

CKKS Bootstrapping in less than 10ms

Rending Ouyang^{1,2}, Jian Liu^{1,2✉}, Peimin Gao³, Pan Xiao^{1,2}, Heng Zhang^{1,2},
Man Ho Au³, Cong Zhang^{1,2}, Kui Ren^{1,2}, and Chun Chen^{1,2}

¹ The State Key Laboratory of Blockchain and Data Security, Zhejiang University
{oyyy,liujian2411,pan.xiao,heng.zhang,0022090,kui ren,chenc}@zju.edu.cn

² Hangzhou High-Tech Zone (Binjiang) Blockchain and Data Security Research
Institute, Hangzhou, China

³ Department of Computing, The Hong Kong Polytechnic University, Hong Kong

Abstract. Bootstrapping enables deep homomorphic computation in CKKS by replenishing multiplicative levels. It remains one of the scheme’s most expensive operations. HEAAN reports a bootstrapping latency of 6.97ms, substantially faster than other publicly documented alternatives. However, the algorithmic and implementation techniques underlying this result have not been disclosed, leaving it unclear how such performance is achieved.

In this work, we present a fully specified CKKS bootstrapping construction that achieves performance comparable to that reported by HEAAN. Building on SHIP (presented in Eurocrypt 2025), we introduce several algorithmic innovations that reduce communication and computational overhead while enabling efficient parallel execution. Our implementation bootstraps a single complex-valued ciphertext in 8.2ms and achieves an amortized latency of 4.6ms per ciphertext for a batch of 64. These results demonstrate practical sub-10ms CKKS bootstrapping.

1 Introduction

Fully homomorphic encryption (FHE) enables computation on encrypted data. Since Gentry’s first construction [22], increasingly efficient lattice-based schemes have been developed [8,9,16,20,21,23]. Among these, the Cheon-Kim-Kim-Song (CKKS) scheme [15] is particularly well suited to numerical workloads. It supports approximate arithmetic over real or complex numbers and uses *slot encoding* to pack many values into a single Ring-LWE [31] ciphertext, enabling high-throughput SIMD-style evaluation. These capabilities make CKKS a natural candidate for privacy-preserving machine learning, statistics, and scientific computing [32,33,35,36]. As a *leveled* FHE scheme, however, CKKS supports homomorphic computation only up to a prescribed multiplicative depth. *Bootstrapping* overcomes this limitation by restoring the ciphertext modulus and replenishing the multiplicative levels available for further computation [12]. Despite substantial progress [6,11,26,27,30], bootstrapping remains one of the principal sources of latency and parameter growth in CKKS.

✉ Jian Liu is the corresponding author.

Traditional CKKS bootstrapping [11,12] begins with a homomorphic linear transformation from slot encoding to coefficient encoding (S2C). Modulus raising (ModRaise) then lifts the ciphertext to a larger modulus, introducing an unknown multiple of the original modulus into the plaintext. A second homomorphic linear transformation (C2S) converts the ciphertext back to slot encoding, but incurs substantially higher computational cost because it operates at the raised modulus. Finally, approximate modular reduction (EvalMod) removes the additional term introduced by modulus raising through homomorphic evaluation of a periodic-function approximation. Evaluating this approximation requires substantial multiplicative depth, necessitating a large ciphertext modulus and often a larger ring degree to maintain the target security level. The sequential dependencies within this evaluation also limit parallel execution, making bootstrapping a major latency bottleneck despite CKKS’s high SIMD throughput.

SHIP [28] instead performs modular reduction through *decryption in the exponent*, mapping modular additions in the CKKS decryption relation to products of complex *roots-of-unity*. It decomposes the secret-dependent computation into independent *blind-rotations* that operate directly on CKKS slot vectors, avoiding the costly C2S transformation at the raised modulus. For a sparse secret key of Hamming weight h , the resulting contributions are combined using a balanced product tree of depth $O(\log h)$. The multiplicative depth is therefore governed primarily by this tree rather than by a high-degree approximation of modular reduction. The blind rotations also admit parallel evaluation with limited inter-thread communication.

Despite these advances, further reducing the concrete latency of CKKS bootstrapping remains an important challenge. A notable point of reference is the HEAAN library [19], which reports a bootstrapping latency of 6.97 ms, substantially lower than that reported for SHIP. However, the algorithmic and implementation techniques underlying this result have not been publicly disclosed, leaving it unclear how such performance is achieved. This gap motivates the development of a fully specified CKKS bootstrapping construction with latency below 10 ms.

1.1 Our contributions

In this work, we present a fully specified CKKS bootstrapping scheme that achieves performance comparable to that reported by HEAAN. Building on SHIP, we introduce several algorithmic innovations that reduce communication and computational overhead while enabling more efficient parallel execution.

Narrow blind-rotation. We exploit the overlap among the coefficient windows required by candidate rotations to reduce the number of ciphertext-plaintext multiplications. This eliminates redundant computation and lowers the cost of blind-rotation evaluation.

Halving the blind-rotation key. We halve the size of the narrow blind-rotation key by packing paired selectors into the real and imaginary components of CKKS slots. This reduces the key’s memory footprint and alleviates memory pressure during GPU evaluation.

Fusing narrow blind rotation with NTT permutations. We express plaintext slot rotations as fixed NTT-domain permutations, eliminating repeated forward NTTs. The resulting formulation supports efficient batched evaluation across ciphertexts and frequencies, improving GPU utilization and reducing amortized latency.

Packing the blind-rotation key. We develop a compact representation of the blind-rotation key that substantially reduces key-transmission costs. The evaluator reconstructs the evaluation-ready key once and reuses it across subsequent bootstrapping operations.

Low-depth functional bootstrapping (FBS). We introduce a low-depth CKKS FBS framework that directly generates the roots-of-unity required for trigonometric-series evaluation through blind-rotations. This eliminates homomorphic approximation of the complex exponential and ciphertext powering.

Implementation and evaluation. We provide a full-fledged implementation and systematically evaluate its performance on CPUs and GPUs. On an NVIDIA RTX PRO 6000 GPU, our implementation bootstraps a single complex-valued ciphertext in 8.2ms and achieves an amortized latency of 4.6ms per ciphertext for a batch of 64. These results demonstrate practical sub-10ms CKKS bootstrapping.

1.2 Related work

SHIP [28] and PaCo [18] are the closest predecessors to our construction. Both replace a deep approximation of modular reduction with a decryption-in-the-exponent approach based on blind-rotations.

SHIP [28] provides column-based and mux-based methods for blind rotation. The column method enumerates candidate offsets using plaintext-ciphertext products and additions, so its cost and selector payload grow linearly with the candidate range. For fixed radix and gadget-decomposition parameters, the Mux method is asymptotically more attractive: its operation and key counts grow logarithmically. However, it uses more expensive key-switching operations with sequential dependencies, whereas column sums expose inexpensive products and parallel accumulation. SHIP therefore combines the two methods, and its selected hybrid outperforms either pure method.

PaCo [18] performs blind rotations via multiplication by encrypted secret monomials in polynomial rings, followed by a partial C2S transformation that maps the rotated coefficients into CKKS slots. It supports efficient parallel bootstrapping by independently refreshing subsets of plaintext coefficients and recombining the results using inexpensive additions and monomial multiplications. Nevertheless, the procedure still requires partial linear transformations at a large modulus. The published SageMath/NTL implementation is a proof-of-concept rather than a speed-optimized implementation.

Both the hybrid SHIP and PaCo achieves better asymptotic than our scheme. However, a smaller asymptotic count does not necessarily imply lower latency, because different operations have different costs and parallelization opportunities. Mux-based rotations involve sequential key-switching stages, while PaCo requires

additional linear transformations at a large modulus. Our construction instead concentrates blind-rotation work in inexpensive, parallel table-selector products. Narrow tables reduce redundant computation, selector packing lowers key-memory requirements, and NTT fusion enables efficient batched execution. Together, these features favor low concrete latency despite asymptotically linear candidate enumeration.

2 Preliminaries

2.1 Notation

We use $[n]$ to denote the set $\{0, 1, \dots, n-1\}$ and $[a, b]$ to denote the set $\{a, a+1, \dots, b-1\}$. Bold lowercase letters (e.g., $\mathbf{x} \in \mathbb{R}^{1 \times n}$) denote vectors, and bold uppercase letters (e.g., $\mathbf{X} \in \mathbb{R}^{n \times m}$) denote matrices. For a vector $\mathbf{x}$, $\mathbf{x}[i]$ denotes its i -th element. We write $\text{Rot}_r(\mathbf{x})$ for cyclic right rotation of $\mathbf{x}$ by r positions:

$$\text{Rot}_r(\mathbf{x})[i] = \mathbf{x}[(i-r) \bmod n].$$

Table 10 (in Appendix A) summarizes the notations used throughout the paper.

2.2 The CKKS Scheme

The CKKS scheme [15] is an approximate fully homomorphic encryption scheme that supports SIMD arithmetic over packed real or complex values. We briefly review the components of CKKS used in our construction.

CKKS ciphertext. Let

$$\mathcal{R} = \mathbb{Z}[X]/(X^N + 1) \text{ and } \mathcal{R}_Q = \mathcal{R}/Q\mathcal{R} \cong \mathbb{Z}_Q[X]/(X^N + 1).$$

A CKKS ciphertext

$$\text{ct} = \text{Enc}(m) = (a, b) \in \mathcal{R}_Q^2$$

encrypts a message polynomial $m \in \mathcal{R}_Q$ under a secret key $s \in \mathcal{R}$, whose coefficients are typically sampled from $\{0, \pm 1\}$. It satisfies

$$b = -a \cdot s + m + \epsilon \pmod{Q},$$

where $\epsilon \in \mathcal{R}$ is a noise polynomial with small coefficients. Consequently, decryption computes

$$a \cdot s + b = m + \epsilon \pmod{Q}.$$

To efficiently represent a large modulus, RNS-CKKS [13] decomposes

$$Q = \prod_{\ell=0}^L q_\ell$$

into pairwise distinct primes. By the Chinese remainder theorem,

$$\mathcal{R}_Q \cong \mathcal{R}_{q_0} \times \dots \times \mathcal{R}_{q_L}.$$

Arithmetic can therefore be performed independently modulo each q_ℓ .

Slots and coefficients. A CKKS ciphertext can encrypt a plaintext vector $\mathbf{m} \in \mathbb{C}^n$, with $n = N/2$. The vector needs to be encoded into a polynomial $m \in \mathcal{R}_Q$ before encryption. There are two kinds of encoding: *slot encoding* and *coefficient encoding*.

In slot encoding, the conversion between $\mathbb{C}^n$ and $\mathcal{R}_Q$ is facilitated by the discrete Fourier transform (DFT) and its inverse (iDFT):

$$\begin{aligned}\text{Ecd}_{\text{slot}}(\mathbf{m}) &= \lfloor \Delta \cdot \text{iDFT}(\mathbf{m}) \rfloor \in \mathcal{R}_Q, \\ \text{Dcd}_{\text{slot}}(m) &= \frac{1}{\Delta} \text{DFT}(m) \in \mathbb{C}^n.\end{aligned}$$

A scaling factor Δ is used to make the encoded message large relative to the encryption noise and rounding error.

Slot encoding supports SIMD operations. For two vectors $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{C}^n$,

$$\begin{aligned}\text{Ecd}_{\text{slot}}(\mathbf{m}_1) + \text{Ecd}_{\text{slot}}(\mathbf{m}_2) &= \text{Ecd}_{\text{slot}}(\mathbf{m}_1 + \mathbf{m}_2), \\ \text{Rescale}(\text{Ecd}_{\text{slot}}(\mathbf{m}_1) \cdot \text{Ecd}_{\text{slot}}(\mathbf{m}_2)) &= \text{Ecd}_{\text{slot}}(\mathbf{m}_1 \odot \mathbf{m}_2),\end{aligned}$$

where $\odot$ denotes entrywise multiplication.

