Low-Latency Parallel Digit Decomposition and Sign Evaluation for CKKS

Anonymous Submission

Abstract. In CKKS, the sign is commonly evaluated by polynomial approximation, and nonlinear functions such as comparison, ReLU, and max are computed through the sign. However, the discontinuity of the sign makes it harder to approximate with a polynomial as the required input bit precision p grows. Alternatively, the sign can be computed by first decomposing the input into radix-B digits. Existing methods extract the $u \approx p/\log B$ digits one at a time, taking u sequential bootstrappings.

Instead, we approximately round u copies of the input to each digit's grid and obtain all u digits in parallel, up to carries, with a single bootstrapping. The carries are then corrected in $O(\log u)$ depth. For the sign, a scan of the same form returns the sign directly, without recovering the exact digits.

We show that our method distinguishes inputs separated by about 2^{-p} with $\Theta(p)$ modulus bits, at the cost of $\Theta(u)$ slots per input. Neither the digits nor the carries need to be computed to high precision, so our circuit, including its bootstrapping, runs at low precision. We also analyze polynomial approximation under an explicit noise and scale model, and show that, in contrast, its modulus consumption grows to $\Omega(p^2)$ bits.

Benchmarks show a 2.04× speedup over [AKP25] for coefficient-encoded integer digit decomposition at ten radix-64 digits and speedups of 5.37× over [LLNK22] and 7.83× over [CKK20] for sign evaluation on real input pairs at 2^{-60} separation. In the tested settings, the method of [AKP25] is faster for $u \leq 4$ and that of [LLNK22] for $p \leq 36$, while our method is faster at the larger tested settings.

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. In CKKS, the sign function is typically evaluated by polynomial approximation. Comparison, ReLU, and max are commonly computed through the sign, so comparing close inputs or computing ReLU and max with high output precision requires the sign in high precision. Since the sign is discontinuous at zero, any polynomial approximation of it is inaccurate on some interval around zero, and higher precision requires this interval to be narrower. Optimized compositions of low-degree polynomials reduce the cost, but the required depth still grows as this interval narrows [CKK20, LLNK22].

Instead of approximating the nonlinear function over a continuous interval, another approach represents the input as an integer with u radix-B digits at the precision required by the application. In CKKS, a lookup is evaluated by an interpolating polynomial, whose degree and required precision grow with the lookup domain size. Thus, rather than a lookup over B^u values, this approach evaluates functions of these digits via small-domain lookups and combines them into the function value. It first requires digit decomposition, which obtains these digits from a ciphertext. Beyond sign evaluation, digit decomposition is a building block of large-integer arithmetic and modular reduction [CPL26, KN25].

However, computing each digit directly would again require a lookup over up to B^u values. [AKP25, KN25] instead extract the least significant digit through a bootstrapping that evaluates a small-domain lookup, removes it, and proceeds to the next. Since each step uses the result of the previous one, extracting u digits takes u sequential bootstrappings. As bootstrapping is generally the most expensive operation in CKKS, the latency is dominated by the u bootstrappings and grows linearly with u.

Our method avoids the sequential extraction by preparing, for each of the u digits, a value that determines it up to a carry. A single bootstrapping processes these values and evaluates multiple lookups on them with a domain size that depends only on B, not u. A carry look-ahead scan on the lookup outputs then resolves the carries, giving every digit or the value of a function at every digit. We also obtain the sign directly, without recovering the exact digits.

1.1 Contributions

Parallel digit decomposition in one bootstrapping. The u radix-B digits of an encrypted input are obtained with one bootstrapping, whereas iterative methods [AKP25, KN25] require u bootstrappings. This removes the u sequential bootstrappings from the latency, at the cost of $\Theta(u)$ slots per input. For coefficient-encoded integer inputs with $(u, B) = (10, 64)$, our method is 2.04× faster than [AKP25].

Ternary sign evaluation without digit recovery. A prefix scan over lookups on two-digit windows of the input returns the sign $T \in \{-1, 0, 1\}$ directly, without recovering the exact digits. By Proposition 2, T is the sign outside an interval around zero and either the sign or 0 inside it. Unlike a polynomial approximation, which can take any value in $[-1, 1]$ on that interval, T always takes one of these three values.

Linear modulus consumption in precision. To decide the sign of inputs as close as 2^{-p} to zero, a polynomial approximation method needs depth $\Theta(p)$ [CKK20]. Under the noise and scale assumptions of Section 4.3, we show that its ciphertext modulus consumption is $\Omega(p^2)$, even when only a few bits of output precision are required. Our method uses $\Theta(p)$ modulus bits for output precision up to about p bits, and the digit circuit itself takes only $O((\log p)^2)$.

1.2 Technical Overview

Obtaining the digit pairs. Write $a_0, \ldots, a_{u-1}$ for the radix-B digits of the input, with a_0 the least significant. A lookup for a_j requires scaling the input to make the integer part modulo B equal to a_j, which turns $a_0, \ldots, a_{j-1}$ into a fractional part. Since a lookup needs its input to be close to an integer, iterative methods [AKP25, KN25] remove $a_0, \ldots, a_{j-1}$ first.

Instead of removing $a_0, \ldots, a_{j-1}$, we use discretization [KN25]. With no need to remove any digit first, we process all u digits in a single bootstrapping, on copies of the input each scaled for a different digit. Discretization rounds the ciphertext so that the value it encrypts lies on a grid of W equally spaced values. The part of the value finer than the grid is rounded off and, together with the discretization error, forms an error that moves the value by an integer number of grid points. Let $\mathcal{E}$ bound the magnitude of this error. A shift of $\mathcal{E}$ added before discretization places the error in $(0, 2\mathcal{E})$. The choice $W = B$ keeps only a_j, so the error adds directly to a_j and cannot be estimated from the discretized value alone. We therefore use $W = B^2$, so that a_j and a_{j-1} are both kept. We call the discretized value for digit j its window w_j, and write $w_j = Bb_j + c_j$ with $0 \le b_j, c_j < B$. We take B large enough that $b_j = a_j + \delta_j \bmod B$ with $\delta_j \in \{0, 1\}$.

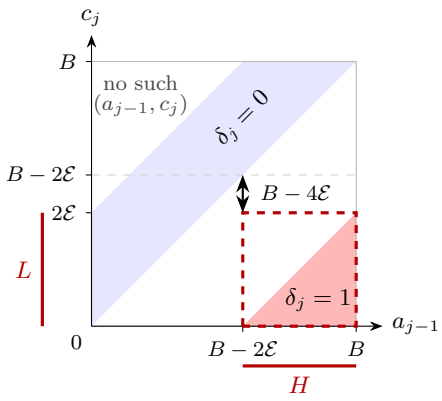

Figure 1: The two carry zones. Only the shaded pairs (a_{j-1}, c_j) occur; the dashed box is the corner triangle widened to the square, and it does not overlap the band on the diagonal when $B > 4\mathcal{E}$.

Figure 1 shows where (a_{j-1}, c_j) lies for $\delta_j = 0$ and $\delta_j = 1$. Let $H = [B - 2\mathcal{E}, B)$ and $L = [0, 2\mathcal{E})$. If $\delta_j = 0$, then (a_{j-1}, c_j) lies in the diagonal band $a_{j-1} < c_j < a_{j-1} + 2\mathcal{E}$; if $\delta_j = 1$, then $a_{j-1} \in H$ and $c_j \in L$. When $B > 4\mathcal{E}$, these two cases do not overlap, so $\delta_j = \mathbb{1}_H(a_{j-1})\mathbb{1}_L(c_j)$ for every j. Thus the carry is a product of two one-variable lookups.

