MSc Computer Science
Master Thesis

Towards Efficient Order Statistics in CKKS: A Study of Robustness and Efficiency Through Non-Continuous Function Approximation

Vasco Rikkers

Supervisors: Federico Mazzone & Florian W. Hahn

February, 2025

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Abstract

This thesis explores methods to improve ranking, order statistics, and sorting algorithms within the CKKS encryption scheme, with a focus on approximating discontinuous functions such as the sign function. Fully Homomorphic Encryption (FHE) ensures data privacy by enabling computations directly on encrypted data, but its high computational complexity poses significant challenges. To address these challenges, this study analyzes the balance between accuracy and computational efficiency in two key approximation techniques: the Tchebycheff and Composite Minimax approximation algorithms. Our experimental results show that composite minimax polynomials outperform polynomials created using Tchebycheff approximation in memory usage and computational efficiency, making them more suitable for high-performance applications. To increase their robustness against approximation errors, this thesis also presents a revised algorithm for determining the (arg)min and (arg)max of a vector, which substitutes the usage of the comparison function with the usage of the max or min function. Our findings indicate that using the max or min function instead of the comparison function improves robustness against approximation errors when determining the smallest value in a vector. However, the opposite is the case when computing the argmin, as then the robustness decreases. These results contribute to the development of more robust and efficient privacy-preserving algorithms for the CKKS encryption scheme, with potential applications in secure cloud computing, encrypted machine learning, and privacy-conscious data analysis.

1 Introduction

Fully Homomorphic Encryption (FHE) has been a significant advancement in cryptography, as it enables computations to be done on encrypted data without ever needing to decrypt it. This feature ensures data privacy and secrecy even when the computational infrastructure cannot be entirely trusted. However, FHE faces notable challenges, as its significant computational overhead limits the complexity and efficiency of algorithms that can be executed on encrypted data.

FHE is increasingly relevant in privacy-preserving applications, including encrypted machine learning, secure cloud computing, and private data analysis.[15, 10]. It allows for the creation of various algorithms essential for these applications, such as sorting, ranking, and calculating order statistics under FHE. These algorithms are crucial for programs to perform private data aggregation, secure collaborative filtering, and encrypted database management [17]. The secure and efficient implementation of these processes is crucial for real-world usability. However, using non-continuous and non-polynomial functions in these algorithms, such as comparisons and sign functions, can pose a significant challenge.

The sign function and comparison operations are essential building blocks at the core of the algorithms for calculating order statistics and ranking. The sign function determines whether a given input number is positive, zero, or negative. When used correctly, this function can support calculating other operations, such as the comparison operation that identifies the larger of two numbers. Together with the basic operations in the FHE schemes, these functions provide the foundation for creating more advanced algorithms, such as those used for order statistics and ranking.

The Cheon-Kim-Kim-Song (CKKS) scheme is an FHE scheme that is popular due to its support for floating-point numbers and SIMD-based (Single Instruction, Multiple Data) parallel data processing [9]. CKKS is particularly well-suited for machine learning and data analytics tasks, which suffice with using approximate arithmetic. However, CKKS has limitations: it supports only addition and multiplication on encrypted data, so non-polynomial functions such as the sign function must be approximated using polynomials. This constraint emphasizes the necessity of effective approximation techniques to address the challenges posed by non-continuous functions.

Since sign and comparison functions are crucial for order statistics under FHE, they must be approximated using polynomials to be evaluated under CKKS. However, these functions are non-polynomial, so approximation techniques must be used that balance efficiency and accuracy. A higher level of accuracy in the approximation means that the resulting polynomial will more accurately represent the original function. However, this also requires more computational resources for computing and evaluating the polynomial.

The depth of a polynomial is a key factor in determining its computational efficiency. Depth refers to the number of sequential operations needed to evaluate a polynomial over encrypted data, and a greater depth often results in a more accurate approximation. However, a greater depth also means more resources are required. Therefore, reducing the required depth while maintaining accuracy is crucial for improving the usability of applications that rely on FHE.

Approximating a non-polynomial function with a polynomial function often introduces errors. These errors, known as approximation errors, are the difference between the polynomial approximation and the target function. They are most noticeable near areas of discontinuity in a function and can significantly affect the accuracy of computations. Therefore, it is important to consider the robustness of an algorithm or computation, which refers to the impact of approximation errors near non-continuous regions on the final output. For example, in Subsection 4.1, we will see that the sign function can be used to calculate both the comparison and max functions. However, an approximation error of 0.5 will have a greater impact on the final result when the sign function is used to calculate the comparison function than when it is used to

calculate the max function. The more robust an algorithm or computation is, the less these errors will affect the final result.

Different approaches to approximating non-polynomial functions have been investigated in this field. Cheon et al. [7] proposed methods for balancing accuracy and efficiency in the approximation of the sign function. Lee et al. [21] introduced a minimax approximation algorithm to optimize computation depth and accuracy. Meanwhile, other studies have optimized algorithms for sorting and order statistics under CKKS through techniques such as k-way networks [19], bitonic sort [23], and parallel comparisons [24]. Despite these developments, less is known about robustness against approximation errors.

Considering these challenges and opportunities, we start with the work of Mazzone et al. [24], who propose state-of-the-art algorithms for ranking, order statistics, and sorting under CKKS. Their approach relies on the Tchebycheff approximation algorithm [25] to approximate the comparison function. However, more recent methods, such as the Composite Minimax approximation algorithm introduced by Lee et al. [21], claim to offer greater efficiency. In addition, we have identified that the robustness of their order statistics algorithms can be improved by using the max function instead of the comparison function. Building on these insights, this work investigates the following research questions:

1. To what extent is it possible to improve the algorithms using the Composite Minimax approximation of the homomorphic comparison operation?

2. Knowing the max function is more robust than the comparison function, to what extent can it be used to improve the robustness of the algorithms?

Our first goal is to improve the algorithms for ranking, sorting, and order statistics under CKKS by implementing the Composite Minimax approximation algorithm to approximate the comparison function, as proposed by Lee et al. [21]. This approach aims to improve computational efficiency and reduce the circuit depth of these algorithms. Additionally, it is expected to lower approximation errors compared to polynomials generated using the Tchebycheff method.

The composite minimax polynomials will be evaluated over encrypted data using the OpenFHE library, replacing the SEAL library used in the original work. This update ensures compatibility with the implementation of the algorithms by Mazzone et al. [24], which also relies on OpenFHE. OpenFHE is well-suited for their algorithms due to features such as hardware acceleration support [1]. Moreover, it offers a user-friendly interface with functionalities like automatic key-switching, making it easier to work with than other libraries.

Furthermore, we propose the use of the max function as a robust alternative to the comparison function for calculating order statistics. Its continuity helps minimize the impact of approximation errors, resulting in more accurate computations. This approach is expected to significantly improve the robustness of these algorithms.

Additionally, this work includes an open-source pipeline with complete implementations for determining optimal polynomial degrees, approximating the sign function using composite polynomials, and evaluating these approximations over encrypted data using CKKS. The pipeline also includes code for calculating the minimum of a vector using our improved algorithm. Ultimately, the goal of this work is to enhance the usability of FHE-based, privacy-preserving order statistics in machine learning and encrypted database applications.

2 Background

This section serves as an introduction to the core concepts and methodologies that are central to this thesis. The aim is to provide a comprehensive and concise overview of the fundamental principles, notations, and prior research related to FHE, the CKKS scheme, polynomial approximation, and the algorithms for ranking, sorting, and order statistics created by Mazzone et al. [24]. These topics form the theoretical and practical foundation upon which the contributions of this work are built.

In Subsection 2.1, we introduce the mathematical and algorithmic foundations for the thesis. This is followed by an explanation of the notation used for vectors and matrices in Subsection 2.2. We then discuss the basics of FHE in Subsection 2.3. This subsection also introduces CKKS, an FHE scheme that allows for approximate arithmetic on encrypted real numbers. Next, we delve into the theory and application of polynomial approximation in Subsection 2.4, an essential mathematical tool used throughout this thesis to perform non-polynomial computations within the encrypted domain. Finally, in Subsection 2.5, we discuss the algorithms for ranking and order statistics designed by Mazzone et al..

By the end of this section, the reader will have a solid understanding of these topics, enabling them to comprehend the subsequent chapters and appreciate the significance of the thesis contributions within the broader research landscape.

2.1 Mathematical and Algorithmic Definitions

This thesis references a set of mathematical functions, formally defined and explained below. Each function is presented with a graphical representation for better understanding.

Sign Function: The sign function, $\text{sign}(x)$, indicates whether a given number is positive, negative, or zero. Formally, it is defined as:

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

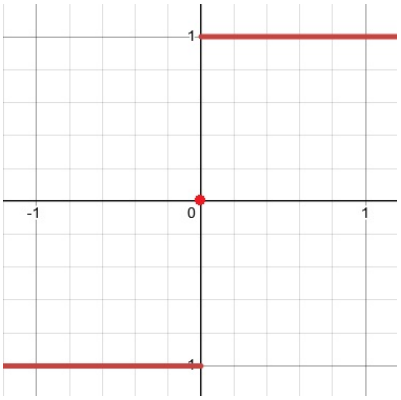

FIGURE 1: Graphical representation of the sign function.

Indicator Function: The indicator function, $\mathrm{Ind}_{[a,b]}(x)$, determines if a given number lies within a specified range. Formally, it is defined as:

$$\mathrm{Ind}_{[a,b]}(x) = \begin{cases} 1 & \text{if } a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases}$$

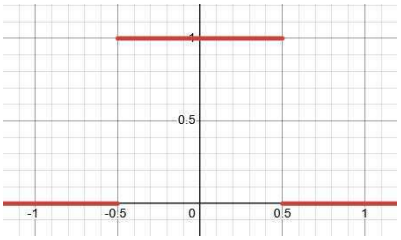

FIGURE 2: Graphical representation of the indicator function with $a = -0.5$ and $b = 0.5$.

Comparison Function: The comparison function, $\mathrm{cmp}(u, v)$, compares two input values and determines their relative magnitude. Formally, it is defined as:

$$\mathrm{cmp}(u, v) = \begin{cases} 0 & \text{if } v > u, \\ \frac{1}{2} & \text{if } v = u, \\ 1 & \text{if } v < u. \end{cases}$$

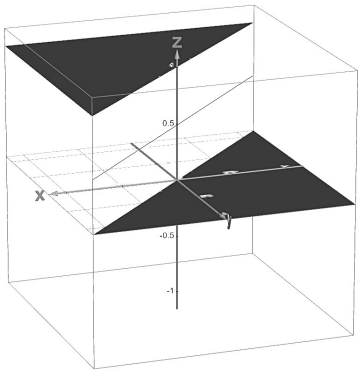

FIGURE 3: Graphical representation of the comparison function.

Max Function: The max function, $\max(u, v)$, returns the larger of two input values. Formally, it is defined as:

$$\max(u, v) = \begin{cases} v & \text{if } v \geq u, \\ u & \text{if } u > v. \end{cases}$$

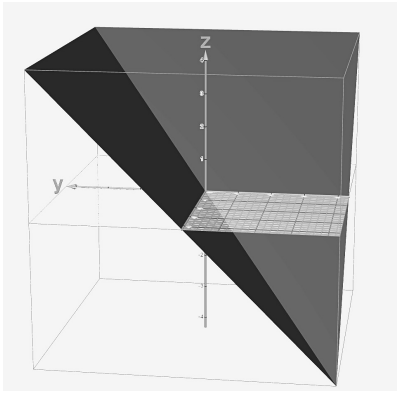

FIGURE 4: Graphical representation of the max function.

Min Function: The min function, $\min(u, v)$, returns the smaller of two input values. Formally, it is defined as:

$$\min(u, v) = \begin{cases} v & \text{if } v \leq u, \\ u & \text{if } u < v. \end{cases}$$

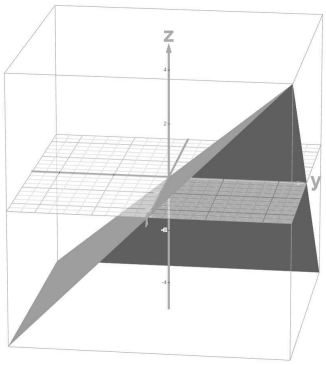

FIGURE 5: Graphical representation of the min function.

Composite Polynomials: Composite polynomials are polynomials of the form:

$$P(x) = P_1(P_2(\cdots P_k(x)\cdots)),$$

Used to enhance approximation accuracy and computational efficiency.

Order Statistics: The k-th order statistic is the k-th smallest value in a dataset. Examples include the minimum and maximum.
Ranking and Sorting: Ranking assigns a position to each element based on its value while sorting rearranges elements in ascending or descending order.
Min of a vector: The min of a vector is the value of the smallest element in that vector.
Argmin of a vector: The argmin of a vector is the index of the smallest value in that vector.

Finally, in Table 1 you will find the specific notations that are used throughout the thesis to represent, e.g., implementations of the different functions using the approximation methods.

2.2 Vector and Matrix Notation

Let $\vec{v}$ represent a plaintext vector. Each element in this vector corresponds to a slot in CKKS encryption, which is an important concept in understanding how data is organized and processed within the encryption scheme. This will be explained in more detail in Subsection 2.3.

Many of the algorithms presented in this paperwork work with one-dimensional vectors. For clarity, these vectors are often depicted as two-dimensional matrices when we explain these algorithms. For instance, the vector

$$v = (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9)$$

would be row-wise encoded as the following matrix:

$$\begin{bmatrix} v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 \\ v_7 & v_8 & v_9 \end{bmatrix}$$

This representation helps us to understand the structure and manipulation of the data during algorithmic processes.

Term	Definition
sign^{C}	The sign function approximated using the minimax composite polynomial approximation algorithm.
sign^{T}	The sign function approximated using the Tchebycheff approximation algorithm.
$\max^{\text{C}}$	The maximum function implemented using sign^{C}.
cmp^{C}	The comparison function implemented using sign^{C}.
$\min_v^{comp}$	Algorithm for calculating the min of a vector designed by Mazzone et al. that uses the comparison function.
argmin_v^{comp}	Algorithm for calculating the argmin of a vector designed by Mazzone et al. that uses the comparison function.
$\min_v^{min}$	Algorithm for calculating the min of a vector designed that uses the min function.
argmin_v^{min}	Algorithm for calculating the argmin of a vector designed that uses the min function.

TABLE 1: Table displaying the notation and definition of different terms used in the paper.

2.3 FHE and CKKS

FHE is a type of encryption that supports arbitrary computations on ciphertexts. This means that a user can perform complex operations on encrypted data and obtain an encrypted result, which, when decrypted, matches the result of performing the same operations on the plaintext data [15]. This allows FHE to protect data privacy during the computing process.

First, we will discuss some basic terms related to FHE that are needed to understand the discussions in the rest of the thesis.

- *Plaintext*: The original, unencrypted data.
- *Ciphertext*: The encrypted representation of plaintext data.
- *Homomorphic Operations*: Operations such as addition and multiplication that can be performed directly on ciphertexts in an FHE scheme.
- *Circuit*: A sequence of homomorphic multiplications done on encrypted data
- *Depth Consumption*: The number of sequential homomorphic operations in a circuit

FHE schemes typically combine the following set of properties that describe its characteristics:

- **Additive Homomorphism**: The scheme supports addition operations on ciphertexts, corresponding to addition on plaintexts.
- **Multiplicative Homomorphism**: The scheme supports multiplication operations on ciphertexts, corresponding to multiplication on plaintexts.
- **Arbitrary Computation**: By enabling both addition and multiplication, FHE supports any computation expressible as a circuit of these basic operations. For example, if we can express or approximate a function using a polynomial, we can compute it over encrypted data.

The underlying mathematical structures and methods used to guarantee security, and homomorphic characteristics make FHE computationally demanding despite its potential [18]. More on this will follow later when we discuss depth consumption.

One kind of FHE made to facilitate approximations on real or complex numbers is the CKKS encryption method [10]. CKKS was first introduced in 2017 to solve the problem of arithmetic on non-integer data, which is essential for applications such as signal processing and machine learning.

By encoding real numbers as polynomial coefficients, CKKS makes it possible to do calculations on encrypted approximate values. Among its notable characteristics are:

- **Approximate Arithmetic**: Instead of exact results, CKKS produces approximate results that are sufficient for practical purposes.
- **Efficient Encoding and Decoding**: CKKS maps real numbers to a polynomial ring, allowing efficient arithmetic operations within this mathematical framework.
- **Support for Batch Operations**: The scheme supports batching, enabling the encryption of multiple plaintext values into a single ciphertext, which can then be processed simultaneously.

As previously mentioned, the CKKS scheme uses slots to store individual complex or real numbers within a plaintext vector. This scheme also supports packed encryption, allowing for multiple values to be encoded and encrypted into a single ciphertext. These values are organized into slots within the ciphertext. For example, if we have a plaintext vector $\vec{v} = (v_0, v_1, \ldots, v_{n-1})$, each v_i will occupy a specific slot when encrypted using CKKS. The number of available slots is determined by the ring dimension, denoted as N in CKKS. Since a ciphertext in CKKS is represented as a polynomial, the number of slots is typically $N/2$ to accommodate the packed encoding of real or complex numbers. A larger ring dimension allows for more slots, enabling the encryption of larger vectors or more data points simultaneously. This is crucial for efficiency when using homomorphic operations.

