Bootstrapping via Ring Switching without Slot Recovery

Independent Leaf Evaluation for SIMD-Packed Ciphertexts

Zhaoyang Liang[1⋆] and Dan Ding[1,2,3(✉)]

[1] College of Cryptology and Cyber Sciences, Nankai University, Tianjin 300350, China P. R.
liangzhaoyang@mail.nankai.edu.cn, dingdan@nankai.edu.cn
[2] College of Computer Sciences, Nankai University, Tianjin 300350, China P. R.
dingdan@nankai.edu.cn
[3] Serica Laboratory, Beijing 100084, China P. R.
dingdan@serica-tech.com.cn

Abstract. Ring switching provides a natural way to reduce bootstrapping cost in ring-based fully homomorphic encryption by moving computation to smaller rings. For SIMD-packed ciphertexts, however, ring switching changes the slot layout, so conventional approaches apply slot recovery to ensure correct computation on the original slot values, consuming capacity and requiring synchronization across ciphertexts. In this paper, we show that slot recovery is not indispensable: CKKS and BGV/BFV bootstrapping can proceed directly after ring switching. We further prove that a continuous slotwise function admits independent leaf evaluation for unrestricted real CKKS inputs exactly when it is affine. Our construction provides an extra layer of parallelism, enables cheaper arithmetic in smaller rings, and can further simplify bootstrapping by exploiting the leaf structure. For CKKS with $N = 2^{17}$ and $n = 2^{16}$, we achieve a 1.13× serial speedup; evaluating two leaves in parallel yields a 2.03×–2.18× speedup and 2.00×–2.14× the throughput of direct bootstrapping. For BGV with $p = 65537$, $N = 2^{16}$, and $n = 2^{15}$, we achieve a 1.30× serial speedup and 1.63× the throughput of direct general bootstrapping under matched transform settings, while reducing key-generation time by 50.6% and server-key size by 48.6%.

Keywords: Fully Homomorphic Encryption · Ring Switching · Bootstrapping · SIMD operations

1 Introduction

Fully homomorphic encryption (FHE) enables computation over encrypted data without decrypting it. In modern ring-based FHE schemes such as BGV [12],

⋆ Supported by the Supercomputing Center of Nankai University(NKSC)

BFV [11,21], and CKKS [17], bootstrapping [25], which enables unbounded homomorphic computation by refreshing ciphertexts, remains one of the main performance bottlenecks. These schemes encode messages as plaintext polynomials in cyclotomic rings, where SIMD packing allows one ciphertext to carry multiple plaintext values in parallel [43]. The ring dimension controls both the number of available packed slots and the modulus budget that can be supported at a given security level [8]. Larger rings can therefore provide more packed slots and support larger modulus budgets, but make operations more expensive and require larger evaluation keys and more memory.

Ring switching [26] provides a natural way to move computation to smaller rings. By homomorphically decomposing the plaintext polynomial into coefficient components, it splits a large-ring ciphertext into several smaller-ring ciphertexts that can later be merged back into the top ring. If subsequent operations do not mix these components, the resulting ciphertexts can be evaluated independently, benefiting from cheaper arithmetic and parallel execution across CPU cores.

However, for SIMD-packed ciphertexts, ring switching does not preserve the original slot values in the smaller-ring ciphertexts. The conventional approach therefore applies *slot recovery*, a linear transform across these ciphertexts that reconstructs the original slot values [26,39]. Prior work has observed that the additional cost of slot recovery can offset or even exceed the savings from computation in smaller rings [39].

This concern is especially relevant for large dimension reductions, as recovery must combine numerous smaller-ring ciphertexts. For bootstrapping, however, the required modulus budget and leaf-ring security limit the dimension reduction and hence recovery's arithmetic cost. But recovery still consumes capacity and requires cross-leaf synchronization, adding intermediate communication to execution across FPGAs (as in HEAP [1]) or HPC cluster nodes. Without slot recovery, each leaf can bootstrap independently before Merge.

The need for slot recovery depends on subsequent computation. Although recovery is natural when computation is viewed as acting on the original SIMD slots, what matters is whether that computation can operate directly on the transformed slot representation produced by Split. We show that some computations, including conventional bootstrapping, can be performed in this representation while preserving the intended plaintext result.

1.1 Our Contributions

The preceding discussion motivates us to reassess a key assumption implicit in prior ring-switching constructions and to ask the central question of this work:

Must the original SIMD slot layout be recovered after ring switching before further homomorphic computation?

The answer is negative. We establish this by tracking the plaintext polynomials and slot values carried by the ciphertexts. Our main contributions are summarized as follows:

(i) Slot Recovery Is Not Inherent to Ring Switching. To our knowledge, this is the first work to establish that the altered SIMD slot representation produced by ring switching can serve directly as a valid computational representation. We characterize when subsequent computation admits independent leaf evaluation. In particular, on the full space of real CKKS slots, a continuous slotwise function admits independent leaf evaluation if and only if it is affine.

(ii) Independent Leaf Bootstrapping and Optimizations. We bootstrap SIMD-packed ciphertexts without slot recovery by independently applying conventional bootstrappers to the smaller-ring leaves. This enables simple leaf parallelism and can reduce CtS/StC operation counts and server-key storage. In suitable BGV settings, a general bootstrap becomes two thin leaf bootstraps; for an already-thin input, only one leaf needs refreshing. Under uniform ternary ModRaise secrets and the same modeled tail target, the correction range is approximately $1/\sqrt{k}$ of the top-ring range. The correctness statements inherit the ordinary approximation and digit-removal conditions of the underlying leaf bootstrappers.

(iii) Parameter Selection and Performance Analysis. We jointly select ring dimensions, moduli, key-switching parameters, and CtS/StC partitions under the stated security, correctness, and output-capacity conditions. Our CKKS and BGV experiments distinguish the effects of smaller-ring arithmetic, leaf parallelism, and slot recovery. We reassess slot recovery and find little runtime overhead in our setting. The main CKKS setting achieves 2.00×–2.14× Direct's throughput and reduces server-key storage by 51.8% at 15 output levels. The throughput ratio is 2.21× with an alternative dense-key bootstrapper. With matched column counts and partitions, general BGV achieves a 1.30× serial speedup and 1.63× Direct's throughput, with lower initial and higher remaining capacity. At comparable remaining capacity, these ratios are 1.17× and 1.02×. Thin BGV achieves a 1.89× speedup and 22.2% higher throughput at equal initial capacity.

1.2 Technical Overview

Let $N = kn$ and $Y = X^k$. The decomposition $P(X) = \sum_{r=0}^{k-1} X^r P_r(X^k)$ defines $\mathsf{Split}_k(P) = (P_0, \ldots, P_{k-1})$, with inverse Merge_k. Write SR_k for slot recovery across the leaves [26] and BTS_m for a conventional bootstrapper in dimension m. The conventional route [39] and our route are

$$\mathcal{B}_{\mathrm{rec}} = \mathsf{Merge}_k \circ \mathsf{SR}_k^{-1} \circ (\mathsf{BTS}_n)^{\oplus k} \circ \mathsf{SR}_k \circ \mathsf{Split}_k,$$
$$\mathcal{B}_{\mathrm{ours}} = \mathsf{Merge}_k \circ (\mathsf{BTS}_n)^{\oplus k} \circ \mathsf{Split}_k.$$

The formulas omit the key switches before Split and after Merge. Each leaf runs a complete bootstrap as an independent task, avoiding recovery's level or capacity cost and deferring synchronization to Merge. All leaves share a secret and bootstrap parameters, so one evaluation-key set serves all k leaf bootstraps.

The key observation is that CtS exposes each leaf polynomial's coefficients to the nonlinear refresh stage: EvalMod in CKKS or Unpack–digit removal–Pack

in BGV/BFV. StC returns the refreshed coefficients to polynomial form. Write $\Phi_m := \mathsf{StC}_m \circ \psi_m \circ \mathsf{CtS}_m$ for the ideal refresh after modulus raising $\mathsf{ModRaise}_m$, with ψ_m denoting the nonlinear stage. For a common lifted input T, using the same scalar rule in both dimensions gives

$$T(X) = \sum_{r=0}^{k-1} X^r T_r(X^k), \qquad \Phi_N(T)(X) = \sum_{r=0}^{k-1} X^r \Phi_n(T_r)(X^k).$$

Independent leaf refreshes therefore give the same ideal result after Merge. For CKKS, Split and Merge preserve the joint coefficient ℓ_2-norm, introducing no additional amplification of the assumed error bounds. For BGV/BFV, exact leaf digit removal recovers the top-ring plaintext under the ordinary correctness conditions. These identities require no probabilistic independence between leaves.

The leaf structure also simplifies BGV bootstrapping when top-ring slots have extension degree k and leaf slots have degree one. For $k = 2$, Split gives, in compatible slot bases,

$$\text{general: } a_j + b_j\alpha_j \longmapsto (a_j, b_j), \qquad \text{thin: } a_j \longmapsto (a_j, 0),$$

where α_j is a top-slot root and $a_j, b_j \in \mathbb{Z}_t$. Both leaves have scalar slots and admit thin bootstrapping with trivial Unpack/Pack [29]; thin inputs require refreshing only the first leaf.

Beyond bootstrapping, consider a plaintext map $\mathcal{F}$ on a reachable domain $\mathcal{D}$. A required output leaf is independently computable exactly when equal corresponding input leaves imply equal output leaves (Lemma 5). Applying this criterion to continuous slotwise functions on the full space of real CKKS slots yields the affine classification.

1.3 Related Work

The original ring-switching construction recovers SIMD slot values through an inverse linear transform [26,39]. We review this construction in Section 3 and distinguish our approach from the closest prior work below.

HERMES for ring packing [7]. HERMES avoids the slot-recovery problem by working with coefficient-encoded CKKS ciphertexts, for which ring switching directly rearranges polynomial coefficients without changing the relevant plaintext representation. In contrast, we retain SIMD packing, where ring switching does change the slot layout, and show that the original layout need not always be restored before subsequent homomorphic computation.

Subring Secret Encapsulation [39]. Dense-key bootstrapping via subring secret encapsulation avoids slot recovery by keeping the ciphertext in the original ring while key-switching its secret to an embedded subring. Our work instead switches the ciphertext ring and shows that conventional bootstrapping can operate directly on the resulting leaf ciphertexts without slot recovery. We also use this method as an alternative in our CKKS experiments to show that our construction is not restricted to a particular leaf bootstrapper.

Bootstrapping Architectures. SHIP [16] and PaCo [19] introduce CKKS bootstrapping architectures, while blind-rotation methods support both CKKS and BGV/BFV [31]. These works focus on bootstrapper design. We show that conventional bootstrapping can operate directly on the slot representation produced by ring switching, without slot recovery. This enables independent leaf evaluation in smaller rings, with parallelism across leaves and further optimizations based on their structure.

Other works [4,5,22,32,41,42] improve homomorphic computation through alternative algebraic representations of plaintexts or ciphertexts, ring choices, and packing or slot structures, whereas our work shows that smaller-ring ciphertexts can serve directly as a valid computational representation whenever subsequent computation permits independent leaf evaluation.

1.4 Paper Organization

Section 2 introduces the notation, packed encodings, ciphertext semantics, and conventional bootstrapping. Section 3 reviews ring switching and slot recovery. Sections 4 and 5 present our CKKS and BGV/BFV constructions and analyze correctness preservation and cost. Section 6 examines slot recovery fusion and its limits, and Section 7 characterizes independent leaf evaluation for general plaintext computations. Section 8 discusses security and joint parameter selection, and Section 9 presents the implementation and experiments.

2 Preliminaries

2.1 Notation

Let the top and leaf ring dimensions N and n be powers of two, with $N = kn$ and splitting factor k. For an indeterminate Z and $m \in \{N, n\}$, let $\mathcal{R}_{\mathbb{Z},m} := \mathbb{Z}[Z]/(Z^m + 1)$, $\mathcal{R}_m := \mathbb{R}[Z]/(Z^m + 1)$, and $\mathcal{R}_{Q,m} := \mathbb{Z}_Q[Z]/(Z^m + 1)$ for an integer modulus Q. We use X and Y in the top and leaf rings, respectively, with embedding $Y \mapsto X^k$. We reserve q for an input or current ciphertext modulus and Q, including decorated forms, for working moduli. In BGV/BFV, the plaintext modulus is $t = p^e$, where p is an odd prime and $e \geq 1$. For a positive integer a, write $[a] := \{0, \ldots, a-1\}$. We use $\log = \log_2$, the coefficient norm $\|A\|_\infty := \max_i |a_i|$ for $A = \sum_i a_i Z^i$, and $\odot$ for componentwise multiplication.

Let $\zeta_M = \exp(2\pi i/M)$, and order the roots $\Omega_m = \{\omega \in \mathbb{C} : \omega^m + 1 = 0\}$ as $\omega_{m,j} := \zeta_{2m}^{2j+1}$, $0 \leq j < m$. Write $\mathrm{ev}_m(P) := (P(\omega_{m,j}))_{0 \leq j < m}$ for the full evaluation vector. For BGV/BFV, fix a common splitting extension of $\mathbb{Z}_t$ for Z^N+1 and Z^n+1, with roots ordered compatibly with the factor and Frobenius-orbit indexing. We use Ω_m for these ordered roots and ev_m for evaluation at them. In both cases, $\mathrm{ev}_m(P)$ has m coordinates and is distinct from the SIMD slot vector $\mathsf{slot}_m(P)$.

2.2 Ciphertexts in CKKS and BGV/BFV

We recall SIMD packing [43] and the decryption relations of RLWE [36] ciphertexts in CKKS and BGV/BFV, explaining how each ciphertext carries a plaintext polynomial and its decoded slots. We call this interpretation *ciphertext semantics*.

Encoding in CKKS and BGV/BFV For CKKS [17], define the real-linear isomorphism $\mathsf{DFT}_m : \mathcal{R}_m \to \mathbb{C}^{m/2}$ by $\mathsf{DFT}_m(P) := (P(\omega_{m,j}))_{0\le j<m/2}$. For a message $z \in \mathbb{C}^{m/2}$ and scale $\Delta > 0$, encoding and decoding are $\mathsf{Ecd}_{\Delta,m}(z) = \lfloor \Delta \mathsf{DFT}_m^{-1}(z) \rceil \in \mathcal{R}_{\mathbb{Z},m}$ and $\mathrm{Dcd}_{\Delta,m}(P) = \Delta^{-1}\mathsf{DFT}_m(P)$. We write the decoded slot vector at the scale assigned to P as

$$\mathsf{slot}_m(P) := \mathrm{Dcd}_{\Delta,m}(P) = (P(\omega_{m,j})/\Delta)_{0\le j<m/2} \in \mathbb{C}^{m/2}.$$

Thus $\mathsf{slot}_m(\mathsf{Ecd}_{\Delta,m}(z)) \approx z$. Throughout the CKKS discussion, polynomial evaluations are unscaled: for $0 \le j < m/2$, $(\mathrm{ev}_m(P))_j = \Delta(\mathsf{slot}_m(P))_j$ and $(\mathrm{ev}_m(P))_{m-1-j} = \Delta\overline{(\mathsf{slot}_m(P))_j}$. Addition and suitably rescaled multiplication act approximately componentwise on these complex slots. Hence CKKS slots are independent approximate complex SIMD coordinates.

For BGV/BFV, we follow the packed representation of [24,23]. Write $d_m = \mathrm{ord}_{2m}(p)$ and $\ell_m = m/d_m$. The irreducible factors of $Z^m + 1$ modulo p have degree d_m. Their pairwise coprime monic Hensel lifts $F_{m,j} \in \mathbb{Z}_t[Z]$ satisfy $Z^m + 1 = \prod_{j=0}^{\ell_m-1} F_{m,j}(Z)$. Set $E_{t,m} := \mathbb{Z}_t[Z]/(F_{m,0}(Z))$, let θ_m be the residue class of Z, and fix factor isomorphisms $\iota_{m,j} : E_{t,m} \to \mathbb{Z}_t[Z]/(F_{m,j}(Z))$, indexed by the Frobenius-orbit order induced by the root ordering. The CRT gives the ring isomorphism

$$\mathsf{slot}_m : \mathcal{R}_{t,m} \longrightarrow E_{t,m}^{\ell_m}, \qquad P \longmapsto \left(\iota_{m,j}^{-1}(P \bmod F_{m,j})\right)_{0\le j<\ell_m}.$$

Fix a Frobenius automorphism σ of this extension, and write σ_m for its restriction to the embedded copy of $E_{t,m}$. Up to the fixed root ordering, $\mathrm{ev}_m(P) = \left(\sigma_m^a\big((\mathsf{slot}_m(P))_j\big)\right)_{(j,a)\in[\ell_m]\times[d_m]}$. Each slot therefore represents one Frobenius orbit and has d_m coordinates over $\mathbb{Z}_t$ in the basis $1, \theta_m, \ldots, \theta_m^{d_m-1}$.

