# RBOOT: Accelerating Homomorphic Neural Network Inference by Fusing ReLU within Bootstrapping

Zhaomin Yang$^{[0009-0002-5907-1447]}$, Chao Niu$^{[0000-0002-6672-7515]}$, Benqiang Wei$^{[0009-0001-0653-2070]}$, Zhicong Huang$^{[0000-0003-1338-611X]}$, Cheng Hong$^{\star[0009-0008-0477-0359]}$, and Tao Wei$^{[0000-0001-9537-7051]}$

Ant Group, China
{`yangzhaomin.yzm,niuchao.niu,weibenqiang.wbq,`
`zhicong.hzc,vince.hc,lenx.wei`}`@antgroup.com`

**Abstract.** A major bottleneck in secure neural network inference using Fully Homomorphic Encryption (FHE) is the evaluation of non-linear activation functions like ReLU, which are inefficient to compute under FHE. State-of-the-art solutions approximate ReLU using high-degree polynomials, incurring significant computational overhead. We propose novel methods for *functional bootstrapping with CKKS*, and based on these methods we present RBOOT, an optimized framework that seamlessly integrates ReLU evaluation into CKKS bootstrapping, significantly reducing multiplication depth and boosting efficiency. Our key insight is that the EvalMod step in CKKS bootstrapping is composed of trigonometric functions, which can be transformed into various common non-linear functions. By co-optimizing these components, we can exploit such non-linearity to construct ReLU (and other non-linear functions) within the bootstrapping process itself, greatly reducing the computation overhead. Results on four widely used CNN models show that RBOOT achieves 2.77× faster end-to-end inference and 81% lower memory usage compared to previous polynomial approximation works, while maintaining comparable accuracy.

## 1 Introduction

Cloud-based machine learning services like face recognition [14,68], fraud detection [12], and healthcare analysis systems [37] require clients to upload raw data to remote servers. These servers process the data through some machine learning models (typically neural networks) and return the results to the clients. As a result, such services inherently pose privacy risks if the clients' data contains sensitive information. Fully Homomorphic Encryption (FHE) allows computation on encrypted data, and is a promising solution for privacy-preserving applications. A substantial line of research has been devoted to enabling homomorphic neural network inference on encrypted inputs [36,10,44,56].

$^\star$ Corresponding author: Cheng Hong (vince.hc@antgroup.com)

Modern FHE schemes fall into two categories: Single-Instruction-Multiple-Data (SIMD)-compatible schemes such as BFV [27], BGV [11], and CKKS [20]; and SIMD-incompatible[1] schemes such as TFHE [24]. The former are optimized for batch processing of vectorized data, making them well-suited for linear layers in neural networks. However, their functionality is limited to addition and multiplication operations, which cannot directly represent nonlinear functions like ReLU. In contrast, TFHE-like schemes support arbitrary functions through the mechanism of functional bootstrapping, at the cost of lacking SIMD abilities. To implement the nonlinear layers in neural networks, several kinds of approaches could be considered:

- Polynomial approximation (Most widely-used). Studies [45,44] show that carefully composed high-degree polynomials can effectively replace nonlinear functions in neural networks with minimal accuracy degradation. While this approach is effective and has become prevalent in related works [58,38,23,26], it consumes a substantial amount of multiplicative depth (up to 20), introducing significant overhead in FHE computations.
- Scheme switching. Some studies [49] use SIMD-compatible FHEs for linear layers, and use scheme switching [9,52] to switch to TFHE-like schemes for nonlinear layers. However, this method faces scalability challenges. For example, tens of millions of ReLUs are required in the ResNet50 model. Such bulk nonlinear computations are impractical for SIMD-incompatible schemes.
- Functional bootstrapping for SIMD-compatible FHEs. [51,2] introduce the concept of functional bootstrapping within the BFV and CKKS schemes, and have the potential for realizing nonlinear functions in a SIMD way. However, their approaches only support look-up-table style functions, whose inputs and outputs have a small bit-width. As a result, the precision of such functions are limited.

The unsatisfactory status quo raises our question: *Can we significantly reduce computational overhead while preserving numerical precision for non-linear evaluation in encrypted neural networks?*

In this work, we affirmatively answer this question by introducing RBOOT, a cryptographically optimized fusion of ReLU evaluation and CKKS bootstrapping that achieves both depth reduction and precision preservation. By decomposing the target nonlinear function and composing it with the sine and arcsin functions in CKKS bootstrapping, our 'two-in-one' approach reduces the function's multiplication depth consumption by more than 40% (from 14$\sim$20 to 8 levels per ReLU) while maintaining same bit precision. While we specifically focus on ReLU, our approaches pave the way for general CKKS functional bootstrapping supporting continuous real-valued functions, unlike prior method [2] that is limited to discrete functions for small integers.

We implement a homomorphic neural network inference framework based on RBOOT and compare it with the influential work of Lee et al. [44] across

[1] [54,31,47,48] propose SIMD methods for TFHE-like schemes, but they either lack implementations or fail to be substantially faster than standard TFHE.

several CNN models on the CIFAR datasets. Our results demonstrate more than $2.7\times$ acceleration in end-to-end inference runtime with more than 80% reduction in memory usage. The inferences show no essential (less than 0.1% on top-1) accuracy loss compared to the inference in plain with standard ReLU.

As a side contribution, we developed an optimized version of the Microsoft SEAL library that supports multiple special primes for more efficient key switching [32], whereas the original SEAL implementation only accommodates a single special prime. This enhancement may be of broader interest to the FHE community.

## 1.1 Technical Overview

Our observation is first inspired by the migration of workflows of the CKKS bootstrapping algorithm. With the early workflow $\mathsf{ModRaise} \to \mathsf{CoeffsToSlots} \to \mathsf{EvalMod} \to \mathsf{SlotsToCoeffs}$, there is no feasible way to do any meaningful evaluation on the original message slots during the process, since the evaluation is on the original plaintext polynomial coefficients along with extra overflows of modulus $q_0$. On the other hand, it is becoming common to use a newer type of workflow for CKKS bootstrapping $\mathsf{SlotsToCoeffs} \to \mathsf{ModRaise} \to \mathsf{CoeffsToSlots} \to \mathsf{EvalMod}$, which also fits the purpose to raise modulus and eliminate $q_0$-overflows, while having the advantages in efficiency, and the possibility of evaluating an extra function $f$ during this procedure. In fact, for this purpose, we only need to compose the function with the original circuit approximating the modular reduction operation modulo $q_0$, resulting approximately in a $q_0$-periodic function which is identical to $f$ on the space of message slots $(-q_0/2, q_0/2)$. By replacing the evaluation of modular reduction with this periodic function, we are able to evaluate $f$ during the process of bootstrapping.

On the other hand, our further observation is that there is no need to approximate $f$ over the whole range of $(-q_0/2, q_0/2)$ since the absolute size of the message in practice will be sufficiently smaller than $q_0$. Indeed, a common algorithm for evaluating the modular reduction applies the composition of function $\arcsin(\cdot)$ with function $\sin(\cdot)$, which covers a central half range of each period (Fig. 2a). If we compose the extra function $f$ with the arcsin function, produce a circuit approximating this composite function (with minimax polynomial) and replace the original circuit for approximating arcsin, then we are able to perform functional bootstrapping with the CKKS scheme, nearly without affecting the computation overhead, i.e. with just a similar amount of depths consumed.

Nevertheless, for a function $f$ that requires a high-degree polynomial to approximate, fitting it into the original circuit for arcsin will sacrifice precision of output from evaluating $f$, since the circuit for arcsin usually requires only a relatively low-degree polynomial for the approximation. For mitigating this problem, we have the insight that the evaluation of sin can also output the result of function cos with little or none of extra cost, and every real function defined around 0 can be split into a sum of an odd function and an even function. If we let $f(x) = f_{\text{odd}}(x) + f_{\text{even}}(x)$ be the decomposition, then we can approxi-

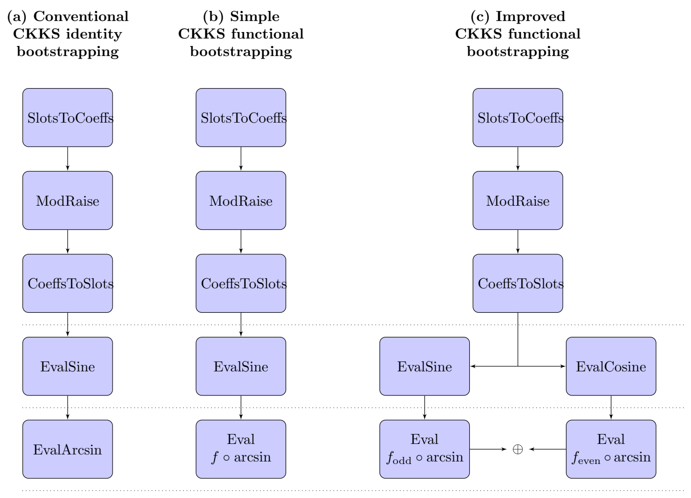

(a) Conventional CKKS identity bootstrapping
(b) Simple CKKS functional bootstrapping
(c) Improved CKKS functional bootstrapping
SlotsToCoeffs
ModRaise
CoeffsToSlots
EvalSine
EvalCosine
EvalArcsin
Eval $f \circ$ arcsin
Eval $f_{\text{odd}} \circ$ arcsin
Eval $f_{\text{even}} \circ$ arcsin
$\oplus$

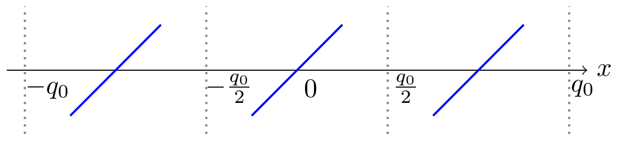
Fig. 1: The functional bootstrapping workflow, where steps between a same pair of dotted lines consume approximately the same levels. Some minor tweaks are omitted in subfigure (c). Details refer to Section 4.2.

mate $f_{\text{odd}} \circ \arcsin$ and $f_{\text{even}} \circ \arcsin$[2] with minimax polynomials independently, which improves the approximation precision and makes better use of the original bootstrapping process.

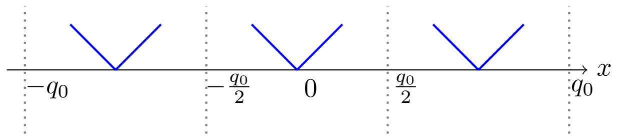
We apply such idea to the evaluation of the $\mathsf{ReLU}$ function, which proves to be successful. First, it is easy to see that $\mathsf{ReLU}(x) = \frac{1}{2}x + \frac{1}{2}|x|$, i.e., the odd and the even parts of $\mathsf{ReLU}$ are the scaled identity function and the absolute value function, respectively. Therefore, we have $\mathsf{ReLU}(x \bmod q_0) = \frac{1}{2}(x \bmod q_0) + \frac{1}{2}|x \bmod q_0|$ which in turn equals $\frac{\pi}{4}\arcsin(\sin(2\pi/q_0 \cdot x)) - \frac{\pi}{8}\arcsin(\cos(4\pi/q_0 \cdot x)) + \frac{1}{16}$ when $x$ is guaranteed to be within $[-1/4 \cdot q_0, 1/4 \cdot q_0] + q_0\mathbb{Z}$ (Fig.2). Hence, the computation of $\mathsf{ReLU}$ is able to be integrated into the original bootstrapping process with only minor modification, and the resulting precision can be sufficiently satisfactory with only a moderate increase in computational overhead. Furthermore, since evaluating the non-linear operations in a neural network model in an HE-based privacy-preserving inference task costs much more computational overhead, in particular, usually there is only one $\mathsf{ReLU}$ evaluation between two bootstrapping when performing the inference with CKKS, it is seamless to integrate the evaluation of $\mathsf{ReLU}$ into the process of bootstrapping.

