BootNet: Homomorphic CNN Inference with Convolution and ReLU Fused in Bootstrapping

Zhaomin Yang
Ant Group
China
yangzhaomin.yzm@antgroup.com

Chao Niu
Ant Group
China
niuchao.niu@antgroup.com

Cheng Hong*
Ant Group
China
vince.hc@antgroup.com

Tao Wei
Ant Group
China
lenx.wei@antgroup.com

ABSTRACT

Fully homomorphic encryption (FHE) enables privacy-preserving neural network inference but suffers from high overhead from homomorphic convolutions, polynomial activation approximations, and CKKS bootstrapping. This paper presents BootNet, a unified framework that fuses all three operations into a single bootstrapping invocation per CNN layer, achieving convolution, ReLU, and noise refresh simultaneously.

Prior works are able to fuse convolution into bootstrapping using CinS encoding (NeuJeans, CCS 2024) or ReLU into functional bootstrapping (RBOOT, USENIX Security 2026), but combining both for end-to-end ImageNet inference introduces numerous undocumented challenges. BootNet resolves these through a suite of codesign techniques and optimizations, including four representative solutions: (1) a splitandmerge bootstrapping schedule that halves bootstrapping calls for ResNet shortcuts; (2) an improved RBOOT configuration paired with a model quantization method that complete the activation function with fewer multiplication depth; (3) a fusion of NeuJeans' masking layer with RBOOT's arcsin step that yields additional depth reduction; (4) tailoring EvalRound+ (IEEE Access 2025) to fit slim bootstrapping and RBOOT for further depth reduction.

We evaluate BootNet on multiple ResNet models on the ImageNet dataset. Compared to the state-of-the-art Orion (ASPLOS 2025), BootNet reduces end-to-end latency by 67–73% and storage by 76–90% while preserving plaintext accuracy. We also introduce BootNet-mini, enabling full ImageNet inference at ring degree $N = 2^{15}$ (with over 120-bit security via sparse-secret encapsulation) for the first time, while all previous works have to use $N = 2^{16}$ or larger. BootNet-mini performs similarly in latency but reduces 30–55% storage requirement than BootNet, making it potentially more friendly for hardware acceleration.

KEYWORDS

Homomorphic Encryption, Privacy-Preserving Machine Learning, Convolutional Neural Network, Implementation

1 INTRODUCTION

Machine learning inference services are widespread: users send their queries, such as medical scans, biometric templates, financial records and other private data, to remote cloud services and receive predictions in return. This convenience, however, comes at the cost of sharing private data with the service provider. *Privacy-preserving inference* aims to eliminate this exposure by allowing the server to compute the prediction while learning nothing about the clients input.

Fully homomorphic encryption (FHE) is a promising tool for realizing privacy-preserving inference. It enables a server to evaluate arbitrary circuits directly on encrypted client data and return an encrypted result that can only be decrypted by the client. While FHE was largely theoretical in its early years, modern schemes and implementations have advanced dramatically. Recent work [10, 12] have even demonstrated the possibility of privacy-preserving inference at ImageNet scale using the CKKS [8] FHE scheme. However, their computational and storage costs remain high: according to [10], a single inference on one ImageNet image can take tens of thousands of seconds and require terabytes of storage.

The three-bottleneck problem. For convolutional neural network (CNN), which is the dominant architecture in vision, audio, and many other domains, three fundamental operations dominate the cost of FHE-based inference.

(1) Homomorphic convolution. In the standard FHE encoding, a feature map is packed into the *slot* representation of a polynomial ciphertext. Evaluating a $k \times k$ convolution then requires $O(k^2)$ ciphertext slot-rotation operations, each incurring significant keyswitching overhead.

(2) Non-linear activations. CKKS natively supports only polynomial (i.e., additive and multiplicative) operations, and cannot support non-linear functions like ReLU directly. They must therefore be approximated by high-degree polynomials, consuming significant multiplicative depth.

(3) Bootstrapping. Each homomorphic multiplication increases the noise in the ciphertext. Once this noise exceeds the budget required for subsequent operations, *bootstrapping* is needed to refresh the noise budget. Bootstrapping is itself expensive: it demands

*Corresponding author: Cheng Hong (vince.hc@antgroup.com)

high multiplicative depth and typically costs tens of seconds per invocation.

How prior work helps. Recent works have attacked these bottlenecks from different angles. On the convolution side, [17, 22, 23] focus on reducing rotations and improving packing strategies, for example by multiplexed packing. On the activation side, optimized polynomial approximations and network fine-tuning [1, 2, 19, 27] lower the required degree and thus the multiplicative depth. On the bootstrapping side, in addition to algorithmic improvements to bootstrapping itself [3, 6, 7], a line of works focus on optimally placing bootstrapping operations within the computation graph [9, 16, 30].

Among these, two notable works optimize *two* of the three bottlenecks simultaneously. NeuJeans [12] introduces the Coefficients-in-Slot (CinS) encoding, which stores feature maps in the polynomial *coefficient* domain so that a single homomorphic multiplication performs an entire $k \times k$ convolution, eliminating the requirement of $O(k^2)$ slot rotations. The key structural insight is that converting a CinS ciphertext back to the slot domain is algebraically equivalent to the Discrete Fourier Transform (DFT) in the beginning of CKKS bootstrapping: consequently, convolutions and bootstrappings can be fused at no extra cost.

RBOOT [31] takes an orthogonal approach. The EvalMod step inside bootstrapping already evaluate a highly non-linear modular-reduction function; by re-targeting this polynomial evaluation to compute ReLU, activation becomes "fused" directly the bootstrapping pass. This form of functional bootstrapping remove the needs of a separate activation stage.

However, the three bottlenecks are not independent. In particular, the rigid modulus-chain and scale-management constraints of CKKS impose tight sequential dependencies among them. Consequently, any technique that addresses only a subset of the bottlenecks is inherently sub-optimal when applied to a full CNN.

1.1 Our approach: triple fusion.

We observe that the optimizations above are not merely compatible but *mutually reinforcing*: key aspects of these techniques can be jointly optimized and fused to further reduce the overall multiplicative depth, which in turn enlarges the design space available for depth-critical operations such as the DFT. It is therefore possible to design a single bootstrapping operation that simultaneously attacks the three bottlenecks. Our contributions are threefold:

- **Independent optimizations on NeuJeans and RBOOT.** On the NeuJeans side, we contribute two speedups not addressed in [12]: a *confluent bootstrap* schedule that halves the heavy bootstrap tail at every projection-shortcut residual-block merge, and a mask-elision optimization that saves one multiplicative level on each convolution that follows a homomorphic downsampling transform. On the RBOOT side, we adapt [31] from CIFAR-scale networks to ImageNet-scale ResNets by fitting two degree-15 Chebyshev minimax arcsine polynomials with uniform 8-bit precision over the entire activation domain, and by enforcing the required pre-ReLU range bound through Scale-Aligned Post-Training Quantization (SAPTQ)—a forward-only, gradient-free graph-rewrite procedure that changes no weights and incurs less than 0.2 percentage points of top-1 accuracy across ResNet-18/34/50.
- **BootNet: a 3-in-1 bootstrapping for CNN.** We integrate NeuJeans, RBOOT, and—where the modulus budget is tight—our slim-bootstrap adaptation EvalRound++ of EvalRound+ [29] into a single fused bootstrapping unit that simultaneously refreshes noise, evaluates one convolutional layer, and applies the subsequent ReLU, with no separate homomorphic operations for any of the three. A SIMD-level coefficient pre-multiplication further fuses the restore-zero-padding mask into RBOOT's polynomial evaluation, recovering enough modulus headroom to fit a fused unit at the aggressive $N = 2^{15}$ ring degree.
- **Implementation and evaluation.** We fully implement BootNet on top of Lattigo [26] and benchmark it against Orion [10] (which is the only open-source FHE CNN inference framework at ImageNet scale) on ImageNet dataset. At ring degree $N = 2^{16}$, BootNet delivers a 2.31–4.49× end-to-end latency reduction (e.g., ResNet-34 drops from 5.1 h to 1.1 h) and a 4.3× to ≥10× reduction in precomputed-cache storage (e.g., $\geq 4TB \rightarrow$ 398 GB on ResNet-50) over Orion. At the aggressive $N = 2^{15}$ enabled by our modulus tightening, storage requirement drops further by 30% − 55% (e.g., 238 GB on ResNet-50). Our top-1 ImageNet accuracy drops by approximately 1 percentage than plaintext across all network models.

1.2 Related Work

Hybrid MPC+FHE approaches. Systems such as GAZELLE [13] and DELPHI [24] use FHE only for linear layers and garbled circuits or secret sharing for ReLU, avoiding the high cost of homomorphic ReLU approximation. These approaches require multiple communication rounds between client and server, introduce latency proportional to network depth. As a result, hybrid approaches are unfriendly for clients without stable network conditions, unlike FHE-based methods where all the encrypted CNN computation are performed server-side.

Early FHE-based inference. CryptoNets [11] demonstrated the first FHE evaluation of a neural network, but required replacing all non-linearities with low-degree polynomial activations and was limited to small networks on simple datasets. Subsequent work [5] improved throughput via batching and better encoding but retained the restriction to low-degree polynomial activations.

FHE-only inference with bootstrapping. A line of work evaluates full ReLU-activated networks entirely within FHE by combining polynomial ReLU approximations with bootstrapping after each activation layer [9, 10, 17, 21]. These systems pay the full cost of each operation separately: $O(k^2)$ rotations for convolution, polynomial levels for ReLU, and then a bootstrapping call to recover.

FHE Transformer inference. Recently, a growing line of research [25, 28, 32] has investigated privacy-preserving transformer or even large language model inference using FHE. While these efforts are highly promising, we focus on CNN inference that is closer to practical deployment. On a single CPU, our FHE-based ImageNet-scale

inference takes thousands of seconds. With hardware acceleration, this can potentially drop to the order of seconds, possibly meeting requirements of high-security applications such as secure face recognition. In contrast, transformer inference remains more computationally demanding, often requiring minutes per output token even with hardware acceleration.

2 PRELIMINARIES

We provide the background necessary to understand our design: the CKKS homomorphic encryption scheme (Section 2.1), standard and EvalRound-based CKKS bootstrapping (Section 2.2), and the conventional techniques for evaluating CNN layers homomorphically (Section 2.3).

2.1 The CKKS Scheme

Ring structure and parameters. CKKS [8] operates over the polynomial ring $R = \mathbb{Z}[X]/(X^N + 1)$, where N is a power of two (typically $N = 2^{15}$–2^{17}). A ciphertext is a pair of polynomials in $R_q = R/(qR)$ for a current modulus q. The *level* of a ciphertext refers to its position in the modulus chain $q_L > q_{L-1} > \cdots > q_0$; a fresh ciphertext has level L and the chain can support at most L multiplications before bootstrapping.

Slot encoding. The canonical encoding maps a plaintext vector $\mathbf{z} \in \mathbb{C}^{N/2}$ to a polynomial $m \in R$ via the inverse Discrete Fourier Transform (iDFT) composed with a ring isomorphism. The $N/2$ complex values occupy the *slots* of m, so that homomorphic addition (HAdd) and multiplication (HMul) act componentwise on slots. Slot rotation (HRot) cyclically shifts the slot vector and is among the most expensive FHE operations, dominating the cost of standard homomorphic convolution.

Coefficient encoding and CinS. Data can alternatively be stored in the *coefficient* domain of m, i.e., directly as the coefficients of the polynomial, without applying the iDFT. NeuJeans [12] calls this *Coefficients-in-Slot* (CinS) encoding and exploits it to evaluate 2-D convolutions with a single HMul (see Section 2.3).

Noise and rescaling. CKKS is an *approximate* scheme: every plaintext carries a small encryption noise, and each HMul roughly squares the noise magnitude and doubles the scale factor. A *rescaling* operation divides the ciphertext by q_i/q_{i-1} and drops one level, keeping the scale under control.

2.2 CKKS Bootstrapping

Motivation. The multiplicative depth of a non-trivial CNN inference far exceeds the level budget supplied by a fresh ciphertext; bootstrapping must therefore be invoked several times per inference to restore consumed levels.

Standard pipeline. The standard CKKS bootstrapping procedure [6] consists of four steps:

1. **ModRaise.** The ciphertext at the bottom modulus q_0 is interpreted as if encrypted under a much larger modulus $Q \gg q_0$, introducing an error term $q_0 \cdot I(X)$ where $I(X)$ is a polynomial with small integer coefficients.
2. **CoeffToSlot (CtS).** A DFT-like linear map transforms the data from the coefficient domain to the slot domain, consuming $\approx \lfloor \log_2(N/2) \rfloor$ levels because each DFT butterfly must be scaled up to avoid rounding error, and each scale-up requires a rescaling.
3. **EvalMod.** A polynomial approximation of the modular reduction function (typically based on a scaled sin approximation) removes the integer part $q_0 \cdot I$ introduced by ModRaise, restoring approximate correctness. This step consumes $\approx \lceil \log_2 d_{\text{em}} \rceil$ levels for a degree-d_{em} approximation.
4. **SlotToCoeff (StC).** The inverse DFT maps data back to the coefficient domain, consuming the same level budget as CtS.

CtS and StC together account for roughly half the levels consumed per bootstrapping invocation; EvalMod accounts for the other half.

EvalRound. Kim et al. [15] identify the large DFT scaling factors in CtS/StC as the root cause of their level consumption: to prevent rounding errors during the DFT, each scaling factor must be large, and each post-DFT rescaling costs one level. Their EvalRound algorithm replaces EvalMod with a complementary operation: instead of removing $q_0 \cdot I(X)$ (modular reduction), EvalRound *recovers* $q_0 \cdot I(X)$ (rounding recovery), allowing the DFT factors in CtS/StC to be kept small without introducing plaintext error. This saves $\approx$ 2–3 multiplicative levels in CoeffToSlot per bootstrapping call compared to the standard pipeline.