Resolving the carries. Computing $\delta_j = \mathbb{1}_H(a_{j-1})\mathbb{1}_L(c_j)$ needs the true digit a_{j-1}, but w_{j-1} gives only b_{j-1}, which differs from a_{j-1} by δ_{j-1}. Hence δ_j depends on δ_{j-1} through a recurrence $\delta_j = g_j + h_j\delta_{j-1}$, where g_j and h_j depend only on the windows. Following this recurrence one digit at a time would again take $O(u)$ sequential steps. Instead, we compute all carries in $O(\log u)$ rounds by a parallel prefix scan, as in the carry stage of the Kogge–Stone adder [KS73]. Once the carries are known, each digit is recovered as $a_j = b_j - \delta_j \bmod B$. For a digit function f, we obtain $f(a_j)$ as $f(b_j) + \delta_j\,[f(b_j - 1) - f(b_j)]$. Both $f(b_j)$ and $f(b_j - 1)$ are lookups on w_j within the same bootstrapping, whereas evaluating f on the recovered a_j would need a separate lookup after the scan.

Direct sign evaluation. If we only need to determine whether the input is negative, a single function of a_{u-1} suffices. Separating zero from positive inputs, however, depends on all digits. We write both negativity and zero as recurrences and evaluate them in one scan, which yields the sign without computing the carries.

1.3 Related Work

Polynomial methods for comparison and related nonlinear functions. [CKK20] studies efficient homomorphic comparison, and [LLNK22] optimizes composite polynomial approximations to the sign function. When bootstrapping is used, its numerical error can affect the following sign approximation; [HLKL26] studies this interaction explicitly.

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. [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 digit decomposition procedures, higher digits are obtained only after lower

digits have been obtained and removed, making the digit extraction sequential. Our method instead obtains information of all digits at once and combines them using a parallel carry computation.

Conversion for homomorphic arithmetic logic units. [GZ26] introduces an encoding, called triangle encoding, for SIMD word arithmetic, and converts it to a boolean mode with an amortized constant number of bootstrappings per ciphertext. Independently and concurrently with our work, [WMGZ26, SLWL26, LMWW26] improve this conversion and, like our method, obtain all digits with a constant number of bootstrappings. They start from an arithmetic-encoded ciphertext rather than a CKKS ciphertext of a real-valued input. The latter, however, can be transformed into a form that their methods handle, so their methods apply to our setting as well. [SLWL26, LMWW26] cancel the modular overflow of bootstrapping with linear combinations of neighboring digits, computed on the output of a first bootstrapping. This requires a second bootstrapping in sequence. In contrast, our window w_j already keeps a_j and a_{j-1}, so the carry lookups for all digits are evaluated in parallel within one bootstrapping. Moreover, their goal is conversion between encodings, whereas we also evaluate digit functions and the sign without recovering the digits.

2 Preliminaries

2.1 Notations

For a positive integer n, write $[n] = \{0, \dots, n-1\}$, and log denotes the base-2 logarithm. 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 $\lfloor x \rceil = \lfloor x + 1/2 \rfloor$ for rounding to the nearest integer and $[x]_Q := x - Q\lfloor x/Q \rceil$ for the representative of x mod 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. Let $\zeta_W := \exp(2\pi i/W)$ be the primitive W-th root of unity. We write $\mathsf{DFT} : \mathbb{R}[X]/(X^N+1) \to \mathbb{C}^{N/2}$, $v(X) \mapsto \left(v(\zeta_{2N}^{5^i})\right)_{i \in [N/2]}$, for the canonical embedding and iDFT for its inverse, and $\|v\|_{\mathrm{can}} := \|\mathsf{DFT}(v)\|_\infty$ for the canonical norm. Finally, $\mathbb{1}_C(x)$ is the indicator of membership in a set C, $\mathbb{1}[\mathcal{P}]$ the indicator of a predicate $\mathcal{P}$.

2.2 CKKS Basics

CKKS [CKKS17] is a fully homomorphic encryption scheme that encrypts a plaintext polynomial in $\mathcal{R}$.

CKKS encoding. Before encryption, a message is encoded into a plaintext polynomial; we recall two standard encodings, coefficient encoding and slot encoding.

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

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

Slot encoding encodes $\boldsymbol{z} \in \mathbb{C}^{N/2}$ with scaling factor Δ to

$$\mathsf{Ecd}_{\mathsf{slot}}(\boldsymbol{z}; \Delta) := \lfloor \Delta \cdot \mathsf{iDFT}(\boldsymbol{z}) \rceil \in \mathcal{R}.$$

When clear from context, Δ may be omitted. We say an encoded plaintext m has *scale* Δ if $m = \mathsf{Ecd}_{\mathsf{coeff}}(\boldsymbol{z}; \Delta)$ or $m = \mathsf{Ecd}_{\mathsf{slot}}(\boldsymbol{z}; \Delta)$ for some $\boldsymbol{z}$. The homomorphic DFT-like linear transforms StC (slot-to-coefficient) and CtS (coefficient-to-slot) convert a ciphertext between the two encodings.

CKKS encryption. A CKKS ciphertext has the form $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, where Q is called the *ciphertext modulus*. We say that ct encrypts a polynomial $m \in \mathcal{R}$ under a secret key $s \in \mathcal{R}$ if $b + as \equiv m + e \pmod{Q}$ for some small error polynomial $e \in \mathcal{R}$, and write $\mathsf{ct} = \mathsf{Enc}_Q(m)$. We omit the modulus Q when clear from context. The secret key s is typically ternary, that is, its coefficients lie in $\{-1, 0, 1\}$. Its Hamming weight is $\mathsf{hwt}(s) := \|s\|_1$, and a secret key with small Hamming weight is called *sparse*.

CKKS operations. Throughout, all operations are approximate. Given $\mathsf{ct} = \mathsf{Enc}_Q(m)$ where m has scale Δ, rescaling to a smaller modulus Q' outputs $\mathsf{RS}_{Q\to Q'}(\mathsf{ct}) = \mathsf{Enc}_{Q'}(m')$, where m' encodes the same vector as m with scale $\frac{Q'}{Q}\Delta$. With rational rescale [CCK+25], Q' need not divide Q. Given $\mathsf{ct} = \mathsf{Enc}_Q(m)$ under secret s and a switching key from s to s', key switching outputs $\mathsf{KS}_{s\to s'}(\mathsf{ct}) = \mathsf{Enc}_Q(m)$ under secret s'.

