKP	66	11	62.2	108.6
		Sparse-THI	70	12	64.2	107.8

G Complex-valued LUT

Here, we summarize our observations for the complex-valued LUT evaluations.

G.1 Evaluating complex-valued LUT using Sparse-THI

We now focus on complex-valued LUT $f : \mathbb{Z}_p \to \mathbb{C}$. One example of such a LUT is the symbol translation in [CPL26] that maps integers in $[0, 2B-1)$ to $\{0, \frac{1}{2}, i\}$.

A complex-valued LUT $f : \mathbb{Z}_p \to \mathbb{C}$ is equivalent to two real-valued LUTs f_1 and f_2 with $f = f_1 + if_2$. To move the evaluation result of f_2 into the imaginary part, we can multiply the ciphertext by a monomial $X^{N/2}$.

Number of Homomorphic Multiplications. The evaluation of two real-valued LUTs can reuse the preparation step of Paterson-Stockmeyer. Therefore, the total number of HMult is

$$(k-1) + 2(m-1) + 2(n+1)(2^{m-1}-1) + (\log p - \lfloor \log k \rfloor) + (3n-1)$$

By choosing $k \approx \sqrt{(n+1)p/2}$ as above, the number of HMult is

$$\sqrt{2(n+1)p} + \frac{3}{2}\log p - \frac{3}{2}\log(2(n+1)) + O(n).$$

Depth. The level consumption is the same as for evaluating a real-valued LUT, as the multiplication by $X^{N/2}$ does not consume level.

Input noise threshold. We discuss the calculation of input noise threshold under specific scenarios. When evaluating a complex-valued LUT using Sparse-THI, the resulting noise becomes $v' + iv''$, which is bounded by $\sqrt{2}\max\{|v'|, |v''|\}$, and the bound for $|v'|$ and $|v''|$ can be derived as in Theorem 7. Similarly, if we evaluate a real-valued LUT under full packing, the noise exhibits similar amplification-by-$\sqrt{2}$ behavior due to the combination of two ciphertexts into one ciphertext before SlotsToCoeffs.

G.2 Tables for complex-valued LUT

The theoretical complexity and latencies are reported in Table A3 and Table A4, respectively. Note that here we use Full-THI, whose first order is CKKL.

Compared with AKP and BKSS, Sparse-THI shows similar performance characteristics as for a real-valued LUT. Compared with Full-THI, Sparse-THI shows a larger number of key switching operations. Concretely, Sparse-THI shows a performance advantage compared to Full-THI when the LUT size p is large and the interpolation order n is 1. Otherwise, Sparse-THI has a slightly worse performance than Full-THI.

Table A3. Comparison of complexities for different interpolation approaches for complex-valued LUT.

Method	**Order** n	**# HMult**	**# Levels**
AKP	1	$2\sqrt{p} + \log(p)$	$\log(p) + 1$
BKSS	1	$\frac{7}{2}\sqrt{p} + \log(p)/2$	$\log(p) + 1$
Full-THI	1	$2\sqrt{p} + \log(p)$	$\log(p) + 2$
Sparse-THI	1	$2\sqrt{p} + \frac{3}{2}\log(p)$	$\log(p) + 1$

Table A4. Experimental results of discrete-CKKS functional bootstrapping for complex-valued LUT. We use scaling factor of $\Delta = 2^{59}$. For $p = 2^4$, we use ring dimension of $N = 2^{16}$. For $p = 2^8$ and $p = 2^{10}$, we use ring dimension of $N = 2^{17}$. We use sparse packing with the number of slots being 2^{14}.

p	n	Method	# KeySwitch	# Levels	EvalLUT(s)[†]	Total (s)
2^4	1	AKP	11	5	0.441	8.59
		BKSS	14	5	0.571	8.46
		Full-THI	11	6	0.503	8.05
		Sparse-THI	15	5	0.594	8.36
	2	AKP	14	5	0.642	8.75
		Full-THI	13	6	0.669	9.06
		Sparse-THI	20	6	0.954	9.37
	3	AKP	16	6	0.795	9.21
		Full-THI	16	7	0.944	10.2
		Sparse-THI[⋆]	25	7	1.39	10.3
2^8	1	AKP	39	9	9.69	37.8
		BKSS	58	9	14.1	41.1
		Full-THI	39	10	11.6	43.3
		Sparse-THI	42	9	10.1	38.3
	2	AKP	47	9	13.5	41.9
		Full-THI	47	10	15.7	46.1
		Sparse-THI	51	10	16.2	46.4
	3	AKP	56	10	19.9	51.6
		Full-THI	56	11	20.8	54.2
		Sparse-THI	60	11	23.1	56.7
2^{10}	1	AKP	73	11	35.3	66.4
		BKSS	115	11	53.4	85.5
		Full-THI	73	12	40.3	75.4
		Sparse-THI	77	11	35.8	67.3
	2	AKP	89	11	50.6	82.3
		Full-THI	89	12	58.1	93.7
		Sparse-THI	94	12	57.4	89.6

[†] We only record the latency of EvalLUT for *one* ciphertext due to sparse packing.
[⋆] This row does not satisfy 128-bit security and is only for illustration purposes.