The concept of slots is crucial in enhancing the efficiency and capability of FHE systems, particularly in the CKKS scheme. Slots allow for parallelism by enabling operations such as addition, multiplication, and scaling to be applied simultaneously across multiple elements within a ciphertext, resulting in improved computational efficiency. This structure also supports packing, where multiple data points are encoded into a single ciphertext, reducing the overhead of repeated encryptions. The ability to operate on multiple slots simultaneously is known as Single Instruction, Multiple Data (SIMD), a key feature in modern FHE systems like CKKS. SIMD is achieved through batching, which utilizes packed data to facilitate parallel processing and reduce encryption and decryption overhead. This capability makes SIMD particularly advantageous for data-intensive tasks in areas such as machine learning, signal processing, and statistical analysis, where vectorized operations are common. Overall, the combination of slots and SIMD enables CKKS to efficiently handle large datasets while maintaining privacy, making it a powerful tool for real-world applications.

To further understand CKKS, it is essential to examine its core algorithms [4, 20]:

- **Gen**$(\lambda) \rightarrow (pk, sk, evk)$: Generates a public key (pk), secret key (sk), and evaluation key (evk) based on the security parameter λ.
- **Enc**$(m, pk) \rightarrow c$: Encrypts a message m using the public key pk, resulting in ciphertext c.
- **Dec**$(c, sk) \rightarrow m$ or $\perp$: Decrypts a ciphertext c with the secret key sk to recover the message m. If decryption fails, it returns $\perp$.
- **Add**$(c_1, c_2, evk) \rightarrow c_{add}$: Adds two ciphertexts c_1 and c_2 (representing messages m_1 and m_2), producing c_{add} where $\text{Dec}(c_{add}) = m_1 + m_2$.
- **Mult**$(c_1, c_2, evk) \rightarrow c_{mult}$: Multiplies two ciphertexts c_1 and c_2 (representing m_1 and m_2), producing c_{mult} where $\text{Dec}(c_{mult}) = m_1 \cdot m_2$.
- **Rotate**$(c, k, evk) \rightarrow c_{rotated}$: Rotates a ciphertext c by k positions in its underlying vector, producing $c_{rotated}$. This can be done to the left or right.

Understanding how the rotation operation works can be a bit tricky without a visual example. Therefore, we have created an example below that shows how a rotation to the left on a 3×3 matrix works. A rotation to the right works the same but just moves in another direction.

$$\begin{bmatrix} v_1 & 0 & 0 \\ v_2 & 0 & 0 \\ v_3 & 0 & 0 \end{bmatrix} \xrightarrow{\text{Rotate(c,2,evk)}} \begin{bmatrix} 0 & 0 & v_2 \\ 0 & 0 & v_3 \\ 0 & 0 & v_1 \end{bmatrix}$$

CKKS encryption is well-suited for privacy-preserving computations, where small deviations in numerical accuracy are acceptable in exchange for improved computational efficiency. For instance, in healthcare predictive analytics, hospitals can collaborate on training encrypted predictive models for disease diagnosis without directly sharing sensitive patient data. We can do this using CKKS because machine learning models, such as neural networks, are tolerant of minor numerical errors. Slight variations in weights or activations do not significantly impact the final prediction, making CKKS a suitable choice for privacy-preserving inference without compromising model performance [16]. However, training a model like a neural network can be a time-consuming and resource-intensive process, making it crucial to focus on the improvement of its efficiency. This highlights the capability of CKKS to enable secure and efficient data analysis while maintaining privacy.

The number of consecutive homomorphic operations that can be performed on ciphertexts without the need for decryption or re-encryption is known as depth consumption. The re-encryption of a ciphertext, called bootstrapping, is necessary because homomorphic operations on CKKS ciphertexts increase the amount of noise within the ciphertexts. CKKS encryption produces approximate ciphertexts, meaning that the decrypted result is an approximation of the original plaintext. The level of noise in the ciphertext determines the accuracy of this approximation. As the noise increases, it can reach a point where decrypting the ciphertext fails to produce a meaningful plaintext, resulting in incorrect or unusable outputs. In other words, the depth of a circuit refers to the maximum number of consecutive homomorphic operations that can be performed before the ciphertext becomes too noisy to handle without losing information.

When encrypting data in CKKS, we specify a maximum depth usage for the resulting ciphertext. More sequential operations and more intricate calculations are made possible when using a greater depth, but the increased complexity of the operations and the need for noise management result in longer processing times and larger resource consumption. In conclusion, shallow circuits execute faster with lower overhead but may limit computational capability, while deeper circuits enable complex calculations at the cost of increased noise management and processing time.

Using the Residue Number System (RNS) for effective arithmetic operations, the RNS-based CKKS (RNS-CKKS) scheme is an improvement on the CKKS encryption scheme [11]. RNS breaks down numbers into residues regarding a set of pairwise

coprime moduli. Consider the number 23 and a set of pairwise coprime moduli 5, 7, and 11. The residues are calculated to be 3, 2, and 1 respectively. So, in RNS, the number 23 is represented as the set of coprime residues $(3, 2, 1)$. This representation allows arithmetic operations to be performed independently on each residue, enabling parallel computation and reducing overhead.

Key benefits of RNS-CKKS include:

- **Parallelism**: Computations on residues can be executed independently, leading to substantial performance gains.
- **Reduction in Modulus Switching Costs**: RNS-CKKS minimizes the computational cost of modulus switching, a critical operation in FHE schemes.
- **Scalability**: The approach scales efficiently with larger problem sizes and more complex computations.

2.4 Approximation of Non-Polynomial Functions

Evaluation of non-polynomial functions, such as logarithms, exponentials, or trigonometric functions, on encrypted data, is necessary for the application of CKKS to many real-world scenarios. However, like most homomorphic encryption systems, the CKKS scheme only allows for homomorphic addition and multiplication. This limitation requires us to express non-polynomial functions using polynomials to evaluate them on encrypted data. To overcome this challenge, we can use the mathematical principle of approximation. This involves replacing a complex function defined over a specific interval with one or more polynomials that closely mimic its behavior, as shown in Figure 6.

While no approximation can perfectly replicate the original function, the goal is to minimize the error between the function and its polynomial representation. One way to achieve this, as seen in Figure 6, is by increasing the degree of the approximating polynomial. This allows for a better capture of the nuances of the function. However, this comes at a cost as polynomials with a higher degree require more computational resources, both in terms of time and space, especially when used in homomorphic computations. When evaluating a polynomial over encrypted data, more accurate approximations typically demand a greater circuit depth, resulting in the need for more homomorphic operations. Therefore, balancing the trade-off between accuracy and efficiency is a critical aspect of designing an effective approximation algorithm.

When talking about approximation methods, there are a few ways you can discuss how well such a method works. One of the first metrics, and often the main one, that you can look at is the difference between a function and its approximating polynomial. The approximation error is defined as the absolute difference between the actual function, $f(x)$, and its approximation, $\tilde{f}(x)$. More formally, the error is given by:

$$\text{Error}(x) = \left| f(x) - \tilde{f}(x) \right|.$$

This metric allows us to measure how well the approximation aligns with the true function. It is crucial when discussing the final accuracy of algorithms that $\tilde{f}(x)$ will be used in.

Another key performance metric for approximation algorithms, as well as any other algorithm performed on encrypted data, is time complexity. Time complexity, often expressed in big-O notation, describes the computational complexity of an algorithm. It measures the performance of an algorithm by estimating how the computation time increases as variables such as the input size change, which is vital for understanding the scalability of algorithms. Time consumption, on the other hand, refers to the actual time it takes for a task to be completed. In this thesis, the term specifically refers to the time needed to evaluate a (composite) polynomial on a ciphertext. This metric is essential for evaluating the efficiency of the algorithms under consideration, as minimizing time consumption is crucial for improving their performance.

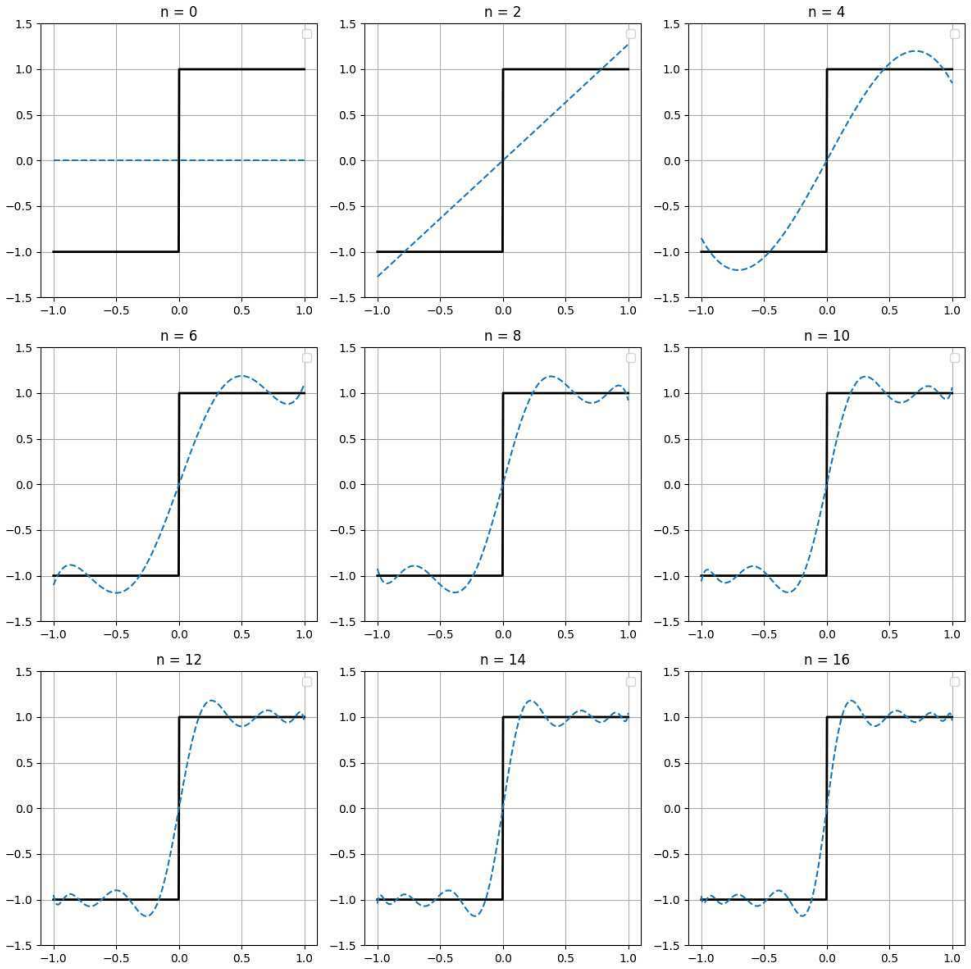

FIGURE 6: Tchebycheff progression for the approximation of the sign function. Here n is the number of coefficients in the polynomial and the degree of the approximating polynomial is $n-1$ for $n > 0$

One of the algorithms that is widely used for approximating non-polynomial functions is Tchebycheff (or Chebyshev) approximation [25]. The Tchebycheff algorithm approximates a function $f(x)$ with a polynomial $p_n(x)$ using a sum of Tchebycheff polynomials, which are defined recursively as follows.

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \text{ for } n \geq 2$$

We do this by providing the algorithm with the degree of approximation and a continuous interval that the function is estimated over. Approximation quality improves with a higher polynomial degree and a narrower interval, as shown in Figure 6. The approximation is expressed as follows, where $T_i(x)$ are Tchebycheff polynomials, and a_i are the coefficients to be determined.

$$p_n(x) = \sum_{i=0}^{n} a_i T_i(x)$$

Tchebycheff approximation aims to minimize the largest error, as opposed to other polynomial approximation techniques that minimize the sum of squared errors (known as least squares). When the resulting polynomial is not truncated, meaning it is not limited in its degree, Tchebycheff approximation guarantees uniform convergence for continuous functions and minimizes the largest error (known as the worst-case

deviation) over the entire interval by satisfying the equioscillation property. However, to make the approximation more computationally feasible and manageable, it is often necessary to truncate it. Truncating the infinite series to a fixed degree N does not necessarily result in the true minimax polynomial (the best uniform approximation) of degree N. This means that truncation sacrifices the optimality of the approximation, although it is often a very close approximation. The error of the resulting polynomial generally oscillates but does not necessarily equioscillate (meaning it does not have equal maximum deviations in magnitude). If the original function $f(x)$ is sufficiently smooth, the truncated Tchebycheff series is very close to the minimax polynomial. However, there is an algorithm, known as Remez's algorithm (which will be discussed in Section 3.1), that is used in practice to compute the exact minimax polynomial, improving upon Tchebycheff truncation.

We learn more about the behavior of the error in the best polynomial approximation when we look at the Tchebycheff Equioscillation Theorem:

- The error between the function $f(x)$ and the approximation $p_n(x)$ does not vary randomly across the interval.
- The error *oscillates*: it reaches its maximum and minimum values, alternating in sign, at least $n+2$ times, where n is the degree of the polynomial.
- This controlled oscillation ensures that the error is spread out as evenly as possible, preventing large deviations at any particular point.

Particularly for functions with abrupt changes or oscillations, this oscillating characteristic makes the Tchebycheff polynomial approximation extremely effective and guarantees that the approximation is as close to the true function as possible.

The function $f(x)$ must be sampled at certain locations, known as Tchebycheff nodes, to calculate the Tchebycheff polynomial approximation. These nodes minimize the Runge phenomenon (large oscillations at the edges of the interval) by spreading the interpolation points more densely near the ends of the interval. This increases approximation accuracy and remedies other typical issues with numerical techniques.

Below, we can find the pseudocode Tchebycheff approximation in Algorithm 1. Figure 6 shows an example approximation of the sign function. The accuracy of the approximation itself increases with the degree of approximation.

Algorithm 1 Tchebycheff Approximation Algorithm

Require: Function $f(x)$, degree n
Ensure: Coefficients $c_0, c_1, \ldots, c_n$ of the approximating polynomial

for $k = 0$ to n **do**
  $x_k \leftarrow \cos\left(\frac{2k+1}{2n+2}\pi\right)$
end for
for $k = 0$ to n **do**
  $y_k \leftarrow f(x_k)$
end for
for $j = 0$ to n **do**
  $c_j \leftarrow \frac{2}{n+1}\sum_{k=0}^{n} y_k \cos\left(j \arccos(x_k)\right)$
  if $j = 0$ **then**
    $c_j \leftarrow \frac{c_j}{2}$
  end if
end for
$P_n(x) \leftarrow \sum_{j=0}^{n} c_j T_j(x)$
Return $c_0, c_1, \ldots, c_n$ or the polynomial $P_n(x)$

In summary, given a non-polynomial function $f(x)$, a finite degree N, and a continuous interval, Tchebycheff approximation enables us to find a polynomial $p(x)$ that

closely approximates $f(x)$. Since we truncate the approximation to a finite degree N, the resulting polynomial will not be the true minimax polynomial. However, under appropriate conditions, this truncated polynomial can be very close to the minimax polynomial and will have a relatively low max error.

Using the Tchebycheff approximation, CKKS can support a broader range of computations while maintaining its approximate nature. Although this approach introduces some errors, the degree of approximation can be controlled to ensure that it is negligible for practical purposes.

To optimize the efficiency and usability of an approximating polynomial, it is important to not only consider the approximation method itself and how it can be optimized, but also the algorithm in which the polynomial is used. By designing an algorithm that considers the issues caused by approximation errors, such as close to discontinuous areas of the function being approximated, the impact of these errors can be minimized. In FHE, the concept of robustness refers to the effect of approximation errors on the final output of an algorithm, particularly in areas where the function being approximated is discontinuous. A more robust algorithm reduces the influence of these errors on the results, ensuring that computations can be performed without significant degradation in quality, even as errors accumulate during the process.

2.5 Algorithms For Ranking, Order Statistics, and Sorting

Using approximated polynomials and basic operations in the CKKS encryption scheme, various algorithms can be developed to analyze encrypted datasets. Recently, there has been a focus on efficient algorithms for ranking, order statistics, and sorting under the CKKS scheme [24]. The main objective of this research is to reduce the computational complexity of these algorithms, particularly in terms of their depth consumption. Mazzone et al. proposed a unique method that allows for high parallelizability and potential hardware acceleration, resulting in a constant comparison depth ($O(1)$) for these algorithms [24]. The comparison depth refers to the number of non-parallelizable homomorphic comparisons required during an algorithm.

One of the main new methods presented in that paper is an algorithm for ranking the values in an encrypted array. We give the algorithm a vector of values as input, it then creates a vector containing the ranking of these values; the lowest value receives a rank of 1, and the highest value receives a rank equal to the vector's length. They do this by using a set of operations created especially for encrypted matrices, such as:

- ReplR(X) replicates a matrix by replicating its values into the other rows, assuming that only the first row is non-zero.
- TransR(X) transposes a square matrix, assuming that only the first row is non-zero.
- Cmp($X_R, X_C; d$) Compares all the values in the matrix X_R with the values at the same index in the matrix X_C, where d indicates the degree of the comparison.
- SumR(X) adds up each row component-wise and then saves the result in the first row.
- MaskR(X, k) replaces the row with index k in the matrix with zeros

A visualization of these operations can be found in Figure 7 below, where we show their effect on a $4x4$ matrix.

