

# Low-Latency Parallel Digit Decomposition and Sign Evaluation for CKKS

Jung Hee Cheon   Kyungah Cho   Junyoung Jung   Junho Lim  
Seoul National University  
{jhcheon,chokya,jhaeg0312,noviajun}@snu.ac.kr

## Abstract

Nonlinear functions such as sign, comparison, and ReLU are commonly evaluated in CKKS by approximating them with polynomials. The sign jumps at zero, so inputs on the two sides of zero, however close, must be mapped to outputs that differ by a constant. Comparison is the sign of a difference, and ReLU and max inherit this requirement when computed through the sign at high output precision. To distinguish inputs separated by about  $2^{-p}$ , polynomial approximation needs  $\Theta(p)$  multiplicative depth, and we show that, in a standard model where every level must keep the values of these inputs apart above the noise, its modulus consumption grows to  $\Theta(p^2)$  bits. Evaluating the functions from radix- $B$  digits avoids this cost, but existing methods extract the  $u$  digits of a real-valued input one at a time, with  $u$  sequential bootstrappings.

We obtain all  $u$  digits in parallel with a single bootstrapping: each digit is produced with an error of at most one carry from the digit below, and carry look-ahead corrects all carries in  $O(\log u)$  depth. For the sign, the same carry look-ahead combines the digits directly, without recovering them. As a result, close inputs separated by about  $2^{-p}$  are distinguished with  $\Theta(p)$  modulus bits instead of  $\Theta(p^2)$ , using  $O(\log B)$  depth for table lookups and  $O(\log u)$  depth for carry look-ahead, at the cost of  $\Theta(u)$  slots per input. Benchmarks show a  $2.04\times$  speedup for coefficient-integer digit decomposition at ten radix-64 digits and a  $7.83\times$  speedup for sign evaluation on 1,024 real input pairs at  $2^{-60}$  separation. The gain comes from where the precision is spent: a polynomial must keep the values of close inputs apart at every level, whereas the digits take only finitely many values, so our circuit, including its bootstrapping, runs at low precision, and the output precision is raised only at the end.

**Keywords:** Fully homomorphic encryption , CKKS , Digit decomposition , Sign evaluation

## 1 Introduction

Fully homomorphic encryption (FHE) allows computation on encrypted data. The Cheon–Kim–Kim–Song (CKKS) scheme [CKKS17] supports approximate arithmetic over real or complex values. Nonlinear operations such as sign, comparison, and ReLU, however, are not natively supported and are typically evaluated through polynomial approximation. Approximating them to high precision requires resolving their behavior near discontinuities or non-smooth points, which requires a high polynomial degree and a large multiplicative depth.

Comparison illustrates this: after subtraction and range normalization, it reduces to the sign of a real value. The approximation polynomial of the sign function can be accurate only outside a narrow region around zero, and higher precision narrows the inaccurate region further. Optimized compositions of low-degree polynomials reduce the cost, but the required depth still grows as that region shrinks [CKK20, LLNK22].

Another approach is to use the finite input precision required by the application. At a chosen precision, a real value with bounded precision can be represented by an integer with  $u$  radix- $B$  digits. Functions of these digits can then be evaluated by small-domain lookups and combined to obtain the desired output, rather than approximating the nonlinear function directly over the full

real interval. This requires digit decomposition, which obtains the  $u$  digits from the input. Digit decomposition is also used for large-integer arithmetic and modular reduction [CPL26, KN25].

Each digit can be obtained by a lookup over up to  $B^u$  values, but the interpolation degree and required precision of a lookup grow with its domain size. [AKP25] avoids this by obtaining the least significant digit with a small-domain lookup in a bootstrapping, removing it, and proceeding to the next. Since each step uses the result of the previous one, extracting  $u$  digits takes  $u$  sequential bootstrappings, the most expensive operation in CKKS, so the latency grows linearly with  $u$ .

Our method avoids the sequential extraction by preparing the information for all  $u$  digits at once, in separate slots of one ciphertext, each evaluated by a lookup over a domain whose size depends only on  $B$ , not  $u$ . From this information, we obtain every digit or a function of every digit, and the sign without recovering the digits.

## 1.1 Technical Overview

**Obtaining the digit pairs.** Write  $a_0, \dots, a_{u-1}$  for the radix- $B$  digits of the input. To process all digit positions in parallel, we obtain each digit directly from the original ciphertext without removing the lower digits. One multiplication by a fixed polynomial copies the input into  $u$  coefficients of a single ciphertext, each scaled so that a different digit is the leading one, so one bootstrapping covers every position. The difficulty is that scaling the input for a target digit also brings the lower digits into the fractional part, which interferes with the lookups.

We control this effect using discretization [KN25]. Discretization rounds the ciphertext to a fixed grid, turning the fractional part and the ciphertext error into a bounded displacement on that grid. Let  $\mathcal{E}$  denote a bound on this displacement, measured in grid steps. If we use a  $B$ -point grid, which holds only one radix- $B$  digit, the displacement adds directly to the digit, and the discretized value alone gives no way to estimate it. We therefore use a  $B^2$ -point grid, which holds two neighboring digits together. Before discretization, we add a fixed shift of  $\mathcal{E}$  so that the displacement becomes nonnegative. We write the resulting window at position  $j$ , which holds  $a_j$  and  $a_{j-1}$  up to this displacement, as

$$w_j = Bb_j + c_j.$$

We take  $B$  large enough so that the upper digit  $b_j$  differs from the true digit  $a_j$  by at most one carry, that is,  $b_j = a_j + \delta_j \bmod B$  with  $\delta_j \in \{0, 1\}$ . Thus, after discretization, each window already contains the desired digit up to a single carry bit.

Figure 1 shows how this carry appears. The value  $c_j$  is  $a_{j-1}$  plus the nonnegative displacement, minus  $B$  if this sum reaches  $B$ . If the sum stays below  $B$ ,  $(a_{j-1}, c_j)$  lies in the diagonal band. Otherwise,  $a_{j-1}$  is close to  $B$  and  $c_j$  is close to zero. Let

$$H = [B - 2\mathcal{E}, B), \quad L = [0, 2\mathcal{E}).$$

When  $B > 4\mathcal{E}$ , the diagonal band does not intersect  $H \times L$ , so

$$\delta_j = \mathbb{1}_H(a_{j-1})\mathbb{1}_L(c_j),$$

for every possible window. The carry can therefore be detected using two one-variable range lookups.

**Digit decomposition.** The lookups above use the true previous digit  $a_{j-1}$ , but the previous window gives only  $b_{j-1}$ . These two values may differ by the previous carry  $\delta_{j-1}$ . Since every carry is either 0 or 1, each digit position can be described by whether it generates a carry, propagates an incoming carry, or kills it. This gives a recurrence  $\delta_j = g_j + h_j\delta_{j-1}$ , where  $g_j$  and  $h_j$  depend only on  $w_j$  and  $w_{j-1}$ .

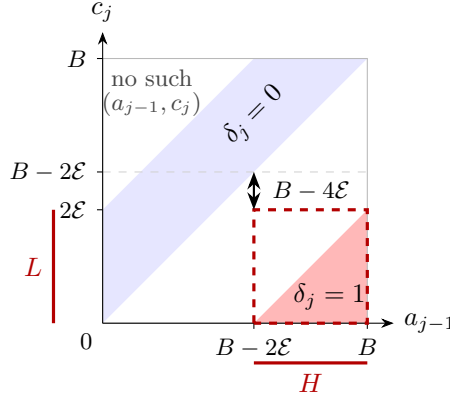


Figure 1: The two carry zones. Only the shaded pairs  $(a_{j-1}, c_j)$  occur; the dashed box is the corner triangle widened to the square, and it does not overlap the band on the diagonal when  $B > 4\epsilon$ .

Following this recurrence one digit at a time would again take  $O(u)$  sequential steps. Instead, we compute all carries by a parallel prefix scan, as in the carry stage of the Kogge–Stone adder [KS73], which takes  $O(\log u)$  rounds. Once the carries are known, each digit is recovered by correcting  $b_j$  with  $\delta_j$ . When a function  $f : \mathbb{Z}_B \rightarrow \mathbb{R}$  of each digit is wanted, we use  $f(a_j) = f(b_j) + \delta_j[f(b_j - 1) - f(b_j)]$ . Both  $f(b_j)$  and  $f(b_j - 1)$  are functions of  $w_j$ , so they are obtained by lookups in the same bootstrapping.

**Direct sign evaluation.** We also evaluate the sign without recovering the digits. Whether the input is negative is determined by a lookup on its most significant digit, so the function evaluation above at the most significant digit gives it. Whether the input is zero depends on all digits being zero, which follows a recurrence of the same form as the carry, so both are obtained by the same scan. The result is a ternary sign.

**Modulus consumption of the sign.** We compare the cost of our proposed sign evaluation with that of existing polynomial approximation methods, focusing on modulus consumption. A polynomial approximation of the sign must keep its values at the two ends of the narrow region within  $2^{-p}$  of zero apart above the noise at every level. Since each level at most doubles the polynomial degree, this difference grows by at most about one bit per level, so about  $p$  levels are needed, and the scale needed at each level starts at about  $p$  bits and drops by at most one bit per level. Summed over these levels, the modulus consumed is quadratic in  $p$ . Our sign evaluation holds the input at about  $p$  bits for only a constant number of levels. The scan then works on digits, which take finitely many values, so it runs at low precision, and the ternary output is raised to any output precision afterwards by cleaning, that is, by repeatedly applying a fixed polynomial that maps each of  $-1, 0$  and  $1$  to itself and is flat around it, at a cost linear in that precision.

## 1.2 Contributions

**Parallel digit decomposition in one bootstrapping.** The  $u$  radix- $B$  digits of an encrypted input are obtained from one bootstrapping, rather than the  $u$  that iterative methods [AKP25, KN25] spend. With its lookup tables changed, the same circuit returns any function  $f : \mathbb{Z}_B \rightarrow \mathbb{R}$  of every digit, with no digit restored first. The lookups consume  $O(\log B)$  levels and the scan  $O(\log u)$  levels. The latency thus no longer includes  $u$  sequential bootstrappings, at the cost of  $\Theta(u)$  slots per input. The coefficient-integer benchmark shows a  $2.04\times$  speedup over [AKP25].

**Sign evaluation without digit recovery.** Loaded with different tables, the same scan returns a ternary sign of the rounded input, one value per message. Outside a gap around the origin whose width Proposition 2 bounds, this value is the exact sign; inside the gap it is either the sign or 0, unlike a polynomial approximation, which can take any value in  $[-1, 1]$  there. To do so, the scan holds the intercept and slope of an affine map in one complex slot, extending the three carry states that [CPL26] place on  $i, \frac{1}{2}$  and 0 to an arbitrary real pair.

**A  $\Theta(p)$  versus  $\Theta(p^2)$  modulus separation.** To decide the sign of inputs as close as  $2^{-p}$  to zero, polynomial approximation needs depth linear in  $p$  and, under a standard noise and scale model, modulus quadratic in  $p$ , even when the output needs only a few bits, while our route uses  $\Theta(p)$  modulus bits for output precision up to about  $p$  bits, of which the digit circuit takes only  $O((\log p)^2)$ . Since the number of bootstrappings grows with the modulus consumption, our method also achieves lower latency in the higher-precision benchmarks reported in Section 5.

### 1.3 Related Work

**Polynomial methods for comparison and related nonlinear functions.** [CKK20] study efficient homomorphic comparison, and [LLNK22] optimize composite polynomial approximations to the sign function. These methods have also been applied to comparison, maximum, and ReLU in RNS-CKKS [LLKN21]. When bootstrapping is used, its numerical error can affect the following sign approximation; [HLKL26] study this interaction explicitly. These works reduce the cost of polynomial approximation, but the required computation still grows as the input gap becomes smaller.

**Discrete computation and functional bootstrapping in CKKS.** Bootstrapping methods for bits and small integers use the discrete message space of CKKS to evaluate functions over small finite domains [BCKS24, BKSS24]. Integer arithmetic has been built on this discrete setting [Kim25, Kim26, CPL26]. [AKP25] combines functional bootstrapping with multi-precision sign evaluation and digit decomposition, and also considers CKKS inputs whose slots already contain integers. [KN25] independently uses iterative digit decomposition on top of small-integer bootstrapping [BKSS24], and extends the method to real-valued inputs through discretization based on modulus switching. In their iterative multi-precision procedures, higher digits are obtained only after lower digits have been obtained and removed, making the digit extraction sequential. Our method instead prepares information for multiple digit positions at once and combines them using a parallel carry computation.

**Conversion for homomorphic arithmetic logic units.** [GZ26] introduces triangle encoding for SIMD word arithmetic and obtains amortized constant bootstrapping cost per ciphertext. [WMGZ26], [SLWL26], and [LMWW26] improve conversion for this encoding. All three are independent of and concurrent with ours. These works aim at conversion between encodings, while we aim at digit decomposition and the evaluation of digit functions. They also start from an arithmetic-encoded ciphertext, while we start from a CKKS ciphertext of a real-valued input. However, a CKKS ciphertext of a real-valued input can also be transformed into a form that their methods handle, so their methods also apply to our setting. To cancel the overflow, [SLWL26, LMWW26] take linear combinations of neighboring digit positions on the output of a functional bootstrapping, so they need two sequential functional bootstrappings. In contrast, our window  $w_j = Bb_j + c_j$  already contains two neighboring digits, so the carry lookups at all digit positions are evaluated in parallel, and one bootstrapping is enough. Finally, we also give a method that evaluates the sign without recovering the digits.

## 2 Preliminaries

### 2.1 Notations

For a positive integer  $n$ , write  $[n] = \{0, \dots, n-1\}$ . Let  $N \geq 2$  be a power of two and  $Q \geq 2$  an integer, and write  $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$  and  $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$ . We write  $\lfloor x \rfloor = \lfloor x + 1/2 \rfloor$  for rounding to the nearest integer and  $[x]_Q := x - Q\lfloor x/Q \rfloor$  for the representative of  $x \bmod Q$  in  $[-Q/2, Q/2)$ , both extended coefficient-wise to polynomials. For a polynomial  $v$  with real coefficients,  $\|v\|_\infty$  and  $\|v\|_1$  are the maximum and the sum of the absolute values of its coefficients, and  $\zeta_W := \exp(2\pi i/W)$  is the  $W$ -th root of unity. Finally,  $\mathbb{1}_C(x)$  is the indicator of membership in a set  $C$ ,  $\mathbb{1}[\mathcal{P}]$  the indicator of a predicate  $\mathcal{P}$ , and  $\text{rev}_n(k)$  the reversal of the  $\log_2 n$  bits of  $k < n$ .