In coefficient encoding, the real and imaginary components of $\mathbf{m}$ are placed directly in the two halves of the polynomial coefficients:

$$\text{Ecd}_{\text{coef}}(\mathbf{m}) = \sum_{i=0}^{n-1} \text{Re}(\Delta \cdot \mathbf{m}[i])X^i + \sum_{i=0}^{n-1} \text{Im}(\Delta \cdot \mathbf{m}[i])X^{i+n} \in \mathcal{R}_Q,$$

Conversely, for $m(X) = \sum_{i=0}^{N-1} m_i X^i$, coefficient decoding is

$$\mathbf{m} = \text{Dcd}_{\text{coef}}(m) = \Delta^{-1} [m_i + \iota m_{i+n}]_{0 \leq i < n} \in \mathbb{C}^n.$$

CKKS bootstrapping. When successive multiplications are performed over the CKKS ciphertext, the number of RNS moduli keeps decreasing and eventually reaches a minimum value, denoted as q_0 . At this stage, the ciphertext can no longer support further multiplications, and bootstrapping is required to restore the modulus chain and enable subsequent computations. Bootstrapping can be summarized as follows:

S2C. The ciphertext $\text{Enc}(\text{Ecd}_{\text{slot}}(\mathbf{m})) \in \mathcal{R}_{q_0}^2$ is homomorphically transferred into

$$\text{Enc}(m) = \text{Enc}(\text{Ecd}_{\text{coef}}(\mathbf{m})).$$

ModRaise. $\text{Enc}(m) \in \mathcal{R}_{q_0}^2$ is directly turned into $\text{Enc}(m + Iq_0) \in \mathcal{R}_Q^2$.

C2S. The ciphertext is homomorphically transferred into

$$\text{Enc}(\text{Ecd}_{\text{slot}}(\mathbf{m} + q_0 \mathbf{I})) \in \mathcal{R}_Q^2.$$

EvalMod. This operation homomorphically evaluates

$$\text{Enc}(\text{Ecd}_{\text{slot}}(\mathbf{m} + q_0 \mathbf{I})) \text{ modulo } q_0,$$

removing the redundant term $q_0 \mathbf{I}$ and yielding $\text{Enc}(\text{Ecd}_{\text{slot}}(\mathbf{m})) \in \mathcal{R}_Q^2$. The modulo function is typically simulated by the sine function near its zero-crossing points.

2.3 Computing the roots-of-unity via blind-rotation

Let $\text{ct} = (a, b) \in \mathcal{R}_{q_0}^2$ be the coefficient-encoded ciphertext output by S2C. Then,

$$as + b = m + \epsilon.$$

Let $\mathbf{a} = [a_i]_{i \in [N]}$, $\mathbf{b} = [b_i]_{i \in [N]}$, $\mathbf{s} = [s_i]_{i \in [N]}$ and $\text{supp}(\mathbf{s}) = \{j \in [N] : s_j \neq 0\}$. Let $\omega = e^{\iota\eta}$ with $\eta = \frac{2\pi}{q_0}$, hence $\omega^{q_0} = 1$. Then,

$$[\omega^{m_i + \epsilon_i}]_{i \in [n]} = [\omega^{b_i}]_{i \in [n]} \odot \prod_{j \in \text{supp}(\mathbf{s})} [\omega^{a_i - j s_j \alpha_{ij}}]_{i \in [n]}, \quad (1)$$

where $\forall i \in [n]$ and $\forall j \in [N]$

$$\alpha_{i,j} = \begin{cases} 1, & 0 \leq j \leq i, \\ -1, & \text{otherwise.} \end{cases} \quad (2)$$

Define $\mathbf{u}_j \in \mathbb{Z}_{q_0}^N \forall j \in \text{supp}(\mathbf{s})$ s.t.

$$\sum_{i \in [N]} \mathbf{u}_j[i] X^i = s_j X^j \sum_{i \in [N]} a_i X^i \pmod{X^N + 1}.$$

Then,

$$[\omega^{m_i + \epsilon_i}]_{i \in [n]} = [\omega^{b_i}]_{i \in [n]} \odot \prod_{j \in \text{supp}(\mathbf{s})} [\omega^{\mathbf{u}_j[i]}]_{i \in [n]}, \quad (3)$$

For a vector $\mathbf{v} \in \mathbb{Z}_{q_0}^N$, define

$$\begin{aligned} \omega_1(\mathbf{v}) &= [\omega^{v_i}]_{i \in [n]}, & \omega_2(\mathbf{v}) &= [\omega^{v_i}]_{i \in [n, N]}, \\ \pi_1(\mathbf{v}) &= \omega_1(\mathbf{v}), & \pi_2(\mathbf{v}) &= \omega_1(-\mathbf{v}), \\ \pi_3(\mathbf{v}) &= \omega_2(\mathbf{v}), & \pi_4(\mathbf{v}) &= \omega_2(-\mathbf{v}). \end{aligned} \quad (4)$$

Notice that the j -th factor in Equation (3) is precisely $\omega_1(\mathbf{u}_j)$.

Lemma 2.1 (Lemma 1 in SHIP [28]). *For each $j \in [N]$ and $s_j \in \{\pm 1\}$, there exist four binary masks $\{\mathbf{r}_j^{(k)}\}_{k=1}^4$, determined only by (j, s_j) , such that*

$$\omega_1(\mathbf{u}_j) = \sum_{1 \leq k \leq 4} \text{Rot}_j(\pi_k(\mathbf{a})) \odot \mathbf{r}_j^{(k)}. \quad (5)$$

We refer to the original SHIP [28] paper for the proof.

Combining Equations (3) and (5) yields

$$[\omega^{m_i + \epsilon_i}]_{i \in [n]} = [\omega^{b_i}]_{i \in [n]} \odot \prod_{j \in \text{supp}(\mathbf{s})} \left(\sum_{1 \leq k \leq 4} \text{Rot}_j(\pi_k(\mathbf{a})) \odot \mathbf{r}_j^{(k)} \right). \quad (6)$$

It has been proved that the CKKS encryption scheme remains secure even under a structured secret key distribution [18, 28], where the secret key is partitioned

into h blocks, each containing a single randomly located non-zero entry with a value in $\{1, -1\}$. Based on this assumption, the *blind-rotation key* is defined as

$$\text{BRKey} = \left\{ \kappa_{x \cdot g + y}^{(k)} \leftarrow \text{Enc} \left(\text{Ecd}_{\text{slot}} \left(\mathbf{1}_{y=o_x} \odot \mathbf{r}_{x \cdot g + o_x}^{(k)} \right) \right) \right\}_{x \in [h], y \in [g], k \in \{1, 2, 3, 4\}}, \quad (7)$$

where $g = N/h$ is the block size and o_x denotes the non-zero position in the x -th block. Let

$$\beta = \text{Ecd}_{\text{slot}}([\omega^{b_0}, \dots, \omega^{b_{n-1}}])$$

and

$$\gamma = \text{Enc}(\text{Ecd}_{\text{slot}}([\omega^{m_0 + \epsilon_0}, \dots, \omega^{m_{n-1} + \epsilon_{n-1}}]))$$

Then,

$$\gamma = \beta \prod_{x \in [h]} \left(\sum_{y=0}^{g-1} \sum_{k=1}^4 \text{Ecd}_{\text{slot}}(\text{Rot}_{x \cdot g + y}(\pi_k(\mathbf{a}))) \cdot \kappa_{x \cdot g + y}^{(k)} \right). \quad (8)$$

3 Narrow Blind-Rotation

The polynomial a has $N = 2n$ coefficients but a CKKS plaintext has only n slots. SHIP therefore divides the coefficients of a into two n -slot base tables:

$$\pi_1(\mathbf{a}) = [\omega^{a_i}]_{i \in [n]} \text{ and } \pi_3(\mathbf{a}) = [\omega^{a_i}]_{i \in [n, N]}.$$

The other two tables do not store additional coefficients; they reverse the sign of the exponent:

$$\pi_2(\mathbf{a}) = \pi_1(-\mathbf{a}) \text{ and } \pi_4(\mathbf{a}) = \pi_3(-\mathbf{a}).$$

In this section, we show that two *block-specific* tables are sufficient for most output slots, while the few remaining slots can be packed across blocks.

3.1 Block-specific table

In the original SHIP formula (cf. Equation (8)), the candidate position $j = xg + y$ appears as a full rotation $\text{Rot}_{x \cdot g + y}(\pi_k(\mathbf{a}))$: the rotation itself contains both the coarse block offset $x \cdot g$ and the within-block offset y . We reorganize this rotation as $\text{Rot}_y(\pi_{k,x}(\mathbf{a}))$, where

$$\pi_{1,x}(\mathbf{a}) = [\omega^{a_i - x \cdot g}]_{i \in [n]} \text{ and } \pi_{3,x}(\mathbf{a}) = [\omega^{a_i - x \cdot g}]_{i \in [n, N]}.$$

That is, each block-specific table has absorbed the block offset $x \cdot g$. The original formulation requires one plaintext rotation and one forward NTT per block, whereas this reorganization uses two plaintext rotations but still only one forward NTT. The additional rotation acts on plaintext and can be absorbed into table construction through index remapping. It therefore incurs no extra cost.

3.2 Coefficient window

Lemma 3.1 (Coefficient window). *Fix a block x . Over all output slots $i \in [n]$ and all candidate offsets $y \in [g]$, the values $a_{i-(xg+y)}$ belong to the coefficient window*

$$\mathcal{W}_x = \{a_{-(g-1)-xg}, \dots, a_{n-1-xg}\}.$$

In particular, $\mathcal{W}_x$ contains $n + g - 1$ coefficients.

The proof of this lemma is provided in Appendix B.

Lemma 3.1 explains the redundancy in the original SHIP construction. Two complete n -slot tables process $2n$ coefficients per block, although at most $n + g - 1$ coefficients can affect the result. To exploit this overlap, we split the output into

$$I_{\text{head}} = [n - g + 1], \quad I_{\text{tail}} = \{n - g + 1, \dots, n - 1\},$$

hence the head has $n - g + 1$ slots and the tail has $g - 1$ slots.

Lemma 3.2 (Head coefficient window). *Fix a block x . For all head indices $i \in I_{\text{head}}$ and all candidate offsets $y \in [g]$, the coefficients $a_{i-(xg+y)}$ belong to*

$$\mathcal{H}_x = \{a_{-(g-1)-xg}, \dots, a_{n-g-xg}\}.$$

Moreover, $\mathcal{H}_x$ contains n coefficients.

Lemma 3.3 (Tail coefficient window). *Fix a block x . For all tail indices $i \in I_{\text{tail}}$ and all candidate offsets $y \in [g]$, the coefficients $a_{i-(xg+y)}$ belong to*

$$\mathcal{T}_x = \{a_{n-2g+2-xg}, \dots, a_{n-1-xg}\}.$$

Moreover, $\mathcal{T}_x$ contains $2(g - 1)$ coefficients.

The proofs of Lemmas 3.2 and 3.3 are provided in Appendices C and D, respectively. Notice that $\mathcal{W}_x = \mathcal{H}_x \cup \mathcal{T}_x$.

3.3 Computing the head

For each block x , define the *head table* based on $\mathcal{H}_x$:

$$\pi'_x(\mathbf{a}) = [\omega^{a_{i-xg}\alpha_{i,xg}}]_{i \in [n-g+1]} \parallel [\omega^{-a_{n+i-xg}}]_{i \in [n-g+1, n]} \quad (9)$$

When $i \in [n - g + 1]$, the coefficients are $\{a_{-xg}, \dots, a_{n-g-xg}\}$. Set $j = i - n$. When $j = \{-(g - 1), \dots, -1\}$, the coefficients are $\{a_{-(g-1)-xg}, \dots, a_{-1-xg}\}$. Therefore, the two cases together form $\mathcal{H}_x$.

