Approximate Methods for the Computation of Step Functions in Homomorphic Encryption

Tairong Huang[1], Shihe Ma[2], Anyu Wang[1,3,4(✉)], and Xiaoyun Wang[1,3,4,5,6]

[1] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China, htr19@mails.tsinghua.edu.cn,anyuwang,xiaoyunwang@tsinghua.edu.cn
[2] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China, msh21@mails.tsinghua.edu.cn
[3] Zhongguancun Laboratory, Beijing, China
[4] National Financial Cryptography Research Center, Beijing, China
[5] Shandong Institute of Blockchain, Jinan, China
[6] Key Laboratory of Cryptologic Technology and Information Security (Ministry of Education), School of Cyber Science and Technology, Shandong University, China

Abstract. The computation of step functions over encrypted data is an essential issue in homomorphic encryption due to its fundamental application in privacy-preserving computing. However, an effective method for homomorphically computing general step functions remains elusive in cryptography. This paper proposes two polynomial approximation methods for general step functions to tackle this problem. The first method leverages the fact that any step function can be expressed as a linear combination of shifted sign functions. This connection enables the homomorphic evaluation of any step function using known polynomial approximations of the sign function. The second method boosts computational efficiency by employing a composite polynomial approximation strategy. We present a systematic approach to construct a composite polynomial $f_k \circ f_{k-1} \circ \cdots \circ f_1$ that increasingly approximates the step function as k increases. This method utilizes an adaptive linear programming approach that we developed to optimize the approximation effect of f_i while maintaining the degree and coefficients bounded. We demonstrate the effectiveness of these two methods by applying them to typical step functions such as the round function and encrypted data bucketing, implemented in the HEAAN homomorphic encryption library. Experimental results validate that our methods can effectively address the homomorphic computation of step functions.

Keywords: Step function · Homomorphic encryption · CKKS · Polynomial approximation · Round function · Encrypted data bucketing

1 Introduction

Fully homomorphic encryption (FHE) is a powerful cryptographic primitive which enables performing any computation on encrypted data without having access to the secret key. Since Gentry developed the first FHE scheme [19], various FHE schemes have been proposed following Gentry's blueprint [35,20]. According to the type of computations to which they are suitable, these FHEs can be divided into three categories. The first category contains GSW [21] and its improvements FHEW/TFHE [15,13], which are ideal for evaluating Boolean circuits since they *bit-wisely* encrypt the input data. The second category contains BGV/FV [6,7,18], which pack their input data into finite fields or finite rings and are frequently used to evaluate integer arithmetic with a fixed modulus. The CKKS scheme [10,9,8], which forms the third category, can process fixed-point input numbers and supports approximate computations over complex and real numbers. In BGV/FV and CKKS, the input data is *word-wisely* encrypted. The operations on these numbers can be performed in a Single Instruction Multiple Data (SIMD) fashion [34], i.e, encrypted numbers are packed in slots such that the operations performed on a single ciphertext are automatically performed on each slot in parallel. Due to the SIMD property, these word-wise FHEs are very efficient in homomorphic addition, multiplication, and, more generally, polynomial evaluation. However, the effective evaluation of non-polynomial functions in word-wise FHEs presents a challenge and has recently garnered significant attention.

For CKKS, a natural way to tackle this issue involves approximating non-polynomial functions with polynomials. This approach has been successfully applied to evaluate a range of non-polynomial functions, such as logistic regression [22,24], inverse [12], square root [12,30], etc [26,27,23]. Nevertheless, the homomorphic evaluation of discontinuous functions, such as the sign function and the step function, presents a significantly greater challenge. These functions have attracted considerable attention due to their importance in various practical applications, including privacy-preserving machine learning [1,29,5]. Several methods have been proposed for the homomorphic computation of the sign function. For instance, polynomial iteration algorithms were introduced in [12], offering an approximation with an exponentially small error rate. In [11], Cheon et al. re-investigated the polynomial approximation of the sign function and proposed a composite polynomial approach to address this issue, which was proven to be asymptotically optimal. Subsequently, Lee et al. [25] explored the composition of minimax approximate polynomials of the sign function and proposed a practically optimal sign function approximation. Despite these advancements, these methods are not directly applicable to general step functions, and an effective method for homomorphically computing step functions remains to be devised.

1.1 Our Results

This paper delves into the polynomial approximation problem for general step functions. Let $\kappa(x)$ be a step function on the interval $[a, b]$ such that

$$\kappa(x) = y_i \text{ for } x \in (a_{i-1}, a_i), 1 \leq i \leq n,$$

where $a = a_0 < a_1 < \cdots < a_n = b$. The main contribution of this paper is two systematical methods for solving the polynomial approximation problem of $\kappa(x)$.

Method I (SgnToStep). This method utilizes the fact that a step function $\kappa(x)$ can be expressed as a linear combination of shifted sign functions, i.e.,

$$\kappa(x) = c_1\mathtt{sgn}(x - a_1) + \cdots + c_{n-1}\mathtt{sgn}(x - a_{n-1}) + c_n,$$

where c_i's are real constants defined in Lemma 1, and sgn is the sign function defined in Section 2. Based on the polynomial approximations of $\mathtt{sgn}(x)$ as provided in [11,12,25], we show that this connection can be used to generate polynomial approximations for any step function $\kappa(x)$. We present a comprehensive analysis of the evaluation complexity and the required homomorphic multiplicative depth of this method. Moreover, we demonstrate that this method can be generalized to address the polynomial approximation problem for any piece-wise polynomial.

Method II (AdaptiveLP). This method reduces the number of multiplications by employing the composite polynomial strategy. Specifically, we construct a composite polynomial $g \circ f_k \circ \cdots \circ f_1$ approximating $\kappa(x)$ in two steps.

The first step aims to construct polynomials $f_1, f_2, \cdots, f_k$ which progressively map the intervals (a_{i-1}, a_i) to smaller intervals around their midpoint $\frac{1}{2}(a_i + a_{i-1})$ for $1 \le i \le n$. We demonstrate that the task of determining f_j is equivalent to solving the *weighted minimax polynomial approximation* problem as defined in Problem 1. An additional desirable property of f_j's is that their coefficients can be bounded, thereby allowing for high precision homomorphic evaluation [23]. We introduce an *adaptive linear programming algorithm* (see Algorithm 2), which gives the optimal weighted minimax polynomial approximation for step functions while keeping the coefficients bounded.

The second step involves constructing a polynomial $g(x)$ that maps the midpoints to $y_i, 1 \le i \le n$. We demonstrate that the optimal $g(x)$, which has bounded coefficients and minimizes the approximation error, can be derived using the adaptive linear programming algorithm again.

Applications to Concrete Step Functions. We demonstrate the two methods by presenting polynomial approximations for the round function and the bucketing function. Specifically, we give concrete polynomial approximations for the 7-step function $\frac{1}{3}\lfloor 3x \rceil$ and a 5-step function obtained from a bucketing example, and provide explicit error rates and running time for these approximations by evaluating them with the HEAAN library. According to experiments, it appears that `SgnToStep` has an advantage in terms of bit consumption, while `AdaptiveLP` demonstrates more desirable performance in terms of running time. The source code is available at https://anonymous.4open.science/r/code_upload-131E/.

1.2 Related Works

Numerical Analysis on Piece-wise Functions The problem of polynomial approximation for piece-wise functions has been studied for decades in numerical analysis. Some of these works focus on the polynomial approximations of piece-wise smooth functions [33,4,32,31]. Because step functions have discontinuities and piece-wise smooth functions are continuous, these results are not applicable to step functions. Another portion of works focus on functions with a single discontinuity, such as the sign function [16,32,17]. However, as observed in [11], when the approximation error needs to be exponentially small, the degree of the approximation polynomial becomes quite large, resulting in exponential homomorphic evaluation complexity.

Composite Polynomial Approximation of Sign Function To improve the homomorphic computational complexity for the sign function, Cheon et al. proposed composite polynomial method that achieves asymptotic computational optimality [12,11]. Later, Lee et al. proposed a minimax composite polynomial method that achieves practical computational optimality [25,26]. However, these methods cannot be extended to handle polynomial approximations for general step functions because the intervals and values of a step function can be intricacy.

1.3 Organization

Section 2 introduces some notations. Section 3 and Section 4 propose `SgnToStep` and `AdaptiveLP` respectively. To demonstrate our method, we apply `SgnToStep` and `AdaptiveLP` to the round function and bucketing function in Section 5, and present experimental results of evaluating these step functions in HEAAN library in Section 6.