The technical core of our approach lies in the precise and efficient approximation of the arcsin function, a critical component in composing ReLU with CKKS bootstrapping. Section 3 details our extensive experimental efforts to derive minimax polynomials for arcsin that balance numerical precision, computational efficiency, and compatibility with FHE constraints. The challenge stems from the need to approximate arcsin over domains close to its singularities (e.g., near -1 or 1), where naive polynomial approximations suffer from coefficient

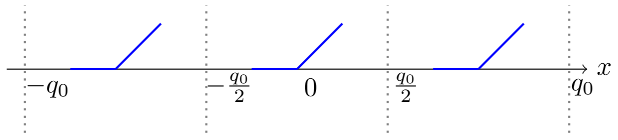
[2] Some minor tweak is omitted here. Ref. Section 4.2.



(a) $x \bmod q_0,\ x \in [-q_0/4, q_0/4] + q_0\mathbb{Z}$



(b) $|x \bmod q_0|,\ x \in [-q_0/4, q_0/4] + q_0\mathbb{Z}$



(c) $\mathsf{ReLU}(x \bmod q_0) = \frac{1}{2}(x \bmod q_0) + \frac{1}{2}|x \bmod q_0|$

Fig. 2: Turning original $\mathsf{EvalMod}$ (subfigure (a)) into different modes of evaluation

explosion and numerical instability. To address this, we adapt the Remez algorithm with some key optimizations: (1) dedicate tuning on the precision setting in the multi-precision floating-point arithmetic, (2) initializing the algorithm using Fourier series approximations for faster convergence, and (3) employing adaptive root-finding to handle densely clustered extrema near domain boundaries. We also examine different approximation domains and determine the most suitable one for homomorphic evaluation. These optimizations enable high-precision minimax polynomials (e.g., achieving 14-bit accuracy with degree-127 approximations) while avoiding overflow/underflow issues inherent in fixed-point FHE computations. The resulting arcsin approximations directly enable a precise evaluation of the absolute value function along with modular reduction, which is the key component in evaluating $\mathsf{ReLU}$ during bootstrapping.

We also propose a simple technique for merging the evaluation of the extra cosine function used in our method into the original step of evaluation of the sine function in the bootstrapping process, in which way the evaluation of the cosine function comes with minimal cost.

## 1.2 Related Works

*FHE-based neural network (NN) inference.* Some privacy-preserving inference works [30,3,59] employ low-degree polynomial approximations for activation functions to reduce computational complexity. They essentially modify the model and require additional fine-tuning to preserve accuracy, which introduces practical deployment challenges. First, the fine-tuning process often requires access to the original training dataset, which may be unavailable. Second, the fine-tuned approximations may lack generalizability for other models or tasks.

Other works [23,38,26] employ high-degree polynomial approximations to achieve reasonable accuracy without fine-tuning requirements. They inherently accept the substantial computational overhead of activation functions, and focus on optimizing other components (e.g., efficient convolution implementations or bootstrapping scheduling). Our work is orthogonal to these works since we specifically optimize the nonlinear activation bottleneck. This complementary relationship suggests that our method could be integrated with them to achieve more efficient FHE-based NN inference.

*MPC-based NN inference.* Works [35,63,40] combining homomorphic encryption and Secure multi-party computation (MPC) have achieved remarkable efficiency, enabling ResNet-50 inference on ImageNet in minutes. However, their reliance on frequent client-server interaction presents a fundamental limitation compared to non-interactive FHE computations. These methods may be preferable in environments with exceptionally reliable, high-bandwidth network connectivity.

## 2 Preliminaries

In this section, we outline the essential background knowledge of FHE and CKKS, and establish the notations used throughout the paper.

We denote individual elements—such as numbers or polynomials—in italics, e.g. $a$ and $b$, and vectors in bold, e.g., $\mathbf{a}$ and $\mathbf{b}$. The notation $\langle \mathbf{a}, \mathbf{b} \rangle$ represents the inner product between the two vectors. We denote by $\| \cdot \|_\infty$ (resp. $\mathsf{hw}(\cdot)$) the infinity norm (resp. the Hamming weight) of a vector, or of the vector of coefficients of a polynomial. We denote by $[x]_Q$ the remainder of $x$ modulo $Q$, and by $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$, and $\lfloor \cdot \rceil$ the flooring, ceiling and rounding functions, respectively. For $x$ being a polynomial, let $\lfloor x \rfloor$, $\lceil x \rceil$ or $\lfloor x \rceil$ denote these operations performed coefficient-wisely. The base of the logarithm in this paper is two.

Let $\mathcal{C} = \{q_0, q_1, \ldots, q_L\}$ be a set of pairwise co-prime positive integers (modulus chain), and $Q_L \coloneqq \prod_{i=0}^{L} q_i$, $Q_\ell \coloneqq \prod_{i=0}^{\ell} q_i$, $\ell = 0, \ldots, L-1$. Then the RNS (short for residue number system) representation of $a \in \mathbb{Z}_Q$ with respect to $\mathcal{C}$ is denoted by $[a]_\mathcal{C} = ([a]_{q_0}, [a]_{q_1}, \ldots, [a]_{q_L}) \in \mathbb{Z}_{q_0} \times \cdots \times \mathbb{Z}_{q_L}$, which by the Chinese Remainder Theorem is a isomorphism.

For a power-of-two integer $N \in \mathbb{Z}$, we define $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ to be the ring of integers of $2N$-th cyclotomic field which is usually treated as set of polynomials of degree $< N$, and let $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$ for some integer $Q$.

It is known that the canonical embedding of the cyclotomic field $\mathbb{Q}[X]/(X^N + 1)$ into copies of $\mathbb{C}$ is an FFT-like linear transformation. Now we let $\mathsf{DFT}$ : $\mathbb{R}[X]/(X^N + 1) \to \mathbb{C}^{N/2}$ be this mapping generalized on polynomials with $\mathbb{R}$-coefficients and only taking one conjugation from each conjugate pair in the following manner:

$$\mathsf{DFT}(p(X)) = \{p(\zeta^{5^i}); i = 0, \ldots, N/2 - 1\} \in \mathbb{C}^{N/2},$$
$$\forall p(X) \in \mathbb{R}[X]/(X^N + 1),$$

where $\zeta$ is a fixed primitive complex $2N$-th root of unity. The inverse of this process is denoted as $\mathsf{iDFT}\colon \mathbb{C}^{N/2} \to \mathbb{R}[X]/X^N+1$.

For convenience, we refer to functions $\{\sin(ax) : a \in \mathbb{R}_+\}$ (resp. $\{\cos(ax) : a \in \mathbb{R}_+\}$) as the sine functions (resp. cosine functions). We call the monomials $\{1, x, x^2, \ldots, x^d\}$ as the power basis of polynomial of degree $\leq d$, and call the set of first $d+1$ Chebyshev polynomials $\{T_0(x), T_1(x), \ldots, T_d(x)\}$ as the Chebyshev basis of polynomial of degree $\leq d$.

## 2.1 Approximate Homomorphic Encryption (CKKS)

We now revisit the CKKS homomorphic encryption scheme [20], which is built upon the Ring-Learning with Errors (RLWE) hard problem and enables homomorphic computation over fixed-point numbers.

*Basic operations.*

- $\mathsf{Ecd}(\mathbf{m}, \mathit{\Delta}, N)$(*coefficients→slots*): Given a message $\mathbf{m} \in \mathbb{C}^{N/2}$, the CKKS encoding $\mathbb{C}^{N/2} \to \mathbb{R}[X]/(X^N+1) \to \mathcal{R}$ is defined as:

$$\forall \mathbf{m} \in \mathbb{C}^{N/2} : \mathsf{Ecd}(\mathbf{m}) = \lfloor \mathit{\Delta} \cdot \mathsf{iDFT}(\mathbf{m}) \rceil ,$$

  where $\mathit{\Delta}$ represents the scaling factor. We call the resulting element in $\mathcal{R}$ the plaintext polynomial, and call the entries $m_i \in \mathbf{m}$ the slots of the plaintext (or the ciphertext when after encryption). Due to the property of canonical embedding of being isomorphism, it is easy to see that the CKKS encoding algorithm is invertible and homomorphic approximately.
- $\mathsf{Dcd}(p, \mathit{\Delta}, N)$ (*slots→coefficients*): Given plaintext polynomial $p \in \mathcal{R}$, the CKKS decoding algorithm is the approximate inverse of $\mathsf{Ecd}$, defined as:

$$\forall p \in \mathcal{R} : \mathsf{Dcd}(p) = \mathit{\Delta}^{-1} \cdot \mathsf{DFT}(p).$$

- $\mathsf{Enc}(\cdot)$: Given a plaintext polynomial $p = \mathsf{Ecd}(\mathbf{m}) \in \mathcal{R}$ for some message $\mathbf{m}$ and secret key $\mathsf{sk}$, generate an RLWE sample $(b, a) \in \mathcal{R}_Q^2$, and let the encryption on $p$ be $(b' = b + p, a)$.
- $\mathsf{Dec}(\cdot)$: Given a ciphertext $\mathsf{ct} = (b, a) \in \mathcal{R}_Q^2$ under a secret key $\mathsf{sk} = (1, s)$, the decryption computes a plaintext polynomial $\langle \mathsf{ct}, \mathsf{sk} \rangle = b + as = p \in \mathbb{R}[X]/(X^N+1)$. If the ciphertext is supposed to be carrying plaintext polynomial $p_0$ either by fresh encryption or by homomorphic evaluation, then we say the decryption is successful if $p \approx p_0$.
- $\mathsf{Add}(\mathsf{ct}_1,\mathsf{ct}_2)$: For $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}_{Q_\ell}^2$, output $\mathsf{ct}_{\mathrm{add}} = \mathsf{ct}_1 + \mathsf{ct}_2 \pmod{Q_\ell}$.
- $\mathsf{cMult}(\mathsf{ct}, a)$: For $a \in \mathbb{Z}$, output $\mathsf{ct}_{\mathrm{cmult}} = a \cdot \mathrm{ct}$ without applying $\mathsf{Ecd}(\cdot)$ , relinearization and rescale $\mathsf{RS}(\mathsf{ct})$ after $\mathsf{Mult}$.
- $\mathsf{Mult}(\mathsf{ct}_1,\mathsf{ct}_2)$: Given $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}_{Q_\ell}^2$, output a level-downed (explained below) ciphertext $\mathsf{ct}_{\mathrm{mult}} \in \mathcal{R}_{Q_{\ell-1}}^2$. Let $\mathsf{ct}_1 = (b_1, a_1)$, $\mathsf{ct}_2 = (b_2, a_2)$. With the tools of the key switching operation and rescaling operation introduced below, this operation first applies tensor product of $\mathsf{ct}_{\mathrm{new}} = \mathsf{ct}_1 \otimes \mathsf{ct}_2 = (b_1b_2, b_1a_2 + b_2a_1, a_1a_2) \in \mathcal{R}_{Q_\ell}^3$, then do a key switching, summing into $\mathsf{ct}_{\mathrm{rel}} = \mathsf{KS}_{s^2 \to s}((0, a_1a_2), \mathsf{swk}_{\mathrm{rel}}) + (b_1b_2, b_1a_2 + b_2a_1) \in \mathcal{R}_{Q_\ell}^2$, and finally rescale it into $\mathsf{ct}_{\mathrm{mult}} = \mathsf{RS}(\mathsf{ct}_{\mathrm{rel}}) \in \mathcal{R}_{Q_{\ell-1}}^2$.

- Key switching ($\mathsf{KS}$) is required during homomorphic multiplication and automorphism. It is defined as $\mathsf{KS}_{s'\to s}(\mathsf{ct}_{s'}, \mathsf{swk}_{s'\to s})$, where the key switching key $\mathsf{swk}$ is approximately the encryption of $s'$ under $s$. The operation takes as input a ciphertext $\mathsf{ct}_{s'}$ under the secret key $s'$, and outputs $\mathsf{ct}_s$ which is a ciphertext with the same messages under the secret key $s$.
- $\mathsf{RS}(\mathsf{ct})$: For $\mathsf{ct} \in R^2_{Q_\ell}$, output $\mathsf{ct}_{\mathsf{RS}} = \lfloor q_\ell^{-1} \cdot \mathsf{ct} \rceil \pmod{Q_{\ell-1}}$. The rescaling process serves two purposes: it reduces the error size and maintains the scaling factor for each slot.