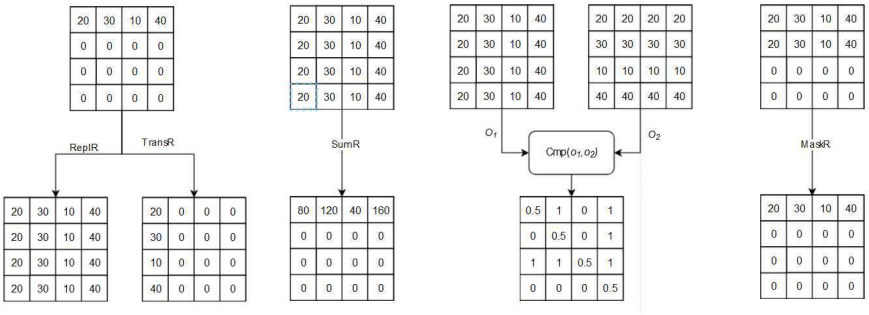

FIGURE 7: Visualization of matrix operations applied to a $4x4$ matrix.

It should be noted that the aforementioned actions can be carried out both column-wise, by for instance copying a column into another column, and row-wise, by for instance copying a row into another row. MaskC, SumC, ReplC, and TransC are the notations for column-wise operations. Using these operations, we can calculate the rank of elements in a vector using the steps shown in Figure 8. The algorithm computes the fractional ranking of an encrypted vector $v = (v_1, \ldots, v_N)$ under the CKKS encryption scheme. The process begins by expanding the input vector V into a matrix V_R by replicating the values of v from the first row across all rows of the matrix. Simultaneously, V is transposed into a column matrix and then expanded into a matrix V_C, where every column holds the elements of v.

Next, the algorithm compares the elements of V_R and V_C using the comparison function. This comparison, guided by an approximate degree d, generates a matrix C where each entry is the result of comparing corresponding elements from the two matrices. The rows of C are then summed component-wise, producing a vector that reflects the number of other elements each value in v is greater than or equal to.

To complete the ranking, the algorithm adds a constant vector $(0.5, \ldots, 0.5)$ to the resulting summation, which adjusts the values to account for fractional rankings. The output of the algorithm is the vector R, representing the fractional rankings of the encrypted values in v. More formally, we can define this process using Algorithm 2.

Using the rank of the algorithm, we can calculate the k-statistic of a vector, also known as the k-th order statistic. This statistic represents the value of the element with a rank of k if such a rank exists. This is achieved by first calculating the rank of the vector, and then using an indicator function to determine if the element falls within the range $[k - 0.5, k + 0.5]$. The indicator function returns a value of 1 if the input value is within the range, and 0 if it is outside. This creates a mask with a value of 1 at the index of the element with the desired rank, and 0 everywhere else. The index of the element can then be found by running argmax over the mask, or the actual value of the element can be calculated by multiplying the original vector with the mask using a dot product and dividing it by the sum of the values in the mask. The formal description for creating the mask can be found in Algorithm 3.

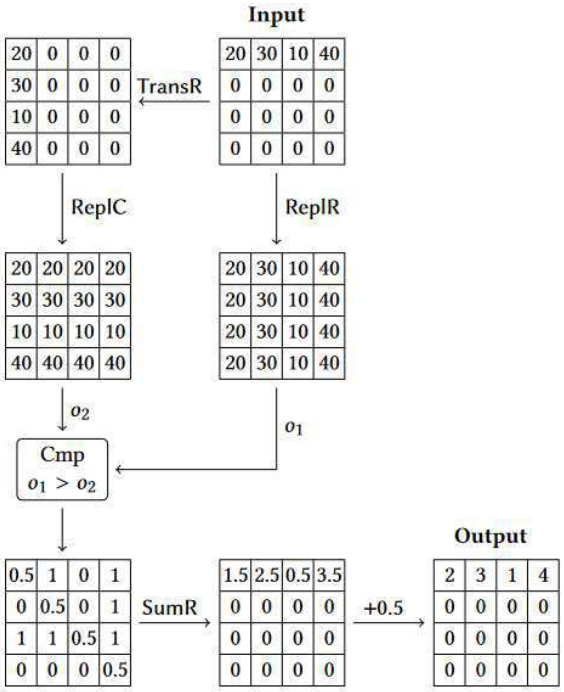

FIGURE 8: Overview of the ranking algorithm

Algorithm 2 Ranking algorithm from [24]

Input V, encryption of $v = (v_1, \ldots, v_N) \in \mathbb{R}^N$, approximation degree $d \in \mathbb{N}$

Output R, encryption of a vector in $\mathbb{R}^N$ representing the (fractional) ranking of v

1: $V_R \leftarrow \text{RepIR}(V)$
2: $V_C \leftarrow \text{RepIC}(\text{TransR}(V))$
3: $C \leftarrow \text{Cmp}(V_R, V_C; d)$
4: $R \leftarrow \text{SumR}(\text{C}) + (0.5, \ldots, 0.5)$
5: **Return** R

Algorithm 3 Order statistics algorithm from [24]

Input V encryption of $v = (v_1, \ldots, v_N) \in \mathbb{R}^N$, approximation degrees $d_C, d_I \in \mathbb{N}$, index $k \in \{1, \ldots, N\}$.

Output Q encryption of a Boolean vector in $\{0, 1\}^N$ that has value 1 in position i if and only if v_i has rank k.

1: $R \leftarrow \text{Rank}(V; d_C)$
2: $O \leftarrow \text{Ind}_k(R; d_I)$
3: **Return** O

3 Sign function approximation

This section addresses the first research question: **To what extent can the algorithms for ranking and order statistics be improved using the Composite Minimax approximation of the homomorphic comparison operation?** We focus on designing, implementing, and evaluating the minimax composite polynomial approximation algorithm to approximate the sign function. The aim is to provide a detailed account of its implementation, its experimental comparison against the Tchebycheff approximation algorithm, and the insights gained from this comparison.

We will begin by explaining the principles underlying the design of the Minimax Composite approximation algorithm in Subsection 3.1. This will include highlighting its theoretical foundations and implementation details. Next, we will describe the experimental setup in Subsection 3.2, outlining the methods and metrics used to compare the performance of the composite minimax and Tchebycheff algorithms. Key metrics such as time consumption and approximation error will be used to assess the algorithms' computational efficiency and accuracy. The section will conclude by analyzing the numerical results obtained from these experiments in Subsection 3.3. We will discuss the implications of the findings, concluding the effectiveness and efficiency of the minimax composite polynomial approximation in addressing the challenges posed by the first research question. Finally, we will briefly discuss the conclusions that can be drawn from these results in subsection 3.4. By the end of this section, the reader will have a comprehensive understanding of the design process, experimental evaluation, and the broader significance of the results in advancing the thesis objectives.

3.1 Approximation Using Composite Minimax Polynomials

In their study, Lee et al. [20, 22] presented an effective method for calculating maximum values and performing homomorphic comparisons. Their approach aims to achieve performance as close to the theoretical ideal as possible. They do this by using a series of composite polynomials optimized over multiple intervals, rather than a single polynomial optimized over a single interval, as in Tchebycheff approximation. This allows them to simulate non-polynomial functions while maintaining a lower total polynomial degree, which reduces the computational depth and preserves the maximum approximation error.

Figure 9, illustrates the Composite Minimax algorithm. The first polynomial is approximated over the range $[a_1, b_1]$. Subsequently, succeeding polynomials are estimated across intervals such as $[1 - a_2, 1 + b_2]$, where a_2 and b_2 represent the maximum approximation error of the previous polynomial. Compared to the Tchebycheff method, the composite polynomials achieve a lower total degree by distributing the errors over multiple intervals. This reduces computational complexity and, consequently, results in a shorter computation time.

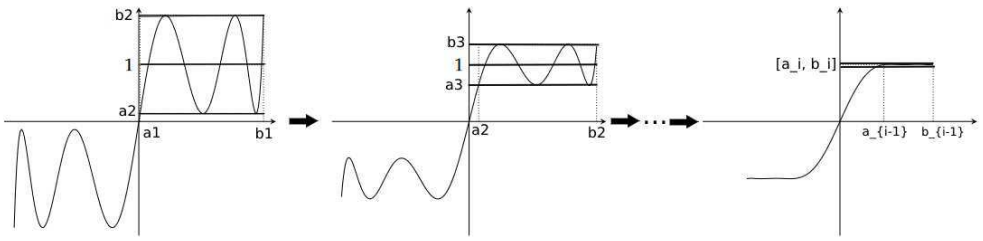

FIGURE 9: Composite minimax approximation of the sign function. Adapted from [20].

Lee et al. used their algorithm to specifically approximate the sign function, which

can be used to implement the comparison and maximum functions. The relationship between the comparison and sign function is defined as:

$$\mathrm{cmp}(u, v) = \frac{1}{2}(\mathrm{sign}(u - v) + 1) \tag{1}$$

Algorithm 4 outlines a three-step process for computing the comparison function by approximating the sign function with a composite minimax polynomial. The first step is determining the optimal set of polynomial degrees that minimize depth and time consumption while ensuring that the max approximation error remains below a desired threshold. Next, a minimax polynomial is constructed over the interval $\tilde{R}_\epsilon = [(-1, -\epsilon), (\epsilon, 1)]$. This is followed by the generation of subsequent composite polynomials over intervals $R_\tau = [(-1 - \tau, -1 + \tau), (1 - \tau, 1 + \tau)]$, where τ represents the minimax error of the previous polynomial. Finally, the comparison function is calculated using these composite polynomials.

The algorithm takes two precision parameters α and ϵ as input. This parameter decides the maximum approximation error, the approximation interval, and the maximum depth consumption used for calculating the optimal degrees.

Algorithm 4 OptMinimaxComp Algorithm [22]

Input: $a, b \in (0, 1)$, precision parameters α, ϵ, and depth D.
Output: Approximate value of $\mathrm{comp}(a, b)$.

$M_{\mathrm{degs}} \leftarrow \mathrm{ComputeMinTimeDegs}(\alpha, \epsilon, D)$
$p_1 \leftarrow \mathrm{MP}(\tilde{R}_\epsilon, 1; d_1)$ ▷ MP: Minimax Polynomial
$\tau_1 \leftarrow \mathrm{ME}(\tilde{R}_\epsilon, 1; d_1)$ ▷ ME: Minimax Error
for $i \leftarrow 2$ to k **do**
    $p_i \leftarrow \mathrm{MP}(R_{\tau_{i-1}}, d_i)$
    $\tau_i \leftarrow \mathrm{ME}(R_{\tau_{i-1}}, d_i)$
end for
return $\frac{p_k \circ p_{k-1} \circ \ldots \circ p_1(a-b)+1}{2}$

To optimize composite polynomials in terms of time and depth, the ComputeMinTimeDegs algorithm, presented in Algorithm 5, calculates the optimal set of degrees M_{degs}. It finds the polynomial with the lowest computational time while ensuring the minimax error remains below a specified threshold over a given interval.

The algorithm begins by constructing two tables:

- $\tilde{u}$: Stores the Inverse Minimax approximation Error (IME) for combinations of depth and non-scalar multiplications. The IME reflects how accurately a polynomial approximates the target function.
- $\tilde{V}$: Contains the polynomial degrees corresponding to the IME values in $\tilde{u}$.

The IME of a composite polynomial is the interval over which the polynomial will have a minimax error below the target threshold. By analyzing $\tilde{u}$, the algorithm selects the polynomial with the lowest time consumption that satisfies the required IME threshold, resulting in both M_{time} and M_{degs}.

The Multi-Interval Remez approximation method is an iterative algorithm for finding the minimax polynomial of a non-continuous function over a set of given intervals. This algorithm takes as input: a degree, a set of intervals over which the function to be approximated is continuous, and an approximation parameter that stands for the target minimax error. It iteratively finds polynomials with lower and lower maximum errors until it finds a polynomial that has a minimax error below the target threshold. In short, the algorithm makes an initial guess for the approximating polynomial. Following this, if the polynomial is not close enough to the target function, it iteratively refines it until it is.

The theorem is based on the Equioscillation theorem, which is also used in the Tchebycheff approximation. It uses this theorem to create an iterative algorithm for

Algorithm 5 ComputeMinTimeDegs Algorithm [22]

Input: Precision parameters α, ϵ, and depth D.
Output: Minimum time M_{time} and optimal degrees M_{degs}.

1: $\tilde{u}, \tilde{V} \leftarrow \text{ComputeUV}(2^{1-\alpha}, D)$
2: **for** $j \leftarrow 0$ to $t_{\max}$ **do**
3: **if** $\tilde{u}(j, D) \geq \delta = \frac{1-\epsilon}{1+\epsilon}$ **then**
4: $M_{\text{time}} \leftarrow j$
5: **Go to Line 11**
6: **end if**
7: **if** $j = t_{\max}$ **then**
8: **return** $\perp$
9: **end if**
10: **end for**
11: $M_{\text{degs}} \leftarrow \tilde{V}(M_{\text{time}}, D)$
12: **return** M_{degs}, M_{time}

finding the minimax polynomial. The process begins with an initialization step, where an initial guess is made for the $n + 2$ extremal points, where n is the degree of the polynomial + 1. The approximation error at these points is expected to alternate in sign. Next, a polynomial $P_n(x)$ is constructed by solving for its coefficients so that the error alternates between positive and negative maximum values at the chosen extremal points. This ensures that the equioscillation property is progressively satisfied during the process. The algorithm then updates the extremal points by finding the locations where the current error $E(x)$ achieves its local maxima in absolute value. These new points become the extremal points for the next iteration. The iterative process continues until the equioscillation property is satisfied, and we have a low enough minimax error. At this point, the error alternates in sign and has equal magnitude at $n + 2$ points, indicating that the algorithm has converged. The Multi-Interval Remez algorithm enforces the equioscillation property throughout the approximation process, making it a more general and flexible approach than Tchebycheff approximation. For example, Remez can approximate functions over arbitrary intervals and handle more weights or constraints, expanding its applicability [26].

More formally, the algorithm involves the following key steps, as illustrated in Algorithm 6:

1. **Initialization:** A set of $n + 1$ initial points is chosen within the domain $D = \bigcup_{i=1}^{l} [a_i, b_i]$, where the function f is defined. (Line 1)

2. **Polynomial Construction:** A polynomial $p(x)$ is computed using a specified basis (e.g., Tchebycheff polynomials) such that the error $p(x) - f(x)$ alternates in sign and achieves a maximal deviation E at these points. (Line 2)

3. **Error Analysis:** The extreme points of $p(x) - f(x)$ within D, along with the domain boundaries, are identified as candidate points. We use the concavity function to filter out positive minima, and negative maxima from these points. (Line 3)

4. **Refinement:** From these candidates, a new set of $n + 1$ points is selected, satisfying the alternating error and maximum sum condition. The algorithm iteratively refines the polynomial until the relative error difference converges below a predefined tolerance γ. (Line 4-10)

This process ensures that the output polynomial achieves the desired minimax approximation properties.

Algorithm 6 Improved Multi-Interval Remez Algorithm

Input A basis $\{\phi_1, \ldots, \phi_n\}$, an approximation parameter γ, an input domain $D = \bigcup_{i=1}^{l} [a_i, b_i] \subset \mathbb{R}$, and a continuous function f on D

Output The minimax approximate polynomial p for f

1: Choose $x_1, \ldots, x_{n+1} \in D$ such that $x_1 < x_2 < \cdots < x_{n+1}$
2: Find the polynomial $p(x)$ in terms of $\{\phi_1, \ldots, \phi_n\}$ such that $p(x_i) - f(x_i) = (-1)^i E$, for $1 \leq i \leq n+1$ and some E
3: Collect all the extreme points of $p - f$ on D such that $\mu(x)(|p(x) - f(x)|) \geq |E|$ and put them in a set B with the boundary points
4: Find $n+1$ extreme points $y_1 < y_2 < \cdots < y_{n+1}$ in B that satisfy the alternating condition and maximum absolute sum condition
5: $\epsilon_{\max} \leftarrow \max_{1 \leq i \leq n+1} |p(y_i) - f(y_i)|$
6: $\epsilon_{\min} \leftarrow \min_{1 \leq i \leq n+1} |p(y_i) - f(y_i)|$
7: **if** $\frac{\epsilon_{\max} - \epsilon_{\min}}{\epsilon_{\min}} < \gamma$ **then**
8: return p(x)
9: **else**
10: replace x_i with y_i for all i, go to line 2.
11: **end if**

The algorithm illustrated in Algorithm 6 makes use of the concavity function $\mu(x)$, which is defined as:

$$\mu(x) = \begin{cases} 1 & \text{if } p(x) - f(x) \text{ is concave at } x \text{ on } D \\ -1 & \text{if } p(x) - f(x) \text{ is convex at } x \text{ on } D \\ 0 & \text{otherwise} \end{cases}$$

This function is used to determine whether the error function $p(x) - f(x)$ is concave or convex at specific points. The Improved Multi-Interval Remez Algorithm uses the concavity function to enforce the alternating condition and filter out positive minima and negative maxima. To better understand how this function works, let us examine the graph shown in Figure 10. When selecting the extrema, points **a, b, c,** and **d** are of particular interest. To filter out positive minima and negative maxima using the concavity function, we multiply its results by the sign function at the same point. Any point with an output of < 0 is a positive minimum or a negative maximum, as is the case for points **a** and **b**. Once we have eliminated positive minima and negative maxima, we can ensure that the points are alternating by multiplying the results of the concavity function for consecutive points. For instance, if we take points **b** and **d** from the graph, we know they are alternating if the points are alternatingly concave and convex. In that case, when we multiply the results of the concavity function for both points it will result in a value of -1.