2 Preliminary

- We denote $\mathbb{Z}, \mathbb{R}$ and $\mathbb{C}$ to be the ring of integers, the field of real numbers and the field of complex numbers, respectively.
- The Chebyshev polynomials $T_n(x)$ on the interval $[-1, 1]$ are defined by $\cos n\theta = T_n(\cos\theta)$, which satisfy the following recursion: $T_0(x) = 1, T_1(x) = x, T_i(x) = 2xT_{i-1}(x) - T_{i-2}(x)$ for $i \geq 2$.
- For a real function f defined over $\mathbb{R}$, let $f^{(d)} \coloneqq f \circ f \circ \cdots \circ f$ denote the d-time composition of f. We use the infinite norm to measure the accuracy of polynomial approximations as suggested in [11,12]. For a function f and a a compact set $I \subset \mathbb{R}$, the infinite norm is defined by

$$\|f\|_I \coloneqq \sup_{x\in I}|f(x)|.$$

  Besides, let $\mathcal{C}_{max}(f)$ denote the maximum absolute value of f's coefficients (in terms of a polynomial basis, such as the power basis or the Chebyshev basis, depending on the context).
- Let $\log(\cdot)$ denote the logarithm of base 2. For $x \in \mathbb{R}$, let $\lfloor x \rceil = \lfloor x + 1/2 \rfloor$ denote the integer closest to x, and let $\texttt{sgn}(x)$ denote the sign function

$$\texttt{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}. \tag{1}$$

2.1 Step Function

The step functions considered in this paper are piece-wise constant real functions with finitely many pieces, which can be formally defined as follows.

Definition 1 (Step Function). *A real function $\kappa(x)$ defined on the interval $[a,b]$ is a step function if there exists a finite partition $a = a_0 < a_1 < \cdots < a_n = b$ such that $\kappa(x)$ is a constant function on each interval (a_{i-1}, a_i), i.e., there exist $y_i \in \mathbb{R}$ such that*

$$\kappa(x) = y_i \text{ for } x \in (a_{i-1}, a_i), 1 \le i \le n.$$

We call $(a_{i-1}, a_i), 1 \le i \le n$, the intervals of $\kappa(x)$, and call $y_i, 1 \le i \le n$, the values of $\kappa(x)$. For convenience we always assume $\kappa(a_0) = y_1$ and $\kappa(a_n) = y_n$. The value of $\kappa(a_i)$, $1 \le i \le n-1$, is not specified in the definition. From the perspective of polynomial approximation, we do not mind the specific value of $\kappa(a_i)$ for $1 \le i \le n-1$. Besides, we always assume that a_i is a jump discontinuity, i.e., $y_i \ne y_{i+1}$ for $1 \le i \le n-1$.

Since $\kappa(x)$ can not be approximated by polynomials near the discontinuities a_i's, $1 \le i \le n-1$, we follow the approach adopted in [11,12,25] and define the following measurement of approximation error.

Definition 2. *For small real numbers $2^{-\alpha}, \epsilon > 0$, we say a polynomial $f(x)$ is (α, ϵ)-close to a step function $\kappa(x)$ with partition $a = a_0 < a_1 < \cdots < a_n = b$ if*

$$\|f(x) - \kappa(x)\|_I \le 2^{-\alpha},$$

where $I = [a,b] - \bigcup_{1 \le i < n} (a_i - \epsilon, a_i + \epsilon)$.

Definition 3. *For a step function $\kappa(x)$ with partition $a = a_0 < a_1 < \cdots < a_n = b$ and constant values $y_1, \ldots, y_n$. We say $\kappa(x)$ is normalized if $y_1 = a, y_n = b$ and $y_i = \frac{1}{2}(a_{i-1} + a_i)$ for $1 < i < n$. We say $\tilde{\kappa}(x)$ is the normalization of $\kappa(x)$ if $\tilde{\kappa}(x)$ is normalized and $\tilde{\kappa}(x)$ has the same partition as $\kappa(x)$.*

2.2 Homomorphic Encryption Scheme

In this paper we focus on word-wise FHEs, which can be specified by the following algorithms.

- $\texttt{KeyGen}(L, \lambda)$. `KeyGen` takes a level parameter L and a security parameter λ as input, and outputs a public key `pk`, a secret key `sk`, and an evaluation key `evk`.
- $\texttt{Enc}(\texttt{pk}, m)$. `Enc` takes a public key `pk` and a message m as input, and outputs the ciphertext `ct`.
- $\texttt{Dec}(\texttt{sk}, \texttt{ct})$. `Dec` takes a secret key `sk` and a ciphertext `ct` as input, and outputs the plaintext m.
- $\texttt{Add}(\texttt{evk}, \texttt{ct}_1, \texttt{ct}_2)$. `Add` takes as input an evaluation key `evk` and the ciphertexts $\texttt{ct}_1$ and $\texttt{ct}_2$ of two messages m_1 and m_2, and outputs the ciphertext $\texttt{ct}_{\text{add}}$ of the message $m_1 + m_2$.

– $\texttt{Mult}(\texttt{evk}, \texttt{ct}_1, \texttt{ct}_2)$. $\texttt{Mult}$ takes as input an evaluation key $\texttt{evk}$ and the ciphertexts $\texttt{ct}_1$ and $\texttt{ct}_2$ of two messages m_1 and m_2, and outputs the ciphertext $\texttt{ct}_{\text{mult}}$ of the message $m_1 \cdot m_2$.

For approximate FHE (i.e., CKKS), $\texttt{Dec}$ outputs an approximate value of the message m instead of the exact value. Because $\texttt{Mult}$ is significantly more expensive than $\texttt{Add}$, we mainly consider the number and depth consumption of non-scalar multiplications in this paper.

3 SgnToStep: Step Function Approximation by Using the Connection with sgn

In this section, we provide a linear relation between the step function and the sign function. Based on this connection, any step function $\kappa(x)$ can be homomorphically evaluated by using the approximations of $\texttt{sgn}(x)$ as presented in [11,12,25,28].

3.1 A Connection between Step Function and Sign Function

It is obvious that a step function $\kappa(x)$ with n intervals can be expressed as a linear combination of at most n indicator functions of intervals. In fact, the following lemma states that $\kappa(x)$ can also be written as a linear combination of $n-1$ shifted sign functions.

Lemma 1. *A step function $\kappa(x)$ with partition $a_0 < a_1 < \cdots < a_n$ and values $y_1, \cdots, y_n$ can be expressed by a linear combination of $n-1$ shifted sign functions, i.e.,*

$$\kappa(x) = \sum_{i=1}^{n-1} c_i \texttt{sgn}(x - a_i) + c_n, \tag{2}$$

where $c_i = \frac{1}{2}(y_{i+1} - y_i)$ for $1 \le i \le n-1$ and $c_n = \frac{1}{2}(y_1 + y_n)$.

Proof. It suffices to check equation (2) for the intervals $(a_{i-1}, a_i), 1 \le i \le n$. Suppose $x \in (a_{i-1}, a_i)$, then the left hand side of (2) is $\kappa(x) = y_i$. Note that $\texttt{sgn}(x-a_1) = \cdots = \texttt{sgn}(x-a_{i-1}) = 1$ and $\texttt{sgn}(x-a_i) = \cdots = \texttt{sgn}(x-a_{n-1}) = -1$ for $x \in (a_{i-1}, a_i)$, then the right hand side of (2) is

$$\sum_{j=1}^{i-1} c_j - \sum_{j=i}^{n-1} c_j + c_n = \frac{1}{2}(y_i - y_1) - \frac{1}{2}(y_n - y_i) + \frac{1}{2}(y_1 + y_n) = y_i.$$

Thus equation (2) holds. □

Because a linear combination of k shifted sign functions has at most k discontinuities, $n-1$ is the smallest number of shifted sign functions required to represent $\kappa(x)$ linearly.

3.2 Step Function Approximation Based on the Linear Combination

We demonstrate how to use Lemma 1 and a polynomial approximation of $\mathtt{sgn}(x)$ to obtain a polynomial approximation of a step function $\kappa(x)$. Suppose $g(x)$ is a (composite) polynomial approximation of $\mathtt{sgn}(x)$ as constructed in [12,25], such that $g(x)$ is (α, ϵ)-close to $\mathtt{sgn}(x)$ on $[-1, 1]$, i.e.

$$\|g(x) - \mathtt{sgn}(x)\|_{[-1,-\epsilon]\cup[\epsilon,1]} \le 2^{-\alpha}. \tag{3}$$

Then an approximation of $\kappa(x)$ can be constructed as follows.