We say a ciphertext is at level $\ell$ if its modulus $Q_\ell$ is the product of $\ell+1$ primes, denoting $Q_\ell = q_0 q_1 \dots q_\ell$. It is a common practice to perform a rescaling following each $\mathsf{cMult}$ or each $\mathsf{Mult\text{-}KS}$ pair, hence a product ciphertext from multiplying ciphertexts/a ciphertext and a plaintext of level $\ell$ will certainly be of level $\ell' = \ell - 1$. We also use the term "depths" along with "levels" interchangeably.

*Bootstrapping.* In this paper by CKKS (identity) bootstrapping we refer to the algorithms which utilize modulus raising and evaluation of sine function. (There are also a few other CKKS identity bootstrapping methods relying on third generation FHE, e.g. [41], which are out of our discussion.) The bootstrapping process of the CKKS scheme [16] seeks to elevate the ciphertext to a higher modulus, enabling further homomorphic evaluation. The process consists of four subroutines: $\mathsf{SlotsToCoeffs}$, $\mathsf{ModRaise}$, $\mathsf{CoeffsToSlots}$ and $\mathsf{EvalMod}$, which are composed into one of the following workflow.

$$p(x) \xrightarrow{\mathsf{ModRaise}} p(x) + q_0 I(x) \xrightarrow{\mathsf{CtS}} \mathbf{p} + q_0\mathbf{I} \xrightarrow{\mathsf{EvalMod}} \mathbf{z} \xrightarrow{\mathsf{StC}} p(x)$$
$$\mathbf{z} \xrightarrow{\mathsf{StC}} z(x) \xrightarrow{\mathsf{ModRaise}} z(x) + q_0 I(x) \xrightarrow{\mathsf{CtS}} \mathbf{z} + q_0\mathbf{I} \xrightarrow{\mathsf{EvalMod}} \mathbf{z}$$

We briefly explain these subroutines for clarity.

**SlotsToCoeffs:** The inverse of the canonical embedding evaluates $\mathsf{DFT}$ matrix multiplication homomorphically on the encrypted ciphertext that can decrypt to the vector $\mathbf{z}$. This operation converts it into the polynomial form $z(x)$.

**ModRaise:** The ciphertext at (w.l.o.g.) modulus $q_0$ is expressed in the larger modulus $Q \gg q_0$. This results in a ciphertext that decrypts to $[\langle \mathrm{ct}, \mathrm{sk}\rangle]_Q = z(x) + q_0 I(x)$, where $q_0 I(x)$ is an integer polynomial with coefficients that are small multiples of the base modulus $q_0$.

**CoeffsToSlots:** The canonical embedding evaluates the $\mathsf{iDFT}$ matrix multiplication homomorphically on $z(x) + q_0 I(x)$. To enable parallel (slot-wise) evaluation, the polynomial must be encoded in the slot domain, converting it to a ciphertext that decrypts to $\mathbf{z} + q_0\mathbf{I}$.

**EvalMod:** A polynomial approximation of the modular reduction by $q_0$ is homomorphically evaluated, thus removing the redundant $q_0\mathbf{I}$ and resulting in a ciphertext decrypted to $\mathbf{z}$. The modular reduction function is usually approximated by a sine function, or in a more advanced way, by composition of a sine functions with a scaled arcsin function or by combination of sine functions (i.e. Fourier approximation).

**Different Variants of Bootstrapping Workflow** There are mainly two variants of CKKS identity bootstrapping in terms of the overall workflow. In early works (e.g. [16,13,34]), the bootstrapping procedure first raises the modulus of a ciphertext from $q_0$ to a large $Q' = q_0 \dots q_L p_0 \dots p_{k-1}$, and then consumes $k$ levels for the evaluation of $\mathsf{CoeffsToSlots}$, the approximate modular reduction function $\mathsf{EvalMod}$, and $\mathsf{SlotsToCoeffs}$, to result in a level-$L$ ciphertext carrying messages somehow close to the original ones. The idea behind this procedure is very clear: after level lifting, the encrypted plaintext polynomial turns from $p(X) = c_0 + c_1 X + \cdots + c_{n-1}X^{n-1}$ into $p(X) + q_0 \cdot I(X) = (c_0 + I_0 q_0) + \cdots + (c_{n-1} + I_{n-1} q_0)X^{n-1}$, where the $q_0$-overflows within the coefficients are what we want to eliminate. On the other hand, it is not feasible to perform SIMD operations on the coefficients of a plaintext, hence before (resp. after) the $\mathsf{EvalMod}$ step, a $\mathsf{CoeffsToSlots}$ (resp. a $\mathsf{SlotsToCoeffs}$) process is performed. It is easy to see the correctness of this whole procedure as a CKKS bootstrapping algorithm.

**Algorithm 1** Workflow of CKKS Identity Bootstrapping (Early Variant)

```
   Input: $\mathsf{ct} = \mathsf{Enc}_{\mathsf{sk}}(\mathbf{m}) \in \mathcal{R}_q^2$.
   Output: $\mathsf{ct}_{\text{out}} \in \mathcal{R}_Q^2$.
1: $\mathsf{ct}_1 \leftarrow \mathsf{ModRaise}(\mathsf{ct})$
2: $\mathsf{ct}_2 \leftarrow \mathsf{CoeffsToSlots}(\mathsf{ct}_1)$
3: $\mathsf{ct}_3 \leftarrow \mathsf{EvalMod}(\mathsf{ct}_2)$
4: $\mathsf{ct}_{\text{out}} \leftarrow \mathsf{SlotsToCoeffs}(\mathsf{ct}_3)$
5: return $\mathsf{ct}_{\text{out}}$
```

Nevertheless, a newer type of CKKS bootstrapping workflow (also called slim bootstrapping) which run $\mathsf{SlotsToCoeffs}$ at first is more widely-used. In this paper we use this slim variant and describe it as follows.

**Algorithm 2** Workflow of CKKS Identity Bootstrapping (Slim Variant)

```
   Input: $\mathsf{ct} = \mathsf{Enc}_{\mathsf{sk}}(\mathbf{m}) \in \mathcal{R}_q^2$.
   Output: $\mathsf{ct}_{\text{out}} \in \mathcal{R}_Q^2$.
1: $\mathsf{ct}_1 \leftarrow \mathsf{CoeffsToSlots} \circ \mathsf{ModRaise} \circ \mathsf{SlotsToCoeffs}(\mathsf{ct})$
2: $\mathsf{ct}_2 \leftarrow 1/2 \cdot (\mathsf{conj}(\mathsf{ct}_1) + \mathsf{ct}_1)$
3: $\mathsf{ct}_{\text{out}} \leftarrow \mathsf{EvalMod}(\mathsf{ct}_2)$
4: return $\mathsf{ct}_{\text{out}}$
```

## 2.2 Approximation Theory

The potential of polynomials in approximating a targeted continuous function is guaranteed by Weierstrass' well-known theorem.

**Theorem 1 (Weierstrass approximation theorem[60]).** *Suppose $f$ is a continuous real-valued function defined on a real interval $[a, b]$. For any $\epsilon > 0$, there exists a polynomial $p$ such that $\|f - p\|_{\infty,[a,b]} \leq \epsilon$.*

When a polynomial $p(x)$ among those of degree $\leq d$ achieves the minimal uniform norm distance from $f(x)$ on the interval $[a, b]$, we call $p$ the minimax polynomial of degree $\leq d$ of $f$ on $[a, b]$. Chebyshev's equioscillation theorem characterizes the minimax polynomials, which is also a guidance of the Remez Algorithm.

**Theorem 2 (Chebyshev's Equioscillation Theorem[60]).** *Let $f(x)$ be a continuous function from $[a, b] \subset \mathbb{R}$ to $\mathbb{R}$, and let $p(x)$ be a polynomial among those of degree $d$ which has the minimal uniform norm distance from $f$, denoted by $E \coloneqq \|f - p\|_{\infty,[a,b]}$. Then it holds that there are at lease $d + 2$ points $a \leq x_0 < x_1 < \cdots < x_{d+1} \leq b$ such that $f(x_i) - p(x_i) = \sigma\|f - p\|_\infty$ where $\sigma = \pm 1$. The converse is also true.*

The following algorithm by E. Remez [64] is proved to converge and output the degree-$d$ minimax polynomial with target function and interval given.

For our reference below, we also list Jackson's theorem on the asymptotic decreasing of distance between the minimax polynomial $p$ and the target function $f$, when $f$ is a smooth function.

**Theorem 3 (Jackson's Theorem).** *Suppose $f(x) \in C^\infty([a, b])$, and denote by $p_d(x)$ its minimax polynomial of degree $d$ on $[a, b]$, then there exist real constants $k < 0, l, 0 < \alpha < 1$, such that $\|f - p_d\|_{\infty,[a,b]} \leq \mathrm{e}^{kd^\alpha + l}$.*

## 3 Minimax Approximation for arcsin

Recapping our idea of evaluating ReLU: after computing a pair of sine and cosine functions during bootstrapping, we compose them with arcsin. This transforms them into a scaled identity function and absolute value function, respectively. ReLU is then obtained by summing these two results:

$$\begin{aligned} \mathsf{ReLU}(x \bmod 1) &= \frac{1}{2}(x \bmod 1) + \frac{1}{2}|x \bmod 1| \\ = \frac{1}{4\pi}\arcsin(\sin(2\pi x)) &+ \frac{1}{8\pi}\arcsin(-\cos(4\pi x)) + \frac{1}{16}. \end{aligned} \tag{1}$$

One of our main challenges is to produce polynomial approximations of arcsin over a suitable domain. Intuitively the smaller the domain is, the closer an approximation could be. Notice that we focus on the case of $x$ being relatively close to the modulus 1, and $-\cos(4\pi\bullet)$ maps a neighborhood of $k \in \mathbb{Z}$ $[k - \beta, k + \beta]$ to $[-1, \beta']$ where $\beta' = -\cos(4\pi\beta)$, hence we need to approximate arcsin near -1.

On the other hand, we encountered the problems of coefficient explosion or intermediate value explosion (see Section 3.2 for more details) when searching for an approximation of the scaled arcsin function over $[-1, \beta]$ with a small $\beta$ close to -1. Ultimately, we chose to perform the approximation over the larger symmetric

**Algorithm 3** Remez Algorithm for Minimax Approximation

```
Input:
    Target function $f(x)$
    Interval $[a, b]$
    Polynomial degree $d$
    Tolerance $\epsilon$
    Maximum iterations $K_{\max}$
Output:
    Optimal coefficients $\mathbf{c}$
    Reference node set $\{x_k\}$
1: Initialize reference nodes (usually with $n+2$ Chebyshev nodes $\{x_k^{(0)}\}$ on $[a, b]$)
2: for $i \leftarrow 1$ to $K_{\max}$ do
3:     Build and solve linear system:
               $\begin{cases} \sum_{j=0}^{n} c_j x_k^j + (-1)^k E = f(x_k) \\ \forall x_k \in \{x_k^{(i-1)}\} \end{cases}$
           Solve the system to obtain $\mathbf{c}^{(i)}, E^{(i)}$
4:     Define error function: $e^{(i)}(x) \coloneqq f(x) - \sum_{j=0}^{n} c_j^{(i)} x^j$
5:     Find extrema:
           Locate local extrema of $e^{(i)}(x)$ between adjacent zeros of the error function
           Select $n+2$ extremum candidates $\{\tilde{x}_k\}$
6:     if $|E^{(i)} - E^{(i-1)}| < \epsilon$ then break
7:     Update reference nodes: $\{x_k^{(i)}\} \leftarrow \{\tilde{x}_k\}$
8: Return: $\mathbf{c} \leftarrow \mathbf{c}^{(i)}$, $\{x_k\} \leftarrow \{x_k^{(i)}\}$
```

domain $[-1, 1]$, with a slight relaxation. We make the following definition which is migrated from [45].