H Main methods of homomorphic polynomial evaluation

In this section, we summarize the main two flavors of polynomial evaluation methods in the literature: the Paterson-Stockmeyer based methods [PS73] and the Baby-Step Giant-Step based methods [HK20].

We will assume that each ciphertext-ciphertext and ciphertext-plaintext multiplication consumes one level, as the coefficients are often not integers.

H.1 Paterson-Stockmeyer algorithm

Paterson and Stockmeyer observed that for a degree d polynomial $f(x) = \sum_{j=0}^{d} c_j x^j$, by selecting k and m such that $k(2^m - 1) > d$, we can evaluate the polynomial in a divide-and-conquer manner, with an optimal number of non-scalar multiplications.

$$\begin{aligned} f(x) &= \tilde{f}(x) - x^{k(2^m-1)} \\ &= \left(x^{2^{m-1}k} q(x) + \tilde{r}(x) + x^{k(2^{m-1}-1)}\right) - x^{k(2^m-1)} \\ &= \left(x^{2^{m-1}k} q(x) + c(x)q(x) + s(x) + x^{k(2^{m-1}-1)}\right) - x^{k(2^m-1)} \\ &= \left[\left(x^{2^{m-1}k} + c(x)\right) q(x) + \tilde{s}(x)\right] - x^{k(2^m-1)}. \end{aligned}$$

Note that $q(x)$ and $\tilde{s}(x)$ are monic with $\deg q(x) = \deg \tilde{s}(x) = k(2^{m-1} - 1)$. We also have $\deg c(x) < k$ as $\deg \tilde{r}(x) < 2^{m-1}k$. Therefore, $c(x)$ can be evaluated using **bs**, while $q(x)$ and $\tilde{s}(x)$ can be recursively evaluated.

Algorithm 1 Paterson-Stockmeyer algorithm

Input: $(c_0, c_1, \ldots, c_d)$, u
Output: $f(u) = \sum_{j=0}^{d} c_j u^j$
Preparation:
Find positive integers k, m such that $k \approx \sqrt{d/2}$ and $k(2^m - 1) > d$
Prepare **bs** $= (u, u^2, u^3, \ldots, u^k)$ ▷ $(k-1)$ HMult
Prepare **gs** $= (u^k, u^{2k}, u^{4k}, \ldots, u^{2^{m-1}k})$ ▷ $(m-1)$ HMult
Prepare $u^{k(2^m-1)}$ by multiplying $u^k \cdot u^{2k} \cdot u^{4k} \cdots u^{2^{m-1}k}$ ▷ $(m-1)$ HMult
Evaluation:
Let $\tilde{f}(x) = f(x) + x^{k(2^m-1)}$
Using long division, write $\tilde{f}(x) = x^{2^{m-1}k} q(x) + r(x)$
Set $\tilde{r}(x) = r(x) - x^{k(2^{m-1}-1)}$
Using long division, write $\tilde{r}(x) = c(x)q(x) + s(x)$
Evaluate $c(x)$ at u using **bs**.
Set $\tilde{s}(x) = s(x) + x^{k(2^{m-1}-1)}$
Recursively evaluate $q(x)$ and $\tilde{s}(x)$ at u ▷ Two children calls
Compute $\tilde{f}(u) = (u^{2^{m-1}k} + c(u))q(u) + \tilde{s}(u)$ ▷ 1 HMult, $u^{2^{m-1}k}$ from **gs**
return $f(u) = \tilde{f}(u) - u^{k(2^m-1)}$

The preparation of baby steps $u, u^2, \cdots, u^k$ takes $k-1$ HMult and $\lceil \log(k) \rceil$ levels; The preparation of giant steps $u^k, u^{2k}, \ldots, u^{2^{m-1}k}$ and the product of them $u^{(2^m-1)k}$ takes $2(m-1)$ HMult and $(m-1)+1$ levels. The body of each recursion takes 1 HMult. The recursion depth is $m-1$ and each recursive call makes two children calls, so there are $1 + 2 + \cdots + 2^{m-2} = 2^{m-1} - 1$ HMult. In summary, Paterson-Stockmeyer takes $k + 2m + 2^{m-1} - 4$ HMult and $\lceil \log(k) \rceil + m$ levels.

In typical parameterization [CCS19], we have $k \approx \sqrt{d/2}$, therefore $2^m \approx \sqrt{2d}$ and $2^{m-1} \approx \sqrt{2d}/2$, and $m \approx \frac{1}{2}\log d$. Therefore the number of HMult is $\sqrt{2d} + \log(d) + O(1)$ HMult.

For the case of discrete CKKS, the degree d typically has special shape, and level consumption can be explicitly calculated.

Proposition A4. *For power-of-two p, when evaluating a degree-$(p-1)$ polynomial using non-trivial Paterson-Stockmeyer, the minimum number of levels consumed is $\log p + 1$; when the degree is $(p/2)$, the minimum number of levels consumed is $\log p$.*

Proof. For $d = p-1$, the parameter selection ensures $p < k(2^m - 1) < 2p$ (cannot equal p as $2^m - 1$ is odd and is not 1). Therefore the preparation of $u^{(2^m-1)k}$ consumes $\log(p) + 1$ level.