Theorem 1. *Let $\kappa(x)$ be a step function with partition $-1 = a_0 < a_1 < \cdots < a_n = 1$ and values $y_1, \cdots, y_n$. Suppose $g(x)$ is (α, ϵ)-close to $\mathtt{sgn}(x)$ on $[-1, 1]$. Then the function*

$$f(x) = \sum_{i=1}^{n-1} \frac{1}{2}(y_{i+1} - y_i) \cdot g(\frac{x - a_i}{1 + |a_i|}) + \frac{1}{2}(y_1 + y_n)$$

is (α', ϵ')-close to $\kappa(x)$ on $[-1, 1]$, where $\alpha' = \alpha - \log(\sum_{i=1}^{n-1} \frac{1}{2}|y_{i+1} - y_i|)$ and $\epsilon' = (1 + \max\{|a_1|, |a_{n-1}|\})\epsilon$.

Proof. We first show that $g(\frac{x-a_i}{1+|a_i|})$ is an approximation of $\mathtt{sgn}(x - a_i)$ on $I \coloneqq [-1, 1] - \bigcup_{1\le i<n}(a_i - \epsilon', a_i + \epsilon')$. Denote $y = \frac{x-a_i}{1+|a_i|}$, then for $x \in I$ it has $|y| \le \frac{|x|+|a_i|}{1+|a_i|} \le 1$ and $|y| = \frac{|x-a_i|}{1+|a_i|} \ge \frac{\epsilon'}{1+|a_i|} \ge \epsilon$, i.e., $y \in [-1, -\epsilon] \cup [\epsilon, 1]$. Thus $\left\|g(\frac{x-a_i}{1+|a_i|}) - \mathtt{sgn}(x - a_i)\right\|_I \le \|g(y) - \mathtt{sgn}((1 + |a_i|)y)\|_{[-1,-\epsilon]\cup[\epsilon,1]} = \|g(y) - \mathtt{sgn}(y)\|_{[-1,-\epsilon]\cup[\epsilon,1]} \le 2^{-\alpha}$. Therefore, by Lemma 1 it has

$$\begin{aligned}\|f(x) - \kappa(x)\|_I &= \left\|\sum_{i=1}^{n-1} \frac{1}{2}(y_{i+1} - y_i) \cdot (g(\frac{x - a_i}{1 + |a_i|}) - \mathtt{sgn}(x - a_i))\right\|_I \\ &\le \sum_{i=1}^{n-1} \frac{1}{2}|y_{i+1} - y_i| \cdot 2^{-\alpha} = 2^{-\alpha'},\end{aligned}$$

which completes the proof. □

Remark 1. Different approximations for $\mathtt{sgn}(x - a_i)$ can be chosen to balance the overall error rate. Specifically, suppose $g_i(x)$ is (α_i, ϵ_i)-close to $\mathtt{sgn}(x)$ on $[-1, 1]$ for $1 \le i < n$. Then it can be similarly proved that the function $f(x) = \sum_{i=1}^{n-1} \frac{1}{2}(y_{i+1} - y_i) \cdot g_i(\frac{x-a_i}{1+|a_i|}) + \frac{1}{2}(y_1 + y_n)$ is (α', ϵ')-close to $\kappa(x)$ on $[a, b]$, where $\alpha' = \log(\sum_{i=1}^{n-1} \frac{1}{2}|y_{i+1} - y_i| \cdot 2^{-\alpha_i})$ and $\epsilon' = \max_{1\le i<n}\{(1 + |a_i|)\epsilon_i\}$.

Computation Complexity. The polynomial approximation for $\mathtt{sgn}(x)$ is usually given in a composite polynomial form to reduce the number of homomorphic multiplications, i.e., $g = h_k \circ h_{k-1} \circ \cdots \circ h_1$. Then the $g(\frac{x-a_i}{1+|a_i|})$ in Theorem 1

Algorithm 1: Compute step function by using Theorem 1.

Input: A real number $x_0 \in [-1, 1]$
Input: A step function $\kappa(x)$ with partition $a_0 < \cdots < a_n$ and values $y_1, \cdots, y_n$
Input: A sub-algorithm `ComputeG` that computes $g(x)$, where $g(x)$ is a composite polynomial approximation of $\texttt{sgn}(x)$
Output: Approximate value of $\kappa(x_0)$

for i from 1 to $n-1$ **do**
  $z_i = \texttt{ComputeG}(\frac{x_0 - a_i}{1+|a_i|})$
end for
$z = \sum_{i=1}^{n-1} \frac{1}{2}(y_{i+1} - y_i) \cdot z_i + \frac{1}{2}(y_1 + y_n)$
return z

should be evaluated individually before performing the linear combination (Algorithm 1).

The required multiplicative depth for Algorithm 1 is roughly the same as that for `ComputeG` (or $g(x)$), and the number of multiplications is $n-1$ times as that of `ComputeG` (or $g(x)$). The total running time can be reduced if each $g(\frac{x-a_i}{1+|a_i|})$ can be computed in parallel.

3.3 Extension to Piece-wise Polynomials

Suppose $\rho(x)$ is a piece-wise polynomial defined on $[a, b]$ such that

$$\rho(x) = p_i(x) \text{ for } x \in (a_{i-1}, a_i), 1 \le i \le n, \tag{4}$$

where $a = a_0 < a_1 < \cdots < a_n = b$, and $p_i(x)$'s are polynomials defined on $[a, b]$. Similar to Lemma 1, the following lemma can be proved.

Lemma 2. *$\rho(x)$ can be expressed as*

$$\rho(x) = \sum_{i=1}^{n-1} \frac{1}{2}(p_{i+1}(x) - p_i(x)) \cdot \texttt{sgn}(x - a_i) + \frac{1}{2}(p_1(x) + p_n(x)) \tag{5}$$

for $x \in [a, b]$ other than the singularity points.

Then a polynomial approximation of $\rho(x)$ can be constructed based on the polynomial approximation of $\texttt{sgn}(x)$ as follows.

Theorem 2. *Suppose $\rho(x)$ is a piece-wise polynomial on $[-1, 1]$ and $g(x)$ is (α, ϵ)-close to $\texttt{sgn}(x)$ on $[-1, 1]$. Then the function*

$$f(x) = \sum_{i=1}^{n-1} \frac{1}{2}(p_{i+1}(x) - p_i(x)) \cdot g(\frac{x - a_i}{1 + |a_i|}) + \frac{1}{2}(p_1(x) + p_n(x))$$

is (α', ϵ')-close to $\rho(x)$ on $[-1, 1]$, i.e, $\|\rho(x) - f(x)\|_I \le 2^{-\alpha'}$, where $I = [-1, 1] - \bigcup_{1 \le i < n}(a_i - \epsilon', a_i + \epsilon')$ and $\alpha' = \alpha - \log(\sum_{i=1}^{n-1} \frac{1}{2}\|p_{i+1}(x) - p_i(x)\|_I, \epsilon' = (1 + \max\{|a_1|, |a_{n-1}|\})\epsilon$.

Proof. It can be proved as in Theorem 1 that $\|g(\frac{x-a_i}{1+|a_i|}) - \mathtt{sgn}(x-a_i)\|_I \le 2^{-\alpha}$. Then by Lemma 2 it has

$$\begin{aligned}\|f(x)-\rho(x)\|_I &= \left\| \sum_{i=1}^{n-1} \frac{1}{2}(p_{i+1}(x)-p_i(x)) \cdot (g(\frac{x-a_i}{1+|a_i|}) - \mathtt{sgn}(x-a_i)) \right\|_I \\ &\le \sum_{i=1}^{n-1} \frac{1}{2} \|p_{i+1}(x)-p_i(x)\|_I \cdot 2^{-\alpha} = 2^{-\alpha'},\end{aligned}$$

which completes the proof. □

4 AdaptiveLP: Step Function Approximation by Polynomial Composition

In this section, we consider the composite polynomial strategy to approximate step functions. For any step function $\kappa(x)$, we aim to construct a composite polynomial $g \circ f_k \circ \cdots \circ f_1$ that approximates $\kappa(x)$. The construction can be divided into two steps, which are specified in Section 4.1 and Section 4.2 respectively.

Step 1. Construct a composite polynomial $f = f_k \circ \cdots \circ f_1$ approximating $\tilde{\kappa}(x)$, where $\tilde{\kappa}(x)$ is the normalization of $\kappa(x)$.
Step 2. Construct a polynomial $g(x)$ such that $g(\tilde{\kappa}(x)) \approx \kappa(x)$.