**Definition 1.** *For $\alpha > 0$ and $0 < \epsilon < 1$, a polynomial $p$ is said to be $(\alpha, \epsilon)$-close to a function $f(x)$ over $[-1, 1]$ if $p$ satisfies the following: $\|p(x) - f(x)\|_{\infty,[-1+\epsilon,1-\epsilon]} \leq 2^{-\alpha}$, where $\|\cdot\|_{\infty,D}$ denotes the uniform norm over the domain $D$.*

We exclude the $\epsilon$-neighborhoods of the end points only for relaxing the condition on the approximation task. Indeed, there is no essential obstacle in searching for the minimax polynomial over the "full domain" $[-1, 1]$. This limitation also exists in prior works. It is inevitable for [45] to exclude a small neighborhood of 0 when approximating the sign function ($x \mapsto 1$ if $x \geq 0$, or else 0), because the sign function is discontinuous, and any polynomial will display a non-neglectable distance of up to 1 in some neighborhood of 0.

### 3.1 Challenges in Approximating arcsin

Although in theory Alg. 3 is general for piecewisely differentiable functions, one would face many challenges in implementation and execution of the algorithm, as in our case of approximating arcsin.

*Singularities and coefficient explosion.* The approximation of the arcsin function faces inherent challenges due to its singularities at $x = \pm 1$, where the derivative diverges to infinity. In the power basis representation, polynomial approximations of such functions exhibit coefficient explosion – exponentially growing coefficients that destabilize numerical computations. This phenomenon is exacerbated in fixed-precision FHE settings (typically 40–60 fractional bits).

On the other hand, while polynomials over Chebyshev basis mitigate coefficient explosion by leveraging their oscillatory structure, the coefficients may still grow unboundedly along with the degree of approximation polynomial. In our case, this becomes a tradeoff over the approximation domain: although an approximation of arcsin in the area near $-1$ (i.e. $[-1, \beta]$ for a small $\beta$) is sufficient to be turned into an approximation of $|\cdot|$ (and in turn $\mathsf{ReLU}(\cdot)$) near 0, choose a "full" domain $[-1, 1]$ helps effectively bound the coefficients of the minimax polynomials over Chebyshev basis, enabling practical CKKS homomorphic evaluation.

*Precision-exception trade-offs.* Polynomial degree directly impacts multiplicative depth in FHE, creating a critical trade-off between approximation accuracy and computational efficiency. While Theorem 3 guarantees exponential error decay for smooth functions with increasing degree $d$ (Theorem 3.3), with the same degree $d$ the minimax polynomials on different ranges could provide different precision result. For example, from our experiment achieving 8-bit accuracy on the full domain of arcsin $[-1, 1]$ would require a degree-32 polynomial, which consumes 5 multiplicative levels in CKKS homomorphic evaluation. On the other hand, if we relax the approximation range to $[-1 + \epsilon, 1 - \epsilon]$ with $\epsilon \approx 10^{-4}$, the same level of precision may be achieved by a lower-degree polynomial, as shown in Table 1, at the cost that there would be an exception area $(-\epsilon', \epsilon')$ in the approximation of the absolute value (and in turn the $\mathsf{ReLU}$) function.

Fortunately, the numerical stability of convolutional neural networks allows such exception in the range of an approximated operator. We run simulation experiments in plaintext, and the results in Section 5.2 show that there is no essential impact by replacing standard $\mathsf{ReLU}$ by our approximation in inference.

*Implementation complexity and numerical stability.* The practical realization of the Remez algorithm for arcsin approximation introduces certain critical implementation challenges tied to numerical precision and algorithmic robustness.

- **High-precision arithmetic requirements:** The Remez iteration inherently demands multi-precision arithmetic to avoid numerical instabilities during polynomial optimization. For instance, near the singularities at $x =$

$\pm 1$, the error function $e(x) = f(x) - p(x)$ exhibits extreme sensitivity to coefficient perturbations, causing catastrophic cancellation in low-precision settings. While previous works [69] utilize 1000-bit multi-precision arithmetic using the NTL library, our implementation leverages the `mpmath` library with only 100$\sim$300-bit floating-point precision due to our dedicated realization of the Remez Algorithm.

– **Derivative computation and root-finding:** Locating extrema of the error function $e(x)$ (Step 5 of Algorithm 3) requires robust derivative calculations and root-finding. Unfortunately usual multi-precision arithmetic libraries lack the ability of symbolic differentiation. We adopt the workaround of computing the expression of the derivative function $f'(x)$ independently. Furthermore, the dense clustering of extrema (e.g. near $x = \pm 1$ in our case) demands denser scanning of roots of the error function.
– **Iterative stability and initialization:** The Remez algorithm's convergence is highly sensitive to initial reference nodes. Poorly chosen starting points (e.g., equidistant nodes) may lead to oscillatory coefficient updates and divergence. Additionally, solving the linear system in Step 3 of Alg. 3 would involve ill-conditioned Vandermonde matrices, when insufficient reference nodes are produced in the previous iteration.

These challenges highlight the gap between theoretical guarantees of the Remez algorithm and its practical deployment in FHE-critical contexts. Our solutions decribed below explicitly address these barriers, transforming a mathematically elegant method into a robust tool for privacy-preserving machine learning.

## 3.2 Algorithmic Innovations for Robustness

**Chebyshev Basis for Stability** Although in mathematics we can approximate a function with polynomials with accuracy as high as possible, in the case of CKKS homomorphic encryption we face the limit of numerical precision, where numerical data is encoded as fixed-point numbers with usually only 40-60 bits of fractional part. For an input real number $x$ and a sufficiently large $d$, computing its $d$-th power will either end up in overflow or underflow, depending on whether $|x| > 1$ or $|x| < 1$. What is worse, usually the value of the polynomial evaluated on $x$ is small in norm, while the intermediate values of $x$'s powers are much larger numbers, which means precision is being lost therein.

Fortunately there are other approaches for evaluating a polynomial. In fact, evaluating the polynomial through Chebyshev basis instead of power basis can mitigate the problem above. [13] demonstrated with experiments that representing the approximation of sine functions with Chebyshev basis can reduce the coefficients by magnitudes. In our case of approximating arcsin, representing the resulting polynomial with Chebyshev basis also reduces the coefficients a lot, and since the input of arcsin is always within $[-1, 1]$, there is no worry about overflowing.

On the other hand, with resulting minimax polynomial represented with Chebyshev basis, the maximum norm of coefficients still grows rapidly with the increase of degree $d$, when the arcsin function is approximated over $[-1, \alpha]$ for almost any possible $\alpha$. Fortunately, this problem disappears if we set the approximation domain to be an symmetric one $[-1, 1]$: in this case, all Chebyshev coefficients are small positive real numbers within $(0, 1)$, which is quite friendly to CKKS homomorphic computation. Indeed, we have the following lemma.

**Lemma 1.** *(Informal) The coefficients of the minimax polynomial of* arcsin *over Chebyshev basis is close to the coefficients of the Fourier series of the triangle wave function* $\arcsin \circ \sin$*, if the approximation domain is* $[-1, 1]$ *(or its relax* $[-1+\varepsilon, 1-\varepsilon]$ *for small* $\varepsilon$*).*

We leave further demonstration of the lemma in Appendix F. Notice that this decision of approximation domain does not come with no cost: a larger domain makes the process of approximation slower as the growth of polynomial degree. We leave the exploration of a more formal solution for future work.

**Fourier Initialization for Faster Convergence** Lemma 1 above indicates the fact that the coefficients over Chebyshev basis of minimax approximation polynomial of arcsin are close to the coefficients of certain Fourier series. On the other side, this proximity can also guide us to better initiate Alg. 3.

In practice, starting Alg. 3 with a randomly initialized polynomial would lead to error functions with bad behavior (e.g. with highly dense zeros). Usually, the initial reference nodes in the Remez algorithm are set to be the Chebyshev node of the interval $[a, b]$, i.e. the set

$$\left\{ p_i = a + \frac{b-a}{2} \cdot \left( 1 - \cos \frac{(2i+1)\pi}{2(d+2)} \right), i = 0, \ldots, d+1 \right\}.$$

In our case of approximating arcsin over $[-1+\epsilon, 1-\epsilon]$, the Fourier series (3) approximates the triangle wave function $\arcsin \circ \sin$ uniformly, hence the corresponding Fourier polynomial of degree $d$ may be a good approximation with respect to uniform norm distance. Therefore, we can start the iteration of Alg. 3 from this polynomial. In Appendix E we show that initialization with Fourier coefficients indeed helps accelerate the convergence of Alg. 3.

Notice that this technique is general for our CKKS functional bootstrapping method. Suppose now we want to evaluate a function $f$ during bootstrapping. For simplicity here we assume $f$ is odd. Now it is required to evaluate an approximation of $f \circ \arcsin$. Suppose $f(x) \approx \sum_{k=1}^{d} c_k \sin(kx)$ is the $d$-th Fourier approximation of $f$, then we can transform this into a degree-$d$ polynomial $F_d$ of $\sin(x)$, so that $f(x) \approx F_d(\sin(x))$, from which it is easy to see $(f \circ \arcsin)(x) \approx F_d(x)$. Hence we can use this degree-$d$ Fourier polynomial $F_d(x)$ to initiate Alg. 3 when approximating $f \circ \arcsin$.

**Adaptive Root-Finding with Ridder's Method** To address the challenge of locating densely clustered roots near interval boundaries in minimax approximation, we propose an adaptive root-finding strategy with multi-stage partition refinement. This is important in the iteration of Alg. 3, since in the first few iterations the candidate polynomial may be extremely far away from the target function in some area, which means in those area the zeros of the error function can be extremely dense.

In our case, when approximating arcsin over $[-1, 1]$, the minimax polynomial oscillates around the target function more frequently near the ends of the interval. Hence we adopt a three-phase hierarchy in which the three phases search for zeros of the error function on the whole interval, the left-most 5% fraction of the interval and the right-most 5% fraction of the interval, respectively. We scan sub-intervals of width $\Delta = (b - a)/N$ in the first phase where $N$ is the number of sub-intervals we divide the domain into, and scan sub-intervals of width $\Delta/10$ in the next phases. Combined with delicate selection of $N$ we manage to locate all the zeros of the error function, guaranteeing the process of the iterations.

**Algorithm 4** Adaptive Root-Finding with Boundary Refinement

```
Input: Error function $e(x)$, interval $[a, b]$, initial partitions $N$, precision $p$
Output: Root set $\mathcal{Z}$
1: Initialize $\mathcal{Z} \leftarrow \emptyset$, $\Delta \leftarrow (b - a)/N$
   Phase 1: Global coarse search
2: for $k \leftarrow 0$ to $N - 1$ do
3:     Apply interval zero finding method on subinterval $[a + k\Delta, a + (k + 1)\Delta]$
4:     Add valid roots to $\mathcal{Z}$ with tolerance $2^{-p}$
   Phase 2: Boundary refinement; executed twice for left/right
5: Repeat step 5-7 search near $a$ and resp. $b$, with 10× finer partition $\Delta/10$
6: Return Deduplicated root set $\mathcal{Z}$
```

For the zero finding within subintervals in step 6, we prefer Ridder's method [65] over bisection. The Ridders' method combines bisection with exponential extrapolation, achieving superlinear convergence while maintaining bracketing reliability.

## 3.3 Experimental Validation

We implemented the Remez algorithm with `SageMath` and the `mpmath` library. With the practices above, we search for approximations of the scaled arcsin with polynomial degree from 15 to 127.

Below we show the precision results, including the property of the corresponding approximation of absolute value function, i.e. the composition of the approximate arcsin function with cosine function. Since the component with sine in our ReLU approximation (i.e. the first term of the right hand side of

(1)) can be made precise more easily, the maximum and average error of the ReLU approximation can be obtained by just taking half of the figures in the last column.