Lemma 3.4 (Rotation of the head table). *For every $x \in [h]$, $y \in [g]$, and $i \in I_{\text{head}}$,*

$$(\text{Rot}_y(\pi'_x(\mathbf{a})))[i] = \omega^{a_{i-(xg+y)}\alpha_{i,xg+y}}.$$

Proof. When $i \geq y$, rotation reads element $i - y$ from the first branch of Equation (9):

$$(\text{Rot}_y(\pi'_x(\mathbf{a})))[i] = \omega^{a_{i-y}-xg} \alpha_{i-y, xg} = \omega^{a_{i-(xg+y)} \alpha_{i-y, xg}}.$$

Furthermore, by definition of α (cf. Equation (2)), both $\alpha_{i-y, xg}$ and $\alpha_{i, xg+y}$ equal 1 when $xg + y \leq i$; otherwise, both equal -1 . Then,

$$\alpha_{i-y, xg} = \alpha_{i, xg+y}.$$

Therefore,

$$(\text{Rot}_y(\pi'_x(\mathbf{a})))[i] = \omega^{a_{i-(xg+y)} \alpha_{i-y, xg}} \text{ when } i \geq y.$$

When $i < y$, rotation wraps and reads position $n + i - y$. Since $y \leq g - 1$, this position lies in the second branch of (9).

$$(\text{Rot}_y(\pi'_x(\mathbf{a})))[i] = \omega^{-a_{(n+i-y)-xg}},$$

which is the negacyclic representation of $\omega^{a_{i-(xg+y)}}$. In this case, $xg + y > i$ and hence $\alpha_{i, xg+y} = -1$. Then,

$$(\text{Rot}_y(\pi'_x(\mathbf{a})))[i] = \omega^{a_{i-(xg+y)} \alpha_{i-y, xg}} \text{ when } i < y.$$

Combining both cases proves the lemma. $\square$

For the positive and negative coefficients respectively, define the following two head tables:

$$\pi'_{1,x}(\mathbf{a}) = \pi'_x(\mathbf{a}), \quad \pi'_{2,x}(\mathbf{a}) = \pi'_x(-\mathbf{a}).$$

For every $x \in [h]$, $y \in [g]$, and $k \in \{1, 2\}$, define

$$b_{x,y}^{(k)} := \begin{cases} 1, & y = o_x, \ k = 1, \ s_{j_x} = 1, \\ 1, & y = o_x, \ k = 2, \ s_{j_x} = -1, \\ 0, & \text{otherwise.} \end{cases}$$

The head-selector mask is

$$\mathbf{r}_{x,y}'^{(k)} := b_{x,y}^{(k)} \mathbf{1}_{n-g+1} \parallel \mathbf{0}_{g-1}.$$

The corresponding encrypted selector is

$$\kappa_{xg+y}'^{(k)} \leftarrow \text{Enc}\left(\text{Ecd}_{\text{slot}}\left(\mathbf{r}_{x,y}'^{(k)}\right)\right).$$

Proposition 3.1 (Head computation). *Let*

$$\beta' = \text{Ecd}_{\text{slot}}\left([\omega^{b_0}, \dots, \omega^{b_{n-g}}] \parallel \mathbf{0}_{g-1}\right),$$

and

$$\gamma' = \beta' \prod_{x \in [h]} \left(\sum_{y=0}^{g-1} \sum_{k=1}^2 \text{Ecd}_{\text{slot}}\left(\text{Rot}_y(\pi'_{k,x}(\mathbf{a}))\right) \cdot \kappa_{xg+y}'^{(k)} \right).$$

Then,

$$\gamma' = \text{Enc}\left(\text{Ecd}_{\text{slot}}\left([\omega^{m_0}, \dots, \omega^{m_{n-g}}] \parallel \mathbf{0}_{g-1}\right)\right).$$

Proof. For block x , recall that its nonzero secret-key position is

$$j_x = xg + o_x.$$

The encrypted selectors remove every term except:

$$y = o_x,$$

and

$$k = \begin{cases} 1, & s_{j_x} = 1, \\ 2, & s_{j_x} = -1. \end{cases}$$

If $s_{j_x} = 1$, the table is $\pi'_{1,x}(\mathbf{a}) = \pi'_x(\mathbf{a})$. Lemma 3.4 gives, for every $i \in I_{\text{head}}$,

$$(\text{Rot}_{o_x}(\pi'_{1,x}(\mathbf{a}))) [i] = \omega^{a_{i-(xg+o_x)}\alpha_{i,xg+o_x}} = \omega^{a_{i-j_x}\alpha_{i,j_x}}.$$

If $s_{j_x} = -1$, the table is $\pi'_{2,x}(\mathbf{a}) = \pi'_x(-\mathbf{a})$, hence

$$(\text{Rot}_{o_x}(\pi'_{2,x}(\mathbf{a}))) [i] = \omega^{-a_{i-j_x}\alpha_{i,j_x}}.$$

Therefore, in either case, the inner sum for block x contributes

$$\omega^{s_{j_x} a_{i-j_x} \alpha_{i,j_x}}$$

to head slot i . Multiplying these contributions over all h blocks and multiplying by the i -th entry ω^{b_i} of β' gives

$$\gamma'[i] = \omega^{b_i} \prod_{x \in [h]} \omega^{s_{j_x} a_{i-j_x} \alpha_{i,j_x}} = \omega^{b_i + \sum_{x \in [h]} s_{j_x} a_{i-j_x} \alpha_{i,j_x}} = \omega^{m_i}.$$

Notice that the last $g-1$ entries of β' are zero. Hence $\gamma'[i] = 0 \ \forall i \in I_{\text{tail}}$. Consequently,

$$\gamma' = \text{Enc}(\text{Ecd}_{\text{slot}}([\omega^{m_0}, \dots, \omega^{m_{n-g}}] \parallel \mathbf{0}_{g-1})).$$

□

3.4 Computing the tail

For each block x , define the tail table based on $\mathcal{T}_x$:

$$\pi''_x(\mathbf{a}) = [\omega^{a_{i-xg}\alpha_{i,xg}}]_{i \in [n-2(g-1), n]}. \quad (10)$$

Lemma 3.5 (Rotation of the tail table). *For every $x \in [h]$, $y \in [g]$, and $i \in I_{\text{tail}}$,*

$$(\text{Rot}_y(\pi''_x(\mathbf{a}))) [i] = \omega^{a_{i-(xg+y)}\alpha_{i,xg+y}}.$$

The proof of this lemma is provided in Appendix E.

For positive and negative secret coefficients, respectively, define

$$\pi''_{1,x}(\mathbf{a}) = \pi''_x(\mathbf{a}), \quad \pi''_{2,x}(\mathbf{a}) = \pi''_x(-\mathbf{a}).$$

A simple way to compute the tail is to process every block separately. Specifically, for each block x and each sign $k \in \{1, 2\}$,

1. place its $2(g-1)$ -entry tail table in the last $2(g-1)$ slots of an n -slot plaintext and fill the preceding slots with zeros;
2. evaluate the g candidate rotations;
3. use the encrypted selectors to retain $y = o_x$ and the signed table corresponding to s_{j_x} ;
4. the two signed tables form one tail blind-rotation and directly produce the contribution of block x on I_{tail} .

This method is simple, but inefficient. For every block, it uses a separate pair of n -slot plaintexts and one blind-rotation even though only $2(g-1)$ slots of each plaintext contain tail-table entries. The unused slots suggest a direct improvement. Since every tail table has only $2(g-1)$ entries, one n -slot plaintext can contain

$$B = \left\lfloor \frac{n}{2(g-1)} \right\rfloor$$

tail tables for each sign. For each pack $q \in [\lceil h/B \rceil]$, let

$$B_q = \min\{B, h - qB\}.$$

Thus pack q contains the blocks $qB, \dots, qB + B_q - 1$. For $k \in \{1, 2\}$, define its packed tail table by

$$\tau_{k,q}(\mathbf{a}) = \pi''_{k,qB}(\mathbf{a}) \parallel \dots \parallel \pi''_{k,qB+B_q-1}(\mathbf{a}) \parallel \mathbf{0}_{n-2B_q(g-1)}. \quad (11)$$

One packed blind rotation now processes the same candidate offsets for all B blocks simultaneously.

The encrypted selectors are packed in the same way as the tables. Specifically, for every $y \in [g]$ and $k \in \{1, 2\}$, define

$$b_{x,y}^{(k)} = \begin{cases} 1, & y = o_x, \ k = 1, \ s_{j_x} = 1, \\ 1, & y = o_x, \ k = 2, \ s_{j_x} = -1, \\ 0, & \text{otherwise.} \end{cases}$$

The mask for block x is

$$\mathbf{r}_{x,y}''^{(k)} = \mathbf{0}_{g-1} \parallel b_{x,y}^{(k)} \mathbf{1}_{g-1}.$$

Following the same order as the packed table, define the packed mask

$$\mathbf{r}_{q,y}''^{(k)} = \mathbf{r}_{qB,y}''^{(k)} \parallel \dots \parallel \mathbf{r}_{qB+B_q-1,y}''^{(k)} \parallel \mathbf{0}_{n-2B_q(g-1)}. \quad (12)$$

The corresponding encrypted selector is

$$\kappa_{q,y}''^{(k)} \leftarrow \text{Enc}\left(\text{Ecd}_{\text{slot}}\left(\mathbf{r}_{q,y}''^{(k)}\right)\right).$$

Proposition 3.2 (Packed tail computation). *For each $q \in [\lceil h/B \rceil]$, let*

$$\zeta_q'' = \sum_{y=0}^{g-1} \sum_{k=1}^2 \text{Ecd}_{\text{slot}}(\text{Rot}_y(\tau_{k,q}(\mathbf{a}))) \cdot \kappa_{qBg+y}''^{(k)}.$$

For each block $x = qB + \ell$ in pack q , its $2(g-1)$ -slot group in ζ_q'' contains

$$\mathbf{0}_{g-1} \parallel [\omega^{s_{jx} a_i - jx \alpha_{i,jx}}]_{i \in I_{\text{tail}}}.$$

The proof of this proposition is provided in Appendix F.

Proposition 3.3 (Recovering the tail from the packed ciphertexts). *Let*

$$\beta'' = \text{Ecd}_{\text{slot}}(\mathbf{0}_{n-g+1} \parallel [\omega^{b_{n-g+1}}, \dots, \omega^{b_{n-1}}]),$$

and

$$\gamma'' = \beta'' \prod_{q=0}^{\lceil h/B \rceil - 1} \prod_{\ell=0}^{B_q - 1} \text{Rot}_{n-2(\ell+1)(g-1)}(\zeta_q''). \quad (13)$$

Then,

$$\gamma'' = \text{Enc}(\text{Ecd}_{\text{slot}}(\mathbf{0}_{n-g+1} \parallel [\omega^{m_{n-g+1}}, \dots, \omega^{m_{n-1}}])).$$

The proof of this proposition is provided in Appendix G.

3.5 Combining the head and tail

Theorem 3.1 (Narrow blind rotation). *Let γ' be the head ciphertext from Proposition 3.1 and let γ'' be the tail ciphertext from Proposition 3.3,*

$$\gamma_{\text{narrow}} = \gamma' + \gamma'' = \text{Enc}(\text{Ecd}_{\text{slot}}([\omega^{m_0 + \epsilon_0}, \dots, \omega^{m_{n-1} + \epsilon_{n-1}}])).$$

The proof of this Theorem is provided in Appendix H.

Table 1. Efficiency comparison between the original SHIP and our narrow version.

Overhead	SHIP	Ours
Ciphertext-plaintext table-selector multiplications	$4gh$	$2g(h + \lceil h/B \rceil)$
Ciphertext rotations for tail alignment	0	at most h
Ciphertext-ciphertext product-tree multiplications	$h-1$	$h + B + \lceil h/B \rceil$
Ciphertext-plaintext multiplications by $\omega^{\mathbf{b}}$	1	2
Blind-rotation key size (in number of ciphertexts)	$4gh$	$2g(h + \lceil h/B \rceil)$

Efficiency analysis. The original SHIP construction evaluates four signed length- n coefficient tables for each block. In contrast, our narrow construction evaluates two length- n head tables for each block and packs the two length- $2(g-1)$ tail tables of up to B blocks together. Hence, the number of signed-table evaluations

decreases from $4h$ to $2(h + \lceil \frac{h}{B} \rceil)$. Table 1 gives the detailed comparison. In particular, the number of table-selector multiplications decreases from $4gh$ to $2g(h + \lceil \frac{h}{B} \rceil)$. The blind-rotation key contains the same number of encrypted selectors, so the key size is reduced by the same factor.