For $\mathsf{ct} = \mathsf{Enc}(\mathsf{Ecd}_{\mathsf{slot}}(\boldsymbol{z}))$, rotation by r gives $\mathsf{Rot}_r(\mathsf{ct}) = \mathsf{Enc}(\mathsf{Ecd}_{\mathsf{slot}}(\boldsymbol{z}'))$ with $z'_j = z_{(j+r) \bmod N/2}$, and conjugation gives $\mathsf{Conj}(\mathsf{ct}) = \mathsf{Enc}(\mathsf{Ecd}_{\mathsf{slot}}(\bar{\boldsymbol{z}}))$; each needs one key switching [CHK+18]. Addition requires operands of the same modulus and scale. A multiplication of a ciphertext of scale Δ_1 by a ciphertext or plaintext of scale Δ_2 at modulus Q may be followed by a rescale from Q to a smaller modulus Q', giving scale $\frac{Q'}{Q}\Delta_1\Delta_2$; a ciphertext–ciphertext multiplication needs one key switching, and a plaintext–ciphertext multiplication 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 $\mathsf{ct} \in \mathcal{R}_Q^2$ to a ciphertext $\mathsf{ct}' \in \mathcal{R}_{Q'}^2$ for some $Q' > Q$, encrypting approximately the same message [CHK+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 [BCKS24, BKSS24, AKP25, KN25].

Discretization. Discretization [KN25] rounds the components of a ciphertext so that each coefficient of the plaintext it encrypts represents an element of $\mathbb{Z}_W$.

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

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

Each coefficient is thus rounded to the nearest multiple of Q/W; we call these multiples the *grid*.

For a ciphertext $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, we define discretization of the ciphertext as $\mathsf{Disc}_{W,Q}(\mathsf{ct}) := (\mathsf{Disc}_{W,Q}(a), \mathsf{Disc}_{W,Q}(b))$.

Discretizing the two components does not place the plaintext on the grid exactly: for a ciphertext encrypting m under s, Section 3.4 shows that the plaintext after discretization differs from $\mathsf{Disc}_{W,Q}(m)$ by an integer multiple of Q/W and a small residue, both bounded in terms of $\mathsf{hwt}(s)$.

Functional bootstrapping. For a positive integer W and $f : \mathbb{Z}_W \to \mathbb{C}$, *functional bootstrapping* in CKKS [BKSS24, AKP25] takes a coefficient-encoded ciphertext of $\boldsymbol{z} \in \mathbb{Z}_W^{N/2}$ at modulus Q_0 and scale Q_0/W, and returns $\mathsf{Enc}_Q(\mathsf{Ecd}_{\mathsf{slot}}(f(\boldsymbol{z}); \Delta))$ for some output modulus Q and scale Δ, where f is applied componentwise.

Functional bootstrapping consists of two steps. The first, root production Root_W, maps the input to an encryption of $\zeta_W^{\tilde{\boldsymbol{z}}}$ in the slots. It first raises the ciphertext modulus by

ModRaise, which adds an integer overflow $W\boldsymbol{I}$ to the message, with $\|\boldsymbol{I}\|_\infty$ depending on the Hamming weight of the secret key. It then moves the message to the slots by CtS and evaluates $x \mapsto \zeta_W^x$ by EvalExp, which removes the overflow since $\zeta_W^{\boldsymbol{z}+W\boldsymbol{I}} = \zeta_W^{\boldsymbol{z}}$.

The second, lookup-table evaluation LUT_W^f, evaluates on $\mathsf{Root}_W(\mathsf{ct})$ an interpolant of f at the nodes ζ_W^w, $w \in \mathbb{Z}_W$, yielding $\mathsf{Enc}(\mathsf{Ecd}_{\mathsf{slot}}(f(\boldsymbol{z})))$. 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, chosen as a Hermite interpolant. A complex-valued f is evaluated as its real and imaginary parts, combined by multiplication by i. Evaluating the interpolant takes $\lceil \log W \rceil + O(1)$ levels. All tables evaluated on the same $\mathsf{Root}_W(\mathsf{ct})$ share the powers of the root [AKP25] and differ only in their coefficients. We refer to the composition $\mathsf{LUT}_W^f \circ \mathsf{Root}_W$ as functional bootstrapping for f.

Interpolation near the roots. Root production returns approximate roots rather than the exact interpolation nodes ζ_W^w. Hermite interpolation reduces the effect of a small phase error [AKP25]. For example, a trigonometric interpolant $\tilde{f}$ satisfying $\tilde{f}(w/W) = f(w)$ and $\tilde{f}'(w/W) = 0$ obeys $\tilde{f}(w/W + \eta) = f(w) + O(|\eta|^2)$ for small real η, with a constant depending on $\tilde{f}$.

Iterative digit decomposition. Functional bootstrapping with $W = B$ and $f = \mathrm{id}$ extracts radix-B digits one at a time. Let $\mathsf{ct} = \mathsf{Enc}_{Q_0}(\mathsf{Ecd}_{\mathsf{coeff}}(\boldsymbol{z}; Q_0/B^u))$ with $\boldsymbol{z} = \sum_{j\in[u]} \boldsymbol{a}_j B^j$ and $\boldsymbol{a}_j \in \mathbb{Z}_B^{N/2}$. Multiplying ct by B^{u-1-j} gives a coefficient-encoded ciphertext of $\boldsymbol{z}/B^j \bmod B$ at scale Q_0/B, since $\boldsymbol{a}_{j+1}, \ldots, \boldsymbol{a}_{u-1}$ vanish modulo Q_0. If $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{j-1}$ remain, they form the fractional part of $\boldsymbol{z}/B^j \bmod B$, and $\mathsf{LUT}_B^{\mathrm{id}} \circ \mathsf{Root}_B$ no longer returns $\boldsymbol{a}_j$. The digits are therefore extracted in the order $j = 0, 1, \ldots, u-1$: after $\mathsf{LUT}_B^{\mathrm{id}} \circ \mathsf{Root}_B$ returns $\boldsymbol{a}_j$ in the slots, StC moves it back to the coefficients, and its B^j multiple is subtracted from ct at a matching scale. After these subtractions, the value at step j is exactly $\boldsymbol{a}_j$. Extracting all u digits [AKP25, KN25] thus takes u sequential root productions and lookups, with $u-1$ StC in between.

Cleaning. When the exact values of a computation lie in a finite set $\mathcal{V}$, the CKKS error of its output can be reduced by a *cleaning polynomial* CI with $\mathsf{CI}(v) = v$ and $\mathsf{CI}'(v) = 0$ for every $v \in \mathcal{V}$. Then $\mathsf{CI}(v+\eta) = v + O(|\eta|^2)$ for small η, so each iteration roughly doubles the number of accurate bits once the error is small enough.

3 Digit Decomposition

This section presents our digit decomposition, which obtains all u digits with one root production. The input is a CKKS ciphertext whose slots pack the messages, as produced by ordinary CKKS arithmetic. If the input is already coefficient-encoded, the StC at the start of the circuit is skipped. The output is a public real-valued function of each restored digit of every message, in the slots of one ciphertext; the identity function returns the digits themselves. For simplicity, we assume integer input; real input uses the same circuit, which then returns the digits of an adjacent integer (Proposition 1). Sections 3.1–3.3 describe the procedure, and Section 3.4 bounds the discretization error.