Table 1: Approximation results on arcsin function.

| Setting | Degree $d$ of poly. | Max. and avg. error of approx. arcsin | Max. and avg. error of corresponding approx. abs. func. |
|---|---|---|---|
| Param-I | 15 | $(2^{-8.78}, 2^{-9.43})$ | $(2^{-7.78}, 2^{-8.42})$ |
| Param-II | 31 | $(2^{-10.24}, 2^{-10.86})$ | $(2^{-9.20}, 2^{-9.86})$ |
| Param-III | 63 | $(2^{-12.02}, 2^{-12.69})$ | $(2^{-11.02}, 2^{-11.61})$ |
| Param-IV | 127 | $(2^{-14.54}, 2^{-15.15})$ | $(2^{-13.54}, 2^{-13.97})$ |

Notice that the average error is computed on the full domain $[-1, 1]$, and although we exclude space of $\epsilon$ on each end when searching for the minimax approximation, this hardly affect the average error. Below we show results of Param-I graphically. On the error function's graph, it exceeds the uniform maximum only within an extremely small neighborhood around -1 or 1, which is actually too tiny to be recognized.

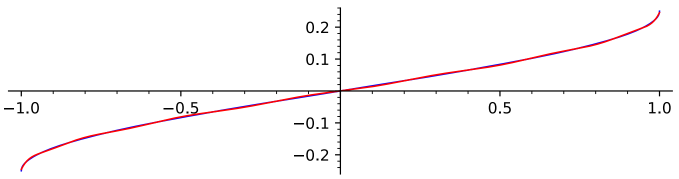


Fig. 3: The function $\frac{1}{2\pi}\arcsin(x)$ (blue) and the approximation with Param-I (red).

In addition to generating polynomial outputs for homomorphic evaluation, we compare our implementation of the Remez algorithm with those in [46,69], which employ significantly higher numerical precision in arithmetic (see Appendix D).

## 4 Methods of CKKS Functional Bootstrapping with Application to ReLU

There can be various methods for integrating the evaluation of arbitrary function into the process of CKKS bootstrapping. For example, consider the most general case when the distribution range of message slots is $[-q_0/2, q_0/2]$, then applying

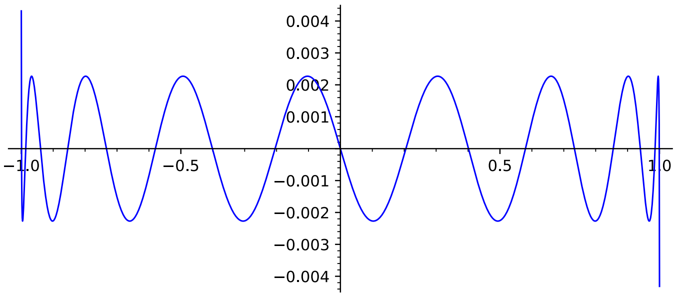


Fig. 4: Difference between $\frac{1}{2\pi}\arcsin(x)$ and the approximation with Param-I.

Fourier series may be the best choice. For convenience, we call this case full-domain functional bootstrapping, which differs from practical cases when the message slots are usually small relative to $q_0$. On the other hand, most of the methods of CKKS identity/functional bootstrapping only vary in their final stage (i.e. those operations after the evaluation of sine), hence we give the following definition.

**Definition 2.** *We call a process with the CKKS homomorphic encryption scheme a functional bootstrapping process for target function $f : \mathbb{R} \to \mathbb{R}$ with $\ell$-level final stage, if it not only lifts the modulus of the ciphertext, but also results in ciphertext carrying the approximate values of $f$ evaluated on the original messages, while consuming $\ell$ levels after the evaluation of the sine function.*

In our methods, the final stage are always the evaluation of polynomials approximating the composition of some function and the arcsin function, and the traditional method of identity bootstrapping evaluating the arcsin function following sine can be seen as a special case of our functional bootstrapping with $\ell_0$-level final stage where $\ell_0$ is the level consumption in evaluating arcsin. Therefore, in practice a functional bootstrapping method with $\ell$-level final stage for some $f$ is acceptable only if $\ell$ is not much larger than in the case of an identity bootstrapping [7].

### 4.1 Simple Functional Bootstrapping: Evaluating Minimax Polynomial of Composition Functions

Since we don't always need a full-domain evaluation with CKKS bootstrapping (where $|M|$ can be up to $q_0/2$), it is more practical to assume the message slots before bootstrapping are within a moderately large range so that we can utilize methods that will be faster in converging and less costly in computation. In practice, we assume that the messages $M$ contained in the ciphertext to bootstrap are within the half range $(-q_0/4, q_0/4)$.

In the case when it is guaranteed $-q_0/4 < M < q_0/4$, it is known that the composition of functions $z = \arcsin(y)$ and $y = \sin(2\pi x)$ is identical to $\mathsf{id}$ within this range, and can be approximated by polynomials in terms of uniform norm, hence the evaluation of a sine function followed by a scaled arcsin function is the most common method for evaluating modular reduction in the CKKS identity bootstrapping. Now if a function $f : (-1/4, 1/4) \to \mathbb{R}$ is just following the bootstrapping procedure, then traditionally one would choose to evaluate $f$ after the evaluation of the sine function and the arcsin. However, this type of workflow may possibly require more depth consumption, in the case when we can merge the evaluation of $f$ into the bootstrapping process. In that, we propose to compose the function $f$ with arcsin, denoted as $\mathsf{EvalFArcsin} := f \circ \text{arcsin}$, and then approximate it with a polynomial of a degree close to that which approximates arcsin formerly. In this way, we turn an identity bootstrapping with $\ell$-level final stage, into a functional bootstrapping with $\ell'$-level final stage, where $\ell' \approx \ell$. Therefore, we are able to obtain an evaluation of $f$ without affecting the depth of the entire workflow much, while making the bootstrapping process functional. Below we list the workflow described above.

**Algorithm 5** Workflow of CKKS Functional Bootstrapping with Target Function

```
    Input: $\mathsf{ct} = \mathrm{Enc}_{\mathsf{sk}}(\mathbf{m}) \in \mathcal{R}_q^2$, a polynomial approximating $f \circ \frac{1}{2\pi}$ arcsin.
    Output: $\mathsf{ct}_{\text{out}} \in \mathcal{R}_Q^2$.
1:  $\mathsf{ct}_1 \leftarrow \mathsf{CoeffsToSlots} \circ \mathsf{ModRaise} \circ \mathsf{SlotsToCoeffs}(\mathsf{ct})$
2:  $\mathsf{ct}_2 \leftarrow 1/2 \cdot (\mathsf{conj}(\mathsf{ct}_1) + \mathsf{ct}_1)$
3:  $\mathsf{ct}_3 \leftarrow \mathsf{EvalSine}(\mathsf{ct}_2)$
4:  $\mathsf{ct}_{\text{out}} \leftarrow \mathsf{EvalFArcsin}(\mathsf{ct}_3)$
5:  return $\mathsf{ct}_{\text{out}}$
```

As mentioned above, for the approximation of $F = f \circ \text{arcsin}$, many methods could be considered, including the method of Taylor/Fourier series. On the other hand, since we have confined the input to a fixed range, a minimax polynomial that has a minimal uniform norm distance from the given target function on a given interval would be a more suitable method for our case.

In terms of searching for a minimax polynomial on a differentiable function over a given interval, the Remez algorithm is de facto the standard method. The practice of applying the Remez algorithm is somewhat similar to a training process of a statistical machine learning model: it iteratively computes some "loss" value and updates the candidate polynomial based on it. On the other hand, in Section 3, we have shown that there are still many details in the algorithm to be considered in practice, and we have made a lot of fine-tuning to make the algorithm work and obtain the polynomial satisfying the minimax property.

Besides the practice of applying the Remez algorithm for minimax approximation, we also implemented machine learning-based algorithms for searching for minimax polynomials of target functions over a certain range, which proved not successful. The experiments using machine learning seemed simple and robust, where we just chose some arbitrary polynomial as the initialization, and the gradient descent process ran steadily. However, when the process converged, what we obtained is not an approximation satisfying the minimax condition. We attribute this to the fact that the space of polynomials is large and there are lots of local minima during the gradient descent process. This indicates again that searching for minimax polynomials is a task far from being trivial, since there are basically only the Remez-style algorithms that work, which still require dedicated fine-tuning in practice as described in previous sections.

## 4.2 Improved Functional Bootstrapping: Splitting Evaluation into Odd and Even Part

Although in theory, for each continuous function $f$ to be evaluated and for a degree $d$, there exists a minimax polynomial approximating $F = f \circ \arcsin$, it often happens that it is not feasible to search for this polynomial with the Remez algorithm, or that for the required precision the polynomial would need to be of a degree $d$ that is too high to be depth-saving. Intuitively, the more regular a target function looks, the easier it can be approximated. In practice, we can expect all the target functions $f$ to be nice enough such that their composition with the arcsin function $F$ is close to a low-degree polynomial. We could instead apply some techniques so as to make the target function for approximation somehow more regular.

Now we introduce a delicate technique when combining a function $f$ into the composition of arcsin following the triangle functions. Recall that in the first stage of evaluating modular reduction in a bootstrapping process, it is almost free to produce the evaluation of both the sine functions and the cosine functions, and notice that the sine functions (and their compositions with arcsin) are odd functions, while the cosine functions and their compositions with arcsin are even functions. This inspires us to split the function $f(x)$ into an odd part $f_{\text{odd}}(x)$ and an even part $f_{\text{even}}(x)$ such that $f(x) = f_{\text{odd}}(x) + f_{\text{even}}(x)$, and then approximate $F_{\text{odd}} := f_{\text{odd}} \circ \arcsin$ and $F_{\text{even}} := \mathsf{HF}(f_{\text{even}}) \circ \arcsin$ independently, where $\mathsf{HF}(\cdot)$ represents the flipping the left half of the function graph, and details are explained below. In this way, we can fully make use of the output pair of triangle functions, where the target functions are always odd functions, which process some symmetry, and it is plausible to believe that they are easier to approximate to some extent.

Indeed, each real function $f$ defined over $\mathbb{R}$ (or a symmetric interval $[-a, a] \subset \mathbb{R}$) can be split into an odd function $f_{\text{odd}}$ and an even function $f_{\text{even}}$ such that $f$ is the sum of them:

$$\begin{cases} f_{\text{odd}} = \frac{1}{2}(f(x) - f(-x)), \\ f_{\text{even}} = \frac{1}{2}(f(x) + f(-x)). \end{cases} \tag{2}$$

Now we define

$$\mathsf{HF}(f) := x \mapsto \begin{cases} f(x), x \geq 0; \\ -f(x), x < 0. \end{cases}$$

This turns an even function into an odd one (w.l.o.g., we can assume that even function maps 0 to 0). Notice that when evaluating a cosine function followed by a scaled arcsin function, our goal is to create a periodic function which is $|\cdot|$ around 0, and for the even function $f_{\text{even}}(\cdot)$ we have $\mathsf{HF}(f_{\text{even}}) \circ \mathsf{abs} = f_{\text{even}}$. This is the reason that we flip half of $f_{\text{even}}$ before approximating its composition with arcsin. We list the improved workflow of functional bootstrapping based on the above ideas.

**Algorithm 6** Improved Workflow of CKKS Functional Bootstrapping with Target Function

```
    Input: $\mathsf{ct} = \mathsf{Enc}_{\mathsf{sk}}(\mathbf{m}) \in \mathcal{R}_q^2$, polynomials approximating $f_{\text{odd}} \circ \frac{1}{2\pi}\arcsin$ and
    $\mathsf{HF}(f_{\text{even}}) \circ \frac{1}{2\pi}\arcsin$, respectively.
    Output: $\mathsf{ct}_{\text{out}} \in \mathcal{R}_Q^2$.
1:  $\mathsf{ct}_1 \leftarrow \mathsf{CoeffsToSlots} \circ \mathsf{ModRaise} \circ \mathsf{SlotsToCoeffs}(\mathsf{ct})$
2:  $\mathsf{ct}_2 \leftarrow 1/2 \cdot (\mathsf{conj}(\mathsf{ct}_1) + \mathsf{ct}_1)$
3:  $\mathsf{ct}_3 \leftarrow \mathsf{EvalSine}(\mathsf{ct}_2),\ \mathsf{ct}_4 \leftarrow \mathsf{EvalCosine}(\mathsf{ct}_2)$
4:  $\mathsf{ct}_{\text{out}} \leftarrow \mathsf{EvalFoddArcsin}(\mathsf{ct}_3) + \mathsf{EvalFevenArcsin}(\mathsf{ct}_4)$
5:  return $\mathsf{ct}_{\text{out}}$
```

For some functions such as $f(x) = \mathsf{ReLU}(x)$, we don't even bother to apply the Remez algorithm, and can easily find that $\mathsf{ReLU}(x) = f_{\text{odd}}(x) + f_{\text{even}}(x)$ where $f_{\text{odd}}(x) = \frac{1}{2}x$, $f_{\text{even}}(x) = \frac{1}{2} \cdot |x|$. Within the range $(-1/4, 1/4)$, the two functions are just linearly transformed $\arcsin \circ \sin(2\pi \bullet)$ and $\arcsin \circ \cos(2\pi \bullet)$ (in other words, we have already found the minimax polynomials, which are degree-1 polynomials, and the approximation is just exact). Formally, let $x := k + m/q_0, k \in \mathbb{Z}, -1/4 < x = m/q_0 \bmod q_0 < 1/4$, we have

$$\mathsf{ReLU}(m) = q_0 \mathsf{ReLU}(x \bmod q_0).$$

Then, $\mathsf{ReLU}(x \bmod q_0)$ being

$$\frac{\pi}{4}\arcsin(\sin(2\pi x)) - \frac{\pi}{8}\arcsin(\cos(4\pi x)) + \frac{1}{16},$$

where $x \in (-1/4, 1/4) + \mathbb{Z}$.