On the other hand, our narrow construction incurs three additional costs in Equation (13):

- For each block, a copy of its packed ciphertext is rotated to move that block’s contribution to I_{tail} , requiring at most h ciphertext rotations.
- The tail computation adds a product-tree evaluation involving $h + B + \lceil h/B \rceil$ ciphertext-ciphertext multiplications.
- The tail computation requires one additional ciphertext-plaintext multiplication by β'' .

For $B = O(h)$, these additions require only $O(h)$ homomorphic operations, whereas our narrow construction saves $O(N)$ ciphertext-plaintext multiplications.

4 Halving the Blind-Rotation Key

Section 3 computes narrow blind rotation using h head factors and $\lceil h/B \rceil$ packed-tail factors. Each factor uses two signed selectors for every candidate offset $y \in [g]$. Thus, its blind-rotation key contains

$$2g \left(h + \left\lceil \frac{h}{B} \right\rceil \right)$$

RLWE ciphertexts. We now halve this already narrow key by placing each pair of real selector vectors in the real and imaginary parts of one ciphertext.

This packing acts directly on the head selectors $\kappa_{xg+y}^{(1)}$, $\kappa_{xg+y}^{(2)}$ and the packed-tail selectors $\kappa_{q,y}^{(1)}$, $\kappa_{q,y}^{(2)}$ defined in Section 3. It does not change their slot layouts or the subsequent head-tail reconstruction. The required public table pairs are conjugates:

$$\pi'_{2,x}(\mathbf{a}) = \overline{\pi'_{1,x}(\mathbf{a})} \text{ and } \tau'_{2,q}(\mathbf{a}) = \overline{\tau'_{1,q}(\mathbf{a})}. \quad (14)$$

The second identity also holds on the zero-padding slots of the packed tail tables.

4.1 Packing the head and tail selectors

Recall the indicator bits $b_{x,y}^{(k)}$ from Section 3, where $k = 1$ and $k = 2$ select the positive and negative secret coefficients, respectively. For every head block $x \in [h]$ and offset $y \in [g]$, generate

$$\tilde{\kappa}'_{xg+y} \leftarrow \text{Enc} \left(\text{Ecd}_{\text{slot}} \left(\left(b_{x,y}^{(1)} + \iota b_{x,y}^{(2)} \right) (\mathbf{1}_{n-g+1} \parallel \mathbf{0}_{g-1}) \right) \right). \quad (15)$$

The real part of its plaintext contains the mask encrypted by $\kappa_{xg+y}^{(1)}$, and the imaginary part contains the mask encrypted by $\kappa_{xg+y}^{(2)}$. Both parts retain the head slots and are zero on the tail.

For every tail pack $q \in [\lceil h/B \rceil]$ and offset $y \in [g]$, use the packed masks $\mathbf{r}_{q,y}^{(k)}$ (cf. (12)) to generate

$$\tilde{\kappa}_{q,y}'' \leftarrow \text{Enc}\left(\text{Ecd}_{\text{slot}}\left(\mathbf{r}_{q,y}^{(1)} + \iota \mathbf{r}_{q,y}^{(2)}\right)\right). \quad (16)$$

The masks keep exactly the same block order and zero slots as in the packed tail computation.

The packed narrow key is therefore

$$\widetilde{\text{BRKey}}_{\text{narrow}} = \{\tilde{\kappa}'_{xg+y}\}_{x \in [h], y \in [g]} \cup \{\tilde{\kappa}''_{q,y}\}_{q \in [\lceil h/B \rceil], y \in [g]}. \quad (17)$$

4.2 Narrow blind rotation with the packed key

Multiplication by a complex-packed selector introduces cross terms. The following identity cancels them using two transformed public tables and one conjugation.

Lemma 4.1 (Packing two real selectors). *Let $z \in \mathbb{C}$ and $k_1, k_2 \in \mathbb{R}$. Let*

$$\tilde{k} = k_1 + \iota k_2, \quad \tilde{\pi}_1 = \frac{z - \iota \bar{z}}{2}, \quad \tilde{\pi}_2 = \frac{\bar{z} - \iota z}{2}.$$

Then

$$\tilde{\pi}_1 \tilde{k} + \overline{\tilde{\pi}_2 \tilde{k}} = zk_1 + \bar{z}k_2. \quad (18)$$

Proof. Expanding the two products gives

$$\begin{aligned} \tilde{\pi}_1 \tilde{k} &= \frac{zk_1 + \bar{z}k_2}{2} + \frac{\iota}{2}(zk_2 - \bar{z}k_1), \\ \overline{\tilde{\pi}_2 \tilde{k}} &= \frac{zk_1 + \bar{z}k_2}{2} - \frac{\iota}{2}(zk_2 - \bar{z}k_1). \end{aligned}$$

Adding them cancels the cross terms and proves the claim. $\square$

Apply this transformation to the head tables:

$$\begin{aligned} \tilde{\pi}'_{1,x}(\mathbf{a}) &= \frac{\pi'_{1,x}(\mathbf{a}) - \iota \pi'_{2,x}(\mathbf{a})}{2}, \\ \tilde{\pi}'_{2,x}(\mathbf{a}) &= \frac{\pi'_{2,x}(\mathbf{a}) - \iota \pi'_{1,x}(\mathbf{a})}{2}, \end{aligned} \quad (19)$$

and to the already packed tail tables:

$$\begin{aligned} \tilde{\tau}_{1,q}(\mathbf{a}) &= \frac{\tau_{1,q}(\mathbf{a}) - \iota \tau_{2,q}(\mathbf{a})}{2}, \\ \tilde{\tau}_{2,q}(\mathbf{a}) &= \frac{\tau_{2,q}(\mathbf{a}) - \iota \tau_{1,q}(\mathbf{a})}{2}. \end{aligned} \quad (20)$$

All these transformations act on public plaintexts and require no homomorphic multiplication.

Proposition 4.1 (Packed evaluation of the narrow factors). *For every head block $x \in [h]$, tail pack $q \in [\lceil h/B \rceil]$, and $t \in \{1, 2\}$, compute*

$$\begin{aligned}\tilde{\zeta}'_{t,x} &= \sum_{y=0}^{g-1} \text{Ecd}_{\text{slot}}(\text{Rot}_y(\tilde{\pi}'_{t,x}(\mathbf{a}))) \cdot \tilde{\kappa}'_{xg+y}, \\ \tilde{\zeta}''_{t,q} &= \sum_{y=0}^{g-1} \text{Ecd}_{\text{slot}}(\text{Rot}_y(\tilde{\tau}_{t,q}(\mathbf{a}))) \cdot \tilde{\kappa}''_{q,y}.\end{aligned}\tag{21}$$

Define the completed factors by

$$\tilde{\zeta}'_x = \tilde{\zeta}'_{1,x} + \text{Conj}(\tilde{\zeta}'_{2,x}), \quad \tilde{\zeta}''_q = \tilde{\zeta}''_{1,q} + \text{Conj}(\tilde{\zeta}''_{2,q}),\tag{22}$$

where Conj applies homomorphic slotwise conjugation to the complete ciphertext, including key switching. Then $\tilde{\zeta}'_x$ and $\tilde{\zeta}''_q$ encrypt the same intended slot vectors as the corresponding head and packed-tail factors of Section 3, respectively:

$$\begin{aligned}\tilde{\zeta}'_x &= \sum_{y=0}^{g-1} \sum_{k=1}^2 \text{Ecd}_{\text{slot}}(\text{Rot}_y(\pi'_{k,x}(\mathbf{a}))) \cdot \kappa'^{(k)}_{xg+y} = \zeta'_x \\ \tilde{\zeta}''_q &= \sum_{y=0}^{g-1} \sum_{k=1}^2 \text{Ecd}_{\text{slot}}(\text{Rot}_y(\tau_{k,q}(\mathbf{a}))) \cdot \kappa''^{(k)}_{q,y} = \zeta''_q.\end{aligned}$$

Proof. Fix a head block x . Since

$$\text{Rot}_y(\pi'_{2,x}(\mathbf{a})) = \overline{\text{Rot}_y(\pi'_{1,x}(\mathbf{a}))},$$

Lemma 4.1 applies slotwise to the two real selector masks. Using additivity of conjugation gives

$$\begin{aligned}\tilde{\zeta}'_x &= \tilde{\zeta}'_{1,x} + \text{Conj}(\tilde{\zeta}'_{2,x}) \\ &= \sum_{y=0}^{g-1} \left(\text{Ecd}_{\text{slot}}(\text{Rot}_y(\tilde{\pi}'_{1,x}(\mathbf{a}))) \cdot \tilde{\kappa}'_{xg+y} + \text{Conj}(\text{Ecd}_{\text{slot}}(\text{Rot}_y(\tilde{\pi}'_{2,x}(\mathbf{a}))) \cdot \tilde{\kappa}'_{xg+y}) \right) \\ &= \sum_{y=0}^{g-1} \sum_{k=1}^2 \text{Ecd}_{\text{slot}}(\text{Rot}_y(\pi'_{k,x}(\mathbf{a}))) \cdot \kappa'^{(k)}_{xg+y}.\end{aligned}$$

For a tail pack q , the tables likewise satisfy

$$\text{Rot}_y(\tau_{2,q}(\mathbf{a})) = \overline{\text{Rot}_y(\tau_{1,q}(\mathbf{a}))}.$$

Applying the same lemma to the paired packed masks therefore yields

$$\begin{aligned}\tilde{\zeta}''_q &= \tilde{\zeta}''_{1,q} + \text{Conj}(\tilde{\zeta}''_{2,q}) \\ &= \sum_{y=0}^{g-1} \sum_{k=1}^2 \text{Ecd}_{\text{slot}}(\text{Rot}_y(\tau_{k,q}(\mathbf{a}))) \cdot \kappa''^{(k)}_{q,y}.\end{aligned}$$

This identity holds in every packed slot, including the guard and padding slots where both masks are zero. Hence both completed factors encrypt the corresponding Section 3 slot vectors. $\square$

By the additivity of conjugation,

$$\begin{aligned}\text{Conj}(\tilde{\zeta}'_{2,x}) &= \sum_{y=0}^{g-1} \text{Conj}(\text{Ecd}_{\text{slot}}(\text{Rot}_y(\tilde{\pi}'_{2,x}(\mathbf{a}))) \cdot \tilde{\kappa}'_{xg+y}) \\ &= \text{Conj}\left(\sum_{y=0}^{g-1} \text{Ecd}_{\text{slot}}(\text{Rot}_y(\tilde{\pi}'_{2,x}(\mathbf{a}))) \cdot \tilde{\kappa}'_{xg+y}\right).\end{aligned}$$

Therefore, in both the head and the tail computation, only one ciphertext conjugation is needed for each block.

Key size and overhead. The packed key contains gh head selectors and $g\lceil h/B \rceil$ tail selectors, giving the reduction

$$2g\left(h + \left\lceil \frac{h}{B} \right\rceil\right) \longrightarrow g\left(h + \left\lceil \frac{h}{B} \right\rceil\right).$$

Each narrow factor still evaluates two transformed tables against the same g packed selectors. Hence, the total number of table-selector multiplications remains $2g(h + \lceil h/B \rceil)$; the arithmetic count is not halved. Nevertheless, jointly processing the two transformed plaintext tables through matrix multiplication can reduce repeated selector loads and improve matrix-tile utilization. The additional work is $h + \lceil h/B \rceil$ ciphertext conjugations, one per head block or tail pack, all sharing one reusable conjugation key.