Encryption in CKKS and BGV/BFV For a secret key $s \in \mathcal{R}_{\mathbb{Z},m}$ and ciphertext $\mathbf{ct} = (c_0, c_1) \in \mathcal{R}_{q,m}^2$, define $\mathsf{Dec}_s(\mathbf{ct}) := [c_0 + c_1 s]_q$, with coefficientwise reduction. We use $\vDash$ to specify the carried plaintext and its decoding.

Definition 1 (Semantics of a ciphertext). *For $m \in \{N, n\}$, we write $\mathbf{ct} \vDash_{m,q} P$ if $\mathbf{ct}$ carries P under the scheme's decryption-correctness conditions, including the prescribed CKKS error tolerance. Its message is $\mathsf{slot}_m(P)$, using the stated scale or CRT map. The decryption relations are*

$$\begin{cases} \mathsf{Dec}_s(\mathbf{ct}) = P + \varepsilon_{\mathrm{dec}} \pmod q, & \text{for CKKS, } P \in \mathcal{R}_{\mathbb{Z},m}, \\ \mathsf{Dec}_s(\mathbf{ct}) = P + t\varepsilon_{\mathrm{dec}} \pmod q, & \text{for BGV, } P \in \mathcal{R}_{t,m}, \\ \mathsf{Dec}_s(\mathbf{ct}) = \left\lfloor \frac{q}{t} P \right\rceil + \varepsilon_{\mathrm{dec}} \pmod q, & \text{for BFV, } P \in \mathcal{R}_{t,m}. \end{cases}$$

Here $\varepsilon_{\mathrm{dec}}$ is the decryption error, subject to these conditions. Rounding is coefficientwise; BGV/BFV plaintexts use fixed centered integer lifts.

2.3 Coefficient Packing and Slot Transforms

After capacity raising, the values to be refreshed are the coefficients of the plaintext polynomial, whereas homomorphic evaluation acts slotwise. CoeffToSlot therefore moves these coefficients into slots, and SlotToCoeff reverses this representation change.

Let $m \in \{N, n\}$ and $P(Z) = \sum_{i=0}^{m-1} a_i Z^i$, with $P \in \mathcal{R}_{\mathbb{Z},m}$ for CKKS and $P \in \mathcal{R}_{t,m}$ for BGV/BFV. At the plaintext level, the ideal map CtS_m produces a plaintext B whose slots contain the coefficients of P in the arrangements below. For ideal CKKS transformations, we also allow $B \in \mathcal{R}_m$ in $\vDash$, with the encoding error absorbed into $\varepsilon_{\mathrm{dec}}$. To define the corresponding ciphertext semantics independently of how this representation is produced, we write $\mathbf{ct} \vDash^{\mathsf{cpack}}_{m,q} P$ if and only if there exists such a B with $\mathbf{ct} \vDash_{m,q} B$. For CKKS,

$$\big(\mathsf{slot}_m(B)\big)_j = a_j + \mathrm{i}a_{j+m/2}, \qquad 0 \le j < m/2.$$

This equality specifies the ideal relation; concrete transformation errors are included in $\varepsilon_{\mathrm{dec}}$. EvalMod subsequently processes the real and imaginary parts separately [9].

For BGV/BFV, let $\zeta_{4,m} := \theta_m^{m/2}$, so $\zeta_{4,m}^2 = -1$. When $p \equiv 3 \pmod 4$, d_m is even. The coefficient packing is, for $0 \le v < \ell_m$,

$$\big(\mathsf{slot}_m(B)\big)_v = \begin{cases} \displaystyle\sum_{h=0}^{d_m-1} a_{vd_m+h}\theta_m^h, & p \equiv 1 \pmod 4, \\ \displaystyle\sum_{h=0}^{d_m/2-1} \big(a_{vd_m/2+h} + \zeta_{4,m} a_{(m+vd_m)/2+h}\big)\theta_m^h, & p \equiv 3 \pmod 4. \end{cases}$$

Unpacking follows the conventions of [24,23]. If $p \equiv 1 \pmod 4$, then $\zeta_{4,m} \in \mathbb{Z}_t$, and Unpack separates the d_m coordinates in the power basis. If $p \equiv 3 \pmod 4$, then $E'_{t,m} := \mathbb{Z}_t[\zeta_{4,m}]$ is free of rank two over $\mathbb{Z}_t$, while $E_{t,m}$ is free of rank $d_m/2$ over $E'_{t,m}$, with basis $1, \theta_m, \ldots, \theta_m^{d_m/2-1}$. Unpack separates these $d_m/2$ coordinates, then decomposes each in the basis $1, \zeta_{4,m}$. Both cases produce d_m ciphertexts with slots in $\mathbb{Z}_t$; Pack reverses this arrangement.

Thus $\mathbf{ct} \vDash_{m,q} \mathsf{CtS}_m(P)$ implies $\mathbf{ct} \vDash^{\mathsf{cpack}}_{m,q} P$. Below, CtS_m and StC_m denote the ideal maps; their CKKS implementations are approximate.

2.4 Conventional Bootstrapping: A Semantic View

Bootstrapping refreshes a ciphertext $\mathbf{ct}_{\mathrm{in}} \vDash_{N,q} P$ whose homomorphic capacity is nearly exhausted. Conventional CKKS and BGV, and the BFV procedures considered here, follow four stages [9,24,39]. Let $Q_R, Q_1, Q_2, Q_{\mathrm{out}}$ denote their

successive output moduli. In BGV/BFV, τ is the number of removed p-ary digits; $\vDash$ and the packing maps use plaintext modulus $p^{e+\tau}$ between capacity raising and digit removal, and p^e thereafter.

(1) Capacity raising. $\mathsf{ModRaise}$ produces $\mathbf{ct}_R \vDash_{N,Q_R} T$ at a larger working modulus. For CKKS, $T = P + qI$, with $I \in \mathcal{R}_{\mathbb{Z},N}$. For BGV/BFV with input plaintext modulus p^e, $T = p^\tau P + I \in \mathcal{R}_{p^{e+\tau},N}$, where $P \in \mathcal{R}_{p^e,N}$ and correctness requires $\|I\|_\infty < p^\tau/2$.

(2) CoeffToSlot. CtS_N places the coefficients of T into slots, producing $\mathbf{ct}_1 \vDash^{\mathsf{cpack}}_{N,Q_1} T$.

(3) Homomorphic rounding. The stage ψ_N applies EvalMod in CKKS or Unpack–digit removal–Pack in BGV/BFV, producing $\mathbf{ct}_2 \vDash^{\mathsf{cpack}}_{N,Q_2} P'$ for a refreshed plaintext P'. CKKS requires $\|\mathsf{slot}_N(P') - \mathsf{slot}_N(P)\|_\infty$ to lie within the prescribed tolerance; BGV/BFV gives $P' = P$ under the correctness condition.

(4) SlotToCoeff. StC_N maps the refreshed coefficient packing back into the top ring, producing $\mathbf{ct}_{\text{out}} \vDash_{N,Q_{\text{out}}} P'$.

Thus the ciphertext semantics are $\mathbf{ct}_{\text{in}} \vDash_{N,q} P \xrightarrow{\mathsf{ModRaise}} \mathbf{ct}_R \vDash_{N,Q_R} T \xrightarrow{\mathsf{CtS}_N} \mathbf{ct}_1 \vDash^{\mathsf{cpack}}_{N,Q_1} T \xrightarrow{\psi_N} \mathbf{ct}_2 \vDash^{\mathsf{cpack}}_{N,Q_2} P' \xrightarrow{\mathsf{StC}_N} \mathbf{ct}_{\text{out}} \vDash_{N,Q_{\text{out}}} P'$.

3 Ring Switching and Slot Recovery

3.1 Ring Switching and the Split Map

Ring switching splits a low-level ciphertext into multiple ciphertexts over a lower-dimensional subring. Throughout this paper, ring switching means key switching followed by Split; conventional SIMD-packed constructions additionally use slot recovery to restore the original slot layout [39]. For $N = kn$ and $Y = X^k$, ring switching takes a top-ring ciphertext $\mathbf{ct} \vDash_{N,q} P(X)$ to leaf-ring ciphertexts $(\mathbf{ct}_r \vDash_{n,q} P_r(Y))_{0 \le r < k}$, where, for $P(X) = \sum_{i=0}^{N-1} a_i X^i$, the leaf plaintexts are defined by $P_r(Y) = \sum_{j=0}^{n-1} a_{r+kj} Y^j$ for $0 \le r < k$. Equivalently, they satisfy $P(X) = \sum_{r=0}^{k-1} X^r P_r(X^k)$. In our cyclotomic setting, ring switching takes two steps:

1. **Key switching.** Switch the secret of $\mathbf{ct}$ from $s \in \mathcal{R}_{\mathbb{Z},N}$ to $s'(X) = s_n(X^k)$ for some $s_n \in \mathcal{R}_{\mathbb{Z},n}$.
2. **Split**[4]**.** For $\mathbf{ct} = (c_0, c_1)$, write $\mathsf{Split}_k(c_i) = (c_{i,r})_{0 \le r < k}$ for $i \in \{0, 1\}$, and form the k leaf ciphertexts $\mathbf{ct}_r = (c_{0,r}, c_{1,r}) \vDash_{n,q} P_r(Y)$.

Here $\mathsf{Split}_k : \mathcal{R}_{q,N} \to \mathcal{R}^k_{q,n}$ is defined by $\mathsf{Split}_k(F) = (F_0, \ldots, F_{k-1})$, where $F_r(Y) \in \mathcal{R}_{q,n}$ are uniquely determined by $F(X) = \sum_{r=0}^{k-1} X^r F_r(X^k)$. Its inverse map is denoted by $\mathsf{Merge}_k : \mathcal{R}^k_{q,n} \to \mathcal{R}_{q,N}$. We use the same notation for these maps over the other coefficient rings. When applied to ciphertexts, these maps act componentwise on the two ciphertext components. If an output under the

[4] Also called break in the original paper.

original secret is required, Merge_k is followed by a key switch from s' back to the original secret. This does not affect the plaintext arguments below.

The security of the subring key-switching step is based on the RLWE problem over the leaf ring, as formalized by the following lemma.

Lemma 1. *[26] If the decisional RLWE problem over $\mathcal{R}_{q,n}$ with secret distribution χ'_s and error distribution χ'_e is hard, then so is the decisional RLWE problem over $\mathcal{R}_{q,N}$ with error distribution $\chi_e = \mathsf{Merge}_k((\chi'_e)^k)$, where the secret is sampled from χ'_s in the embedded copy of $\mathcal{R}_{q,n}$.*

This lemma shows that, although the key-switched secret is sparse in the top ring, its security is inherited from the leaf-ring RLWE problem. Consequently, the leaf ring constrains the available modulus budget. Moving bootstrapping to the smaller ring therefore reduces computational cost, but also limits the attainable output capacity.

3.2 Slot Recovery

This subsection recalls the conventional slot-recovery transform induced by Split. We express it as a linear map across the leaves and characterize its action on each leaf slot. For CKKS this gives an invertible Vandermonde matrix; for BGV/BFV the same description applies when $d_N = d_n$, while the general case is given by a CRT isomorphism.

The embedding $Y \mapsto X^k$ induces the root map $\rho_k : \Omega_N \to \Omega_n$, $\alpha \mapsto \alpha^k$. For BGV/BFV, ρ_k denotes the analogous map on these roots. For $\beta \in \Omega_n$, the fiber $\rho_k^{-1}(\beta) = \{\alpha \in \Omega_N : \alpha^k = \beta\}$ contains k roots. Under our ordering, $\rho_k^{-1}(\omega_{n,j}) = \{\omega_{N,j+bn} : 0 \le b < k\}$ for $0 \le j < n$. If $\mathsf{Split}_k(P) = (P_0, \dots, P_{k-1})$, then $P(\alpha) = \sum_{r=0}^{k-1} \alpha^r P_r(\beta)$ for every $\alpha \in \rho_k^{-1}(\beta)$. Thus, for each β, slot recovery maps $(P_0(\beta), \dots, P_{k-1}(\beta))$ to the evaluation vector $(P(\alpha))_{\alpha \in \rho_k^{-1}(\beta)}$.

Set $\mathbf{P} = (P_0, \dots, P_{k-1})^T$, and let $\mathbf{P}^{\mathrm{rec}} = (P_0^{\mathrm{rec}}, \dots, P_{k-1}^{\mathrm{rec}})^T$ denote the recovered leaf plaintexts. The recovery map is $\mathbf{P}^{\mathrm{rec}} = H\mathbf{P}$, with entries $P_a^{\mathrm{rec}} = \sum_r h_{a,r} P_r$. For CKKS, $H = (h_{a,r}) \in \mathcal{R}_n^{k \times k}$, while for BGV/BFV, $H \in \mathcal{R}_{t,n}^{k \times k}$. Since every $h_{a,r}$ is a leaf plaintext, the action of H in leaf slot j is given by a $k \times k$ matrix H_j. We use SR_k for both this plaintext map and its homomorphic evaluation.

Slot Recovery for CKKS. For $0 \le j < n/2$, let $\beta_j = \omega_{n,j}$ and $\alpha_{j,b} = \omega_{N,j+bn}$ for $0 \le b < k$. Define the recovered slots by

$$\big(\mathsf{slot}_n(P_b^{\mathrm{rec}})\big)_j := \frac{P(\alpha_{j,b})}{\Delta} = \sum_{r=0}^{k-1} \alpha_{j,b}^r \big(\mathsf{slot}_n(P_r)\big)_j, \qquad H_j = \big(\alpha_{j,b}^r\big)_{0 \le b,r < k}.$$

The second equality follows from $\alpha_{j,b}^k = \beta_j$. Since $\mathsf{slot}_n : \mathcal{R}_n \to \mathbb{C}^{n/2}$ is a real-linear isomorphism, these slots determine unique plaintexts P_b^{rec}.

The matrix H_j is an invertible Vandermonde matrix. Moreover, $\alpha_{j,b} = \alpha_{j,0}\zeta_k^b$, so

$$H_j = F_k D_j, \qquad (F_k)_{b,r} = \zeta_k^{br}, \qquad D_j = \mathrm{diag}\big(1, \alpha_{j,0}, \dots, \alpha_{j,0}^{k-1}\big).$$

As j and b vary, the roots $\alpha_{j,b}$ contain exactly one root from each complex-conjugate pair in Ω_N. Hence the values $P(\alpha_{j,b})/\Delta$, after the fixed reordering and the required conjugations, give $\mathsf{slot}_N(P)$.

Slot Recovery for BGV/BFV. Let $j \in [\ell_n]$, and define $\beta_j := \iota_{n,j}^{-1}(Z \bmod F_{n,j}) \in E_{t,n}$. Under the fixed embedding into the splitting extension, we also regard β_j as the corresponding root. Then $\big(\mathsf{slot}_n(P_r)\big)_j = P_r(\beta_j)$. Since the coefficients of P_r lie in $\mathbb{Z}_t$, this value also determines the evaluations of P_r on the Frobenius orbit of β_j.

The reduction of Frobenius from modulus $2N$ to $2n$ gives $d_n \mid d_N$. With $d := d_N/d_n$, the k roots in each fiber split into k/d orbits of $\sigma_N^{d_n}$, each of size d, so $d \mid k$. The corresponding orbit polynomials are pairwise coprime degree-d factors of $X^k - \beta_j$ over $E_{t,n}$, and slot recovery is the resulting CRT isomorphism. When $d_N = d_n$, we have $d = 1$, so the CRT components are linear and $H_j = (\alpha^r)_{\alpha \in \rho_k^{-1}(\beta_j),\, 0 \le r < k}$ is the corresponding Vandermonde matrix. When $d_N > d_n$, each CRT component contributes d coordinates in the basis $1, X, \ldots, X^{d-1}$, which are stored in d recovered leaf slots. Thus H_j is invertible in both cases.

Together, the matrices H_j determine the plaintext transform H. Homomorphically, the a-th output is $\mathbf{ct}_a^{\mathrm{rec}} = \sum_{r=0}^{k-1} h_{a,r}\mathbf{ct}_r$, evaluated using plaintext-ciphertext multiplications in the leaf ring. The BGV/BFV transform is exact modulo t. For CKKS, plaintext encoding and scale management introduce the usual approximation error. Inverse slot recovery is evaluated in the same way using H^{-1}.