### 2.2 CKKS basics

CKKS [CKKS17] is a fully homomorphic encryption scheme that encrypts a plaintext polynomial in  $\mathcal{R}$ . To encrypt a cleartext, the actual data one wishes to encrypt, one must first encode it into a plaintext in  $\mathcal{R}$ .

**CKKS Encoding.** CKKS supports two kinds of encoding, called coefficient encoding and slot encoding.

Coefficient encoding encodes  $\mathbf{z} = (z_i)_{i \in [N]} \in \mathbb{R}^N$  with scaling factor  $\Delta$  to

$$\text{Ecd}_{\text{coeff}}(\mathbf{z}; \Delta) := \sum_{i \in [N]} \lfloor \Delta \cdot z_i \rfloor X^i \in \mathcal{R}.$$

Slot encoding uses the inverse discrete Fourier transform iDFT, where

$$\text{DFT} : \mathbb{R}[X]/(X^N + 1) \rightarrow \mathbb{C}^{N/2}, \quad m(X) \mapsto \left( m\left(\zeta_{2N}^{5^i}\right) \right)_{i \in [N/2]},$$

where iDFT is its inverse. Then the slot encoding encodes  $\mathbf{z} \in \mathbb{C}^{N/2}$  with scaling factor  $\Delta$  to

$$\text{Ecd}_{\text{slot}}(\mathbf{z}; \Delta) := \lfloor \Delta \cdot \text{iDFT}(\mathbf{z}) \rfloor \in \mathcal{R}.$$

When clear from context,  $\Delta$  may be omitted. We say an encoded plaintext  $m$  has scale  $\Delta$  if  $m = \text{Ecd}_{\text{coeff}}(\mathbf{z}; \Delta)$  or  $m = \text{Ecd}_{\text{slot}}(\mathbf{z}; \Delta)$  for some  $\mathbf{z}$ . The linear transforms CtS and StC convert between these representations.

**CKKS Encryption.** A CKKS ciphertext has the form  $\text{ct} = (a, b) \in \mathcal{R}_Q^2$ , where  $Q \geq 2$  is an integer called the ciphertext modulus.  $\text{ct}$  encrypts an encoded polynomial  $m \in \mathcal{R}$  with respect to the secret key  $s \in \mathcal{R}$  if it satisfies

$$b + as = m + e \pmod{Q}, \tag{1}$$

for some small error polynomial  $e \in \mathcal{R}$ . In this case, we write  $\text{ct} = \text{Enc}_Q(m)$ . We omit the modulus when clear from context. For a ternary secret key  $s(X)$ , its coefficients lie in  $\{-1, 0, 1\}$  and its Hamming weight is  $\text{hwt}(s) = \|s\|_1$ .

**CKKS operations.** Throughout, all operations are approximate. CKKS supports the following operations.

- **Rescale:** Given  $\text{ct} = \text{Enc}_Q(m)$  where  $m$  has scale  $\Delta$ , rescaling to a smaller modulus  $Q'$  outputs  $\text{RS}_{Q \rightarrow Q'}(\text{ct}) = \text{Enc}_{Q'}(m')$  where  $m'$  has scale  $\Delta' = \frac{Q'}{Q}\Delta$ . Using rational rescale [CCK<sup>+</sup>25],  $Q'$  need not divide  $Q$ .

- **Key Switching:** Given  $\text{ct} = \text{Enc}_Q(m)$  under secret  $s$  and a switching key from  $s$  to  $s'$ , outputs  $\text{KS}_{s \rightarrow s'}(\text{ct}) = \text{Enc}_Q(m)$  under secret  $s'$ .
- **Rotation/Conjugation:** Given  $\text{ct} = \text{Enc}_Q(m)$  with  $m = \text{Ecd}_{\text{slot}}(z)$  for some  $z \in \mathbb{C}^{N/2}$ , outputs  $\text{Rot}_r(\text{ct}) = \text{Enc}_Q(m')$  where  $m' = \text{Ecd}_{\text{slot}}(\text{Rot}_r(z))$ ,  $(\text{Rot}_r(z))_j = z_{(j+r) \bmod N/2}$ . Also,  $\text{Conj}(\text{ct}) = \text{Enc}_Q(m'')$  where  $m'' = \text{Ecd}_{\text{slot}}(\bar{z})$ . Rotation and conjugation need 1 key switching each [CHK<sup>+</sup>18].
- **Addition:** For  $\text{ct} = \text{Enc}_Q(m)$ ,  $\text{ct}' = \text{Enc}_Q(m')$  and an encoded plaintext  $m'' \in \mathcal{R}_Q$ ,
$$\text{ct} \pm m'' = \text{Enc}_Q(m \pm m''), \quad \text{ct} \pm \text{ct}' = \text{Enc}_Q(m \pm m').$$

- **Multiplication:** For  $\text{ct} = \text{Enc}_Q(m)$  and  $\text{ct}' = \text{Enc}_Q(m')$  of scales  $\Delta_1$  and  $\Delta_2$ , and an encoded plaintext  $m'' \in \mathcal{R}_Q$  of scale  $\Delta_2$ ,  $m''\text{ct}$  and  $\text{Mult}(\text{ct}, \text{ct}')$  denote the product followed by a rescale to  $Q'$ :

$$m''\text{ct} = \text{Enc}_{Q'}(mm''), \quad \text{Mult}(\text{ct}, \text{ct}') = \text{Enc}_{Q'}(mm'),$$

both of scale  $\frac{Q'}{Q}\Delta_1\Delta_2$ .  $\text{Mult}$  needs one key switching, and  $m''\text{ct}$  needs none.

Each multiplication followed by its rescale consumes one *level* of multiplicative depth and reduces the ciphertext modulus. Once the ciphertext modulus  $Q$  is too small to rescale further, *bootstrapping* restores  $\text{ct} \in \mathcal{R}_Q^2$  to a ciphertext  $\text{ct}' \in \mathcal{R}_{Q'}^2$  for some  $Q' > Q$ , encrypting approximately the same message [CHK<sup>+</sup>18].

## 2.3 Discrete CKKS

Unlike ordinary CKKS, which approximates real- or complex-valued messages, discrete CKKS restricts messages to integer representatives of  $\mathbb{Z}_W$  for an integer  $W \geq 2$ , and supports integer and Boolean computation. Under the usual slot encoding, addition and multiplication over these integer representatives are inherited from CKKS as operations over  $\mathbb{Z} \subset \mathbb{C}$ , with an additional technique for modular reduction and bootstrapping tailored to  $\mathbb{Z}_W$  [BCKS24, BKSS24, AKP25, KN25].

**Discretization.** Discretization [KN25] rounds the components of a ciphertext so that the message it encrypts can be read as an element of  $\mathbb{Z}_W$ .

**Definition 1** (Discretization). For positive integers  $W, Q$ ,  $\text{Disc}_{W,Q} : \mathcal{R}_Q \rightarrow \mathcal{R}_Q$  is defined coefficient-wise by

$$\sum_{i \in [N]} c_i \cdot X^i \mapsto \sum_{i \in [N]} \left\lfloor \frac{Q}{W} \left\lfloor \frac{W}{Q} c_i \right\rfloor \right\rfloor \cdot X^i,$$

i.e., each coefficient is snapped to the nearest of the multiples of  $Q/W$ , which we call the *grid*.

For a ciphertext  $\text{ct} = (a, b) \in \mathcal{R}_Q^2$ , we define discretization of the ciphertext as

$$\text{Disc}_{W,Q}(\text{ct}) := (\text{Disc}_{W,Q}(a), \text{Disc}_{W,Q}(b)).$$

Discretizing the two components does not place the message on the grid exactly: Section 3.4 shows the decrypted value is a grid point up to an integer number of grid steps and a residue below the grid, both bounded in terms of  $\text{hwt}(s)$ .

**Ciphertext reduction and plaintext residues.** For  $\text{ct} = (a, b) \in \mathcal{R}_Q^2$  and  $Q' \mid Q$ , we write  $[\text{ct}]_{Q'} := ([a]_{Q'}, [b]_{Q'}) \in \mathcal{R}_{Q'}^2$ . Unlike rescale, restriction does not change the scale of the underlying plaintext. Equation (1) gives the exact relation

$$[[b]_{Q'} + [a]_{Q'}s]_{Q'} = [m + e]_{Q'}, \tag{2}$$

i.e.,  $[\text{ct}]_{Q'}$  encrypts  $[m]_{Q'}$  approximately.

---

**Algorithm 1** Root<sub>W</sub>

---

**Input:** A coefficient ciphertext at scale  $Q_0/W$  whose selected coordinates encode  $\mathbf{z} \in \mathbb{Z}_W^{N/2}$  in the packing above.

**Output:** Root<sub>W</sub>(ct) = Enc(Ecd<sub>slot</sub>( $\zeta_W^{\mathbf{z}}$ )), where  $\zeta_W^{\mathbf{z}} := (\zeta_W^{z_i})_{i \in [N/2]}$ .

- 1: ct  $\leftarrow$  ModRaise(ct)
  - 2: ct  $\leftarrow$  CtS(ct)
  - 3: **return** EvalExp(ct)
- 

**Functional bootstrapping.** For  $W \in \mathbb{Z}_{>0}$  and  $f: \mathbb{Z}_W \rightarrow \mathbb{C}$ , *functional bootstrapping* in CKKS [BKSS24, AKP25] takes a coefficient-encoded ciphertext at scale  $Q_0/W$  whose selected coordinates encode  $\mathbf{z} \in \mathbb{Z}_W^{N/2}$ . Here  $Q_0$  is the ciphertext modulus at which the bootstrap is entered. It returns Enc<sub>Q</sub>(Ecd<sub>slot</sub>( $f(\mathbf{z}); \Delta$ )), where  $f$  is applied componentwise. Here  $Q$  and  $\Delta$  are the modulus and scale of the bootstrapped output. The output scale  $\Delta$  need not equal the input scale  $Q_0/W$ . It is determined by the output modulus and the chosen bootstrap's precision.

Functional bootstrapping breaks down into two steps. The first, root production, lifts a coefficient-encoded ciphertext to slot-encoded roots of unity via three operations (Algorithm 1): ModRaise lifts the ciphertext modulus, introducing an integer overflow  $W\mathbf{I}$  with  $\|\mathbf{I}\|_\infty$  depending on the Hamming weight of the secret key; CtS homomorphically converts the coefficient encoding to the slot encoding; and EvalExp homomorphically evaluates  $x \mapsto \exp(2\pi ix/W)$ , so that the integer overflow  $W\mathbf{I}$  vanishes by periodicity, i.e.,  $\exp(2\pi i(\mathbf{z} + W\mathbf{I})/W) = \exp(2\pi i\mathbf{z}/W)$ .

The second, LUT evaluation, interpolates a target function over these roots. Given Root<sub>W</sub>(ct) and a public function  $f: \mathbb{Z}_W \rightarrow \mathbb{C}$ , LUT<sub>W</sub><sup>f</sup> homomorphically evaluates an interpolant of  $f$  at the nodes  $\exp(2\pi ia/W)$ ,  $a \in \mathbb{Z}_W$ , yielding Enc(Ecd<sub>slot</sub>( $f(\mathbf{z})$ )). Following [AKP25], for real-valued  $f$  we take the interpolant to be the real part of a polynomial of degree  $W - 1$  in the root, and we evaluate a complex-valued  $f$  as its real and imaginary parts, combined by multiplication by  $i$ . Its evaluation takes  $\lceil \log_2 W \rceil + O(1)$  levels. All tables evaluated on the same Root<sub>W</sub>(ct) share the powers of the root [AKP25] and differ only in their public coefficients, which are prepared before the online circuit. We refer to the composition LUT<sub>W</sub><sup>f</sup>  $\circ$  Root<sub>W</sub> as functional bootstrapping for  $f$ . When  $f = \text{id}$ , it returns the integer representatives of  $\mathbf{z}$  in the slots.

**Interpolation near the roots.** Root production returns approximate roots  $\zeta_W^a + \eta$  with a small error  $\eta$ , rather than the exact interpolation nodes  $\zeta_W^a$ . Hermite interpolation reduces the effect of  $\eta$  by prescribing derivatives as well as values [AKP25]. For example, an interpolant  $P_f$  satisfying

$$P_f(\zeta_W^a) = f(a), \quad P'_f(\zeta_W^a) = 0$$

obeys  $P_f(\zeta_W^a + \eta) = f(a) + O(|\eta|^2)$ , with a constant depending on  $P_f$ . Prescribing the derivatives up to order  $k - 1$  gives  $O(|\eta|^k)$  at a higher degree, which for a fixed  $k$  adds  $O(1)$  levels. The evaluation of  $P_f$  adds its own CKKS error. These two errors make up the error of a LUT evaluation.

**Cleaning.** When the ideal values of a computation lie in a finite set  $\mathcal{V}$ , the CKKS error of its output can be reduced by a *cleaning polynomial* Cl, which fixes each  $v \in \mathcal{V}$  with zero derivative there. As for the interpolant above,  $\text{Cl}(v + \eta) = v + O(|\eta|^2)$  locally, so each iteration roughly doubles the number of accurate bits once the error is small enough. For bits,  $\mathcal{V} = \{0, 1\}$  and BLEACH [DMPS24] uses  $3t^2 - 2t^3$ ; the precision phase of [CKK20] iterates polynomials of this kind near  $\pm 1$ .

**Iterative digit decomposition.** One base- $B$  digit at a time can be read by functional bootstrapping with  $f = \text{id}$ . Starting from a coefficient encryption of  $\mathbf{z} \in \mathbb{Z}_{B^u}^{N/2}$  at scale  $Q/B^u$ , or

of a real vector after discretization, multiplying by  $B^{u-1-j}$  places digit  $j$  in the leading position, the digits above it wrapping around modulo  $Q$ , and  $(\text{LUT}_B^{\text{id}} \circ \text{Root}_B)$  returns it in the slots. The digit  $j$  is removed before the digit  $j+1$  is read, since it would otherwise remain as a fractional tail, displacing the value from the interpolation nodes: **StC** returns the digit  $j$  to the coefficient domain, and it is subtracted. Running this for  $j = 0, \dots, u-1$  gives the  $u$  digits [AKP25, KN25], at  $u$  root productions and LUT evaluations and  $u-1$  **StC** in as many sequential stages.

### 3 Digit Decomposition

This section makes the construction of Section 1.1 precise. The input is a CKKS ciphertext whose slots pack the messages, and the output is a public real-valued function of each restored digit, in the slots of one ciphertext, obtained with one root production; the identity function returns the digits themselves. We first obtain the noisy digit pairs (Section 3.1) and describe their carry recurrence, which a parallel prefix scan computes (Section 3.2). Section 3.3 obtains all lookup values from one root of period  $B^2$ , and Section 3.4 bounds the discretization error. For ease of exposition, we assume integer input; real input uses the same circuit, which then returns the digits of an adjacent integer (Proposition 1).