3.1 Obtaining the digit pairs

Write each message as $z = \sum_{j\in[u]} a_j B^j \in [0, B^u)$ with radix-B digits a_j, and put $a_{-1} := 0$. We place the messages $\Lambda := 2^{\lceil \log(u+1) \rceil}$ apart, z_m at slot $m\Lambda$, so that each message has Λ consecutive slots, which we call positions: one per digit, and the rest padding initialized to zero, whose purpose appears in Section 3.2. A single ciphertext then carries $M := N/(2\Lambda)$

Algorithm 1 FormWindows

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

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

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

messages $(z_m)_{m\in[M]}$, assuming $\Lambda \mid N/2$. We number the positions of z_m from -1 to $\Lambda - 2$, so that position j is slot $m\Lambda + 1 + j$; every position outside $0, \ldots, u-1$ is padding. Write $D := \{m\Lambda + 1 + j : m \in [M],\ j \in [u]\}$ for the digit positions. Let $\mathrm{rev}(k)$ denote the reversal of the $\log(N/2)$ bits of $k \in [N/2]$. Since StC is implemented using the fast Fourier transform [HHC19], slot k corresponds to coefficient $\mathrm{rev}(k)$ after StC. After StC, we multiply by the fanout polynomial

$$F(X) := \sum_{j=0}^{u-1} B^{u-1-j} \cdot X^{\mathrm{rev}(1+j)}.$$

Multiplying by B^{u-1-j} moves the pair (a_j, a_{j-1}) into the two most significant digits modulo B^u. We then add the shift polynomial

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

This shifts the error at each digit position into $(0, 2\mathcal{E})$ and sets the high digit at each padding position to 1.

Discretization. After rescaling to the bootstrapping input modulus Q_0, the discretization Disc_{B^2,Q_0} (Definition 1) rounds every coefficient to a multiple of Q_0/B^2. Position j then holds the window w_j at scale Q_0/B^2, up to a residue smaller than Q_0/B^2:

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

Writing

$$\begin{aligned} \delta_j &:= \left\lfloor \frac{a_{j-1} + \mathcal{E} + \varepsilon_j}{B} \right\rfloor \in \{0, 1\}, \\ b_j &= (a_j + \delta_j) \bmod B, \qquad c_j = a_{j-1} + \mathcal{E} + \varepsilon_j - B\delta_j, \end{aligned}$$

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 digit pairs.

Algorithm 1 summarizes the preparation, with $S(X)$ encoded at the indicated coefficient scale; Remark 2 of Section 3.4 explains the switch to a sparser secret $\tilde{s}$ in line 4.

3.2 Carry resolution

Determining the carry. By (1), $a_j \equiv b_j - \delta_j \pmod{B}$, so recovering the digits amounts to determining every δ_j. Assuming $B > 4\mathcal{E}$, δ_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. □

The recurrence. Lemma 1 cannot be used directly: its right side needs the true digit a_{j-1}, but w_{j-1} gives only b_{j-1}. Since $a_{j-1} \equiv b_{j-1} - \delta_{j-1} \pmod B$, we instead express $\mathbb{1}_H(a_{j-1})$ through the indicators of b_{j-1} and the carry δ_{j-1}. Let $P := \{0\}$ be the set of b through which a carry propagates. For $b \in [B]$ and $\delta \in \{0, 1\}$,

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

since $\mathbb{1}_H\big((b-1) \bmod B\big)$ 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 (2) into Lemma 1 gives the following recurrence.

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

$$\delta_j = \mathbb{1}_L(c_j)\,\big(\mathbb{1}_H(b_{j-1}) + \mathbb{1}_P(b_{j-1})\,\delta_{j-1}\big). \tag{3}$$

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 (1), applying (2) leaves only $\mathbb{1}_L(c_j)\,1[b_{j-1} = B - 2\mathcal{E}]\,\delta_{j-1} = 0$ 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. □

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

$$\hat\delta_j := \mathbb{1}_L(c_j)\,\hat\sigma_{j-1}, \qquad \hat\sigma_j := \mathbb{1}_H(b_j) + \mathbb{1}_P(b_j)\,\hat\delta_j \qquad (0 \leq j < u),$$

so that $\hat\delta$ follows (3) and $\hat\sigma_j = G_j + P_j\,\hat\sigma_{j-1}$ with

$$(G_j, P_j) := \big(\mathbb{1}_H(b_j),\ \mathbb{1}_L(c_j)\,\mathbb{1}_P(b_j)\big). \tag{4}$$

Whereas the indicators in (3) involve two windows, both entries of (4) depend only on w_j, so each pair is one lookup on its own slot (Section 3.3).

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 (3).

Real input. For a real message $z \in [0, B^u)$, the windows (1) hold the digits of $\lfloor z \rfloor$, with $a_{-1} := \lfloor B(z - \lfloor z \rfloor) \rfloor$ its first fractional digit.

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 δ_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 $\mu \mapsto g + h\mu$, 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. With $(G_{j:0}, P_{j:0}) := (G_j, P_j) * \cdots * (G_0, P_0)$ and $G_{-1:0} := 0$, this gives $\hat{\sigma}_j = G_{j:0}$ and

$$\hat{\delta}_j = \mathbb{1}_L(c_j)\, G_{j-1:0}, \qquad 0 \le j < u. \tag{5}$$

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 $\lceil \log u \rceil$ rounds. Round r composes each position j with position $j - 2^r$, doubling the block it holds, so after $\lceil \log u \rceil$ rounds every position holds the composite of a block of length $2^{\lceil \log u \rceil} \ge u$.

Packing the pairs. Each position of the carry scan holds a pair of real numbers, which fits in a single complex slot. [CPL26] assigns the complex values i, $\frac{1}{2}$ and 0 to the three carry states: generate, propagate, and neither. We extend that assignment to an arbitrary real pair, so that it also covers the sign recurrence of Section 4.1. We call $\chi(g, h) := \frac{1}{2}(h + i\, g)$, $g, h \in \mathbb{R}$, the *symbol* of the pair. Conjugation recovers g and h as $i(\bar{\chi} - \chi)$ and $\chi + \bar{\chi}$, respectively. Since i is a Gaussian integer, neither recovery consumes a multiplicative level. Expanding both sides shows that the composition $*$ becomes

$$\chi * \chi' = \chi + (\chi + \bar{\chi})\left(\chi' - \tfrac{1}{2}\right). \tag{6}$$

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 (4), and its symbol is

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

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

$$\xi \longmapsto \xi + (\xi + \bar{\xi})\big(\mathsf{Rot}_{-2^r}(\xi) - \tfrac{1}{2}\big),$$

at one rotation, one conjugation and one multiplication per round. We write $\mathsf{PrefixScan}(\mathsf{ct}.\xi)$ for the $\lceil \log u \rceil$ rounds applied to a ciphertext $\mathsf{ct}.\xi$ 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 positions 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, (5) gives it from

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

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

For any $f\colon \mathbb{Z}_B \to \mathbb{R}$, its value at the restored digit is

$$\begin{aligned} f(\hat{a}_j) &= f(b_j) + \hat{\delta}_j\,[f(b_j - 1) - f(b_j)] \\ &= f(b_j) + G_{j-1:0}\, d(w_j), \end{aligned} \tag{7}$$

where $d(w_j) := \mathbb{1}_L(c_j)\,[f(b_j - 1) - f(b_j)]$. With $f = \mathrm{id}$, this returns the digits themselves.

Algorithm 2 Digit functions

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

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

3.3 Root production and lookups

The scan in Section 3.2 and the reconstruction in (7) use the values ξ_j, $f(b_j)$ and $d(w_j)$, all of which are functions of w_j. One call to Root_{B^2} on the window returns $\mathsf{ct.Rw}$, holding $\zeta_{B^2}^{w_j}$ at position j, on which every one of those values is a single B^2-point lookup. Since ξ_j is complex-valued, this makes four lookups on a single root (Section 2.3). Algorithm 2 states the route end-to-end; after the root production, it consumes $\lceil \log B^2 \rceil + O(1)$ levels for the lookups, $\lceil \log u \rceil$ for the scan, and one for line 6.