However, as good as this looks in theory, if the concavity function is used exclusively to find the sets of extreme points it will cause issues. When using only the concavity functions, endpoints or any extreme points next to them are often filtered out, as they consistently have the same concavity as their closest extreme point. Due to this, we cannot pick enough extreme points that are alternatively concave and convex.

To give an example, Figure 11 illustrates the error function $e(x) = p(x) - \text{sign}(x)$ during the first iteration of the Multi-Interval Remez algorithm for a degree-3 polynomial. During this step, we want to pick 5 extreme points among the candidate extreme points $e(x = -1), e(a_1), e(x = -\epsilon), e(x = \epsilon), e(b_1)$, and $e(x = 1)$. Since all the points on the left side of 0 are convex and all of the points on the right side are concave, the concavity function would only permit the selection of a maximum of two alternating points. However, adding a new function $\psi(x)$ guarantees that all required extremes are chosen. What exactly $\psi(x)$ is, will be explained in the next paragraph.

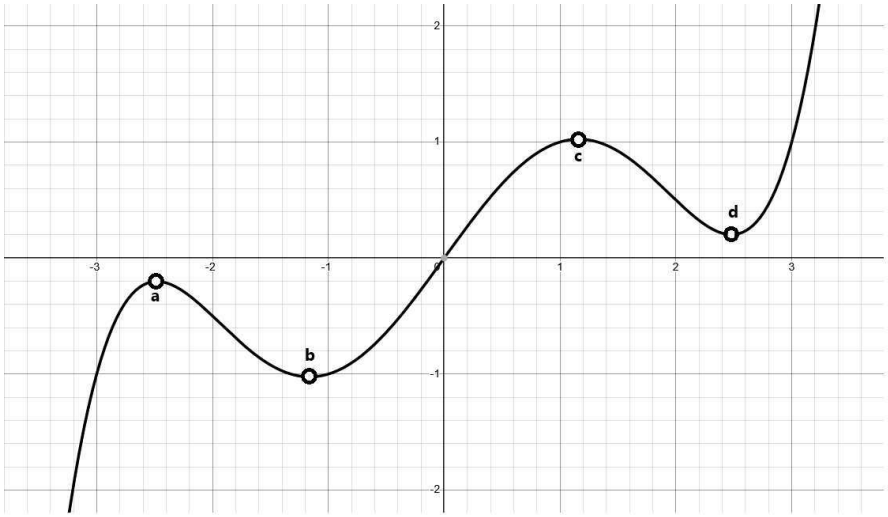

FIGURE 10: Curved Polynomial Function with Maxima and Minima

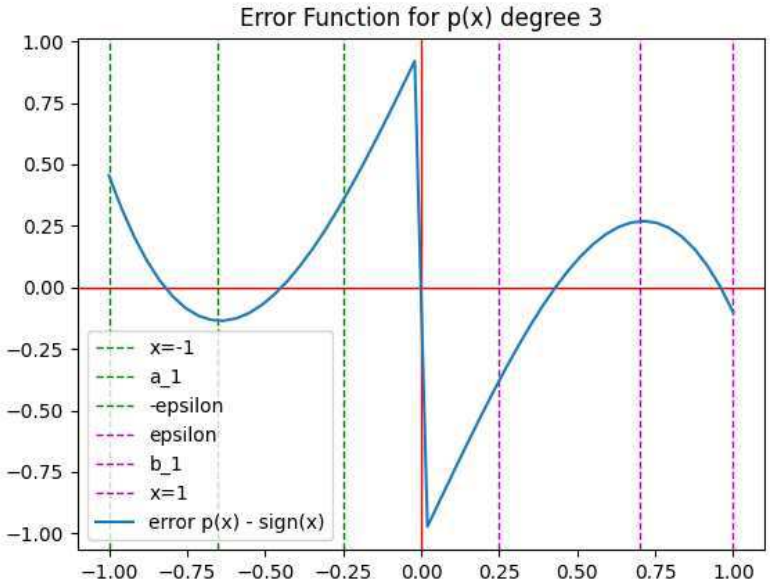

FIGURE 11: Error function for sign approximation with a degree-3 polynomial.

We propose a refinement to address this issue. Instead of relying solely on the concavity function $\mu(x)$ to filter positive minima and negative maxima and enforce the alternating condition, we have separated these tasks. Our approach utilizes the sign function to enforce the alternating condition, while the concavity function is still employed for the first task. These two functions can be combined to create $\psi(x)$.

$$\psi(x) = \mu\left(p(x) - f(x)\right) \cdot \operatorname{sign}(p(x) - f(x)).$$

This modification ensures that:

1. Positive minima and negative maxima are filtered out using $\mu(x)$, as before.
2. The alternating condition is independently enforced by $sign(p(x) - f(x)$, allowing the algorithm to retain sufficient candidate points, including crucial endpoints.

This small yet critical change ensures the algorithm selects $n + 1$ valid extremes without excluding necessary endpoints near the boundaries.

By introducing $\psi(x)$, the Multi-Interval Remez algorithm achieves robust enforcement of the alternating condition, enabling effective approximation even for challenging functions across disjoint intervals.

3.2 Experimental Setup

To compare the Tchebycheff and Composite Minimax approaches for approximating functions, we measured the time consumption and approximation errors of polynomials generated using these two methods. We conducted two tests for this purpose: a micro test, where we approximated the sign function and evaluated it over data encrypted with CKKS, and a macro test, where we approximated the comparison function and used it to calculate the rank of encrypted data.

Unfortunately, the Tchebycheff and Composite Minimax approximation methods do not use the same parameters. While Tchebycheff approximates over one interval and finds the best polynomial for a given degree, the composite approximation method finds the best degrees and corresponding polynomials for a given maximum error over multiple intervals. To compare polynomials from both methods, we introduce a new variable E, which represents the maximum error of a polynomial over the interval $[(-1, -\frac{E}{2}), (\frac{E}{2}, 1)]$. This ensures we compare approximating polynomials with the same maximum guaranteed error over the same interval. This variable can be directly translated to α, which is used to calculate most of the parameters for the Composite Minimax approximation algorithm, using the formula $\alpha = 1 - \log_2(E)$. The higher the value of α (and the lower the value of E), the more accurate the approximation will be over the interval $[(-1, -\frac{E}{2}), (\frac{E}{2}, 1)]$. However, outside of this interval, no guarantees can be made for the accuracy of the approximation. For Tchebycheff polynomials, we use a binary search to find polynomials that meet the maximum error requirements. The resulting degrees for each value of E for both approaches can be found in Table 2.

We use a vector of 128 points encrypted with the OpenFHE library as inputs for the micro and macro tests. The micro test algorithms contain points in the range $(\frac{E}{2}, 1)$, while the macro test algorithms contain points in the range $(0, 1)$. The reason for this difference is that the macro test algorithms take the difference between two points as input, which would result in a range of $(0, 1 - E)$ if the tests were conducted over a vector of points in the range $(\frac{E}{2}, 1)$. This would significantly reduce the number of points in the range over which the polynomials are optimized.

When evaluating the performance of approximated functions, it is crucial to conduct tests on encrypted rather than plaintext data. This approach reflects real-world conditions and ensures the results are meaningful and applicable in practical scenarios. For instance, it enables the use of the Paterson-Stockmeyer algorithm, which is used in OpenFHE [6]. FHE introduces certain computational overheads, such as ciphertext size, noise growth, and depth consumption, which can impact performance and are not present in plaintext evaluations. Additionally, Lee et al. have developed an algorithm to select the optimal degrees of composite polynomials for minimizing computation time and reducing error when evaluating functions over data encrypted using RNS-CKKS [21]. So, evaluating polynomials on plaintext data would not yield relevant results.

	Tchebycheff	Composite Minimax (Remez)
Error (E)	**Degree**	**Degrees**
0.5	3	{3}
0.4	5	{3}
0.3	7	{7}
0.2	37	{15}
0.1	161	{7,7}
0.09	181	{7,7}
0.08	245	{47}
0.07	291	{7,11}
0.06	343	{7,13}
0.05	301	{3,7,7}
0.04	651	{13,13}
0.03	1601	{15,15}
0.02	1715	{5,7,13}
0.01	3201	{3,5,7,13}
0.009	15189	{13,7,15}
0.008	17377	{15,7,15}
0.007	18315	{15,7,3,7}
0.006	28617	{15,11,13}
0.005	31881	{15,13,15}
0.004	35199	{15,3,7,15}
0.003	97089	{15,15,27}
0.002	412137	{15,19,27}
0.001	1139201	{15,7,9,23}

TABLE 2: Degree of the Lowest-degree Polynomial With Max Error Values E

We used the OpenFHE library [1] [1] to encrypt the data and conduct the tests. OpenFHE is a comprehensive, open-source toolkit for FHE, and we specifically used it to implement the RNS-CKKS encryption scheme. The encryption context for the CKKS scheme was configured with a decimal precision of 48 bits and an integral precision of 12 bits. The multiplicative depth, which determines the maximum depth of encrypted computations, was set to the lowest suitable value for evaluating the target polynomial or method. All parameters were chosen following the Homomorphic Encryption Standard, ensuring a security level of 128 bits [2, 3]. The tests were conducted on the student partition of the EEMCS High-Performance Computing (HPC) Cluster at the University of Twente. This partition is powered by an Intel Xeon E5-2698 v4 CPU with 20 cores and 512 GB of RAM operating at 2.20 GHz and runs on a Linux-based system.

3.3 Experimental Results & Discussion

In this subsection, we will discuss the micro and macro test results to compare the minimax composite approximation approach with the Tchebycheff approximation method. We will first go over the micro test results and then discuss the results of the macro tests.

One of the main factors determining the computational efficiency of a polynomial approximation under FHE is its depth consumption. This is directly related to the evaluation time, as a greater depth consumption will result in a longer evaluation time. Figure 12 illustrates the depth consumption for evaluating sign^{C} and sign^{T} with varying maximum errors (E). The key observation is that, in general, sign^{C} has a lower depth consumption compared to sign^{T} for most values of E.

[1] https://github.com/openfheorg/openfhe-development

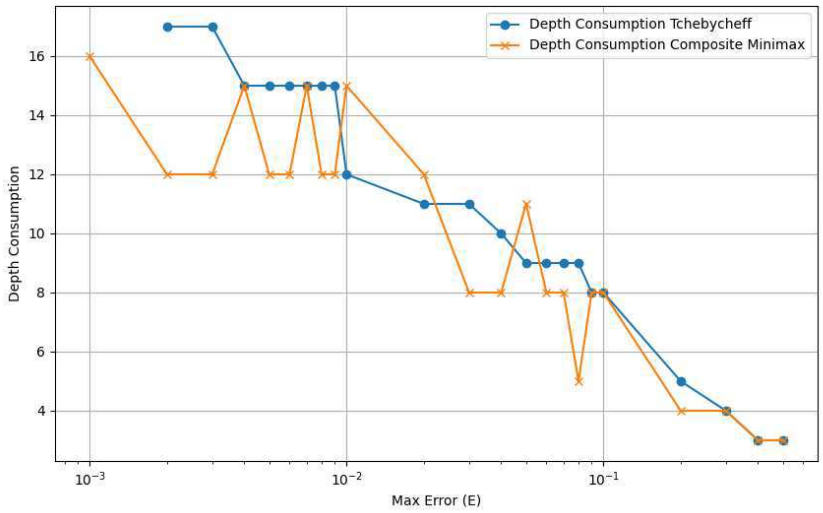

FIGURE 12: Depth consumption vs. maximum error for Tchebycheff and Composite Minimax methods, with a logarithmic x-axis.

Next to depth consumption, we also examined the time required to evaluate the (composite) polynomials, as shown in Figure 13. The graphic illustrates that for all tested E values below 0.3, composite minimax polynomials consistently result in reduced time consumption compared to Tchebycheff polynomials. This time efficiency is particularly beneficial when the guaranteed maximum error is crucial, highlighting the advantage of using the Composite Minimax algorithm.

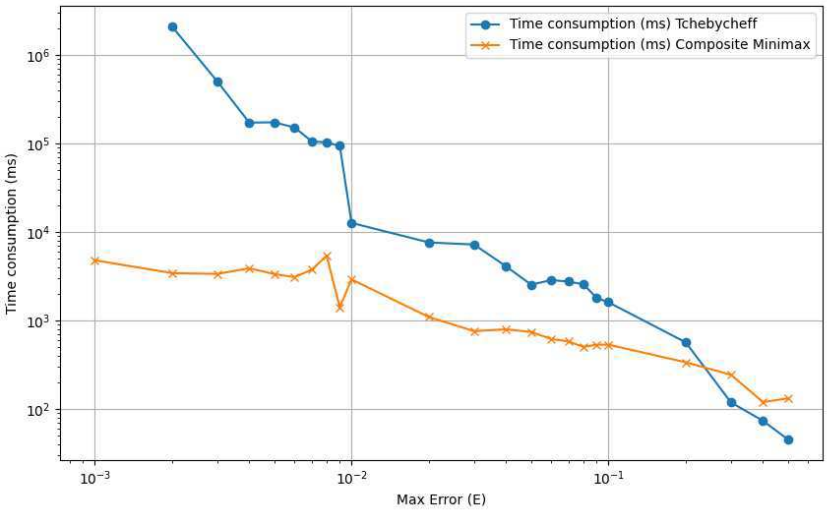

FIGURE 13: Time consumption vs. maximum error for Tchebycheff and Composite Minimax methods, with both axes on a logarithmic scale.

For most assured maximum errors, sign^{C} shows larger average errors than sign^{T}, despite its lower max error. Figure 14 illustrates this tradeoff by comparing the average and maximum errors of the two approaches.

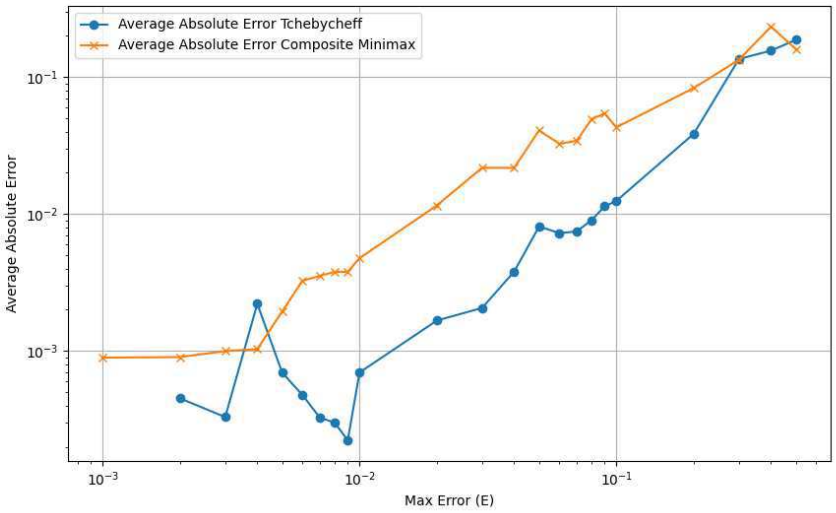

FIGURE 14: Average error vs. maximum error for Tchebycheff and Composite Minimax methods, with both axes on a logarithmic scale.

However, the Composite Minimax algorithm regains an advantage when considering time consumption relative to average error, as illustrated in Figure 15. For average errors below approximately 0.04, sign^{C} outperforms sign^{T} in terms of time efficiency.

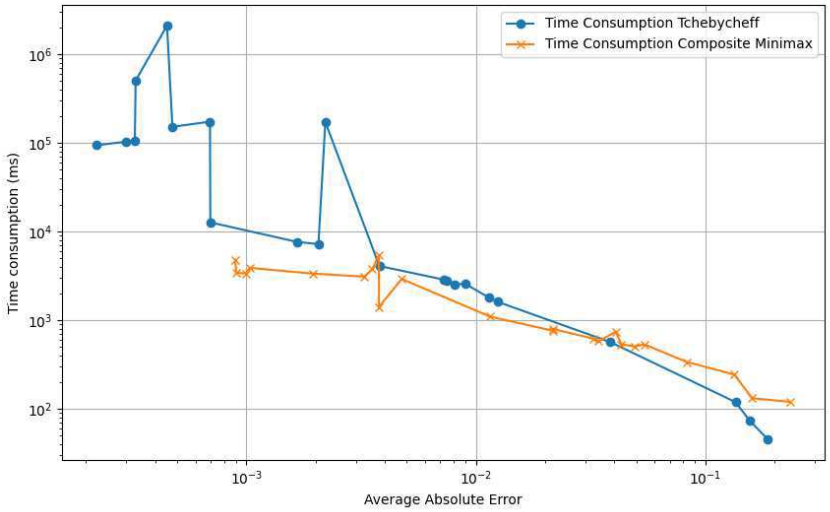

FIGURE 15: Average error vs. time consumption for Tchebycheff and Composite Minimax methods, with both axes on a logarithmic scale.