4 CKKS Bootstrapping via Ring Switching

4.1 Why Slot Recovery Can Be Removed

Let $\mathsf{Split}_k(P) = (P_r)_{r\in[k]}$, and let $\mathsf{SR}_k((P_r)_{r\in[k]}) = (P_r^{\mathrm{rec}})_{r\in[k]}$ denote slot recovery, whose outputs collectively encode the original top-ring SIMD values. For a fixed slot j, write $\mathbf{x}$ for the unrecovered leaf values and $H_j\mathbf{x}$ for their recovered values. A nonlinear map $f : \mathbb{C} \to \mathbb{C}$ applied coordinatewise need not commute with recovery:

$$H_j^{-1} f^{\oplus k}(H_j\mathbf{x}) \neq f^{\oplus k}(\mathbf{x}).$$

Thus applying f directly to unrecovered leaf slots need not give the intended SIMD computation. However, for CKKS bootstrapping, consider the following pipeline with slot recovery retained:

$$\begin{aligned}
&\mathbf{ct}_{\mathrm{in}} \vDash_{N,q} P \xrightarrow{\mathsf{Split}_k} \big(\mathbf{ct}_r \vDash_{n,q} P_r\big)_{r=0}^{k-1} \xrightarrow{\mathsf{SR}_k} \big(\mathbf{ct}_b^{\mathrm{rec}} \vDash_{n,q} P_b^{\mathrm{rec}}\big)_{b=0}^{k-1} \\
&\xrightarrow{(\mathsf{ModRaise}_n)^{\oplus k}} \big(\mathbf{ct}_{b,R}^{\mathrm{rec}} \vDash_{n,Q_R} T_b^{\mathrm{rec}}\big)_{b=0}^{k-1} \xrightarrow{(\mathsf{CtS}_n)^{\oplus k}} \boxed{\big(\mathbf{ct}_{b,1}^{\mathrm{rec}} \vDash^{\mathsf{cpack}}_{n,Q_1} T_b^{\mathrm{rec}}\big)_{b=0}^{k-1}} \\
&\xrightarrow{(\mathsf{EvalMod}_n)^{\oplus k}} \big(\mathbf{ct}_{b,2}^{\mathrm{rec}} \vDash^{\mathsf{cpack}}_{n,Q_2} P_b^{\prime,\mathrm{rec}}\big)_{b=0}^{k-1} \xrightarrow{(\mathsf{StC}_n)^{\oplus k}} \big(\mathbf{ct}_b^{\prime,\mathrm{rec}} \vDash_{n,Q_{\mathrm{out}}} P_b^{\prime,\mathrm{rec}}\big)_{b=0}^{k-1} \\
&\xrightarrow{\mathsf{SR}_k^{-1}} \big(\mathbf{ct}_r' \vDash_{n,Q_{\mathrm{out}}} P_r'\big)_{r=0}^{k-1} \xrightarrow{\mathsf{Merge}_k} \mathbf{ct}_{\mathrm{out}} \vDash_{N,Q_{\mathrm{out}}} P'.
\end{aligned}$$

As shown in the boxed part of the formula, CtS_n has placed the coefficients of each raised leaf plaintext into slots. Within its valid approximation range, $\mathsf{EvalMod}_n$ approximately removes the modulus-raising corrections coefficientwise, and StC_n returns the refreshed coefficients to polynomial form. This coefficientwise refresh also applies to unrecovered leaves. Split partitions the plaintext coefficients and Merge restores their original positions, so both SR_k and SR_k^{-1} can be omitted. Section 4.2 formalizes this coefficientwise identity and the transfer of the assumed error bounds.

Section 6 studies fusion with CtS or StC when slot recovery is required, including the capacity savings and remaining synchronization cost.

4.2 Correctness and Error Preservation

We establish the coefficientwise refresh identity and an error-transfer bound for independent leaf refresh followed by Merge. The underlying bootstrapper's approximation behavior and the error budgets below are taken as given. We do not derive circuit-specific noise or failure-probability bounds here, as these depend on the chosen bootstrapper and its parameters; see, e.g., [44,10].

For $m \in \{N, n\}$ and $A(Z) = \sum_i b_i Z^i$, define $\mathcal{F}_m(A) := \sum_i f_q(b_i) Z^i$, using the same scalar refresh function $f_q : \mathbb{R} \to \mathbb{R}$ in both dimensions. This is the ideal CtS–EvalMod–StC map, with public scaling included in f_q. Write $\|A\|_{2,\mathsf{coeff}} := (\sum_i |b_i|^2)^{1/2}$ and $\|(A_r)_r\|_{2,\oplus} := (\sum_r \|A_r\|_{2,\mathsf{coeff}}^2)^{1/2}$. For $A \in \mathcal{R}_N$, write $\mathsf{Split}_k(A) = (A_r)_{r\in[k]}$.

Lemma 2 (Split and coefficientwise refresh). *For $A \in \mathcal{R}_N$ with leaves A_r, Split preserves the coefficient norm, $\|\mathsf{Split}_k(A)\|_{2,\oplus} = \|A\|_{2,\mathsf{coeff}}$. If $\mathcal{F}_N$ and $\mathcal{F}_n$ apply the same scalar refresh to each coefficient, then*

$$\mathsf{Merge}_k((\mathcal{F}_n(A_r))_{r\in[k]}) = \mathcal{F}_N(A).$$

Proof. Split partitions the coefficients, preserving their Euclidean norm. Each coefficient is refreshed by the same f_q, and Merge restores its position.

Let $T = P + qI$, with $P, I \in \mathcal{R}_{\mathbb{Z},N}$, be the ideal ModRaise output and $e \in \mathcal{R}_N$ the input error. In top-ring coefficient coordinates, η_{pre} collects real perturbations before scalar refresh (including the initial key switch and ModRaise), while η_{post} collects all remaining output errors, including complex CtS errors propagated through EvalMod.

Proposition 1 (CKKS error transfer). *For an ideal ModRaise output $T = P + qI$ and input error e, suppose the merged output satisfies $P'_{\mathrm{leaf}} = \mathcal{F}_N(T + e + \eta_{\mathrm{pre}}) + \eta_{\mathrm{post}}$, with $\|\eta_{\mathrm{pre}}\|_{2,\mathsf{coeff}} \le B_{\mathrm{pre}}$ and $\|\eta_{\mathrm{post}}\|_{2,\mathsf{coeff}} \le B_{\mathrm{post}}$. If the scalar refresh f_q is differentiable with $|f'_q| \le L_{\mathcal{I}} < \infty$ on $\mathcal{I} \subseteq \mathbb{R}$, and $\mathcal{I}$ contains the intervals between corresponding coefficients of T and $T + e + \eta_{\mathrm{pre}}$, then*

$$\|P'_{\mathrm{leaf}} - P\|_{2,\mathsf{coeff}} \le \|\mathcal{F}_N(T) - P\|_{2,\mathsf{coeff}} + L_{\mathcal{I}}(\|e\|_{2,\mathsf{coeff}} + B_{\mathrm{pre}}) + B_{\mathrm{post}}.$$

Proof. Add and subtract $\mathcal{F}_N(T)$ in $P'_{\mathrm{leaf}} - P$, then apply the mean value theorem coefficientwise and the triangle inequality.

The three terms bound the reference error, amplified pre-refresh error, and remaining evaluation error. The proposition transfers the assumed error bounds to the merged output; Split and Merge introduce no additional coefficient-norm amplification. For an output error E at scale Δ_{out}, the mean absolute error over the N real and imaginary CKKS components is at most $\|E\|_{2,\mathsf{coeff}}/(\sqrt{2}\,\Delta_{\mathrm{out}})$, by Parseval and Cauchy–Schwarz. Section B gives the corresponding budget composition and a polynomial derivative bound.

4.3 EvalMod Cost and Key Size Reduction

Before the embedding key switch, $\mathsf{ModRaise}$ uses s; between key switching and Split, it uses $s' = s_n(X^k)$; after Split, it uses s_n. The latter two placements give identical leaf plaintexts for the same modulus transition and coefficient representatives.

For uniform ternary secrets in dimension $m \in \{n, N\}$, the weight h_m has mean $2m/3$. Conditioned on h_m, correction coefficients are modeled as centered Gaussians with variance $(h_m + 1)/12$ [39]. With matched tail targets for secret weight and correction range over all N coefficients, the leaf and top-ring range estimates K_n, K_N satisfy $K_n/K_N \approx k^{-1/2}$. A smaller range can lower EvalMod degree and hence depth or runtime, or improve approximation for a fixed circuit [9,10]. Other secret distributions require their own weight bounds.

All leaves share s_n and bootstrap parameters, so one evaluation-key set evk_n suffices:

$$\mathbf{ct}'_r = \mathsf{BTS}_n(\mathbf{ct}_r; \mathsf{evk}_n) \quad (r \in [k]), \qquad S_{\mathrm{ours}} = |\mathsf{evk}_n| + S_{\mathrm{extra}}.$$

Here $|\mathsf{evk}_m|$ denotes serialized key size, and S_{extra} counts additional top-ring and switching material. At matched counts of distinct keys, decomposition sizes, and numbers of RNS moduli, the dominant storage scales linearly with dimension, giving $|\mathsf{evk}_n| \approx |\mathsf{evk}_N|/k$.

5 BGV/BFV Bootstrapping via Ring Switching

We establish that coefficientwise digit removal commutes with Split and Merge, then study digit-removal cost, key size, and the reduction from general to thin leaf bootstrapping. We assume the homomorphic stages satisfy their usual decryption correctness conditions. We do not derive circuit-specific noise bounds here; see, e.g., [24]. The structural argument also applies to BFV procedures when their capacity increase uses $p^\tau P + I$.

5.1 Coefficientwise Correctness

Let p be an odd prime and $e, \tau \geq 1$. For $m \in \{N, n\}$, use CtS and Unpack modulo $p^{e+\tau}$, and Pack and StC modulo p^e. Write $\langle\cdot\rangle_u$ for coefficientwise centered

reduction modulo u, and define $\mathsf{DR}_{e,\tau}(z) := (z - \langle z \rangle_{p^\tau})/p^\tau \bmod p^e$. Let $\mathcal{D}_m$ be the ideal CtS–Unpack–digit removal–Pack–StC map using this scalar rule on the m exposed coefficients.

For the following lemma, assume that CtS/Unpack and Pack/StC use compatible Hensel lifts, factor isomorphisms, bases, and coordinate order.

Lemma 3 (Coefficientwise digit removal). *Under these compatibility assumptions, the ideal refresh $\mathcal{D}_m$ applies digit removal to each coefficient and commutes with Split and Merge:*

$$\mathcal{D}_m\left(\sum_i z_i Z^i\right) = \sum_i \mathsf{DR}_{e,\tau}(z_i) Z^i, \qquad \mathsf{Merge}_k \circ \mathcal{D}_n^{\oplus k} \circ \mathsf{Split}_k = \mathcal{D}_N.$$

Proof. CtS and Unpack expose each coefficient; Pack and StC restore its position. Split and Merge only redistribute coefficients, so the identity does not require $d_N = d_n$ or identical slot orderings.

Let $P \in \mathcal{R}_{p^e,N}$, $\mathsf{Split}_k(P) = (P_r)_r$, and suppose the raised leaf plaintexts have the form $T_r = p^\tau P_r + I_r$. Since $\mathsf{DR}_{e,\tau}(p^\tau a + \eta) = a$ in $\mathbb{Z}_{p^e}$ whenever $|\eta| < p^\tau/2$, the bound $\max_r \|I_r\|_\infty < p^\tau/2$ gives $\mathsf{Merge}_k((\mathcal{D}_n(T_r))_r) = P$. This yields exact plaintext recovery under the assumed decryption-correctness conditions for all homomorphic stages, including the return key switch.

5.2 Digit-Removal Cost and Key Size Reduction

For an unpacked input $w = p^\tau a + \eta \in \mathbb{Z}_{p^{e+\tau}}$, with $a \in \mathbb{Z}_{p^e}$ and $|\eta| < K < p^\tau/2$, null-polynomial methods use $O\left(\sqrt{(e+\tau)K/\tau}\right)$ ciphertext-ciphertext multiplications when $2K - 1 < p$ [37,39]. Under the uniform ternary model and matched tail targets of Section 4.3, the $\sqrt{K}$ factor scales by approximately $k^{-1/4}$ for fixed e, τ; capacity savings depend on the circuit. This scaling need not hold for fixed-weight secrets, but smaller-ring arithmetic and thin leaf circuits can still reduce cost (Section 5.3).

The same leaf key sharing applies to BGV/BFV. For matched counts of distinct keys and RNS moduli, using c_m decomposition columns in dimension m gives $|\mathsf{evk}_n|/|\mathsf{evk}_N| \approx c_n/(kc_N)$ for the dominant key material. More columns can therefore offset storage savings; Section 8.2 explains the security constraint, and Section F.2 reports the measured components.

5.3 General and Thin Bootstrapping

For plaintext modulus $t = p^e$, general bootstrapping supports arbitrary slots in the degree-d_m slot ring $E_{t,m}$. Thin bootstrapping restricts slots to its scalar subring $\mathbb{Z}_t$, requiring one digit-removal ciphertext rather than d_m [14]. Ring switching connects these cases as follows.

Let $P \in \mathcal{R}_{t,N}$, write $\mathsf{Split}_k(P) = (P_r)_{r \in [k]}$, and choose corresponding top- and leaf-slot roots α_j, β_j with $\beta_j = \alpha_j^k$.

Proposition 2 (Slot decomposition under Split). *If $d_N = kd_n$, compatible slot embeddings give $\ell_N = \ell_n$ and*

$$(\mathsf{slot}_N(P))_j = \sum_{r=0}^{k-1} \alpha_j^r (\mathsf{slot}_n(P_r))_j.$$

Moreover, $1, \alpha_j, \ldots, \alpha_j^{k-1}$ is an $E_{t,n}$-basis of $E_{t,N}$. If $d_n = 1$, every leaf admits thin bootstrapping for arbitrary P.

Proof. Split gives the identity. Under $d_N = kd_n$, the fiber CRT in Section A has the single factor $X^k - \beta_j$; its quotient power basis gives the stated basis. If $d_n = 1$, then $E_{t,n} = \mathbb{Z}_t$, so every leaf is thin while the collection retains all coordinates of the general input.

Corollary 1 (Zero leaves for thin inputs). *With compatible slot embeddings and $d_N = kd_n$, a thin top-ring plaintext satisfies $\mathsf{Split}_k(P) = (P_0, 0, \ldots, 0)$, with P_0 thin. Only P_0 needs refreshing.*

A thin slot lies in $\mathbb{Z}_t \subseteq E_{t,n}$, so uniqueness of basis coordinates makes all but the zeroth coordinate vanish. In our BGV setting, $p = N+1 = 65537$, $t = p$, and $k = 2$, so $d_N = 2$, $d_n = 1$. A general slot $a_j + b_j\alpha_j$ splits into (a_j, b_j), preserving both scalar coordinates while making Unpack/Pack trivial [29]. A thin slot has $b_j = 0$, so only the first leaf needs bootstrapping.

6 Fusing Slot Recovery with Linear Transformations

Having shown that slot recovery can be omitted for the bootstrapping constructions above, we ask how much of its cost can be removed when a subsequent computation or existing pipeline still requires the original slot layout. For two consecutive linear transforms represented by matrices A and B, one can ordinarily precompute $C = AB$ and evaluate C directly [6]. This suggests asking whether slot recovery can likewise be absorbed into a neighboring CtS or StC transform.

Slot recovery is less straightforward because CtS and StC act independently on the leaves with block-diagonal form $L^{\oplus k}$, whereas recovery couples leaves. Absorbing it can therefore change the sparsity, BSGS structure, rotations, and key requirements, and in the CtS-first order requires moving forward recovery across $\mathsf{ModRaise}$. Let $H = \mathsf{SR}_k$ be the recovery map from Section 3.2. We characterize this obstruction for general k, and show for $k = 2$ how inverse recovery can be absorbed into the final StC factor.

6.1 Fusing Inverse Recovery

For $k = 2$, let $h \in \mathcal{R}_n$ satisfy $h(\beta_j) = \alpha_{j,0}$. Write $\mathsf{StC}_n = A_d \circ W$, with A_d the final grouped factor and W the preceding factors. For $\mathsf{EvalMod}$ outputs z_0, z_1, set

$$V_0 = A_d\left(\frac{Wz_0 + Wz_1}{2}\right), \qquad V_1 = h^{-1}A_d\left(\frac{Wz_0 - Wz_1}{2}\right).$$

Linearity gives $(V_0, V_1) = H^{-1}(\mathsf{StC}_n(z_0), \mathsf{StC}_n(z_1))$. The leaves evaluate W independently, then synchronize to form the sum and difference before applying the final factors concurrently.

Since $1/2$ and $h^{-1}/2$ are plaintext diagonal multipliers, they can be folded into the diagonals of the final StC factors. The fused circuit therefore retains the same StC factorization, diagonal support, rotations, and keys, while avoiding two standalone recovery multiplications and their rescale calls. This saves one level without changing EvalMod, but still requires cross-leaf synchronization before the final StC factors. Without recovery, each leaf completes StC independently before Merge.

The explicit construction in this subsection is for $k = 2$; Section D gives the corresponding structural identity for arbitrary power-of-two k and records the additional cross-leaf mixing that it entails.