#### 3.1 Obtaining the digit pairs

Each message is an integer  $z \in [0, B^u)$  with radix- $B$  digits  $z = \sum_{j \in [u]} a_j B^j$ , and we put  $a_{-1} := 0$ . As in Section 1.1, we obtain for each digit  $j$  a two-digit window  $w_j = Bb_j + c_j$  that holds  $a_j$  and  $a_{j-1}$  together with an integer error of absolute value below  $\mathcal{E}$ , whose value Section 3.4 fixes. We take  $B > 4\mathcal{E}$ , the condition Section 3.2 needs to recover the digits from these windows.

**Placing the digits.** Each message enters a single slot and is spread over a block of  $\Lambda := 2^{\lceil \log_2(u+1) \rceil}$  places, one per digit and the rest padding, whose purpose appears in Section 3.2. The messages are therefore placed  $\Lambda$  apart,  $z_m$  at slot  $m\Lambda$ , and a single ciphertext carries  $M := N/(2\Lambda)$  of them  $(z_m)_{m \in [M]}$ , assuming  $\Lambda \mid N/2$ . Before discretization, we first multiply by the fanout polynomial

$$F(X) := \sum_{j=0}^{u-1} B^{u-1-j} \cdot X^{\text{rev}_{N/2}(1+j)}.$$

This spreads the packed value  $z_m$  across  $u$  coefficients, one per digit  $j$ . We call these places positions, numbering them so that position  $j$  is slot  $m\Lambda + 1 + j$ ; the block runs from  $-1$  to  $\Lambda - 2$ , and every position outside  $0, \dots, u-1$  is padding. Multiplying by  $B^{u-1-j}$  moves the pair  $(a_j, a_{j-1})$  into the two most significant digit positions. Bit reversal is needed since **StC** is implemented using the fast Fourier transform, which bit-reverses indices over the  $N/2$  slots [HHC19]. Write  $D := \{m\Lambda + 1 + j : m \in [M], j \in [u]\}$  for the digit positions. We then add the shift polynomial

$$S(X) := \mathcal{E} \sum_{k \in D} X^{\text{rev}_{N/2}(k)} + (B + \mathcal{E}) \sum_{k \in [N/2] \setminus D} X^{\text{rev}_{N/2}(k)}.$$

This places the padding's high digit at  $b = 1$ .

**Discretization.** The discretization  $\text{Disc}_{B^2, Q_0}$  (Definition 1) then rounds every coefficient to the grid of step  $Q_0/B^2$ . Afterwards, position  $j$  holds the window  $w_j$  at scale  $Q_0/B^2$ , up to a residue below the grid step:

$$\begin{aligned} w_j &\equiv Ba_j + a_{j-1} + \mathcal{E} + \varepsilon_j \pmod{B^2}, & \varepsilon_j &\in \mathbb{Z}, |\varepsilon_j| < \mathcal{E}, \\ &=: Bb_j + c_j. \end{aligned} \tag{3}$$

---

**Algorithm 2** FormWindows
 

---

**Input:**  $\text{ct} = \text{Enc}_Q(\text{Ecd}_{\text{slot}}(\mathbf{z}')) \in \mathcal{R}_Q^2$  where  $\mathbf{z}' \in \mathbb{R}^{N/2}$  supported on  $\{m\Lambda : m \in [M]\}$ ,  $z'_{m\Lambda} = z_m$ .

**Output:** Coefficient ciphertext  $\text{ct.w}$  at scale  $Q_0/B^2$ , holding  $w_j$  from (3) for  $z_m \bmod B^u$  at coefficient  $\text{rev}_{N/2}(m\Lambda + 1 + j)$  and a value in  $(B, B + 2\mathcal{E})$  at coefficients  $\text{rev}_{N/2}(k)$  for  $k \in [N/2] \setminus D$ .

- 1:  $\text{ct} \leftarrow \text{StC}(\text{ct})$   $\triangleright$  scale  $Q'/B^u$ ,  $Q'$  the resulting modulus
  - 2:  $\text{ct} \leftarrow F(X) \cdot \text{ct} + S(X)$   $\triangleright$  reinterpret at scale  $Q'/B^2$
  - 3:  $\text{ct} \leftarrow \text{RS}_{Q' \rightarrow Q_0}(\text{ct})$
  - 4:  $\text{ct.w} \leftarrow (\text{Disc}_{B^2, Q_0} \circ \text{KS}_{s \rightarrow \tilde{s}})(\text{ct})$
  - 5: **return**  $\text{ct.w}$
- 

Writing

$$\delta_j := \left\lfloor \frac{a_{j-1} + \mathcal{E} + \varepsilon_j}{B} \right\rfloor \in \{0, 1\},$$

$$b_j = (a_j + \delta_j) \bmod B, \quad c_j = a_{j-1} + \mathcal{E} + \varepsilon_j - B\delta_j,$$

for  $j \in [u]$ , with  $a_{-1}$  as above. The shift  $\mathcal{E}$  keeps  $a_{j-1} + \mathcal{E} + \varepsilon_j \in [0, B + 2\mathcal{E}]$ , and  $B > 2\mathcal{E}$  puts it below  $2B$ , so  $\delta_j \in \{0, 1\}$  and  $c_j \in [B]$ .  $(b_j, c_j)_{j \in [u]}$  are the noisy digit pairs.

Algorithm 2 summarizes the preparation, with  $S(X)$  encoded at the indicated coefficient scale; Remark 3 of Section 3.4 explains the switch to a sparser secret  $\tilde{s}$  in line 4.

### 3.2 Carry resolution

**Determining the carry.** By (3),  $a_j \equiv b_j - \delta_j \pmod{B}$ , so recovering the digits amounts to determining every  $\delta_j$ . Assuming  $B > 4\mathcal{E}$ ,  $\delta_j$  is determined by two one-variable range tests.

**Lemma 1.** *Let  $L = [0, 2\mathcal{E}]$  and  $H = [B - 2\mathcal{E}, B]$ , and suppose  $B > 4\mathcal{E}$ . If  $a \in [0, B]$  and  $|\varepsilon| < \mathcal{E}$ , and  $a + \mathcal{E} + \varepsilon = B\delta + c$  with  $\delta \in \{0, 1\}$  and  $c \in [B]$ , then  $\delta = \mathbb{1}_H(a) \mathbb{1}_L(c)$ .*

*Proof.* Suppose  $\delta = 1$ , so  $t := a + \mathcal{E} + \varepsilon \geq B$ . Then  $a = t - (\mathcal{E} + \varepsilon) > B - 2\mathcal{E}$ , so  $a \in H$ ; and  $c = t - B < (B + 2\mathcal{E}) - B = 2\mathcal{E}$ , so  $c \in L$ . Both indicators are 1.

Suppose  $\delta = 0$ , so  $c = t$ . If  $a \notin H$  the product is 0. If  $a \in H$ , then  $c = t > a \geq B - 2\mathcal{E} > 2\mathcal{E}$ , the last step by  $B > 4\mathcal{E}$ , so  $c \notin L$  and the product is 0 again.  $\square$

Applied with  $a = a_{j-1}$ , this gives  $\delta_j = \mathbb{1}_H(a_{j-1}) \mathbb{1}_L(c_j)$  for every  $j \in [u]$ . The second case is where  $B > 4\mathcal{E}$  enters: it is what keeps the two zones from overlapping. Figure 1 depicts it in the plane of  $(a_{j-1}, c_j)$ .

**The recurrence.** Lemma 1 cannot be used directly: its right side needs the true digit  $a_{j-1}$ , while what we hold is  $b_{j-1}$ , and the two are related by  $a_{j-1} \equiv b_{j-1} - \delta_{j-1} \pmod{B}$ . So we instead use the indicators of  $b_j$ . With  $P := \{0\}$ , for  $b \in [B]$  and  $\delta \in \{0, 1\}$ ,

$$\mathbb{1}_H((b - \delta) \bmod B) = \mathbb{1}_H(b) + \delta (\mathbb{1}_P(b) - \mathbb{1}[b = B - 2\mathcal{E}]), \quad (4)$$

since  $\mathbb{1}_H((b - 1) \bmod B)$  and  $\mathbb{1}_H(b)$  differ only at  $b = 0$ , where the former is 1, and at  $b = B - 2\mathcal{E}$ , where the latter is 1. Substituting (4) into Lemma 1 gives the following recurrence.

**Lemma 2.** *Let  $B > 4\mathcal{E}$ . For  $1 \leq j < u$ ,*

$$\delta_j = \mathbb{1}_L(c_j) (\mathbb{1}_H(b_{j-1}) + \mathbb{1}_P(b_{j-1}) \delta_{j-1}). \quad (5)$$

*Proof.* Lemma 1 with  $a = a_{j-1}$  gives  $\delta_j = \mathbb{1}_L(c_j) \mathbb{1}_H(a_{j-1})$ . Since  $a_{j-1} = (b_{j-1} - \delta_{j-1}) \bmod B$  by (3), applying (4) leaves only  $\mathbb{1}_L(c_j) \mathbb{1}[b_{j-1} = B - 2\mathcal{E}] \delta_{j-1} = 0$  to show. For this term to be nonzero we would need  $b_{j-1} = B - 2\mathcal{E}$  and  $\delta_{j-1} = 1$ , whence  $a_{j-1} = B - 2\mathcal{E} - 1$  and  $t_j := a_{j-1} + \mathcal{E} + \varepsilon_j < B - 1$ . Then  $\delta_j = 0$ , so  $c_j = t_j > B - 2\mathcal{E} - 1 \geq 2\mathcal{E}$ , the last step by  $B > 4\mathcal{E}$ . Hence  $c_j \notin L$ , and the term is zero.  $\square$

Starting from  $\hat{\sigma}_{-1} := 0$ , put

$$\hat{\delta}_j := \mathbb{1}_L(c_j) \hat{\sigma}_{j-1}, \quad \hat{\sigma}_j := \mathbb{1}_H(b_j) + \mathbb{1}_P(b_j) \hat{\delta}_j \quad (0 \leq j < u),$$

so that  $\hat{\delta}$  follows (5) and  $\hat{\sigma}_j = G_j + P_j \hat{\sigma}_{j-1}$  with

$$(G_j, P_j) := (\mathbb{1}_H(b_j), \mathbb{1}_L(c_j) \mathbb{1}_P(b_j)). \quad (6)$$

Unlike the coefficients of (5), which involve both  $c_j$  and  $b_{j-1}$ , both entries of (6) are functions of the single window  $w_j$ , so each pair is one lookup on its own slot (Section 3.3). In carry-lookahead terms,  $\hat{\sigma}_j$  is the carry out of position  $j$  before the factor  $\mathbb{1}_L(c_{j+1})$  is applied: it is generated when  $b_j \in H$ , propagated when  $b_j = 0$  and  $c_j \in L$ , and killed otherwise.

The initial value  $\hat{\sigma}_{-1} = 0$  is what the padding of Section 3.1 supplies: it holds  $b = 1$ , where  $\mathbb{1}_H$  and  $\mathbb{1}_P$  both vanish, so the initial value comes from the layout itself, and position 0 needs no separate handling. Since  $a_{-1} = 0$ ,  $\delta_0 = 0 = \hat{\delta}_0$ , so  $\hat{\delta}_j = \delta_j$  for every  $j$  by (5).

**Real input.** For a real message  $z \in [0, B^u)$ , the windows (3) hold the digits of  $\lfloor z \rfloor$ , with  $a_{-1} := \lfloor B(z - \lfloor z \rfloor) \rfloor$  its first fractional digit. The true carry  $\delta_0 = \mathbb{1}_H(a_{-1}) \mathbb{1}_L(c_0)$  of Lemma 1 is then out of reach, since the circuit sees  $\mathbb{1}_L(c_0)$  but not  $\mathbb{1}_H(a_{-1})$ . Nevertheless, the returned digits are those of  $\lfloor z \rfloor$  or  $\lfloor z \rfloor + 1$ .

**Proposition 1.** *Let  $B > 4\mathcal{E}$  and  $z \in [0, B^u)$ . The digits  $\hat{a}_j := b_j - \hat{\delta}_j \bmod B$  returned by the recurrence with initial value  $\hat{\delta}_0 = 0$  satisfy*

$$\sum_{j \in [u]} \hat{a}_j B^j \equiv \lfloor z \rfloor + \delta_0 \pmod{B^u},$$

where  $\delta_0 = \mathbb{1}_H(a_{-1}) \mathbb{1}_L(c_0)$  is the true carry at position 0, and this integer differs from  $z$  by

$$-1 < (\lfloor z \rfloor + \delta_0) - z < \frac{2\mathcal{E}}{B}.$$

In particular, if  $|z - k| < 1/B$  for an integer  $k$ , then  $\lfloor z \rfloor + \delta_0 = k$ ; for integer  $z$ ,  $\hat{a}_j = a_j$  for every  $j$ .

The carries  $\delta_j$  and  $\hat{\delta}_j$  differ only on a run of digits equal to  $B - 1$ ; the proof is in Appendix A.1.

**Resolving every carry at once.** The recurrence for  $\hat{\sigma}_j$  is affine in  $\hat{\sigma}_{j-1}$ . Writing  $(g, h)$  for the affine map  $s \mapsto g + hs$ , with intercept  $g$  and slope  $h$ ,

$$(g, h) * (g', h') = (g + hg', hh'),$$

so composing two such maps gives another, and the state stays a pair. Unrolling an affine recurrence  $s_j = g_j + h_j s_{j-1}$  gives  $s_j = g_{j:0} + h_{j:0} s_{-1}$ , where  $(g_{j:0}, h_{j:0}) := (g_j, h_j) * \dots * (g_0, h_0)$ ; with  $s_{-1} = 0$ ,  $s_j = g_{j:0}$ . Applying at  $(G_{j:0}, P_{j:0})$  the composites of (6) and  $G_{-1:0} := 0$ , this gives  $\hat{\sigma}_j = G_{j:0}$  and

$$\hat{\delta}_j = \mathbb{1}_L(c_j) G_{j-1:0}, \quad 0 \leq j < u. \quad (7)$$

Computed one after another, the composites  $(G_{j:0}, P_{j:0})$  take depth linear in  $u$ . Composition is associative, however, so they may be regrouped into a tree, as in the carry stage of the Kogge–Stone adder [KS73], and a parallel prefix scan forms all of them in  $\lceil \log_2 u \rceil$  rounds. Round  $r$  composes each position  $j$  with position  $j - 2^r$ , doubling the block it holds, so after  $\lceil \log_2 u \rceil$  rounds every position holds the composite of a block of length  $2^{\lceil \log_2 u \rceil} \geq u$ .

**Packing the pairs.** Every position of the carry scan holds a pair, and a pair of real numbers fits in a single complex slot, so all the pairs occupy one ciphertext, one position per slot.