4.1 Construction of the Composite polynomial f

Suppose $\tilde{\kappa}(x) = z_i$ for $x \in (a_{i-1}, a_i)$, where $z_1 = a, z_n = b$ and $z_i = \frac{1}{2}(a_{i-1} + a_i), 1 < i < n$. Our goal is to construct polynomials $f_1, \cdots, f_k$ such that they gradually map the intervals to small intervals. For a small positive real number ϵ, denote $I_{10} = [a_0, a_1 - \epsilon], I_{n0} = [a_{n-1} + \epsilon, a_n]$ and $I_{i0} = [a_{i-1} + \epsilon, a_i - \epsilon]$ for $1 < i < n$. Then the polynomials $f_1, \cdots, f_k$ should satisfy

$$I_{i0} \xrightarrow{f_1} [z_i - t_{i1}, z_i + t_{i1}] \xrightarrow{f_2} \cdots \xrightarrow{f_k} [z_i - t_{ik}, z_i + t_{ik}] \tag{6}$$

for $1 \le i \le n$, where $t_{i1} > \cdots t_{ik} > 0$. Denote $I_{ij} = [z_i - t_{ij}, z_i + t_{ij}]$ for $1 \le i \le n$ and $1 \le j \le k$, and let $t_{10} = a_1 - a_0 - \epsilon$, $t_{n0} = a_n - a_{n-1} - \epsilon$, and $t_{i0} \coloneqq \frac{1}{2}(a_i - a_{i-1}) - \epsilon$ for $1 < i < n$. Then the optimal polynomial f_{j+1} should minimize the ratio

$$\max_{1\le i\le n} \frac{t_{i,j+1}}{t_{ij}} = \max_{1\le i\le n} \frac{1}{t_{ij}} \cdot \|f_{j+1}(x) - z_i\|_{I_{ij}} = \max_{1\le i\le n} \frac{1}{t_{ij}} \cdot \|f_{j+1}(x) - \tilde{\kappa}(x)\|_{I_{ij}}. \tag{7}$$

On the other hand, we want the coefficients of f_j to be bounded by a real constant number B_j to ensure evaluation precision. In other words, for $0 \le j \le k-1$, the polynomial f_{j+1} is a solution to the following optimization problem.

Problem 1 (Weighted Minimax Polynomial Approximation) *For input step function* $\tilde{\kappa}(x)$*, constant numbers* $t_{ij} > 0$*, intervals* I_{ij} *for* $1 \le i \le n$*, find a polynomial* $f_{j+1}(x)$ *with degree no more than* d_{j+1} *and coefficients bounded by* $\mathcal{C}_{max}(f_{j+1}) \le B_{j+1}$ *that minimizes*

$$\max_{1\le i\le n} \frac{1}{t_{ij}} \cdot \|f_{j+1}(x) - \tilde{\kappa}(x)\|_{I_{ij}}. \tag{8}$$

Solving Problem 1 via Adaptive Linear Programming. Suppose c_{opt} is the minimum value of (8). The adaptive linear programming algorithm iteratively computes a polynomial $\hat{f}_{j+1}$ such that the value $\max_{1\le i\le n}\{\frac{1}{t_{ij}} \cdot \|\hat{f}_{j+1}(x) - \tilde{\kappa}(x)\|_{I_{ij}}\}$ approaches c_{opt}.

To begin with, we choose a set of reference points $\mathcal{X} \subset \cup_{1\le i\le n} I_{ij}$, and consider the conditions

$$\begin{cases} \frac{1}{t_{ij}} \cdot |f_{j+1}(x_l) - \tilde{\kappa}(x_l)| \le c, \forall 1 \le i \le n, \text{ for } x_l \in \mathcal{X}; \\ \mathcal{C}_{max}(f_{j+1}) \le B_{j+1}, \end{cases} \tag{9}$$

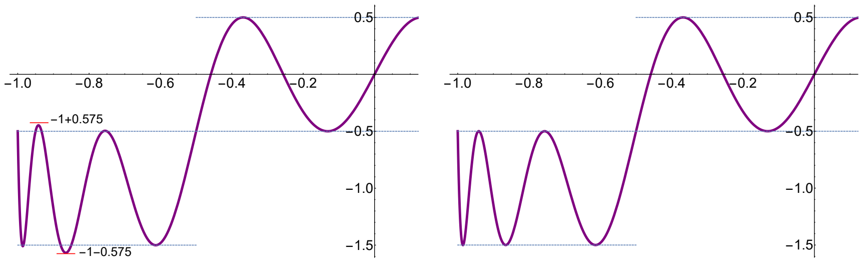
where c is the objective to be minimized. Then (9) provides linear constrains on the coefficients of f_{j+1} and c. As a result, we can obtain a polynomial $\hat{f}_{j+1}$ and a real number $c_l > 0$ by using linear programming to minimize c. Clearly c_l is a lower bound of c_{opt} since the solution $f_{j+1}^{(\mathrm{opt})}$ to Problem 1 must satisfy $\frac{1}{t_{ij}} \cdot |f_{j+1}^{(\mathrm{opt})}(x_l) - \tilde{\kappa}(x_l)| \le c_{\mathrm{opt}}, \forall x_l \in \mathcal{X}$.

On the other hand, for the polynomial $\hat{f}_{j+1}(x)$ obtained by solving (9), let

$$c_u \coloneqq \max_{1\le i\le n} \frac{1}{t_{ij}} \cdot \|\hat{f}_{j+1}(x) - \tilde{\kappa}(x)\|_{I_{ij}}.$$

Clearly c_u is an upper bound of c_{opt}. In order to decrease c_u, we collect all the extreme and boundary points $x' \in \cup_{1\le i\le n} I_{ij}$ of the polynomial $\hat{f}_{j+1}(x)$ such that $\frac{1}{t_{ij}} \cdot |\hat{f}_{j+1}(x') - \tilde{\kappa}(x')| > c_l$, add all these points to the set $\mathcal{X}$, and repeat the linear programming process. Algorithm 2 summarizes the above procedure. For the choice of polynomial basis, it was observed that the Chebyshev basis is suitable for minimax polynomial approximation [8,26]. Besides, an efficient homomorphic computation method for the Chebyshev basis has been proposed [27]. Thus we also adopt the Chebyshev basis for polynomial approximation in this paper.

Termination and Runtime of the Algorithm. When performing Algorithm 2, it is clear that the c_l gradually increases because more linear constrains are added to (9). Moreover, through experiments, we find that the c_u quickly approaches c_l and thus approaches c_{opt}. Fig. 1 depicts the first two iterations of Algorithm 2 for solving the weighted minimax problem that corresponds to construct f_1 for $\tilde{\kappa}(x) = [x], x \in [-1, 1]$, and $\epsilon = 2^{-16}$. From the



(a) $\hat{f}_{j+1}$ after the first iteration. (b) $\hat{f}_{j+1}$ after the second iteration.

Fig. 1: Illustration of the first two iterations of adaptive linear programming algorithm. The graph of f_{j+1} is symmetric with respect to the origin.

figure, we can see that the gap between c_u and c_l is narrowed after the second iteration. In fact, Algorithm 2 outputs the $f_{j+1}(x)$ in a few iterations according to our experiments. For example, Table 1 lists the number of iterations required for constructing composite polynomial approximation of the step function $\frac{1}{3}\lfloor 3x \rceil, x \in [-1, 1], \epsilon = 2^{-16}$, $\gamma = 2^{-30}$.

Algorithm 2: Adaptive linear programming

Input: A step function $\tilde{\kappa}(x)$ and an approximation factor $\gamma \in \mathbb{R}^+$
Input: Real numbers $t_{ij} > 0$ and intervals I_{ij} for $1 \le i \le n$
Input: Polynomial degree $d_{j+1} \in \mathbb{Z}^+$ and coefficient bound $B_{j+1} > 0$
Input: A polynomial basis $\{p_l(x)\}_{1 \le l \le d_{j+1}}$
Output: Approximate polynomial $f_{j+1}(x)$ that minimize (8)

1: Choose a set of reference points $\mathcal{X} \subset \cup_{1 \le i \le n} I_{ij}$
2: Solve the following linear programming problem and obtain $\hat{f}_{j+1}$ and c_l
  Minimize c
  Subject to $\mathcal{C}_{max}(f_{j+1}) \le B_{j+1}$ and $|f_{j+1}(x_l) - \tilde{\kappa}(x_l)| \le ct_{ij}, \forall x_l \in \mathcal{X}$
3: Collect the extreme and boundary points $x' \in \cup_{1 \le i \le n} I_{ij}$ such that $|\hat{f}_{j+1}(x') - \tilde{\kappa}(x')| > c_l t_{ij}$, and add them to $\mathcal{X}$
4: Compute $c_u = \max_{1 \le i \le n}\{\frac{1}{t_{ij}} \cdot \|\hat{f}_{j+1}(x) - \tilde{\kappa}(x)\|_{I_{ij}}\}$.
5: **if** $c_u < (1+\gamma)c_l$ **then**
6:   **return** $\hat{f}_{j+1}$
7: **else**
8:   Go to line 2
9: **end if**

In each iteration of Algorithm 2, a linear programming algorithm is employed to solve c_l. It is shown in [14] that solving such linear programming takes $\mathcal{O}^*(|\mathcal{X}|^c \log(|\mathcal{X}|/\delta))$ time, where $2 < c < 3$ is a constant determined by the matrix multiplication algorithm, and δ is the relative accuracy. According to our experiment, for a step function with n intervals, and a polynomial degree d, a coefficient bound B, setting $|\mathcal{X}| = \mathcal{O}(nd)$ and $\delta = \mathcal{O}(\epsilon/(dB))$ suffices for the computation.