6.2 Why Forward Recovery Cannot Generally Cross ModRaise

In the CtS-first order, fusing forward recovery would require moving H across $\mathsf{ModRaise}$. Let $\Lambda = (\mathcal{R}_{\mathbb{Z},n})^k$, and define $\mathrm{Red}_q(x) := x - q\lfloor x/q + 1/2 \rfloor$, with values in $[-q/2, q/2)$, extended coefficientwise to polynomials and leaf vectors. For a lifted input $P + qI$, the reordered circuit applies $H^{-1}\,\mathrm{Red}_q(H(P + qI))$. After applying H, the correction becomes qHI. It remains removable modulo q only if HI still has integer coefficients, which is the correction-lattice condition below.

Lemma 4 (Criterion for moving $\mathsf{ModRaise}$). *Let H be an invertible map linear over the plaintext ring. For $P \in \mathcal{R}_n^k$, let $S \subseteq \Lambda$ be the set of possible corrections and assume that all coefficients of HP lie in $(-q/2, q/2)$. Then $H^{-1}\,\mathrm{Red}_q(H(P + qI)) = P$ for every such P and every $I \in S$ if and only if $H(S) \subseteq \Lambda$.*

Split preserves the integer correction lattice by rearranging coefficients. The linear recovery map need not preserve this lattice:

Proposition 3 (Recovery does not preserve the correction lattice). *Let $n, k \geq 2$ be powers of two. Then CKKS slot recovery satisfies $H(\Lambda) \not\subseteq \Lambda$.*

Consequently, in the CtS-first ordering, forward fusion generally cannot cross $\mathsf{ModRaise}$: modular reduction can no longer remove the correction as a multiple of q. Proofs of both statements and the explicit counterexamples are given in Section C. In an StC-first ordering, forward recovery is adjacent to the initial transform and can be fused there, but it still requires an input level and leaves inverse recovery as a separate operation.

7 Independent Leaf Evaluation

Beyond bootstrapping, we study which plaintext computations can be performed after ring switching without combining information from different leaves. We first

model this problem in Lemma 5, then study general linear maps and ring automorphisms with leaf permutations allowed. We finally classify continuous slotwise functions and use ReLU, a common nonlinear activation in neural networks, to analyze independent leaf evaluation when approximation error is allowed.

Let $\mathcal{D} \subseteq \mathcal{R}_N$ be the set of reachable plaintexts and $J \subseteq [k]$ the required output branches, allowing partial outputs as in PaCo [19]. For $\mathcal{F} : \mathcal{D} \to \mathcal{R}_N$, write $\pi_r(P) = P_r := (\mathsf{Split}_k(P))_r$ and $\mathcal{F}_r := \pi_r \circ \mathcal{F}$. We ask when $\mathcal{F}_r(P)$ is determined by P_r alone. The plaintext-level criterion and the linear-map and automorphism results also apply to BGV/BFV over $\mathcal{R}_{t,N}, \mathcal{R}_{t,n}$. The results below on slotwise functions and ReLU are stated for CKKS.

Lemma 5 (Independent leaf evaluation). *For every $r \in J$, a map $G_r : \pi_r(\mathcal{D}) \to \mathcal{R}_n$ satisfying $\mathcal{F}_r(P) = G_r(P_r)$ exists if and only if $P_r = Q_r$ implies $\mathcal{F}_r(P) = \mathcal{F}_r(Q)$ for all $P, Q \in \mathcal{D}$. For $J = [k]$, merging these outputs gives $\mathcal{F}(P)$.*

Proof. Necessity is immediate. Conversely, for $A \in \pi_r(\mathcal{D})$, choose any $P \in \mathcal{D}$ with $P_r = A$ and define $G_r(A) := \mathcal{F}_r(P)$. The implication makes G_r well defined and gives $\mathcal{F}_r(P) = G_r(P_r)$. The final statement follows by applying Merge_k.

These conditions characterize independent leaf evaluation at the plaintext level. We next ask what changes if a public permutation of the leaves is allowed. Such a permutation only reorders the leaf ciphertexts and requires no homomorphic cross-leaf transform. For the CKKS slotwise results below, let $\mathcal{D}_{\mathbb{R}} := \{P \in \mathcal{R}_N : \mathsf{slot}_N(P) \in \mathbb{R}^{N/2}\}$. For a continuous $\phi : \mathbb{R} \to \mathbb{R}$, define $\Phi_{\phi,N}(P) := \mathsf{slot}_N^{-1}\big(\phi^{\oplus N/2}(\mathsf{slot}_N(P))\big)$ on this domain.

For a public assignment $\mu : J \to [k]$, the criterion becomes: $\mathcal{F}_r(P)$ must depend only on $P_{\mu(r)}$. When $J = [k]$ and $\mu = \tau^{-1}$ for a permutation τ, set $A_s := G_{\tau(s)}$ and define $(\mathsf{Perm}_\tau \boldsymbol{Z})_{\tau(s)} = Z_s$ for $\boldsymbol{Z} = (Z_0, \ldots, Z_{k-1})$. Then

$$\mathcal{F} = \mathsf{Merge}_k \circ \mathsf{Perm}_\tau \circ \left(\bigoplus_{s=0}^{k-1} A_s \right) \circ \mathsf{Split}_k.$$

Ring automorphisms admit this form when their induced leaf permutation is allowed. Proofs of the next two propositions appear in Section E.1.

Proposition 4 (Linear maps). *Let $L : \mathcal{R}_N \to \mathcal{R}_N$ be linear and write $\mathsf{Split}_k \circ L \circ \mathsf{Merge}_k = (L_{rs})_{r,s\in[k]}$, where $L_{rs} : \mathcal{R}_n \to \mathcal{R}_n$. On the full plaintext space, output branch r depends only on $P_{\mu(r)}$ if and only if $L_{rs} = 0$ for every $s \neq \mu(r)$.*

When $J = [k]$ and μ is a permutation, the map is block diagonal after permuting the output leaves.

Proposition 5 (Ring automorphisms). *Let $a \in [2N]$ be coprime to $2N$, and define $\sigma_a(P)(X) = P(X^a)$. For $r \in [k]$, write $ar = \tau_a(r) + ku_r$ with $\tau_a(r) \in [k]$, and define $A_{a,r}(P_r) := Y^{u_r} P_r(Y^a)$. Then τ_a is a permutation and $\sigma_a = \mathsf{Merge}_k \circ \mathsf{Perm}_{\tau_a} \circ \left(\bigoplus_{r=0}^{k-1} A_{a,r}\right) \circ \mathsf{Split}_k$. Without the public permutation, output branch r depends only on the corresponding input leaf P_r if and only if $(a-1)r \equiv 0 \pmod{k}$.*

Theorem 1 (Independent Leaf Evaluation on Real CKKS Slots). *Assume $n, k \geq 2$ and $J \neq \varnothing$. For every continuous $\phi : \mathbb{R} \to \mathbb{R}$, the induced slotwise map $\Phi_{\phi,N}$ admits independent leaf evaluation on $\mathcal{D}_{\mathbb{R}}$ for the branches in J if and only if $\phi(x) = cx + d$ for some $c, d \in \mathbb{R}$.*

Proof. Fix $r \in J$ and $j \in [n/2]$, and define $S_r(y) := \sum_{b=0}^{k-1} \zeta_k^{-br} y_b$. Since $H_j = F_k D_j$, we have $(H_j^{-1} z)_r = \alpha_{j,0}^{-r} S_r(z)/k$. For any $y, y' \in \mathbb{R}^k$ with $S_r(y) = S_r(y')$, surjectivity of $\mathsf{slot}_N : \mathcal{D}_{\mathbb{R}} \to \mathbb{R}^{N/2}$ gives plaintexts $P, Q \in \mathcal{D}_{\mathbb{R}}$ whose j-th fibers are y, y' and whose other representative fibers agree. The preceding identity then gives $P_r = Q_r$. By Lemma 5, their r-th output leaves are equal, and applying the same identity to the outputs gives $S_r(\phi^{\oplus k}(y)) = S_r(\phi^{\oplus k}(y'))$.

Set $\sigma = (-1)^r$, so $\zeta_k^{-r(k/2)} = \sigma$. Comparing $y = ue_0 + ve_{k/2}$ with $y' = (u + \sigma v)e_0$ gives equal S_r-values and hence $g(u) + \sigma g(v) = g(u + \sigma v)$, where $g = \phi - \phi(0)$. Setting $u = 0$ gives $g(\sigma v) = \sigma g(v)$, so replacing v by σv proves additivity. Continuity then gives $g(u) = cu$, so $\phi(u) = cu + d$.

Conversely, if $\phi(u) = cu + d$, then $H_j^{-1} \phi^{\oplus k}(H_j x) = cx + de_0$, since $H_j e_0 = \mathbf{1}$. Thus every output leaf is determined by the corresponding input leaf, proving independent leaf evaluation.

The preceding characterization concerns functions applied to the original SIMD slots. The corresponding result for complex CKKS slots is given in Theorem 2, while the discrete BGV/BFV analogue is given in Theorem 3. Bootstrapping remains compatible because it realizes the identity on the original slots while refreshing the coefficients exposed by CtS.

We next examine the limitation for a nonlinear slotwise map. For real slots, ReLU gives an approximation lower bound for independent leaf evaluation on bounded inputs. Write $\Phi_{\mathrm{ReLU},N}$ for the choice $\phi = \mathrm{ReLU}$, and set $\Phi_{\mathrm{ReLU},N,r} := \pi_r \circ \Phi_{\mathrm{ReLU},N}$. The following proposition identifies the optimal worst-case error on $\mathcal{D}_B$, even with arbitrary nonlinear leaf maps and arbitrary k; Section E.4 gives the proof and a restricted domain on which exact independent evaluation is possible.

Proposition 6 (Optimal ReLU approximation). *Assume $n, k \geq 2$ and $B > 0$, and set $\mathcal{D}_B := \{P \in \mathcal{D}_{\mathbb{R}} : \|\mathsf{slot}_N(P)\|_\infty \leq B\}$. Let $\mathcal{G}$ range over maps $P \mapsto \mathsf{Merge}_k((G_r(P_r))_{r\in[k]})$, with arbitrary $G_r : \pi_r(\mathcal{D}_B) \to \mathcal{R}_n$. Then*

$$\inf_{\mathcal{G}} \sup_{P \in \mathcal{D}_B} \left\| \mathsf{slot}_N(\mathcal{G}(P)) - \mathsf{slot}_N(\Phi_{\mathrm{ReLU},N}(P)) \right\|_\infty = \frac{B}{4}.$$

The optimum is attained by $G_0(A) = A/2 + \Delta B/4$ and $G_r(A) = A/2$ for $r > 0$, where $\Delta B/4$ is a constant polynomial.

Thus, on this domain, even nonlinear leaf maps cannot improve on the affine approximation $x/2 + B/4$ in worst-case error.

8 Security Analysis and Parameter Selection

We use the standard ring-switching security analysis [26] and the secret choices and assumptions of the underlying bootstrappers [10,39]. We do not revisit these security analyses. This section examines how leaf-ring security constrains the modulus budget, bootstrapping parameters, and output capacity.

The main secrets define two RLWE instances: the top ring with the original secret s, and the leaf ring with secret s_n. Although $s'(X) = s_n(X^k)$ is represented in the top ring, Lemma 1 shows that its security is governed by the leaf-ring RLWE instance. The k leaf ciphertexts share s_n and thus provide multiple samples from a single leaf-ring instance, rather than k independent assumptions. The switching keys in both directions are treated under the circular-security assumption commonly used for FHE evaluation keys, following [10]; see also [13] for KDM and circular security.

Let λ_{top} and λ_{leaf} denote the main-secret estimates, with $\lambda_{\text{est}} = \min\{\lambda_{\text{top}}, \lambda_{\text{leaf}}\}$. Both use the estimator's secret and error models in Section 9 and the largest relevant modulus appearing in ciphertexts or evaluation keys. Any additional sparse or subring secrets must also satisfy the underlying bootstrapper's security conditions. The leaf-ring instance typically imposes the tighter constraint in our parameter sets, so the ring dimensions, modulus budget, key-switching parameters, and CtS/StC partitions must be selected jointly.

8.1 Ring Dimensions and Modulus Budget

Parameter selection begins with the application requirements. For CKKS, these are the required number of slots, output levels, precision, and failure probability. For BGV/BFV, they are the number of slots, plaintext modulus, remaining capacity, and correctness condition. When only a small output depth is needed, minimal-level CKKS bootstrapping avoids paying for unused output levels [35].

We choose N to meet the slot requirement and top-ring security target, then choose k and set $n = N/k$. Increasing k gives smaller leaf rings and more leaves to run in parallel, but reduces the modulus budget at the target security level, limiting output levels or capacity. For the evaluated circuits and output requirements, $k = 2$ is the default and $k = 4$ is used mainly for minimal-level CKKS. Here, $n = 2^{15}$ is near the smallest practical leaf dimension; at $n = 2^{14}$, the modulus budget is generally insufficient.

We next determine Q_R, the largest ciphertext modulus used during leaf bootstrapping. For CKKS, we work backward from Q_{out} through the primes consumed by StC, EvalMod, and CtS. For BGV/BFV, whose remaining capacity is not determined by fixed stage losses, we search for parameters near the security limit. With P_{KS} denoting the auxiliary evaluation-key modulus, the leaf-ring instance with the total modulus $P_{\text{KS}}Q_R$ usually gives the tighter RLWE estimate.

8.2 Key-Switching Parameters

Consider a key switch from a source secret s' to a target secret s at ciphertext modulus Q. Its main parameters are the auxiliary modulus P_{KS} and the number c of RNS decomposition columns.

Auxiliary modulus P_{KS}. For the j-th decomposition component, a switching-key sample satisfies $b_j + a_j s = P_{\mathrm{KS}} B_j s' + \varepsilon_j \pmod{P_{\mathrm{KS}} Q}$, where B_j is a public decomposition weight. ModDown removes P_{KS} after key switching and returns the ciphertext to modulus Q. Increasing P_{KS} suppresses switching-key noise after ModDown, while rounding error remains. It also increases the modulus entering the security estimate, key size, and computation. In our implementations, P_{KS} consists of special RNS primes and is not part of the ciphertext Q-chain or consumed by CKKS rescaling [15,30].

RNS decomposition columns. With $u = \sum_{j=0}^{c-1} B_j u_j$, the error before ModDown contains $\sum_{j=0}^{c-1} u_j \varepsilon_j$. Increasing c reduces digit sizes and typically lowers this error, but increases key size, key-generation time, and key-switching cost [28]. The leaf ring's smaller budget for $P_{\mathrm{KS}} Q$ may require reducing P_{KS} and increasing c to maintain the required error bound. Thus, P_{KS} and c jointly trade switching error against security, key size, and computation.

8.3 Linear Transform Parameters

Let A be the CtS or StC matrix (the FFT core for BGV/BFV), and write $A = D_{L-1} \cdots D_1 D_0$ for its sparse factorization [9,23]. Given $0 = r_0 < r_1 < \cdots < r_d = L$, define $A_i := D_{r_i - 1} \cdots D_{r_{i-1}}$ for $1 \le i \le d$, so that $A = A_d \cdots A_1$. Using more groups typically gives sparser matrices but increases sequential multiplication depth. This can reduce transform time at the cost of more CKKS levels; the effect on BGV capacity depends on the evaluation parameters. Since the leaf ring has a tighter modulus budget, partitions with more groups may be infeasible even when faster.

Operation counts and costs. We evaluate grouped matrices using the BSGS diagonal method [27]. For a selected factorization and partition over s slots, let M_s and R_s denote the total plaintext-ciphertext multiplication and rotation counts. If all nonzero diagonals lie in $0, \ldots, D-1$, the choices $g = \lceil \sqrt{D} \rceil$ and $h = \lceil D/g \rceil$ give at most D multiplications and $g + h - 2$ rotations; the full identity and concrete Group III counts are given in Section F.1.

At fixed modulus size and switching parameters, rotations cost $O(m \log m)$ and multiplications by precomputed plaintexts cost $O(m)$ in dimension m. The transform cost is therefore $O(m(R_s \log m + M_s))$.

For CKKS, ring switching reduces the slot count from $N/2$ to $n/2$. For BGV/BFV, $\ell_n/\ell_N = d_N/(kd_n)$: equal slot degrees reduce the count by k, whereas $d_N = kd_n$ preserves it. In the latter case, set $\nu_m = d_m$ for $p \equiv 1 \pmod 4$ and $\nu_m = d_m/2$ for $p \equiv 3 \pmod 4$. With compatible roots $\beta_i = \alpha_i^k$, the FFT cores satisfy $(U_N)_{i,j} = \alpha_i^{\nu_N j} = \beta_i^{\nu_n j} = (U_n)_{i,j}$. Their BSGS counts therefore

agree under matched orderings, factorizations, and partitions. Changes between the common and local root bases within each slot are accounted for separately.