Functional bootstrapping. Functional bootstrapping [31] modifies the target of the EvalMod polynomial: rather than computing modular reduction, it evaluates an arbitrary function (here, ReLU) while simultaneously performing the modular cleanup. The ReLU output falls in the same scale as the bootstrapped plaintext, so no additional level is consumed for the activation; the modified EvalMod step performs both tasks at the cost of one.

2.3 Homomorphic CNN Inference

Standard convolution via slot rotations. A feature map of shape (C_{in}, H, W) is typically packed into ciphertexts with each of the $N/2$ slots holding one element of the flattened tensor. Evaluating a $k \times k$ convolution with C_{out} output channels requires generating, for each filter weight position (i, j), a copy of the input ciphertext rotated by the corresponding slot offset, multiplied by the filter weight, and summed across all k^2 positions. The total cost is $O(k^2 C_{\text{in}} C_{\text{out}})$ HRot and HMul operations, with HRot typically dominating.

CinS convolution. Under CinS encoding [12], a 2-D feature map is stored in the coefficient domain. Convolution then corresponds to *polynomial multiplication modulo* $X^N + 1$: the filter polynomial encodes all k^2 weights and a single HMul implements the full spatial convolution. This reduces the per-layer rotation count from $O(k^2)$ to $O(1)$.

After a CinS convolution, the result is in the coefficient domain and must be converted back to the slot domain before the next operation. NeuJeans observe that this conversion is the CtS step of

bootstrapping; the two computations are therefore fused by design: no extra levels or time are spent on the CinS-to-slot conversion.

ReLU approximation. Since CKKS supports only polynomial operations, ReLU $f(x) = \max(0, x)$ must be approximated homomorphically. Composite polynomial approximation achieves ReLU in $\approx$ 13–14 multiplicative levels per activation layer, typically via a minimax-optimal sign-function approximation of degrees {15, 15, 27} [31]. In a standard CNN pipeline, this cost is incurred *after* each convolutional layer, followed immediately by a bootstrapping call to restore the level budget before the next layer.

Bootstrapping placement in CNN inference. In practice, bootstrapping is inserted at fixed points in the network graph where the remaining level budget falls below a threshold. Each bootstrapping call restores the full budget but introduces additional computation and latency. Minimizing the number and cost of bootstrapping calls is therefore the central optimization problem in FHE-based CNN inference.

3 OVERVIEW OF THE BOOTNET FRAMEWORK

We now present BootNet, our framework for FHE-based private CNN inference. BootNet compiles a CNN into a sequence of *fused bootstrapping units*, each of which performs exactly one CKKS bootstrapping invocation that simultaneously (i) refreshes the ciphertext noise budget, (ii) evaluates one convolutional layer, and (iii) applies the subsequent ReLU activation—with no separate homomorphic operations for any of these three tasks.

The design integrates three published bootstrapping techniques that target distinct, non-overlapping sub-steps of the bootstrap, plus one plaintext-side preparation step:

- **NeuJeans** [12]: CinS encoding fuses the $k\times k$ convolution into a kernel-fused SlotToCoeff (StC) DFT, eliminating the $O(k^2)$ slot rotations of standard homomorphic convolution (Section 4).
- **RBOOT** [31]: *functional bootstrapping* re-targets the EvalMod polynomial so that it evaluates ReLU and the modular-reduction step in a single fused approximation, eliminating the separate polynomial-ReLU pass that non-fused pipelines incur (Section 5.1).
- **EvalRound++** (Section 6.2, Appendix C): our slim-bootstrap adaptation of EvalRound+ [29], used to free CtS modulus headroom when targeting the aggressive $N = 2^{15}$ parameter set (Section 6.3).
- **SAPTQ** (Section 5.2, Appendix A): a Scale-Aligned Post-Training Quantization pass that aligns every pre-ReLU activation tensor with the polynomial-approximation domain $[-B, B]$ through forward-only int8 calibration followed by graph-level scale rewrites; the rewrites fold into existing convolution weights, so SAPTQ contributes no extra homomorphic operations at inference time.

We first describe the fused bootstrapping pipeline that underpins a single unit (Section 3.1), then explain how units chain into a complete CNN inference (Section 3.2), and finally forward-reference the modulus-budget optimizations engaged at the $N = 2^{15}$ parameter set (Section 3.3).

3.1 Fused Bootstrapping Pipeline

A standard CKKS bootstrapping invocation comprises four sequential stages—ModRaise, CtS, EvalMod, StC—as reviewed in Section 2.2. BootNet adopts the *slim* reordering of NeuJeans [12], which hoists StC to the front of each fused unit so that one unit's slot-domain output feeds directly into the next. A fused unit is therefore the composition

$$\mathsf{EvalMod}_{\mathrm{RBOOT}} \circ \mathsf{CtS} \circ \mathsf{ModRaise} \circ \mathsf{StC}_{\mathrm{kernel}},$$

in which the leading StC and the trailing EvalMod are modified to absorb a homomorphic convolution and a ReLU activation, respectively; the two middle stages are unchanged.

Stage 1: Convolution-fused StC. A unit consumes the slot-domain ciphertext produced by the previous unit's EvalMod. The kernel-fused StC simultaneously (i) maps the slot representation back to the coefficient representation natural to CinS encoding and (ii) multiplies by the precomputed convolution kernel, because both the slot-to-coefficient DFT and the $k\times k$ convolution can be expressed as a single linear map [12]. The kernel is folded into the StC matrix at parameter setup, so the fused stage consumes the same multiplicative levels as a plain StC—the $O(k^2)$ slot rotations of standard homomorphic convolution are eliminated entirely.

Stage 2: ModRaise (unchanged). The convolved coefficient-domain ciphertext at the bottom modulus q_0 is raised to a large modulus Q, introducing the additive error $q_0 \cdot I(X)$ that subsequent EvalMod will remove. This stage consumes no multiplicative levels.

Stage 3: CtS (slim DFT, EvalRound++ at N=2^{15}). A linear DFT brings the ciphertext from the coefficient domain back to the slot domain in preparation for EvalMod. At the aggressive $N = 2^{15}$ parameter set, the standard CtS is replaced by the EvalRound++ variant of Section 6.2, which trims the modulus consumed by the DFT scaling factors at the cost of a sign flip on the cosine-branch coefficients of Stage 4.

Stage 4: ReLU-fused EvalMod (functional bootstrapping). RBOOT [31] re-targets the EvalMod polynomial so that it evaluates ReLU and the modular reduction simultaneously. ReLU is decomposed as the sum of a sine branch and a cosine branch (Equation (1) of Section 5.1), each fitted by a degree-15 Chebyshev minimax polynomial. The fused stage consumes 13 multiplicative levels per call—approximately 5 levels above the standard EvalMod and 10–22 levels below the cost of a separate polynomial-ReLU pass [18, 31]. No additional levels or homomorphic operations are spent on ReLU after bootstrapping.

Prerequisite: bounded activations. The Stage 4 polynomial is fitted over a bounded interval $[-B, B]$. SAPTQ (Section 5.2) enforces $\|\mathbf{a}\|_\infty \le B$ at every pre-ReLU site through a forward-only int8 calibration on ImageNet plus four algebraic graph rewrites that fold every per-tensor scale into the adjacent convolution. A final uniform rescaling sets the effective pre-EvalMod quantization scale to $B/128$, sending every pre-ReLU tensor into $[-B, 127B/128] \subset [-B, B]$. The procedure changes no weights, costs no homomorphic operations at inference time, and incurs less than 0.2 percentage points of top-1 ImageNet accuracy across ResNet-18, ResNet-34, and ResNet-50 (Appendix A).

3.2 CNN Compilation into Fused Units

Given a CNN with L convolutional layers, each followed by a ReLU activation, BootNet maps the network to a chain of L fused units $U_1, U_2, \ldots, U_L$. Each unit U_ℓ executes the pipeline of Section 3.1 with the ℓ-th convolutional filter and outputs a slot-domain ciphertext whose data has been convolved, activated, and refreshed; this output feeds directly into $U_{\ell+1}$ as the input to its kernel-fused StC.

End-to-end flow.

1. **Client-side encoding.** The client encodes the input image in CinS format, encrypts it under CKKS, and sends the ciphertext together with the appropriate Galois keys to the server.
2. **Server-side inference.** The server evaluates $U_1 \to U_2 \to \cdots \to U_L$ sequentially. Apart from the bookkeeping operations described below (residual merges, downsampling), no rotations, separate polynomial evaluations, or standalone bootstraps are performed between units.
3. **Tail layers.** After U_L, the slot-domain output ciphertext is consumed by the average-pool and fully connected layers using standard slot-packing techniques [12].
4. **Client-side decoding.** The client decrypts the result ciphertext and reads the classification output.

Residual blocks (Section 4.3). A ResNet residual block computes $y = \mathrm{ReLU}\big(F(x)+\mathrm{shortcut}(x)\big)$. Because StC, ModRaise, and CtS are all $\mathbb{C}$-linear and EvalMod approximates the identity on the CKKS valid message range, BootNet's *confluent bootstrap* adds the two branches in the coefficient domain at level 0, immediately after their respective front-half StCs, and then applies a single shared ModRaise + CtS + EvalMod tail to the sum. The post-add ReLU is absorbed into the same shared $\mathrm{EvalMod}_{\mathrm{RBOOT}}$, so a projection-shortcut merge consumes one heavy tail rather than two (Figure 1); identity-shortcut blocks, whose shortcut path has no separate bootstrap to share, are unaffected.

Batch normalization. Folded into the preceding convolutional weights at export time (standard practice [20]); no homomorphic operation is needed.

Stride-2 downsampling (Section 4.2). Realized as a homomorphic linear transform whose diagonals are tuned so that every other coefficient of the output is structurally zero. This zero pattern lets the following fused unit elide the restore-zero-padding mask that NeuJeans normally inserts before StC, saving one multiplicative level on each affected unit—six per forward pass in ResNet-$\{18, 34, 50\}$.

3.3 Tightening the Modulus Budget for $N = 2^{15}$

The pipeline of Section 3.1 fits comfortably at the standard $N = 2^{16}$ parameter set. Halving the ring degree to $N = 2^{15}$ also halves the cost of every polynomial operation but cuts the available modulus budget correspondingly. Section 6 introduces two further optimizations that together recover the headroom needed to fit a fused unit at $N = 2^{15}$:

- **Mask fusion via per-slot coefficient pre-multiplication** (Section 6.1). Because CKKS polynomial evaluation is SIMD, the restore-zero-padding mask of Section 4.1 is folded directly into RBOOT's arcsine coefficients at parameter setup, saving one multiplicative level per residual-block merge—roughly 8% of the per-bootstrap budget—without any inference-time homomorphic multiplications.
- **EvalRound++** (Section 6.2). Adapting EvalRound+ [29] to BootNet's slim-bootstrap structure trims the modulus consumed by CtS via a complementary rounding-recovery polynomial, composing with RBOOT through a single sign flip on the cosine-branch coefficients (Appendix C).

Section 6.3 reports the resulting concrete-parameter set: $\log QP \approx 950$ at $N = 2^{15}$, combined with sparse-secret encapsulation using an ephemeral Hamming weight $\tilde{h} = 32$ to keep ModRaise sparse while moving the high-modulus keys to a denser main secret. This yields a 120+-bit security claim for BootNet-mini.

4 COEFFICIENT-IN-SLOT ENCODING: CHALLENGES AND OPTIMIZATIONS

NeuJeans's coefficient-in-slot (CinS) convolution promises an $\mathcal{O}(k^2) \to \mathcal{O}(1)$ reduction in slot rotations, but faithfully reproducing the full pipeline on Lattigo turns up details the original paper underspecifies. In this section we first document the implementation challenges we encountered during reproduction. We then propose a mask-elision optimization that saves one multiplicative level on each convolution following a homomorphic downsampling. Last but not least, we introduce the *confluent bootstrap*, which halves the heavy bootstrap tail at every projection-shortcut residual-block merge. Together the two optimizations cut wall time by approximately 20% over the faithful NeuJeans reproduction.

4.1 Implementation Challenges not Discussed in NeuJeans

Our pipeline begins by reproducing the NeuJeans [12] CinS-encoded convolution on Lattigo [26]. Although the high-level construction of [12] is well documented, several mathematical details had to be reconstructed, and several library-level extensions had to be written, before the reproduced pipeline matched the numerical precision reported in the original paper. We describe the most substantial gaps below; the two optimizations in Section 4.2 and Section 4.3 build on the reproduced baseline without altering any of these choices.

Kernel-fused StC matrix with custom stage distribution. The CinS convolution relies on a StC linear map whose precomputed matrix must simultaneously encode (i) the DFT itself, (ii) the convolution kernel, and (iii) a wrap-around constant that compensates for the padding pattern once the 2-D tensor is folded into the DFT. Lattigo's built-in DFT-matrix abstraction neither exposes a decode-reencode hook for fusing the kernel, nor accepts a custom per-stage level distribution. We extended both, and cross-checked the fused matrix against a plain StC baseline at each stage count we use.

1-D-as-2-D convolution indexing. NeuJeans packs a 2-D feature map as a 1-D polynomial and evaluates $k\times k$ convolutions as polynomial multiplications. The index-translation relations between kernel size, stride, padding, and channel grouping on the

one hand, and the resulting shift offsets on the other, are not tabulated in [12]; we derived them by hand for the 3×3, 1×1, and stride-2 7×7 kernels (with or without padding) required by ResNet-18, ResNet-34 and ResNet-50. Stride-2 was the most delicate: the "every-other-slot" intuition is correct only after a compensating rotation whose sign depends on the channel layout.