By achieving lower time consumption for low maximum and average errors, the micro tests demonstrate that the Composite Minimax approximation algorithm is more efficient in approximating the sign function than the Tchebycheff algorithm when evaluated under RNS-CKKS. This makes composite minimax polynomials suitable for situations where time efficiency is a top priority, particularly in applications with limited computational resources or latency restrictions. However, there is a bit of

a trade-off between accuracy and efficiency. While Tchebycheff polynomials perform better in terms of average error against maximum error, composite minimax polynomials excel in terms of time consumption and depth efficiency. Specifically, sign^{T} yields smaller average errors in the range $[\frac{E}{2}, 1]$ compared to sign^{C}, highlighting their potential usefulness in applications where accuracy within specific error bounds is more crucial than time consumption.

The results of the macro tests support our findings. During the macro tests, we evaluated the time consumption and error of using the approximated sign function to calculate the ranks of elements in a vector. As depicted in Figure 16 and Figure 18, sign^{C} outperforms sign^{T} in terms of time consumption versus maximum error E and average error versus time consumption, while sign^{T} demonstrates better average error performance versus maximum error in Figure 17.

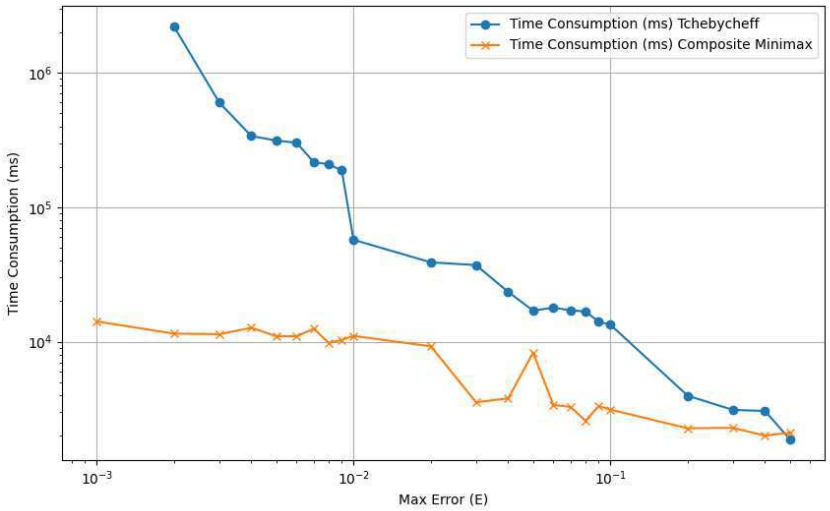

FIGURE 16: Time consumption vs. maximum error for Tchebycheff and Composite Minimax methods in macro tests, with both axes on a logarithmic scale.

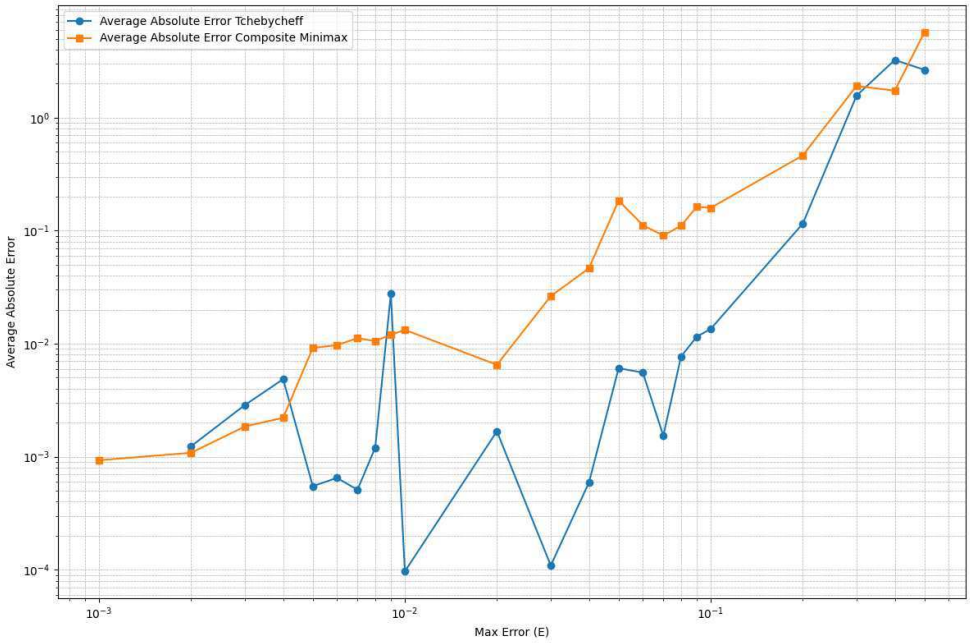

FIGURE 17: Average error vs. maximum error for Tchebycheff and Composite Minimax methods in macro tests, with both axes on a logarithmic scale.

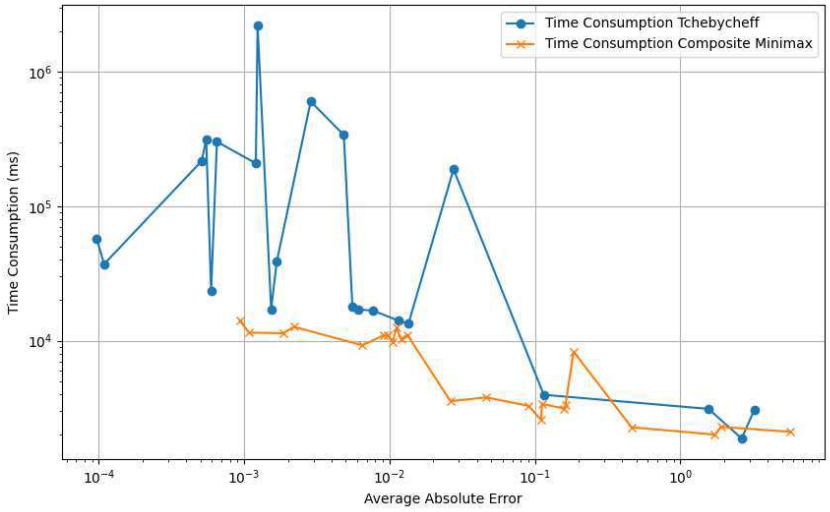

FIGURE 18: Average error vs. time consumption for Tchebycheff and Composite Minimax methods in macro tests, with both axes on a logarithmic scale.

Before drawing any conclusions from these results, it is important to consider one final critical factor that affected the limits of polynomial evaluation: memory consumption. Our micro and macro tests indicated indirectly that the memory usage of sign^{T} significantly increased in comparison to sign^{C} as the maximum error (E) decreased. As a result, we measured the memory consumption required for evaluating the (composite) polynomials to see what was happening, as depicted in Figure 19.

The results show that for values of E less than 0.009, the memory usage of sign^{C} is significantly lower than that of sign^{T}. Although both sign^{C} and sign^{T} require a

considerable amount of memory for these smaller values of E. sign^{C} consistently stays below 30 GB while sign^{T} exceeds 40 GB. This difference emphasizes the significant memory efficiency advantage of sign^{C} for smaller error thresholds.

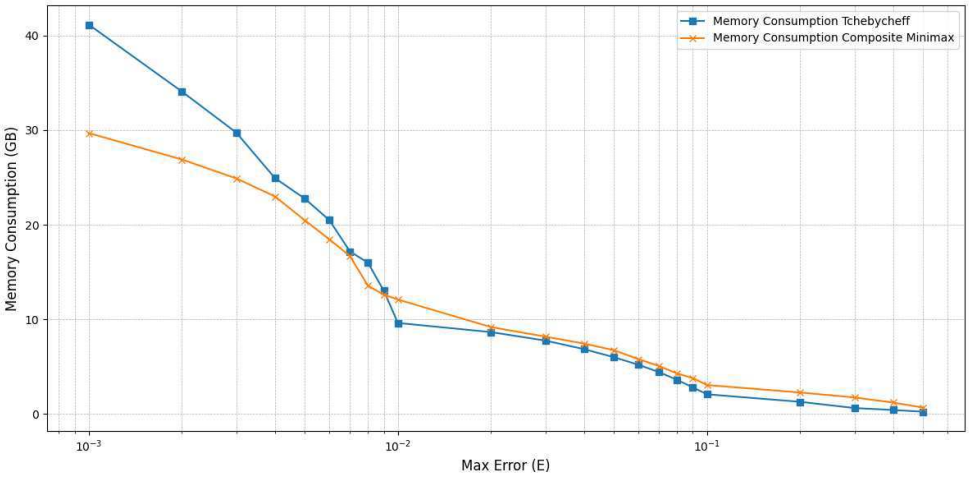

FIGURE 19: Memory consumption for evaluating sign^{C} and sign^{T} under CKKS

Initially, our tests were conducted on the main partition of the HPC Cluster, which also ran many other processes at the same time. However, when attempting to evaluate the highest-degree polynomial for sign^{T} in this cluster, it terminated the process. To resolve this issue, we reran the tests on the student partition, which had fewer processes running in parallel and, at times, no other processes. This allowed us to evaluate the polynomial approximation successfully.

3.4 Conclusion

When approximating the sign function, experimental results show that composite minimax polynomials are more computationally efficient under FHE than polynomials created using the Tchebycheff algorithm. Specifically, the Composite Minimax method requires less time and depth overall, making it a suitable choice for applications with strict performance requirements. However, this efficiency comes at the cost of accuracy, as Tchebycheff polynomials consistently yield lower average errors for the same guaranteed maximum errors. The results of the macro test support these conclusions, demonstrating that the Composite Minimax method is preferable when time efficiency is a top priority and can provide a more accurate approximation for the same time consumption. These findings provide a balanced assessment of the advantages and disadvantages of both approaches, allowing readers to make well-informed decisions based on the needs of their application. The approximated sign functions can be used in ranking, sorting, and order statistic algorithms. Ultimately, the improvements in time efficiency and accuracy will positively impact the performance of these algorithms, resulting in overall increased efficiency and reduced errors.

4 Calculating (Arg)Min and (Arg)Max

This section addresses the second research question: "**Knowing the** max **function is more robust than the comparison function, to what extent can it be used to improve the robustness of the algorithms?**" We focus on developing and evaluating an algorithm for calculating the (arg)min and (arg)max of a vector that is more robust than the one provided by Mazzone et al. [24]. This section provides a detailed examination of the original algorithm, the modifications made to enhance its robustness, and a comparative analysis of its performance under different configurations.

We will begin by describing the design and operational mechanics of the original algorithm for order statistics in Subsection 4.1. This will offer insights into its computational approach and theoretical underpinnings. In the same subsection, we will also detail the modifications that were made to the algorithm, specifically the integration of a comparison function and a max function, to optimize its accuracy and efficiency. Moving on to Subsection 4.2, we will present the experimental setup and methodology used to compare the two versions of the algorithm. We will analyze metrics such as error rates to highlight the trade-offs and improvements achieved through these modifications. In Subsection 4.3, we will discuss the results of these experiments and conclude on the implications of these findings for RQ2 and their relevance to the goals of the thesis. Finally, in Subsection 4.4, we will provide our final thoughts on the conclusions that can be drawn from these results.

By the end of this section, the reader will have a comprehensive understanding of the algorithm its design, the modifications that were made, and the comparative evaluation of its different versions. Additionally, the significance of these developments within the context of the research will be discussed.

4.1 Modification of Algorithm For (Arg)Min and (Arg)Max

To address the second research question, our focus shifts from analyzing the efficiency of different approximation methods to examining the robustness of the algorithms proposed by Mazzone et al. for calculating the (arg)max/(arg)min of a vector. By robustness, we mean the impact of approximation errors near discontinuities in a function on the final output of an algorithm. A more robust algorithm minimizes the influence of these errors on the final results. For this algorithm, we approximate the sign function using the Composite Minimax approximation algorithm.

When approximating a non-polynomial function, such as the sign function, using a polynomial, errors are introduced, particularly near points of discontinuity. For example, the sign function is discontinuous at 0, and any polynomial approximation will perform poorly in that region. Furthermore, even small approximation errors can result in significant changes in the output of a discontinuous function. The sign function can be used to compute the comparison and maximum functions using the following formulas:

$$\mathrm{cmp}^{\mathrm{C}}(u, v) = \frac{1}{2}(\mathrm{sign}(u - v) + 1) \tag{2}$$

$$\max^{\mathrm{C}}(u, v) = \frac{(u + v) + (u - v)\mathrm{sign}(u - v)}{2} \tag{3}$$

$$\min^{\mathrm{C}}(u, v) = (u + v) - \max^{\mathrm{C}}(u, v) \tag{4}$$

When using the Composite Approximation algorithm in the algorithms for order statistics designed by Mazzone et al., the concept of robustness becomes crucial. For example, the comparison function, which relies on the sign function as shown in equation 2 above, is used to determine the maximum and minimum values within a vector. Any errors in the estimated sign value can result in significant fluctuations in

the comparison value, as the comparison function itself is discontinuous, as depicted in Figure 21. In contrast, the max function, which can also be derived from the sign function, is continuous, as shown in Figure 20. This continuity ensures that there are no abrupt jumps in the max function.

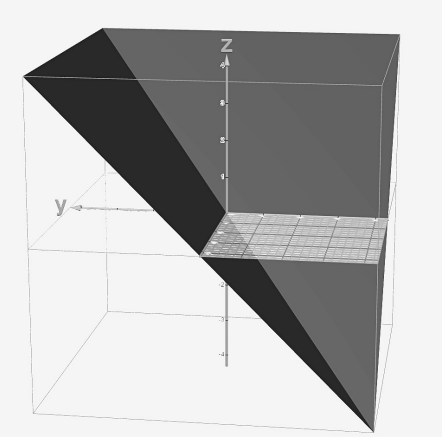

FIGURE 20: Graph of $z = \max^{\mathrm{C}}(x, y)$

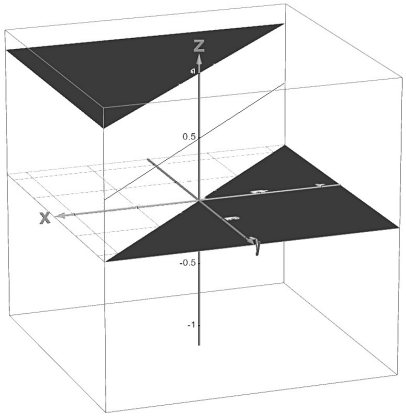

FIGURE 21: Graph of $z = \mathrm{cmp}^{\mathrm{C}}(x, y)$

To illustrate the differences between these functions, consider the results for $\mathrm{cmp}^{\mathrm{C}}(u, v)$ and $\max^{\mathrm{C}}(u, v)$ when $u = v = 0.6$. Ideally, we expect $\mathrm{cmp}^{\mathrm{C}}(u, v) = 0.5$ and $\max^{\mathrm{C}}(u, v) = 0.6$. However, let us suppose the approximated sign value is 0.5 instead of 0 due to approximation errors. For $\mathrm{cmp}^{\mathrm{C}}$, we then compute:

$$\mathrm{cmp}^{\mathrm{C}}(u, v) = \frac{1}{2}(0.5 + 1) = 0.75.$$

While for $\max^{\mathrm{C}}$, we calculate:

$$\max^{\mathrm{C}}(u, v) = \frac{(1.2) + (0 \times 0.5)}{2} = 0.6.$$

In this case, the error in $\max^{\mathrm{C}}$ is 0, while the error in $\mathrm{cmp}^{\mathrm{C}}$ is 0.25.

The algorithm for computing the rank of a vector, designed by Mazzone et al., uses a comparison function. The resulting rank can then be used to determine the maximum or minimum value of a vector by applying the indicator function over the range of $[k - 0.5, k + 0.5]$, where k represents the rank of the maximum or minimum value, respectively. However, it is possible to modify this algorithm to directly use the max and min functions for calculating the maximum or minimum values of a vector. The updated versions of the algorithm are presented in Figures 22 and 23. The original ranking algorithm by Mazzone et al. is depicted in Figure 8.

In both the comparison-based algorithm and the updated versions that use the max/min functions, the initial steps are identical up to the application of the relevant function. This means that when both versions of the algorithm are run on the same input with the same sign function approximation, they will yield the same approximation errors.

The final output of the approximation is a mask with its highest value at the index of the target element. This mask can be used to compute two results: the argmin/argmax and the min/max. The min and max represent the actual values of the elements at those indices, while the argmin and argmax represent the indices of the minimum and maximum values, respectively.