5 Fusing Narrow Blind-Rotation with NTT Permutations

Recall that Section 4 expresses the head rotation as (cf. Equation (21)):

$$\tilde{\zeta}'_{t,x} = \sum_{y=0}^{g-1} \text{Ecd}_{\text{slot}}(\text{Rot}_y(\tilde{\pi}'_{t,x}(\mathbf{a}))) \cdot \tilde{\kappa}'_{xg+y}. \quad (23)$$

A direct implementation constructs every rotated plaintext in coefficient form and applies a separate forward NTT before the pointwise products. Plaintext rotation is inexpensive, whereas the repeated forward NTTs incur substantial overhead. In this section, we replace these repeated NTTs with cheap NTT-domain permutations, eliminating redundant transformations and enabling GPU-friendly blind-rotation evaluation.

Since the head and tail computations share the same structure, we explain the head computation in detail; the tail follows analogously.

5.1 Narrow Blind-rotation in NTT forms

In Equation (23), each $\tilde{\kappa}'_{xg+y}$ is an RLWE ciphertext consisting of two polynomials (a, b) . Since a and b are processed identically in ciphertext-plaintext multiplication. For simplicity, we only describe one here. Denote $\mathbf{k}_{xg+y} \in \mathbb{Z}_Q^{N \times 1}$, $\mathbf{u}_{x,y}^{(1)} \in \mathbb{Z}_Q^{N \times 1}$ and $\mathbf{u}_{x,y}^{(2)} \in \mathbb{Z}_Q^{N \times 1}$ by the NTT forms of $\tilde{\kappa}'_{xg+y}$, $\text{Rot}_y(\tilde{\pi}'_{1,x}(\mathbf{a}))$ and $\text{Rot}_y(\tilde{\pi}'_{2,x}(\mathbf{a}))$ respectively. Then, we could re-write Equation (23) as:

$$\mathbf{z}_{t,x} = \sum_{y=0}^{g-1} \left(\mathbf{k}_{xg+y} \odot \mathbf{u}_{x,y}^{(t)} \right). \quad (24)$$

5.2 Fusing with NTT permutations

CKKS implements slot rotations using ring automorphisms [15]. In the RNS-NTT representation [13], these automorphisms act as permutations, yielding the identities below, which we use to reorganize narrow blind rotation.

Lemma 5.1 (Slot rotations as NTT permutations). *Let $m_i(X) = \text{Ecd}_{\text{slot}}(\text{Rot}_i(\mathbf{v}))$ $\forall i \in [n]$. For each RNS prime $q_\ell \equiv 1 \pmod{2N}$, there exists a permutation matrix $\mathbf{P}_i$ such that*

$$\text{NTT}_{q_\ell}(m_i) = \mathbf{P}_i \cdot \text{NTT}_{q_\ell}(m_0).$$

Proof. First,

$$m_i(X) = m_0(X^{5^{-i}}) \pmod{X^N + 1}.$$

The substitution in the above equation has the required effect on the CKKS slots:

$$m_i(\zeta^{5^j}) = m_0(\zeta^{5^{j-i}}),$$

which reads slot $(j - i) \bmod n$ of the original plaintext.

Fix an RNS prime q_ℓ and a primitive $2N$ -th root ξ_ℓ modulo q_ℓ . In logical NTT order, the k -th entry is the polynomial evaluated at ξ_ℓ^{2k+1} . Therefore,

$$[\text{NTT}_{q_\ell}(m_i)][k] = m_i(\xi_\ell^{2k+1}) = m_0(\xi_\ell^{5^{-i}(2k+1)}).$$

Since 5^{-i} is invertible modulo $2N$, multiplying by it permutes the odd exponents $1, 3, \dots, 2N - 1$. Thus, for each $k \in [N]$, there is a unique index $\rho_i(k) \in [N]$ satisfying

$$2\rho_i(k) + 1 \equiv 5^{-i}(2k + 1) \pmod{2N}.$$

Consequently,

$$[\text{NTT}_{q_\ell}(m_i)][k] = [\text{NTT}_{q_\ell}(m_0)][\rho_i(k)].$$

Define $\mathbf{P}_i$ by $(\mathbf{P}_i \mathbf{u})[k] = \mathbf{u}[\rho_i(k)]$, which confirms the proof.

The index map uses only arithmetic modulo $2N$, so it is common to all RNS primes. A fixed storage order, such as bit-reversed order, merely reindexes this same permutation. $\square$

By lemma (5.1), each $\mathbf{u}_{x,y}^{(t)}$ in Equation (24) can be written as $\mathbf{u}_{x,y}^{(t)} = \mathbf{P}_y \cdot \mathbf{u}_{x,0}^{(t)}$. Then, we could re-write Equation (24) as:

$$\mathbf{z}_{t,x} = \sum_{y=0}^{g-1} \left(\text{diag}(\mathbf{k}_{xg+y}) \cdot \mathbf{P}_y \cdot \mathbf{u}_{x,0}^{(t)} \right).$$

Let

$$\mathbf{W}_x = \sum_{y=0}^{g-1} (\text{diag}(\mathbf{k}_{xg+y}) \cdot \mathbf{P}_y), \quad (25)$$

then,

$$\mathbf{z}_{t,x} = \mathbf{W}_x \cdot \mathbf{u}_{x,0}^{(t)}.$$

This transformation effectively eliminates the need to apply a separate forward NTT to every rotated plaintext. A direct evaluation of the head computation requires $2gh$ forward NTTs, one for each $x \in [h]$, $y \in [g]$, and $k \in \{1, 2\}$. In contrast, our formulation requires only $2h$ forward NTTs, one for each base table $\tilde{\pi}'_{t,x}(\mathbf{a})$; all rotated variants are obtained through fixed NTT-domain permutations.

The head blind-rotation key $\{\tilde{\kappa}'_{xg+y}\}_{x \in [h], y \in [g]}$ is reorganized into the structured sparse operators $\{\mathbf{W}_x\}_{x \in [h]}$. Notice that each $\mathbf{W}_x$ is the sum of g diagonal-permutation matrices and therefore contains at most gN nonzero values. These values are precisely the NTT coefficients already stored in the corresponding g selector components. Since the permutation positions are fixed and need not be stored explicitly, this transformation changes only the layout of the blind-rotation key, not its coefficient payload or asymptotic size.

Furthermore, we could combine $\mathbf{u}_{x,0}^{(t)}$ from different blind-rotation instances into a matrix:

$$\mathbf{U}_{x,0}^{(t)} \in \mathbb{Z}_Q^{N \times t}.$$

Then, we could compute

$$\mathbf{Z}_x = \mathbf{W}_x \cdot \mathbf{U}_{x,0}^{(t)}.$$

which turns the dominant part of blind-rotation into a sparse matrix-matrix multiplication mapping naturally onto GPUs.

6 Packing the Blind-Rotation Key

Our narrow blind-rotation key (in Section 4) consists of $g \left(h + \lceil \frac{h}{B} \rceil \right)$ ciphertexts, which still introduces a large communication overhead for transmission. In this section, we adopt a key packing method [3] that can pack the blind-rotation key into four ciphertexts.

6.1 Packing the indicator bits

Recall that the original blind-rotation key is

$$\text{BRKey} = \left\{ \kappa_{x \cdot g + y}^{(k)} \leftarrow \text{Enc} \left(\text{Ecd}_{\text{slot}} \left(\mathbf{1}_{y=o_x} \odot \mathbf{r}_{x \cdot g + o_x}^{(k)} \right) \right) \right\}_{x \in [h], y \in [g], k \in \{1, 2, 3, 4\}}.$$

Applying the technique in Section 4, we obtain,

$$\widetilde{\text{BRKey}} = \left\{ \tilde{\kappa}_{xg+y}^{(k)} \leftarrow \text{Enc} \left(\text{Ecd}_{\text{slot}} \left(\left(b_{x,y}^{(k)} + \iota b_{x,y}^{(k+1)} \right) \mathbf{1}_n \right) \right) \right\}_{x \in [h], y \in [g], k \in \{1, 3\}},$$

which consists of $2gh = 2N$ encrypted selectors. Observe that these selectors are determined by the secret indicator bits:

$$b_{x,y}^{(k)} = \begin{cases} 1, & y = o_x, k = 1, s_{j_x} = 1, \\ 1, & y = o_x, k = 2, s_{j_x} = -1, \\ 0, & \text{otherwise.} \end{cases}$$

Although the blind-rotation key contains many ciphertexts, its secret-dependent information is the collection

$$\left\{ b_{x,y}^{(k)} \right\}_{x \in [h], y \in [g], k \in \{1, 2, 3, 4\}}$$

which contains $4N$ bits. We denote these indicator bits by

$$\{b_{i \cdot N + j}\}_{i \in [4], j \in [N]}.$$

For each $i \in [4]$, we construct

$$p_i(X) = \sum_{j=0}^{N-1} b_{i \cdot N + j} X^j.$$

Then, the packed blind-rotation key is

$$\text{BRKey}_{\text{pack}} = \left\{ \text{ct}_i \leftarrow \text{Enc}(p_i(X)) \right\}_{i \in [4]}.$$

6.2 Recovering the indicator bits

Let $N = 2^\ell$, and let $\text{Sub}(\text{ct}, k)$ denote the automorphism $X \mapsto X^k$. For each $i \in [4]$, we decompress ct_i using a binary tree: at depth $r \in \{0, \dots, \ell - 1\}$, each current node ct is replaced by two children:

$$\text{ct}^{(0)} = \text{ct} + \text{ct}' \quad \text{and} \quad \text{ct}^{(1)} = X^{-2^r}(\text{ct} - \text{ct}'), \quad \text{where} \quad \text{ct}' = \text{Sub} \left(\text{ct}, 1 + \frac{N}{2^r} \right).$$

To explain this decomposition, suppose that the current node encrypts

$$f(X) = \sum_j b_j X^{2^r j}.$$

Since

$$\left(X^{2^r j}\right)^{1+N/2^r} = X^{2^r j + jN} = (-1)^j X^{2^r j} \pmod{X^N + 1},$$

the sum $\mathbf{ct} + \mathbf{ct}'$ retains the even-indexed coefficients, whereas the difference $\mathbf{ct} - \mathbf{ct}'$ retains the odd-indexed coefficients. Multiplication by X^{-2^r} shifts the odd-indexed coefficients so that the same procedure can be applied at the next depth. After ℓ rounds, for every $j \in [N]$, one leaf encrypts

$$N \cdot b_{i \cdot N + j}$$

as a constant polynomial, up to a fixed public ordering. Multiplying each leaf by the public scalar N^{-1} therefore recovers a ciphertext encrypting

$$b_{i \cdot N + j} \mathbf{1}_{N/2}.$$

Each decompression tree contains $N - 1$ internal nodes and hence requires $N - 1$ substitutions and key switchings. Notice that these three trees share the same $\ell = \log_2 N$ reusable key-switching keys.

6.3 Reconstructing the encrypted selectors for narrow blind-rotation

The decompressed leaves contain replicated bits, whereas $\kappa'_{xg+y}{}^{(k)}$ and $\kappa''_{q,y}{}^{(k)}$ have the slot layouts defined in Section 3. The evaluator therefore applies the corresponding public slot masks after decompression. In particular, the head selector is formed by placing $b_{x,y}^{(k)}$ in I_{head} and zero in I_{tail} . For a tail pack q , the indicators

$$b_{qB,y}^{(k)}, \dots, b_{qB+B_q-1,y}^{(k)}$$

are placed in the last $g - 1$ slots assigned to their respective blocks, producing $\kappa''_{q,y}{}^{(k)}$. At this point, the head selectors $\kappa'_{xg+y}{}^{(1)}, \kappa'_{xg+y}{}^{(2)}$ and tail selectors $\kappa''_{q,y}{}^{(1)}, \kappa''_{q,y}{}^{(2)}$ have been recovered. The complex-packed selectors required by Section 4 are then obtained as

$$\tilde{\kappa}'_{xg+y} = \kappa'_{xg+y}{}^{(1)} + \iota \kappa'_{xg+y}{}^{(2)}, \quad \tilde{\kappa}''_{q,y} = \kappa''_{q,y}{}^{(1)} + \iota \kappa''_{q,y}{}^{(2)}.$$

Together, these selectors form the packed narrow blind-rotation key $\widetilde{\text{BRKey}}_{\text{narrow}}$ defined in Equation (17). The required head and packed-tail selectors in the matrix form (cf. Equation (25)) must still be reconstructed.