Table 1: Complexity of the CtS/StC FFT cores, comparing the top ring with one leaf ring.

Case	Number of slots	Cost of linear transform
CKKS	$N/2 \to n/2$	$O\big(N(R_{N/2}\log N + M_{N/2})\big) \to O\big(n(R_{n/2}\log n + M_{n/2})\big)$
BGV/BFV, $d_N = d_n$	$\ell_N \to \ell_N/k$	$O(N(R_{\ell_N}\log N + M_{\ell_N})) \to O\big(n(R_{\ell_N/k}\log n + M_{\ell_N/k})\big)$
BGV/BFV, $d_N = kd_n$	$\ell_N \to \ell_N$	$O(N(R_{\ell_N}\log N + M_{\ell_N})) \to O(n(R_{\ell_N}\log n + M_{\ell_N}))$

The table compares one top-ring FFT core with one leaf core. Ring switching reduces the ring dimension and can also reduce operation counts for CKKS or BGV/BFV with $d_N = d_n$. When $d_N = kd_n$, only the ring dimension decreases: for fixed k, all leaf cores together have the same asymptotic cost as the top-ring core but can run concurrently. This excludes basis conversions, Unpack/Pack savings, and skipped leaves. Homomorphic NTT provides further acceleration for power-of-two BGV [38].

For the CKKS and BGV experiments below, we select n to support the required $P_{\mathrm{KS}}Q_R$ at the target security level. We jointly choose P_{KS}, c, and the CtS/StC partitions to balance switching error, transform cost, and output capacity.

9 Implementation

The performance experiments in this section were conducted on a machine with two Intel Xeon E5-2667 v4 CPUs at 3.20 GHz, 1.5 TiB of DDR4 memory, and Ubuntu 22.04.5 LTS.

For CKKS and BGV, we estimate the security of top- and leaf-ring main-secret RLWE instances using the Lattice Estimator [2,8] at commit `27a581bb8`, with uniform ternary secrets, the full RLWE dimension, and an unbounded number of samples. For each main-secret instance, we use $q_{\mathrm{est}} = 2^{\lceil \log_2(P_{\mathrm{KS}}Q_R)\rceil}$ as the estimator modulus. We report the minimum estimate over primal-uSVP [3], primal-BDD [34], and dual-hybrid attacks under the `RC.MATZOV` cost model [40].

9.1 CKKS Bootstrapping

We implemented ring-switched CKKS bootstrapping in Lattigo v6 at commit `b98d89155` [20]. For security estimation, the error is modeled as a discrete Gaussian with standard deviation $\sigma = 3.19$.

In the main and dense-key CKKS experiments, each bootstrap uses one worker without internal parallelism: Direct runs one top-ring bootstrap, while Ours runs k leaf bootstraps concurrently. After one warm-up per configuration, we report arithmetic means over ten rounds with alternating execution order.

Parameter settings. Let L_{out} denote the number of remaining levels after bootstrapping; each pair in Table 2 lists Direct followed by Ours. Groups I–II retain

Table 2: Main CKKS parameters. The RLWE row reports the minimum, in bits, among the relevant top- and leaf-ring estimates for each parameter pair.

Metric	**Group I**		**Group II**		**Group III**							
	$L_{\text{out}} = 3$		$L_{\text{out}} = 3$		$L_{\text{out}} = 5$		$L_{\text{out}} = 8$		$L_{\text{out}} = 12$		$L_{\text{out}} = 15$	
$\log N$	16	16	17	17	17	17	17	17	17	17	17	17
$\log n$	16	15	17	15	17	16	17	16	17	16	17	16
$k = N/n$	1	2	1	4	1	2	1	2	1	2	1	2
$\log \Delta$	35	31	35	31	35	35	35	35	35	35	35	35
$\log(P_{\text{KS}} Q_R)$	1276	849	1276	849	1346	1346	1451	1451	1591	1591	1696	1696
RLWE est.	130.82		130.82		169.03		155.73		141.25		132.08	

three levels for short post-bootstrap computations without paying for unused levels [35,18], while Group III considers $L_{\text{out}} \in \{5, 8, 12, 15\}$. Group I evaluates $N = 2^{16}$ with the smallest leaf ring $n = 2^{15}$. Under the target leaf-ring security level, the available modulus budget after accounting for the P_{KS} required by Split and Merge is insufficient for three output levels at $\log \Delta = 35$, so we use $\log \Delta = 31$.

For sparse-secret encapsulation, we follow the parameterization of the underlying bootstrapper [10]. Let $\delta_{\text{range}} := \Pr[\max_r \|I_r\|_\infty > K]$, where K is the configured correction bound and r ranges over all refreshed paths (one for Direct). We use $\delta^{(1)}_{\text{range}}$ for a single leaf and report model-based tail estimates. With the default $K = 16$, both Direct and Ours have estimated $\log_2 \delta_{\text{range}} \approx -138.7$ for $N = 2^{16}$ and -137.7 for $N = 2^{17}$. These estimates concern correction range; the tables below report measured precision for the benchmark configurations.

For applications already requiring $N = 2^{17}$, Group II explores aggressive dimension reduction with $2^{17} \to 2^{15}$ and four concurrent leaves, while Group III uses $2^{17} \to 2^{16}$ with two leaves to support higher-capacity settings. The largest setting $L_{\text{out}} = 15$ approaches the roughly 16 output levels supported by direct bootstrapping at this scale.

We use a fixed vector of $S = N/2$ input slots with real and imaginary parts in $[-1, 1]$. Let (x_i, y_i) and $(\widehat{x}_i, \widehat{y}_i)$ denote the decoded slot components before and after bootstrapping. We report the measured precision $-\log_2 \epsilon$ of each run, where $\epsilon = \frac{1}{2S} \sum_{i=0}^{S-1} (|x_i - \widehat{x}_i| + |y_i - \widehat{y}_i|)$. Following [9], we compute per-run throughput as

$$\text{Throughput (Mbps)} = \frac{S L_{\text{out}} \log_2 \Delta \, (-\log_2 \epsilon)}{10^6 \cdot \text{Running Time}}.$$

Results. As shown in Table 3, executing CtS, EvalMod, and StC in the leaves yields stage speedups of 1.23×–2.86×, 1.99×–4.27×, and 1.80×–3.61×, respectively, while Split and Merge together account for at most 2.8% of our runtime. Group I achieves a 1.51× latency speedup at $N = 2^{16}$, but its lower scale and precision limit the throughput ratio to 1.05×. This illustrates the precision–capacity tradeoff of aggressive dimension reduction. Moving from Group I to Group II doubles the number of packed slots and increases Direct's runtime

Table 3: CKKS bootstrapping performance. Each pair lists Direct followed by Ours. We use all $N/2$ slots, a uniform ternary main secret, and a Hamming-weight-32 ephemeral secret for sparse-secret encapsulation.

	Group I		**Group II**		**Group III**							
	$L_{\text{out}} = 3$		$L_{\text{out}} = 3$		$L_{\text{out}} = 5$		$L_{\text{out}} = 8$		$L_{\text{out}} = 12$		$L_{\text{out}} = 15$	
	Time (sec)											
Split	–	0.03	–	0.11	–	0.10	–	0.10	–	0.10	–	0.10
ModRaise†	0.84	0.29	1.90	0.29	2.05	1.06	2.59	1.34	3.49	1.75	4.21	2.07
CtS	9.27	7.54	21.22	7.43	23.66	10.19	30.89	13.29	38.67	18.46	52.17	23.87
EvalMod	5.06	2.39	10.57	2.47	12.92	6.35	16.42	8.21	21.92	11.01	26.97	13.32
StC	2.79	1.55	6.19	1.72	8.10	3.50	11.37	5.73	16.22	7.70	21.56	9.81
Merge	–	0.06	–	0.23	–	0.25	–	0.33	–	0.50	–	0.73
Total‡	17.96	11.86	39.88	12.25	46.74	21.46	61.27	29.00	80.29	39.53	104.91	49.91
	Precision (bits)											
	21.96	17.13	20.87	16.19	20.87	20.48	20.87	20.48	20.87	20.47	20.87	20.47
	Throughput (Mbps)											
	4.22	4.41	3.61	8.07	5.13	10.96	6.26	12.96	7.16	14.31	6.85	14.14
	Key Generation (sec)*											
	29.85	25.00	64.19	28.80	85.45	41.40	95.36	46.33	130.46	63.12	167.25	80.85
	Server Key Size (GiB)**											
	4.69	3.92	10.50	4.45	14.22	6.86	15.86	7.65	21.66	10.44	27.57	13.29

† ModRaise includes ScaleDown and ModUp, while Split and Merge include their key switches.
‡ Total time covers online bootstrapping, excluding decryption, verification, and key serialization.
∗ Key-generation time covers the top-ring key pair and all method-specific evaluation keys.
∗∗ Server key size counts serialized evaluation material.

2.22-fold. Ours uses four concurrent leaves instead of two, increasing its time by only 3.3%, from 11.86 to 12.25 seconds.

In Group III, increasing L_{out} increases the output modulus and runtime, while the speedup remains 2.03×–2.18×. Each Direct/Ours pair uses the same slot count, number of output levels, and scale, with about a 0.4-bit precision difference, yielding 2.00×–2.14× Direct's throughput. Overall, $n = 2^{15}$ provides the largest speedup for minimal-level bootstrapping, whereas $n = 2^{16}$ allows a larger modulus budget and supports up to 15 output levels with stable gains. Section F.2 provides a detailed analysis of server-key storage.

Performance attribution. To isolate the sources of our performance gain, we compare five configurations for the final Group III setting in a separate batch (Table 4). D1 and O1 use one worker, with O1 running the two leaves sequentially; D2 uses real/imaginary (RI) parallelism within EvalMod, O2 runs the two leaves concurrently, and O4 combines both forms of parallelism. CtS and StC remain sequential within each bootstrapper: parallelizing Lattigo's linear transforms would require nontrivial implementation changes, with benefits depending on scheduling overhead. Leaf parallelism follows directly from the independent leaf structure.

Table 4: Performance comparison under different parallel configurations. D and O denote Direct and Ours, and the numeral gives the worker count.

Metric	**D1**	**D2**	**O1**	**O2**	**O4**
Bootstrapping paths	1	1	2	2	2
RI parallelism	No	Yes	No	No	Yes
Peak concurrency	1	2	1	2	4
Wall time (sec)	99.111	86.468	87.383	45.881	39.115
CPU-seconds	99.100	100.837	87.374	90.938	90.868
CPU/wall ratio	1.000	1.166	1.000	1.982	2.323
CtS time (sec)	49.131	49.209	40.163	21.190	20.962
CtS CPU/wall	1.000	1.003	1.000	2.000	2.007
EvalMod time (sec)	26.286	13.499	25.237	12.856	6.633
EvalMod CPU/wall	1.000	1.997	1.000	1.998	3.979
Precision (bits)	20.870	20.870	20.474	20.475	20.473
Throughput (Mbps)	7.245	8.305	8.062	15.357	18.012

Each configuration is warmed up once and measured in 10 fresh processes in counterbalanced order, with workers pinned to dedicated cores within the same NUMA node.

These comparisons separate the effects of reduced serial work, leaf parallelism, and parallelism within each bootstrapper. (i) D1 vs. O1: with one worker, O1 reduces wall and CPU time by 11.8%, showing savings from smaller-ring arithmetic and cheaper CtS/StC. (ii) O1 vs. O2: concurrent execution of the same two leaf bootstraps gives a 1.90× speedup with 4.1% more CPU time. (iii) D1 vs. D2: RI parallelism nearly halves EvalMod time, while CtS and StC remain unchanged, isolating the benefit of within-bootstrap parallelism. (iv) D2 vs. O2: with two workers, O2 raises the CPU/wall ratio from 1.166 to 1.982, nearly fully utilizing both workers. D2 parallelizes only the EvalMod branches, whereas O2 runs two complete leaf bootstrappers concurrently, improving CPU utilization with straightforward engineering changes. (v) D2, O2, and O4: adding RI parallelism within O2's concurrent leaves gives O4 a further 1.17× speedup. Both D2 and O4 use RI parallelism; O4 also runs two leaf bootstrappers concurrently. Smaller-ring computation and additional leaf parallelism reduce wall time from 86.468 seconds for D2 (two workers) to 39.115 seconds for O4 (four workers).

Taken together, these comparisons attribute the gain over Direct to reduced serial work and additional leaf concurrency, and show that leaf and RI parallelism compose. Whether a future implementation also parallelizes CtS or StC is orthogonal to this contribution: such optimizations would act within each leaf, while independent leaf execution would remain an additional parallel structure. We next isolate the effect of slot recovery.

Effect of slot recovery. To isolate the cost of slot recovery, we compare the three routes in Table 5 under the final Group III parameters: $N = 2^{17}$, $n = 2^{16}$, $k = 2$, $L_{\text{out}} = 15$, and $\log \Delta = 35$. The routes share the leaf bootstrapping parameters and StC factorization, and use two concurrent leaves. We also report

post-EvalMod time, namely the interval from the end of EvalMod through StC and any recovery, to assess whether recovery materially contributes to runtime.

Table 5: Performance comparison of slot-recovery strategies. Throughput uses output levels minus the required input level in place of L_{out}.

Route	Input	Output	Usable	Precision	Throughput	Post-EvalMod
No recovery	0	15	15	20.476	14.689	8.446
Separate	1	14	13	20.332	12.723	8.547
Fused inverse	1	15	14	20.405	13.779	8.503

Input, Output, and Usable report levels; Precision is in bits, Throughput in Mbps, and post-EvalMod time in seconds.

Precision exceeds 20 bits for all routes and is slightly higher without recovery. Separate recovery loses two usable levels, and fused recovery loses one; this is modest at $L_{\text{out}} = 15$ but consumes a much larger capacity fraction at $L_{\text{out}} = 3$. Consequently, raw throughput comparisons between configurations with and without slot recovery are of limited significance unless their capacity budgets are matched. Their post-EvalMod times differ by about 1.2%. This is expected: for $k = 2$, recovery is only a 2×2 linear map, so its direct cost is small, as reflected in the fusion analysis of Section 6. Fusing recovers one level, raising throughput by 8.3% over separate recovery to 93.8% of no recovery. The approximately $2\times$ gain therefore comes from smaller-ring arithmetic and leaf parallelism, while removing recovery preserves capacity and independent leaf execution.

Compatibility of our method. We also evaluate the dense-key bootstrapper of Ma et al. [39] as the leaf bootstrapper. This method also exploits subring structure and, by the semantic argument in Section 4, can serve as a leaf bootstrapper without slot recovery.

Table 6: Dense-key bootstrapping with subring secret encapsulation [39]. Here N' is the dimension of the subring secret. We use $N = 2^{17}$, $L_{\text{out}} = 15$, $\log \Delta = 35$, $\log n = 17$ for Direct and 16 for Ours, $\log N' = 12$, and $\log(P_{\text{KS}} Q_R) = 1730$.

Method	n/N'	Time (s)	$-\log_2 \epsilon$	Throughput	Speedup	Gain
Direct	32	139.99	18.56	4.57Mbps	–	–
Ours	16	66.00	19.38	10.11Mbps	2.12×	2.21×

With this bootstrapper, ring switching gives a 2.12× speedup and 2.21× Direct's throughput; measured precision rises from 18.56 to 19.38 bits. With $K = 102$, the single-leaf estimate is $\log_2 \delta_{\text{range}}^{(1)} \approx -15.82$ [39]; a union bound over the two leaves yields the same estimate $\log_2 \delta_{\text{range}} \approx -14.82$ as for Direct. This compatibility experiment uses the underlying bootstrapper's configured correction bound.

9.2 BGV Bootstrapping

We implemented ring switching in HElib v2.2.0 [30] and compared general and thin bootstrapping. Both methods use the StC-first ordering of Ma et al. [38]. For our scalar-slot leaves, let A_r be the initial StC output. ModRaise yields $p^\tau A_r + I_r$; CtS exposes its coefficients, and digit removal recovers those of A_r, hence the original leaf slots, when $\|I_r\|_\infty < p^\tau/2$. Public scaling and plaintext-factor normalization are included in these ideal maps.

Parameter settings. We use $t = p = 65537$, $e = 1$, $N = 2^{16}$, and $n = 2^{15}$ for Ours. Thus $\ell_N = \ell_n = 2^{15}$, with slot degrees $d_N = 2$ and $d_n = 1$. In Table 7, Direct Set I matches Ours' initial modulus, whereas Sets II–III allow more capacity. All reported performance configurations meet the 128-bit target under these estimates [8]. Set IV illustrates the tighter leaf-ring constraint: four columns give a leaf estimate of 117.54 bits, below the target. Eight columns reduce P_{KS} and raise this estimate to 134.23 bits in Set V. Meeting the security target therefore also affects the switching decomposition and its cost, as discussed in Section 8.2.