Real-in-Complex encoding and mask duplication. Folding the length-$2N$ real polynomial into a length-N complex polynomial—the *Real-in-Complex* (RinC) encoding, so named in parallel to NeuJeans' Coefficients-in-Slot (CinS)—halves the length of the sub-polynomials (i.e. the 1-D tensors), but forces every plaintext mask used downstream to be duplicated: the real and imaginary halves of each slot must be masked independently, and a naive two-plaintext-multiplication implementation would consume two multiplicative levels—one more than the pre-RinC routine.

Algorithm 1 and Algorithm 2 give the pre- and post-RinC mask routines, both of which consume a single level. In the RinC case, the single-level budget is recovered by choosing *real* mask plaintexts (equal to $\frac{1}{2}$ on the active positions of each half), and by using a conjugation-add to extract the real half and a conjugation-subtract to extract the imaginary half. Both conjugation and homomorphic addition are level-preserving, so the two plaintext multiplications in lines 1–2 of Algorithm 2 are executed on the same input ciphertext at the same level rather than in sequence, and the mask is completed before the single trailing rescale. A naive implementation that encoded the upper-half mask as the imaginary value $-\mathrm{i}/2$ would break this conjugation symmetry and force an additional multiplicative level.

Algorithm 1 Single-level mask (plain CinS).

Input: CinS ciphertext ct at level ℓ, scale Δ; mask plaintext pt_m encoding $m \in \{0,1\}^n$ at level ℓ with scale q_ℓ.
Output: ct' at level $\ell - 1$, scale Δ, with $\mathrm{ct}'_i = m_i \cdot \mathrm{ct}_i$.
1: $\mathrm{ct}' \leftarrow \mathrm{Mult}_{\mathrm{pt}}(\mathrm{ct}, \mathrm{pt}_m)$ ▷ level ℓ
2: $\mathrm{ct}' \leftarrow \mathrm{Rescale}(\mathrm{ct}')$ ▷ level $\ell - 1$

Algorithm 2 Single-level mask (RinC-CinS).

Input: RinC-CinS ciphertext ct at level ℓ, scale Δ, encoding $p \in \mathbb{R}^{2n}$ as slots $\tilde{p}_i = p_i + \mathrm{i}\, p_{i+n}$; real mask plaintexts $\mathrm{pt}_{m_\mathrm{R}}, \mathrm{pt}_{m_\mathrm{I}}$ at level ℓ, scale q_ℓ, each equal to $\frac{1}{2}$ on the active positions of the respective half and 0 elsewhere.
Output: ct' at level $\ell - 1$, scale Δ, with the mask applied jointly to both halves of p.
1: $\mathrm{ct}_\mathrm{R} \leftarrow \mathrm{Mult}_{\mathrm{pt}}(\mathrm{ct}, \mathrm{pt}_{m_\mathrm{R}})$ ▷ level ℓ
2: $\mathrm{ct}_\mathrm{I} \leftarrow \mathrm{Mult}_{\mathrm{pt}}(\mathrm{ct}, \mathrm{pt}_{m_\mathrm{I}})$ ▷ level ℓ, same ct
3: $\mathrm{ct}_\mathrm{R} \leftarrow \mathrm{Add}\big(\mathrm{ct}_\mathrm{R}, \mathrm{Conj}(\mathrm{ct}_\mathrm{R})\big)$ ▷ extract real half
4: $\mathrm{ct}_\mathrm{I} \leftarrow \mathrm{Sub}\big(\mathrm{ct}_\mathrm{I}, \mathrm{Conj}(\mathrm{ct}_\mathrm{I})\big)$ ▷ extract imag. half
5: $\mathrm{ct}_\mathrm{R} \leftarrow \mathrm{Rescale}(\mathrm{ct}_\mathrm{R});\ \mathrm{ct}_\mathrm{I} \leftarrow \mathrm{Rescale}(\mathrm{ct}_\mathrm{I})$ ▷ level $\ell - 1$
6: $\mathrm{ct}' \leftarrow \mathrm{Add}(\mathrm{ct}_\mathrm{R}, \mathrm{ct}_\mathrm{I})$

Overcoming these trickiness above enables us to reproduce the baseline construction of [12]; in particular, the flexibility and the end-to-end numerical precision are preserved. The next two subsections identify two aspects of this baseline that admit further speedup without altering either.

Table 1: Cost of the post-activation mask step per affected conv call, before and after the elision; bottom rows report the cascaded end-to-end effect on ResNet-34 / ImageNet at $N=2^{16}$.

	Before	After
Per affected conv call		
Plaintext multiplications	2	0
Conjugations	2	0
Rescales	2	0
Mult. levels consumed	1	0
End-to-end (ResNet-34, ImageNet, $N=2^{16}$)		
Linear-transform cache	235 GB	226 GB
Wall time / image	74 min	68 min
Speedup	—	≈9%

4.2 Eliding the Restore-Zero-Padding Mask after Downsampling

In the baseline NeuJeans pipeline, every fused conv+bootstrap call applies a pair of *restore-zero-padding* plaintext multiplications immediately before the final StC—the real and imaginary RinC masks of Algorithm 2, which in NeuJeans' Algorithm 2 [12] zero the slots left occupied by the padded conv-as-polynomial encoding. We observe that this mask is redundant when the conv's input comes directly from a homomorphic downsampling linear transform—specifically, the stride-2 split that separates a ResNet block-0 boundary, or the downsampling pool used in the same position—and can be elided at no correctness cost, saving one multiplicative level per affected call.

Why the mask is redundant. The downsampling transforms are realized as homomorphic matrix–vector products whose diagonals are constructed to write only to valid positions of the smaller output tensor; every other coefficient of the output polynomial is *structurally zero*, not merely noise-level zero. Multiplying this clean output by a restore-zero-padding mask therefore performs no nonzero-to-zero masking—it only scales the already-valid positions by the mask value (here $\frac{1}{2}$, the RinC half-weight used for the conjugation-split identity of Algorithm 2). This scaling is equivalent to folding $\frac{1}{2}$ into the upstream matrix diagonals, and our precomputed split and pool matrices already carry this $\frac{1}{2}$ factor, so the post-activation mask only re-applies a factor already absorbed and can be removed with no compensating change on the inference side.

Cost savings. Table 1 summarizes the per-call savings and their end-to-end effect. The saved level cascades into the subsequent StC and bootstrap and into a smaller disk-cached linear-transform precomputation—the dominant source of the end-to-end speedup, since the direct cost of the two plaintext multiplications and two rescales is small relative to the bootstrap itself.

Scope. The optimization applies to exactly those convolutions whose producers guarantee structural zeros at padding positions. In ResNet-{18, 34, 50} these are the block-0 main-path conv and the

block-0 shortcut conv of layers 2, 3, and 4—six conv calls per forward pass. All other convolutions retain the RinC masks of Algorithm 2, since their producers cannot produce a structurally clean output. We verify correctness empirically by comparing per-layer L_∞ output error against the masked baseline, finding no deviation beyond the rounding noise already present in the reproduction.

4.3 Decoupling the Bootstrap for Residual-Block Merges

A ResNet residual block evaluates $y = \mathrm{ReLU}(F(x) + \mathrm{shortcut}(x))$. The naive NeuJeans schedule for this merge is Figure 1a: each branch completes a full bootstrap— the kernel-fused StC followed by the heavy tail ModRaise + CtS + EvalMod—the two branches are added post-tail in the slot domain, and a polynomial ReLU is applied to the sum. We observe that, because StC is $\mathbb{C}$-linear, both branches can instead be added *in the coefficient domain at level* 0, right after their StC front halves and *before* the heavy tail runs. A single shared tail on the summed ciphertext then suffices (Figure 1b), halving the ModRaise + CtS + EvalMod work at every residual-block merge. We call this the *confluent bootstrap*: two homomorphic branches confluence into a single shared bootstrap tail.

Construction. We split the fused bootstrap into a front half that stops in coefficient domain at level 0, and a tail that runs the remaining ModRaise + CtS + EvalMod. The main-path front half is the kernel-fused StC of NeuJeans; the shortcut branch invokes a lightweight companion that applies a plain StC (no convolution, no restore-zero-padding mask) whose matrix scale is tuned so that both branches exit at the identical coefficient-domain scale. The merge is a single homomorphic addition at level 0, followed by one shared tail on the sum. Writing $\mathsf{T} = \mathsf{EvalMod} \circ \mathsf{CtS} \circ \mathsf{ModRaise}$ for the heavy tail (so a full bootstrap factors as $\mathsf{T} \circ \mathsf{StC}$), the confluent-bootstrap transformation from Figure 1a to Figure 1b is precisely the algebraic identity

$$\mathsf{T}\big(\mathsf{StC}(\mathrm{ct}_1)\big) + \mathsf{T}\big(\mathsf{StC}(\mathrm{ct}_2)\big) \equiv \mathsf{T}\big(\mathsf{StC}(\mathrm{ct}_1) + \mathsf{StC}(\mathrm{ct}_2)\big),$$

i.e., *full-bootstrap each branch and then add* equals *add the front-half outputs and then share a single tail.* The identity holds because StC, ModRaise, and CtS are $\mathbb{C}$-linear, and EvalMod approximates the identity on the CKKS valid message range.

Bonus: composition with RBOOT. The split isolates all modular-reduction work inside the tail and places the merge's Add *before* ModRaise. This is exactly the handoff point that an RBOOT-style functional EvalMod (Section 5.1) expects as input, so the shared tail can be replaced by an RBOOT tail with no change to the front halves. The post-add ReLU that the naive schedule evaluates as a separate polynomial is then absorbed into the same shared tail (Figure 1c); the merge costs a single bootstrap tail total, including the residual block's ReLU.

Cost savings. The per-merge primitive counts are summarized in Table 2. The added plain StC on the shortcut is an order of magnitude cheaper than a full bootstrap tail—a single DFT linear map with no modular reduction—so it does not come close to offsetting the halved tail.

We take the micro-benchmark on ResNet-18 at $\log N = 16$ as an example. Each of layers 1–4 in ResNet-18 contains two basic blocks:

Table 2: Per projection-block merge: primitives executed in the naive NeuJeans schedule (Figure 1a) vs. the confluent-bootstrap schedule (Figure 1b).

	Before	After
ModRaise + CtS + EvalMod	×2	×1
StC front halves	×2	×2
Level-0 branch-merge Add	0	1
Shortcut StC (plain)	0	1

Table 3: ResNet-18 per-layer wall time at $\log N = 16$ (hot run), with and without the confluent bootstrap.

	Layer 1	Layer 2	Layer 3	Layer 4
w/o confluent bootstrap	6m22s	10m04s	3m57s	11m19s
with confluent bootstrap	6m22s	8m34s	3m01s	10m45s

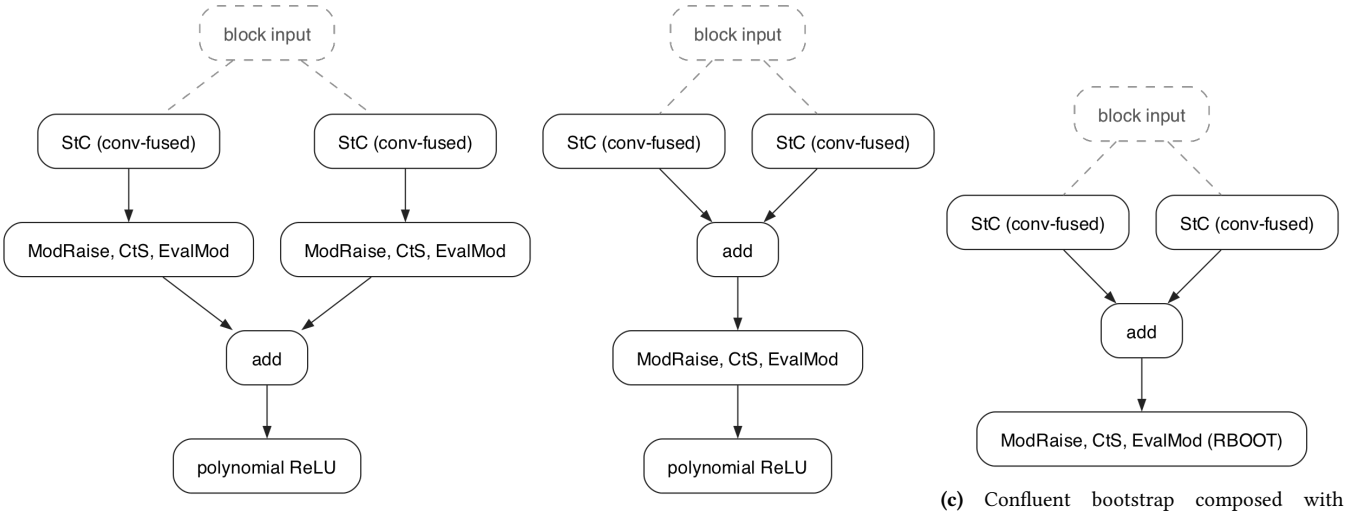
layer 1 uses identity shortcuts on both blocks (no channel growth), while layers 2–4 each use one projection shortcut on the first block and one identity shortcut on the second. Table 3 reports the per-layer wall time before and after applying the confluent bootstrap. Aggregated over the four layers, the confluent bootstrap reduces wall time by approximately 10%; layer 1, with no projection shortcut, is unaffected.

Scope. The optimization applies at every projection-shortcut merge in ResNet-{18, 34, 50}—the first block of layers 2–4—and is structurally agnostic to basic block (used in shallow ResNet) vs. bottleneck block (used in deep ResNet such as ResNet-50) design. It does not interact with the mask-elision optimization of Section 4.2: the two target disjoint call sites and compose by inclusion.