Consequently, coefficient packing reduces the persistent key from the $g(h + \lceil h/B \rceil)$ evaluation-ready selector ciphertexts to four coefficient-packed ciphertexts, together with the shared substitution keys.

7 Low-depth Functional Bootstrapping

Functional bootstrapping (FBS) extends bootstrapping by simultaneously refreshing the ciphertext modulus and evaluating a non-linear target function f_T (e.g.,

GELU, Tanh, Sigmoid, or a lookup table), effectively providing function evaluation “for free”. In CKKS, recent works realize FBS by modifying the `EvalMod` stage into `EvalSeries`, where a periodic extension of f_T is approximated by a trigonometric series and evaluated homomorphically [2,5,34]. Specifically, f_T is approximated by a degree- d trigonometric polynomial

$$p(x) = a_0 + \sum_{k=1}^d (a_k \cos(k\eta x) + b_k \sin(k\eta x)) = \sum_{k=-d}^d c_k \omega^{kx},$$

where $\eta = \frac{2\pi}{q}$ for a period q , and $\omega = e^{\iota\eta}$ with $\iota = \sqrt{-1}$.

Although carefully designed trigonometric Remez approximants can reduce d for a fixed L_∞ error [34], existing approaches still incur substantial multiplicative degree overhead: generating the encrypted roots-of-unity ($z = \omega^x$) requires complex exponential approximations, and the subsequent computation of powers ($z^k \forall k \in [-d, d]$) further inflates the degree.

With the roots-of-unity computed by blind-rotation, we could have a CKKS FBS paradigm that eliminates exponential approximation and power computation. We depict the FBS pipeline as follows:

1. Run trigonometric Remez offline to obtain a minimax approximant for the target function f_T :

$$p(x) = \sum_{k=-d}^d c_k \omega^{kx}.$$

2. Given a ciphertext $\text{Enc}(\text{Ecd}_{\text{slot}}(\mathbf{m})) \in \mathcal{R}_{q_0}^2$, run `S2C` and `ModRaise` to obtain $\text{Enc}(m + Iq_0) \in \mathcal{R}_Q^2$.
3. Run blind-rotation to obtain $\text{Enc}(\text{Ecd}_{\text{slot}}(\omega^m)) \in \mathcal{R}_Q^2$.
4. Evaluate $p(m)$ by homomorphic powering and additions.

For the real-valued target functions considered here, the trigonometric coefficients satisfy $c_{-k} = \overline{c_k}$. Moreover, since m is real, $\omega^{-km} = \overline{\omega^{km}}$. Hence,

$$p(m) = c_0 + \sum_{k=1}^d (c_k \omega^{km} + \overline{c_k \omega^{km}}).$$

The zero-frequency term c_0 is a public constant, and the negative-frequency terms are obtained by conjugating the positive-frequency terms. We therefore need only d blind-rotation instances, rather than $2d + 1$.

For each $k \in \{1, \dots, d\}$, replace ω by ω^k in Step 3 and fold c_k into the public factor $(\omega^{kb_i})_{i \in [n]}$. This produces $\text{Enc}(\text{Ecd}_{\text{slot}}(c_k \omega^{km}))$ directly. Step 4 then requires only conjugations and additions, without ciphertext powering.

These d instances share the same blind-rotation key and can be evaluated together using the matrix formulation of Section 5.2. If one additional multiplication level is available, we could follow SWIFT [14] and generate only a subset $\mathcal{A}$ of all frequency basis. In this variant, we first compute the unweighted terms

z^a , where $z = \omega^m$ and $a \in \mathcal{A}$. Every required positive frequency is either in $\mathcal{A}$ or expressible as $a + b$ with $a, b \in \mathcal{A}$, allowing reconstruction through $z^{a+b} = z^a z^b$. Negative frequencies follow by conjugation, and the coefficients c_k are applied after reconstruction. Thus, the blind-rotation batch width decreases from d to $|\mathcal{A}|$. Selecting a smallest such basis is an order-two instance of the Global Postage-Stamp Problem (GPSP) [10,29]; we use the GPSP-based approach described in APSI [17], resulting in

$$\sqrt{\frac{d}{0.4585}} < |\mathcal{A}| < \sqrt{\frac{d}{0.2891}}.$$

8 Performance Evaluation

8.1 Implementation

We implemented our scheme based on Microsoft SEAL [1], using AVX2 instructions and OpenMP for CPU parallelism. Following SHIP [28], ciphertexts are initially encrypted under a dense ternary secret. After S2C, we switch at the lowest modulus level to a sparse secret [7] with Hamming weight $h = 31$. We also retain SHIP’s S2C implementation, which evaluates the ciphertext-plaintext matrix multiplication using the diagonal method with baby-step giant-step optimization [24,25] rather than a Cooley-Tukey decomposition. Keeping S2C unchanged makes its cost directly comparable to SHIP and isolates the effect of our blind-rotation optimizations.

We follow the parameter sets used by SHIP: LL13, LL14, HT14 and HT15, summarized in Table 2. The LL configurations target low latency and retain one multiplicative level after bootstrapping, whereas the HT configurations target high throughput and retain multiple levels. LL13 and HT14 target approximately 4 bits of precision, while LL14 and HT15 target more than 16 bits. When bootstrapping requires relatively few key-switching operations, we can reallocate part of the modulus budget from key switching to the product tree, thereby improving precision. We report precision as $\log_2(\epsilon^{-1})$, where ϵ is the maximum absolute error over all output slots. All four parameter sets satisfy the standard 128-bit security target.

We also implement a CUDA backend for our scheme, combining custom kernels with operators from EasyFHE [37]. To accelerate blind-rotations in sparse matrix form (Equation (25)), we concurrently execute Tensor Core and CUDA Core kernels on disjoint RNS workloads using exact integer arithmetic. The remaining homomorphic stages, including S2C, key switching, and product-tree evaluation, also run on the GPU, while the blind-rotation keys remain resident in GPU memory throughout evaluation.

8.2 Experimental setup

All experiments were conducted on a dual-socket server equipped with two AMD EPYC 7K62 processors and 1 TB of DDR4-3200 DRAM. The memory data

Table 2. Parameter sets used for the experiments.

Param. set	N	$\log(PQ)$	dnum	(Avg. prime bits) $\times$ (# of primes)				
				q_0	KS	Mul	S2C	Prod. Tree
LL13	2^{13}	218	8	25×1	28×1	20×1	20×1	25×5
LL14	2^{14}	437	4	48×1	50×2	37×1	37×1	43×5
HT14	2^{14}	438	8	28×1	29×2	23×9	23×1	25×5
HT15	2^{15}	881	4	50×1	53×4	39×9	39×1	46×5

transfer rate matches that used in the SHIP [28] evaluation. Each processor has a base frequency of 2.6 GHz and a maximum boost frequency of 3.3 GHz, both slightly lower than those of the AMD EPYC 7473X processor used by SHIP. We disabled simultaneous multithreading (SMT) and used `numactl` to control the number of CPU cores available to each experiment. The GPU experiments were conducted on an NVIDIA RTX PRO 6000 GPU.

Following SHIP, we define throughput as

$$\text{Throughput} = \frac{(\# \text{ of slots}) \times (\# \text{ of remaining levels}) \times (\text{precision})}{\text{latency}}.$$

All throughput values are normalized to the throughput of the single-core SHIP LL13 configuration, which is set to $1 \times$.

We compare our scheme with SHIP [28], PaCo [18], OpenFHE [4], and FEFBS [5]. We run the publicly available OpenFHE and FEFBS implementations on our platform. SHIP’s implementation was not publicly available at the time of evaluation, so we use the results reported in its paper. The comparison is fair as our hardware configuration is similar to SHIP’s, although our CPUs operate at slightly lower frequencies. PaCo provides a SageMath/NTL proof-of-concept implementation that is explicitly not performance-optimized; we therefore evaluate our own reimplementations of PaCo.

The blind-rotation key-packing scheme in Section 6 requires only one-time key reconstruction. We treat this reconstruction as a setup cost and exclude it from the reported bootstrapping timings.

8.3 Results on CPU

Comparison with SHIP. Table 3 compares our scheme with SHIP on real- and complex-valued inputs under the same four parameter sets. Our implementation improves precision for every configuration, with gains ranging from 0.73 bits for LL13 to 6.20 bits for LL14.

For real-valued inputs, the largest latency improvements occur for the lower-precision configurations. For LL13, our implementation reduces latency from 3.02 s to 1.62 s on one core and from 239 ms to 96.0 ms on 48 cores, corresponding to speedups of $1.86 \times$ and $2.49 \times$, respectively. For HT14, the corresponding speedups

Table 3. Comparison with SHIP (for both real and complex numbers).

Input	Param.	Method	Precision (bits)	Latency		Normalized throughput	
				1-core	48-core	1-core	48-core
Real	LL13	SHIP	4.45	3.02 s	239 ms	1.00×	12.6×
		Ours	5.18	1.62 s	96.0 ms	2.17×	36.6×
	LL14	SHIP	16.9	4.91 s	342 ms	4.67×	67.1×
		Ours	23.1	12.44 s	410 ms	2.52×	76.4×
	HT14	SHIP	4.55	13.2 s	915 ms	4.21×	60.7×
		Ours	9.20	10.3 s	564 ms	10.9×	199×
	HT15	SHIP	18.1	29.6 s	1.62 s	14.9×	273×
		Ours	24.3	77.2 s	2.68 s	7.69×	221×
	LL13	SHIP	4.45	6.04 s	430 ms	1.00×	14.0×
		Ours	5.18	2.34 s	116 ms	3.00×	60.6×
Complex	LL14	SHIP	16.9	9.82 s	667 ms	4.67×	68.8×
		Ours	23.1	22.8 s	612 ms	2.75×	102×
	HT14	SHIP	4.55	26.4 s	1.67 s	4.21×	66.6×
		Ours	9.20	15.9 s	718 ms	14.1×	313×
	HT15	SHIP	18.1	59.2 s	2.96 s	14.9×	299×
		Ours	24.3	140 s	3.63 s	8.44×	327×

are $1.28\times$ and $1.62\times$. In contrast, our LL14 and HT15 implementations are slower than SHIP despite their higher precision.

The complex-valued results show a similar pattern. For LL13, our implementation is $2.58\times$ faster on one core and $3.71\times$ faster on 48 cores. For HT14, it reduces latency from 26.4 s to 15.9 s on one core and from 1.67 s to 718 ms on 48 cores. For LL14, our single-core implementation is slower, but its 48-core latency decreases from 667 ms to 612 ms and its normalized throughput increases from $68.8\times$ to $102\times$. HT15 remains slower than SHIP in both core configurations. Overall, the clearest improvements occur for LL13 and HT14, while the larger high-precision configurations benefit less.

Comparison with PaCo and OpenFHE. Table 4 compares our LL13 implementation with PaCo and OpenFHE for real-valued inputs. All three configurations target one remaining multiplicative level and achieve measured precision between 5.18 and 5.37 bits. Our scheme uses $N = 2^{13}$ and $\log_2(PQ) = 218$, whereas both baselines use $N = 2^{15}$, with $\log_2(PQ) = 460$ for PaCo and 866 for OpenFHE. On one core, our implementation takes 1.62s, compared with 446s for PaCo and 8.08s for OpenFHE, corresponding to speedups of $275\times$ and $4.99\times$, respectively. On 48 cores, our latency decreases to 96.0ms, yielding speedups of $404\times$ over PaCo and $30.5\times$ over OpenFHE. Overall, our implementation achieves substantially lower latency across all tested core counts in this setting.