Partitions. We use two transform partitions: P1 is $(0, 8, 16)$ for both CtS and StC; P2 is $(0, 4, 7, 12, 16)$ for CtS and $(0, 6, 11, 16)$ for StC. P1 uses fewer groups and preserves more capacity; P2 uses sparser layers and runs faster.

Table 7: BGV parameters. λ is the minimum security estimate over dense ternary main secrets across the rings; λ' is that of the encapsulated bootstrapping secret with Hamming weight $h' = 26$. Estimates use $\sigma = 3.2$.

Set	c	**Method**	$\log N$	$\log n$	$\log Q_R$	$\log P_{\mathrm{KS}}$	**Security (bits)**		k
							λ	λ'	
I	4	Direct	16	16	726.298	215.636	252.17	154.89	1
II	4	Direct	16	16	778.867	231.828	232.61	154.89	1
III	8	Direct	16	16	778.867	109.759	269.51	154.89	1
IV	4	Ours	16	15	726.298	215.636	117.54	134.16	2
V	8	Ours	16	15	726.298	101.879	134.23	134.16	2

λ' uses a balanced-sign approximation at $q_{\mathrm{est}} = 2^{75}$, rounding up the encapsulation modulus $q_{\mathrm{KS}}R$ ($\log_2(q_{\mathrm{KS}}R) \approx 74.39$); q_{KS} and R are the entry and auxiliary key-switching moduli.
After StC, ciphertexts are modulus-switched down and key-switched to the encapsulated secret. Following modulus raising, they are switched back to the main secret before CtS and digit removal.

Measurements. For both Direct and Ours, we report arithmetic means over ten runs after one warm-up. Each run uses fresh keys and ciphertexts and checks every output plaintext coordinate.

Capacity is HElib's estimate [30] $C(\mathbf{ct}) = \log_2 Q_{\mathbf{ct}} - \log_2 \max\{\widehat{B}(\mathbf{ct}), 1\}$, where $Q_{\mathbf{ct}}$ is the active modulus and $\widehat{B}$ the stored noise estimate. *Initial* is fresh input capacity. *Remaining*, denoted C_{rem}, is output capacity minus the budget for entering the next bootstrap: StC (about 60–106 bits), decryption-formula simplification (18 bits), and Split for Ours (below 0.01 bit). Ours measures output after Merge and the return key switch, and reserves the largest such budget over active leaves.

Stage losses exclude modulus-raising gains and reserved capacity, so they need not sum to *Initial* minus *Remaining*. We average the per-run capacity-weighted throughput $SC_{\mathrm{rem}} \log_2 t/(10^6 T_{\mathrm{total}})$, with $S = \ell_N d_N$ scalar coordinates for general inputs and $S = \ell_N$ for thin inputs. Ratios use unrounded means; total time includes unlisted operations.

For Set V/P1, sequential and concurrent runs produced identical ciphertexts.

Results. BGV performance depends on the transform partition, switching decomposition, and modulus budget. Since Direct supports a larger modulus at the same security target, matching initial capacity does not also match remaining capacity or circuit cost. Table 8 compares the methods under matched transform partitions, comparable remaining capacity, or matched initial capacity.

Table 8: Serial BGV bootstrapping with one worker. Ours averages per-run sums of leaf-stage times and maximum capacity losses over active leaves.

Bootstrapping Type		**General**				**Thin**			
Parameter Set		II	III	V		II	I	V	
Method		Direct	Direct	Ours	Ours	Direct	Direct	Ours	Ours
Partition Style		P2	P1	P2	P1	P2	P2	P2	P1
Capacity (bits)	Initial	751.11	751.11	698.54	698.54	751.11	698.54	698.54	698.54
	Split	–	–	19.75	19.75	–	–	19.74	19.74
	SlotToCoeff	80.83	89.50	106.27	86.81	79.88	79.88	106.19	86.73
	CoeffToSlot	181.55	112.96	182.87	109.79	181.45	181.46	182.84	109.69
	Digit removal	410.94	409.17	337.34	336.80	343.33	343.34	337.30	336.73
	Merge	–	–	0.00	0.00	–	–	0.00	0.00
	Remaining	43.91	105.14	38.37	131.46	112.10	59.54	38.45	131.51
Time (sec)	Split	–	–	0.92	0.92	–	–	0.92	0.92
	SlotToCoeff	9.68	33.85	9.96	32.82	10.06	9.69	4.97	16.44
	CoeffToSlot	29.01	155.82	27.82	118.71	30.15	29.09	13.94	59.71
	Digit removal	22.36	27.04	11.62	12.39	11.51	11.32	5.80	6.19
	Merge	–	–	1.58	1.58	–	–	0.85	0.85
	Total†	61.90	217.67	52.80	167.32	52.59	50.95	26.92	84.54
Throughput (Mbps)		0.744	0.506	0.762	0.824	1.118	0.613	0.749	0.816
Key generation (sec)‡		81.92	145.61	72.09	71.87	83.91	81.18	72.07	71.93
Server key size (GiB)*		8.63	15.21	7.82	7.82	8.63	8.63	7.82	7.82

† Total time covers Direct bootstrapping or, for Ours, Split, leaf bootstrapping, and Merge, including both key switches. Setup, serialization, plaintext checks, and measurements of the reserved budget are excluded. The latter use an output copy and stop before modulus raising. General Direct's digit removal includes Unpack/Pack.
‡ Key-generation time covers secret preparation and evaluation keys, excluding context construction.
∗ Server key size counts serialized public evaluation material, with shared leaf keys counted once.

For general bootstrapping with P1 and eight columns, Ours (V) is 1.30× faster than Direct (III), retains 131.46 versus 105.14 bits from a smaller initial modulus, and achieves 1.63× Direct's throughput. Key-generation time and server-key size decrease by 50.6% and 48.6%.

With P2, Ours (V, eight columns) and Direct (II, four columns) retain comparable capacity, 38.37 and 43.91 bits, from initial capacities of 698.54 and 751.11

bits. Ours is 1.17× faster and achieves 1.02× Direct's throughput; its lower latency offsets the slightly lower remaining capacity. Key-generation time and server-key size decrease by 12.0% and 9.4%. At matched initial capacity, Direct Set I completes one refresh but has negative *Remaining* after deducting this budget, whereas Ours retains a positive estimate.

Ring switching converts a general bootstrap into two thin leaf bootstraps while preserving all message coordinates. For P2, smaller-ring arithmetic and trivial leaf Unpack/Pack reduce digit-removal time and capacity loss: 11.62 versus 22.36 s and 337.34 versus 410.94 bits. Split and Merge add circuit overhead, while shared evaluation keys limit setup and storage costs. Both methods use weight-26 ModRaise secrets, so the modeled $1/\sqrt{k}$ range reduction does not explain these gains.

For Set V, P2's sparser layers run faster, but P1 retains more capacity and yields higher capacity-weighted throughput. The choice depends on the required post-bootstrap depth. For thin inputs, only one leaf is refreshed, roughly halving Ours' runtime. Direct also benefits: under Set II and P2, its digit-removal capacity loss drops from 410.94 to 343.33 bits. Thin inputs therefore narrow Ours' capacity advantage over Direct.

Direct can exploit this smaller gap in two ways. Reducing its modulus to the Set I value matches Ours' initial capacity while retaining 59.54 versus 38.45 bits. Ours (V) takes 26.92 s versus 50.95 s for Direct (I), giving a 1.89× speedup and 22.2% higher throughput. Alternatively, Direct (II) keeps the larger modulus, retains 112.10 bits, and achieves 1.49× Ours' throughput under P2. Ours still benefits from smaller-ring arithmetic, while the tighter leaf-ring modulus budget limits its capacity at the same partition.

Table 9: General BGV bootstrapping with two workers. Speedup is relative to each configuration's one-worker time in Table 8.

Method	Wall time	CPU time	Remaining	Throughput	Speedup
Direct (III, P1)	117.97 s	226.05 s	105.14 bits	0.935 Mbps	1.85×
Direct (II, P2)	37.86 s	65.08 s	43.91 bits	1.216 Mbps	1.64×
Ours (V, P1)	87.68 s	172.63 s	131.46 bits	1.572 Mbps	1.91×
Ours (V, P2)	28.74 s	54.91 s	38.37 bits	1.400 Mbps	1.84×

Parallel execution. With two workers (Table 9), Ours runs both leaves concurrently for general inputs, gaining 1.84–1.91× over one worker; Direct gains 1.64–1.85×. For P2 at comparable remaining capacity, Ours' speedup over Direct rises from 1.17× to 1.32×, with 1.15× Direct's throughput. For P1, Ours is 1.35× faster with 1.68× Direct's throughput. Remaining capacity is unchanged; the gains come from lower latency through leaf concurrency.

10 Conclusion and Future Work

Conventional SIMD ring-switching constructions restore the slot layout before further computation [26,39]. We show that independent leaf computation can

proceed without this recovery, and characterize its scope through coefficientwise correctness, fusion limits, and the affine classification.

Once slot recovery is no longer required, the main practical constraint is the modulus budget permitted by leaf-ring security. More broadly, this changes how the smaller-ring ciphertexts produced by ring switching should be viewed: not merely as an intermediate form awaiting restoration, but as a valid computational representation whenever subsequent computation admits independent leaf evaluation.

This also gives FHE compilers a new optimization choice beyond bootstrap placement and target level [33,35,18]: whether to ring-switch and how small the leaf ring should be. Accelerators such as HEAP [1] motivate studying execution across devices using complete leaf bootstraps as independent tasks. Another direction is to retain the leaf representation across stages that admit independent leaf evaluation, delaying Merge.

References

1. Agrawal, R., Chandrakasan, A., Joshi, A.: HEAP: A fully homomorphic encryption accelerator with parallelized bootstrapping. In: 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA). pp. 756–769. IEEE (2024). https://doi.org/10.1109/ISCA59077.2024.00060
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015). https://doi.org/10.1515/jmc-2015-0016
3. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum Key Exchange—A New Hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343. USENIX Association, Austin, TX (Aug 2016)
4. Arita, S., Handa, S.: Subring homomorphic encryption. In: Kim, H., Kim, D.C. (eds.) Information Security and Cryptology – ICISC 2017. Lecture Notes in Computer Science, vol. 10779, pp. 112–136. Springer (2018). https://doi.org/10.1007/978-3-319-78556-1_7
5. Aung, K.M.M., Lim, E., Sim, J.J., Tan, B.H.M., Wang, H.: Bootstrapping with RMFE for fully homomorphic encryption. In: Jager, T., Pan, J. (eds.) Public-Key Cryptography – PKC 2025, Part V. Lecture Notes in Computer Science, vol. 15678, pp. 71–101. Springer (2025). https://doi.org/10.1007/978-3-031-91832-2_3
6. Bae, Y., Cheon, J.H., Hanrot, G., Park, J.H., Stehlé, D.: Plaintext-ciphertext matrix multiplication and FHE bootstrapping: Fast and fused. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part III. Lecture Notes in Computer Science, vol. 14922, pp. 387–421. Springer (2024). https://doi.org/10.1007/978-3-031-68382-4_12
7. Bae, Y., Cheon, J.H., Kim, J., Park, J.H., Stehlé, D.: HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 37–69. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-38551-3_2
8. Bossuat, J.P., Cammarota, R., Chillotti, I., Curtis, B.R., Dai, W., Gong, H., Hales, E., Kim, D., Kumara, B., Lee, C., Lu, X., Maple, C., Pedrouzo-Ulloa, A., Player, R., Polyakov, Y., Lopez, L.A.R., Song, Y., Yhee, D.: Security guidelines for implementing homomorphic encryption. IACR Communications in Cryptology **1**(4) (2025). https://doi.org/10.62056/anxra69p1
9. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 587–617. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_21
10. Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: 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. Lecture Notes in Computer Science, vol. 13269, pp. 521–541. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-09234-3_26
11. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. pp. 868–886. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50

12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. ACM Transactions on Computation Theory **6**(3), 13:1–13:36 (2014). https://doi.org/10.1145/2633600
13. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Advances in Cryptology – CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 505–524. Springer (2011). https://doi.org/10.1007/978-3-642-22792-9_29
14. Chen, H., Han, K.: Homomorphic lower digits removal and improved FHE bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. Lecture Notes in Computer Science, vol. 10820, pp. 315–337. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-78381-9_12, https://doi.org/10.1007/978-3-319-78381-9_12
15. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: Cid, C., Jacobson, Jr., M.J. (eds.) Selected Areas in Cryptography – SAC 2018. pp. 347–368. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-10970-7_16
16. Cheon, J.H., Hanrot, G., Kim, J., Stehlé, D.: SHIP: A shallow and highly parallelizable CKKS bootstrapping algorithm. In: Fehr, S., Fouque, P.A. (eds.) Advances in Cryptology – EUROCRYPT 2025. pp. 398–428. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-91131-6_14
17. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 409–437. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
18. Cheon, S., Lee, Y., Kim, D., Lee, J.M., Jung, S., Kim, T., Lee, D., Kim, H.: DaCapo: Automatic bootstrapping management for efficient fully homomorphic encryption. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 6993–7010. USENIX Association, Philadelphia, PA (Aug 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/cheon
19. Coron, J.S., Seuré, T.: PaCo: Bootstrapping for CKKS via partial CoeffToSlot. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025. pp. 36–67. Springer Nature Singapore, Singapore (2026). https://doi.org/10.1007/978-981-95-5122-4_2
20. EPFL-LDS and Tune Insight SA: Lattigo v6. Online: https://github.com/tuneinsight/lattigo/commit/b98d89155f0bca89b4317ab3b346edfc45207da7 (Mar 2025), commit b98d89155
21. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144
22. Gao, M., Zheng, H.: FHE for SIMD arithmetic logic units with amortized $o(1)$ bootstrapping per ciphertext. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology – CRYPTO 2026. Lecture Notes in Computer Science, vol. 16801, pp. 299–330. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-35374-0_10, https://doi.org/10.1007/978-3-032-35374-0_10
23. Geelen, R.: Revisiting the slot-to-coefficient transformation for BGV and BFV. IACR Communications in Cryptology **1**(3) (2024). https://doi.org/10.62056/a01zogy4e-
24. Geelen, R., Vercauteren, F.: Bootstrapping for BGV and BFV revisited. Journal of Cryptology **36**(2), 12 (2023). https://doi.org/10.1007/s00145-023-09454-6

25. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. pp. 169–178. Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440
26. Gentry, C., Halevi, S., Peikert, C., Smart, N.P.: Ring switching in BGV-style homomorphic encryption. In: Visconti, I., De Prisco, R. (eds.) Security and Cryptography for Networks. pp. 19–37. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32928-9_2
27. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in HElib. In: Advances in Cryptology – CRYPTO 2018. pp. 93–120. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-96884-1_4
28. Halevi, S., Shoup, V.: HElib design principles (2020), https://homenc.github.io/HElib/documentation/Design_Document/HElib-design.pdf, version 0.1, August 20
29. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (2021). https://doi.org/10.1007/s00145-020-09368-7, https://www.shoup.net/papers/boot.pdf
30. HomEnc Project: HElib v2.2.0. Online: https://github.com/homenc/HElib/releases/tag/v2.2.0 (Sep 2021)
31. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General bootstrapping approach for RLWE-based homomorphic encryption. IEEE Transactions on Computers **73**(1), 86–96 (2024). https://doi.org/10.1109/TC.2023.3318405
32. Kim, D., Song, Y.: Approximate homomorphic encryption over the conjugate-invariant ring. In: Lee, K. (ed.) Information Security and Cryptology – ICISC 2018. Lecture Notes in Computer Science, vol. 11396, pp. 85–102. Springer (2019). https://doi.org/10.1007/978-3-030-12146-4_6
33. Krastev, A., Samardzic, N., Langowski, S., Devadas, S., Sanchez, D.: A tensor compiler with automatic data packing for simple and efficient fully homomorphic encryption. Proceedings of the ACM on Programming Languages **8**(PLDI) (2024). https://doi.org/10.1145/3656382
34. Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: An update. In: Dawson, E. (ed.) Topics in Cryptology – CT-RSA 2013. pp. 293–309. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36095-4_19
35. Liu, Y., Lai, J., Li, L., Sui, T., Xiao, L., Yuan, P., Zhang, X., Zhu, Q., Chen, W., Xue, J.: ReSBM: Region-based scale and minimal-level bootstrapping management for FHE via min-cut. In: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1. pp. 924–939. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3669940.3707276
36. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM **60**(6), 43:1–43:35 (2013). https://doi.org/10.1145/2535925
37. Ma, S., Huang, T., Wang, A., Wang, X.: Accelerating BGV bootstrapping for large p using null polynomials over $\mathbb{Z}_{p^e}$. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. pp. 403–432. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-58723-8_14
38. Ma, S., Huang, T., Wang, A., Wang, X.: Faster BGV bootstrapping for power-of-two cyclotomics through homomorphic NTT. In: Chung, K.M., Sasaki, Y. (eds.)