5 ADAPTING RBOOT FOR 8-BIT FUNCTIONAL BOOTSTRAPPING

RBOOT [31] as published targets ~14-bit ReLU precision and assumes that the message entering EvalMod already lies within $[-q_0/4, q_0/4]$. Neither assumption survives a direct port into BootNet: the level budget freed by the fused pipeline of Section 3.1 cannot accommodate a 14-bit polynomial, and generic CNN pre-activations do not respect the half-range. Section 5.1 relaxes the precision target to ~8 bits—sufficient for accurate CNN inference [31]—via two degree-15 arcsine polynomials and a zero-level activation pre-scaling, while Section 5.2 enforces the activation bound through a lightweight, forward-only post-training quantization pass.

Unlike prior fits on the relaxed interval $[-1+\epsilon,\ 1-\epsilon]$ [18, 31]—which sidestep the arcsine singularities at the cost of localized error bumps at the kink x=0 and the input-domain boundary $x = \pm q_0/4$—the full-domain minimax fit of Section 5.1 achieves a uniform worst-case error across the entire activation domain. The resulting top-1 accuracy drop is therefore analytically tractable due to the regularity of the full-domain minimax error, which means we are able to give a reasonable interpretation of the accuracy drop of the homomorphic inference (Appendix B). On the other hand, our practices show that it is the growth trend across layers rather than the absolute scale of errors that determines the final accuracy. For instance there is only a 0.2 percentage points drop in accuracy even with 7-bit (i.e. let $B = 1$ in 5.2) quantized ReLU .



(a) NeuJeans baseline. **(b)** Confluent bootstrap (Section 4.3). **(c)** Confluent bootstrap composed with RBOOT (Section 5.1).

Figure 1: Bootstrap schedule at a ResNet residual-block merge. (a) naive NeuJeans; (b) confluent bootstrap (Section 4.3); (c) confluent bootstrap composed with RBOOT (Section 5.1), absorbing the post-add ReLU into the shared tail.

5.1 Functional ReLU Polynomial: Full-Domain Minimax with Range Pre-Scaling

For a message $x \in [-q_0/4,\ q_0/4] + q_0\mathbb{Z}$ at the bottom modulus q_0, RBOOT [31] decomposes ReLU as

$$\mathrm{ReLU}(x \bmod q_0) = \underbrace{\tfrac{1}{4\pi}\ \arcsin\big(\sin(2\pi x/q_0)\big)}_{\text{sine branch producing } (x \bmod q_0)/2} + \underbrace{\tfrac{1}{8\pi}\ \arcsin\big(-\cos(4\pi x/q_0)\big) + \tfrac{1}{16}}_{\text{cosine branch producing } |x \bmod q_0|/2}, \tag{1}$$

where the inner cosine is obtained from the sine that EvalMod already produces via one extra double-angle iteration [31]. Although both branches share the outer $\arcsin(\cdot)$, their inner arguments traverse different sub-intervals of $[-1,\ 1]$, and BootNet therefore fits *two distinct* degree-15 Chebyshev minimax polynomials $P_{\sin}$ and $P_{\cos}$, one per branch; their coefficients appear in Appendix E. The full functional ReLU consumes $1 + 4 = 5$ extra multiplicative levels per bootstrapping (the leading 1 is for deriving the required cosine function basing on double angle formula) which is 10–22 levels below a separate polynomial-ReLU pass [18, 31].

Full-domain minimax: a first. The minimax guarantee for $P_{\cos}$ holds on the *entire* interval $y \in [-1,\ 1]$, with no ϵ-neighborhood of the arcsine singularities excluded. Prior work [18, 31] fits on the relaxed interval $[-1{+}\epsilon,\ 1{-}\epsilon]$ because Remez fitting near the singularities is numerically delicate—but in ReLU coordinates $y = -1$ is the kink $x = 0$ and $y = +1$ is the input-domain boundary $x = \pm q_0/4$: precisely where ReLU is most sensitive. RBOOT's deployed configurations (Param-IIIV) degrade to ~ 8.74-bit precision inside the excluded zones [31]. BootNet's refined Remez fit achieves a uniform $2^{-7.399}$ maximum error on all of $[-1,\ 1]$, so the corresponding ReLU maximum error is $2^{-8.399}$—comfortably above the 8-bit target and with no error bump at the kink or boundary.

Range pre-scaling, absorbed into coefficients. To reach ~ 14-bit sine-branch precision at degree 15, $P_{\sin}$ is fit on the tighter domain $|x|/q_0 \le 0.2$ rather than the full 0.25 that $P_{\cos}$ tolerates—leaving $P_{\cos}$ as the accuracy bottleneck. We respect the tighter sine-branch domain with a one-time plaintext pre-scale of 0.8 applied at each fused bootstrap's CinS input; by positive homogeneity of ReLU, a 1.25× recovery restores the output, and both factors fold into the arcsine coefficients at parameter setup—consuming no multiplicative level. The pre-scale further folds into SAPTQ (Section 5.2) by re-targeting the activation bound B to $0.8\,B$ in the uniform rescaling step, so no new calibration constants or homomorphic multiplications are introduced. The pre-scale costs $\log_2(1/0.8) \approx 0.32$ bits, still leaving $>$ 8-bit precision over the full $[-1,\ 1]$ activation domain.

5.2 Taming Activation Ranges via Scale-Aligned Post-Training Quantization

CKKS-based homomorphic ReLU evaluation relies on a polynomial approximation fitted over a bounded interval $[-B,\ B]$ [18]: if any activation v satisfies $|v| > B$ at evaluation time, the polynomial diverges and the layer output becomes meaningless. Every plaintext model entering BootNet must therefore guarantee $\|\mathbf{a}\|_\infty \le B$ at every layer—a property that standard training does not enforce.

Prior work addresses this via per-layer scaling folded into the weights [20] (which costs one extra homomorphic multiplication per layer and requires re-calibration on weight changes), range-aware training with a dedicated penalty loss [5, 14] (which demands bespoke training pipelines), or replacing ReLU with a polynomial activation [11] (which degrades accuracy on deeper networks).

We instead use *Scale-Aligned Post-Training Quantization* (SAPTQ), a lightweight alternative that retains the pretrained weights unchanged and aligns the plaintext model's pre-ReLU activation domain with $[-B, B]$ through a sequence of value-preserving graph rewrites. A single forward-only calibration pass over a held-out input subset selects a symmetric int8 scale s_ℓ for each pre-ReLU tensor (no gradient update of any kind). A sequence of algebraic rewrites then (i) folds every calibrated s_ℓ into the adjacent convolution's weight and bias, exploiting the positive homogeneity of ReLU, and (ii) applies a uniform scalar rescaling across all per-activation scales so that the effective pre-ReLU quantization scale becomes $s_{\text{pre-ReLU}} = \frac{B}{128}$, sending every pre-ReLU tensor into $[-B, 127\,B/128] \subset [-B, B]$ as required by the homomorphic polynomial. Because the rewrites are exact algebraic identities, the handed-off plaintext network is functionally identical to the calibrated int8 post-training-quantized model; and because all per-activation scales have been folded into convolution parameters, no additional homomorphic multiplication is needed at inference time.

On ImageNet, SAPTQ costs less than 0.2 percentage points of top-1 accuracy across ResNet-18/34/50 relative to the float baseline (Appendix A). The residual accuracy loss is entirely due to the int8 calibration step; the subsequent graph rewrites are lossless.

6 TIGHTENING THE MODULUS BUDGET

This section explores optimizations for further reducing BootNet's modulus consumption.

6.1 Integrating RBOOT with RinC-CinS

Two structural properties simplify the integration of RBOOT's functional ReLU into the RinC-CinS slim bootstrap of Section 4.3. First, slim bootstrapping's CtS already emits its output as a pair of real-valued ciphertexts (one per halved-RinC channel), which is exactly the pure-real input format on which RBOOT's polynomial EvalMod is defined—no extra step is needed to bridge the encodings. Second, because CKKS polynomial evaluation is SIMD, a per-slot scalar pre-multiplication of the arcsine coefficients fuses any 0/1 masking of the ReLU output *into the polynomial evaluation itself*, absorbing the restore-zero-padding mask of Section 4.1 at no extra multiplicative level.

Split-to-real for RBOOT input. Given a RinC-CinS ciphertext ct at the input of the bootstrap tail—coefficient domain at level 0, slot i holding $\tilde{p}_i = p_i + \mathrm{i}\, p_{i+n}$—the slim tail proceeds as Algorithm 3. After ModRaise and CtS, a conjugation-add and a conjugation-subtract extract the two real-valued halves $\mathrm{ct_R}$ (carrying $p_0, \ldots, p_{n-1}$) and $\mathrm{ct_I}$ (carrying $p_n, \ldots, p_{2n-1}$). RBOOT's polynomial EvalMod is then applied independently to each real branch, and the two outputs are recombined into a single RinC ciphertext whose slot i holds $\mathrm{ReLU}(p_i) + \mathrm{i}\,\mathrm{ReLU}(p_{i+n})$. Neither the split nor the recombination consumes multiplicative levels beyond the standard cost of slim CtS.

Mask fusion via per-slot coefficient pre-multiplication. RBOOT's polynomial EvalMod (Section 5.1) ends with two degree-15 Chebyshev minimax polynomials $P_{\sin}$ and $P_{\cos}$. Absorbing the additive constant of Equation (1) into the zeroth coefficient of $P_{\cos}$, we write the combined evaluation schematically as $f(y_{\sin}, y_{\cos}) = P_{\sin}(y_{\sin}) + P_{\cos}(y_{\cos})$. Because CKKS polynomial evaluation is SIMD, the evaluator applies the same coefficient plaintext at every slot in parallel. We observe that this plaintext does not have to be slot-uniform: if the plaintext carrying c_j is replaced by one whose slot i holds $k_i \cdot c_j$ for a per-slot scalar k_i, then by linearity of the polynomial evaluator—and independently of the baby-step/giant-step schedule—the output at slot i becomes $k_i \cdot f(y_{\sin,i},\ y_{\cos,i})$, at the same multiplicative-level cost as the unmasked evaluation.

Algorithm 3 Slim RBOOT bootstrap on a RinC-CinS ciphertext. Scalar factors of 2 from the conjugation identities are folded into the EvalMod scaling constants and omitted below for clarity.

Input: Coefficient-domain ciphertext ct at level 0, encoding $p \in \mathbb{R}^{2n}$ as RinC slots $\tilde{p}_i = p_i + \mathrm{i}\, p_{i+n}$.

Output: ct' refreshed, RinC-encoding $\mathrm{ReLU}(p_i) + \mathrm{i}\,\mathrm{ReLU}(p_{i+n})$ at slot i.

$\mathrm{ct} \leftarrow \mathsf{ModRaise}(\mathrm{ct})$ ▷ lift modulus
$z \leftarrow \mathsf{CtS}(\mathrm{ct})$ ▷ to slot domain
$\mathrm{ct_R} \leftarrow \mathsf{Add}\big(z, \mathsf{Conj}(z)\big)$ ▷ real half
$\mathrm{ct_I} \leftarrow -\mathrm{i} \cdot \mathsf{Sub}\big(z, \mathsf{Conj}(z)\big)$ ▷ imag half
$\mathrm{ct_R} \leftarrow \mathsf{EvalMod}_{\mathrm{RBOOT}}(\mathrm{ct_R})$ ▷ ReLU on real
$\mathrm{ct_I} \leftarrow \mathsf{EvalMod}_{\mathrm{RBOOT}}(\mathrm{ct_I})$ ▷ ReLU on imag
$\mathrm{ct}' \leftarrow \mathsf{Add}\big(\mathrm{ct_R}, \mathrm{i} \cdot \mathrm{ct_I}\big)$ ▷ recombine RinC

For a 0/1 indicator mask $k_i \in \{0, 1\}$, the post-bootstrap output carries $\mathrm{ReLU}(p_i)$ at active slots and 0 at masked slots— exactly the effect that Algorithm 2 achieves as a separate pre-bootstrap step. Algorithm 4 gives the resulting masked $\mathsf{EvalMod}_{\mathrm{RBOOT}}$ subroutine; plugging it into the two RBOOT calls of Algorithm 3 (with per-branch masks k_{R} and k_{I}, respectively) absorbs the entire restore-zero-padding mask of Section 4.1 into the existing RBOOT tail. The per-call savings are the primitives enumerated in the top rows of Table 1: two plaintext multiplications, two conjugations, two rescales, and one multiplicative level. Although a single multiplicative level may look small in absolute terms, it is far from marginal in the BootNet setting: the functional bootstrap of Section 5.1 already consumes 13 multiplicative levels per call, so freeing one level per residual-block merge amounts to roughly 8% of the per-bootstrap budget—slack that either accommodates deeper networks at the current ciphertext modulus or, as in Section 6.3, allows us to fit an entire bootstrap at $N = 2^{15}$ instead of $N = 2^{16}$ for the same network.Unlike the downsampling-specific elision of Section 4.2, which only applies when the producer is a known-clean linear transform, the fused-mask saving here is available at *every* merge point that hands a ciphertext to the RBOOT tail.

6.2 EvalRound++: Slim-Bootstrap EvalRound+

EvalRound+ [29] reduces CtS/StC modulus consumption in conventional CKKS bootstrapping by replacing EvalMod with a complementary rounding polynomial whose error cancels against a high-precision skip connection. We adapt this technique to BootNet's slim bootstrap (where StC is hoisted out of the tail) and to RBOOT's functional $\mathsf{EvalMod}_{\mathrm{RBOOT}}$, calling the resulting algorithm *EvalRound++*. RBOOT composition costs only a sign flip on the cosine-branch coefficients of RBOOT's arcsine polynomial. With

Algorithm 4 Mask-fused $\mathsf{EvalMod}_{\mathrm{RBOOT}}$ on a real ciphertext (one branch of Algorithm 3 lines 5–6). Pre-multiplying every arcsine coefficient by the mask vector k makes the polynomial evaluator emit $k_i \cdot f(y_i)$ at slot i, regardless of the BSGS schedule.