[CPL26] place the three states a carry can be in — generate, propagate, or neither — on the complex values  $i$ ,  $\frac{1}{2}$  and  $0$ ; we extend that assignment to an arbitrary real pair. Put

$$\chi(g, h) := \frac{1}{2}(h + i g), \quad g, h \in \mathbb{R},$$

so that conjugation recovers  $g$  and  $h$  by  $i(\bar{\chi} - \chi)$  and  $\chi + \bar{\chi}$ , respectively. Since  $i$  is a Gaussian integer, neither recovery consumes a multiplicative level. We call  $\chi(g, h)$  the symbol of the pair. Expanding both sides shows that composition is

$$\chi * \chi' = \chi + (\chi + \bar{\chi}) \left( \chi' - \frac{1}{2} \right). \quad (8)$$

For the three carry values, [CPL26] writes  $\chi' - \chi$  in place of  $\chi' - \frac{1}{2}$ ; the two agree exactly when  $h^2 = h$  and  $gh = 0$ .

The pair the scan carries is (6), and its symbol is

$$\xi_j := \chi(G_j, P_j) = \frac{1}{2}(\mathbb{1}_L(c_j)\mathbb{1}_P(b_j) + i\mathbb{1}_H(b_j)).$$

Consecutive positions occupy adjacent slots (Section 3.1), so composing with position  $j - 2^r$  is the rotation  $\text{Rot}_{-2^r}(\cdot)$ , and round  $r$  is

$$\xi \mapsto \xi + (\xi + \bar{\xi})(\text{Rot}_{-2^r}(\xi) - \frac{1}{2}),$$

at one rotation, one conjugation and one multiplication per round. We write  $\text{PrefixScan}(\text{ct}, \xi)$  for the  $\lceil \log_2 u \rceil$  rounds applied to a ciphertext  $\text{ct}, \xi$  of symbols. Every padding position carries  $\xi = 0$ , which every round leaves unchanged. A position whose block reaches into the padding composes with  $\xi' = 0$  and returns  $(G, 0)$ : its slope is 0, so it is a fixed point of every later round, and its intercept is already the composite  $G_{j:0}$ . When  $j - 2^r < -1$ , the rotation brings position  $j$  a value from the block of  $z_{m-1}$  (indices modulo  $M$ ), but the block  $[j - 2^r + 1, j]$  that position  $j$  already holds then contains position  $-1$ , so position  $j$  is already a fixed point and is unaffected.

**Restoring the digits and evaluating functions of them.** After the last round, the symbol at position  $j$  has intercept  $G_{j:0}$ . If the carry itself is wanted, (7) gives it from

$$G_{j-1:0} = i(\overline{\text{Rot}_{-1}(\xi)} - \text{Rot}_{-1}(\xi)) \quad \text{at position } j,$$

and a multiplication by  $\mathbb{1}_L(c_j)$ .

Then any  $f: \mathbb{Z}_B \rightarrow \mathbb{R}$  of the restored digit is

$$\begin{aligned} f(\hat{a}_j) &= f(b_j) + \hat{\delta}_j [f(b_j - 1) - f(b_j)] \\ &= f(b_j) + G_{j-1:0} d(w_j), \end{aligned} \quad (9)$$

where  $d(w_j) := \mathbb{1}_L(c_j) [f(b_j - 1) - f(b_j)]$ . Both  $f(b_j)$  and  $d(w_j)$  are functions of  $w_j$ , which can be obtained by lookup. So evaluating  $f(\hat{a}_j)$  does not require restoring the digit first. With  $f = \text{id}$ , this returns the digits themselves.

*Remark 1* (Digit functions within the scan). The right side of (9) needs one multiplication by  $d(w_j)$  after the scan, which Algorithm 3 performs. This right side is the intercept of the composite of  $(f(b_j), d(w_j))$  with  $(G_{j-1:0}, P_{j-1:0})$ . Hence, putting  $\psi_j := \chi(f(b_j), d(w_j))$  and updating it in every round by (8),

$$\psi \mapsto \psi + (\psi + \bar{\psi})(\text{Rot}_{-2^r}(\xi) - \frac{1}{2}),$$

gives  $f(\hat{a}_j) = i(\bar{\psi}_j - \psi_j)$  after the last round, while the update of  $\xi$  is unchanged. This saves one level at the cost of one more conjugation and one more multiplication per round.

---

**Algorithm 3** Digit functions
 

---

**Input:**  $\text{ct} = \text{Enc}_Q(\text{Ecd}_{\text{slot}}(z')) \in \mathcal{R}_Q^2$  where  $z' \in \mathbb{R}^{N/2}$  is supported on  $\{m\Lambda : m \in [M]\}$ ,  $z'_{m\Lambda} = z_m \in [0, B^u)$ ; public  $f: \mathbb{Z}_B \rightarrow \mathbb{R}$ .

**Output:** Slot values  $f(\hat{a}_j)$  at position  $j$  of every message, with  $\hat{a}_j$  the digits of  $z_m$  given by Proposition 1.

- 1:  $\text{ct.w} \leftarrow \text{FormWindows}(\text{ct})$   $\triangleright$  Algorithm 2
  - 2:  $\text{ct.Rw} \leftarrow \text{Root}_{B^2}(\text{ct.w})$
  - 3:  $(\text{ct.}\xi, \text{ct.f}, \text{ct.d}) \leftarrow (\text{LUT}_{B^2}^\xi, \text{LUT}_{B^2}^f, \text{LUT}_{B^2}^d)(\text{ct.Rw})$   $\triangleright$  (6)
  - 4:  $\text{ct.}\xi \leftarrow \text{PrefixScan}(\text{ct.}\xi)$   $\triangleright$  Section 3.2
  - 5:  $\text{ct.R} \leftarrow \text{Rot}_{-1}(\text{ct.}\xi)$
  - 6: **return**  $\text{ct.f} + i(\text{Conj}(\text{ct.R}) - \text{ct.R}) \cdot \text{ct.d}$
- 

### 3.3 Root production and lookups

The scan of Section 3.2 and 9 use the values  $\xi_j$ ,  $f(b_j)$  and  $d(w_j)$ , all of which are functions of  $w_j$ . One call to  $\text{Root}_{B^2}$  on the window returns  $\text{ct.Rw}$ , holding  $\zeta_{B^2}^{w_j}$  at position  $j$ , on which every one of those values is a single  $B^2$ -point lookup. Since  $\xi_j$  is complex-valued, this makes four lookups on a single root (Section 2.3). Algorithm 3 states the route end-to-end.

The iterative digit decomposition of Section 2.3 needs  $u$  sequential root productions, whereas our method uses one root production for all  $u$  digit positions. In exchange, each message occupies  $\Lambda$  slots instead of one, which lowers throughput, but the latency no longer includes  $u$  sequential root productions. When the number of messages is at most  $N/(2\Lambda)$ , one ciphertext holds all of them under either method, and only the latency differs. The lookups share the powers of the root and take  $O(\log B)$  levels;  $\text{PrefixScan}$  then takes  $\lceil \log u \rceil$  levels, and the final multiplication takes more level.

*Remark 2* ( $B$ -point lookups on two roots). The values  $\xi_j$ ,  $f(b_j)$  and  $d(w_j)$  can also be formed from  $B$ -point lookups instead of  $B^2$ -point ones. Running the iterative digit decomposition of Section 2.3 with two digits on each window gives separate roots of period  $B$  for  $b_j$  and  $c_j$ , and lookups on these roots and products then form the values (Algorithm 5 in Appendix B). A  $B$ -point lookup is shallower and more accurate than a  $B^2$ -point one, but this route needs one more root production.

### 3.4 Discretization Error

The windows of (3) assume  $|\varepsilon_j| < \mathcal{E}$ , where  $\varepsilon_j$  is the deviation discretization leaves at position  $j$ . The following theorem bounds that deviation by the Hamming weight of the secret it runs under.

**Theorem 1** (Discretization error). *Let  $\text{ct} = (a, b) \in \mathcal{R}_Q^2$  decrypt to  $m := [b + as]_Q$  under a secret  $s$ , and for  $c \in \mathcal{R}_Q$  write  $r_{\text{in}}(c)$  and  $r_{\text{out}}(c)$  for the remainders left by the inner and the outer rounding of  $\text{Disc}_{W,Q}$ , each bounded by  $\frac{1}{2}$  in every coefficient. Then  $\text{Disc}_{W,Q}(\text{ct})$  decrypts to*

$$\text{Disc}_{W,Q}(m) + \frac{Q}{W}\varepsilon + e \pmod{Q},$$

where  $\varepsilon = r_{\text{in}}(b) + r_{\text{in}}(a)s - r_{\text{in}}(m)$  is an integer polynomial and  $e = r_{\text{out}}(b) + r_{\text{out}}(a)s - r_{\text{out}}(m)$ . In particular  $\|\varepsilon\|_\infty \leq \lceil \frac{\text{hwt}(s)+1}{2} \rceil$  and  $\|e\|_\infty \leq \frac{\text{hwt}(s)+2}{2}$ .

The proof is in Appendix A.2.

*Remark 3* (Switching to a sparse secret). The bound of Theorem 1 grows with  $\text{hwt}(s)$ , so we switch to a sparser secret  $\tilde{s}$  immediately before discretization (Algorithm 2, line 4). Sparse-secret-encapsulated bootstrapping [BTH22] already performs this switch; we pull it forward, before discretization, so it costs no extra key switch.

At  $W = B^2$ ,  $Q = Q_0$  and  $s = \tilde{s}$ , the value  $\varepsilon_j$  in (3) is the coefficient of  $\varepsilon$  at that position plus the difference between  $\text{Disc}(m)$  and  $Ba_j + a_{j-1} + \mathcal{E}$  there, which is 0 or 1, and  $\mathcal{E}$  is chosen to bound it. At the padding,  $\text{Disc}(m) = B + \mathcal{E}$ , so the same bound on the coefficient of  $\varepsilon$  keeps  $b = 1$  there. The term  $e$  leaves  $w_j$  unchanged and perturbs the root's argument by  $2\pi\|e\|_\infty/Q_0 \leq \pi(\text{hwt}(\tilde{s}) + 2)/Q_0$ , negligible since  $Q_0 \gg \text{hwt}(\tilde{s})$ .

*Remark 4* (Realizing the discretization). In [KN25], discretization is realized by switching the ciphertext modulus twice, each switch instantiated as a *Rational Rescale* [CCK<sup>+</sup>25]. Rational rescale relies on fast base conversion [CHK<sup>+</sup>19], whose error is added to the coefficients before they are snapped to the grid and therefore enters  $\varepsilon$ , raising its bound to  $O(\ell_{\text{RNS}}\text{hwt}(s))$ , where  $\ell_{\text{RNS}}$  is the number of RNS blocks the modulus change spans (Theorem 3.2 of [CCK<sup>+</sup>25]). Since that bound is what  $\mathcal{E}$  must accommodate, we instead reconstruct each coefficient exactly by CRT before rounding, which gives Theorem 1 with no  $\ell_{\text{RNS}}$ -dependence.

*Remark 5* (Probabilistic bound). Theorem 1 is a worst-case bound. Each coefficient of  $\varepsilon$  is a sum of  $\text{hwt}(s) + 2$  rounding remainders; modelling these as independent and uniform on  $[-\frac{1}{2}, \frac{1}{2}]$ , as in the failure-probability analysis of [BTH22], makes the coefficient a centred Irwin–Hall variable with  $\text{hwt}(s) + 2$  summands. Under this model, at  $\text{hwt}(\tilde{s}) = 32$  and  $N = 2^{16}$ , a union bound over the coefficients gives  $\Pr[\|\varepsilon\|_\infty \geq 14] \leq 2^{-56.9}$ , so that  $|\varepsilon_j| < 15$  even after the rounding of the sub-grid part, whereas the worst case  $\lceil(\text{hwt}(\tilde{s}) + 1)/2\rceil = 17$  would require  $B > 4 \cdot 19 = 76$ .

## 4 Sign Evaluation with Discretized Inputs

Nonlinear functions such as comparison and ReLU can be computed using the sign function. Comparison uses the sign of the difference between two inputs, while  $\text{ReLU}(x) = x(1 + \text{sgn}(x))/2$ . For a normalized real input  $x \in [-1, 1]$ , we require the output  $\widehat{\text{sgn}}(x)$  to satisfy

$$|x| \geq 2^{-p} \implies |\widehat{\text{sgn}}(x) - \text{sgn}(x)| \leq 2^{-q}.$$

Here  $p$  is the input precision: the requirement above applies to every input at least  $2^{-p}$  away from zero, and  $q$  is the required output precision.

To meet this requirement, we first discretize the input and compute the sign from the digit windows of Section 3, without recovering all digits. Section 4.1 gives the sign circuit and its correctness, Section 4.2 shows that its modulus consumption is linear in  $p$ , and Section 4.3 gives a quadratic lower bound for polynomial evaluation of the sign.

### 4.1 Sign evaluation

The sign is not a function of a single digit. When a signed input is represented modulo  $B^u$ , whether it is negative is decided by the most significant digit alone, but whether it is zero depends on all  $u$  digits. The method of Section 3 can restore an indicator of each digit being zero, but only after the scan, so the  $u$  positions must then be combined in a further step. Instead, we derive an affine recurrence whose composites the scan of Section 3.2 computes, so that the scan returns the sign directly.

**Sign recurrence.** For a real input  $y$ , Algorithm 2 yields the windows (3) of  $z := y \bmod B^u$ , since it works modulo  $B^u$ . When  $|y| < B^u/2$ ,  $y$  is the centered representative  $[z]_{B^u}$ , so we regard a residue modulo  $B^{j+1}$  as negative when its centered representative is negative. By Proposition 1, the digits  $\hat{a}_j$  that Section 3.2 restores from these windows are those of  $\hat{z} := [z] + \delta_0$  modulo  $B^u$ , so  $[\hat{z}]_{B^{j+1}} < 0$  exactly when  $\hat{a}_j \geq B/2$ . Let  $f := \mathbb{1}_{[B/2, B)}$ ,  $N_j := \mathbb{1}[[\hat{z}]_{B^{j+1}} < 0] = f(\hat{a}_j)$  for  $j \geq 0$ , and  $N_{-1} := 0$ . Then for  $j \in [u]$ ,

$$\begin{aligned} N_j &= f(b_j) + \hat{\delta}_j [f(b_j - 1) - f(b_j)] \\ &= f(b_j) + N_{j-1} \mathbb{1}_L(c_j) [f(b_j - 1) - f(b_j)], \end{aligned} \tag{10}$$

where the first line is (9). The second holds since  $\hat{\delta}_0 = 0$  and, for  $j \geq 1$ ,  $\hat{\delta}_j = \mathbb{1}_L(c_j)\mathbb{1}_H(\hat{a}_{j-1})$ ; if  $c_j \in L$ , then  $\hat{a}_{j-1} \in [0, 2\mathcal{E}) \cup H$ , on which  $\mathbb{1}_H$  agrees with  $f$  (see the proof of Proposition 2). As in (6), the intercept  $f(b_j)$  and the slope  $\mathbb{1}_L(c_j)[f(b_j - 1) - f(b_j)]$  of (10) are functions of the single window  $w_j$ . The scan of Section 3.2 composes one such affine map per position, and since  $N_{-1} = 0$ ,  $N_j$  is the intercept of the composite over positions  $0, \dots, j$ .