Advances in Cryptology – ASIACRYPT 2024. Lecture Notes in Computer Science, vol. 15484, pp. 143–175. Springer Nature Singapore, Singapore (2024). https://doi.org/10.1007/978-981-96-0875-1_5
39. Ma, S., Huang, T., Wang, A., Wang, X.: Practical dense-key bootstrapping with subring secret encapsulation. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025. pp. 101–130. Springer Nature Singapore, Singapore (2026). https://doi.org/10.1007/978-981-95-5122-4_4
40. MATZOV: Report on the security of LWE: Improved dual lattice attack. Tech. rep., Israel Defense Forces, Israel (Apr 2022). https://doi.org/10.5281/zenodo.6412487
41. Peikert, C., Pepin, Z.: Vive galois! part 1: Optimal SIMD packing and packed bootstrapping for FHE. In: Applebaum, B., Lin, H. (eds.) Theory of Cryptography – TCC 2025, Part I. Lecture Notes in Computer Science, vol. 16268, pp. 185–219. Springer (2026). https://doi.org/10.1007/978-3-032-12287-2_7
42. Peikert, C., Zarchy, D., Zyskind, G.: High-precision exact FHE made simple, general, and fast. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology – CRYPTO 2026. Lecture Notes in Computer Science, vol. 16801, pp. 525–557. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-35374-0_17, https://doi.org/10.1007/978-3-032-35374-0_17
43. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Designs, Codes and Cryptography **71**(1), 57–81 (2014). https://doi.org/10.1007/s10623-012-9720-4
44. Sung, H., Seo, S., Kim, T., Min, C.: EvalRound+ bootstrapping and its rigorous analysis for CKKS scheme. IEEE Access **13**, 140847–140866 (2025). https://doi.org/10.1109/ACCESS.2025.3595177, https://doi.org/10.1109/ACCESS.2025.3595177

Supplementary Material

A CRT Description of BGV/BFV Slot Recovery

This section gives the detailed CRT construction underlying the BGV/BFV slot-recovery map in Section 3.2. Reduction modulo $2n$ maps the subgroup generated by p modulo $2N$ onto the subgroup generated by p modulo $2n$. Hence $d_n \mid d_N$, and $d := d_N/d_n$ divides k. The action of $\sigma_N^{d_n}$ partitions the fiber $\rho_k^{-1}(\beta_j)$ into k/d orbits of size d. Choose a representative $\alpha_{j,s}$ from the s-th orbit and define

$$g_{j,s}(X) := \prod_{u=0}^{d-1} \left(X - \sigma_N^{ud_n}(\alpha_{j,s})\right).$$

Each $g_{j,s}$ is fixed by $\sigma_N^{d_n}$, so its coefficients lie in the image of $E_{t,n}$ under the chosen embedding, and hence $g_{j,s} \in E_{t,n}[X]$. Then

$$X^k - \beta_j = \prod_{s=0}^{k/d-1} g_{j,s}(X).$$

Since $p \nmid k$ and β_j is a unit, $X^k - \beta_j$ is separable modulo p. The factors $g_{j,s}$ are therefore pairwise comaximal in $E_{t,n}[X]$, and the CRT gives

$$E_{t,n}[X]/(X^k - \beta_j) \simeq \prod_{s=0}^{k/d-1} E_{t,n}[X]/(g_{j,s}(X)).$$

Slot recovery applies this map to

$$\sum_{r=0}^{k-1} \big(\mathsf{slot}_n(P_r)\big)_j X^r.$$

Write each residue in the basis $1, X, \ldots, X^{d-1}$ and fix an ordering of the resulting (s,u) coordinates. This identifies the coordinates with the k recovered leaf slots and defines H_j. When $d_N = d_n$, $d = 1$, so the factors are linear and H_j is the Vandermonde matrix; when $d_N > d_n$, the d power-basis coordinates of each residue are stored in d recovered leaf slots. In either case, the CRT isomorphism gives the invertibility of H_j.

B CKKS Error Budget Composition

This appendix gives sufficient choices for the bounds in Proposition 1, assuming bounds for the component errors. All errors below are expressed in the coefficient coordinates and scales of that proposition.

For $\star \in \{\mathrm{pre}, \mathrm{post}\}$, write $\eta_\star = \eta_\star^{\mathrm{top}} + \mathsf{Merge}_k((\eta_{\star,r})_{r\in[k]})$, where $\eta_\star^{\mathrm{top}}$ collects additional contributions from top-ring steps and $\eta_{\star,r}$ is the corresponding error

in leaf r. Suppose $\|\eta_\star^{\mathrm{top}}\|_{2,\mathsf{coeff}} \le b_\star^{\mathrm{top}}$ and $\|\eta_{\star,r}\|_{2,\mathsf{coeff}} \le b_{\star,r}$. The pre-refresh terms exclude the input error e, which is counted separately. The post-refresh terms include propagated CtS errors and subsequent evaluation errors, measured relative to the ideal scalar refresh at the perturbed input. By Lemma 2 and the triangle inequality, valid budgets are

$$B_{\mathrm{pre}} = b_{\mathrm{pre}}^{\mathrm{top}} + \left(\sum_{r=0}^{k-1} b_{\mathrm{pre},r}^2\right)^{1/2}, \qquad B_{\mathrm{post}} = b_{\mathrm{post}}^{\mathrm{top}} + \left(\sum_{r=0}^{k-1} b_{\mathrm{post},r}^2\right)^{1/2}.$$

The sums of squares reflect disjoint coefficient positions under Merge; no statistical independence is required.

For a polynomial scalar refresh $f_q(x) = \sum_{j=0}^{d} a_j x^j$, including public scaling, suppose $\mathcal{I} \subseteq [-R, R]$ for some $R > 0$, with $\mathcal{I}$ satisfying the interval condition of Proposition 1. Differentiating and bounding each term gives

$$\sup_{x \in \mathcal{I}} |f_q'(x)| \le \sum_{j=1}^{d} j|a_j|R^{j-1} =: L_{\mathcal{I}}.$$

This supplies a valid derivative bound, which need not be tight.

C Details for Slot-Recovery Fusion

This section gives the proofs and counterexamples underlying Section 6.2.

C.1 Proof of the ModRaise-Reordering Criterion

Proof (Proof of Lemma 4). If $HI \in \Lambda$, reduction removes qHI and leaves HP, so H^{-1} returns P. Conversely, take $P = 0$: the output vanishes exactly when $\mathrm{Red}_q(qHI) = 0$, or equivalently $HI \in \Lambda$.

C.2 Recovery and the Correction Lattice

Proof (Proof of Proposition 3). Let $e_1 = (0, 1, 0, \ldots, 0) \in \Lambda$. In branch b, $H(e_1)$ is the polynomial $h_b \in \mathcal{R}_n$ satisfying $h_b(\beta_j) = \alpha_{j,b}$, hence $h_b^k = Y$. If h_b had integer coefficients, reduction modulo two and the Frobenius identity would make h_b^k a sum of even powers of Y. Since n is even, reduction modulo $Y^n + 1$ preserves this property, contradicting $h_b^k = Y$. Thus $H(e_1) \notin \Lambda$.

The correction e_1 can occur for a valid ciphertext. For odd $q = 2m + 1$ with $m \ge 1$, set $U = Y^{n/2}$, $s = -1 - U$, $c_1 = -m + U$, and $c_0 = m - (m-1)U$. Then $U^2 = -1$ gives $c_0 + c_1 s = q$. Thus a ternary secret and centered ciphertext coefficients yield plaintext zero modulo q and correction 1 after lifting. Placing this ciphertext in leaf 1 and zeros elsewhere gives $I = e_1$, so Lemma 4 rules out the exchange in general.

For $n = k = 2$, take $h = (1 + Y)/\sqrt{2}$ and $I = (0, 1)$. Exact reduction gives

$$H^{-1} \operatorname{Red}_q(H(qI)) = (0, q(1 - \sqrt{2})).$$

Suppose approximate reduction differs from Red_q at these inputs by at most ε per coefficient. Applying H^{-1} amplifies this perturbation by at most $\sqrt{2}$. The coefficient error after inverse recovery is therefore at least $q(\sqrt{2}-1) - \sqrt{2}\,\varepsilon$, and remains $\Theta(q)$ when $\varepsilon = o(q)$.

D General-k Inverse-Recovery Fusion

Section 6.1 fuses inverse recovery for $k = 2$ while preserving the StC rotation indices and diagonal support. We extend the factorization to general power-of-two k; preserving operation counts and the rescaling schedule also depends on the resulting cross-leaf mixing.

On a representative CKKS recovery fiber, use the notation of Section 3.2 and write $H_j = F_k D_j$, where $(F_k)_{a,r} = \zeta_k^{ar}$ and $D_j = \operatorname{diag}(1, \alpha_{j,0}, \ldots, \alpha_{j,0}^{k-1})$. Let $h \in \mathcal{R}_n$ satisfy $h(\beta_j) = \alpha_{j,0}$; then $h^k = Y$, so h is invertible. For each $a, r \in [k]$, let $\gamma_{a,r} \in \mathcal{R}_n$ be the unique CKKS plaintext whose evaluations at β_j are $k^{-1}\zeta_k^{-ar}$, with conjugate values at conjugate roots. Define $\Gamma_k = (\gamma_{a,r})_{a,r\in[k]}$ as a matrix of slot-diagonal plaintext multipliers and $D_h = \operatorname{diag}(1, h, \ldots, h^{k-1})$. The real-linear CKKS slot isomorphism then gives, on representative fibers,

$$H^{-1} = D_h^{-1}\Gamma_k, \qquad (H^{-1}\mathbf{U})_a = h^{-a}\sum_{r=0}^{k-1}\gamma_{a,r}U_r.$$

Proposition 7 (General-k inverse-recovery fusion). *Let $\mathsf{StC}_n = A_d \circ W$, with the same final grouped factor A_d on every leaf. Then, as an exact plaintext identity,*

$$H^{-1} \circ \mathsf{StC}_n^{\oplus k} = D_h^{-1}\Gamma_k \circ A_d^{\oplus k} \circ W^{\oplus k}.$$

If Γ_k commutes with $A_d^{\oplus k}$ on the conjugate-symmetric CKKS plaintext space, this can be evaluated in fused form as

$$H^{-1} \circ \mathsf{StC}_n^{\oplus k} = D_h^{-1} \circ A_d^{\oplus k} \circ \Gamma_k \circ W^{\oplus k},$$

or, for $\mathbf{w} = W^{\oplus k}\mathbf{z}$,

$$V_a = h^{-a} A_d\left(\sum_{r=0}^{k-1}\gamma_{a,r}w_r\right), \qquad a \in [k].$$

Proof. The first identity follows from $H^{-1} = D_h^{-1}\Gamma_k$ and the factorization $\mathsf{StC}_n = A_d \circ W$. The second uses the stated commutation relation; the last display gives its a-th output component.

For $k = 2$, $\Gamma_2 = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Its entries are real scalars, so the commutation condition holds and the proposition reduces to the sum-and-difference construction in Section 6.1.

The output multipliers h^{-a} can be folded into the plaintext diagonals of the final StC factors without changing their rotation indices. The matrix Γ_k, however, is a dense cross-leaf operation: a direct implementation has k^2 diagonal terms, while a radix-two factorization uses $O(k \log k)$ butterflies across $O(\log k)$ stages and generally introduces nonconstant diagonal factors. These stages still require synchronization, and whether they fit within an existing StC factor or consume an additional level depends on the implementation. Thus the proposition gives a general-k factorization; it does not by itself extend the operation counts and rescaling schedule established for $k = 2$ in Section 6.1.

E Independent Leaf Evaluation: Proofs and Extensions

This section proves the linear-map and automorphism results, establishes the complex CKKS and BGV/BFV leafwise evaluation results and Proposition 6, derives the optimal leafwise ReLU errors, and gives a restricted domain permitting exact evaluation.

E.1 Proofs for Independent Leaf Evaluation

Proof (Proof of Proposition 4). Output branch r equals $\sum_s L_{rs}(P_s)$. Since the P_s vary independently on the full plaintext space, the claim follows.

Proof (Proof of Proposition 5). Direct substitution gives $\big(\mathsf{Split}_k(\sigma_a(P))\big)_{\tau_a(r)} = Y^{u_r} P_r(Y^a)$. Since a is coprime to k, τ_a is a permutation. Each $A_{a,r}$ is invertible, so without leaf permutation the condition is $\tau_a(r) = r$, equivalently $(a-1)r \equiv 0 \pmod{k}$.

E.2 Proof of Independent Leaf Evaluation on Complex CKKS Slots

Theorem 2 (Independent Leaf Evaluation on Complex CKKS Slots). *Assume $n, k \geq 2$ and $J \neq \varnothing$. For every continuous $\phi : \mathbb{C} \to \mathbb{C}$, the map $P \mapsto \mathsf{slot}_N^{-1}(\phi^{\oplus N/2}(\mathsf{slot}_N(P)))$ on $\mathcal{R}_N$ admits independent leaf evaluation without leaf permutations for the branches in J if and only if $\phi(z) = az + b\overline{z} + d$ for some $a, b \in \mathbb{R}$ and $d \in \mathbb{C}$, where $b = 0$ whenever J contains an index r satisfying $2r \not\equiv 0 \pmod{k}$.*

Proof. Fix $r \in J$ and $j \in [n/2]$. Let $x \in \mathbb{C}^k$ be the vector of the j-th leaf slots, set $y = H_j x$, and write $S_r(y) = \sum_{c=0}^{k-1} \zeta_k^{-cr} y_c$. Since $H_j = F_k D_j$, we have $x_r = \alpha_{j,0}^{-r} S_r(y)/k$.

For $c < k/2$, $\alpha_{j,c} = \omega_{N,j+cn}$ is one of the representative roots defining slot_N, so the corresponding output value is $\phi(y_c)$. For $c \geq k/2$, $\overline{\alpha_{j,c}}$ is a representative root, so the output value at $\alpha_{j,c}$ is $\overline{\phi(\overline{y_c})}$. Hence the j-th slot of output leaf r is

$$\frac{\alpha_{j,0}^{-r}}{k}\left(\sum_{c=0}^{k/2-1}\zeta_k^{-cr}\phi(y_c)+\sum_{c=k/2}^{k-1}\zeta_k^{-cr}\overline{\phi(\overline{y_c})}\right).$$

Surjectivity of $\mathsf{slot}_n : \mathcal{R}_n \to \mathbb{C}^{n/2}$ lets the j-th leaf slots vary arbitrarily with all other slots fixed. Invertibility of H_j then makes y arbitrary. By Lemma 5, independent evaluation of branch r requires the preceding sum to depend only on $S_r(y)$.

Replacing ϕ by $g = \phi - \phi(0)$ changes the output only by a fixed plaintext. Take $y_0 = u$, $y_{k/2} = v$, and all other coordinates zero, and compare this with the vector having only $y_0 = u+(-1)^r v$. Their S_r-values agree, so $g(u)+(-1)^r\overline{g(\overline{v})} = g(u+(-1)^r v)$. Setting $u = 0$ gives $(-1)^r\overline{g(\overline{v})} = g((-1)^r v)$; substituting this back gives $g(u)+g((-1)^r v) = g(u+(-1)^r v)$. Since $(-1)^r v$ ranges over $\mathbb{C}$, g is additive. Thus $g((-1)^r v) = (-1)^r g(v)$, and hence $g(v) = \overline{g(\overline{v})}$.

By continuity, g is $\mathbb{R}$-linear, so $g(z) = az + b\overline{z}$ for some $a, b \in \mathbb{C}$. The identity $g(z) = \overline{g(\overline{z})}$ gives $az + b\overline{z} = \overline{a}z + \overline{b}\,\overline{z}$, and therefore $a, b \in \mathbb{R}$.

For $c < k/2$, compare the vector having only $y_c = z$ with the vector having only $y_0 = \zeta_k^{-cr} z$. Their S_r-values agree, so $g(\zeta_k^{-cr} z) = \zeta_k^{-cr} g(z)$. If $2r \not\equiv 0 \pmod{k}$, then $k \geq 4$ and ζ_k^{-r} is nonreal. Taking $c = 1$ gives $b(\zeta_k^r - \zeta_k^{-r})\overline{z} = 0$ for every z, and hence $b = 0$.

Conversely, suppose $g(z) = az + b\overline{z}$ with $a, b \in \mathbb{R}$ and the stated restriction on b. Since $\overline{g(\overline{z})} = g(z)$, the output sum is $aS_r(y)$ when $b = 0$. If $b \neq 0$, then $2r \equiv 0 \pmod{k}$, so all weights ζ_k^{-cr} are real and the sum is $aS_r(y)+b\overline{S_r(y)}$. Since $S_r(y) = k\alpha_{j,0}^r x_r$, the j-th output leaf slot is ax_r when $b = 0$ and $ax_r + b\alpha_{j,0}^{-2r}\overline{x_r}$ when $b \neq 0$. Thus it depends only on the corresponding input leaf slot. Since r and j were arbitrary, independent leaf evaluation follows. Adding back $d = \phi(0)$ contributes only a fixed plaintext.