Input: Real ciphertext ct (post-CtS), encoding $(p_0, \ldots, p_{n-1})$ at slot i; arcsine polynomials $P_{\sin}, P_{\cos}$ with scalar coefficients $\{c_j^{\sin}\}$, $\{c_j^{\cos}\}$ (additive constant absorbed into $c_0^{\cos}$); mask $k \in \{0,1\}^n$.

Output: ct′ with slot i holding $k_i \cdot \big(P_{\sin}(y_{\sin,i}) + P_{\cos}(y_{\cos,i})\big) \approx k_i\,\mathrm{ReLU}(p_i)$.

Parameter setup (once per mask):

for each coefficient index j **do**
  $\mathrm{pt}_j^{\sin} \leftarrow \mathsf{Encode}\big((k_i \cdot c_j^{\sin})_{i=0}^{n-1}\big)$
  $\mathrm{pt}_j^{\cos} \leftarrow \mathsf{Encode}\big((k_i \cdot c_j^{\cos})_{i=0}^{n-1}\big)$
end for

Runtime:

$y_{\sin} \leftarrow$ Chebyshev eval + double-angle(ct) ▷ $\approx \sin(2\pi p_i/q_0)$
$y_{\cos} \leftarrow$ one extra double-angle($y_{\sin}$) ▷ $\approx -\cos(4\pi p_i/q_0)$
$\mathrm{ct}_{\sin} \leftarrow \mathsf{PolyEval}\big(y_{\sin}; \{\mathrm{pt}_j^{\sin}\}\big)$
$\mathrm{ct}_{\cos} \leftarrow \mathsf{PolyEval}\big(y_{\cos}; \{\mathrm{pt}_j^{\cos}\}\big)$
$\mathrm{ct}' \leftarrow \mathsf{Add}(\mathrm{ct}_{\sin}, \mathrm{ct}_{\cos})$

EvalRound++, we are able to further shrink the size of total modulus $\log QP$ to ~950. Due to space limits, the full derivation, algorithm block, and control-flow figure could be found in Appendix C.

6.3 BootNet-Mini: Working under $N = 2^{15}$

Reducing the ring degree from $N = 2^{16}$ to $N = 2^{15}$ would greatly reduce the computational and storage cost for every FHE operation. However, it also halves the available modulus budget, which makes it challenging for complex workloads such as CNN inference. Consequently, prior work has predominantly relied on $N = 2^{16}$ or even $N = 2^{17}$. We present and analyze the the parameters of BootNet-mini, the first FHE-based CNN inference that works under $N = 2^{15}$.

The sparse-secret bottleneck. Following the Lattigo defaults [26] and Bossuat et al. [3], the usual efficient CKKS bootstrap uses a sparse ternary secret, often with Hamming weight $h = 192$. This choice is attractive because the ModRaise artifact that EvalMod must remove is controlled by the secret distribution: a smaller h keeps the modular-reduction interval narrow, reducing the degree and failure probability of the bootstrapping polynomial. The drawback is that the same sparse secret also constrains the largest ciphertext modulus that can be used for the high-level bootstrapping keys. For BootNet-mini, whose fused convolution–ReLU bootstrap still needs a total modulus around $\log QP \approx 950$ after EvalRound++ and the CinS masking–RBOOT fusion, this coupling becomes the main obstacle to a standard-security $N = 2^{15}$ parameter set.

Sparse-secret encapsulation. Bossuat et al. [4] propose a simple way to decouple these two roles. Let s be the main secret used by the high-modulus bootstrapping keys, and let $\tilde{s}$ be a much sparser ephemeral secret. For BootNet-mini, we set the ephemeral Hamming weight to $\tilde{h} = 32$, reducing the sparse-secret density used during ModRaise from the conventional $h = 192$ setting. Immediately before ModRaise, the low-level ciphertext is key-switched from s to $\tilde{s}$; ModRaise is then performed under $\tilde{s}$; immediately afterwards, the ciphertext is key-switched back from $\tilde{s}$ to s. The unwanted multiple of the small modulus introduced by ModRaise is therefore governed by the sparse ephemeral secret, so EvalMod keeps the low-complexity behavior of a sparse-key bootstrap. At the same time, the high-modulus evaluation keys for the rest of the bootstrapping circuit can be generated under a denser main secret, so the large $\log QP$ budget no longer depends on the sparse secret.

Implication for BootNet-mini. We use this as a parameter-level wrapper around BootNet-mini rather than as another online arithmetic primitive. The wrapper adds only two key-switches around ModRaise; it does not increase the multiplicative depth of RBOOT, EvalRound++, or the CinS linear transforms, and these extra key-switches are small relative to the rotation-heavy CtS/StC stages. Thus BootNet-mini retains the circuit cost of a sparse-secret bootstrap while moving its high-modulus security margin to a denser main secret. With sparse-secret encapsulation, the $N = 2^{15}$ BootNet-mini parameter set can therefore support its $\log QP \approx 950$ fused bootstrap while reaching 120+ bits of security.

7 EVALUATION

We evaluate BootNet and BootNet-mini against the state-of-the-art FHE-based CNN inference framework Orion [10]. Orion is the most comprehensive open-source baseline, supporting ResNet variants from ResNet-20 (CIFAR-10) through ResNet-50 (ImageNet). We compare end-to-end inference latency, peak memory consumption, storage requirements, and numerical precision.

Note that beyond the original ResNet models with ReLU activation, Orion also includes SiLU-finetuned variants. Since SiLU is smoother for polynomial approximation, their SiLU versions achieve better efficiency. In contrast, BootNet does not require activation finetuning.

7.1 Experimental Setup

Hardware. All experiments run on a single server equipped with one Intel Xeon 6982P-C processor (96 physical cores / 192 SMT threads, 3.9 GHz max boost), 768 GB RAM, and a 2 TB NVMe SSD, running Ubuntu 22.04.3 LTS (kernel 5.15). Note that the numbers of physical cores and hardware threads are irrelevant to the evaluation of BootNet or Orion, since both works are single-threaded during the online phase.

Software. Orion is built on the Lattigo v6 CKKS library [26] (Go 1.22), with Python bindings via ctypes. All Orion experiments use the default CKKS parameters from the Orion repository: $\log N = 16$, $\log \Delta = 40$, Hamming weight $h = 192$. BootNet is implemented in Go directly on the same Lattigo v6 backbone, extended in-tree with the kernel-fused StC matrix constructors, RinC encoding utilities, and RBOOT EvalMod described in Section 4.1 and Section 5.1; sharing a single CKKS substrate ensures the comparison reflects algorithmic differences rather than backend implementation. We benchmark BootNet at both $\log N = 16$ (matching Orion) and the more aggressive BootNet-mini with $\log N = 15$ ($\log QP = 950$) enabled by our modulus tightening (Section 6.3); both regimes use

Table 4: Per-stage breakdown of BootNet's ciphertext modulus chain at $\log N = 16$ (in bits, base 2). "Rest" denotes the slot-domain budget left to user-circuit operations; Convolution is fused into StC and ReLU is fused into EvalMod.

Base	StC	CtS	EvalMod	Rest	$\log p$
55	45×2	56×4	60×13	45×2	61×2

$\log \Delta = 40$. The $\log N = 16$ setting follows the common sparse-ternary $h = 192$ key, while BootNet-mini's security claim uses the sparse-secret encapsulation wrapper with ephemeral Hamming weight $\tilde{h} = 32$ described in Section 6.3.

Modulus chain. Table 4 reports the per-stage breakdown of BootNet's ciphertext modulus chain at $\log N = 16$. EvalMod is the dominant stage and absorbs ReLU via RBOOT's functional bootstrap (Section 5.1). The kernel-fused StC of Section 4 folds the convolutional weights directly into the DFT matrix, so no separate slot-domain levels are needed for the convolution itself; the "Rest" column holds only the residual budget required for layer 1's three-stage StC and the in-place mask. The aggregate $\log QP \approx 1361$ is well within the 128-bit-secure envelope of $\log QP \leq 1546$ for $h = 192$ established by Bossuat et al. [3].

Benchmarks. We evaluate on the following model–dataset combinations, covering ResNet variants from 18 to 50 layers on ImageNet (224×224):

- **ResNet-18** on ImageNet (224×224, $\log N \in \{15, 16\}$)
- **ResNet-34** on ImageNet (224×224, $\log N \in \{15, 16\}$)
- **ResNet-50** on ImageNet (224×224, $\log N \in \{15, 16\}$)

Orion reports ResNet-50/ImageNet results in their paper (32324 s inference [10]), but we were unable to complete a full reproduction of this case. Our attempt crashed during the offline compilation phase after 9 hours when the precomputed diagonal file exceeded the 2 TB disk capacity at 1.78 TB—with only ∼40% of the layers serialized, implying a final storage requirement of ≥4 TB. The offline compilation also reached 642 GB peak RSS (approaching the 768 GB server limit) and required 342 bootstrap operations. We include Orion's self-reported ResNet-50 numbers in our comparison tables for completeness.

7.2 End-to-End Inference Latency

Table 5 compares the online (inference-only) latency of BootNet against Orion. All Orion baselines are reproduced on the same hardware as BootNet for a fair comparison (see Appendix D for a hardware calibration against the times reported by the original authors).

Several observations stand out. First, Orion's latency scales steeply with model depth: ResNet-34/ImageNet takes 14967 s (4.2 h) in our reproduction, roughly 2.0× slower than ResNet-18/ImageNet (7547 s) despite having only 2× more convolutional layers. This is driven by (i) the quadratic growth of slot-rotation counts with increasing feature-map size, (ii) more bootstrap insertions, and (iii) larger linear-transform diagonals at deeper layers.

Second, Orion's ReLU-based inference is consistently 21–23% slower than the SiLU variant on identical architectures, because the composite Sign polynomial for ReLU requires approximately 15 multiplicative levels per activation versus 7 levels for SiLU(127).

Table 5: Online FHE inference latency (seconds) on ImageNet. All Orion values are reproduced on our server, except † which denotes we cannot reproduce the results and report values derived from their original paper.

Model	$\log N$	Act.	Orion [10]	BootNet	Speedup
ResNet-18	16	SiLU	7547	3268	2.31×
ResNet-18	16	ReLU	9108		2.79×
ResNet-34	16	SiLU	14967	4097	3.65×
ResNet-34	16	ReLU	18382		4.49×
ResNet-50	16	SiLU	32324†	10701	3.02×†
ResNet-50	16	ReLU	—		—
				BootNet-mini	
ResNet-18	15	ReLU	—	3774	2.43×
ResNet-34	15	ReLU	—	4949	3.71×
ResNet-50	15	ReLU	—	14154	—

This forces roughly 2× more bootstrap insertions (e.g., 250 vs. 120 for ResNet-34/ImageNet), each of which is itself the most expensive operation in the pipeline. Notably, the Orion paper reports ImageNet-scale results only for SiLU, omitting the ReLU latency entirely.

In contrast, on ResNet-18/ImageNet ($\log N = 16$), BootNet completes inference in 3268 s (54.5 min), achieving a 2.31× speedup over Orion's SiLU mode and 2.79× over Orion's ReLU mode. On ResNet-34/ImageNet, the speedup grows to 3.65–4.49× (4097 s vs. 14967–18382 s), because Orion's cost scales super-linearly with depth while ours scales linearly. On ResNet-50/ImageNet, BootNet achieves 10701 s (2 h 58 min) with a 3.02× speedup, demonstrating that the linear scaling trend holds even as bottleneck blocks quadruple the channel count at each layer.

For BootNet-mini with $\log N = 15$ (with $\log QP = 950$), we complete ResNet-18 in 3774 s (1 h 3 min), ResNet-34 in 4949 s (1 h 22 min), and ResNet-50 in 14154 s (3 h 56 min). BootNet-mini's latency increases by 16–32% than BootNet due to the reduced auxiliary prime count ($\#p = 1$ vs. $\#p = 2$), which requires more key-switching rounds per bootstrap. However, BootNet-mini's memory and storage requirements decrease substantially (see Section 7.3).

7.3 Memory and Storage

Orion's resource requirements grow significantly with model size. Table 6 summarizes the precomputation-stage runtime (for generation of the key and weight materials), total cache storage, and peak resident memory (RSS) for both systems. For Orion, the precomputed cache storage increases from 929GB (ResNet-18) to 1,683GB (ResNet-34) and reaches an estimated ≥ 4TB for ResNet-50. This substantial storage demand is primarily driven by the inefficient packing of the large number of weights in the convolutional layers. The total offline pipeline takes 3.8–3.9h for ResNet-18 and 7.0h for ResNet-34. ResNet-50 could not be completed on our 2TB disk: the compilation crashed after 9h, having reached a peak RSS of 642GB while writing 1.78TB of diagonals. Note that such storage and memory usually requires **expensive high-speed memory** such as HBM for practical hardware acceleration (otherwise, frequent I/O becomes the primary bottleneck). Consequently, Orions storage scaling makes it costly for GPU/FPGA/ASIC acceleration.

Table 6: precomputation-stage runtime, total cache storage, and online peak memory for Orion (our reproduction) versus BootNet. Orion's ResNet-50 case crashed during offline compilation, which leads to no data on online peak memory.

Setup	Offline RT (h)	Storage (GB)	Peak Mem (GB)
Orion (our reproduction)			
ResNet-18, SiLU	3.82	929	271
ResNet-18, ReLU	3.90	938	279
ResNet-34, SiLU	7.04	1683	369
ResNet-34, ReLU	7.01	1693	428
ResNet-50, SiLU	≥9.0 (est.)	≥4000 (est.)	–
BootNet			
ResNet-18, $\log N = 16$	1.36	218	227
ResNet-34, $\log N = 16$	1.35	229	267
ResNet-50, $\log N = 16$	3.86	398	349
BootNet-mini			
ResNet-18, $\log N = 15$	0.48	93	121
ResNet-34, $\log N = 15$	0.63	102	139
ResNet-50, $\log N = 15$	3.95	238	242

BootNet use Neujeans[12]'s CinS encoding to absorb slot permutations into the bootstrapping DFT, substantially reducing the storage required for precomputed weight materials. As a result, storage grows modestly from 218GB for ResNet-18 to only 229GB for ResNet-34a mere 5% increase despite twice as many convolutional layers. This is because the dominant costs (Galois keys and average-pooling cache) are shared across layers. For ResNet-50, the storage requirement rises to 398GB, which is only 1.74× that of ResNet-34 despite the 4× channel expansion in bottleneck blocks. Overall, this scaling behavior makes BootNet significantly more amenable to hardware acceleration than prior approaches.