$N_{u-1}$  separates negative inputs from the others, but not zero from positive inputs. Let  $Z_j := \mathbb{1}[\hat{z}_{B^{j+1}} = 0]$  for  $j \geq 0$  and  $Z_{-1} := 1$ . By definition,  $Z_j = \mathbb{1}[\hat{a}_j = 0]Z_{j-1}$ , an affine recurrence with intercept 0 and slope  $\mathbb{1}[\hat{a}_j = 0]$ . Each slot of the scanned ciphertext holds one pair: the intercept and slope of an affine map. To have  $N_j$  and  $Z_j$  fit in one such pair, we make the two recurrences share their slope; with different slopes, a third value would be needed. Since  $f(x-1) - f(x) = \mathbb{1}[x=0] - \mathbb{1}[x=B/2]$ , replacing  $\mathbb{1}[\hat{a}_j = 0]$  by  $f(\hat{a}_j - 1) - f(\hat{a}_j)$  changes it only at  $\hat{a}_j = B/2$ , to  $-1$ . When  $Z_{j-1} = 1$  and  $j \geq 1$ , we have  $c_j \in L$  (see the proof of Proposition 2), and  $\hat{a}_{j-1} = 0 \notin H$  gives  $\hat{\delta}_j = \mathbb{1}_L(c_j)\mathbb{1}_H(\hat{a}_{j-1}) = 0$ , so  $\hat{a}_j = b_j$  and

$$\mathbb{1}_L(c_j)[f(b_j - 1) - f(b_j)] = f(\hat{a}_j - 1) - f(\hat{a}_j).$$

At  $j = 0$ ,  $\hat{a}_0 = b_0$  still holds, but  $c_0 \in L$  need not, so sharing the slope also introduces the factor  $\mathbb{1}_L(c_0)$ . Abusing notation, we write  $Z_j$  for the resulting recurrence,

$$Z_j = \mathbb{1}_L(c_j)[f(b_j - 1) - f(b_j)]Z_{j-1}, \quad Z_{-1} := 1,$$

and take

$$T := 1 - 2N_{u-1} - Z_{u-1}$$

as the output. Proposition 2 shows that, despite the resulting differences in  $Z_j$ ,  $T$  is the sign of  $y$  except near zero, where  $T$  may also be 0.

**Proposition 2.** *Let  $B$  be even with  $B > 4\mathcal{E}$  and  $|y| \leq B^u/2 - 1$ , and let the windows of  $z := y \bmod B^u$  satisfy (3). Then  $T \in \{0, \text{sgn}(y)\}$ . Moreover,  $T = \text{sgn}(y)$  if  $|y| \geq 2\mathcal{E}/B$ , and  $T = 0$  if  $|y| < 1/B$ .*

The proof is in Appendix A.3. When  $T = \text{sgn}(y)$ , we have  $1 - T^2 = \mathbb{1}[y = 0]$  and  $(T^2 + T)/2 = \mathbb{1}[y > 0]$ . To compare reals  $x$  and  $x'$ , we input  $y = 2^p(x - x')$ , obtained by reinterpreting the scale; let  $y'$  be the value the ciphertext encrypts, and assume  $|y'| \leq 2^p$  and  $|y' - y| < 1/B$ . Then the condition  $|y'| \leq B^u/2 - 1$  of Proposition 2 holds once  $B^u > 2^{p+1}$ . If  $x = x'$  or  $|x - x'| \geq 2^{-p}$ , then  $|y'| < 1/B$  or  $|y'| > 1 - 1/B \geq 2\mathcal{E}/B$  with  $\text{sgn}(y') = \text{sgn}(y)$ , so  $T = \text{sgn}(x - x')$  by Proposition 2, and  $1 - T^2 = \mathbb{1}[x = x']$  and  $(T^2 + T)/2 = \mathbb{1}[x > x']$ .

**Evaluating the recurrence.** At each position, we pack the pair as the symbol

$$\psi_j := \chi(f(b_j), \mathbb{1}_L(c_j)[f(b_j - 1) - f(b_j)]).$$

Since  $\psi_j$  is a function of  $w_j$ , a lookup on the root of the windows produces  $\psi_j$  at every position, sharing the windows and the root with Algorithm 3. At the padding,  $b = 1$  gives  $\psi = 0$ . To match the initial values  $(N_{-1}, Z_{-1}) = (0, 1)$ , we add  $\frac{1}{2}\mathbf{1}_{\text{pad}}$ , where  $\mathbf{1}_{\text{pad}} \in \{0, 1\}^{N/2}$  is 1 exactly at the padding, so that the padding holds the identity map  $\chi(0, 1) = \frac{1}{2}$ , and then apply **PrefixScan**. Only position  $u - 1$  is used, and the composite it holds after the scan covers  $2^{\lceil \log_2 u \rceil} \leq \Lambda$  positions, which are digit positions of the same message or padding. Writing  $\psi$  for the symbol at position  $u - 1$  after the scan, we have

$$T = 1 + (2i - 1)\psi - (2i + 1)\bar{\psi},$$

so no level is consumed after  $\psi$  is obtained. Algorithm 4 summarizes the circuit, which uses one  $\text{Root}_{B^2}$  and one complex-valued lookup, that is, two real lookups.

---

**Algorithm 4** Sign evaluation

---

**Input:**  $\text{ct} = \text{Enc}_Q(\text{Ecd}_{\text{slot}}(\mathbf{z}')) \in \mathcal{R}_Q^2$  where  $\mathbf{z}' \in \mathbb{R}^{N/2}$  is supported on  $\{m\Lambda : m \in [M]\}$  and  $\mathbf{z}'_{m\Lambda} = y_m$  with  $|y_m| \leq B^u/2 - 1$ .

**Output:** Slot values  $T$  at position  $u - 1$  of every message.

- 1:  $\text{ct.w} \leftarrow \text{FormWindows}(\text{ct})$   $\triangleright$  Algorithm 2
  - 2:  $\text{ct.Rw} \leftarrow \text{Root}_{B^2}(\text{ct.w})$
  - 3:  $\text{ct.}\psi \leftarrow \text{LUT}_{B^2}^\psi(\text{ct.Rw}) + \text{Ecd}_{\text{slot}}(\frac{1}{2}\mathbf{1}_{\text{pad}})$
  - 4:  $\text{ct.}\psi \leftarrow \text{PrefixScan}(\text{ct.}\psi)$
  - 5: **return**  $1 + (2i - 1)\text{ct.}\psi - (2i + 1)\text{Conj}(\text{ct.}\psi)$
- 

*Remark 6* (Cleaning the ternary output). Proposition 2 concerns  $T$  as defined above; the circuit returns it up to CKKS error. Since  $T \in \{0, \pm 1\}$  for every input within the range, a cleaning polynomial with  $\mathcal{V} = \{-1, 0, 1\}$ , suppresses the CKKS error of the output for every such input. By contrast, the output of a polynomial approximation of the sign is continuous, so on the gap around zero it takes every value between  $-1$  and  $1$ , and for the inputs mapped away from  $-1$ ,  $0$  and  $1$ , applying the cleaning polynomial has no specified target.

## 4.2 Cost of the Discretized Route

The output precision  $q$  depends on the application: comparison needs only a few bits, while ReLU and  $\max(a, b)$ , which multiply the sign by its input, need  $q$  close to  $p$ . A larger  $q$  is rarely useful: the error of ReLU inside the gap is already up to  $2^{-p}$ , so  $q$  beyond  $p$  helps only when the sign is multiplied by larger values or used many times, which adds a logarithmic number of bits. We state the costs for general  $p$  and  $q$ , since the gap and the output precision enter different parts of the cost. We call  $(-\gamma, \gamma)$  with  $\gamma := 2^{-p}$  the *gap*: it contains the inputs that are too close to 0 for their sign to be decided at precision  $p$ , and there we only require  $|\widehat{\text{sgn}}(x)| \leq 1$ . We run Algorithm 4 on  $y = 2^p x$  as in Section 4.1, with  $B > 4\mathcal{E}$  and  $B^u > 2^{p+1}$ , so that  $u = \Theta(p/\log B)$ . By Proposition 2,  $T = \text{sgn}(x)$  for  $|x| \geq \gamma$  and  $T \in \{0, \text{sgn}(x)\} \subset [-1, 1]$  for  $|x| < \gamma$ , so  $T$  meets the requirement in exact arithmetic, and cleaning reduces the CKKS error of the evaluated  $T$  below  $2^{-q}$ .

The cost has four parts: the input stage, which holds the input at  $p + \log_2 B + O(1)$  bits through the fanout and slot-to-coefficient conversion before discretization; the root production of Algorithm 4, one bootstrap; the window read and prefix scan, of depth  $O(\log B) + \lceil \log_2 u \rceil$ ; and the cleaning to  $q$  bits. Each rescaling adds a rounding noise of canonical norm at most  $\|e_{\text{rs}}\|_{\text{can}}$  at every level.

**Proposition 3** (Modulus cost of the digit circuit). *The values carried by the window reads and the prefix scan are bounded and their targets lie in a finite set, but lookup errors accumulate through the scan, so the scan needs  $O(\log u)$  bits of working precision and its scale is  $2^{O(\log u)}\|e_{\text{rs}}\|_{\text{can}}$ . For fixed  $B$ , the digit circuit consumes*

$$M_{\text{mod}}^{\text{digit}} = O((\log B + \log u) \log u) = O((\log p)^2).$$

The proof is in Appendix A.4. The working precision of the digit circuit does not depend on  $q$ : its target values lie in a finite set, and as long as the output is close enough to the target value, cleaning brings the output to that value with  $q$  bits of precision.

**Cleaning.** For ternary values  $t \in \{-1, 0, 1\}$  we use the cleaning polynomial  $\frac{1}{2}(5t^3 - 3t^5)$ . Further folding the ternary output to a bit by  $(T^2 + T)/2$  allows using  $3t^2 - 2t^3$ . Since each iteration squares the error of the discrete output up to a constant factor,  $O(\log q)$  iterations reach  $q$  bits of precision. Evaluating each iteration at a scale matched to its output precision

makes the total consumed modulus a geometric sum, so  $M_{\text{mod}}^{\text{clean}} = \Theta(q)$ ; Appendix C gives the bound including the homomorphic noise of each iteration.

**Proposition 4** (Modulus budget of the discretized route). *Under the window hypothesis (3), the discretized route evaluates the sign with  $\gamma = 2^{-p}$  to output precision  $q$  with*

$$M_{\text{mod}}^{\text{disc}} = \underbrace{\Theta(p)}_{\text{input stage}} + \underbrace{O((\log p)^2)}_{M_{\text{mod}}^{\text{digit}}} + \underbrace{\Theta(q)}_{M_{\text{mod}}^{\text{clean}}},$$

which is  $\Theta(p)$  for  $q = O(p)$ , together with a single root production of precision  $O(\log B + \log u)$  bits.

*Proof.* The input stage holds the input at  $p + \log_2 B + O(1)$  bits over a constant number of levels; the other two terms are Proposition 3 and the cleaning bound above.  $\square$

### 4.3 Asymptotic Analysis for Polynomial Routes

Previous methods evaluate the sign by applying polynomials to the real input directly; we call such a circuit a *polynomial route* and, to compare it with the discretized route of Section 4.2, give a lower bound of its modulus consumption. A polynomial route must keep its values at  $x = \pm\gamma$  apart at every level, while the circuit of Section 4.1 only needs to decide which grid point its input should land on. We consider the composite polynomial methods of [CKK20] and their minimax refinement [LLNK22] as the previous state of the art. [CKK20] show that comparing  $a, b \in [0, 1]$  with  $|a - b| \geq \gamma$ , which is the sign of  $a - b$  with gap  $\gamma$ , to output precision  $q$  needs  $\Theta(\log(1/\gamma)) + \Theta(\log q)$  compositions of constant-degree polynomials, and that this is asymptotically optimal. The depth  $\Theta(p)$  is thus paid for the gap, for amplifying the gap margin to  $\Omega(1)$ , while the output precision costs only  $\Theta(\log q)$ .

This depth is necessary for any polynomial route. We show that the modulus consumption is then quadratic; to do so, we bound the separation at every level rather than only at the output. Write  $P_\ell$  for the polynomial in  $x$  that the circuit holds at level  $\ell$ , with  $P_0(x) = x$  and  $P_D$  the output, and  $A_\ell := \sup\{|P_\ell(x)| : x \in [-1, 1]\}$  for its magnitude. We assume that every product is rescaled before it is multiplied again, and that all values at level  $\ell$  share one scale  $\Delta_\ell$  and, before the output, have magnitude at most 1, as for the iterates and powers in composite polynomial evaluation. The scales need not be RNS primes, so we assume rescaling to an arbitrary bit-length, as rational rescale [CCK<sup>+</sup>25] provides.

Define the *separation* at level  $\ell$  as  $d_{\text{sep},\ell} := |P_\ell(\gamma) - P_\ell(-\gamma)|$ , the relative separation as  $\sigma_\ell := d_{\text{sep},\ell}/A_\ell$ , and write  $\theta_\ell := \log_2(\sigma_\ell/\sigma_0)$  for its cumulative gain, in bits, up to level  $\ell$ . As in Section 4.2, a ciphertext at scale  $\Delta$  that has been rescaled at least once may carry an error of size  $\|e_{\text{rs}}\|_{\text{can}}/\Delta$  in its decoded value. We require correctness for every noise within these bounds. If the noise at level  $\ell$  can reach half the separation, it can make two boundary inputs indistinguishable, and no later computation separates them again. Hence at every level the noise must stay below the separation, for at least one value if several ciphertexts are present, and we take  $P_\ell$  to be that value; for  $r \geq 1$  bits of margin,  $\|e_{\text{rs}}\|_{\text{can}}/\Delta_\ell < 2^{-r}d_{\text{sep},\ell}$ . As  $A_\ell \leq 1$ , we have  $d_{\text{sep},\ell} \leq \sigma_\ell = 2\gamma \cdot 2^{\theta_\ell} = 2^{\theta_\ell+1-p}$ , so the scale must satisfy

$$\log_2 \Delta_\ell \geq r' + p - 1 - \theta_\ell, \quad r' := r + \log_2 \|e_{\text{rs}}\|_{\text{can}}.$$

Counting only the rescales, each consuming  $2 \log_2 \Delta_\ell - \log_2 \Delta_{\ell+1}$  bits from level  $\ell$  to  $\ell + 1$ , gives a lower bound on  $M_{\text{mod}}$ .