Remark 1 (B-point lookups on two roots). The values ξ_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. 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 (1) assume $|\varepsilon_j| < \mathcal{E}$, where ε_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* $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$ *decrypt to* $m := [b + as]_Q$ *under a ternary secret* s*, and for* $c \in \mathcal{R}_Q$ *write* $r_{\mathrm{in}}(c)$ *and* $r_{\mathrm{out}}(c)$ *for the remainders left by the inner and the outer rounding of* $\mathsf{Disc}_{W,Q}$*, each bounded by* $\frac{1}{2}$ *in every coefficient. Then* $\mathsf{Disc}_{W,Q}(\mathsf{ct})$ *decrypts to*

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

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

The proof is in Appendix A.2.

Remark 2 (Switching to a sparse secret). The bound of Theorem 1 grows with $\mathrm{hwt}(s)$, so we switch to a sparser secret $\tilde{s}$ immediately before discretization (Algorithm 1, 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 ε_j in (1) is the coefficient of ε at that position plus the difference between $\mathrm{Disc}(m)$ 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, $\mathrm{Disc}(m) = B + \mathcal{E}$, so the same bound on the

coefficient of ε keeps $b = 1$ there. The term e leaves w_j unchanged and perturbs the root's argument by $2\pi\|e\|_\infty/Q_0 \le \pi(\mathrm{hwt}(\tilde{s}) + 2)/Q_0$, negligible since $Q_0 \gg \mathrm{hwt}(\tilde{s})$.

Remark 3 (Realizing the discretization). In [KN25], discretization is realized by switching the ciphertext modulus twice, each switch instantiated as a *rational rescale* [CCK+25]. Rational rescale relies on fast base conversion [CHK+19], whose error raises the error bound to $O(\ell_{\mathrm{RNS}} \cdot \mathsf{hwt}(s))$, where ℓ_{RNS} is the number of RNS blocks the modulus change spans (Theorem 3.2 of [CCK+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 ℓ_{RNS}-dependence.

Remark 4 (Probabilistic bound). Theorem 1 is a worst-case bound. Each coefficient of ε is a sum of $\mathsf{hwt}(\tilde{s}) + 2$ 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 $\mathsf{hwt}(\tilde{s}) + 2$ summands. Under this model, at $\mathrm{hwt}(\tilde{s}) = 32$ and $N = 2^{16}$, a union bound over the coefficients gives $\Pr[\|\varepsilon\|_\infty \ge 14] \le 2^{-56.9}$, so that $|\varepsilon_j| < 15$, whereas the worst case $\lceil(\mathrm{hwt}(\tilde{s}) + 1)/2\rceil = 17$ would require $B > 4 \cdot 19 = 76$.

4 Sign Evaluation with Discretized Inputs

For a normalized real input $x \in [-1, 1]$, we require the output $\widehat{\mathrm{sgn}}(x)$ to satisfy

$$|x| \ge 2^{-p} \quad \Longrightarrow \quad |\widehat{\mathrm{sgn}}(x) - \mathrm{sgn}(x)| \le 2^{-q}.$$

Here p and q are the input and output precision, respectively.

To meet this requirement, we first discretize the input and evaluate the sign from the digit windows of Section 3, without recovering all digits (Section 4.1). This route consumes $\Theta(p)$ modulus bits (Section 4.2), whereas polynomial routes need $\Omega(p^2)$ (Section 4.3).

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. 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 1 yields the windows (1) 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 $\hat{z} := \lfloor z \rfloor + \delta_0$ modulo B^u, so $[\hat{z}]_{B^{j+1}} < 0$ exactly when $\hat{a}_j \ge B/2$. Let $f := \mathbb{1}_{[B/2,B)}$, $N_j := \mathbb{1}\big[[\hat{z}]_{B^{j+1}} < 0\big] = f(\hat{a}_j)$ for $j \ge 0$, and $N_{-1} := 0$. Then for $j \in [u]$,

$$\begin{aligned} N_j &= f(b_j) + \hat{\delta}_j\,[f(b_j - 1) - f(b_j)] \\ &= f(b_j) + N_{j-1}\,\mathbb{1}_L(c_j)\,[f(b_j - 1) - f(b_j)], \end{aligned} \tag{8}$$

where the first line is (7). The second holds since $\hat{\delta}_0 = 0$ and, for $j \ge 1$, $\hat{\delta}_j = \mathbb{1}_L(c_j)\mathbb{1}_H(\hat{a}_{j-1})$; 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 (4), the intercept $f(b_j)$ and the slope $\mathbb{1}_L(c_j)[f(b_j - 1) - f(b_j)]$ of (8) 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 $0, \ldots, j$.

N_{u-1} separates negative inputs from the others, but not zero from positive inputs. Let $Z_j := \mathbb{1}\big[[\hat{z}]_{B^{j+1}} = 0\big]$ for $j \ge 0$ and $Z_{-1} := 1$. By definition, $Z_j = \mathbb{1}[\hat{a}_j = 0]\, Z_{j-1}$, an affine

Algorithm 3 Sign evaluation

Input: $\mathsf{ct} = \mathsf{Enc}_Q(\mathsf{Ecd}_{\mathsf{slot}}(\boldsymbol{z}')) \in \mathcal{R}_Q^2$ where $\boldsymbol{z}' \in \mathbb{R}^{N/2}$ is supported on $\{m\Lambda : m \in [M]\}$ and $z'_{m\Lambda} = y_m$ with $|y_m| \le B^u/2 - 1$.
Output: Slot values T at position $u-1$ of every message.

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

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. Abusing notation, we write Z_j for the resulting recurrence,

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

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 $|y| \le B^u/2 - 1$, and let the windows of $z := y \bmod B^u$ satisfy (1). Then $T \in \{0, \mathrm{sgn}(y)\}$. Moreover, $T = \mathrm{sgn}(y)$ if $|y| \ge 2\mathcal{E}/B$, and $T = 0$ if $|y| < 1/B$.*

The proof is in Appendix A.3. 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 $|y'| \le 2^p$ and $|y' - y| < 1/B$. Then the condition $|y'| \le B^u/2 - 1$ of Proposition 2 holds once $B^u \ge 2^{p+1} + 2$. If $x = x'$ or $|x - x'| \ge 2^{-p}$, then $|y'| < 1/B$ or $|y'| > 1 - 1/B \ge 2\mathcal{E}/B$ with $\mathrm{sgn}(y') = \mathrm{sgn}(y)$, so $T = \mathrm{sgn}(x - x')$ 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

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

Since ψ_j is a function of w_j, a lookup on the root of the windows produces ψ_j at every position, sharing the windows and the root with Algorithm 2. At the padding, $b = 1$ gives $\psi = 0$. To match the initial values $(N_{-1}, Z_{-1}) = (0, 1)$, we add $\frac{1}{2}\mathbf{1}_{\mathrm{pad}}$, where $\mathbf{1}_{\mathrm{pad}} \in \{0,1\}^{N/2}$ is 1 exactly at the padding, so that the padding holds the identity map $\chi(0,1) = \frac{1}{2}$, and then apply $\mathsf{PrefixScan}$. Only position $u-1$ is used, and the composite it holds after the scan covers $2^{\lceil \log u \rceil} \le \Lambda$ positions, which are digit positions of the same message or padding. Writing ψ for the symbol at position $u-1$ after the scan, we have $T = 1 + (2i-1)\psi - (2i+1)\psi$, so no level is consumed after ψ is obtained. Algorithm 3 summarizes the circuit, which uses one Root_{B^2} and one complex-valued lookup, that is, two real lookups.