7.4 Accuracy

Plaintext-simulation. Table 7 reports the top-1 accuracy of BootNet on the full ImageNet validation set (50 000 images) under our plaintext simulation: every ReLU in the pretrained ResNet-18/34/50 is replaced by the same composite arcsine polynomial that BootNet evaluates inside RBOOT's $\mathsf{EvalMod}_{\mathrm{RBOOT}}$. Results show that our accuracy drop by about 1% than vanilla ReLU networks. This is possibly introduced by our SAPTQ method and RBOOT's arcsine approximation.

Validation with Encrypted Inference. Since a full validation on 50 000 images using FHE would be rather time consuming, we ran BootNet end-to-end on 100 ImageNet samples under encryption, and compared the per-image output logits against the plaintext simulation results: the maximum absolute deviation is below 2^{-20}, at the significant-digit limit of 32-bit floating-point arithmetic. Since our plaintext and homomorphic implementations perform mathematically equivalent operations, the observed deviations arise solely from two sources: (1) the difference between CKKSs approximate fixed-point arithmetic and floating-point plaintext arithmetic, and (2) the inherent accuracy of CKKS bootstrapping. Both sources are highly accurate, which explains the small 2^{-20}-scale error. Since this deviation is far smaller than the smallest top-1/top-2 logit gap observed across the validation set, the ciphertext run essentially never flips a top-2 decision relative to the plaintext simulation. We conclude that the accuracies in Table 7 can reflect our encrypted pipeline.

Table 7: ImageNet top-1 accuracy (%) of BootNet under plaintext simulation, alongside the floating-point baseline of the unmodified torchvision ResNet.

Model	plaintext baseline	BootNet
ResNet-18	70.55	69.43
ResNet-34	74.29	73.17
ResNet-50	77.03	76.17

Activation function choice. ReLU is the standard activation function in CNNs, yet its nonlinearity is challenging to represent in FHE. Evidence is that Orion's ReLU mode is consistently 21–23% slower, consumes 58–60 GB more peak memory, requires ∼ 2× more bootstraps, and delivers 9–16 fewer bits of precision than its SiLU version.

In contrast, BootNet supports ReLU natively via RBOOT's functional bootstrapping, which fuses the activation directly into the `EvalMod` step. Thus, a head-to-head accuracy comparison with Orion [10] is not directly meaningful: Orion does not report any FHE-side accuracy on full ImageNet, and their cleartext baselines come from a SiLU fine-tuned variant rather than the unmodified ReLU that BootNet evaluates.

8 CONCLUSION

BootNet's central observation is that homomorphic convolution, ReLU approximation, and bootstrapping—historically treated as independent bottlenecks of FHE-based CNN inference—are in fact distinct sub-steps of a single CKKS bootstrap that can be made to coincide; the 2–4× speedup, ∼10× smaller precomputed-cache footprint, and 2–3× lower peak memory all follow primarily from this fusion.

Three conjectures remain open. First, lower storage usage suggests that BootNet is more amenable to GPU/FPGA/ASIC acceleration, and dedicated hardware implementations are possible. Second, BootNet-mini's $N = 2^{15}$ deployment can use sparse-secret encapsulation with ephemeral Hamming weight $\tilde{h} = 32$ to retain the circuit cost of a sparse-key bootstrap while moving the high-modulus keys to a denser main secret, raising the concrete security claim to 120+ bits. Finally, whether the techniques of BootNet generalises to other applications such as transformer remains an open question.

REFERENCES

[1] Moran Baruch, Nir Drucker, Gilad Ezov, Yoav Goldberg, Eyal Kushnir, Jenny Lerner, Omri Soceanu, and Itamar Zimerman. 2023. Training Large Scale Polynomial CNNs for E2E Inference over Homomorphic Encryption. arXiv preprint arXiv:2304.14836. (2023). https://arxiv.org/abs/2304.14836

[2] Moran Baruch, Nir Drucker, Gilad Ezov, Yoav Goldberg, Eyal Kushnir, Jenny Lerner, Omri Soceanu, and Itamar Zimerman. 2024. Polynomial Adaptation of Large-Scale CNNs for Homomorphic Encryption-Based Secure Inference. In *Cyber Security, Cryptology, and Machine Learning (CSCML 2024) (Lecture Notes in Computer Science)*, Vol. 15349. Springer, 3–22. https://doi.org/10.1007/978-3-031-76934-4_1

[3] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-Sparse Keys. In *Advances in Cryptology – EUROCRYPT 2021 (Lecture Notes in Computer Science)*, Vol. 12696. Springer, 587–617. https://doi.org/10.1007/978-3-030-77870-5_21

[4] Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. In *Applied Cryptography and Network Security (ACNS 2022) (Lecture Notes in Computer Science)*, Vol. 13269. Springer, 521–541. https://doi.org/10.1007/978-3-031-09234-3_26

[5] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low Latency Privacy Preserving Inference. In *Proceedings of the 36th International Conference on Machine Learning (ICML 2019) (Proceedings of Machine Learning Research)*, Vol. 97. PMLR, 812–821. https://proceedings.mlr.press/v97/brutzkus19a.html

[6] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2018 (Lecture Notes in Computer Science)*, Vol. 10820. Springer, 360–384. https://doi.org/10.1007/978-3-319-78381-9_14

[7] Jung Hee Cheon, Guillaume Hanrot, Jongmin Kim, and Damien Stehlé. 2025. SHIP: A shallow and highly parallelizable CKKS bootstrapping algorithm. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 398–428.

[8] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017 (Lecture Notes in Computer Science)*, Vol. 10624. Springer, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[9] Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. 2024. DaCapo: Automatic Bootstrapping Management for Efficient Fully Homomorphic Encryption. In *33rd USENIX Security Symposium (USENIX Security 2024)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/cheon

[10] Austin Ebel, Karthik Garimella, and Brandon Reagen. 2025. Orion: A Fully Homomorphic Encryption Framework for Deep Learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2025)*. ACM. https://doi.org/10.1145/3676641.3716246

[11] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016) (Proceedings of Machine Learning Research)*, Vol. 48. PMLR, 201–210. https://proceedings.mlr.press/v48/gilad-bachrach16.html

[12] Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. 2024. NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and FHE Bootstrapping. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS 2024)*. ACM. https://doi.org/10.1145/3658644.3690375

[13] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 2018)*. USENIX Association, 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[14] Miran Kim, Jaehyung Park, Taekyung Kim, Seonghak Kim, and Chohong Min. 2022. HEAR: Human Action Recognition via Neural Networks on Homomorphically Encrypted Data. Cryptology ePrint Archive, Paper 2022/757. (2022). https://eprint.iacr.org/2022/757

[15] Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. 2022. EvalRound Algorithm in CKKS Bootstrapping. In *Advances in Cryptology – ASIACRYPT 2022 (Lecture Notes in Computer Science)*, Vol. 13792. Springer, 161–187. https://doi.org/10.1007/978-3-031-22966-4_6

[16] Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. 2024. A Tensor Compiler with Automatic Data Packing for Simple and Efficient Fully Homomorphic Encryption. *Proceedings of the ACM on Programming Languages* 8, PLDI (2024), 126:1–126:25. https://doi.org/10.1145/3656382

[17] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In *Proceedings of the 39th International Conference on Machine Learning (ICML 2022) (Proceedings of Machine Learning Research)*, Vol. 162. PMLR, 12403–12422. https://proceedings.mlr.press/v162/lee22e.html

[18] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. 2022. Minimax Approximation of Sign Function by Composite Polynomial for Homomorphic Comparison. *IEEE Transactions on Dependable and Secure Computing* 19, 6 (2022), 3711–3727. https://doi.org/10.1109/TDSC.2021.3105111

[19] Junghyun Lee, Eunsang Lee, Young-Sik Kim, Yongwoo Lee, Joon-Woo Lee, Yongjune Kim, and Jong-Seon No. 2023. Optimized Layerwise Approximation for Efficient Private Inference on Fully Homomorphic Encryption. arXiv preprint arXiv:2310.10349. (2023). https://arxiv.org/abs/2310.10349

[20] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. 2023. Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data. *IEEE Access* 11 (2023), 62062–62076. https://doi.org/10.1109/ACCESS.2023.3287564

[21] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054. https://doi.org/10.1109/ACCESS.2022.3159694 Preprint: arXiv:2106.07229.

[22] Yuxiao Lu, Jie Lin, Chao Jin, Zhe Wang, Min Wu, Khin Mi Mi Aung, and Xiaoli Li. 2021. FFConv: Fast Factorized Convolutional Neural Network Inference on Encrypted Data. arXiv preprint arXiv:2102.03494. (2021). https://arxiv.org/abs/2102.03494

[23] Byeong-Seo Min and Joon-Woo Lee. 2024. Low-Latency Linear Transformations with Small Key Transmission for Private Neural Network on Homomorphic Encryption. Cryptology ePrint Archive, Paper 2024/883. (2024). https://eprint.iacr.org/2024/883

[24] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 2020)*. USENIX Association, 2505–2522. https://www.usenix.org/conference/usenixsecurity20/presentation/mishra

[25] Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2025. THOR: Secure Transformer Inference with Homomorphic Encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS 2025)*. ACM. https://doi.org/10.1145/3719027.3765150 Cryptology ePrint Archive, Paper 2024/1881.

[26] Christian Mouchet, Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2024. Lattigo: A Multiparty Homomorphic Encryption Library in Go. https://github.com/tuneinsight/lattigo. (2024).

[27] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. 2022. AESPA: Accuracy preserving low-degree polynomial activation for fast private inference. *arXiv preprint arXiv:2201.06699* (2022).

[28] Jaiyoung Park, Sejin Park, Jai Hyun Park, Jung Ho Ahn, Jung Hee Cheon, Guillaume Hanrot, Jung Woo Kim, Minje Park, and Damien Stehlé. 2026. Scaling up Privacy-Preserving ML: A CKKS Implementation of Llama-2-7B. arXiv preprint arXiv:2601.18511. (2026). https://arxiv.org/abs/2601.18511

[29] Hyewon Sung, Sieun Seo, Taekyung Kim, and Chohong Min. 2025. EvalRound+ Bootstrapping and Its Rigorous Analysis for CKKS Scheme. *IEEE Access* 13 (2025), 140847–140866. https://doi.org/10.1109/ACCESS.2025.3595177

[30] Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi. 2023. HECO: Fully Homomorphic Encryption Compiler. In *32nd USENIX Security Symposium (USENIX Security 2023)*. USENIX Association, 4715–4732. https://www.usenix.org/conference/usenixsecurity23/presentation/viand

[31] Zhaomin Yang, Chao Niu, Benqiang Wei, Zhicong Huang, Cheng Hong, and Tao Wei. 2025. RBOOT: Accelerating Homomorphic Neural Network Inference by Fusing ReLU within Bootstrapping. Cryptology ePrint Archive, Paper 2025/1534. (2025). https://eprint.iacr.org/2025/1534

[32] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2024. Secure transformer inference made non-interactive. *Cryptology ePrint Archive* (2024).

A SCALE-ALIGNED POST-TRAINING QUANTIZATION: FULL DERIVATION

This appendix gives the full derivation of Scale-Aligned Post-Training Quantization (SAPTQ), the model preparation procedure summarized in Section 5.2. We first recap the activation range problem (Appendix A.1), then present the SAPTQ pipeline and its accuracy impact (Appendix A.2).

A.1 The Activation Range Problem

As discussed in Section 2, CKKS-based homomorphic ReLU evaluation relies on polynomial approximation—typically a minimax composite polynomial [18]—that is fitted over a bounded interval $[-B, B]$ for a scheme-specific $B > 0$. If any plaintext slot value v satisfies $|v| > B$ at evaluation time, the polynomial diverges rapidly and the layer output becomes meaningless. Formally, let $p(x)$ be the degree-d composite polynomial approximating $\mathrm{ReLU}(x)$ on $[-B, B]$; then for $|v| > B$ the approximation error $|p(v) - \mathrm{ReLU}(v)|$ grows as $O\big((v/B)^d\big)$, rendering subsequent layers unusable.

Hence, *every* activation tensor entering a homomorphic ReLU must satisfy $\|\mathbf{a}\|_\infty \le B$. Guaranteeing this property across all layers and all possible inputs is non-trivial: standard CNN training imposes no such constraint, and the dynamic range of intermediate features varies widely across layers, architectures, and input distributions.

A.2 Scale-Aligned Post-Training Quantization (SAPTQ)

Background: symmetric int8 post-training quantization. A symmetric int8 quantize/dequantize (Q/DQ) pair applied to a real-valued tensor v computes

$$\hat{v} \;=\; s \cdot \mathrm{clamp}\big(\lfloor v/s \rceil,\, -128,\, 127\big), \tag{2}$$

where $s > 0$ is a per-tensor scale factor; the result $\hat{v}$ lies in $[-128\,s,\ 127\,s]$ and is rounded to a multiple of s. In *post-training quantization* (PTQ), the scale s is chosen by a short *calibration* pass: the pretrained model is run in forward mode over a small held-out input subset, a histogram of each activation's values is collected, and s is set so that a high-confidence range of the histogram maps onto the int8 grid (we use KL-divergence-minimizing histogram clipping, a standard PTQ heuristic). No weight update is performed at any point; the pretrained parameters remain exactly as shipped.