**Proposition 5** (Lower bound for polynomial sign evaluation). *Under the assumptions above, let a circuit of depth  $D$  evaluate the sign with an output precision  $q \geq 1$ , satisfy  $\|e_{\text{rs}}\|_{\text{can}}/\Delta_\ell <$*

$2^{-r}d_{\text{sep},\ell}$  at every level for some  $r \geq 1$ , and return its output at a scale of  $r' + O(p)$  bits. Then  $\sigma_\ell \leq 2^{\ell+1}\gamma/\sqrt{1-\gamma^2}$  for every  $\ell$ , so  $D \geq p - O(1)$ , and in this model, for any scale schedule,

$$M_{\text{mod}} \geq \sum_{\ell < D} (r' + p - 1 - \theta_\ell) - O(p) \geq \frac{p^2}{2} + r'p - O(p).$$

*Proof.* Since  $P_\ell(\gamma) - P_\ell(-\gamma)$  is the integral of  $P'_\ell$  over  $[-\gamma, \gamma]$ ,  $d_{\text{sep},\ell} \leq 2\gamma \max_{|x| \leq \gamma} |P'_\ell(x)|$ , and as  $P_\ell$  has degree at most  $2^\ell$ , which holds for complex coefficients, Bernstein's inequality  $|P'_\ell(x)| \leq 2^\ell A_\ell / \sqrt{1-x^2}$  gives the bound on  $\sigma_\ell$ ; as the output lies within  $2^{-q}$  of  $\pm 1$  at  $\pm\gamma$  and has magnitude at most  $1 + 2^{-q}$ ,  $\sigma_D = \Omega(1)$ , so  $D \geq p - O(1)$ . The bound on  $\sigma_\ell$  gives  $\theta_\ell \leq \ell + O(\gamma^2)$ , so the scale inequality bounds each  $\log_2 \Delta_\ell$  separately, regardless of the scale schedule. Summing the per-level modulus cost gives  $M_{\text{mod}} \geq \sum_{\ell < D} \log_2 \Delta_\ell + \log_2(\Delta_0/\Delta_D)$ , whose boundary term is  $-O(p)$  by the scale inequality at  $\ell = 0$  and the bound on the output scale. Since  $\sigma_\ell \leq 2$ , also  $\theta_\ell \leq p$ , so every term of the sum is at least  $r' - 1 \geq 0$ , and the terms with  $\ell < p - O(1)$  give the stated bound.  $\square$

Appendix D gives a sharper bound for the iterated constructions in the literature, NewComp and NewCompG [CKK20] and the domain-extension polynomials of [CKP22a]: iterating an odd polynomial  $\phi$  costs at least  $p^2/(2\kappa_\phi)$ , where  $\kappa_\phi < 1$  is the separation gain per level of  $\phi$ .

**Theorem 2** (Asymptotic separation). *Under the noise and cost models of this section, with  $1 \leq q = O(p)$ , every circuit as in Proposition 5 uses  $\Omega(p^2)$  modulus bits, and the composite method of [CKK20] uses  $\Theta(p^2)$ . Under the window hypothesis (3), the discretized route uses  $\Theta(p)$  modulus bits and one root production.*

*Proof.* The lower bound is Proposition 5. The composite method uses  $O(p + \log q) = O(p)$  levels, and a scale of  $r' + p + q + O(1) = r' + O(p)$  bits at every level meets both the scale inequality and the output precision, so it uses  $O(p^2)$  bits. The discretized route is Proposition 4.  $\square$

**Proposition 6** (Bootstrapping precision). *In a circuit as in Proposition 5, a bootstrap applied at level  $\ell$  must keep its added error below  $2^{-r}d_{\text{sep},\ell}$ , and thus preserve at least  $r + p - 1 - \theta_\ell$  bits, in particular  $r + p - 1$  bits at the first level. In the discretized route, the root production and every bootstrap of the digit circuit need only  $O(\log B + \log u)$  bits.*

*Proof.* The noise argument before Proposition 5 applies to any error added at level  $\ell$ , including the error of a bootstrap; the second claim follows from Proposition 3.  $\square$

A bootstrap preserves only a bounded absolute precision  $p_{\text{BTS}}$ , set by its parameters; beyond it, a high-precision bootstrap such as in [CKSS25] or Meta-BTS [BCC<sup>+</sup>22] with  $\Theta(p/p_{\text{BTS}})$  bootstraps is needed. A polynomial route therefore needs high-precision bootstrapping and large scale factors from its first level, while it amplifies the gap margin, whereas the discretized route needs large scales only for the input stage, and high-precision bootstrapping not before the cleaning. In the cleaning, the values already lie near  $\{-1, 0, 1\}$ , so a bootstrap at standard precision followed by further cleaning suffices, provided the final cleaning iterations, about  $2q$  bits of modulus, fit between two bootstraps.

*Remark 7.* (Slot usage) Each message occupies  $\Lambda = 2^{\lceil \log_2(u+1) \rceil} = \Theta(u) = \Theta(p/\log B)$  slots, against one slot for a polynomial route. Per-value throughput therefore loses a factor  $\Theta(p/\log B)$ , and the advantage of Theorem 2 is lost in an amortized comparison at fixed  $B$ ; it is a trade-off of slot usage against modulus budget. Under a latency metric the advantage remains: the bootstraps of a polynomial route form a sequential chain that batching cannot eliminate.

Table 1: Digit decomposition of one exact coefficient integer. Time is the mean  $\pm$  sample standard deviation over five trials; output precision is computed from the worst observed complex target error. BTS counts actual bootstrap calls.

| Digits $u$ | Ours             |             |     | AKP [AKP25]      |             |     |
|------------|------------------|-------------|-----|------------------|-------------|-----|
|            | Time (s)         | Output bits | BTS | Time (s)         | Output bits | BTS |
| 2          | $22.12 \pm 1.16$ | 7.74        | 1   | $8.20 \pm 0.18$  | 7.94        | 2   |
| 4          | $23.56 \pm 0.60$ | 7.39        | 1   | $17.51 \pm 0.37$ | 8.09        | 4   |
| 6          | $23.18 \pm 0.66$ | 6.25        | 1   | $25.77 \pm 0.92$ | 7.17        | 6   |
| 8          | $23.58 \pm 0.80$ | 7.09        | 1   | $36.52 \pm 1.06$ | 8.06        | 8   |
| 10         | $22.25 \pm 0.62$ | 6.42        | 1   | $45.39 \pm 1.21$ | 8.26        | 10  |

## 5 Implementation and Evaluation

**Environment and parameters.** We compare our circuits with AKP’s iterative digit decomposition [AKP25] and NewCompG’s polynomial sign evaluation [CKK20]. All implementations use HEaven/HEaaN2 with eight CPU threads; runs are sequential. We fix  $N = 2^{16}$  and  $B = 64$ . Our circuits use a  $B^2 = 4096$  window, one bootstrapping, and post-bootstrap scale  $2^{38}$ , without output cleaning. For separation  $2^{-p}$ , the sign circuit uses  $u = \lceil p/6 \rceil + 1$  digits, including a guard digit.

NewCompG evaluates its final  $f$  polynomial once. Its ordinary scale is  $2^{40}$  for  $p \leq 24$  and  $2^{46}$  for  $p = 30$ . For  $p \geq 36$ , it uses a high-precision prefix with nominal scale  $2^{97}$  and a 112-bit base modulus, followed by an ordinary tail at scale  $2^{48}$  with a 57-bit base modulus.

**Measurements.** We measure total latency from encrypted input to encrypted output, including input conversions, bootstrapping, and final-output computation. Setup, key generation, public-coefficient preparation, encryption, warmup, and diagnostic decryption and validation are excluded. We report the mean and sample standard deviation over five trials per setting, reusing the 25 AKP measurements for the same coefficient-input workload. BTS counts refer to calls to the library’s bootstrap routines; their internal branches are not counted separately.

NewCompG starts from a fresh encryption with the post-bootstrap modulus budget. A preceding bootstrap and its noise are excluded, while bootstraps within the evaluated circuits are included in both latency and output error. Our circuit uses  $\Theta(u)$  slots per input; the reported timings compare matched workloads without evaluating maximum SIMD throughput.

**Accuracy.** Observed output precision is  $-\log_2 e_{\max}$ , where  $e_{\max}$  is the maximum complex target error over all tested outputs and five trials at a setting. Every rounded real digit is correct with finite complex error below  $1/2$ ; every sign output has the correct sign with finite complex error at most  $2^{-7}$ .

### 5.1 Digit Decomposition

Digit decomposition uses  $f(d) = d$  in our digit-function circuit. For  $u \in \{2, 4, 6, 8, 10\}$ , both methods extract all  $u$  digits from the same exact encrypted coefficient integer modulo  $B^u$ ; the largest tested modulus is  $64^{10} = 2^{60}$ .

Our mean latency stays between 22.12 and 23.58 seconds, while AKP’s increases from 8.20 to 45.39 seconds (Table 1; Figure 2, left). AKP is faster at  $u = 2, 4$ ; ours is faster at  $u = 6, 8, 10$ , reaching  $2.04\times$  at  $u = 10$ . Our circuit uses one bootstrapping, compared with  $u$  sequential bootstrappings for AKP.

Table 2: Sign evaluation on the same 1,024 real input pairs per trial. Time is the mean  $\pm$  sample standard deviation over five trials; output precision is computed from the worst observed complex target error. BTS counts actual bootstrap calls.

| Separation<br>bits $p$ | Ours             |                |     | NewCompG [CKK20]  |                |     |
|------------------------|------------------|----------------|-----|-------------------|----------------|-----|
|                        | Time (s)         | Output<br>bits | BTS | Time (s)          | Output<br>bits | BTS |
| 12                     | $16.63 \pm 0.73$ | 10.51          | 1   | $3.23 \pm 0.08$   | 7.62           | 1   |
| 18                     | $16.66 \pm 0.67$ | 9.73           | 1   | $4.83 \pm 0.12$   | 7.62           | 1   |
| 24                     | $17.74 \pm 0.78$ | 10.03          | 1   | $12.09 \pm 0.62$  | 7.62           | 2   |
| 30                     | $17.39 \pm 0.69$ | 9.58           | 1   | $13.76 \pm 0.24$  | 7.62           | 3   |
| 36                     | $17.11 \pm 0.62$ | 9.73           | 1   | $46.07 \pm 0.95$  | 7.62           | 8   |
| 42                     | $17.23 \pm 1.05$ | 9.75           | 1   | $74.49 \pm 3.31$  | 7.62           | 11  |
| 48                     | $16.95 \pm 0.71$ | 9.45           | 1   | $89.91 \pm 1.99$  | 7.62           | 13  |
| 54                     | $17.26 \pm 0.66$ | 9.54           | 1   | $106.10 \pm 2.81$ | 7.62           | 15  |
| 60                     | $16.36 \pm 0.40$ | 9.48           | 1   | $128.06 \pm 2.76$ | 7.62           | 18  |

## 5.2 Sign Evaluation

For  $p \in \{12, 18, 24, 30, 36, 42, 48, 54, 60\}$ , both methods evaluate  $\text{sgn}(a - b)$  on the same 1,024 real input pairs per trial, with minimum nonzero separation  $2^{-p}$ . Input conversion preserves the  $2^{-p}$  resolution. After discretization, we produce the roots at the precision required by the  $B^2$ -point windows and the subsequent lookups and scan. We keep the bootstrap parameters fixed across all tested  $p$ , while increasing the precision of input conversion with  $p$ .

Our mean batch latency stays between 16.36 and 17.74 seconds (Table 2; Figure 2, right). NewCompG is faster at the tested settings with  $p \leq 30$ , and ours at  $p \geq 36$ . At  $p = 60$ , the times are 16.36 and 128.06 seconds, a  $7.83\times$  speedup, with one bootstrap for ours and 18 for NewCompG (14 high-precision and four ordinary).

Observed output precision is 9.45–10.51 bits for our circuit and about 7.62 bits for NewCompG.

**Modulus consumption.** Outside bootstrapping, cumulative Q consumption is about 768 bits for our digit circuit and 6,152 bits for AKP at  $u = 10$ , and 1,000 bits for our sign circuit and 11,108 bits for NewCompG at  $p = 60$ . These are cumulative drops across evaluated segments, not peak Q or QP.

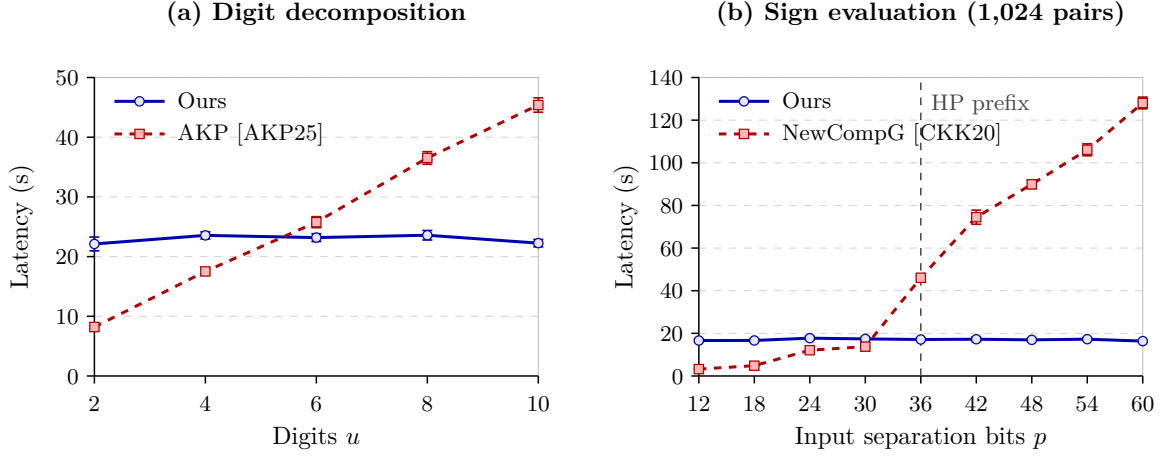


Figure 2: Online latency for (left) extracting all  $u$  radix-64 digits from one exact coefficient integer against AKP [AKP25], and (right) sign evaluation on 1,024 matched real input pairs against NewCompG [CKK20]. Points show means and error bars show sample standard deviations over five trials; lines are visual guides. In the sign experiment, the minimum nonzero separation is  $2^{-p}$ , and NewCompG uses a high-precision prefix for the tested settings  $p \geq 36$ .