When $J = [k]$ and $k = 2$, both branches satisfy $2r \equiv 0 \pmod{k}$, giving $\phi(z) = az + b\overline{z} + d$ with $a, b \in \mathbb{R}$ and $d \in \mathbb{C}$. When $k \geq 4$, branch $r = 1$ forces $b = 0$, so the classification becomes $\phi(z) = az + d$ with $a \in \mathbb{R}$ and $d \in \mathbb{C}$.

E.3 Proof of BGV/BFV Leafwise Evaluation

We give the BGV/BFV analogue in terms of additive maps over the finite plaintext rings.

Theorem 3 (Leafwise Evaluation on BGV/BFV Slots). *Assume $d_N = kd_n$, and choose compatible CRT slot embeddings such that, for every $j \in [\ell_n]$, $E_{t,N} = \bigoplus_{r=0}^{k-1} \alpha_j^r E_{t,n}$ with $\alpha_j^k = \beta_j$. Let $\varphi : E_{t,N} \to E_{t,N}$ be a plaintext slotwise map, and let $\Phi_{\varphi,N}$ apply φ to every top-ring slot. Assume that all k output branches are required and that no public permutation of the leaves is allowed.*

Then $\Phi_{\varphi,N}$ admits exact independent leaf evaluation if and only if, with $d := \varphi(0)$, the map $f := \varphi - d$ is additive (equivalently, $\mathbb{Z}_t$-linear) and

$$f\big(\alpha_j^r E_{t,n}\big) \subseteq \alpha_j^r E_{t,n} \qquad (j \in [\ell_n],\ r \in [k]).$$

Equivalently, for every j and r, there is an additive map $g_{j,r} : E_{t,n} \to E_{t,n}$ such that

$$\varphi\left(\sum_{r=0}^{k-1} \alpha_j^r x_r\right) = d + \sum_{r=0}^{k-1} \alpha_j^r g_{j,r}(x_r)$$

for all $x_0, \ldots, x_{k-1} \in E_{t,n}$.

Proof. Fix a fiber j and write $y = \sum_r \alpha_j^r x_r$. The r-th output leaf is the α_j^r-coordinate of $\varphi(y)$. By Lemma 5, its independent evaluation is equivalent to this coordinate depending only on x_r, which yields the stated decomposition. At $x_0 = \cdots = x_{k-1} = 0$, uniqueness gives $g_{j,r}(0) = 0$ for every r. Thus $f = \varphi - d$ preserves every summand; the direct-sum decomposition then gives $f(u+v) = f(u) + f(v)$, so f is additive and hence $\mathbb{Z}_t$-linear. Conversely, if f is additive and preserves every summand, define $g_{j,r}$ by $f(\alpha_j^r x_r) = \alpha_j^r g_{j,r}(x_r)$. The stated decomposition follows, so each output branch depends only on its corresponding input leaf. Applying this on every fiber and merging proves the claim.

In the setting $d_n = 1$, every additive endomorphism of $E_{t,n} = \mathbb{Z}_t$ is multiplication by a scalar. For $k = 2$, the theorem therefore specializes to

$$\varphi(a + b\alpha_j) = d + c_0 a + c_1 b\alpha_j, \qquad c_0, c_1 \in \mathbb{Z}_t.$$

The BGV/BFV bootstrapping result of Lemma 3 is stronger and does not require $d_N = k d_n$: its digit-removal map is coefficientwise and commutes with Split_k and Merge_k for the general CRT decomposition.

E.4 ReLU on Full and Restricted Domains

We first prove Proposition 6.

Proof. After Merge, the stated leaf maps give $P/2 + \Delta B/4$, whose slots approximate ReLU within $B/4$ on $[-B, B]$. For the lower bound, fix a recovery fiber and choose $y \in \{\pm B\}^k$ with equally many positive and negative entries. Compare it with $y' = 0$, setting all other representative fibers to zero. Since the zeroth row of H_j^{-1} is $\mathbf{1}^T/k$, both plaintexts have $P_0 = 0$, whereas their target slots in leaf 0 are $B/2$ and 0. Any G_0 therefore incurs leaf-slot error of at least $B/4$ on one input. Each row of H_j^{-1} has absolute row sum one, so inverse recovery cannot increase the maximum slot error. Root reordering and conjugation preserve this norm; hence the original top-ring slot error is at least $B/4$.

This bound concerns ideal plaintext maps and excludes encoding and homomorphic evaluation errors. The same maps minimize the worst-case error in each leaf. Denote branch r of the target by $\Phi_{\mathrm{ReLU},N,r}(P)$, and define its optimal error by

$$E_{r,B} := \inf_{G_r : \pi_r(\mathcal{D}_B) \to \mathcal{R}_n} \sup_{P \in \mathcal{D}_B} \left\| \mathsf{slot}_n(G_r(P_r)) - \mathsf{slot}_n(\Phi_{\mathrm{ReLU},N,r}(P)) \right\|_\infty .$$

Proposition 8 (Leafwise ReLU optimum). *For $n, k \geq 2$, $B > 0$, and the domain $\mathcal{D}_B$ of plaintexts with real slots bounded by B, the optimal leaf errors satisfy $E_{0,B} = B/4$ and, for $0 < r < k$, $E_{r,B} = \frac{B}{2d \sin(\pi/d)}$, where $d = \frac{k}{\gcd(r,k)}$.*

Proof. Write $x = H_j^{-1} y$ for the input leaf slots of a fiber y. The case $r = 0$ follows from the preceding proof: the target leaf slot is $x_0/2 + (2k)^{-1} \sum_b |y_b|$, and the second term lies in $[0, B/2]$. For $r > 0$, set $g = \gcd(r, k)$, $w_b = \zeta_k^{-br}$, and $\alpha = \alpha_{j,0}$. On a fiber $y \in [-B, B]^k$, the target leaf slot is

$$\frac{x_r}{2} + \frac{\alpha^{-r}}{2k} \sum_b w_b |y_b|.$$

The weights are the d-th roots of unity, each repeated g times. Since d is even, maximizing a projection selects $d/2$ consecutive roots. The geometric-series identity gives

$$\left| \sum_{\ell=0}^{d/2-1} e^{2\pi \mathrm{i} \ell/d} \right| = \frac{2}{|1 - e^{2\pi \mathrm{i}/d}|} = \csc(\pi/d).$$

Hence $\max_{0 \leq t_b \leq B} |\sum_b w_b t_b| = Bg \csc(\pi/d)$. This gives the upper bound for $G_r(A) = A/2$. For the lower bound, pair opposite weights $w, -w$, choosing w from $d/2$ consecutive roots. On each pair use inputs $(B, 0)$ and $(0, -B)$. Both weighted input sums equal Bw, while their ReLU output sums differ by Bw. Across all pairs and repetitions, the target leaf slots differ by $Bg \csc(\pi/d)/k$. With other representative fibers zero, the entire input leaves coincide, so any G_r incurs at least half this difference.

For a polynomial approximating ReLU within η on $[-B, B]$, independent leaf evaluation has worst-case error at least $\max\{0, E_{r,B} - \eta\}$ in leaf r and $\max\{0, B/4 - \eta\}$ in the original slots. These bounds concern ideal plaintext maps.

Restricting the input domain can permit exact ReLU evaluation. Consider the domain defined by $\mathsf{Split}_k(P) = (P_0, 0, \ldots, 0)$, with $\mathsf{slot}_n(P_0) \in \mathbb{R}^{n/2}$. For every recovery fiber, $H_j(x_0, 0, \ldots, 0)^T = (x_0, \ldots, x_0)^T$. Applying ideal ReLU and then H_j^{-1} gives $(\mathrm{ReLU}(x_0), 0, \ldots, 0)^T$. Hence $G_0 = \mathsf{slot}_n^{-1} \circ \mathrm{ReLU}^{\oplus n/2} \circ \mathsf{slot}_n$ and $G_r = 0$ for $r > 0$ give exact independent leaf evaluation. This illustrates why Lemma 5 specifies the reachable input domain $\mathcal{D}$.

F Supplementary Cost Analysis

F.1 BSGS Evaluation and Operation Counts

For a grouped transform A, write $A\mathbf{z} = \sum_j \mathrm{diag}_j(A) \odot \mathrm{Rot}_j(\mathbf{z})$, and suppose that all nonzero diagonals have indices in $0, \ldots, D-1$. Set $g = \lceil \sqrt{D} \rceil$ and $h = \lceil D/g \rceil$. The standard BSGS evaluation [27] is

$$A\mathbf{z} = \sum_{i=0}^{h-1} \mathrm{Rot}_{ig} \left(\sum_{0 \leq j < g,\ ig+j < D} \mathrm{Rot}_{-ig}\big(\mathrm{diag}_{ig+j}(A)\big) \odot \mathrm{Rot}_j(\mathbf{z}) \right). \tag{1}$$

It uses at most D plaintext-ciphertext multiplications and $g + h - 2$ rotations. The $g - 1$ baby-step rotations are reused across groups, and the outer sum adds at most $h - 1$ rotations. The rotated plaintext diagonals can be precomputed. For sparse factors, the actual counts depend on the nonzero diagonal indices.

Table 10: BSGS operation counts for the Group III CKKS linear transforms. Here M_s counts plaintext-ciphertext multiplications and R_s counts rotations.

Transform	**Ring dimension**	**Slots**	$\boldsymbol{M_s}$	$\boldsymbol{R_s}$
CtS	$N = 2^{17}$	2^{16}	109	36
CtS	$n = 2^{16}$	2^{15}	93	32
StC	$N = 2^{17}$	2^{16}	190	42
StC	$n = 2^{16}$	2^{15}	158	38

In the Group III settings, both CtS and StC require fewer plaintext-ciphertext multiplications and rotations in the leaf ring. These counts measure operation invocations. The number of rotation keys depends on the distinct automorphisms required, since each key can be reused across grouped factors.

F.2 Server Key Storage

Let $\mathcal{K}_m$ be a set of distinct bootstrap evaluation keys stored in dimension m. For key a, let c_a be its number of decomposition columns and L_a the total number of RNS moduli in its Q/P basis. At fixed word size and serialization format, each column stores a constant number of polynomials with m coefficients over these moduli [20,28]. The dominant storage in machine words is therefore

$$S_{\mathrm{BTS}}(m) = \Theta\left(m \sum_{a \in \mathcal{K}_m} c_a L_a\right).$$

Matching key counts and per-key parameters gives the $1/k$ scaling from N to n; increasing decomposition column counts can offset this reduction.

Table 11 decomposes the serialized server-key sizes from Tables 3 and 8. Values are means over the same ten runs; the leaf material is counted once across all leaves.

Table 11: Server-key storage components (GiB). BGV decomposition column counts are listed as Direct/Ours. CKKS parameter use Group III, $L_{\mathrm{out}} = 15$.

Configuration	**Direct total**	**Ours leaf**	**Ours extra**	**Ours total**	**Reduction**
CKKS	27.5666	12.3068	0.9844	13.2912	51.8%
BGV, P1 (8/8 c)	15.2056	7.5737	0.2471	7.8208	48.6%
BGV, P2 (4/8 c)	8.6292	7.5737	0.2471	7.8208	9.4%

Extra material consists of Split/Merge switching keys for CKKS and additional top-ring public evaluation material for BGV. Reductions use unrounded means. And c is the short for columns.

The BGV comparison with eight columns in both methods nearly halves storage. With four columns for Direct and eight for Ours, the increased decomposition size offsets much of this benefit. The CKKS leaf keys occupy slightly less than half of Direct's storage before the additional switching keys are counted; the ratio reflects both ring dimension and the required key material, as discussed in Section 4.3.

G Summary of Notation

The section and page numbers below link to the main definitions of the notation. Symbols are interpreted in the context of the linked section.

Table 12: Index of recurring notation.

N, n, k, X, Y	Sec. 1.2, p. 3	$\|\cdot\|_{2,\mathsf{coeff}}, \|\cdot\|_{2,\oplus}$	Sec. 4.2, p. 11
P, P_r	Sec. 1.2, p. 3	$e, \eta_{\mathrm{pre}}, \eta_{\mathrm{post}}$	Sec. 4.2, p. 11
$\mathsf{Split}_k, \mathsf{Merge}_k$	Sec. 1.2, p. 3	$B_{\mathrm{pre}}, B_{\mathrm{post}}$	Sec. 4.2, p. 11
$\mathsf{SR}_k, \mathsf{BTS}_m$	Sec. 1.2, p. 3	Δ_{out}	Sec. 4.2, p. 11
$\mathsf{ModRaise}_m, \psi_m, \Phi_m, T$	Sec. 1.2, p. 3	h_m, K_n, K_N	Sec. 4.3, p. 12
$\mathcal{F}, \mathcal{D}$	Sec. 1.2, p. 3	$\mathsf{evk}_m, S_{\mathrm{ours}}, S_{\mathrm{extra}}$	Sec. 4.3, p. 12
m, Z	Sec. 2.1, p. 5	$\mathsf{DR}_{e,\tau}, \mathcal{D}_m$	Sec. 5.1, p. 12
$\mathcal{R}_{\mathbb{Z},m}, \mathcal{R}_m, \mathcal{R}_{Q,m}$	Sec. 2.1, p. 5	K, c_m	Sec. 5.2, p. 13
$q, Q, t = p^e$	Sec. 2.1, p. 5	h	Sec. 6.1, p. 14
$[a], \log, \|\cdot\|_\infty, \odot$	Sec. 2.1, p. 5	A_d, W	Sec. 6.1, p. 14
$\zeta_M, \Omega_m, \omega_{m,j}$	Sec. 2.1, p. 5	Λ, Red_q	Sec. 6.2, p. 15
$\mathrm{ev}_m(P)$	Sec. 2.1, p. 5	$J, \pi_r, \mathcal{F}_r, G_r$	Sec. 7, p. 15
$\mathsf{DFT}_m, \mathsf{Ecd}_{\Delta,m}, \mathrm{Dcd}_{\Delta,m}$	Sec. 2.2, p. 6	$\mathcal{D}_{\mathbb{R}}, \phi, \Phi_{\phi,N}, \mathcal{D}_B$	Sec. 7, p. 15
$\mathsf{slot}_m(P), \Delta$	Sec. 2.2, p. 6	$\mu, \mathsf{Perm}_\tau, \sigma_a, \tau_a, u_r, A_{a,r}$	Sec. 7, p. 15
$d_m, \ell_m, E_{t,m}, F_{m,j}$	Sec. 2.2, p. 6	$\lambda_{\mathrm{top}}, \lambda_{\mathrm{leaf}}, \lambda_{\mathrm{est}}$	Sec. 8, p. 18
$\theta_m, \sigma, \sigma_m, \iota_{m,j}$	Sec. 2.2, p. 6	P_{KS}	Sec. 8.1, p. 18
$s, \mathbf{ct}, \mathsf{Dec}_s, \vDash_{m,q}$	Sec. 2.2, p. 6	B_j, c	Sec. 8.2, p. 19
$\vDash^{\mathsf{cpack}}_{m,q}, \mathsf{CtS}_m, \mathsf{StC}_m, \zeta_{4,m}$	Sec. 2.3, p. 7	M_s, R_s, ν_m	Sec. 8.3, p. 19
$\mathsf{EvalMod}, \mathsf{Unpack}, \mathsf{Pack}$	Sec. 2.3, p. 7	q_{est}	Sec. 9, p. 20
$Q_R, Q_1, Q_2, Q_{\mathrm{out}}$	Sec. 2.4, p. 7	$L_{\mathrm{out}}, \epsilon$	Sec. 9.1, p. 20
I, τ	Sec. 2.4, p. 7	$\delta_{\mathrm{range}}, \delta^{(1)}_{\mathrm{range}}$	Sec. 9.1, p. 20
s', s_n	Sec. 3.1, p. 8	$C(\mathbf{ct}), Q_{\mathbf{ct}}, \widehat{B}, C_{\mathrm{rem}}$	Sec. 9.2, p. 25
ρ_k	Sec. 3.2, p. 9	$d, \alpha_{j,s}, g_{j,s}$	Sec. A, p. 33
H, H_j	Sec. 3.2, p. 9	$\gamma_{a,r}, \Gamma_k, D_h$	Sec. D, p. 35
$\alpha_{j,b}, \beta_j, F_k, D_j$	Sec. 3.2, p. 9	$E_{r,B}$	Sec. E.4, p. 38
$\mathcal{F}_m, f_q, \mathcal{I}, L_{\mathcal{I}}$	Sec. 4.2, p. 11	D, g, h	Sec. F.1, p. 39