The SAPTQ observation. Standard int8 PTQ produces per-activation scales $\{s_\ell\}$ that depend entirely on the pretrained model's activation statistics and therefore have no a priori relationship to the CKKS ReLU domain $[-B,\ B]$. The SAPTQ observation is that a symmetric int8 PTQ graph admits two classes of *value-preserving* algebraic rewrites that, together, align every pre-ReLU tensor with $[-B,\ B]$ at no additional accuracy cost beyond the initial calibration:

(i) **Scale absorption.** Because ReLU is positively homogeneous ($\mathrm{ReLU}(\alpha x) = \alpha\,\mathrm{ReLU}(x)$ for $\alpha > 0$), a scalar multiplication by a positive constant s_ℓ can be commuted past a subsequent ReLU. Combined with the linearity of the following convolution, this lets each calibrated scale s_ℓ be folded into the next convolution's weight and bias, so that no runtime scalar multiplication survives on the hot path.

(ii) **Uniform rescaling.** After scale absorption, every pre-ReLU tensor has some effective int8 scale s'_ℓ. Multiplying all s'_ℓ by a single common factor c—and applying $1/c$ at the network's final output—is a network-wide value-preserving rewrite. Choosing c so that the resulting uniform pre-ReLU scale equals $B/128$ sends every pre-ReLU tensor to $[-128\,(B/128),\ 127\,(B/128)] = [-B,\ 127\,B/128] \subset [-B,\ B]$, exactly the homomorphic ReLU domain.

The SAPTQ pipeline. Our model preparation procedure consists of five steps.

1. **Graph capture.** Export the pretrained model to a graph-level intermediate representation (e.g., ONNX) whose operators can be matched and rewritten.
2. **Q/DQ insertion and calibration.** Insert a symmetric Q/DQ pair immediately before every ReLU. Run a forward-only calibration pass on the held-out ImageNet validation set (50 000 images), selecting each pre-ReLU scale s_ℓ by KL-divergence-minimizing histogram clipping. No weight update is performed.
3. **Q/DQ expansion.** Replace each Q/DQ pair with its algebraic expansion, $\mathrm{mul}(s) \circ \mathrm{clamp}(-128,\ 127) \circ \mathrm{round} \circ \mathrm{div}(s)$. This is a semantic no-op—the numerical behavior is identical to the original Q/DQ node—but the multiplication and division are now explicit graph edges available to rewrite.
4. **Scale absorption.** At each pre-ReLU site, commute the $\mathrm{mul}(s_\ell)$ past the ReLU via positive homogeneity, then fold the (now post-ReLU) multiplication and the $1/s_\ell$ expected by the following convolution into that convolution's weight and bias. After this step, no runtime scalar multiplication or division remains on the hot path.
5. **Uniform rescaling.** Multiply every surviving per-activation scale by the common factor that brings the effective pre-ReLU scale to $B/128$, and apply the compensating inverse at the final layer. This step is free at inference time (the multiplier is absorbed into the same convolution weights by the rewrite of step 4), but it is what aligns every pre-ReLU tensor with $[-B,\ B]$.

Why SAPTQ works. Accuracy loss in SAPTQ comes exclusively from step 2: the int8 calibration replaces each pre-ReLU tensor by its nearest element on a 256-bin grid, introducing a small per-activation rounding error that propagates through the rest of the network. Steps 3–5 are exact algebraic identities—Q/DQ expansion, commutation with ReLU, and absorption into convolution parameters—and therefore contribute zero additional error. Because the pipeline uses only off-the-shelf PTQ calibration plus four graph-rewrite passes, it requires no custom loss term, no training-time infrastructure, and no homomorphic-side support at inference time.

Accuracy impact. Table 8 reports the top-1 accuracy on ImageNet before and after SAPTQ adaptation for three representative architectures.

The accuracy loss is consistently below 0.2 percentage points across all three models. Because the output of SAPTQ is a plaintext network whose pre-ReLU activations are natively bounded to $[-B,\ B]$, it plugs directly into the BootNet pipeline described in Section 3 without requiring any additional homomorphic operations at inference time.

Table 8: Top-1 accuracy (%) on ImageNet before and after SAPTQ adaptation. Δ denotes the accuracy change relative to the float baseline.

Model	Float baseline	After SAPTQ	Δ
ResNet-18	70.55	70.44	−0.11
ResNet-34	74.29	74.11	−0.18
ResNet-50	77.03	76.85	−0.18

B INTERPRETABILITY OF THE RBOOT-INDUCED ACCURACY DROP

The RBOOT polynomial of Section 5.1 approximates ReLU with a small but non-zero per-activation error that propagates through every ReLU site of the network and emerges as a measurable top-1 accuracy drop on ImageNet. This appendix reports a mechanistic

analysis of *why* that drop is what it is: it is a strictly monotone, super-linear function of the polynomial minimax bound, dominated by the cancellation of lost and gained predictions at the softmax decision boundary, and bounded by an architecture-dependent geometric per-layer growth rate.

B.1 Setup

We instrument a plaintext PyTorch ResNet-18/34/50 with a ReLU-replacement hook that swaps every ReLU for the same composite arcsine polynomial that BootNet evaluates inside RBOOT's EvalMod$_{\text{RBOOT}}$ (Equation (1)). We report two configurations. The *d15* configuration matches the deployed BootNet polynomial ($P_{\sin}$ and $P_{\cos}$ both degree-15 Chebyshev minimax) and has a measured composite approx-ReLU error of $\epsilon_{15} \approx 3.73 \times 10^{-3}$ on $[-1, 1]$. The *d31* configuration swaps $P_{\cos}$ for its degree-31 minimax fit, halving the bound to $\epsilon_{31} \approx 1.83 \times 10^{-3}$ (ratio $\epsilon_{31}/\epsilon_{15} \approx 0.49$). The d31 variant is reference-only: RBOOT caps the EvalMod polynomial degree, so it is not the deployed configuration; it provides a controlled $\sim 2\times$ reduction of ϵ that lets us isolate the exponent governing the bound-to-drop relationship. For each (model, configuration) pair we run the full ImageNet validation set (50 000 images) and compare per-image logits against the deterministic reference (no polynomial replacement).

B.2 Per-Layer Error Growth Is Geometric and Bounded

We first measure how the per-position L_∞ error $\|\mathbf{a}_\ell^{\text{poly}} - \mathbf{a}_\ell^{\text{ref}}\|_\infty$ evolves with layer index ℓ. Across all three networks the growth is well-fitted by a single geometric rate ρ:

$$\|\Delta_\ell\|_\infty \approx \rho^\ell \, \epsilon, \qquad \rho \approx \begin{cases} 1.11, & \text{ResNet-18,} \\ 1.04, & \text{ResNet-34,} \\ 1.03, & \text{ResNet-50.} \end{cases} \tag{3}$$

Two observations follow. First, ρ is determined by the network architecture, not by the polynomial degree: d15 and d31 produce indistinguishable values of ρ at each depth, with only the prefactor ϵ differing. Second, ρ *decreases* with network depth. This is mechanically the result of skip-path averaging: at each residual addition, an approximation-affected branch is summed with a clean shortcut, and as the proportion of skip arrivals grows with depth the per-layer propagation factor is driven toward unity. Equation Equation (3) therefore predicts a structurally smaller per-layer drift in deeper bottleneck networks, even at the same polynomial bound.

B.3 Top-1 Accuracy Drop Scales as Low-Degree Super-Linearity

Table 9 reports the measured top-1 drop on ImageNet val for the d15 and d31 variants, and the empirical exponent η in $\Delta_{\text{top1}} \propto \epsilon^\eta$ obtained as $\log(\Delta_{31}/\Delta_{15})/\log(\epsilon_{31}/\epsilon_{15})$.

The exponent is consistently $\eta \in [1.35, 1.78]$ with mean ≈ 1.5, well above the 0.5 predicted by an i.i.d. central-limit-theorem argument and above the 1 predicted by naive linearization in ϵ. Quantitatively, halving ϵ from d15 to d31 (ratio 0.49) cuts the average drop by a factor of 0.33, matching $0.49^{1.5} \approx 0.34$ to within rounding.

Table 9: Top-1 accuracy drop (Δ, percentage points) on ImageNet val, for the deployed d15 polynomial and the d31 reference variant. The empirical exponent η is computed against the controlled bound ratio $\epsilon_{31}/\epsilon_{15} \approx 0.49$.

Model	Δ_{15} (pp)	Δ_{31} (pp)	η
ResNet-18	-1.12	-0.31	1.78
ResNet-34	-1.12	-0.43	1.35
ResNet-50	-0.86	-0.29	1.51
Mean	-1.03	-0.34	≈ 1.5

B.4 Where the Super-Linearity Comes From

We decompose Δ_{top1} as a four-link chain

$$\epsilon \xrightarrow{\text{A}} \sigma_\Delta \xrightarrow{\text{B}} \Pr[\text{flip}] \xrightarrow{\text{C}} \Pr[\text{lost}] \xrightarrow{\text{D}} \Delta_{\text{top1}},$$

where σ_Δ is the standard deviation of the top-1/top-2 logit-margin perturbation $\Delta_{\text{diff}} = \Delta_{\text{top1}}^{\text{logit}} - \Delta_{\text{top2}}^{\text{logit}}$, $\Pr[\text{flip}]$ is the rate at which the predicted class changes between the reference and approximated forward passes, $\Pr[\text{lost}]$ is the rate at which a *correct* prediction becomes wrong, and the measured drop equals $\Pr[\text{lost}] - \Pr[\text{gained}]$.

Link A ($\epsilon \to \sigma_\Delta$): linear. σ_Δ scales linearly in ϵ, with empirical ratio $\sigma_\Delta^{(31)}/\sigma_\Delta^{(15)} \approx 0.45$ versus the bound ratio 0.49. The top-1 and top-2 logit perturbations are mildly *anti-correlated* (a polynomial-induced bump in one competing class typically depresses the other), so $\sigma_{\Delta_{\text{diff}}}/\sigma_\Delta$ exceeds the $\sqrt{2}$ value implied by independence.

Links B/C ($\sigma_\Delta \to \Pr[\text{flip}], \Pr[\text{lost}]$): near-linear. A Gaussian decision-boundary model $\frac{1}{2}\operatorname{erfc}\big(G/(\sigma_\Delta\sqrt{2})\big)$, with G the per-image baseline logit gap, predicts $\Pr[\text{flip}]$ within 0–3% at d31 and underestimates by 12–14% at d15—the residual gap reflects heavier tails of Δ_{diff} at larger σ_Δ. Flips concentrate almost entirely in the lowest-margin quartile (Q_0, 11–33% of all flips); the top-margin quartile contributes $< 0.01\%$.

Link D ($\Pr[\text{lost}] \to \Delta_{\text{top1}}$): super-linear—this is where the $\epsilon^{1.5}$ arises. Between d15 and d31, $\Pr[\text{lost}]$ shrinks as $\propto \epsilon^{1.16}$, but $\Pr[\text{gained}]$ shrinks *more slowly*, as $\propto \epsilon^{1.04}$. Their difference $\Pr[\text{lost}] - \Pr[\text{gained}]$ therefore shrinks as $\propto \epsilon^{1.51}$: as ϵ tightens, the dominant lost–gained cancellation near the decision boundary tightens super-linearly, and Link D injects the additional half-power that distinguishes the empirical $\epsilon^{1.5}$ from the ϵ^1 predicted by Links A–C alone. The four-link reconstruction recovers the measured top-1 drop to within 0.01 pp on every (depth, polynomial) combination, confirming that this decomposition captures the entire effect.

B.5 Implications

The analysis supports three statements that we use in the body text when reporting accuracy. First, the RBOOT-induced accuracy drop is *bounded and predictable*: it is a strictly monotone, super-linear function of the polynomial minimax bound, well-modelled as $\Delta_{\text{top1}} \propto \epsilon^{1.5}$ with an architecture-dependent prefactor that follows the per-layer geometric growth ρ of Equation (3). Second, the super-linearity is mechanistically attributable to lost–gained cancellation at the softmax decision boundary (Appendix B.4), not to any pathology in RBOOT's polynomial fitting; tightening ϵ converts to drop reduction at a known 1.5-power rate. Third, deeper bottleneck networks (e.g., ResNet-50) inherit a structurally smaller

drop than shallow basic-block networks under the same polynomial because skip-path averaging suppresses ρ. At the BootNet operating point ($\epsilon_{15} \approx 3.73 \times 10^{-3}$ for the deployed d15 polynomial), the ~ 1 pp top-1 drop on ResNet-18/34 and the ~ 0.9 pp drop on ResNet-50 are the analytic floors implied by the polynomial bound and ImageNet's softmax-margin geometry, attainable without any additional homomorphic operations at inference time.

C EVALROUND++: SLIM-BOOTSTRAP EVALROUND+

EvalRound [15] and its refinement EvalRound+ [29] reduce the modulus consumed by the CtS/StC pair of CKKS bootstrapping by replacing the standard EvalMod polynomial with a complementary *rounding* polynomial, then cancelling the introduced rounding error against a parallel high-precision *skip connection*. We refer to BootNet's slim-bootstrap port of EvalRound+ as *EvalRound++*. This appendix derives EvalRound++ assuming a standard slim bootstrap, then describes the further adaptation needed to compose it with RBOOT's functional EvalMod.