## A Omitted Proofs

### A.1 Proof of Proposition 1

*Proof.* Both  $\delta$  and  $\hat{\delta}$  satisfy (5), whose coefficients depend only on  $b_{j-1}$  and  $c_j$ . Their difference  $\nu_j := \delta_j - \hat{\delta}_j$  therefore satisfies

$$\nu_j = \mathbb{1}_L(c_j) \mathbb{1}_P(b_{j-1}) \nu_{j-1}, \quad \nu_0 = \delta_0,$$

and  $\hat{a}_j \equiv a_j + \nu_j \pmod{B}$ . We show by induction that

$$\nu_j = \delta_0 \mathbb{1}[a_0 = \dots = a_{j-1} = B-1].$$

Suppose  $\nu_{j-1} = 1$ . Then  $\delta_{j-1} = 1$  and  $b_{j-1} = (a_{j-1} + 1) \bmod B$ , so  $b_{j-1} = 0$  exactly when  $a_{j-1} = B-1$ , in which case  $c_j = \mathcal{E} + \varepsilon_j - 1 \in L$ . Hence  $\nu_j = \mathbb{1}[a_{j-1} = B-1] \nu_{j-1}$ . This is the carry into position  $j$  when  $\delta_0$  is added to  $\lfloor z \rfloor$ , so  $\hat{a}_j$  is the digit of  $\lfloor z \rfloor + \delta_0$  at position  $j$ .

It remains to consider  $\delta_0$ , which is 1 exactly when  $a_{-1} + \mathcal{E} + \varepsilon_0 \geq B$ . Write  $\{z\} := z - \lfloor z \rfloor \in [0, 1)$ . The lower bound follows from  $(\lfloor z \rfloor + \delta_0) - z = \delta_0 - \{z\}$ . If  $\delta_0 = 1$ , then  $a_{-1} > B - 2\mathcal{E}$  because  $|\varepsilon_0| < \mathcal{E}$ , so  $\{z\} > 1 - 2\mathcal{E}/B$ .

Now let  $|z - k| < 1/B$  for an integer  $k$ . If  $z \geq k$ , then  $\lfloor z \rfloor = k$  and  $a_{-1} = 0$ , so  $\delta_0 = 0$  as  $\mathcal{E} + \varepsilon_0 < B$ . If  $z < k$ , then  $\lfloor z \rfloor = k-1$  and  $a_{-1} = B-1$ , so  $\delta_0 = 1$  as  $\varepsilon_0 > -\mathcal{E}$ . Either way,  $\lfloor z \rfloor + \delta_0 = k$ . For integer  $z$ , the case  $k = z$  gives  $\delta_0 = 0$ , hence  $\nu_j = 0$  and  $\hat{a}_j = a_j$ .  $\square$

### A.2 Proof of Theorem 1

*Proof.* Write  $\text{Disc} := \text{Disc}_{W,Q}$ , and for  $c \in \mathcal{R}_Q$  let  $c^* := \lfloor \frac{W}{Q} c \rfloor$ , an integer polynomial, so that  $\text{Disc}(c) = \lfloor \frac{Q}{W} c^* \rfloor$ . Each of the two roundings leaves a remainder,

$$r_{\text{in}}(c) := c^* - \frac{W}{Q} c, \quad r_{\text{out}}(c) := \text{Disc}(c) - \frac{Q}{W} c^*, \quad \|r_{\text{in}}(c)\|_{\infty} \leq \frac{1}{2}, \quad \|r_{\text{out}}(c)\|_{\infty} \leq \frac{1}{2}.$$

Fix representatives of  $a$  and  $b$ , and let  $I$  be the integer polynomial with  $b + as = m + QI$ .

The inner rounding produces the part of the error that is a multiple of  $Q/W$ . Applying it to  $a$ ,  $b$  and  $m$ ,

$$\begin{aligned} b^* + a^*s - m^* &= \frac{W}{Q}(b + as - m) + r_{\text{in}}(b) + r_{\text{in}}(a)s - r_{\text{in}}(m) \\ &= WI + \varepsilon, \end{aligned}$$

where  $\varepsilon := r_{\text{in}}(b) + r_{\text{in}}(a)s - r_{\text{in}}(m)$ . Since  $\varepsilon = b^* + a^*s - m^* - WI$ , it is an integer polynomial, and  $\|r_{\text{in}}(a)s\|_{\infty} \leq \|r_{\text{in}}(a)\|_{\infty} \text{hwt}(s)$  gives  $\|\varepsilon\|_{\infty} \leq \frac{\text{hwt}(s)+2}{2}$ ; by integrality  $\|\varepsilon\|_{\infty} \leq \lceil \frac{\text{hwt}(s)+1}{2} \rceil$ .

The outer rounding produces the rest of the error:

$$\begin{aligned} \text{Disc}(b) + \text{Disc}(a)s &= \frac{Q}{W}(b^* + a^*s) + r_{\text{out}}(b) + r_{\text{out}}(a)s \\ &= \frac{Q}{W}(m^* + WI + \varepsilon) + r_{\text{out}}(b) + r_{\text{out}}(a)s \\ &= \text{Disc}(m) + QI + \frac{Q}{W}\varepsilon + e, \end{aligned}$$

where  $e := r_{\text{out}}(b) + r_{\text{out}}(a)s - r_{\text{out}}(m)$  satisfies  $\|e\|_{\infty} \leq \frac{\text{hwt}(s)+2}{2}$ . Since  $QI$  vanishes modulo  $Q$ , this is the stated form.  $\square$

### A.3 Proof of Proposition 2

*Proof.* Here  $Z_j$  is the recurrence with the shared slope. By Appendix A.1,  $\nu_j := \delta_j - \hat{\delta}_j \in \{0, 1\}$  and  $\hat{a}_j \equiv a_j + \nu_j \pmod{B}$ .

We first prove the second line of (10). Let  $j \geq 1$ . Since  $\hat{\delta}_{j-1} \leq \delta_{j-1}$ , the proof of Lemma 2 also gives  $\mathbb{1}_L(c_j)\mathbb{1}[b_{j-1} = B - 2\mathcal{E}]\hat{\delta}_{j-1} = 0$ . Applying (4) to  $\hat{a}_{j-1} = (b_{j-1} - \hat{\delta}_{j-1}) \bmod B$  in the definition of  $\hat{\delta}_j$  then yields

$$\hat{\delta}_j = \mathbb{1}_L(c_j)\mathbb{1}_H(\hat{a}_{j-1}).$$

Now let  $c_j \in L$ . If  $\delta_j = 0$ , then  $a_{j-1} \leq 2\mathcal{E} - 2$ , so  $\hat{a}_{j-1} < 2\mathcal{E}$ . If  $\delta_j = 1$ , then  $a_{j-1} \in H$  by Lemma 1, so  $\hat{a}_{j-1} \in H \cup \{0\}$ . As  $B > 4\mathcal{E}$ ,  $\mathbb{1}_H$  agrees with  $f$  on  $[0, 2\mathcal{E}) \cup H$ , and hence  $\hat{\delta}_j = \mathbb{1}_L(c_j)\mathbb{1}_H(\hat{a}_{j-1})$ .

Next we show by induction on  $j$  that

$$Z_j = \mathbb{1}_L(c_0)(\mathbb{1}[[\hat{z}]_{B^{j+1}} = 0] - \mathbb{1}[[\hat{z}]_{B^{j+1}} = -B^{j+1}/2]).$$

For  $j = 0$ , this follows from  $b_0 = \hat{a}_0$  and  $f(x-1) - f(x) = \mathbb{1}[x=0] - \mathbb{1}[x=B/2]$ . For  $j \geq 1$ , there are three cases.

- If  $[\hat{z}]_{B^j} = 0$ , then  $\hat{a}_{j-1} = 0 \notin H$ , so  $\hat{\delta}_j = 0$  and  $b_j = \hat{a}_j$ . Also  $\hat{\delta}_{j-1} = 0$  and  $b_{j-1} = 0$ , so  $(a_{j-1}, \delta_{j-1})$  is  $(0, 0)$  or  $(B-1, 1)$ . Accordingly,  $c_j$  is  $\mathcal{E} + \varepsilon_j$  or  $\mathcal{E} + \varepsilon_j - 1$ , which lies in  $L$  either way. The slope is therefore  $f(\hat{a}_j - 1) - f(\hat{a}_j)$ .
- If  $[\hat{z}]_{B^j} = -B^j/2$ , then as before  $b_{j-1} = B/2$ , so  $a_{j-1} \in \{B/2, B/2 - 1\}$ , which lies outside  $H$ . Thus  $\delta_j = 0$  and  $c_j \geq B/2 > 2\mathcal{E}$ , so the slope is 0.
- Otherwise,  $Z_{j-1} = 0$ .

In the last two cases, the right side is also 0, since  $[\hat{z}]_{B^{j+1}}$  is not a multiple of  $B^j$ .

Finally, let  $n := [\hat{z}]_{B^u}$ . By  $|y| \leq B^u/2 - 1$  and Proposition 1,  $n - y$  and  $\hat{z} - z$  are congruent modulo  $B^u$  and differ by less than  $B^u$ , so  $n - y = \hat{z} - z \in (-1, 2\mathcal{E}/B)$ . In particular,  $n > y - 1 \geq -B^u/2$ , so

$$N_{u-1} = \mathbb{1}[n < 0], \quad Z_{u-1} = \mathbb{1}_L(c_0)\mathbb{1}[n = 0].$$

Thus  $T = \text{sgn}(n)$  if  $n \neq 0$ , and  $T = 1 - \mathbb{1}_L(c_0)$  if  $n = 0$ . These give the three claims.

- $T \in \{0, \text{sgn}(y)\}$ , since  $T = 0$  when  $n = 0$  and  $y \leq 0$ .
- If  $y \leq -2\mathcal{E}/B$ , then  $n < 0$ ; if  $y \geq 2\mathcal{E}/B$ , then  $n > 0$  or  $c_0 \notin L$ ; either way,  $T = \text{sgn}(y)$ .
- If  $|y| < 1/B$ , then  $n = 0$  by Proposition 1 with  $k = 0$  or  $k = B^u$ , so  $T = 0$ .

□

#### A.4 Proof of Proposition 3

*Proof.* The lookups are evaluated on roots of unity, and their exact values,  $\mathbb{1}_H, \mathbb{1}_P, \mathbb{1}_L$  for the digits and  $f(b_j)$  and  $\mathbb{1}_L(c_j)[f(b_j-1) - f(b_j)]$  for the sign, lie in  $\{0, \pm 1\}$ , so the entries of the packed pairs are bounded by a constant  $c$ . If the entries of two composed pairs lie within  $\varepsilon \leq 1$  of their exact values, then  $(g, h) * (g', h') = (g + hg', hh')$  lies within  $(2c+1)\varepsilon + \varepsilon^2 \leq (2c+2)\varepsilon$  of its exact value, so the  $\lceil \log_2 u \rceil$  rounds multiply the lookup error by at most  $u^{O(1)}$ ; a chain of propagate pairs  $(\eta, 1)$  shows that growth linear in  $u$  does occur. Keeping the output within  $\min\{\rho, 1/(4\lambda)\}$  of its exact value, which meets both conditions of Lemma 3 on the initial error, therefore needs  $O(\log u)$  bits, so each of the  $O(\log B) + \lceil \log_2 u \rceil$  levels consumes  $O(\log u) + \log_2 \|e_{\text{rs}}\|_{\text{can}}$  bits, with  $u = \Theta(p/\log B)$ . □

---

**Algorithm 5** Digit functions from two roots of period  $B$ 


---

**Input:**  $\text{ct} = \text{Enc}_Q(\text{Ecd}_{\text{slot}}(z')) \in \mathcal{R}_Q^2$  where  $z' \in \mathbb{R}^{N/2}$  is supported on  $\{m\Lambda : m \in [M]\}$ ,  $z'_{m\Lambda} = z_m \in [0, B^u)$ ; public  $f: \mathbb{Z}_B \rightarrow \mathbb{R}$ .

**Output:** Slot values  $f(\hat{a}_j)$  at position  $j$  of every message, with  $\hat{a}_j$  the digits of  $z_m$  given by Proposition 1.

- 1:  $\text{ct.w} \leftarrow \text{FormWindows}(\text{ct})$  ▷ Algorithm 2
  - 2:  $\text{ct.Rc} \leftarrow \text{Root}_B(B \text{ct.w})$
  - 3:  $\text{ct.c} \leftarrow (\text{KS}_{s \rightarrow \tilde{s}} \circ \text{StC} \circ \text{LUT}_B^{\text{id}})(\text{ct.Rc})$
  - 4:  $\text{ct.b} \leftarrow \text{ct.w} - \text{ct.c}$  ▷ reinterpret at scale  $Q_0/B$
  - 5:  $\text{ct.Rb} \leftarrow \text{Root}_B(\text{ct.b})$
  - 6:  $(\text{ct.H}, \text{ct.P}, \text{ct.f}, \text{ct.d}) \leftarrow (\text{LUT}_B^{\mathbb{1}_H}, \text{LUT}_B^{\mathbb{1}_P}, \text{LUT}_B^f, \text{LUT}_B^{f(\cdot-1)-f})(\text{ct.Rb})$
  - 7:  $\text{ct.L} \leftarrow \text{LUT}_B^{\mathbb{1}_L}(\text{ct.Rc})$
  - 8:  $\text{ct.}\xi \leftarrow \frac{1}{2}(\text{ct.L} \cdot \text{ct.P} + i \text{ct.H})$  ▷ (6)
  - 9:  $\text{ct.d} \leftarrow \text{ct.L} \cdot \text{ct.d}$
  - 10:  $\text{ct.}\xi \leftarrow \text{PrefixScan}(\text{ct.}\xi)$  ▷ Section 3.2
  - 11:  $\text{ct.R} \leftarrow \text{Rot}_{-1}(\text{ct.}\xi)$
  - 12: **return**  $\text{ct.f} + i(\text{Conj}(\text{ct.R}) - \text{ct.R}) \cdot \text{ct.d}$
- 

## B Digit Functions from Two Roots of Period $B$

Algorithm 5 states the route of Remark 2.  $\text{Root}_B$  on the scaled window returns  $\text{ct.Rc}$ , an identity lookup on it and an  $\text{StC}$  return  $c_j$  to the coefficient domain, and subtracting that from the window leaves  $Bb_j$ , whose root is  $\text{ct.Rb}$ .  $\mathbb{1}_H(b_j)$ ,  $\mathbb{1}_P(b_j)$ ,  $f(b_j)$  and  $f(b_j - 1) - f(b_j)$  are four lookups on  $\text{ct.Rb}$ ,  $\mathbb{1}_L(c_j)$  is one lookup on  $\text{ct.Rc}$ , and two multiplications by  $\mathbb{1}_L(c_j)$  form the products.