Since the index of the largest value in the mask matches the index of the target element in the original vector, we can calculate the argmin/argmax by simply applying

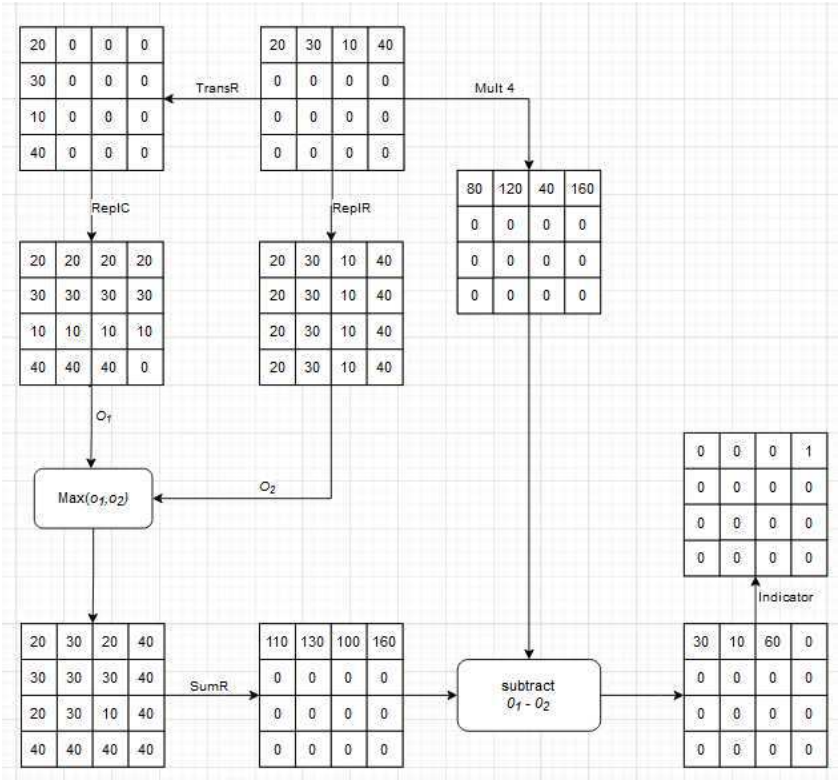

FIGURE 22: Updated algorithm for calculating the maximum of a vector

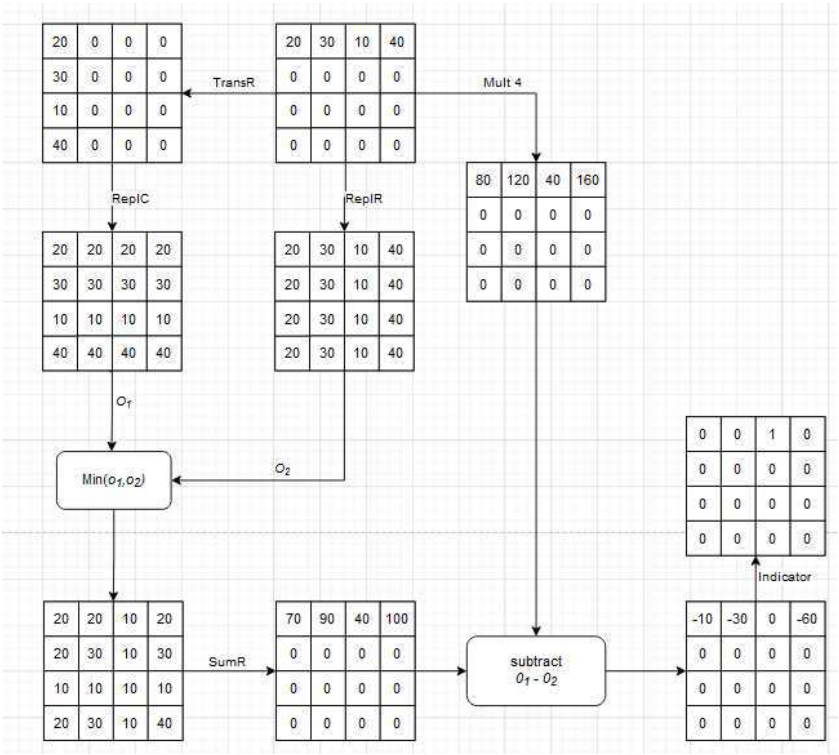

FIGURE 23: Updated algorithm for calculating the minimum of a vector

argmax on the mask. However, calculating the minimum value requires a bit more work. We start by calculating the dot product of the mask with the original vector.

Then, we divide this result by the sum of the mask values, also known as the L_1 norm. For example, we start with the vector:

$$\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

and the mask:

$$\mathbf{m} = \begin{bmatrix} 0.9 \\ 0.2 \\ 0 \end{bmatrix}$$

First, we compute the dot product:

$$\mathbf{v} \cdot \mathbf{m} = (1 \cdot 0.9) + (2 \cdot 0.2) + (3 \cdot 0) = 0.9 + 0.4 + 0 = 1.3$$

Next, we calculate the L_1 norm of the mask, which is the sum of its absolute values:

$$\|\mathbf{m}\|_1 = |0.9| + |0.2| + |0| = 1.1$$

Finally, we compute the result by dividing the dot product by the L_1 norm:

$$\frac{\mathbf{v} \cdot \mathbf{m}}{\|\mathbf{m}\|_1} = \frac{1.3}{1.1} \approx 1.18$$

4.2 Experimental Setup

During the tests, we measured the error of calculating the (arg)min over a vector to evaluate the impact of approximation errors on the algorithm its output. Specifically, we employed two different methods to determine the min and argmin of the vector. The points of the vector were gradually moved closer together, ultimately approaching the non-continuous region of the sign function, to increase the approximation error.

Two separate experiments were carried out to evaluate the impact of approximation errors. In the first experiment, the error was determined by calculating the difference between the expected index and the index generated by the algorithms for argmin. In the second experiment, the error was defined as the difference between the computed minimum value and the expected minimum value for the two algorithms.

The sign function was approximated using composite minimax polynomials with degrees specified in Table 2. These degrees were optimized for calculating the comparison function rather than the max function. However, to evaluate the effect of approximation errors consistently, we used the same approximation for both calculating the min and comparison functions to ensure identical error conditions. Recomputing optimal degrees would prioritize time efficiency over error minimization, which is not the focus of this study.

In the tests, we used vectors of varying sizes with evenly spaced points throughout the same range. As the number of points in the range increased, the distance between points decreased, bringing it closer to the non-continuous zone of the sign function. Specifically, we employed one vector with 128 points in the range $[0.6, 1]$ and vectors with 8, 32, and 128 points in the range $[0.5, 1]$. Figure 24 illustrates that as we increase the number of equidistant points in the vector within the same range, the difference between these points becomes smaller. We chose these specific vector sizes for testing for several reasons. Firstly, the OpenFHE library only accepts vectors up to size 128, as they are converted into 128x128 matrices. Additionally, we tested various sizes ranging from 8 to 128 and found that the sizes represented in the range $[0.5, 1]$ as well as a vector size of 128 in the range $[0.6, 1]$ provide the most informative trends. We also observed similar trends with both smaller and larger distances between points compared to the ones we currently use.

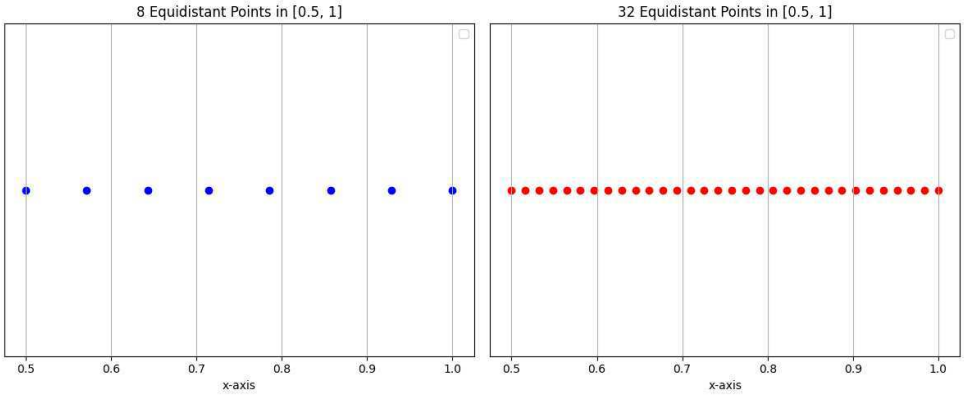

FIGURE 24: Example of point distributions in vectors with 8 and 32 points in the range $[0.5, 1]$

.

In our figures, we will plot the final error of the algorithms against the value of alpha. As a reminder, alpha is the parameter used by the Composite Minimax approximation algorithm to determine the accuracy of the function approximation. The Composite Minimax approximation guarantees a maximum error of $2^{1-\alpha}$ over the interval $[(-1, 2^{1-\alpha}), (2^{1-\alpha}, 1)]$. This is the same as in Section 3, but in that section, we used the maximum error E, which is calculated using α. Outside of this interval, we cannot guarantee the maximum error, but the larger the value of α, the smaller the interval with no guaranteed error around zero will be.

All of these tests were still run using the OpenFHE library, using the same encryption context for the CKKS encryption scheme. We still follow the Homomorphic Encryption Standard to guarantee 128-bit security. And we ran the tests on the same partition of the HPC Cluster.

One last thing we would like to discuss is the fact that we have only tested the algorithm for calculating the argmin and min of a vector and not the algorithm for calculating the argmax and max. In this thesis, we have presented the revised algorithm for calculating both the (arg)min and (arg)max of a vector in Subsection 4.1. The main difference between the two is that we replace the use of the min function with the max function. As shown in Equation 4, we calculate the min function using the max function, so we would expect to see similar results for the updated algorithm. The approximation error introduced will be the same, with the only significant difference being in the input to the indicator function. Ultimately, we are still testing whether the use of the max function is more robust. The same applies when calculating (arg)max using the comparison function. We still calculate the rank, but instead of running the indicator function on the lowest rank, we run it on the highest rank. Therefore, we can draw the same conclusions from the tests for (arg)min as we would for (arg)max.

4.3 Experimental Results & discussion

There are two kinds of results analyzed here. First, we examine the results for calculating the minimum value of a vector, followed by the results for calculating the argmin value.

When examining the error values for calculating the minimum of a vector, as shown in Figures 25 through 28, it is evident that the results for $\min_v^{min}$ have a smaller error for lower values of α and smaller differences between points compared to the results produced by $\min_v^{comp}$. This trend is particularly noticeable for the vector with 128 points in the range $[0.5, 1]$ and alpha values less than 5, as well as the vector with 128 points in the range $[0.6, 1]$ and alpha values less than 6.5. While there are a few spikes where the error for $\min_v^{min}$ is higher than that of $\min_v^{comp}$, overall, $\min_v^{min}$

performs better. However, as the spacing and α values increase, the error shifts in favor of $\min_v^{comp}$. Specifically, as α increases, the error of $\min_v^{comp}$ stabilizes around zero, while the error for $\min_v^{min}$ stabilizes at approximately 0.008.

The spikes in errors in the graph for $\min_v^{min}$ and $\min_v^{comp}$, such as the one at $\alpha = 8.16$ in Figure 25, can be attributed to two factors. Firstly, the indicator function is being applied to values outside the range where the Composite Minimax algorithm guarantees maximum error. This can result in significant errors in the sign function approximation for certain values of alpha.

The second reason for the spikes is due to the implementation of the Multi-Interval Remez approximation algorithm. This implementation involves several variables, two of which are particularly important: precision and step size. The precision discussed here is different from the precision determined by the input value of α. It refers to the number of digits after the decimal point used in the data types. The second variable, step size, is used in determining the optimal points. While I will not go into further detail about how the step size is implemented and used, interested readers can refer to Algorithm 9 in the Appendix. It is worth noting that decreasing the step size and increasing the precision can improve the accuracy of the calculation up to a certain point, but it can also decrease efficiency. In particular, for the higher values of alpha, lower step sizes and higher precision were necessary. However, there is a limit to this improvement. Beyond a certain precision, the libraries used may produce inaccurate results. During our tests, we chose to set the precision and step size variables to values that worked for all values of alpha. However, this means that the values may not be optimized for higher values of alpha.

This is why we see a spike for higher values of alpha in the figures, and why the minimum error stabilizes at approximately 0.008. The second issue could have been solved, but we did not have enough time to calculate the optimal values for these variables.

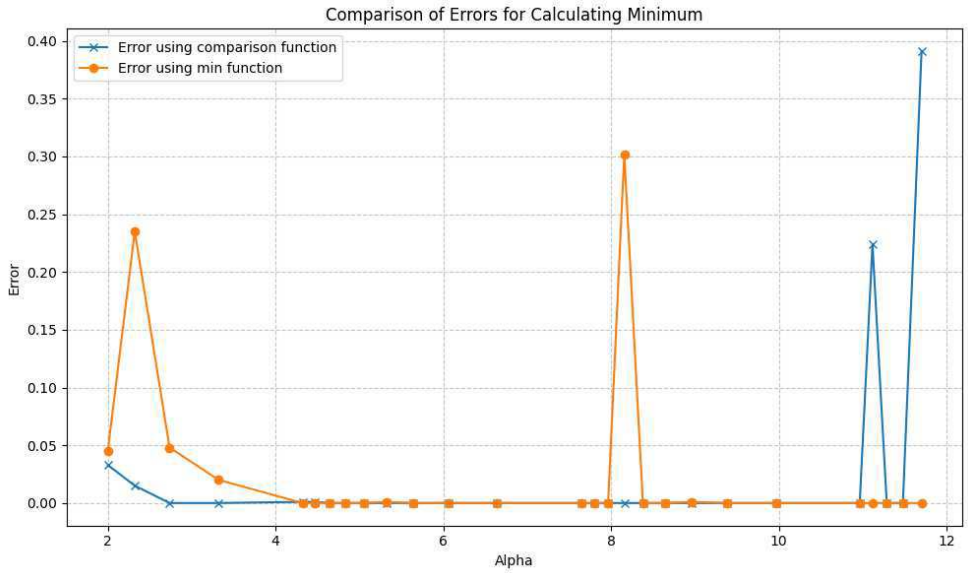

FIGURE 25: Error vs. α for vector size of 8 (range [0.5,1]), min calculation.

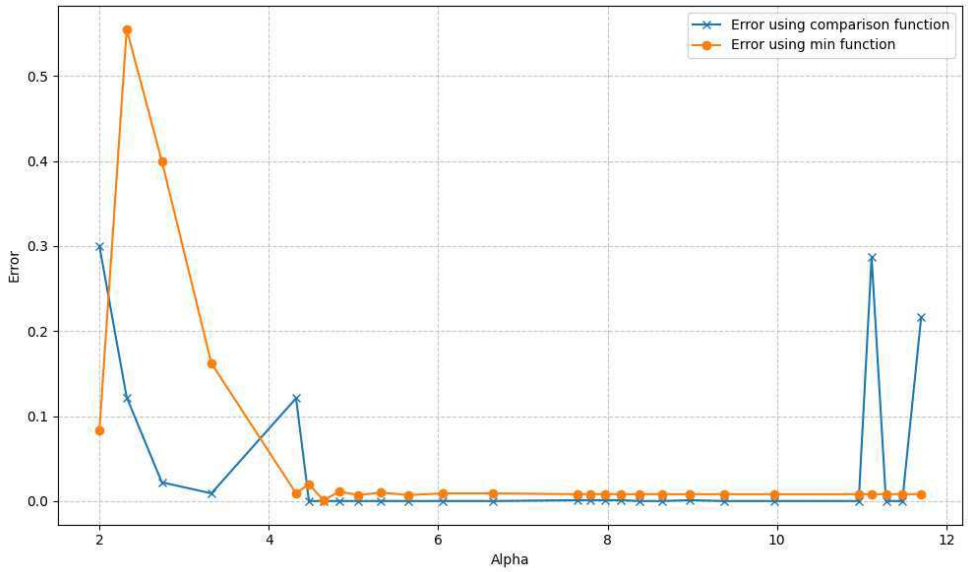

FIGURE 26: Error vs. α for vector size of 32 (range [0.5,1]), min calculation.

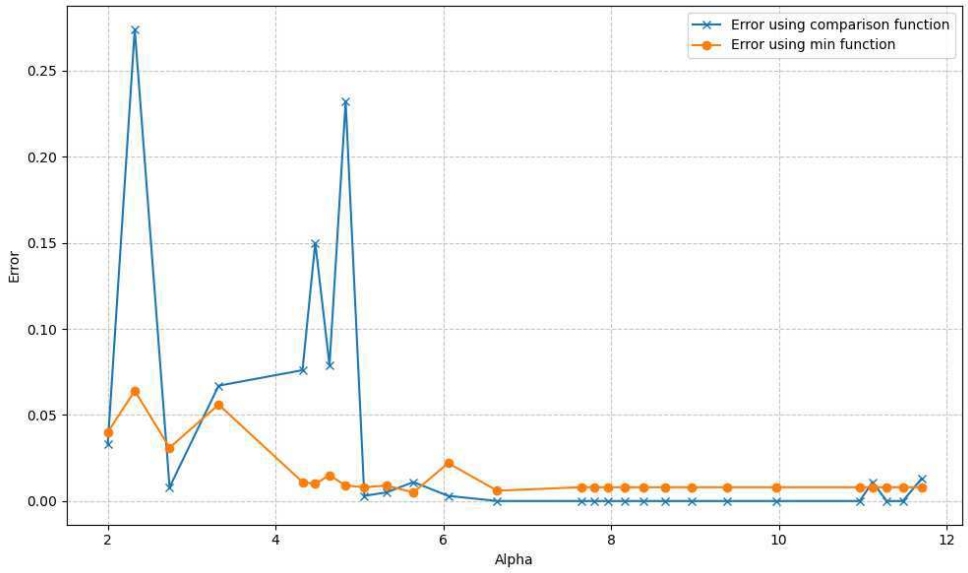

FIGURE 27: Error vs. α for vector size of 128 (range [0.5,1]), min calculation.