In general, it still holds that for a function $f$ which is neither odd nor even, splitting $f$ into odd and even components and approximating each component can produce a more precise approximation. Since an odd or even function is symmetric around the origin, this means the approximation range for the Remez algorithm is reduced by half. For an odd function $f_{\text{odd}}(x)$ or an even function $f_{\text{even}}(x)$ defined on $(-B, B) \subset \mathbb{R}$, we only need to approximate the function on the range $(0, B)$. As a smaller range generally implies a better approximation, our method improves the precision of evaluating the composition of $f$ with the bootstrapping process, and is an improved method for CKKS functional bootstrapping.

*A technique on evaluating cosine for almost free.* In practice, there are optimized methods for evaluating the sine function for CKKS bootstrapping. For example, in the code base [69] we are using, for obtaining $\sin(2\pi x)$, a cosine function $f(x) = \cos(\frac{1}{2^r} \cdot 2\pi \cdot (x - \frac{1}{4}))$ is evaluated first, then applying the double angle formula of cosine function

$$\cos 2\theta = 2\cos^2\theta - 1$$

for $r$ times will produce $\cos(2\pi(x - \frac{1}{4})) = \sin(2\pi x)$. Now, as what we require for the absolute value function is the cosine function $-\cos(4\pi x)$, in this case, we can obtain this required cosine function *with only one extra level consumption.* Indeed, by evaluating the double-angle formula for one more time, we obtain

$$\cos\left(\frac{1}{2^r} \cdot 2\pi \cdot \left(x - \frac{1}{4}\right) \cdot 2^{r+1}\right) = \cos(4\pi x - \pi) = -\cos(4\pi x).$$

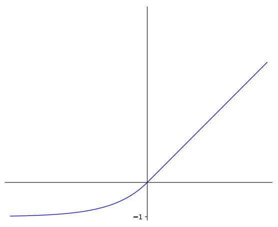
Besides the method described above, there are yet other methods for the step of evaluating sine. We discuss two of them: evaluating $y = \sin(ax), y' = \cos(ax)$ approximated by polynomial and performing the double-angle formulae iteratively: $\sin 2\theta = 2\cos\theta\sin\theta$, $\cos 2\theta = \cos^2\theta - \sin^2\theta$, or evaluating $z = \mathrm{e}^{\mathrm{i}ax}$ approximated by polynomial and squaring. With these methods, since we are evaluating sine and cosine simultaneously, for obtaining $-\cos(4\pi x)$, we just need to run the iteration for one more round.

Above all, in order to obtain the cosine component for evaluating $f(x) = \mathsf{ReLU}(x)$ in the functional bootstrapping manner, it costs only minimal extra overhead (almost none), no matter what method is adopted in the step of sine evaluation.

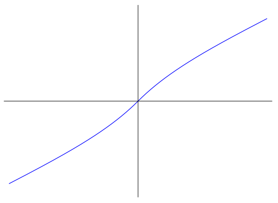
*Activation functions beyond* ReLU*.* We take several other activation functions commonly used in modern neural networks as examples, to demonstrate the possibilities for more general CKKS functional bootstrapping beyond ReLU.

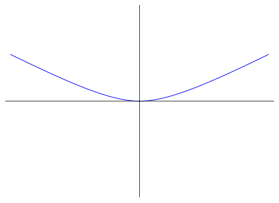
- The tanh function. It is easy to see that the function $\tanh(x) = (\mathrm{e}^x - \mathrm{e}^{-x})/(\mathrm{e}^x + \mathrm{e}^{-x})$ is an odd function. Therefore, we can simply apply the method of Section 4.1, where we produce an approximation polynomial $P(x)$ such that $P \approx \tanh \circ \arcsin$, then for evaluating tanh along with the bootstrapping process we run Alg. 5 where in step 4 we evaluate $P(x)$ homomorphically.
- GELU. The GELU function is defined as the cumulative distribution function for Gaussian distribution $\Phi(x)$ multiplied by $x$. On the other hand, since $\Phi(x)$ is not an elementary function, in practice GELU is applied in an approximate form: $\mathsf{GELU}(x) = \frac{1}{2}x(1 + \tanh(\sqrt{2/\pi} \cdot (x + 0.044715 \cdot x^3)))$. From this expression we can immediately obtain $\mathsf{GELU}_{\text{odd}}(x) = \frac{1}{2}x$, $\mathsf{GELU}_{\text{even}}(x) = \frac{1}{2}x\tanh(\sqrt{2/\pi} \cdot (x + 0.044715 \cdot x^3))$, with which the procedures in Section 4.2 can be applied.
- ELU. For the case of $\mathsf{ELU}(x) = \begin{cases} x, & x \geq 0 \\ \mathrm{e}^x - 1, & x < 0 \end{cases}$, we also suggest applying the improved functional bootstrapping method in Section 4.2, and for the

splitting of the ELU function, we apply equation (2). The odd and even parts are shown in the graphs below. Notice that there is no cusp in the graph of $\mathsf{ELU}_{\text{even}}(x)$, different from the case of the absolute value function $|x|$ we encountered in evaluating ReLU. We believe that this may relax the challenge of approximation on ELU compared to ReLU.



(a) $\mathsf{ELU}(x)$


(b) $\mathsf{ELU}_{\text{odd}}(x)$


(c) $\mathsf{ELU}_{\text{even}}(x)$

The functional bootstrapping methods of the above functions could be explored in future work.

## 5 Evaluation

We implemented RBOOT and compared against Lee et al. [44], using their open-sourced library FHE-MP-CNN [69]. All experiments were performed on an Intel® Xeon Platinum 2.50 GHz server with 128 GB RAM. We run the experiments of homomorphic inference in single thread, as the settings in [44,69].

*Experimental parameters.* For the CKKS instantiation, we set $N = 2^{16}$ and hamming weight of the secret key $h = 192$ as [69]. We list the complete moduli setting in Table 2.

Table 2: Experimental parameters. For modulus consumption in the EvalMod phase, our functional bootstrapping integrates the ReLU function. $\log(p)$ denotes the modulus for key switching. The base of the logarithm is two.

| | | $\log(q)$ | | | | $\log(p)$ |
|---|---|---|---|---|---|---|
| | Base | StC | CtS | EvalMod | rest | |
| [44,69] | 51 | $46 \times 3$ | $51 \times 3$ | $51 \times 8$ (eval. mod.) | $46 \times 16$ (ReLU & conv.) | 51 |
| Ours | 51 | $46 \times 3$ | $51 \times 3$ | $51 \times 16$ (eval. mod. w/ ReLU) | $46 \times 2$ (only conv.) | $51 \times 5$ |

Interestingly, reducing the level consumption not only enables RBOOT to use a smaller ciphertext modulus, which already provides significant speedup, but also allows us to leverage optimizations of [32,33] that introduce more special moduli to accelerate key switching. In the table it can be seen we use 5 special primes of 51 bits (denoted as $\log(p)$ in Table 2) in our parameter setting. In contrast, prior ReLU approximations typically require high multiplication depths and almost exhaust the ciphertext modulus capacity, limiting their benefit from this optimization.

*On the suitability of [44] as a baseline.* We note that more efficient alternatives to [44] exist, such as [42,23,38,26]. And [44] did not integrate some of the recent advances in CKKS such as [8,18]. However, our optimization of activation functions is orthogonal to these factors and can be applied in conjunction with them. We choose [44] because of its wide usage in related works, and its well-functioned codebase.

## 5.1 Functional Bootstrapping

We run the process of RBOOT with common encoding formats, and compare the runtime with that of a normal bootstrapping followed by an independent ReLU evaluation in [69]. For a clearer comparison, we have excluded the two moduli for convolution in Table 2. In terms of message encoding, given our setting of $N = 2^{16}$, we call the case of encoding $2^{15}$ (resp., $2^{14}$, $2^{13}$, $2^{12}$) slots within a plaintext polynomial as full (resp., 1/2-, 1/4-, 1/8-sparse).

Table 3: Runtime (s) for bootstrapping and/with ReLU.

| Message encoding | [44,69] | Ours |
|---|---|---|
| Full | 103.2 | 48.7 |
| 1/2-sparse | 86.1 | 40.4 |
| 1/4-sparse | 75.8 | 34.6 |
| 1/8-sparse | 72.6 | 33.1 |

From Table 3 it can be seen that our method shows a speedup of up to 2.19× compared to existing methods.

The acceleration is due to the reduction in the amount of level consumption of bootstrapping. Below we compare the level consumption of RBOOT against a normal bootstrapping followed by an independent evaluation of ReLU using the implementation of [44,69] for varying precision bits $\alpha$.

Table 4: Level consumption by bootstrapping and/with ReLU.

| $\alpha$ | [45] | [44,69] | Ours | Difference |
|---|---|---|---|---|
| $\geq 8.0$ | 26 | / | 19 | -27% |
| $\geq 10.0$ | 29 | / | 20 | -31% |
| $\geq 12.0$ | 30 | / | 21 | -30% |
| $\geq 14.0$ | 34 | 28 | 22 | -35%, -21% |

## 5.2 Neural Network Inference

*End-to-end homomorphic inference.* We evaluated the costs of end-to-end homomorphic inference with RBOOT on four ResNet models (ResNet20/32/44/56), and compared them with baseline [44,69]. As shown in Table 5, RBOOT accelerates the inference by 2.77× on average. The advantages are greater than those in Table 3 due to the fact that the end-to-end experiments here include the two 46-bit moduli for convolution, which increase the depths and amplify our gains.

Table 5: Inference latency (s) for one CIFAR-10 image.

| Models | [44,69] | Ours |
|---|---|---|
| ResNet20 | 2482 | 917 |
| ResNet32 | 4121 | 1490 |
| ResNet44 | 5733 | 2071 |
| ResNet56 | 7469 | 2664 |

In the aspect of memory efficiency (Table 6), the strength of RBOOT is more clear, with 81.7% decrease in peak memory occupation. This can be attributed to both the reduction of level consumption and the adoption of multiple special primes. (Indeed, with $k$ special primes, the key switching key for ciphertexts of level up to $\ell$ consists of $2 \cdot \lceil \ell/k \rceil \cdot (\ell + 1 + k)$ $\mathcal{R}_{q_i}$-elements, which will be greatly reduced as $k$ increases.) The significant reduction in memory use makes our method quite friendly to hardware implementation.

Table 6: Peak Memory Occupation (GB).

| Models | [44,69] | Ours |
|---|---|---|
| ResNet20/32/44/56 | 288.0 | 52.6 |

*On ImageNet-scale models.* While we haven't found any open-source implementations for FHE-based ImageNet inference, we performed a rough analysis based on data reported by [38]. We estimate that our optimizations would also achieve a similar half reduction in runtime. Please see Appendix A for details.