4.2 Cost of the Discretized Route

The output precision q depends on the application: comparison does not always need high q, while ReLU and $\max(a, b)$, which multiply the sign by its input, need q close to p. We call $(-\gamma, \gamma)$ with $\gamma := 2^{-p}$ the *gap*, and there we require $|\widehat{\mathrm{sgn}}(x)| \le 1$. We run

Algorithm 3 on $y = 2^p x$ as in Section 4.1, with $B > 4\mathcal{E}$, and choose $u = \Theta(p/\log B)$ so that $B^u \geq 2^{p+1} + 2$. By Proposition 2, $T = \mathrm{sgn}(x)$ for $|x| \geq \gamma$ and $T \in \{0, \mathrm{sgn}(x)\} \subset [-1, 1]$ for $|x| < \gamma$. Hence T meets the requirement in exact arithmetic, and cleaning reduces the CKKS error of the evaluated T below 2^{-q}.

We measure the cost by the modulus consumption M_{mod}, assuming that the number of bootstrappings grows linearly with it. The cost has four parts: the input stage, which holds the input at $p + \log B + O(1)$ bits through the fanout and StC before discretization; the root production of Algorithm 3; the lookups on the windows and prefix scan, of depth $O(\log B) + \lceil \log u \rceil$; and the cleaning to q bits. We write $\|e_{\mathrm{rs}}\|_{\mathrm{can}}$ for a bound on the rounding error of one rescaling.

Proposition 3 (Modulus cost of the digit circuit)**.** *The prefix scan can be evaluated with $O(\log u)$ bits of working precision at scale $2^{O(\log u)}\|e_{\mathrm{rs}}\|_{\mathrm{can}}$. For fixed B, the digit circuit consumes*

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

The proof is in Appendix A.4.

Cleaning. For ternary values $t \in \{-1, 0, 1\}$ we can use the cleaning polynomial $\frac{1}{2}(5t^3 - 3t^5)$. 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 $M_{\mathrm{mod}}^{\mathrm{clean}} = \Theta(q)$; Appendix B gives the bound including the homomorphic noise of each iteration.

Proposition 4 (Modulus budget of the discretized route)**.** *Under the window hypothesis (1), the discretized route evaluates the sign with $\gamma = 2^{-p}$ to output precision q with*

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

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 $p + \log B + O(1)$ bits over a constant number of levels; the other two terms are Proposition 3 and the cleaning bound above. □

4.3 Asymptotic Analysis for Polynomial Routes

Polynomial routes evaluate the sign by applying polynomials to the real input directly. To compare it with the discretized route of Section 4.2, we give a lower bound on its modulus consumption. We consider the composite polynomial methods of [CKK20] and their minimax refinement [LLNK22] as the previous state of the art. [CKK20] show that comparing $x, x' \in [0, 1]$ with $|x - x'| \geq \gamma$, which is the sign of $x - x'$ with gap γ, 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 paid for amplifying the gap margin to $\Omega(1)$, while the output precision costs only $\Theta(\log q)$.

We show that the modulus consumption is quadratic in p. Write $\phi_{\ell,1}, \ldots, \phi_{\ell,k_\ell}$ for the polynomials in x that the circuit holds at level ℓ, including values kept from earlier levels, with $\phi_{0,1}(x) := x$ and the output at level ℓ_{out}. We assume that every product is rescaled before it is multiplied again, and that values at level ℓ have scale Δ_ℓ and, for $\ell < \ell_{\mathrm{out}}$, magnitude at most 1. The scales need not be RNS primes, so we assume rescaling to an arbitrary bit-length, as rational rescale [CCK+25] provides.

Since the outputs at $x = \pm\gamma$ must differ by a constant, define the *separation* at level ℓ as $d_{\mathrm{sep},\ell} := \max_i |\phi_{\ell,i}(\gamma) - \phi_{\ell,i}(-\gamma)|$, and write $\theta_\ell := \log(d_{\mathrm{sep},\ell}/d_{\mathrm{sep},0})$ for its cumulative

gain, in bits, up to level ℓ. As in Section 4.2, a ciphertext at scale Δ that has been rescaled at least once may carry an error of size $\|e_{\mathrm{rs}}\|_{\mathrm{can}}/\Delta$ in its decoded value. We require correctness for every noise within these bounds. If the noise at level ℓ can reach half the separation, it can make the inputs $\pm\gamma$ indistinguishable in every value at that level, and no later computation separates them again; hence for $r \geq 1$ bits of margin, $\|e_{\mathrm{rs}}\|_{\mathrm{can}}/\Delta_\ell < 2^{-r} d_{\mathrm{sep},\ell}$. Since $d_{\mathrm{sep},0} = 2\gamma$, we have $d_{\mathrm{sep},\ell} = 2^{\theta_\ell + 1 - p}$, so the scale must satisfy

$$\log \Delta_\ell \geq r' + p - 1 - \theta_\ell, \qquad r' := r + \log \|e_{\mathrm{rs}}\|_{\mathrm{can}}.$$

We allow additional scale adjustments after rescaling and before the next level. A downward adjustment from Δ to $\Delta/2^a$ contributes a bits, and we assume that these contributions sum to $O(p)$ over all levels. Counting only the rescales gives a lower bound on M_{mod}; their total consumption is at least $\sum_{\ell<\ell_{\mathrm{out}}} (2\log\Delta_\ell - \log\Delta_{\ell+1}) - O(p)$.

Proposition 5 (Lower bound for polynomial sign evaluation)**.** *Under the assumptions above, let a circuit of depth ℓ_{out} evaluate the sign with an output precision $q \geq 1$, satisfy $\|e_{\mathrm{rs}}\|_{\mathrm{can}}/\Delta_\ell < 2^{-r} d_{\mathrm{sep},\ell}$ at every level for some $r \geq 1$, and return its output at a scale of $r' + O(p)$ bits. Then $d_{\mathrm{sep},\ell} \leq 2^{\ell+1}\gamma/\sqrt{1-\gamma^2}$ for every $\ell < \ell_{\mathrm{out}}$, $\ell_{\mathrm{out}} \geq p - O(1)$, and in this model, for any scale schedule satisfying this bound,*