Background. After ModRaise lifts the modulus, the encrypted plaintext becomes $\mathrm{pt}+q_0 I$ for an integer polynomial I. The conventional bootstrap recovers pt by applying a polynomial that approximates $x \bmod q_0$ inside EvalMod. EvalRound observes that the *complementary* approximation $x \mapsto x-(x \bmod q_0) = q_0 I$ achieves the same recovery via a slot-side subtraction $(\mathrm{pt}+q_0 I)-q_0 I = \mathrm{pt}$. The two approximating polynomials have the same depth, so the immediate cost is unchanged; the saving comes from a different mechanism. When CtS is evaluated at a smaller scale $\Delta_{\mathrm{CtS}^*} \ll \Delta_{\mathrm{CtS}}$, its rounding error e_{CtS^*} is no longer negligible—but re-encoding the post-CtS ciphertext through the rounding polynomial leaves this error as a constant offset that *cancels* in the subsequent slot-side subtraction. Δ_{CtS^*} may therefore shrink, saving roughly 1–2 multiplicative levels in the CtS chain. EvalRound+ refines the analysis with a rigorous error-bound threshold for Δ_{CtS^*} and applies an analogous rebalancing to StC.

What changes under the slim schedule. The slim CKKS bootstrap reorders the standard pipeline by hoisting StC ahead of ModRaise (Section 3); the bootstrap tail is therefore ScaleDown $\rightarrow$ ModRaise $\rightarrow$ CtS $\rightarrow$ EvalMod, ending in slot domain. Two consequences for EvalRound+ apply. First, the EvalRound subtraction operates entirely in the slot domain—StC is no longer between the two skip-connection paths—so the slot-side scale-back and subtraction can be expressed as a single linear circuit. Second, the EvalRound+ refinement that targets Δ_{StC^*} applies to the *next* slim epoch's StC (which is fused with the next StC-following computation) rather than to the current tail. Within a single bootstrap, the slim port of EvalRound+ realises the EvalRound saving plus the EvalRound+ scale-threshold tightening; the EvalRound+ StC-budget gain propagates forward to the next epoch.

Construction. EvalRound++ is given as Algorithm 5 and the corresponding control-flow as Figure 2. After ModRaise (line 1) we retain a snapshot $\mathrm{ct}_{\mathrm{in}}$ of the post-lift ciphertext. The main path applies a low-scale CtS * (line 2) producing $\mathrm{ct}_{\mathrm{main}}$ in the slot domain at depth L_{CtS^*} shallower than the conventional CtS. EvalMod runs on $\mathrm{ct}_{\mathrm{main}}$ (line 3). A scale-back step (line 4) multiplies $\mathrm{ct}_{\mathrm{main}}$ by a single alignment scalar α, chosen so that the rescaled ciphertext matches EvalMod's output scale; the slot-side subtraction at line 5 then cancels e_{CtS^*}, leaving only the EvalMod approximation error e_{EM}. In parallel, a high-precision *bypass* $\mathrm{CtS}_{\mathrm{HP}}$ on the saved $\mathrm{ct}_{\mathrm{in}}$ (line 6) reuses the levels otherwise reserved for EvalMod's output and recovers the message component pt/q_0 with negligible CtS error. The final addition at line 7 reattaches this component, producing the refreshed ciphertext.

Algorithm 5 EvalRound++. Input is the coefficient-domain ciphertext at level 0 produced by the upstream StC/ScaleDown pair; output is a refreshed slot-domain ciphertext.

Input: Coefficient-domain ciphertext ct at level 0; EvalRound++ parameters $(\Delta_{\mathrm{CtS}^*}, \alpha)$, where $\Delta_{\mathrm{CtS}^*} \ll \Delta_{\mathrm{CtS}}$ is the EvalRound+ rigorous-threshold low scale for the main-path CtS, and α is the alignment scalar that rescales the saved low-scale CtS output to match EvalMod's output scale.

Output: ct' in slot domain, encoding pt/q_0 with bounded EvalMod residual e_{EM}.

1: $\mathrm{ct}_{\mathrm{in}} \leftarrow \mathsf{ModRaise}(\mathrm{ct})$ ▷ lift modulus; save snapshot
2: $\mathrm{ct}_{\mathrm{main}} \leftarrow \mathsf{CtS}^*(\mathrm{ct}_{\mathrm{in}};\ \Delta_{\mathrm{CtS}^*})$ ▷ low-scale main CtS
3: $\mathrm{ct}_{\mathrm{em}} \leftarrow \mathsf{EvalMod}(\mathrm{ct}_{\mathrm{main}})$ ▷ $\approx (\mathrm{pt}/q_0 + e_{\mathrm{CtS}^*} + e_{\mathrm{EM}})^*$
4: $\mathrm{ct}_{\mathrm{sb}} \leftarrow \mathsf{Rescale}(\alpha \cdot \mathrm{ct}_{\mathrm{main}})$ ▷ align scales
5: $\mathrm{ct}_{\Delta} \leftarrow \mathsf{Sub}(\mathrm{ct}_{\mathrm{em}}, \mathrm{ct}_{\mathrm{sb}})$ ▷ cancels e_{CtS^*}
6: $\mathrm{ct}_{\mathrm{bp}} \leftarrow \mathsf{CtS}_{\mathrm{HP}}(\mathrm{ct}_{\mathrm{in}})$ ▷ bypass; reuses EvalMod-side levels
7: $\mathrm{ct}' \leftarrow \mathsf{Add}(\mathrm{ct}_{\Delta}, \mathrm{ct}_{\mathrm{bp}})$ ▷ recombine

Modulus-budget savings. EvalRound++ inherits the CtS modulus reduction of EvalRound+ [29]: every bootstrap pays a strictly smaller per-call CtS budget at no precision cost. StC, the residual scaling primes, and q_0 are unchanged in the current tail; the EvalRound+ StC-budget gain accrues in the next epoch. Average bootstrap precision is preserved at the same depth, since the rounding polynomial's error envelope is tighter than the modular-reduction polynomial's around the integer grid.

Composing EvalRound++ with RBOOT. Substituting RBOOT's functional $\mathrm{EvalMod}_{\mathrm{RBOOT}}$ (Section 5.1) for the standard EvalMod on line 3 of Algorithm 5 and replacing the slot-side reconstruction with a single branch subtraction $\mathrm{ct}_{\mathrm{bp}} - \mathrm{ct}_{\mathrm{em}}$ does not, on its own, compute $\mathrm{ReLU}(\mathrm{pt})$: the bypass branch carries pt at high precision while the main branch carries $\mathrm{ReLU}(\mathrm{pt})$, and the difference $\mathrm{pt} - \mathrm{ReLU}(\mathrm{pt}) = \min(\mathrm{pt}, 0) = -\mathrm{ReLU}(-\mathrm{pt})$ is the negative-part complement of ReLU rather than ReLU itself. The fix is to make the main branch compute $-\mathrm{ReLU}(-x) = \min(x, 0)$ instead of $\mathrm{ReLU}(x)$: because RBOOT realizes $\mathrm{ReLU}(x) \approx \frac{1}{2}x + \frac{1}{2}|x|$ as the sum of a "sine branch" ($\frac{1}{2}x$) and a "cosine branch" ($\frac{1}{2}|x|$) (Section 5.1), multiplying the cosine branch's polynomial coefficients by -1 flips its sign in place, yielding $\frac{1}{2}x - \frac{1}{2}|x| = \min(x, 0)$. This is a one-time scalar tweak at parameter setup, costs no additional multiplicative level, and—together with the $\mathrm{ct}_{\mathrm{bp}} - \mathrm{ct}_{\mathrm{em}}$ recombination—fuses the EvalRound++ modulus saving with RBOOT's ReLU evaluation in a single bootstrap tail.

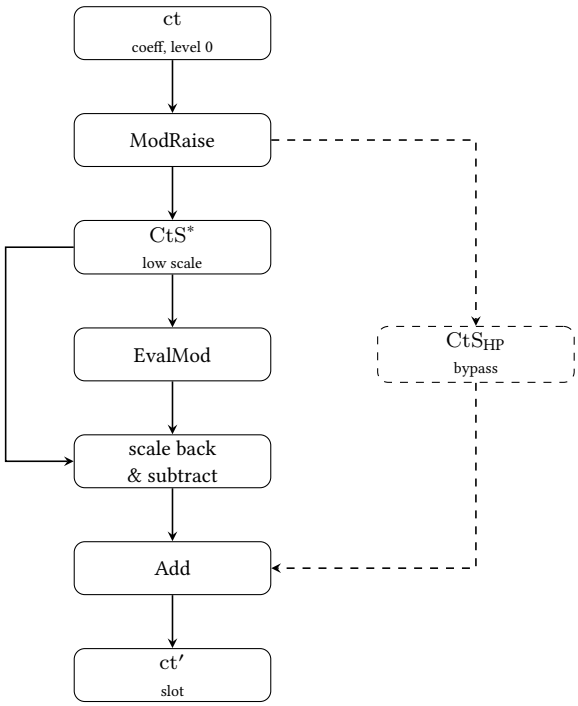


Figure 2: Control-flow of EvalRound++. The main path (CtS^* at low scale → EvalMod → scale-back-and-subtract → Add) runs the rounding-polynomial circuit; the bypass on the right (dashed: $\mathrm{CtS}_{\mathrm{HP}}$ on the saved post-ModRaise snapshot) reuses EvalMod-side levels to recover the message component and joins at the final Add. The short left-side loop carries the saved low-scale CtS^* output past EvalMod into the cancellation step. The canonical (non-EvalRound+) slim tail is the main vertical chain with the bypass and the cancellation edge removed.

Table 10: Hardware calibration: ResNet-18 / Tiny ImageNet ($\log N = 16$). Time in seconds; precision in bits; #Boot counts bootstrap operations; RSS is peak resident set size. "Paper" denotes times reported in [10]; "Ours" denotes our reproduction.

Activation	Time	Prec.	#Boot	RSS (GB)
Orion (paper)				
SiLU	1447	—	—	—
Orion (our reproduction)				
SiLU	2831	21.53	67	218
ReLU	4399	12.70	129	253

D HARDWARE CALIBRATION VIA TINY IMAGENET

To further verify that our server is not faster than the hardware used in the Orion paper [10], we reproduced their ResNet-18 / Tiny ImageNet (64×64, 200 classes) experiment using the default Orion configuration ($\log N = 16$). Table 10 compares our reproduced times against the paper's reported values.

Our server takes 2831 s for the SiLU variant, roughly 2.0× slower than the 1447 s reported in the Orion paper. This confirms that all speedup comparisons in Section 7 are conservative: our reproduced Orion baselines are measured on the *same* (or slower) hardware as our own system, so the reported speedups are not inflated by hardware differences.

Table 11: Odd-index Chebyshev coefficients of the two arcsine polynomials; even-index coefficients are zero up to multiprecision arithmetic noise. Six significant digits.

i	$P_{\cos}[T_i]$	$P_{\sin}[T_i]$
1	2.02725×10^{-1}	2.00772×10^{-1}
3	2.26008×10^{-2}	2.07007×10^{-2}
5	8.19448×10^{-3}	6.39663×10^{-3}
7	4.23067×10^{-3}	2.57277×10^{-3}
9	2.60651×10^{-3}	1.11228×10^{-3}
11	1.79389×10^{-3}	4.71233×10^{-4}
13	1.34005×10^{-3}	1.80533×10^{-4}
15	3.54730×10^{-3}	5.71335×10^{-5}

Table 12: Odd-index Chebyshev coefficients of the reference degree-31 minimax fit of $P_{\cos}$ on $[-1, 1]$, arranged in two row-pairs. Six significant digits.

i	$P_{\cos}^{(\mathrm{d31})}[T_i]$	i	$P_{\cos}^{(\mathrm{d31})}[T_i]$
1	2.02662×10^{-1}	17	7.24900×10^{-4}
3	2.25355×10^{-2}	19	5.86323×10^{-4}
5	8.12553×10^{-3}	21	4.86037×10^{-4}
7	4.15571×10^{-3}	23	4.11477×10^{-4}
9	2.52233×10^{-3}	25	3.54938×10^{-4}
11	1.69585×10^{-3}	27	3.11518×10^{-4}
13	1.22088×10^{-3}	29	2.78029×10^{-4}
15	9.23303×10^{-4}	31	1.56877×10^{-3}

The Tiny ImageNet results also exhibit the same ReLU–SiLU trends observed on ImageNet: ReLU is 55% slower, achieves 8.8 fewer bits of precision, and requires 93% more bootstrap operations than SiLU.

E COEFFICIENTS OF THE ARCSINE APPROXIMATION POLYNOMIALS

Table 11 lists the coefficients of the two degree-15 Chebyshev polynomials $P_{\cos}, P_{\sin}$ used by the functional ReLU of Section 5.1; we reproduce them here as a standalone artifact. Both polynomials approximate $\arcsin(y)/(2\pi)$ in the Chebyshev basis $\{T_0, \ldots, T_{15}\}$ where T_i is the degree-i Chebyshev polynomial of the first kind: $P_{\cos}$ is fit on the full interval $y \in [-1, 1]$; $P_{\sin}$ is fit on the subinterval reached when the pre-activation satisfies $|x|/q_0 \le 0.2$. Because $\arcsin(y)$ is odd the even-index coefficients vanish analytically—the Remez fits return them at the multiprecision noise floor ($\sim 10^{-46}$ for $P_{\cos}$, $\sim 10^{-78}$ for $P_{\sin}$) and we omit them. In deployment, every entry of both columns is multiplied by 1.25 once at parameter setup to fold in the recovery rescaling described in Section 5.1; the values below are the *bare* polynomials before this fold.

Reference degree-31 cosine-branch coefficients. Appendix B additionally evaluates a *degree*-31 reference fit of the cosine-branch arcsine polynomial on $y \in [-1, 1]$, used only for the bound-vs-drop interpretability analysis (it is not deployed in BootNet, which caps the EvalMod polynomial degree at 15). The 16 odd-index Chebyshev coefficients of this reference fit are listed in Table 12; the resulting uniform maximum error on $y \in [-1, 1]$ is $\approx 2^{-9.44} \approx 1.44 \times 10^{-3}$, roughly half the d15 cosine-branch bound of $2^{-7.399}$ cited in Section 5.1.