*On homomorphic inference accuracy.* We evaluated the first 100 CIFAR-10 samples provided from [69] with the RBOOT method, and the predictions are identical to those with [69]. See Appendix B for details.

*Accuracy simulations.* We test the impact of accuracy on plaintext by replacing the ReLU operator with our composite functions. The experiments are performed on ResNet models on CIFAR-10 or ImageNet dataset. For the CIFAR-10 dataset, since the models used by [44,69] are not publicly available at the time of writing, we use the models of [15] in which the ResNet models are identical to those of [44,69]. For the ImageNet dataset, we use the ResNet models provided by PyTorch [62]. The experiments replace the standard ReLU operator with a custom defined function $f(x) = 1/2 \cdot x + 1/2 \cdot \text{approx\_arcsin}(-\cos(\pi x))$, where $\text{approx\_arcsin}(\cdot)$ is the minimax polynomial for arcsin we produced in Section 3. Notice that since 8 bits' precision is believed to be enough for an accurate inference of a convolutional neural network [61], in our simulations we use Param-I in Table 1 – the setting with the *lowest degree output* – which provides an approximation with error up to $2^{-8.78}$. The results below show only minimal fluctuation on top-1 accuracy compared to the baseline model with standard ReLU [62]. Considering we only use our roughest approximation in these simulations, the results give us confidence to generalize the RBOOT method to models on ImageNet dataset in possible future works.

Table 7: Top-1 accuracy with standard ReLU and with our approximation.

| Models | With standard ReLU | With our approximation |
|---|---|---|
| ResNet20 | 92.60% | 92.59% |
| ResNet32 | 93.53% | 93.42% |
| ResNet44 | 94.01% | 94.03% |
| ResNet56 | 94.37% | 94.32% |
| ResNet18 (ImageNet) | 67.28% | 67.14% |
| ResNet34 (ImageNet) | 71.32% | 71.29% |
| ResNet50 (ImageNet) | 74.55% | 74.55% |

## 6 Conclusion

We introduce a new framework for performing functional bootstrapping with the homomorphic encryption scheme CKKS, which is practical and efficient. Based on the framework we are able to evaluate the ReLU operator from machine learning along with the process of CKKS bootstrapping. Our experiments show great advantages on the evaluation of ReLU in both running time and precision, compared to the state-of-the-art method.

## References

1. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. Homomorphic encryption standard. In *Protecting privacy through homomorphic encryption*, pages 31–62. Springer, 2022.
2. Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using CKKS. Cryptology ePrint Archive, Paper 2024/1623, 2024.
3. Wei Ao and Vishnu Naresh Boddeti. {AutoFHE}: Automated adaption of {CNNs} for efficient evaluation over {FHE}. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2173–2190, 2024.
4. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with ckks. page 94–123, Berlin, Heidelberg, 2024. Springer-Verlag.
5. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with ckks. In *Advances in Cryptology – ASIACRYPT 2024: 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9–13, 2024, Proceedings, Part I*, page 330–360, Berlin, Heidelberg, 2024. Springer-Verlag.
6. Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (t)fhe. *Journal of Cryptology*, 36(3):28, 2023.
7. Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617, Cham, 2021. Springer International Publishing.
8. Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 587–617. Springer, 2021.
9. Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.
10. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III*, page 483–512, Berlin, Heidelberg, 2018. Springer-Verlag.

11. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.
12. Michele Carminati, Luca Santini, Mario Polino, and Stefano Zanero. Evasion attacks against banking fraud detection systems. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 285–300, San Sebastian, October 2020. USENIX Association.
13. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 34–54, Cham, 2019. Springer International Publishing.
14. Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, and Yang Zhang. FACE-AUDITOR: Data auditing in facial recognition systems. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 7195–7212, Anaheim, CA, August 2023. USENIX Association.
15. Yaofo Chen. Pytorch cifar models. Online: `https://github.com/chenyaofo/pytorch-cifar-models/`, 2021.
16. Jung Cheon, Han Kyoohyung, Andrey Kim, Miran Kim, and Yongsoo Song. *Bootstrapping for Approximate Homomorphic Encryption*, pages 360–384. 03 2018.
17. Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Damien Stehlé. Homomorphic multiple precision multiplication for ckks and reduced modulus consumption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 696–710, 2023.
18. Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, Jaehyung Kim, Seonghak Kim, Johannes Mono, and Taeyeong Noh. Grafting: Decoupled scale factors and modulus in rns-ckks. *Cryptology ePrint Archive*, 2024.
19. Jung Hee Cheon, Minsik Kang, Taeseong Kim, Junyoung Jung, and Yongdong Yeo. Batch inference on deep convolutional neural networks with fully homomorphic encryption using channel-by-channel convolutions. *IEEE Trans. Dependable Secur. Comput.*, 22(2):1674–1685, 2025.
20. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.
21. Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology – ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II*, page 221–256, Berlin, Heidelberg, 2020. Springer-Verlag.
22. Jung Hee Cheon, Jihwan Kim, and Yongdong Yeo. Overmodraise: Reducing modulus consumption of ckks bootstrapping. *Cryptology ePrint Archive*, 2025.
23. Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. {DaCapo}: Automatic bootstrapping management for efficient fully homomorphic encryption. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6993–7010, 2024.
24. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

25. Leo de Castro, Rashmi Agrawal, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, Chiraag Juvekar, and Ajay Joshi. Does fully homomorphic encryption need compute acceleration? *arXiv preprint arXiv:2112.06396*, 2021.
26. Austin Ebel, Karthik Garimella, and Brandon Reagen. 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*, pages 734–749, 2025.
27. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.
28. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC 2009*, pages 169–178, 2009.
29. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
30. Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016.
31. Antonio Guimarães, Hilder VL Pereira, and Barry Van Leeuwen. Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–35. Springer, 2023.
32. Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. *Cryptology ePrint Archive*, 2020.
33. Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 364–390. Springer, 2020.
34. Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Topics in Cryptology – CT-RSA 2020: The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24–28, 2020, Proceedings*, page 364–390, Berlin, Heidelberg, 2020. Springer-Verlag.
35. Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure {Two-Party} deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, 2022.
36. Takumi Ishiyama, Takuya Suzuki, and Hayato Yamana. Highly accurate CNN inference using approximate activation functions over homomorphic encryption. *CoRR*, abs/2009.03727, 2020.
37. Mohd Javaid, Abid Haleem, Ravi Pratap Singh, Rajiv Suman, and Shanay Rab. Significance of machine learning in healthcare: Features, pillars and applications. *International Journal of Intelligent Networks*, 3:58–73, 2022.
38. Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. 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*, pages 4361–4375, 2024.

39. Charanjit S. Jutla and Nathan Manohar. Sine series approximation of the mod function for bootstrapping of approximate he. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part I*, page 491–520, Berlin, Heidelberg, 2022. Springer-Verlag.
40. Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*, pages 1651–1669, 2018.
41. Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for rlwe-based homomorphic encryption. *IEEE Transactions on Computers*, 73(1):86–96, 2024.
42. Dongwoo Kim and Cyril Guyot. Optimized privacy-preserving cnn inference with fully homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 18:2175–2187, 2023.
43. Eunsang Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. Optimization of homomorphic comparison algorithm on RNS-CKKS scheme. Cryptology ePrint Archive, Paper 2021/1215, 2021.
44. Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422. PMLR, 2022.
45. Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. Minimax approximation of sign function by composite polynomial for homomorphic comparison. Cryptology ePrint Archive, Paper 2020/834, 2020.
46. Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I*, page 618–647, Berlin, Heidelberg, 2021. Springer-Verlag.
47. Feng-Hao Liu and Han Wang. Batch bootstrapping i: A new framework for simd bootstrapping in polynomial modulus. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, page 321–352, Berlin, Heidelberg, 2023. Springer-Verlag.
48. Feng-Hao Liu and Han Wang. Batch bootstrapping ii: Bootstrapping in polynomial modulus only requires $\tilde{O}(1)$ fhe multiplications in amortization. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, page 353–384, Berlin, Heidelberg, 2023. Springer-Verlag.
49. Tzu-Li Liu, Yu-Te Ku, Ming-Chien Ho, Feng-Hao Liu, Ming-Ching Chang, Chih-Fan Hsu, Wei-Chao Chen, and Shih-Hao Hung. An efficient ckks-fhew/tfhe hybrid encrypted inference framework. In *European Symposium on Research in Computer Security*, pages 535–551. Springer, 2023.
50. Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using fhew/tfhe bootstrapping. In *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of*

*Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*, page 130–160, Berlin, Heidelberg, 2022. Springer-Verlag.

51. Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with o˜(1) polynomial multiplications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 101–132. Springer, 2023.
52. Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073. IEEE, 2021.
53. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6), November 2013.
54. Daniele Micciancio and Jessica Sorrell. Ring packing and amortized fhew bootstrapping. *Cryptology ePrint Archive*, 2018.
55. The mpmath development team. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.3.0)*, 2023. `http://mpmath.org/`.
56. Kevin Nam, Youyeon Joo, Seungjin Ha, and Yunheung Paek. Slothe: Lazy approximation of non-arithmetic neural network functions over encrypted data. In *34rd USENIX Security Symposium*, 2025.
57. Chao Niu, Zhicong Huang, Zhaomin Yang, Yi Chen, Liang Kong, Cheng Hong, and Tao Wei. XBOOT: Free-XOR gates for CKKS with applications to transciphering. Cryptology ePrint Archive, Paper 2025/074, 2025.
58. Jaiyoung Park, Donghwan Kim, Jongmin Kim, Sangpyo Kim, Wonkyung Jung, Jung Hee Cheon, and Jung Ho Ahn. Toward practical privacy-preserving convolutional neural networks exploiting fully homomorphic encryption. *arXiv preprint arXiv:2310.16530*, 2023.
59. Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. Aespa: Accuracy preserving low-degree polynomial activation for fast private inference. *arXiv preprint arXiv:2201.06699*, 2022.
60. Michael J. D. Powell. *Approximation theory and methods.* Cambridge University Press, 1981.
61. PyTorch. Quantization – pytorch 2.8 documentation. Online: `https://docs.pytorch.org/docs/stable/quantization.html`, 2019.
62. PyTorch. *ResNet - PyTorch*, 2025. https://pytorch.org/hub/pytorch_vision_resnet/.
63. Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 325–342, 2020.
64. Eugene Y. Remez. Sur la détermination des polynômes d'approximation de degré donnée. *Comm. Soc. Math. Kharkov*, 10(196):41–63, 1934.
65. C. Ridders. A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems*, 26(11):979–980, 1979.
66. Lorenzo Rovida and Alberto Leporati. Encrypted image classification with low memory footprint using fully homomorphic encryption. *International journal of neural systems*, 34(05):2450025, 2024.
67. *SageMath - Open-Source Mathematical Software System*, 2025. `http://sagemath.org/`.
68. Shawn Shan, Emily Wenger, Jiayun Zhang, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Fawkes: Protecting privacy against unauthorized deep learning models. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1589–1604. USENIX Association, August 2020.

69. snu ccl. Fhe-mp-cnn. Online: `https://github.com/snu-ccl/FHE-MP-CNN`, 2022.
70. Elias M. Stein and Rami Shakarchi. *Fourier analysis: An introduction.* Princeton University Press, 2006.
71. Eric W. Weisstein. Fourier series–triangle wave. From MathWorld–A Wolfram Web Resource, 2025.
72. Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. TOTA: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Paper 2021/1347, 2021.
73. Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Ximeng Liu, and Kai Chen. Sok: Fully homomorphic encryption accelerators. *ACM Computing Surveys*, 56(12):1–32, 2024.

## A Estimations on ImageNet-Targeting Models

While [44] uses a single special prime for key switching, [38] requires five, the same as ours. Without our optimizations, this reduces the post-bootstrapping multiplication depth to $\sim 10$, necessitating 2 bootstrappings per ReLU. As reported in Table 2 of [38], a single ResNet50 inference on ImageNet using an A100 GPU takes 56.08s, where ReLU and bootstrapping account for the majority of the cost (53.22s). Consequently, we estimate that our method reduces the associated runtime by approximately half. We emphasize that this is a very rough estimation, as [38] uses a 42-bit scale for ReLU compared to our 51 bits.