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

Proof. Since each $\phi_{\ell,i}(\gamma) - \phi_{\ell,i}(-\gamma)$ is the integral of $\phi'_{\ell,i}$ over $[-\gamma,\gamma]$, and $\phi_{\ell,i}$ has degree at most 2^ℓ, Bernstein's inequality, $|\phi'_{\ell,i}(x)| \leq 2^\ell/\sqrt{1-x^2}$ gives the bound on $d_{\mathrm{sep},\ell}$; at level ℓ_{out}, the output lies within 2^{-q} of ± 1 at $\pm\gamma$, so $d_{\mathrm{sep},\ell_{\mathrm{out}}} \geq 1$, while its magnitude is at most $1 + 2^{-q} \leq 2$, so $\ell_{\mathrm{out}} \geq p - O(1)$. The bound on $d_{\mathrm{sep},\ell}$ gives $\theta_\ell \leq \ell + O(\gamma^2)$, so the scale inequality bounds each $\log\Delta_\ell$ separately, regardless of the scale schedule. Summing the per-level modulus cost gives $M_{\mathrm{mod}} \geq \sum_{\ell<\ell_{\mathrm{out}}} \log\Delta_\ell + \log(\Delta_0/\Delta_{\ell_{\mathrm{out}}}) - O(p)$, whose boundary term is bounded below by $-O(p)$ by the scale inequality at $\ell = 0$ and the bound on the output scale. Since $d_{\mathrm{sep},\ell} \leq 2$, 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. □

Remark 5 (Quadratic modulus consumption with mixed scales). Composite circuits that use multiplication of values at different scales also consume modulus quadratic in p. Consider circuits made of nonlinear polynomial stages of depth bounded independently of p, each taking only the output of the previous stage as input. Since the whole circuit requires $\Omega(p)$ depth, $\Omega(p)$ stages are needed. Inside each stage, suppose that every product is rescaled to a scale that does not exceed the larger scale of its two factors. We also count additional reductions in scale between levels as part of the modulus cost. In a stage with input scale Δ, the output depends on at least one multiplication of two values that depend on the input of the stage. Let $\Delta' \geq \Delta''$ be their scales. Computing the factor at scale Δ'' from the stage input costs at least $\log(\Delta/\Delta'')$ bits. The product has scale $\Delta'\Delta''$, so rescaling it to at most Δ' costs at least $\log\Delta''$ more. Thus, the stage costs at least $\log\Delta$ bits. The scale bound $\log\Delta \geq r' + p - 1 - \theta_\ell$ implies that the first $\Omega(p)$ stages cost $\Omega(p)$ bits each. Subtracting the $O(p)$ cost for additional scale reductions yields $M_{\mathrm{mod}} = \Omega(p^2)$.

Theorem 2 (Asymptotic separation)**.** *Under the noise and cost models of this section, with $1 \leq q = O(p)$, which covers the applications of Section 4.2, 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 (1), 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. Since its constant-degree polynomials have bounded derivatives near $[-1,1]$, input

and evaluation errors accumulate by at most a factor $2^{O(p)}$; we bound the approximation and evaluation errors each by $2^{-(q+1)}$. A scale of $r' + O(p+q) = r' + O(p)$ 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. □

Proposition 6 (Bootstrapping precision)**.** *In a circuit as in Proposition 5, a bootstrap of the values held at level ℓ must keep its added error below $2^{-r} d_{\mathrm{sep},\ell}$, 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 ℓ, including the error of a bootstrap; the second claim follows from Proposition 3. □

A bootstrap preserves at most a bounded absolute precision p_{BTS}, set by its parameters; to preserve more, a high-precision bootstrap [CKSS25] or Meta-BTS [BCC+22] with $\Theta(p/p_{\mathrm{BTS}})$ bootstrappings is needed. A polynomial route thus needs large scales from its first level, and by Proposition 6 any bootstrap it performs at a level where $r+p-1-\theta_\ell > p_{\mathrm{BTS}}$ must be high-precision, whereas the discretized route needs large scales only in its input stage and no high-precision bootstrapping before the cleaning. There the values lie near $\{-1, 0, 1\}$, so a standard bootstrap followed by further cleaning suffices if the last cleaning iterations fit between two bootstrappings.

Remark 6. (Slot usage) Each message occupies $\Lambda = 2^{\lceil \log(u+1) \rceil} = \Theta(u) = \Theta(p/\log B)$ 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 bootstrappings of a polynomial route form a sequential chain that batching cannot eliminate.

5 Implementation and Evaluation

Environment and parameters. We compare our circuits with iterative digit decomposition [AKP25], minimax sign evaluation [LLNK22], and NewCompG's polynomial sign evaluation [CKK20]. All implementations use HEaven 0.3.4 on an Ubuntu 22.04.5 LTS server with two Intel Xeon Silver 4114 CPUs (2.20 GHz) and 251.6 GiB of usable RAM. We use eight CPU threads for all implementations and run benchmarks sequentially. We fix $N = 2^{16}$. Our circuits use $B = 64$, a $B^2 = 4096$ window, one bootstrapping, and post-bootstrap scale 2^{38}, without output cleaning. For input precision p, the sign circuit uses $u = \lceil p/6 \rceil + 1$ digits to satisfy $B^u \geq 2^{p+1} + 2$.

The secrets s and $\tilde{s}$ are fixed-weight ternary secrets with Hamming weights 1024 and 32. Across all tested settings, their evaluation-key moduli satisfy $\log(QP) < 1730$ and $\log(QP) < 234$, respectively, where P is the auxiliary modulus required for key switching. The budget for s follows HEaven's lattice-estimator-based 128-bit target. Our circuits use $\mathcal{E} = 15$ under the probabilistic rounding model of Remark 4.

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 measurements for the method of [AKP25] on the same coefficient-input workload. BTS counts refer to calls to the library's bootstrap routines; their internal branches are not counted separately. Bold values indicate the lowest mean latency in each table row.

NewCompG and the method of [LLNK22] start from a fresh encryption with the post-bootstrap modulus budget. A preceding bootstrap and its noise are excluded, while bootstrappings 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. Every rounded real digit is correct; every sign output has the correct sign. At each setting, we compute output precision from the maximum complex target error over all outputs and trials.

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 $64^{10} = 2^{60}$. These timings exclude initial StC conversion.

Our mean latency stays between 22.12 and 23.58 seconds, while that of [AKP25] increases from 8.20 to 45.39 seconds (Table 1; Figure 2, left). The method of [AKP25] is faster at $u = 2, 4$; Algorithm 2 is faster at $u = 6, 8, 10$, reaching 2.04× at $u = 10$. Our circuit uses one bootstrapping, compared with u sequential bootstrappings for [AKP25].

Table 1: Digit decomposition of one exact coefficient integer.

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

5.2 Sign Evaluation

For $p \in \{12, 18, 24, 30, 36, 42, 48, 54, 60\}$, all three methods evaluate $\mathrm{sgn}(x - x')$ on the same real input pairs per trial, with $x, x' \in [0, 1]$ and $2^{-p} \le |x - x'| \le 1$. The inputs are encrypted in slots; for our circuit, the timings include subtraction and StC. Our input conversion, including StC, preserves the 2^{-p} resolution before discretization. We increase its precision with p and keep our bootstrap parameters fixed.

NewCompG evaluates its final polynomial once. It uses scale 2^{40} for $p \le 24$ and 2^{46} for $p = 30$. For $p \ge 36$, it first uses a scale of about 2^{97} with a 112-bit base modulus (the ciphertext modulus after StC within bootstrapping), then scale 2^{48} with a 57-bit base modulus.

Our implementation of [LLNK22] uses the authors' polynomial composition and evaluation. Products are summed and rescaled before further ciphertext multiplications. We select its parameters in separate runs with a different seed and exclude these runs from the reported means. It uses scale 2^{40} for $p \le 24$, 2^{46} for $p = 30$, and 2^{50} for $p = 36$. For $p \ge 42$, it uses the same scales and base moduli as NewCompG.