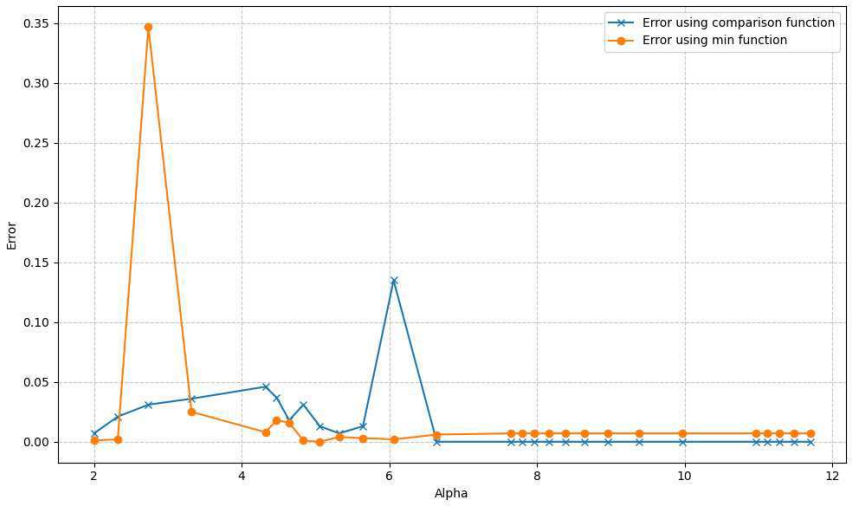

FIGURE 28: Error vs. α for vector size of 128 (range [0.6,1]), min calculation.

After analyzing the results presented above, it is evident that a similar trend can be observed when we focus on specific data points. In Figure 29, we have plotted the error in calculating the minimum value against the difference in points in the vector, with separate graphs for three different values of alpha. It is worth noting that for lower values of alpha and smaller differences in points, the $\min_v^{min}$ algorithm performs better. However, as we increase alpha, the $\min_v^{comp}$ algorithm starts to outperform $\min_v^{min}$ in all test cases. This can be attributed to the fact that lower alpha values result in a higher approximation error, which in turn has a lesser impact on the final result for $\min_v^{min}$ when compared to $\min_v^{comp}$. In other words, a higher approximation error and a smaller difference in points lead to a less significant impact on the final result for $\min_v^{min}$.

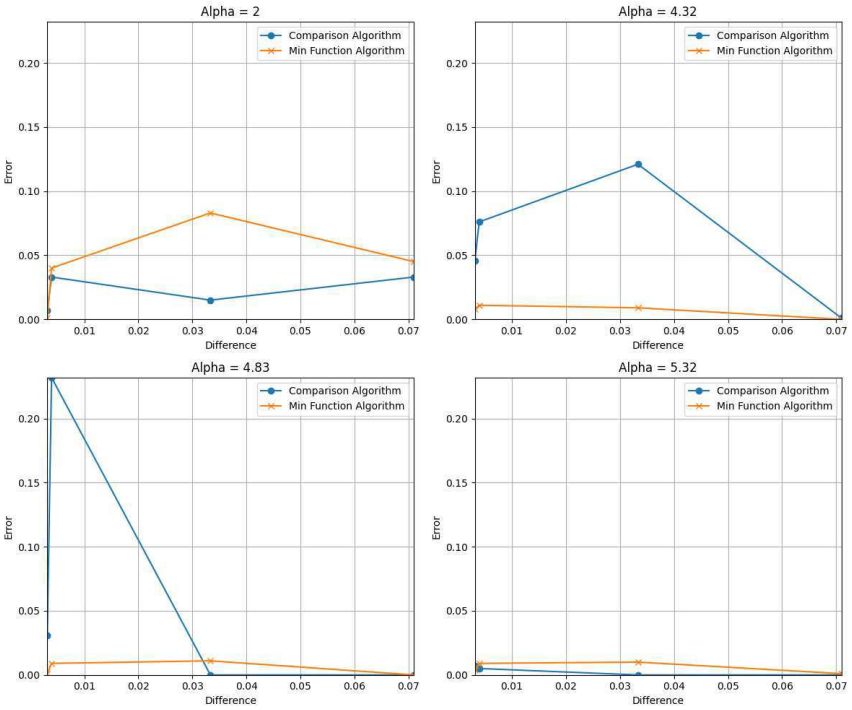

FIGURE 29: Error vs. difference for different α values, min calculation.

For the argmin tests, we can see the results in Figure 30 through Figure 32. We find that argmin_v^{comp} consistently outperforms argmin_v^{min} across all differences and α values. There are a few cases where we see the opposite is true, but overall the trend is that argmin_v^{comp} is the less error-prone algorithm

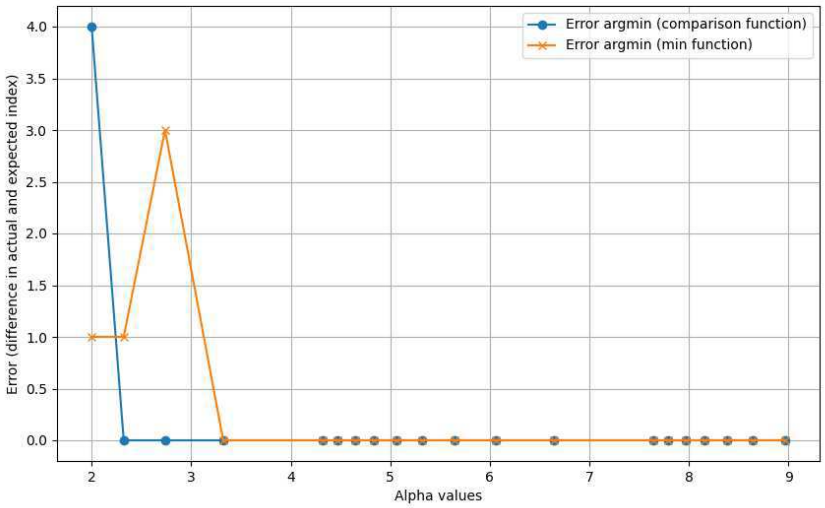

FIGURE 30: Error vs. α for a vector size of 8 (range [0.5,1]), argmin calculation.

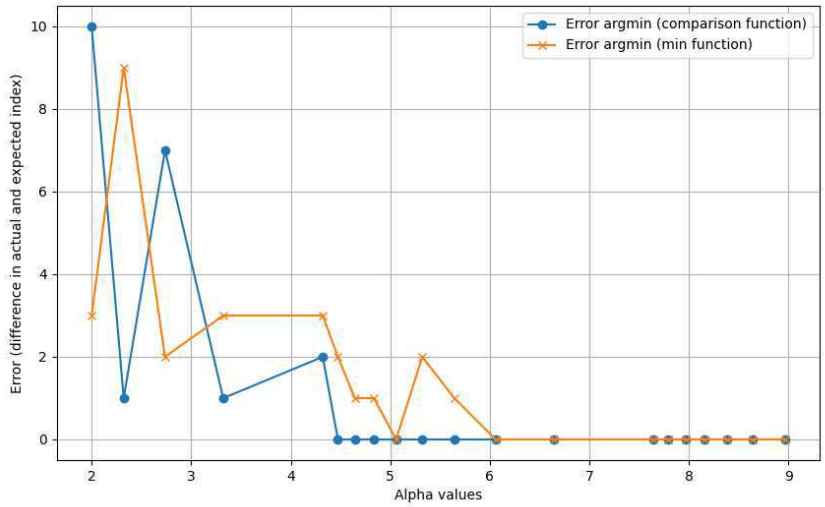

FIGURE 31: Error vs. α for a vector size of 32 (range [0.5,1]), argmin calculation.

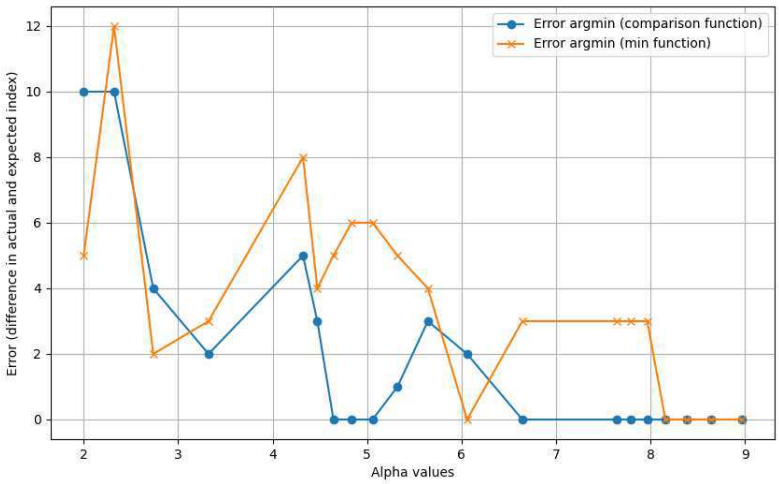

FIGURE 32: Error vs. α for a vector size of 128 (range [0.5,1]), argmin calculation.

This trend is evident when we zoom in and examine the errors for different differences between points, as shown in Figure 33. The pattern holds for both lower and higher values of alpha. Even when introducing less accuracy with lower alpha values, resulting in higher approximation errors, we still observe that argmin_v^{comp} consistently outperforms argmin_v^{min} across all differences and α values. This demonstrates the robustness of argmin_v^{comp} in calculating the argmin of a vector.

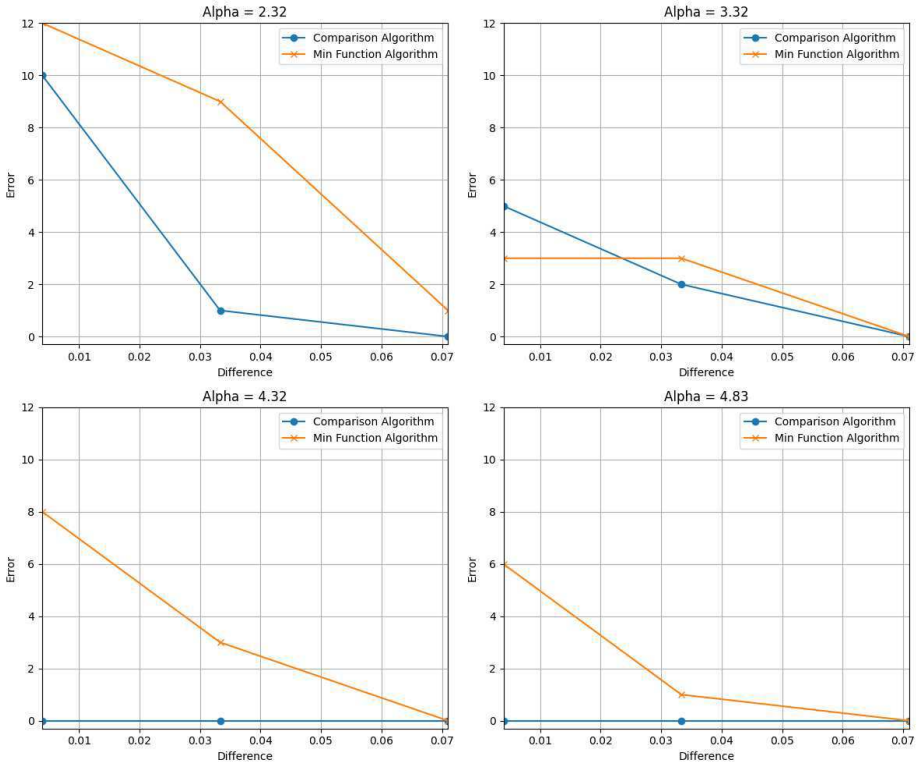

FIGURE 33: Error vs. difference for different α values, argmin calculation.

The characteristics of the mask created during computations are responsible for the variations in these outcomes. In an ideal scenario, we would like to observe a mask with a perfect binary distribution, meaning that there are only zeros and one 1. This is rarely the case, though, particularly when α is lower.

The two methods produce masks with different distributions. We have observed that argmin_v^{comp} creates a mask with a distinct peak, which is advantageous for argmin calculations. However, if the peak is incorrectly positioned, it can result in a higher error for the min calculation. On the other hand, argmin_v^{min} results in a flatter distribution, increasing the likelihood of the peak being at the wrong index in the mask. This is less effective for argmin, as the position of the peak is used in the calculation, but it will result in less error for the min calculation. This is because the min calculation relies on the dot product of the mask with the vector, divided by the L_1 norm. Therefore, the position of the peak matters less as long as it is within the flattened distribution, especially when the distance between points is decreased. This results in more equidistant points within the same range.

Below is an example of two masks produced by the two versions of the algorithm, as shown in Figure 34, with an alpha value of 3.32. It can be observed that the figure has a broken x-axis, as all elements from index three onwards have a value of zero. The algorithm using the comparison function shows a more distinct peak compared to the algorithm using the min function. Furthermore, the peak for the algorithm using the min function is located at index two instead of index one.

If we were to use this mask, the ideal outcome for the algorithm would be for the calculated argmin to return an index of 0, with the actual minimum value being 0.5. When applying the algorithm and using the comparison function to determine the argmin of a vector, the position of the distinct peak allows for a perfect result. However, when determining the true minimum value, $\min_v^{comp}$ provides a value of 0.435022047, while $\min_v^{min}$ yields 0.555267717. In comparison to $\min_v^{comp}$, $\min_v^{min}$ produces a lower error for the computation of the minimum value, even though it may result in a less distinct peak and an incorrect peak position.

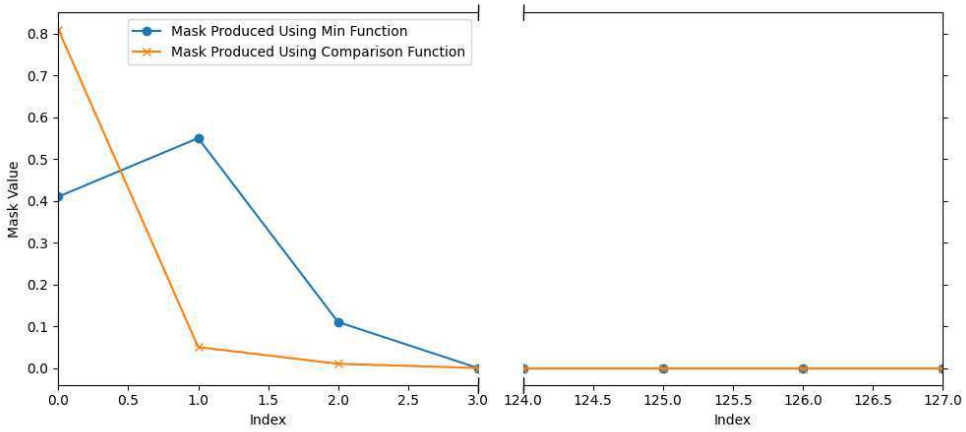

FIGURE 34: Two masks produced by the algorithm for calculating the (arg)min when using the comparison and min function for $\alpha = 3.32$.

The results demonstrate that both approaches have their trade-offs. While using the min function may lead to a slight error when stabilizing at higher α values, it is more resilient to small deviations when calculating the minimal value. On the other hand, utilizing the comparison value yields better performance in argmin calculations due to its sharper peaks in the mask distribution. However, it may not be as accurate in min calculations, particularly if the mask's peak is not correctly positioned. These findings suggest that the specific application and the importance of minimizing errors

in either min or argmin computations should dictate the chosen approach.

4.4 Conclusion

To improve robustness, we have conducted a thorough investigation and experimental evaluation in this section of the modifications to the algorithms for calculating the (arg)min of a vector. The suggested changes offer alternatives to utilizing the comparison function by directly integrating the max and min functions. Our investigations have revealed that while there may be a slight inaccuracy for larger values of α, using the min function turns out to be more resilient when calculating the min of a vector for smaller differences between points (below 0.03) and results in smaller final errors. On the other hand, the comparison function performs better in argmin calculations due to its distinct peak in the mask distribution, but it is less precise when calculating the min.

As previously discussed, these results will also apply to calculating the (arg)max of a vector using the comparison and max functions. The calculation of argmax will still favor the algorithm using the comparison function, as it will result in a more distinct peak in the mask. However, the calculation of the actual minimum value will favor the use of the max function.

Therefore, while our algorithm, which utilizes the max or min function, is more robust for calculating the minimum of an encrypted vector, it is less robust than the algorithm developed by Mazzone et al., which uses the comparison function, when calculating the argmin of an encrypted vector. These results highlight the inherent trade-offs in selecting an algorithm: the comparison approach excels in positional accuracy, while the algorithm using the min function prioritizes stability in value calculations.

5 Related Work

The potential of FHE to enable computations on encrypted data without the need for decryption has been extensively studied. Among the various FHE schemes, the CKKS encryption scheme [10] is widely utilized for privacy-preserving machine learning and encrypted data analytics due to its ability to perform approximate arithmetic on both real and complex numbers.

Approximating non-continuous functions in FHE poses significant challenges due to the trade-off between computational overhead and precision loss. Previous research [18, 5] has utilized polynomial approximations to achieve accurate and efficient evaluations of such functions. The paper "Algorithms in HElib" presents a collection of efficient algorithms implemented within the Homomorphic Encryption Library (HElib), with a focus on optimizing operations such as bootstrapping and key-switching for improved performance in homomorphic encryption [18]. It introduces techniques for compactly handling ciphertexts and demonstrates the practical feasibility of using homomorphic encryption for complex computations. Another paper, written by Boura et al. [5], introduces CHIMERA, a hybrid framework that combines different Ring-LWE-based FHE schemes to leverage their respective strengths. It explores techniques for enhancing efficiency and scalability in homomorphic encryption while maintaining strong security guarantees. However, these methods often involve trade-offs between computational depth and approximation accuracy, which can limit their practical applicability in resource-constrained settings.