Multi-core scaling and batching. Table 5 reports the latency across different

Table 4. Comparison with PaCo and OpenFHE (for real numbers).

Method	N	$\log_2(PQ)$	precision	Target levels	Latency			
					1-core	16-core	32-core	48-core
Ours (LL13)	2^{13}	218	5.18	1	1.62 s	144 ms	94.0 ms	96.0 ms
PaCo	2^{15}	460	5.37	1	446 s	64.7 s	35.3 s	38.8 s
OpenFHE	2^{15}	866	5.20	1	8.08 s	2.67 s	2.78 s	2.93 s

CPU core counts (for real numbers). At 48 cores, the speedups range from $16.9\times$ to $30.3\times$, with latencies of 96.0 ms, 410 ms, 564 ms, and 2.68 s for LL13, LL14, HT14 and HT15, respectively. LL13 reaches its minimum latency of 94.0 ms at 32 cores and then increases slightly at 48 cores, while HT14 improves by only 3.6% from 32 to 48 cores. The larger LL14 and HT15 workloads provide more work per thread and continue to scale to all 48 cores, reaching speedups of $30.3\times$ and $28.8\times$, respectively.

Table 5. Latency across different CPU core counts (for real numbers).

Scheme	Param. set	Latency						
		1-core	2-core	4-core	8-core	16-core	32-core	48-core
Our Work	LL13	1.62 s	819 ms	420 ms	229 ms	144 ms	94.0 ms	96.0 ms
	LL14	12.4 s	6.30 s	3.18 s	1.64 s	936 ms	572 ms	410 ms
	HT14	10.3 s	5.33 s	2.76 s	1.51 s	905 ms	585 ms	564 ms
	HT15	77.2 s	38.8 s	19.4 s	10.1 s	5.28 s	3.28 s	2.68 s

Table 6 reports the amortized per-ciphertext latency on 48 CPU cores (for real numbers) as the number of concurrently processed ciphertexts increases. Increasing the batch size from 1 to 64 reduces the amortized latency from 96 ms to 29 ms for LL13, from 410 ms to 173 ms for LL14, from 564 ms to 181 ms for HT14, and from 2681 ms to 1230 ms for HT15.

Table 6. Amortized latency on 48 CPU cores (for real numbers) as the number of concurrently processed ciphertexts (batch size) increases.

Scheme	Param.	Amortized latency (ms) by batch size						
		1	2	4	8	16	32	64
Our Work	LL13	96	58	44	34	32	29	29
	LL14	410	306	239	205	190	180	173
	HT14	564	359	275	219	202	193	181
	HT15	2681	1815	1483	1335	1294	1266	1230

Functional bootstrapping. Table 7 compares our FBS method with FEFBS for evaluating GeLU, Tanh, and Sigmoid. We select the smallest FEFBS parameter set that achieves comparable final precision and retains one multiplicative level, thereby matching our precision target and output-level budget. In the low-precision setting, our method evaluates the three functions in 190-333 ms, achieving speedups of $12.6\times$ for GeLU, $12.7\times$ for Tanh, and $19.4\times$ for Sigmoid. In the high-precision setting, our method takes 2.71-4.51 s, achieving speedups of $2.55\times$ - $4.59\times$ while retaining 23.3-24.0 bits of precision.

Table 7. Functional bootstrapping comparison with FEFBS [5] (“# Levels” denotes the number of remaining multiplicative levels).

f_T	Scheme	Low Precision			High Precision		
		Latency	Precision	# Levels	Latency	Precision	# Levels
GeLU	FEFBS	3.77 s	7.50	1	12.4 s	23.5	1
	Our work	300 ms	7.17	1	2.71 s	23.3	1
Tanh	FEFBS	4.24 s	8.65	1	11.5 s	23.4	1
	Our work	333 ms	7.89	1	4.51 s	23.4	1
Sigmoid	FEFBS	3.68 s	9.06	1	12.2 s	24.1	1
	Our work	190 ms	9.04	1	2.79 s	24.0	1

8.4 Results on GPU

GPU batching. Table 8 reports the amortized per-ciphertext latency on GPU (for complex numbers) as the number of concurrently processed ciphertexts increases. At batch size 1, LL13 and LL14 take 8.2 ms and 44 ms, respectively. Relative to the corresponding 48-core CPU results in Table 3, these latencies represent speedups of $14.1\times$ for LL13 and $14.0\times$ for LL14. At batch size 64, the amortized latency decreases further to 4.6 ms for LL13 and 29 ms for LL14, yielding additional batching gains of $1.78\times$ and $1.51\times$ relative to batch size 1.

Our single-ciphertext latency approaches the 6.97 ms reported by HEAAN [19], while the amortized latency at batch size 64 falls below this figure. These results demonstrate similarly low millisecond-scale latency with a fully specified construction. A direct performance comparison with HEAAN remains challenging, because its underlying algorithmic and implementation details have not been publicly disclosed. This makes it difficult to reproduce its reported result under matched evaluation conditions, including hardware, output precision, and the number of remaining multiplicative levels.

FBS on GPU. Table 9 reports the GPU latency and precision achieved by our FBS scheme. With LL13, GeLU, Tanh and Sigmoid take 40.4 ms, 40.4 ms and 24.5 ms, and achieve 6.56, 9.19 and 8.04 bits of precision, respectively. With LL14, the corresponding latencies are 250 ms, 382 ms, and 248 ms.

Table 8. Amortized latency on GPU (for complex numbers) as the number of concurrently processed ciphertexts (batch size) increases.

Scheme	Param.	Amortized latency (ms) by batch size						
		1	2	4	8	16	32	64
Our Work	LL13	8.2	6.9	6.8	6.0	5.8	5.1	4.6
	LL14	44	35	32	30	29	29	29

Table 9. Functional bootstrapping evaluation on GPU.

Scheme	Param.	GeLU		Tanh		Sigmoid	
		Latency	Precision	Latency	Precision	Latency	Precision
Our Work	LL13	40.4 ms	6.56	40.4 ms	9.19	24.5 ms	8.04
	LL14	250 ms	23.3	382 ms	23.4	248 ms	24.0

9 Conclusion

We presented a fully specified CKKS bootstrapping construction that combines algorithmic innovations with efficient parallel execution. Building on SHIP, our narrow blind-rotation eliminates redundant table-selector multiplications, while complex-slot packing halves the resulting evaluation-ready blind-rotation key. We further reduce key-transmission costs through coefficient packing with one-time reconstruction. By fusing blind rotations with NTT-domain permutations, we reformulate their dominant computation as structured sparse matrix operations that support efficient batched GPU execution. The same framework enables low-depth functional bootstrapping by directly generating the required frequency components through blind-rotations. Our implementation demonstrates practical sub-10ms CKKS bootstrapping.

References

1. SEAL, <https://github.com/microsoft/SEAL>
2. Alexandru, A., Kim, A., Polyakov, Y.: General functional bootstrapping using CKKS. In: Kalai, Y.T., Kamara, S.F. (eds.) *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part III. Lecture Notes in Computer Science, vol. 16002, pp. 304–337. Springer (2025). https://doi.org/10.1007/978-3-032-01881-6_10, https://doi.org/10.1007/978-3-032-01881-6_10
3. Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. pp. 962–979. IEEE Computer Society (2018). <https://doi.org/10.1109/SP.2018.00062>, <https://doi.org/10.1109/SP.2018.00062>
4. Badawi, A.A., Alexandru, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D.,

- Pascoe, C., Polyakov, Y., Quah, I., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022), <https://eprint.iacr.org/2022/915>
5. Bian, S., Fu, Y., Shen, R., Pan, H., Wang, A., Guan, Z.: High-precision functional bootstrapping for CKKS from Fourier extension. In: Advances in Cryptology – EUROCRYPT 2026, Part IV. Lecture Notes in Computer Science, vol. 16544, pp. 274–303. Springer (2026). https://doi.org/10.1007/978-3-032-25327-9_10, <https://eprint.iacr.org/2026/367>
 6. Bossuat, J., Mouchet, C., Troncoso-Pastoriza, J.R., Hubaux, J.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 587–617. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_21, https://doi.org/10.1007/978-3-030-77870-5_21
 7. Bossuat, J., Troncoso-Pastoriza, J.R., Hubaux, J.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13269, pp. 521–541. Springer (2022). https://doi.org/10.1007/978-3-031-09234-3_26, https://doi.org/10.1007/978-3-031-09234-3_26
 8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS 2012). pp. 309–325. ACM (2012). <https://doi.org/10.1145/2090236.2090262>
 9. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. **43**(2), 831–871 (2014). <https://doi.org/10.1137/120868669>, <https://doi.org/10.1137/120868669>
 10. Challis, M.F., Robinson, J.P.: Some extremal postage stamp bases. Journal of Integer Sequences **13**(2), 3 (2010)
 11. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Advances in Cryptology – EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II. p. 34–54. Springer-Verlag, Berlin, Heidelberg (2019). https://doi.org/10.1007/978-3-030-17656-3_2, https://doi.org/10.1007/978-3-030-17656-3_2
 12. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. IACR Cryptol. ePrint Arch. **2018**, 153 (2018), <https://api.semanticscholar.org/CorpusID:4887112>
 13. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: Cid, C., Jr., M.J.J. (eds.) Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11349, pp. 347–368. Springer (2018). https://doi.org/10.1007/978-3-030-10970-7_16, https://doi.org/10.1007/978-3-030-10970-7_16

14. Cheon, J.H., Jang, D., Kang, J., Rhee, H.: SWIFT: Shallow and SIMD-aware CKKS functional bootstrapping for low-latency. Cryptology ePrint Archive, Paper 2026/1163 (2026), <https://eprint.iacr.org/2026/1163>
15. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_15, https://doi.org/10.1007/978-3-319-70694-8_15
16. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 3–33 (2016). https://doi.org/10.1007/978-3-662-53887-6_1, https://doi.org/10.1007/978-3-662-53887-6_1
17. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 1135–1150. ACM (2021). <https://doi.org/10.1145/3460120.3484760>, <https://doi.org/10.1145/3460120.3484760>
18. Coron, J., Seuré, T.: Paco: Bootstrapping for CKKS via partial coefficient slot. In: Hanaoka, G., Yang, B. (eds.) Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part VII. Lecture Notes in Computer Science, vol. 16251, pp. 36–67. Springer (2025). https://doi.org/10.1007/978-981-95-5122-4_2, https://doi.org/10.1007/978-981-95-5122-4_2
19. CryptoLab: Heaan library (2022), <https://heaan.it>
20. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Paper 2014/816 (2014), <https://eprint.iacr.org/2014/816>
21. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012), <http://eprint.iacr.org/2012/144>
22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009). pp. 169–178. ACM (2009)
23. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology - CRYPTO 2013. Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_5
24. Halevi, S., Shoup, V.: Algorithms in helib. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 554–571. Springer (2014). https://doi.org/10.1007/978-3-662-44371-2_31, https://doi.org/10.1007/978-3-662-44371-2_31
25. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in helib. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th

- Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 93–120. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_4, https://doi.org/10.1007/978-3-319-96884-1_4
26. Han, K., Hhan, M., Cheon, J.H.: Improved homomorphic discrete fourier transforms and FHE bootstrapping. *IEEE Access* **7**, 57361–57370 (2019). <https://doi.org/10.1109/ACCESS.2019.2913850>, <https://doi.org/10.1109/ACCESS.2019.2913850>
 27. Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: Jarecki, S. (ed.) *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020*, San Francisco, CA, USA, February 24-28, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12006, pp. 364–390. Springer (2020). https://doi.org/10.1007/978-3-030-40186-3_16, https://doi.org/10.1007/978-3-030-40186-3_16
 28. Jung, H.C.O., Hanrot, G., Kim, J., Stehlé, D.: Ship: A shallow and highly parallelizable ckks bootstrapping algorithm. In: *International Conference on the Theory and Application of Cryptographic Techniques* (2025), <https://api.semanticscholar.org/CorpusID:278339985>
 29. Kohonen, J.: An improved lower bound for finite additive 2-bases. *Journal of Number Theory* **174**, 518–524 (May 2017). <https://doi.org/10.1016/j.jnt.2016.11.011>, <http://dx.doi.org/10.1016/j.jnt.2016.11.011>
 30. Lee, J., Lee, E., Lee, Y., Kim, Y., No, J.: High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: Canteaut, A., Standaert, F. (eds.) *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 618–647. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_22, https://doi.org/10.1007/978-3-030-77870-5_22
 31. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. *J. ACM* **60**(6), 43:1–43:35 (2013). <https://doi.org/10.1145/2535925>, <https://doi.org/10.1145/2535925>
 32. Mazzone, F., Everts, M.H., Hahn, F., Peter, A.: Efficient ranking, order statistics, and sorting under CKKS. In: Bauer, L., Pellegrino, G. (eds.) *34th USENIX Security Symposium, USENIX Security 2025*, Seattle, WA, USA, August 13-15, 2025. pp. 8541–8558. USENIX Association (2025), <https://www.usenix.org/conference/usenixsecurity25/presentation/mazzone>
 33. Moon, J., Yoo, D., Jiang, X., Kim, M.: THOR: secure transformer inference with homomorphic encryption. In: Huang, C., Chen, J., Shieh, S., Lie, D., Cortier, V. (eds.) *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025*, Taipei, Taiwan, October 13-17, 2025. pp. 3765–3779. ACM (2025). <https://doi.org/10.1145/3719027.3765150>, <https://doi.org/10.1145/3719027.3765150>
 34. Xiao, P., Ouyang, R., Zhang, H., Zhang, J., Liu, J.: Bootstrapping is all you need: Secure transformer inference via improved CKKS functional bootstrapping. *Cryptology ePrint Archive*, Paper 2026/1255 (2026), <https://eprint.iacr.org/2026/1255>
 35. Zhang, J., Yang, X., He, L., Chen, K., Lu, W., Wang, Y., Hou, X., Liu, J., Ren, K., Yang, X.: Secure transformer inference made non-interactive. In: *32nd Annual Network and Distributed System Security Symposium, NDSS 2025*, San Diego, California, USA, February 24-28, 2025.

- The Internet Society (2025), <https://www.ndss-symposium.org/ndss-paper/secure-transformer-inference-made-non-interactive/>
36. Zhang, L., Wang, X., Jie, S.J., Huang, Z., Zhong, J., Wang, H., Duan, P., Lam, K.: MOAI: module-optimizing architecture for non-interactive secure transformer inference. IACR Cryptol. ePrint Arch. **2025**, 991 (2025), <https://eprint.iacr.org/2025/991>
37. Zhao, J., Wu, X., Fan, G., Zhang, M., Yan, S., Ju, L., Ji, Z.: A framework for developing and optimizing fully homomorphic encryption programs on GPUs. In: Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2026), Volume 2. pp. 25–40. ACM (2026). <https://doi.org/10.1145/3779212.3790120>

A Frequently used notations

Table 10. Frequently used notations.

Notation	Description
(a, b)	a CKKS ciphertext
B	$\lfloor n/2(g-1) \rfloor$
c	coefficient of the trigonometric approximant
d	degree of the trigonometric approximant
e	natural logarithm
f_T	target function
g	N/h
h	hamming weight of the secret key
L	prescribed multiplicative depth
m	plaintext
N	lattice dimension
n	# slots, $n = N/2$
o	non-zero position in the secret key
p	trigonometric approximant
q, Q	modulus
s	secret key
t	$2d+1$
ι	$\sqrt{-1}$
η	$2\pi/q$
ω	$e^{\iota\eta}$
κ	blind-rotation key

B Proof of Lemma 3.1

Proof. The blind rotation must support all g candidate offsets. For each y , the source window is shifted:

y	required coefficient indices
0	$-xg, \dots, n-1-xg$
1	$-1-xg, \dots, n-2-xg$
$\vdots$	$\vdots$
$g-1$	$-(g-1)-xg, \dots, n-g-xg$

Their union is

$$\mathcal{W}_x = \{a_{-(g-1)-xg}, \dots, a_{n-1-xg}\}.$$

which has $n+g-1$ coefficients. $\square$

C Proof of Lemma 3.2

Proof. Since $I_{\text{head}} = \{0, \dots, n-g\}$, the required coefficient indices for each y are

y	required coefficient indices
0	$-xg, \dots, n-g-xg$
1	$-1-xg, \dots, n-g-1-xg$
$\vdots$	$\vdots$
$g-1$	$-(g-1)-xg, \dots, n-2g+1-xg$

Their union is

$$\mathcal{H}_x = \{a_{-(g-1)-xg}, \dots, a_{n-g-xg}\},$$

which has n coefficients. $\square$

D Proof of Lemma 3.3

Proof. Since $I_{\text{tail}} = \{n-g+1, \dots, n-1\}$, the required coefficient indices for each y are

y	required coefficient indices
0	$n-g+1-xg, \dots, n-1-xg$
1	$n-g-xg, \dots, n-2-xg$
$\vdots$	$\vdots$
$g-1$	$n-2g+2-xg, \dots, n-g-xg$

Their union is

$$\mathcal{T}_x = \{a_{n-2g+2-xg}, \dots, a_{n-1-xg}\},$$

which has $2(g-1)$ coefficients. $\square$

E Proof of Lemma 3.5

Proof. Since $i \in I_{\text{tail}}$ and $y \in [g]$, we have

$$n - g + 1 \leq i < n, \quad 0 \leq y \leq g - 1.$$

As $2(g - 1) \leq n$, we have $i \geq y$, and the rotation does not wrap around. Moreover, as

$$n - 2(g - 1) \leq i - y < n,$$

the entry labeled $i - y$ is present in the tail table of Equation (10). Hence

$$\begin{aligned} (\text{Rot}_y(\pi''_x(\mathbf{a}))) [i] &= \omega^{a_{(i-y)-xg} \alpha_{i-y, xg}} \\ &= \omega^{a_{i-(xg+y)} \alpha_{i-y, xg}}. \end{aligned}$$

Furthermore, by the definition of α (cf. Equation (2)), both $\alpha_{i-y, xg}$ and $\alpha_{i, xg+y}$ equal 1 when $xg + y \leq i$; otherwise, both equal -1 . Thus

$$\alpha_{i-y, xg} = \alpha_{i, xg+y}.$$

Therefore,

$$(\text{Rot}_y(\pi''_x(\mathbf{a}))) [i] = \omega^{a_{i-(xg+y)} \alpha_{i, xg+y}},$$

which proves the lemma. $\square$

F Proof of Proposition 3.2

Proof. For block $x = qB + \ell < h$, recall that its nonzero secret-key position is

$$j_x = xg + o_x.$$

On the last $g - 1$ slots of this block's packed group, the encrypted selectors remove every term except

$$y = o_x, \quad k = \begin{cases} 1, & s_{j_x} = 1, \\ 2, & s_{j_x} = -1, \end{cases}$$

which follows directly from the definition of the packed masks.

For every $i = n - g + 1 + u \in I_{\text{tail}}$, where $0 \leq u < g - 1$, the corresponding packed slot is

$$p = (2\ell + 1)(g - 1) + u.$$

For every $y \in [g]$, we have

$$2\ell(g - 1) \leq p - y < 2(\ell + 1)(g - 1).$$

Thus, rotation reads an entry within the same packed group, without crossing a group boundary. By the concatenation defining $\tau_{k,q}$ (cf. Equation (11)), this entry corresponds to index $i - y$ of the tail table:

$$(\text{Rot}_y(\tau_{k,q}(\mathbf{a}))) [p] = \pi''_{k,x}(\mathbf{a})[i - y] = (\text{Rot}_y(\pi''_{k,x}(\mathbf{a}))) [i].$$

If $s_{j_x} = 1$, the selected table is $\pi''_{1,x}(\mathbf{a}) = \pi''_x(\mathbf{a})$. Lemma 3.5 therefore gives

$$(\text{Rot}_{o_x}(\tau_{1,q}(\mathbf{a})))[p] = \omega^{a_{i-(xg+o_x)}\alpha_{i,xg+o_x}} = \omega^{a_{i-j_x}\alpha_{i,j_x}}.$$

If $s_{j_x} = -1$, the selected table is $\pi''_{2,x}(\mathbf{a}) = \pi''_x(-\mathbf{a})$, hence

$$(\text{Rot}_{o_x}(\tau_{2,q}(\mathbf{a})))[p] = \omega^{-a_{i-j_x}\alpha_{i,j_x}}.$$

In either case, the sum defining γ''_q contributes

$$\omega^{s_{j_x} a_{i-j_x} \alpha_{i,j_x}}$$

to the packed slot p . Notice that the first $g-1$ entries of every selector mask for this group are zero, so the corresponding slots of γ''_q encrypt zero. Consequently, the $2(g-1)$ -slot group assigned to block x encrypts

$$\mathbf{0}_{g-1} \parallel [\omega^{s_{j_x} a_{i-j_x} \alpha_{i,j_x}}]_{i \in I_{\text{tail}}},$$

as claimed. $\square$

G Proof of Proposition 3.3

Proof. Fix a pack q and a block $x = qB + \ell$ in this pack. For a tail index i , write

$$i = n - g + 1 + u, \quad 0 \leq u < g - 1.$$

By Proposition 3.2, the contribution of block x to tail index i is stored in slot

$$p = (2\ell + 1)(g - 1) + u$$

of ζ''_q , and this slot encrypts

$$\omega^{s_{j_x} a_{i-j_x} \alpha_{i,j_x}}.$$

Slot i of $\text{Rot}_{n-2(\ell+1)(g-1)}(\zeta''_q)$ reads

$$i - (n - 2(\ell + 1)(g - 1)) = (2\ell + 1)(g - 1) + u = p.$$

Therefore, for every $i \in I_{\text{tail}}$, this rotated ciphertext contains the contribution

$$\omega^{s_{j_x} a_{i-j_x} \alpha_{i,j_x}}$$

of block x in slot i . Its other slots may be nonzero, but they do not affect the final result.

As q and ℓ range over the products in (13), $x = qB + \ell$ ranges over every block in $[h]$ exactly once. Hence, for every $i \in I_{\text{tail}}$,

$$\gamma''[i] = \omega^{b_i} \prod_{x \in [h]} \omega^{s_{j_x} a_{i-j_x} \alpha_{i,j_x}} = \omega^{b_i + \sum_{x \in [h]} s_{j_x} a_{i-j_x} \alpha_{i,j_x}} = \omega^{m_i + \epsilon_i}.$$

For every $i \in I_{\text{head}}$, the corresponding entry of β'' is zero. Thus $\gamma''[i] = 0$, regardless of the other values carried by the rotated packed ciphertexts. $\square$

H Proof of Theorem 3.1

Proof. By Proposition 3.1,

$$\gamma' = \text{Enc}(\text{Ecd}_{\text{slot}}([\omega^{m_0+\epsilon_0}, \dots, \omega^{m_{n-g}+\epsilon_{n-g}}] \parallel \mathbf{0}_{g-1})).$$

Thus γ' contains the desired values on I_{head} and is zero on I_{tail} .

By Proposition 3.3,

$$\gamma'' = \text{Enc}(\text{Ecd}_{\text{slot}}(\mathbf{0}_{n-g+1} \parallel [\omega^{m_{n-g+1}+\epsilon_{n-g+1}}, \dots, \omega^{m_{n-1}+\epsilon_{n-1}}])).$$

Thus γ'' is zero on I_{head} and contains the desired values on I_{tail} .

Therefore, homomorphic addition gives

$$\gamma_{\text{narrow}} = \gamma' + \gamma'' = \text{Enc}(\text{Ecd}_{\text{slot}}([\omega^{m_0+\epsilon_0}, \dots, \omega^{m_{n-1}+\epsilon_{n-1}}])),$$

as claimed. □