Our mean batch latency stays between 16.36 and 17.74 seconds (Table 2; Figure 2, right). The method of [LLNK22] is faster at the tested settings with $p \le 36$, and Algorithm 3 at $p \ge 42$. At $p = 60$, the times are 16.36 and 87.80 seconds, a 5.37× speedup, with one bootstrap for Algorithm 3 and 11 for [LLNK22]. The speedup over NewCompG is 7.83×. Observed output precision is 9.45–10.51 bits for our circuit, 8.05–9.82 bits for [LLNK22], and about 7.62 bits for NewCompG.

Table 2: Sign evaluation on the same real input pairs per trial.

Input precision p	Algorithm 3		[LLNK22]		NewCompG [CKK20]	
	Time (s)	BTS	Time (s)	BTS	Time (s)	BTS
12	16.63 ± 0.73	1	$\mathbf{1.33} \pm 0.05$	0	3.23 ± 0.08	1
18	16.66 ± 0.67	1	$\mathbf{4.04} \pm 0.06$	1	4.83 ± 0.12	1
24	17.74 ± 0.78	1	$\mathbf{5.40} \pm 0.24$	1	12.09 ± 0.62	2
30	17.39 ± 0.69	1	$\mathbf{11.85} \pm 0.32$	2	13.76 ± 0.24	3
36	17.11 ± 0.62	1	$\mathbf{11.59} \pm 0.23$	2	46.07 ± 0.95	8
42	$\mathbf{17.23} \pm 1.05$	1	45.45 ± 0.29	6	74.49 ± 3.31	11
48	$\mathbf{16.95} \pm 0.71$	1	59.53 ± 2.18	8	89.91 ± 1.99	13
54	$\mathbf{17.26} \pm 0.66$	1	77.28 ± 0.48	10	106.10 ± 2.81	15
60	$\mathbf{16.36} \pm 0.40$	1	87.80 ± 1.09	11	128.06 ± 2.76	18

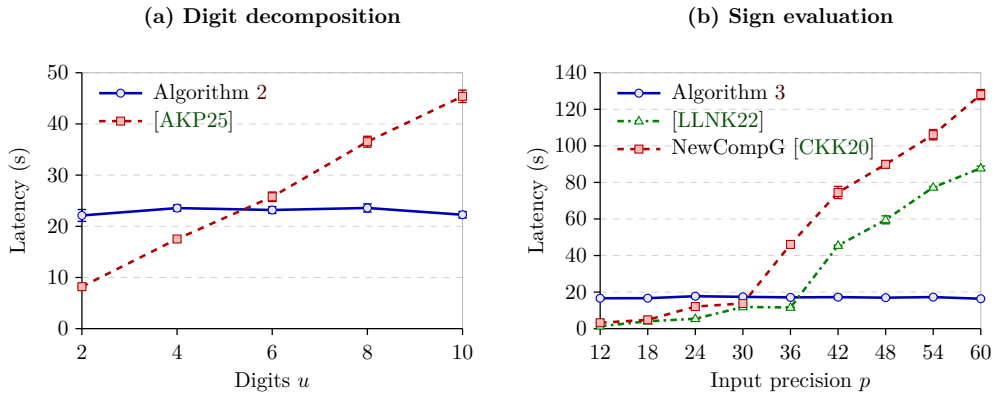

Figure 2: Online latency for (left) extracting all u radix-64 digits from one exact coefficient integer against [AKP25], and (right) sign evaluation on matched real input pairs against [LLNK22] and NewCompG [CKK20]. Points show means and error bars show sample standard deviations over five trials; lines are visual guides. In the sign experiment, the minimum nonzero separation is 2^{-p}, and NewCompG and the method of [LLNK22] use high-precision bootstrapping at the tested settings $p \geq 36$ and $p \geq 42$, respectively.

Modulus consumption. Outside bootstrapping, cumulative Q consumption is about 768 bits for our digit circuit and 6,152 bits for [AKP25] at $u = 10$, and 1,000 bits for our sign circuit, 7,179 bits for [LLNK22], and 11,108 bits for NewCompG at $p = 60$. These are cumulative drops across evaluated segments, not peak Q or QP.

A Omitted Proofs

A.1 Proof of Proposition 1

Proof. Both δ and $\hat{\delta}$ satisfy (3), whose coefficients depend only on b_{j-1} and c_j. Their difference $\nu_j := \delta_j - \hat{\delta}_j$ therefore satisfies $\nu_j = \mathbb{1}_L(c_j)\mathbb{1}_P(b_{j-1})\,\nu_{j-1}$, $\nu_0 = \delta_0$, and $\hat{a}_j \equiv a_j + \nu_j \pmod{B}$. We show by induction that $\nu_j = \delta_0\,\mathbb{1}[a_0 = \cdots = a_{j-1} = B-1]$. 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 δ_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 δ_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 $\hat{a}_j = a_j$. □

A.2 Proof of Theorem 1

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

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

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,

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

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

The outer rounding produces the rest of the error:

$$\begin{aligned} \mathsf{Disc}(b) + \mathsf{Disc}(a)s &= \tfrac{Q}{W}\big(b^* + a^* s\big) + r_{\mathrm{out}}(b) + r_{\mathrm{out}}(a)s \\ &= \mathsf{Disc}(m) + QI + \tfrac{Q}{W}\varepsilon + e, \end{aligned}$$

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

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 (8). 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 (2) 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 $\hat{\delta}_j = \mathbb{1}_L(c_j) N_{j-1}$.

Next we show by induction on j that

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

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}, δ_{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

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

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

- $T \in \{0, \operatorname{sgn}(y)\}$, 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, $T = \operatorname{sgn}(y)$.
- 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 $\lceil \log u \rceil$ 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 $O(\log B) + \lceil \log u \rceil$ levels consumes $O(\log u) + \log \|e_{\mathrm{rs}}\|_{\mathrm{can}}$ bits, with $u = \Theta(p/\log B)$. □

B Cleaning with Homomorphic Noise

Let 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 Cl. Write η_i for the error of the i-th iterate as computed and $\beta_i := -\log|\eta_i|$.

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

Proof. Since $\eta = 0$ is a double root of $\mathsf{Cl}(v+\eta) - v$, we have $\mathsf{Cl}(v+\eta) - v = \eta^2 Q_v(\eta)$ 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 ω_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 $c := \log(2\lambda)$. 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 $\min(2\beta_i - \log\lambda,\, q+1) + O(1) \leq \min(\beta_{i+1}, q) + O(1)$ 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, $M_{\mathrm{mod}}^{\mathrm{clean}} = O(q)$, and the last iteration alone needs a scale of $\Omega(q)$ bits. For $\mathcal{V} = \{-1, 0, 1\}$ and $\mathsf{CI}(t) = \frac{1}{2}(5t^3 - 3t^5)$, the fixed points are $\{0, \pm 1, \pm\sqrt{2/3}\}$, so $\rho = 1 - \sqrt{2/3}$, and $\lambda = 11$. □

C 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. Generative AI was used to prepare the LaTeX code for all tables and figures from the authors' data and specifications. Inputs included manuscript passages, code, and execution logs. Representative Codex prompts 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+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+25] Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, Jaehyung Kim, Seonghak Kim, Johannes Mono, and Taeyeong 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+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+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, 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.

[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.

[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 FHE 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), 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.

[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.