Determine the Composite Polynomial. The polynomials f_{j+1} can be constructed using Algorithm 2 iteratively for $0 \le j \le k-1$. Here the $t_{i,j+1}$'s are determined by $t_{i,j+1} = \|f_{j+1}(x) - \tilde{\kappa}(x)\|_{I_{ij}}$ after f_j has been determined. Due to our choice of $\tilde{\kappa}(x)$ and I_{ij}, it has

$$\begin{aligned}\max_{1\le i\le n} \frac{1}{t_{ij}} \cdot \|f_{j+1}(x) - \tilde{\kappa}(x)\|_{I_{ij}} &< (1+\gamma) \max_{1\le i\le n} \frac{1}{t_{ij}} \cdot \|f_{j+1}^{(\mathrm{opt})} - \tilde{\kappa}(x)\|_{I_{ij}} \\ &\le (1+\gamma) \max_{1\le i\le n} \frac{1}{t_{ij}} \cdot \|x - \tilde{\kappa}(x)\|_{I_{ij}} = (1+\gamma),\end{aligned}$$

i.e. $t_{i,j+1} < (1+\gamma)t_{i,j}$. In our experiment, it holds $t_{i,j+1} < t_{i,j}$ for an appropriate choice of the factor γ, thus the mapping of intervals in (6) can be guaranteed.

Nevertheless, in the encrypted state, f_{j+1} will be homomorphically evaluated, and $f_{j+1}(I_{ij})$ may not fall into $I_{i,j+1}$ due to the homomorphic computation errors. This can cause an evaluation failure of the composite polynomial $f = f_k \circ \cdots \circ f_1$. To solve this problem, we introduce a parameter η_{j+1} which is an upper bound of the homomorphic evaluation error, i.e.,

$$|\mathtt{Eval}(f_j)(x) - f_j(x)| \le \eta_{j+1} \ll 1,$$

for $0 \le j \le k-1$. Besides, we set η_0 to be the encryption error. Then we use the intervals $I'_{ij} := [z_i - t_{ij} - \eta_j, z_i + t_{ij} + \eta_j]$ as input to solve f_{j+1} (instead of I_{ij}), which ensures the mapping of intervals in (6) for the encrypted state. The above process is summarized in Algorithm 3.

4.2 Construction of the polynomial $g(x)$

Using Algorithm 3 we obtain a composite polynomial $f = f_k \circ \cdots \circ f_1$ such that $|f(x) - z_i| \le t_{ik}, x \in I_{i0}$ for all $1 \le i \le n$. Then as discussed in Section 1.1, the polynomial $g(x)$ is determined by minimizing

$$\max_{1\le i\le n} \|g(z) - y_i\|_{[z_i - t_{ik}, z_i + t_{ik}]}$$

for a given degree $\deg(g) \le d$ and coefficient bound $\mathcal{C}_{max}(g) \le B$. Particularly, the following lemma holds.

Table 1: The number of iterations of Algorithm 2 for approximating the step function $\frac{1}{3}\lfloor 3x \rceil$ on $[-1,1]$, where $\{z_i\}_i = \{0, \pm\frac{1}{3}, \pm\frac{2}{3}, \pm 1\}$ and $\{t_{ij}\}_i$ are roughly equal for the same j. The degrees of f_j are set to be 31.

f_{j+1}	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
$t_{ij} \approx$	$1/6 - 2^{-16}$	$1/6 - 2^{-13.5}$	$1/6 - 2^{-11.0}$	$1/6 - 2^{-8.6}$	$1/6 - 2^{-6.2}$	$1/6 - 2^{4.0}$	$1/6 - 2^{-2.8}$	$1/6 - 2^{-2.6}$
#Iterations	5	4	4	4	4	3	3	1

Algorithm 3: Construct the composite polynomial

Input: A step function $\tilde{\kappa}(x)$ and an approximation factor $\gamma \in \mathbb{R}^+$
Input: $t_{i0} \in \mathbb{R}^+$ and intervals I'_{i0} for $1 \le i \le n$
Input: Polynomial degree $d_{j+1} \in \mathbb{Z}^+$, coefficient bound $B_{j+1} \in \mathbb{Z}^+$ and error bound $\eta_j \in \mathbb{R}^+$ for $0 \le j \le k-1$
Input: A polynomial basis $\{p_l(x)\}_l$
Output: Composite polynomial $f = f_k \circ \cdots \circ f_1$ approximating $\tilde{\kappa}(x)$

for j from 0 to $k-1$ **do**
  Compute a polynomial f_{j+1} by using $\tilde{\kappa}(x)$, γ, t_{ij}, I'_{ij}, d_{j+1}, B_{j+1} and $\{p_l(x)\}_l$ as the inputs of Algorithm 2
  Compute $t_{i,j+1} = \|f_{j+1}(x) - \tilde{\kappa}(x)\|_{I'_{ij}}$
  $I'_{i,j+1} \coloneqq [z_i - t_{i,j+1} - \eta_{j+1}, z_i + t_{i,j+1} + \eta_{j+1}]$ for $1 \le i \le n$
end for
return $f_k \circ \cdots \circ f_1$

Lemma 3. *Suppose $|g(z) - y_i| \le 2^{-\alpha}$ for $z \in [z_i - t_{ik}, z_i + t_{ik}], 1 \le i \le n$. Then the composite polynomial $g \circ f$ is an approximation of $\kappa(x)$ such that $\|g \circ f - \kappa\|_I \le 2^{-\alpha}$, where $I = \cup_{i=1}^n I_{i0}$.*

Proof. For any $x \in I_{i0}$, it has $z \coloneqq f(x) \in [z_i - t_{ik}, z_i + t_{ik}]$. Thus $|g(f(x)) - y_i| = |g(z) - y_i| \le 2^{-\alpha}$. □

Using Algorithm 2, the problem of minimizing $\|g \circ f - \kappa\|_I \le 2^{-\alpha}$ can be solved. Taking into account the homomorphic computation errors, $g(x)$ should be computed in a similar way as f_{j+1}. Specifically, let η_{k+1} and η_g be the upper bounds of the homomorphic evaluation error of f_k and g, respectively. Then our goal is to find $g(x)$ that minimizes $\max_{1 \le i \le n} \|g(z) - y_i\|_{[z_i - t_{ik} - \eta_{k+1}, z_i + t_{ik} + \eta_{k+1}]}$. Moreover, due to the error introduced by homomorphically evaluating $g(x)$, the composite polynomial $g \circ f$ is an approximation of $\kappa(x)$ such that $\|g \circ f - \kappa\|_I \le 2^{-\alpha} + \eta_g$ in the encrypted state. We specify the computation of $g(x)$ in Algorithm 4.

Remark 2. To use Algorithms 2, 3, 4 for computing concrete step functions, we need to choose the parameters polynomial degree d, coefficient bound B and error bound η in advance. In our experiments, we set $d \in \{15, 31\}$ and adjust η according to the homomorphic errors, and select B to minimize the number of composite polynomials.

5 Application to Concrete Step Functions

In this section, we apply the two methods `SgnToStep` and `AdaptiveLP` to the round function $\texttt{Round}_m(x)$ and an example of bucketing function in the plaintext state ($\eta = 0$ in Algorithm 2). Suppose a step function $\kappa(x)$ is approximated by a polynomial $f(x)$. Then any step function obtained by applying stretching, shifting and reflecting transformations to $\kappa(x)$ will be approximated by the

Algorithm 4: Compute the polynomial $g(x)$.