## B Results of Homomorphic Inference

We evaluated the first 100 Cifar-10 samples provided from [69] under encryption with the ResNet-20 model, and compared the evaluation predictions with the RBOOT method with the original implementation in [69]. Both methods produced *the same prediction* for each of the 100 samples. The predictions are recorded in the table below, which is read in row-major order. The labels on the left and the right in each cell are the ground truth and the prediction, respectively.

## C On the Effect of Slim Bootstrapping in Inference

We have compared the inference latency of our RBOOT methods with that of the original implementation [69]. Notice that [69] uses the early variant of bootstrapping (Alg. 1), while RBOOT relies necessarily on the slim variant (Alg. 2). In this work, we compared our methods with [69] as is, since it is publicly available and familiar to the community. On the other hand, although a slim variant bootstrapping is able to provide some acceleration on [69], this effect will not be essential, as shown in the table below.

Table 8: ResNet-20 inference results with [69]/with RBOOT

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $3 \to 3$ | $8 \to 8$ | $8 \to 8$ | $0 \to 0$ | $6 \to 6$ | $6 \to 6$ | $1 \to 1$ | $6 \to 6$ | $3 \to 3$ | $1 \to 1$ |
| $0 \to 0$ | $9 \to 9$ | $5 \to 5$ | $7 \to 7$ | $9 \to 9$ | $8 \to 8$ | $5 \to 5$ | $7 \to 7$ | $8 \to 8$ | $6 \to 6$ |
| $7 \to 7$ | $0 \to 0$ | $4 \to 4$ | $9 \to 9$ | $5 \to 5$ | $2 \to 2$ | $4 \to 4$ | $0 \to 0$ | $9 \to 9$ | $6 \to 6$ |
| $6 \to 6$ | $5 \to 5$ | $4 \to 4$ | $5 \to 5$ | $9 \to 9$ | $2 \to 3$ | $4 \to 4$ | $1 \to 1$ | $9 \to 9$ | $5 \to 5$ |
| $4 \to 4$ | $6 \to 6$ | $5 \to 5$ | $6 \to 6$ | $0 \to 0$ | $9 \to 9$ | $3 \to 3$ | $9 \to 9$ | $7 \to 7$ | $6 \to 6$ |
| $9 \to 9$ | $8 \to 8$ | $0 \to 2$ | $3 \to 3$ | $8 \to 8$ | $8 \to 8$ | $7 \to 7$ | $7 \to 7$ | $4 \to 4$ | $6 \to 3$ |
| $7 \to 7$ | $3 \to 3$ | $6 \to 6$ | $3 \to 3$ | $6 \to 6$ | $2 \to 2$ | $1 \to 1$ | $2 \to 2$ | $3 \to 3$ | $7 \to 7$ |
| $2 \to 2$ | $6 \to 6$ | $8 \to 8$ | $8 \to 8$ | $0 \to 0$ | $2 \to 2$ | $9 \to 9$ | $3 \to 3$ | $3 \to 3$ | $8 \to 8$ |
| $8 \to 8$ | $1 \to 9$ | $1 \to 1$ | $7 \to 7$ | $2 \to 2$ | $5 \to 5$ | $2 \to 2$ | $7 \to 7$ | $8 \to 8$ | $9 \to 9$ |
| $0 \to 0$ | $3 \to 3$ | $8 \to 8$ | $6 \to 6$ | $4 \to 4$ | $6 \to 2$ | $6 \to 6$ | $0 \to 0$ | $0 \to 0$ | $7 \to 7$ |

Table 9: Effect of slim bootstrapping on inference latency (s) of [69]

| Models | [69] | [69] modified as per Alg. 2 |
|---|---|---|
| ResNet20 | 2482 | 2186 |
| ResNet32 | 4121 | 3553 |
| ResNet44 | 5733 | 4961 |
| ResNet56 | 7469 | 6335 |

It can be seen that the latencies of inference with [69] modified are still close to those with the original implementation. Even if our method (Table 5) is compared with [69] migrated to the slim variant of bootstrapping, it will still show a 2.38× advantage in latency. Based on the above consideration, we chose the unmodified [69] as our baseline.

## D Comparison on Realization of Algorithm 3

Compared to the implementation in [46,69] which also realizes the Remez algorithm yet with 1000-bit multi-precision arithmetic, our implementation only uses 100∼300-bit precision of data type, which can be attributed to our dedication in tuning the implementation of the algorithm. Furthermore, even with arithmetic with 1000 bits, the implementation of Alg. 3 still fail to produce a degree 64 minimax polynomial.

Table 10: Comparison on realization of Remez Algorithm

| Impl. of Alg. 3 | Precision of multi-prec. arith. | Maximum degree $d$ supported |
|---|---|---|
| [46,69] | 1000 | $\leq 63$ |
| Ours | 100 | $\geq 31$ |
| | 150 | 63 |
| | 310 | 127 |

## E Effects of Initializing Algorithm 3 with Fourier Coefficients

Below we compare the processes of convergence with initialization from Chebyshev nodes and that from coefficients of Fourier series of the triangle wave functions. The two figures correspond to $d = 31$ and $d = 63$, respectively. (Note that initialization from Fourier coefficients means that we skip steps 4-6 in the first iteration, hence there is no $E^{(1)}$ in this case.) From the figures it can be seen that initializing with the Fourier coefficients does make the algorithm converge faster than that with Chebyshev nodes.

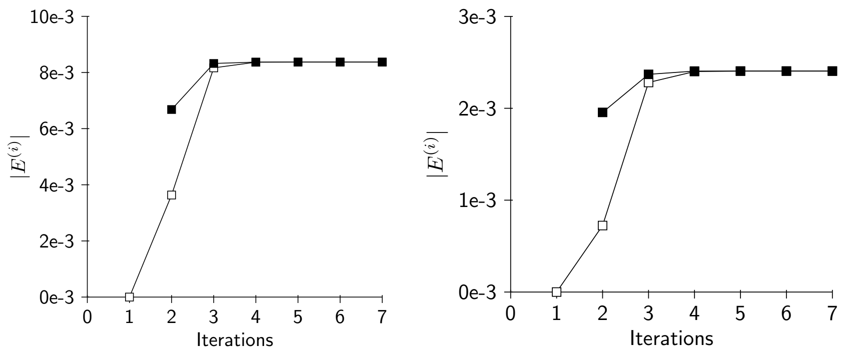


Fig. 6: Convergence processes with Fourier coefficients (solid) and with Chebyshev nodes, for $d = 31$ and $d = 63$, respectively

Although intuitively, it seems that the new initialization method only makes a difference in the first few iterations, these iterations are rather vital: in the beginning the candidate polynomials (and the error functions) can behave badly. The polynomials can be extremely far away from the target function in some area, leading to the zeros of error function highly dense, and thus leading the process of iteration more unstable and requiring much larger numerical precision and finer root-finding process. Therefore, a better initialization with a well-behaving candidate polynomial improves the stability of the whole process.

## F On the Relation between Minimax Polynomials and Certain Fourier Series

It is easy to see that for a Fourier series $\mathcal{F}(x) = a_0 + \sum_{i=1}^{\infty} a_i \cos(\pi x/L) + \sum_{i=1}^{\infty} b_i \sin(\pi x/L)$, if $a_i = 0$ for all odd $i$ and $b_i = 0$ for all even $i$, then $\mathcal{F}(x)$ can be turned into a combination of Chebyshev polynomials of $\cos(\pi x/L)$ or $\sin(\pi x/L)$:

$$\mathcal{F}(x) = a_0 + \sum_{i=0}^{\infty}(-1)^i a_{2i} T_{2i}\left(\cos\left(\frac{\pi x}{L}\right)\right) + \sum_{i=0}^{\infty}(-1)^i b_{2i+1} T_{2i+1}\left(\sin\left(\frac{\pi x}{L}\right)\right).$$

Truncate each of the infinite sum at after index $d$, and denote the resulting polynomial of triangle functions as $\mathcal{F}_d$, then $\mathcal{F}_d(x)$ can be expressed similarly:

$$\mathcal{F}_d(x) = a_0 + \sum_{i=0}^{\lfloor d/2 \rfloor}(-1)^i a_{2i} T_{2i}\left(\cos\left(\frac{\pi x}{L}\right)\right) + \sum_{i=0}^{\lfloor (d-1)/2 \rfloor}(-1)^i b_{2i+1} T_{2i+1}\left(\sin\left(\frac{\pi x}{L}\right)\right).$$

For simplicity, we call the polynomial

$$P_d(x', x) = a_0 + \sum_{i=0}^{\lfloor d/2 \rfloor}(-1)^i a_{2i} T_{2i}(x') + \sum_{i=0}^{\lfloor (d-1)/2 \rfloor}(-1)^i b_{2i+1} T_{2i+1}(x)$$

as the original function $\mathcal{F}(x)$'s corresponding degree-$d$ Fourier polynomial. If the function $\mathcal{F}(x)$ is odd or even itself, then the corresponding Fourier polynomial is a univariate one of $x$ or of $x'$, respectively.

For our case of approximating the function $\frac{1}{2\pi}\arcsin(\cdot)$, first notice that the composition of a sine function and its corresponding arcsin function is a triangle wave function [71]. Let $2L, A$ be the period and the amplitude of a triangle wave function which is denoted as $\mathcal{F}_{\text{asin}}(x; L, A)$, then its Fourier series is

$$\mathcal{F}_{\text{asin}}(x; L, A) = A \sum_{i=0}^{\infty}(-1)^i \frac{8}{\pi^2(2i+1)^2} \sin((2i+1)\pi x/L). \tag{3}$$

Now let $x = \sin(\pi y/L)$ and let $2L = 1, A = \frac{1}{4}$, then we can turn the expression of $\mathcal{F}_{\text{asin}}(y; L, A)$ into

$$\frac{1}{2\pi}\arcsin(x) = \frac{1}{4}\sum_{i=0}^{\infty} \frac{8}{\pi^2(2i+1)^2} T_{2i+1}(x). \tag{4}$$

Abusing notations, we define

$$P_d(x) \coloneqq \frac{1}{4} \sum_{i=0}^{(d-1)/2} \frac{8}{\pi^2(2i+1)^2} T_{2i+1}(x).$$

Denote the minimax polynomial of our target function as $p_d(x)$, by Theorem 3 we have the following empirical heuristic:

**Lemma 2 (Heuristic).** *Over* $[-1+\epsilon, 1-\epsilon]$ *for* $\epsilon \approx 10^{-4}$ *and for all* $d \in \mathbb{Z}$*, we have* $\|p_d(x) - \frac{1}{2\pi}\arcsin(x)\|_{\infty,[-1,1]} \leq \exp(-d^{0.455} - 5.465)$.

Notice that $\frac{1}{2\pi}\arcsin(x) = P_\infty(x)$. Therefore we have a bound on $\|p_d(x) - P_\infty(x)\|_\infty$ over $[-1,1]$, which is also a bound of $\|p_d(\sin(2\pi x)) - \mathcal{F}_{\text{asin}}(x; \frac{1}{2}, \frac{1}{4})\|_\infty$. This in turn bounds $\|p_d(\sin(2\pi x)) - \mathcal{F}_{\text{asin}}(x; \frac{1}{2}, \frac{1}{4})\|_2$ over the same range. Now since over any valid interval, it holds

$$\begin{aligned} &\|p_d(\sin(2\pi x)) - P_d(\sin(2\pi x))\|_2 \\ &\leq \|p_d(\sin(2\pi x)) - \mathcal{F}_{\text{asin}}(x; \tfrac{1}{2}, \tfrac{1}{4})\|_2 + \|P_d(\sin(2\pi x)) - \mathcal{F}_{\text{asin}}(x; \tfrac{1}{2}, \tfrac{1}{4})\|_2, \end{aligned}$$

by estimating the last norm over $[-1,1]$ we are able to obtain a bound on $\|p_d(\sin(2\pi x)) - P_d(\sin(2\pi x))\|_2$, and hence able to bound the difference of coefficients of the two polynomials (over the Chebyshev basis), which would thus support the claim of Lemma 1.