## C Cleaning with Homomorphic Noise

Let  $\text{Cl}$  be a cleaning polynomial of a constant degree for a finite set  $\mathcal{V}$ , and fix a radius  $\rho > 0$  not exceeding the distance from any  $v \in \mathcal{V}$  to another fixed point of  $\text{Cl}$ . Write  $\eta_i$  for the error of the  $i$ -th iterate as computed and  $\beta_i := -\log_2 |\eta_i|$ .

**Lemma 3.** *There is a constant  $\lambda$ , depending only on  $\text{Cl}$  and  $\rho$ , with  $|\text{Cl}(v + \eta) - v| \leq \lambda \eta^2$  for  $v \in \mathcal{V}$  and  $|\eta| \leq \rho$ . If  $|\eta_0| \leq \rho$ ,  $\beta_0 > \log_2(2\lambda)$ , and the  $i$ -th homomorphic iteration adds noise at most  $\max(\lambda \eta_i^2, 2^{-q-1})$ , then  $\beta_{i+1} \geq \min(2\beta_i - \log_2(2\lambda), q)$ , the least  $k$  with  $\beta_k \geq q$  is  $O(\log q)$ , and evaluating the  $i$ -th iteration at a scale matched to  $\min(\beta_{i+1}, q)$  consumes  $M_{\text{mod}}^{\text{clean}} = \Theta(q)$ .*

*Proof.* Since  $\eta = 0$  is a double root of  $\text{Cl}(v + \eta) - v$ , we have  $\text{Cl}(v + \eta) - v = \eta^2 Q_v(\eta)$  for a polynomial  $Q_v$ , and  $\lambda := \max_{v \in \mathcal{V}} \max_{|\eta| \leq \rho} |Q_v(\eta)|$  gives the first claim. With the noise  $\omega_i$  of the  $i$ -th iteration,  $|\eta_{i+1}| \leq \lambda \eta_i^2 + |\omega_i| \leq \max(2\lambda \eta_i^2, 2^{-q})$ , that is  $\beta_{i+1} \geq \min(2\beta_i - c, q)$  with  $c := \log_2(2\lambda)$ . As  $\beta_0 > c$ , the errors decrease, so this holds at every step, and  $\beta_i - c \geq 2^i(\beta_0 - c)$  below the target gives  $k = O(\log q)$ . The noise bound of the  $i$ -th iteration is met at a scale of  $\min(2\beta_i - \log_2 \lambda, q + 1) + O(1) \leq \min(\beta_{i+1}, q) + O(1)$  bits. Reading the recurrence backward from  $\beta_{k-1} < q$  gives  $\beta_{k-1-t} < q2^{-t} + |c|$ , so these scales sum to  $O(q)$ ; as each iteration has constant depth,  $M_{\text{mod}}^{\text{clean}} = O(q)$ , and the last iteration alone needs a scale of  $\Omega(q)$  bits. For  $\mathcal{V} = \{-1, 0, 1\}$  and  $\text{Cl}(t) = \frac{1}{2}(5t^3 - 3t^5)$ , the fixed points are  $\{0, \pm 1, \pm \sqrt{2/3}\}$ , so  $\rho = 1 - \sqrt{2/3}$ , and  $\lambda = 11$ .  $\square$

## D Modulus Bound for Iterated Polynomials

**Proposition 7.** *For an odd polynomial  $\phi$  with  $|\phi| \leq 1$  on  $[-1, 1]$  and  $\phi'(0) > 0$ , let  $\kappa_\phi := \log_2 \phi'(0) / \lceil \log_2(\deg \phi + 1) \rceil$  be its separation gain per level; then  $\kappa_\phi < 1$ . If a circuit as in Proposition 5 brings the relative separation at the gap boundary up to a constant by iterating  $\phi$ , and  $\phi$  satisfies  $\max_{[-1, 1]} |\phi| = 1$  and  $0 \leq \phi(t) \leq \phi'(0)t$  on  $[0, 1]$ , then*

$$M_{\text{mod}} \geq \frac{p^2}{2\kappa_\phi} + \frac{r'p}{\kappa_\phi} - O(p).$$

*Proof.* Since  $\phi$  is odd and continuous with maximum 1,  $\phi([-1, 1]) = [-1, 1]$ , so each iterate has magnitude 1 and relative separation  $2\phi^{\circ k}(\gamma) \leq 2\phi'(0)^k \gamma$ . Bernstein's inequality at the origin gives  $\phi'(0) \leq \deg \phi < 2^{\delta_\phi}$  with  $\delta_\phi := \lceil \log_2(\deg \phi + 1) \rceil$ , hence  $\kappa_\phi < 1$ . Fix a constant  $\sigma_\star < 2$  that the separation exceeds at the end of the iteration. Then  $2\phi^{\circ k}(\gamma) < \sigma_\star$  for every  $k < K := (p - 1 + \log_2 \sigma_\star) / \log_2 \phi'(0)$ , so the circuit performs the  $(k + 1)$ -th evaluation of  $\phi$ , on  $t = \phi^{\circ k}(x)$ , in at least  $\delta_\phi$  levels of its own, as  $\deg \phi \geq 3$  is odd. At depth  $i$  of this evaluation each value is a polynomial of degree at most  $2^i$  in  $t$ , with the same magnitude over  $t$  as over  $x$ , so Bernstein's inequality on  $|t| \leq \sigma_\star/2$  gives  $\theta_\ell \leq k \log_2 \phi'(0) + i + O(1)$ . The scale inequality summed over these levels is then  $\sum_{k < K} \delta_\phi(r' + p - k \log_2 \phi'(0)) - O(p)$ , which is the bound; the other levels cost at least 0, and the boundary term is as in Proposition 5.  $\square$

All values of  $\kappa_\phi$  below take the depth of a degree- $d$  polynomial to be  $\lceil \log_2(d + 1) \rceil$ ; an evaluation by Paterson–Stockmeyer is deeper, so they are optimistic upper bounds.

**The family  $f_n$ .** NewComp [CKK20] iterates

$$f_n(x) = \sum_{j=0}^n \frac{1}{4^j} \binom{2j}{j} x(1 - x^2)^j, \quad \deg f_n = 2n + 1,$$

whose derivative is  $f'_n(x) = f'_n(0)(1 - x^2)^n$  with  $f'_n(0) = (2n + 1)4^{-n} \binom{2n}{n}$ . Hence  $f_n$  is odd and nondecreasing with  $f_n(\pm 1) = \pm 1$ , and  $f_n(t) = \int_0^t f'_n(0)(1 - s^2)^n ds \in [0, f'_n(0)t]$  for  $t \in [0, 1]$ , so Proposition 7 applies. Its gain  $\kappa_{f_n} = \log_2 f'_n(0) / \lceil \log_2(2n + 2) \rceil$  is 0.4129 at  $n = 7$  and 0.4326 at  $n = 15$ , and  $\kappa_{f_n} < \frac{1}{2}$  for every  $n \geq 1$ : by  $4^{-n} \binom{2n}{n} < 1/\sqrt{\pi n}$  it suffices that  $(2n + 1)^2 < \pi n(2n + 2)$ , that is  $(2\pi - 4)(n^2 + n) > 1$ .

**The family  $g_{n,\tau}$ .** NewCompG [CKK20, Alg. 3] first applies  $d_g$  compositions of a polynomial  $g$  that maps  $[\gamma, 1]$  into  $[1 - \tau, 1]$ , and then  $d_f$  compositions of  $f_n$  that map  $[1 - \tau, 1]$  into  $[1 - 2^{-q}, 1]$ ; for  $\gamma = 2^{-p}$ , Corollary 3 there gives  $d_g = O(p)$  and  $d_f = O(\log q)$ . Their  $g$  satisfies, for some  $0 < \delta < 1$ ,  $x < g(x) \leq 1$  on  $(0, \delta)$  and  $g([\delta, 1]) \subseteq [1 - \tau, 1]$ , and is the limit  $g_{n,\tau}^{\text{conv}}$  of a minimax iteration [CKK20, Alg. 2, Thm. 4], which equioscillates between  $1 - \tau$  and 1 on  $[\delta, 1]$  [CKK20, Lemma 1]; hence  $\max_{[-1, 1]} |g_{n,\tau}^{\text{conv}}| = 1$ , attained at its first maximum  $x_1$ , and  $g > 0$  on  $(0, 1]$ . It is concave on  $[0, x_1]$  [CKK20, Sec. 2.2], so  $g(t) \leq g'(0)t$  there, and for  $t \geq x_1$ ,  $g(t) \leq 1 = g(x_1) \leq g'(0)x_1 \leq g'(0)t$ ; hence Proposition 7 applies to  $g_{n,\tau}^{\text{conv}}$ . For the polynomials  $g_n := g_{n,1/4}$  listed there, whose rounded coefficients put their maxima slightly below 1,  $\kappa_{g_n} = 0.5270, 0.5677, 0.7213, 0.6286$  for  $n = 1, 2, 3, 4$ ; the relation  $g'_n(0) \simeq 0.98 f'_n(0)^2$  observed there for  $n \leq 20$  gives  $\kappa_{g_n} \simeq 2\kappa_{f_n}$  in that range, and Proposition 7 gives  $\kappa_{g_n} < 1$  for every  $n$ .

**Domain-extension polynomials.** [CKP22b] compose scaled copies of a *domain-extension polynomial* (DEP), which is close to the identity near the origin and maps a wide interval into a narrower one. Their DEP of degree  $2m + 1$  is

$$d_{(m)}(x) = \int_0^x \left(1 - \frac{t^2}{\Lambda_m^2}\right)^m dt, \quad \Lambda_m = \frac{2m + 1}{4^m} \binom{2m}{m} = f'_m(0),$$

so  $d_{(m)}(x) = f_m(x/\Lambda_m)$ : the two routes iterate one polynomial in two normalizations. One application keeps the separation of  $\pm\gamma$  to first order and divides the magnitude by  $\Lambda_m$ , so  $\kappa_{d_{(m)}} = \kappa_{f_m}$ , and Proposition 7 applies with the same constant.

## E Use of Generative AI

We used Claude (Opus 5.5) and Codex (GPT-6 Astra) for supporting assistance with manuscript drafting and refinement, pseudocode and figure preparation, algorithm and argument review, and experimental implementation, debugging, and analysis. Inputs included manuscript passages, code, and execution logs. Representative Codex prompts, translated from Korean, included “Could you make the writing more academic?” and “Then please revise it. We should make the comparison fair before benchmarking.” The authors reviewed and revised the resulting material, tested the implementation, checked reported measurements against execution logs, and take full responsibility for the paper.

## References

- [AKP25] Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using CKKS. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 304–337, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.
- [BCC<sup>+</sup>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: 29th Conference on Computer and Communications Security*, pages 223–234, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.
- [BCKS24] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part II*, volume 14652 of *Lecture Notes in Computer Science*, pages 94–123, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.
- [BKSS24] Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part I*, volume 15484 of *Lecture Notes in Computer Science*, pages 330–360, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.
- [BTH22] Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 2022: 20th International Conference on Applied Cryptography and Network Security*, volume 13269 of *Lecture Notes in Computer Science*, pages 521–541, Rome, Italy, June 20–23, 2022. Springer, Cham, Switzerland.
- [CCK<sup>+</sup>25] Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, Jaehyung Kim, Seonghak Kim, Johannes Mono, and Taeyeon Noh. Grafting: Decoupled scale factors and modulus in RNS-CKKS. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, pages 1098–1112, Taipei, Taiwan, October 13–17, 2025. ACM Press.

- [CHK<sup>+</sup>18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.
- [CHK<sup>+</sup>19] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J.: Jacobson, Jr., editors, *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*, volume 11349 of *Lecture Notes in Computer Science*, pages 347–368, Calgary, AB, Canada, August 15–17, 2019. Springer, Cham, Switzerland.
- [CKK20] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 221–256, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.
- [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, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.
- [CKP22a] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park. Efficient homomorphic evaluation on large intervals. *IEEE Transactions on Information Forensics and Security*, 17:2553–2568, 2022.
- [CKP22b] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park. Efficient homomorphic evaluation on large intervals. *Cryptology ePrint Archive*, Paper 2022/280, 2022.
- [CKSS25] Hyeongmin Choe, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Leveraging discrete CKKS to bootstrap in high precision. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, pages 1083–1097, Taipei, Taiwan, October 13–17, 2025. ACM Press.
- [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, Part IV*, volume 16545 of *Lecture Notes in Computer Science*, pages 243–273, Rome, Italy, May 10–14, 2026. Springer, Cham, Switzerland.
- [DMPS24] Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. BLEACH: Cleaning errors in discrete computations over CKKS. *Journal of Cryptology*, 37(1):3, January 2024.
- [GZ26] Mingyu Gao and Hongren Zheng. FHE for SIMD arithmetic logic units with amortized  $O(1)$  bootstrapping per ciphertext. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology – CRYPTO 2026, Part II*, volume 16801 of *Lecture Notes in Computer Science*, pages 299–330, Cham, 2026. Springer Nature Switzerland.

- [HHC19] Kyoohyung Han, Minki Hhan, and Jung Hee Cheon. Improved homomorphic discrete fourier transforms and the bootstrapping. *IEEE Access*, 7:57361–57370, 2019.
- [HLKL26] Deokhwa Hong, Heesoo Lee, Young-Sik Kim, and Yongwoo Lee. A white-box bootstrapping approach for high precision comparison over homomorphic encryption. Cryptology ePrint Archive, Paper 2026/418, 2026.
- [Kim25] Jaehyung Kim. Efficient homomorphic integer computer from CKKS. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):873–898, 2025.
- [Kim26] Jaehyung Kim. Faster homomorphic integer computer. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(3):1128–1151, Jul. 2026.
- [KN25] Jaehyung Kim and Taeyeong Noh. Modular reduction in CKKS. *IACR Communications in Cryptology (CiC)*, 2(2):17, 2025.
- [KS73] Peter M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, C-22(8):786–793, 1973.
- [LLKN21] Eunsang Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. Optimization of homomorphic comparison algorithm on RNS-CKKS scheme. Cryptology ePrint Archive, Report 2021/1215, 2021.
- [LLNK22] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3711–3727, 2022.
- [LMWW26] Jiaming Liu, Shihe Ma, Anyu Wang, and Xiaoyun Wang. High-precision homomorphic ALU over arbitrary moduli with  $O(1)$  bootstrapping. Cryptology ePrint Archive, Paper 2026/1869, 2026.
- [SLWL26] Xuan Shen, Zhihao Li, Ruida Wang, and Xianhui Lu. Efficient homomorphic arithmetic logical units with  $O(1)$  bootstrapping. Cryptology ePrint Archive, Paper 2026/1840, 2026.
- [WMGZ26] Yuchen Wei, Kaisheng Ma, Mingyu Gao, and Hongren Zheng. FHE for ALU over large prime moduli and application to one-round threshold ECDSA. Cryptology ePrint Archive, Paper 2026/1836, 2026.