Input: A step function $\kappa(x)$ and an approximation factor $\gamma \in \mathbb{R}^+$
Input: $z_i \in \mathbb{R}$ and $t_{ik} \in \mathbb{R}^+$ for $1 \le i \le n$
Input: Polynomial degree $d \in \mathbb{Z}^+$, coefficient bound $B \in \mathbb{Z}^+$, error bound $\eta_{k+1}, \eta_g \in \mathbb{R}^+$
Input: A polynomial basis $\{p_l(x)\}_{1 \le l \le d}$
Output: Polynomial $g(x)$ approximating $\kappa(x)$, and an error rate $2^{-\alpha}$

1: $I_i' = [z_i - t_{ik} - \eta_{k+1}, z_i + t_{ik} + \eta_{k+1}]$ for $1 \le i \le n$
2: Compute a polynomial $g(x)$ by using $\kappa(x), \gamma, I_i', d, B$ and $\{p_l(x)\}_{1 \le l \le d}$ as the inputs of Algorithm 2
3: Compute $t = \max_{1 \le i \le n} \|g(x) - y_i\|_{I_i'}$
4: Compute $\alpha = -\log(t + \eta_g)$
5: **return** $g(x)$ and $2^{-\alpha}$

polynomial obtained by applying the same transformations to $f(x)$. The approximation error rate will change but can be easily predicted. As a result, in this section, we assume all step functions are defined over $[-1,1]$, and their values also fall in $[-1,1]$.

5.1 Application to the Round Function

The round function in this section is a step function with $2m+1$ intervals defined over $[-1,1]$, i.e.,

$$\texttt{Round}_m(x) = \frac{1}{m}\lfloor mx \rceil = \begin{cases} -1, & x \in (-1, -1 + \frac{1}{2m}) \\ \frac{i}{m}, & x \in (\frac{i}{m} - \frac{1}{2m}, \frac{i}{m} + \frac{1}{2m}) \text{ for } -m < i < m \\ 1, & x \in (1 - \frac{1}{2m}, 1) \end{cases}$$

where m is a positive integer.

Apply `SgnToStep` to $\texttt{Round}_m(x)$. The following corollary directly results from Theorem 1.

Corollary 1. *Suppose $g(x)$ is a polynomial that is (α', ϵ')-close to $\texttt{sgn}(x)$ on $[-1,1]$, then the polynomial*

$$f(x) = \frac{1}{2m} \sum_{i=0}^{m-1} \left(g\left(\frac{mx + i + \frac{1}{2}}{m + i + \frac{1}{2}}\right) + g\left(\frac{mx - i - \frac{1}{2}}{m + i + \frac{1}{2}}\right) \right) \quad (10)$$

is (α, ϵ)-close to $\texttt{Round}_m(x)$ on $[-1,1]$, where $\alpha = \alpha'$ and $\epsilon = (2 - \frac{1}{2m})\epsilon'$.

In the following, we focus on $m = 3$ and give concrete polynomial approximations of $\texttt{Round}_3(x)$ based on the constructions in [11,25]. According to Corollary 1, to obtain a polynomial $f(x)$ that is (α, ϵ)-close to $\texttt{Round}_3(x)$ on $[-1,1]$, it suffices to construct a polynomial $g(x)$ that is $(\alpha, \epsilon/(2 - \frac{1}{2m}))$-close to $\texttt{sgn}(x)$. Such $g(x)$ is chosen as follows.

- Using the construction in Section 3.1 of [11], $g(x)$ can be defined to be the composite polynomial $h_r^{(k)}$ where $h_r(x) = \sum_{i=0}^{r} \frac{1}{4^i}\binom{2i}{i} x(1-x^2)^i$. It is pointed out in [11] that $r = 4$ is asymptotically optimal concerning the number of multiplications. In our example, using $h_r^{(k)}$ with $r = 4$ for $\epsilon \le 2^{-12}$ results in a large k, which requires very large multiplicative depth, thus very large HEAAN parameters. Therefore we set $r = 4$ for $\epsilon = 2^{-8}$ and $r = 7$ for $\epsilon = 2^{-12}, 2^{-16}, 2^{-20}$. Besides, we set k to be the minimum integer such that $h_r^{(k)}$ is $(\alpha, \epsilon/(2-\frac{1}{2m}))$-close to $\mathtt{sgn}(x)$.
- Using the construction in [25], $g(x)$ is defined to be the composite polynomial $g_k \circ \cdots \circ g_1$, where g_i is constructed by solving the minimax problem to $\mathtt{sgn}(x)$. For simplicity, we assume that g_i's have the same degree $d \in \{15, 31\}$, and k is set to be the minimum integer such that $g_k \circ \cdots \circ g_1$ is $(\alpha, \epsilon/(2-\frac{1}{2m}))$-close to $\mathtt{sgn}(x)$.

Based on these $g(x)$'s, we estimate the multiplicative depth and number of multiplications for evaluating $\mathtt{Round}_3(x)$ for different (α, ϵ), which are listed in Table 2.

Apply `AdaptiveLP` to $\mathtt{Round}_m(x)$. Again we focus on $m = 3$ and give concrete polynomial approximations of $\mathtt{Round}_3(x)$ via Section 4, i.e., constructing composite polynomial $f_k \circ \cdots \circ f_1$ that is (α, ϵ)-close to $\mathtt{Round}_3(x)$. In this example, the degree of f_i is set as $d = 31$ for $1 \le i \le k$, $\epsilon \in \{2^{-8}, 2^{-12}, 2^{-16}, 2^{-20}\}$, and k is set to be the minimum integer such that the approximation error rate $2^{-\alpha} < \epsilon$. We note that choosing smaller d, e.g., $d = 15$, in this example will slow down the convergence thus greatly increase the required multiplicative depth (> 100). As a result, we do not take smaller d into consideration in our implementation. For different ϵ, Table 2 lists the multiplicative depth and number of multiplications required for evaluating $\mathtt{Round}_3(x)$.

5.2 Application to the Bucketing Function

In machine learning, bucketing is usually used to map continuous data into discrete categorical values using thresholds, which can be directly viewed as a step function. For example, when training XGBoost, a gradient tree boosting model, the continuous input features can be categorized into buckets (e.g., according to percentiles) to simplify the subtree splitting operation in subsequent training. When a user's data is used by multiple models for training but the models have different granularity for bucketing(i.e. different shape of step function), a user can simply encrypt his data and let the models decide how to perform bucketing.

We consider a bucketing example that maps a latitude data $x \in (-90, 90)$ to discrete data $\{0, 1, 2\}$ for $x \in (-30, 30)$ (low latitude), $x \in (-60, -30) \cup (30, 60)$ (middle latitude), $x \in (-90, -60) \cup (60, 90)$ (high latitude) respectively.

Table 2: The multiplicative depth and number of multiplications for the evaluation of $\texttt{Round}_3(x)$ and $\kappa(x)$ using `SgnToStep` and `AdaptiveLP`. The number of iterations k is minimized such that α satisfies $2^{-\alpha} < \epsilon$.

		Evaluation of $\texttt{Round}_3(x)$				Evaluation of $\kappa(x)$			
		$\epsilon = 2^{-8}$	$\epsilon = 2^{-12}$	$\epsilon = 2^{-16}$	$\epsilon = 2^{-20}$	$\epsilon = 2^{-8}$	$\epsilon = 2^{-12}$	$\epsilon = 2^{-16}$	$\epsilon = 2^{-20}$
`SgnToStep` using [11]	k	8	9	12	14	8	9	12	14
	depth	24	36	48	56	24	36	48	56
	#Mults.	240	432	576	672	160	288	384	448
`SgnToStep` using [25] (deg = 15)	k	4	5	6	7	3	5	6	7
	depth	16	20	24	28	12	20	24	28
	#Mults.	192	240	288	336	96	160	192	224
`SgnToStep` using [25] (deg = 31)	k	3	4	5	6	3	4	5	6
	depth	15	20	25	30	15	20	25	30
	#Mults.	216	288	360	432	144	192	240	288
`AdaptiveLP` (deg = 31)	k	4	6	8	9	3	5	6	8
	depth	20	30	40	45	15	25	30	40
	#Mults.	48	72	96	108	36	60	72	96

By rescaling the data, the corresponding step function can be written as

$$\kappa(x) = \begin{cases} 0, & x \in (-\frac{1}{3}, \frac{1}{3}) \\ \frac{1}{2}, & x \in (-\frac{2}{3}, -\frac{1}{3}) \cup (\frac{1}{3}, \frac{2}{3}) \\ 1, & x \in (-1, -\frac{2}{3}) \cup (\frac{2}{3}, 1) \end{cases} . \tag{11}$$

In the following, we use `SgnToStep` and `AdaptiveLP` to construct polynomial approximations of $\kappa(x)$.

Apply `SgnToStep` to $\boldsymbol{\kappa(x)}$. Suppose $g(x)$ is a polynomial that is (α', ϵ')-close to $\texttt{sgn}(x)$ on $[-1, 1]$, then by Theorem 1 the polynomial

$$f(x) = \frac{1}{4}\left(g(\frac{3}{5}x - \frac{2}{5}) - g(\frac{3}{5}x + \frac{2}{5}) + g(\frac{3}{4}x - \frac{1}{4}) - g(\frac{3}{4}x + \frac{1}{4})\right) + 1 \tag{12}$$

is (α, ϵ)-close to $\kappa(x)$ on $[-1, 1]$, where $\alpha = \alpha'$ and $\epsilon = \frac{5}{3}\epsilon'$.