For $d = p/2$, we can similarly select $p/2 < k(2^m - 1) < p$. □

H.2 Baby-Step Giant-Step

For non-power-of-two d, Han and Ki [HK20] directly chose $2^m > d$, and used the Baby-Step Giant-Step algorithm shown below.

Algorithm 2 Baby-Step Giant-Step in [HK20]

Input: $(c_0, c_1, \ldots, c_d)$, u
Output: $f(u) = \sum_{j=0}^{d} c_j u^j$
Preparation:
Let $m = \lceil \log(d+1) \rceil$ and $l = \lfloor m/2 \rfloor$.
Prepare **bs** $= (u, u^2, u^3, \ldots, u^{2^l})$ ▷ $2^l - 1$ HMult
Prepare **gs** $= (u^{2^l}, u^{2^{l+1}}, u^{2^{l+2}} \ldots, u^{2^{m-1}})$ ▷ $m - l - 1$ HMult
Evaluation:
If $\deg f < 2^l$, directly evaluate $f(u)$ using **bs**.
Write $f(x) = q(x)x^{2^{m-1}} + r(x)$
Recursively evaluate $q(x)$ and $r(x)$ at u
Evaluate $f(u) = u^{2^{m-1}} q(u) + r(u)$ ▷ 1 HMult, $u^{2^{m-1}}$ from **gs**
return $f(u)$.

The total number of HMult consumed is $2^{m-l} + 2^l + m - l - 3$. The level of recursion is $m - l$, therefore the number of HMult consumed in recursion is $2^{m-l} - 1$ as the number of recursion levels is $m - l$.

When d is $p - 1$ with p being a power of two, this can be simplified to $2\sqrt{p} + \log p/2 + O(1)$.

However, as analyzed in [BMTPH21], evaluating low-degree f using **bs** consumes $l + 1$ levels as there is a constant multiplication for non-trivial l, and therefore the total level consumption becomes $m - l + l + 1 = \lceil \log(d+1) \rceil + 1 = \log p + 1$ for $d = p - 1$. Therefore they proposed an alternative recursion strategy by writing

$f(x) = a(x)+b(x)x^{2^m-2^{l-1}}$. This incurs additional cost of $\log(d+1-(2^m-2^{l-1})) \approx \log p$ HMult but consumes the optimal level of $\log p$.

For power-of-two $d = p/2$, the number of levels consumed is $\lceil \log(p/2+1) \rceil = \log p$ no matter how we recurse.

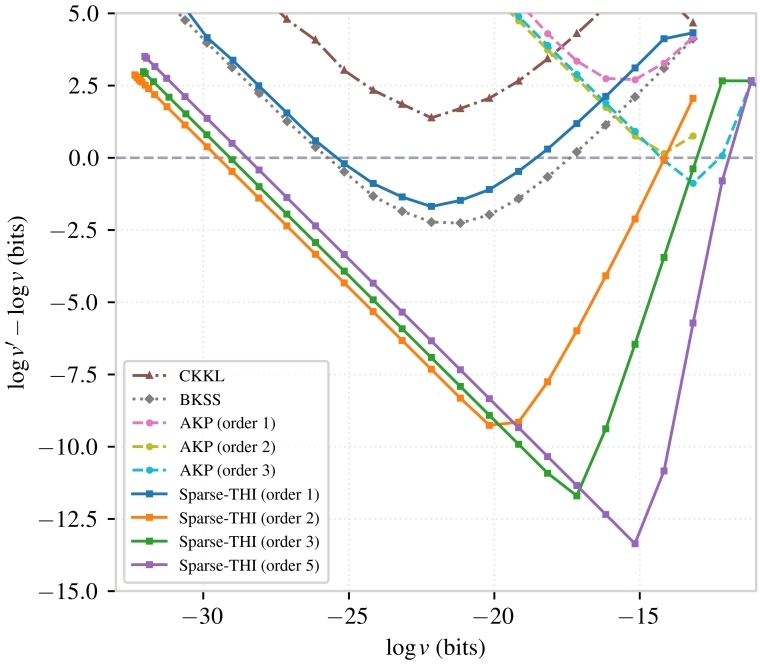


Fig. A2. Noise phase diagram obtained using experiments for Hermite interpolation methods. We use an identity LUT of size $p = 2^8$, scaling factor $\Delta = 2^{59}$, full packing, and the FIXEDMANUAL CKKS scaling technique in OpenFHE, and ring dimension $N = 2^{17}$. The x-axis is $\log v$ (input noise), and the y-axis is $\log v' - \log v$ (noise reduction), with v and v' being the input normalized noise and output normalized noise, respectively. Lower curves indicate stronger noise reduction at a given input. Curves with all positive data points (e.g. CKKL and first-order and second-order AKP) indicate the method has no noise reduction under the current parameter settings, due to large CKKS-induced noise. The read-offs from curves with all positive data points do not directly translate to the values in Table 5 as they live near the crossover region and does not represent either the threshold or the noise floor.