Based on the $g(x)$'s constructed in [11,25] (as chosen in Section 5.1), we estimate the multiplicative depth and number of multiplications for evaluating $\kappa(x)$ for different ϵ, which are listed in Table 2.

Apply `AdaptiveLP` to $\boldsymbol{\kappa(x)}$. According the Section 4, $\kappa(x)$ is approximated by first constructing composite polynomial $f = f_k \circ \cdots \circ f_1$ that approximates

Table 3: The multiplicative depth and number of multiplications for the evaluation of $\texttt{Round}_3(x)$ and $\kappa(x)$ using `SgnToStep` and `AdaptiveLP`, where `SgnToStep` is based on the approximation of `sgn` in [25] with the optimal number of multiplications.

			$\epsilon = 2^{-8}$	$\epsilon = 2^{-12}$	$\epsilon = 2^{-16}$	$\epsilon = 2^{-20}$
Approximate $\texttt{Round}_3(x)$	`SgnToStep` using [25] with optimal #Mults.	k	5	8	10	11
		depth	16	23	30	34
		#Mults.	102	132	180	210
	`AdaptiveLP` with degree 31	k	4	6	8	9
		depth	20	30	40	45
		#Mults.	48	72	96	108
Approximate $\kappa(x)$ in (11)	`SgnToStep` using [25] with optimal #Mults.	k	5	8	10	11
		depth	16	23	30	34
		#Mults.	68	92	120	140
	`AdaptiveLP` with degree 31	k	3	5	6	8
		depth	15	25	30	40
		#Mults.	36	60	72	96

the normalization of $\kappa(x)$, i.e.,

$$\tilde{\kappa}(x) = \begin{cases} -1 & x \in (-1, -2/3) \\ -1/2 & x \in (-2/3, -1/3) \\ 0 & x \in (-1/3, 1/3) \\ 1/2 & x \in (1/3, 2/3) \\ 1 & x \in (2/3, 1) \end{cases},$$

then constructing $g(x)$ such that $g \circ f$ approximates $\kappa(x)$. Similarly, we assume that f_i's have the same degree $d = 31$, g has degree $d_g = 31$, $\epsilon \in \{2^{-8}, 2^{-12}, 2^{-16}, 2^{-20}\}$, and k is the minimum integer such that the approximation error rate $2^{-\alpha} < \epsilon$. Table 2 lists the multiplicative depth and number of multiplications required for evaluating $\kappa(x)$ for different ϵ.

We note that the optimal number of multiplications for approximating `sgn` obtained by the dynamic programming approach is given in [25]. We also give a comparison of `SgnToStep` based on these approximations and `AdaptiveLP` with degree $= 31$ in Table 3.

6 Experimental Results

This section presents some experimental results of homomorphically evaluating the step functions in Section 5. The computation in this section is performed using the HEAAN library on a Linux PC with an Intel Core i9 CPU at 3.00GHz.

6.1 CKKS FHE Scheme

CKKS scheme is introduced by Cheon et al. in [10], which enables approximate homomorphic arithmetic computation over real/complex numbers. We denote λ as the security parameter of CKKS, which is usually set to 128. Let $L \in \mathbb{Z}^+$ denote the bit-length of the initial ciphertext modulus, and define $q_l := 2^l$ for $1 \leq l \leq L$. We denote χ_s, χ_e, χ_r as the distribution of secret, error, and encryption respectively. Let $N \in \mathbb{Z}$ be a power of 2. Denote $R = \mathbb{Z}[X]/(X^N + 1)$ be the $2N$-th cyclotomic ring, and $R_q := R/qR$ for an integer $q > 1$. The isomorphism $\tau : \mathbb{R}[X]/(X^N+1) \to \mathbb{C}^{\frac{N}{2}}$ is used for encoding and decoding of plaintexts. Let Δ denote the scaling factor. The CKKS scheme contains the following algorithms.

- $\texttt{KeyGen}(1^\lambda)$:
  Sample $s \leftarrow \chi_s$, $a \leftarrow \mathcal{U}(R_{q_L})$, $e \leftarrow \chi_e$, $e' \leftarrow \chi_e$ and $a' \leftarrow \mathcal{U}(R_{q_L^2})$;
  Output $\texttt{sk} = (1, s)$, $\texttt{pk} = (-as + e, a) \in R^2_{q_L}$ and $\texttt{evk} = (-a' \cdot s + e' + q_L \cdot s^2, a') \in R^2_{q_L^2}$.
- $\texttt{Enc}_{\text{pk}}(m; \Delta)$:
  For a plaintext $m \in \mathbb{C}^{\frac{N}{2}}$, compute $\mathfrak{m} = \lfloor \Delta \cdot \tau^{-1}(m) \rceil$, sample $v \leftarrow \chi_r$ and $e_0, e_1 \leftarrow \chi_e$;
  Output $v \cdot \texttt{pk} + (m + e_0, e_1) \mod q_L$.
- $\texttt{Dec}_{\text{sk}}(\texttt{ct}; \Delta)$:
  For a ciphertext $\texttt{ct} = (c_0, c_1) \in R^2_{q_l}$, compute $\mathfrak{m}' = c_0 + c_1 \cdot s \mod q_l$;
  Output $m' = \frac{1}{\Delta} \cdot \tau(\mathfrak{m}')$.
- $\texttt{Add}(\texttt{ct}_1, \texttt{ct}_2)$:
  For $\texttt{ct}_1, \texttt{ct}_2 \in R^2_{q_l}$, output $\texttt{ct}_{\text{add}} = \texttt{ct}_1 + \texttt{ct}_2 \mod q_l$.
- $\texttt{Mult}_{\text{evk}}(\texttt{ct}_1, \texttt{ct}_2)$:
  For $\texttt{ct}_1 = (b_1, a_1), \texttt{ct}_2 = (b_2, a_2) \in R^2_{q_l}$, let $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2)$, compute $\texttt{ct}'_{\text{mult}} \leftarrow$
  $(d_0, d_1) + \lfloor q_L^{-1} \cdot d_2 \cdot \texttt{evk} \rceil \mod q_l$;
  Output $\texttt{ct}_{\text{mult}} = \lfloor \Delta^{-1} \cdot \texttt{ct}'_{\text{mult}} \rceil \mod (q_l/\Delta)$.

Note that we can deal with $\frac{N}{2}$ encrypted data in a SIMD manner, so the amortized running time is $\frac{2}{N}$ times the total time.

6.2 Parameters Setting

In our experiment, we set $N = 2^{17}$, and the highest level modulus q_L upto 2^{1700} to achieve 128-bit security estimated by Albrecht's LWE estimator [3,2]. The scaling factor is set to $\Delta = 2^{40}$. Besides, we expect the final modulus after evaluation to be $\log \Delta + 10$ bits long. Then the initial modulus q_L is given as follows.

- For $\texttt{SgnToStep}$, suppose g is the approximate polynomial of $\texttt{sgn}$ used in our construction, then

$$\log q_L = \log \Delta \cdot (\texttt{dep}(g)) + \log \Delta + 10.$$

Table 4: Running time and depth consumption of $\texttt{Round}_3(x)$ and $\kappa(x)$ in HEAAN.

		Evaluation of $\texttt{Round}_3(x)$				Evaluation of $\kappa(x)$			
		$\epsilon = 2^{-8}$	$\epsilon = 2^{-12}$	$\epsilon = 2^{-16}$	$\epsilon = 2^{-20}$	$\epsilon = 2^{-8}$	$\epsilon = 2^{-12}$	$\epsilon = 2^{-16}$	$\epsilon = 2^{-20}$
`SgnToStep` using [11]	k	8	9	12	14	8	9	12	14
	running time	4.63 ms	7.83 ms	13.24 ms	16.32 ms	3.08 ms	4.73 ms	9.38 ms	10.93 ms
	bit consumption	1360	1520	2000*	2320*	1360	1520	2000*	2320*
`SgnToStep` using [25] (deg = 15)	k	4	5	6	8	3	5	6	7
	running time	2.56 ms	3.39 ms	4.17 ms	5.75 ms	1.20 ms	2.26 ms	2.68 ms	3.21 ms
	bit consumption	716	875	1034	1352	557	875	1034	1193
`SgnToStep` using [25] (deg = 31)	k	3	4	5	6	3	4	5	6
	running time	3.01 ms	4.36 ms	6.18 ms	8.09 ms	2.21 ms	3.41 ms	4.11 ms	5.32 ms
	bit consumption	688	891	1094	1297	688	891	1094	1297
`AdaptiveLP` (deg = 31)	k	4	6	8	10	4	6	8	10
	running time	0.74 ms	1.36 ms	1.98 ms	2.81 ms	0.78 ms	1.24 ms	2.15 ms	2.95 ms
	bit consumption	808	1212	1616	2020*	808	1212	1616	2020*

* an asterisk $(*)$ means the parameter set does not achieve 128−bit security for large $\log q_L \geq 1700$ in HEAAN.

– For `AdaptiveLP`, suppose $g \circ f_k \circ \cdots \circ f_1$ is the composite polynomial in our construction, then

$$\log q_L = \log \Delta \cdot (\texttt{dep}(f_1) + \cdots + \texttt{dep}(f_k) + \texttt{dep}(g)) + \log \Delta + 10.$$

Here $\texttt{dep}(\cdot)$ denotes the depth consumption for evaluating a polynomial, which is usually set to be $\lceil \log(d+1) \rceil$ for a polynomial of degree d.

6.3 Evaluating $\texttt{Round}_3(x)$

We evaluate the step function $\texttt{Round}_3(x)$ based on the polynomial approximations constructed in Section 5. To handle the homomorphic evaluation error, we construct composite polynomials by introducing the error bound η as in Algorithm 2, where η is dynamically determined by the coefficient bound in our implementation. The polynomials are evaluated by the BSGS method as in [27], and the amortized running time and bit consumption $\log(\frac{q_L}{q_l})$ for different approximation error rates are listed in Table 4. In the table, we set $\epsilon = 2^{-8}, 2^{-12}, 2^{-16}, 2^{-20}$ and let the number of iterations k to be minimum such that $2^{-\alpha} < \epsilon$.

Through the table we can see that `SgnToStep` shows an advantage in bit consumption, while `AdaptiveLP` provides better performance in amortized running time. For example, comparing with `SgnToStep` that uses the minimax approximation in [25] with polynomial degree 15, `AdaptiveLP` has roughly 1.5× bit consumption but approximately 0.5× running time. Though each evaluation of `sgn` requires less bit consumption and less running time than `AdaptiveLP`, the evaluation of $\texttt{Round}_3(x)$ based on `SgnToStep` involves 6 evaluations of `sgn` thus requires more running time.

6.4 Evaluating Bucketing Function

We evaluate the bucketing example given in (11) based on the polynomial approximations given in Section 5. Again, the dynamic error bound is used to handle the homomorphic evaluation error, and BSGS method is used to evaluate the polynomials. We set $\epsilon = 2^{-8}, 2^{-12}, 2^{-12}, 2^{-16}$ and let the number of iterations k to be minimum such that $2^{-\alpha} < \epsilon$. The amortized running time and bit consumption $\log(\frac{q_L}{q_l})$ for different approximation error rates are listed in Table 4.

Through the table we can see that `AdaptiveLP` still outperforms `SgnToStep` in amortized running time. However, because the bucketing example given in (11) has a less number of intervals $n = 5$, `SgnToStep` requires only 4 evaluations of `sgn`. As a result, `AdaptiveLP` shows less advantage in running time. In general, `SgnToStep` and `AdaptiveLP` provide a trade-off in terms of running time and bit consumption.

References

1. Akavia, A., Leibovich, M., Resheff, Y.S., Ron, R., Shahar, M., Vald, M.: Privacy-preserving decision trees training and prediction. ACM Trans. Priv. Secur. **25**(3), 24:1–24:30 (2022). https://doi.org/10.1145/3517197, https://doi.org/10.1145/3517197
2. Albrecht, M.R.: On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. pp. 103–129. https://doi.org/10.1007/978-3-319-56614-6_4
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015), http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml
4. Andrievskii, V.: Polynomial approximation of piecewise analytic functions on a compact subset of the real line. J. Approx. Theory **161**(2), 634–644 (2009). https://doi.org/10.1016/j.jat.2008.11.015, https://doi.org/10.1016/j.jat.2008.11.015
5. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. pp. 483–512. https://doi.org/10.1007/978-3-319-96878-0_17
6. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. pp. 868–886. https://doi.org/10.1007/978-3-642-32009-5_50
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. pp. 309–325. https://doi.org/10.1145/2090236.2090262
8. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. pp. 34–54. https://doi.org/10.1007/978-3-030-17656-3_2
9. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. pp. 360–384. https://doi.org/10.1007/978-3-319-78381-9_14
10. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. pp. 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

11. Cheon, J.H., Kim, D., Kim, D.: Efficient homomorphic comparison methods with optimal complexity. pp. 221–256. https://doi.org/10.1007/978-3-030-64834-3_8
12. Cheon, J.H., Kim, D., Kim, D., Lee, H.H., Lee, K.: Numerical method for comparison on homomorphically encrypted numbers. pp. 415–445. https://doi.org/10.1007/978-3-030-34621-8_15
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. pp. 3–33. https://doi.org/10.1007/978-3-662-53887-6_1
14. Cohen, M.B., Lee, Y.T., Song, Z.: Solving linear programs in the current matrix multiplication time. pp. 938–942. https://doi.org/10.1145/3313276.3316303
15. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. pp. 617–640. https://doi.org/10.1007/978-3-662-46800-5_24
16. Eremenko, A., Yuditskii, P.: Uniform approximation of sgn x by polynomials and entire functions. Journal d'Analyse Mathématique **101**(1), 313–324 (2007)
17. Eremenko, A., Yuditskii, P.: Polynomials of the best uniform approximation to sgn (x) on two intervals. Journal d'Analyse Mathématique **114**(1), 285–315 (2011)
18. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144
19. Gentry, C.: Fully homomorphic encryption using ideal lattices. pp. 169–178. https://doi.org/10.1145/1536414.1536440
20. Gentry, C.: Computing arbitrary functions of encrypted data. Commun. ACM **53**(3), 97–105 (2010). https://doi.org/10.1145/1666420.1666444, https://doi.org/10.1145/1666420.1666444
21. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. pp. 75–92. https://doi.org/10.1007/978-3-642-40041-4_5
22. Han, K., Hong, S., Cheon, J.H., Park, D.: Logistic regression on homomorphic encrypted data at scale. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 9466–9471. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33019466, https://doi.org/10.1609/aaai.v33i01.33019466
23. Jutla, C.S., Manohar, N.: Sine series approximation of the mod function for bootstrapping of approximate HE. pp. 491–520. https://doi.org/10.1007/978-3-031-06944-4_17
24. Kim, M., Song, Y., Li, B., Micciancio, D.: Semi-parallel logistic regression for GWAS on encrypted data. Cryptology ePrint Archive, Report 2019/294 (2019), https://eprint.iacr.org/2019/294
25. Lee, E., Lee, J.W., Kim, Y.S., No, J.S.: Minimax approximation of sign function by composite polynomial for homomorphic comparison. IEEE Transactions on Dependable and Secure Computing (2021)
26. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. pp. 618–647. https://doi.org/10.1007/978-3-030-77870-5_22
27. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. pp. 551–580. https://doi.org/10.1007/978-3-031-06944-4_19

28. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. Cryptology ePrint Archive, Report 2021/1337 (2021), https://eprint.iacr.org/2021/1337
29. jie Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. pp. 1057–1073. https://doi.org/10.1109/SP40001.2021.00043
30. Panda, S.: Polynomial approximation of inverse sqrt function for FHE. In: Dolev, S., Katz, J., Meisels, A. (eds.) Cyber Security, Cryptology, and Machine Learning - 6th International Symposium, CSCML 2022, Be'er Sheva, Israel, June 30 - July 1, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13301, pp. 366–376. Springer (2022). https://doi.org/10.1007/978-3-031-07689-3_27, https://doi.org/10.1007/978-3-031-07689-3_27
31. Plaskota, L., Wasilkowski, G.W.: Uniform approximation of piecewise r-smooth and globally continuous functions. SIAM J. Numer. Anal. **47**(1), 762–785 (2008). https://doi.org/10.1137/070708937, https://doi.org/10.1137/070708937
32. Plaskota, L., Wasilkowski, G.W., Zhao, Y.: The power of adaption for approximating functions with singularities. Math. Comput. **77**(264), 2309–2338 (2008). https://doi.org/10.1090/S0025-5718-08-02103-0, https://doi.org/10.1090/S0025-5718-08-02103-0
33. Saff, E.B., Totik, V.: Polynomial approximation of piecewise analytic functions. Journal of the London Mathematical Society **2**(3), 487–498 (1989)
34. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptogr. **71**(1), 57–81 (2014). https://doi.org/10.1007/s10623-012-9720-4, https://doi.org/10.1007/s10623-012-9720-4
35. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. pp. 24–43. https://doi.org/10.1007/978-3-642-13190-5_2