The Minimax approximation strategy has emerged as a powerful framework for optimizing polynomial approximations [14]. Despite its extensive application in various computational fields, minimax approximation has received limited attention in FHE, particularly for optimizing composite polynomial approximations of the sign function. This underutilization motivates the present investigation into minimax-based methods to enhance both efficiency and accuracy in CKKS-based computations.

At the same time, a significant amount of effort has been dedicated to developing efficient algorithms for handling encrypted data, with a particular focus on sorting and order statistics. These algorithms often prioritize reducing comparison depth. For example, Hong et al. [19] achieved a comparison depth of $O(k \log_k^2 N)$ in CKKS by utilizing k-way sorting networks, resulting in improved efficiency for floating-point numbers. Similarly, Lu et al. (PEGASUS) [23] implemented Bitonic Sort after transitioning from CKKS, taking advantage of FHEW's efficient look-up tables [13]. Additionally, Léo Ducas and Daniele Micciancio introduced a highly efficient bootstrapping method for FHE in FHEWs, achieving bootstrapping in under a second. A key innovation of FHEW is its use of efficient homomorphic evaluation of look-up tables, allowing for the rapid execution of arbitrary Boolean gates while effectively managing noise. This significantly enhances the practicality of FHE for complex computations.

Minimizing approximation errors and noise is a critical area of study in FHE calculations. The specific subject of studies, as seen in [7, 20, 21], is the approximation error caused during operations on encrypted data. In their paper, "Efficient Homomorphic Comparison Methods with Optimal Complexity," Cheon et al. propose new methods for performing comparisons in homomorphic encryption with optimal complexity [7]. These techniques leverage minimax approximation and optimized polynomial evaluation to efficiently perform comparisons, significantly improving the practicality of comparison operations on encrypted data. In their research, Eunsang Lee et al. introduce a method for approximating the sign function using composite polynomials optimized through the minimax framework [20]. This approach enables efficient and accurate homomorphic comparison operations, particularly within the CKKS scheme, by leveraging the Remez algorithm to minimize approximation errors and enhance performance. Additionally, in another work, Eunsang Lee et al. present advancements in optimizing homomorphic comparison algorithms specifically for the RNS-CKKS scheme [21]. This work introduces techniques to enhance the

computational efficiency and precision of comparison operations by leveraging RNS optimizations, making homomorphic encryption more practical for secure data analysis.

Advancements in FHE methods have greatly reduced errors caused by noise, encryption, and decryption. Notable examples that utilize cutting-edge techniques to improve computational accuracy and robustness include HEAAN [10], which introduces an approximate homomorphic encryption scheme optimized for arithmetic operations on real numbers. RNS-CKKS [8] extends the CKKS scheme by incorporating RNS techniques for improved efficiency and scalability. Another noteworthy example is TFHE [12], a fast and flexible FHE scheme over the torus that achieves high-speed gate bootstrapping.

Our research aims to bridge gaps in the literature by developing efficient and robust algorithms for encrypted computations. This work advances the state-of-the-art in encrypted calculations by integrating minimax approximations and investigating robustness properties.

6 Future Work and Limitations

This section discusses the limitations of the current research and suggests potential avenues for future studies. Our goal is to present a well-rounded view of the scope and significance of our findings, as well as to propose potential directions for further exploration. This will serve as a guide for advancing the research presented in this thesis. By the end of this section, readers will have a comprehensive understanding of the research, its limitations, and the potential for further development based on its contributions.

In this thesis, the focus was on enhancing the efficiency and robustness of algorithms for ranking, order statistics, and sorting within CKKS encryption. However, the research encountered various limitations that restricted its scope and outcomes. Due to time constraints, certain research goals had to be prioritized, leading to a limited exploration of the applicability of the Composite Minimax approximation method to other non-continuous functions beyond the sign function. As a result, a research question had to be scrapped.

The use of a specific CKKS encryption scheme implementation may have facilitated targeted analysis, but it could also have limited the generalizability of the findings. While this approach made targeted analysis and experimentation more manageable, it may have restricted the applicability of the results to other CKKS implementations or modifications. Additionally, any issues with this particular implementation could have influenced the results. However, using the same implementation for all experiments ensured that any potential measurement inconsistencies were systematic rather than arbitrary. It should also be noted that the complete original code base for the Composite Minimax approximation was inaccessible for this research. As a result, a custom implementation of the Multi-Interval Remez algorithm had to be developed, which could have introduced discrepancies compared to the results achievable with the original code.

Furthermore, the accuracy of the approximations presented in this thesis was greatly influenced by certain parameters, such as the number of significant digits used in the approximation methods. However, due to time limitations, further investigation into optimal parameter settings was not pursued. Nevertheless, extensive testing was conducted to ensure that appropriate values were chosen.

The scope of the analysis was limited to the application of the two approximation methods to Mazzone et al.'s ranking and order statistics algorithms [24]. This focus excluded the consideration of other potential strategies for calculating ranking, order statistics, and ranking, which could have potentially provided additional insights and advantages.

By acknowledging these limitations, we aim to not only highlight areas for improvement but also place the results that were achieved more in perspective. Furthermore, it can also be used to establish a foundation for future research.

This thesis, along with its conclusions and limitations, suggests several promising directions for future research. One potential avenue for further exploration is to extend the concept of robustness as it was examined within the framework of the ranking algorithm. For instance, investigating the behavior of sorting algorithms when dealing with equal or closely spaced elements could provide valuable insights into their reliability and potential for improvement.

The Composite Minimax approximation approach can be extended to directly approximate non-continuous and non-polynomial functions, such as the indicator function. This would greatly enhance the method its versatility and applicability to a wider range of mathematical functions and real-world scenarios. It would also be beneficial to explore the possibility of generalizing the approach, as currently, the Multi-Interval Remez approximation approach can only approximate certain functions and cannot calculate optimal degrees. Expanding the second algorithm to include additional functions would greatly improve its applicability, resulting in increased time efficiency and accuracy for a wider range of algorithms, rather than just those using

the sign function or its immediate transformations (e.g., the comparison function).

In addition to extending the Composite Minimax approximation methods to work with other functions, we can also observe its impact when using it to approximate the sign function in other algorithms. While its behavior in ranking computations is well understood, further analysis in other algorithmic paradigms and tasks could provide valuable insights. This would allow us to determine if the same benefits are seen in different scenarios.

By applying Composite Minimax approximation with various CKKS implementations, we can gain insights into their compatibility and generalizability. These efforts would aid in identifying the advantages and drawbacks of different CKKS implementations. It is important to note that these algorithms may not use the same methods for evaluating polynomials as the CKKS implementation of OpenFHE, which could result in different polynomial evaluations and potentially lead to the selection of a different optimal composite polynomial.

Another crucial area of research is the formalization of the definition of robustness in FHE. The theoretical foundations and practical relevance of this study could be enhanced by establishing a precise definition and methodology for evaluating the robustness of FHE methods.

Furthermore, the exploration of how to optimize the parameter selection for the indicator function in the revised minimum calculation algorithm. This function checks if a value falls within a given range, which is dependent on the difference between values in a database. There are various ways to control this, such as setting a maximum precision for the data in the database. Exploring the most effective methods for controlling variables like this to achieve the best results is a crucial aspect that should be investigated in real-world applications.

Lastly, putting these techniques to the test in real-world applications would provide valuable insights and further confirmation of the results observed in this study. Possible fields for implementation include encrypted data analysis, safe multiparty computation, and privacy-preserving machine learning. By applying these methods in practical scenarios, we can better understand their strengths and limitations, which can inform future optimizations.

7 Conclusion

This thesis aims to advance the development of sorting, ranking, and order statistics algorithms within the CKKS encryption scheme. These algorithms are crucial for privacy-preserving applications such as encrypted machine learning, secure cloud computing, and confidential data analysis. In order for these applications to operate effectively under the constraints of homomorphic encryption, robust algorithmic solutions are necessary. Therefore, the main goal of this thesis was to increase the robustness and efficiency of these algorithms, making them more practical for real-world use.

A key focus of this research was the sign function, which plays a central role in algorithms performing ranking, order statistics, and sorting in CKKS encryption. The sign function enables essential comparisons and evaluations, making it a crucial component of these algorithms. Unfortunately, the sign function cannot be evaluated directly over a ciphertext in CKKS and has to be approximated using a (composite)polynomial. So, to improve the performance of the sign function, we explored and compared two distinct methods for approximating it: the Composite Minimax approximation algorithm and the Tchebycheff approximation. By evaluating the trade-offs in time efficiency and accuracy between these two methods, we were able to gain insight into the practical implications of adopting either approach in privacy-preserving computations.

This thesis not only explores the approximation of the sign function, but also introduces a redesigned algorithm for calculating the (arg)max or (arg)min of a vector. This improved method builds upon the work of Mazzone et al. [24] and aims to enhance robustness in the presence of approximation errors. Instead of using a comparison function, we directly use the max or min function, depending on whether the (arg)max or (arg)min value of a vector is being calculated. This redesign is specifically targeted at reducing the impact of approximation errors in discontinuous areas of the function being approximated, ultimately improving the robustness of the algorithm. As we know, approximating a non-continuous function using a continuous polynomial will inevitably introduce some degree of error. Therefore, this refinement addresses specific challenges encountered in encrypted environments, ultimately contributing to more stable and reliable operations within the CKKS framework.

We make a significant contribution to the field of privacy-preserving computation by offering novel advancements within the context of the CKKS encryption scheme. The implementation of the minimax approximation algorithm for the sign function, which is a crucial operation for supporting sorting algorithms, order statistics, and ranking, is the main focus of these contributions. This implementation includes several key improvements, such as optimized polynomial evaluation and error reduction, and will be made available as open-source to encourage further research. The contributions can be summarized as follows.

1. Adaptation and implementation of the Multi-Interval Remez algorithm in Python to efficiently compute composite minimax polynomials.

2. Implementation of algorithms for computing comparison and max functions, specifically designed for use within the CKKS encryption scheme.

3. Development of a pipeline to handle all necessary setup operations, including measurements required for optimal polynomial degree selection.

4. Construction of a pipeline for determining optimal polynomial degrees, approximating the sign function using Composite Minimax algorithms, and evaluating its performance on CKKS-encrypted data using the OpenFHE library.

This study not only implemented new techniques but also analyzed the accuracy and efficiency of composite minimax polynomials compared to Tchebycheff polynomials. The results offer valuable insights into the trade-offs between these two strategies

and provide recommendations for improving algorithmic techniques in CKKS-based applications.

Additionally, the thesis presents a redesigned algorithm for calculating the (arg)min and (arg)max of vectors in encrypted settings. By replacing the traditional comparison function with the max or min function, this approach aimed to improve the robustness and reliability of the algorithm. The redesigned method was thoroughly evaluated in terms of time efficiency and final error, with comparisons made against the original method using the comparison function. These findings contribute to a more profound understanding of how algorithmic adjustments can enhance the performance of operations in encrypted environments.

To support our claims, we conducted tests to demonstrate the superiority of our proposed methodologies and implementations compared to the current approaches. One significant finding was that composite minimax polynomials are more efficient and less prone to errors than the Tchebycheff polynomial approach when approximating the sign function. This was particularly evident in terms of time efficiency, as composite minimax polynomials consistently showed reduced time consumption for maximum errors below 0.3. For example, the maximum observed reduction in execution time was approximately 2088767 milliseconds (2089 seconds or 34 minutes) for a maximum error of 0.00200. On average, the time difference was 154482 milliseconds (2.5747 minutes) in favor of composite minimax polynomials. Additionally, composite minimax polynomials had lower approximation error rates across all evaluated scenarios, further highlighting their advantages. Our tests also demonstrated that the Composite Minimax approximation is more memory-efficient for maximum errors below 0.009.

The evaluation of vector ranking confirmed the time efficiency and accuracy advantages of directly evaluating the sign function. When considering maximum errors below 0.4, composite minimax polynomials outperformed Tchebycheff polynomials in terms of time efficiency. The maximum observed time reduction was approximately 2201680 milliseconds (2201 seconds or 36 minutes) for a maximum error of 0.00200, with an average difference in execution time of 204382 milliseconds (3.4 minutes) in favor of composite minimax polynomials.

In terms of robustness, the performance of the algorithms varied depending on the specific operation being performed. For the calculation of the minimum (min) of a vector, the algorithm using the min/max function demonstrated greater robustness. For example, when evaluating vectors with 128 points in the ranges $[0.5, 1]$ and $[0.6, 1]$, the average differences in errors were 0.031190476 and 0.014428571, respectively, in favor of the min/max function-based method. However, for the calculation of the argument of the minimum (argmin), the comparison function-based algorithm proved more robust. In this case, for vectors of size 8, 32, and 128, the average differences in errors were 0.05, 0.3, and 1.75, respectively, in favor of the comparison function-based method. Similar results can be expected when calculating argmax and max, as their calculation only requires minimal changes to the algorithms. This shows that using the max and min functions is more robust when calculating the max or min of a vector while using the comparison function is more robust when calculating the argmax or argmin of a vector.

These findings highlight the complex trade-offs involved in selecting optimal algorithmic strategies for encrypted computations. They also underscore the importance of tailoring these strategies to the specific requirements of the task, balancing efficiency, accuracy, and robustness to achieve optimal performance in privacy-preserving environments.

Despite its limitations, such as time constraints and reliance on a specific CKKS implementation, this work lays the foundation for future research on efficient and robust algorithms for ranking, order statistics, and sorting within CKKS encryption. Future work could expand the Composite Minimax approximation method to other non-continuous and non-polynomial functions, such as the indicator function, and evaluate its performance across diverse algorithms and CKKS implementations. For-

malizing the concept of robustness within FHE and exploring practical applications, such as privacy-preserving machine learning, could further enhance the theoretical and real-world impact of this research. Additionally, addressing challenges like parameter tuning for real-world scenarios would contribute to refining the proposed methods and increasing their applicability.

In this study, we lay the foundation for future research aimed at enhancing the robustness and efficiency of algorithms within FHE schemes. It illustrates the potential of FHE in real-world applications by showing how algorithmic improvements, such as the replacement of non-continuous functions with continuous counterparts and the use of composite polynomials for sign function approximation, can maximize performance and accuracy. This study lays the foundation for future research aimed at enhancing the robustness and efficiency of algorithms within FHE schemes. this is another test

References

[1] Ahmad Al Badawi et al. "OpenFHE: Open-Source Fully Homomorphic Encryption Library". In: *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. WAHC'22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 53–63. ISBN: 9781450398770. DOI: 10.1145/3560827.3563379. URL: https://doi.org/10.1145/3560827.3563379.

[2] M. Albrecht et al. *Homomorphic Encryption Security Standard*. Tech. Rep. Toronto, Canada: HomomorphicEncryption.org, Nov. 2018.

[3] M. R. Albrecht, R. Player, and S. Scott. "On the Concrete Hardness of Learning with Errors". In: *Journal of Mathematical Cryptology* 9.3 (2015), pp. 169–203.

[4] Frederik Armknecht et al. "A Guide to Fully Homomorphic Encryption". In: *Journal of Cryptology* 37.1 (2024). University of Mannheim and NTNU, pp. 1–45. DOI: 10.1007/s00145-023-09456-9. URL: https://link.springer.com/article/10.1007/s00145-023-09456-9.

[5] Christina Boura et al. *CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes*. Cryptology ePrint Archive, Paper 2018/758. 2018. URL: https://eprint.iacr.org/2018/758.

[6] Hao Chen, Ilaria Chillotti, and Yongsoo Song. "Improved Bootstrapping for Approximate Homomorphic Encryption". In: *Advances in Cryptology - EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Cham: Springer International Publishing, 2019, pp. 34–54.

[7] 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 26*. Springer. 2020, pp. 221–256.

[8] Jung Hee Cheon et al. "A Full RNS Variant of Approximate Homomorphic Encryption". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 347–368. ISBN: 9783030109707. DOI: 10.1007/978-3-030-10970-7_16. URL: http://dx.doi.org/10.1007/978-3-030-10970-7_16.

[9] Jung Hee Cheon et al. "A Full RNS Variant of the CKKS Scheme". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 931.

[10] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 409–437. ISBN: 9783319706948. DOI: 10.1007/978-3-319-70694-8_15. URL: http://dx.doi.org/10.1007/978-3-319-70694-8_15.

[11] Jung Hee Cheon et al. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Takahiro Matsuda and Palash Sarkar. Vol. 10624. Lecture Notes in Computer Science. Springer, 2018, pp. 409–437. DOI: 10.1007/978-3-319-70694-8_15.

[12] Ilaria Chillotti et al. *TFHE: Fast Fully Homomorphic Encryption over the Torus*. Cryptology ePrint Archive, Paper 2018/421. https://eprint.iacr.org/2018/421. 2018. URL: https://eprint.iacr.org/2018/421.

[13] Léo Ducas and Daniele Micciancio. “FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2015, pp. 617–640. ISBN: 9783662468005. DOI: 10.1007/978-3-662-46800-5_24. URL: http://dx.doi.org/10.1007/978-3-662-46800-5_24.

[14] Charles F Dunkl and Yuan Xu. *Orthogonal Polynomials of Several Variables*. Cambridge University Press, 2001.

[15] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the 41st annual ACM symposium on Theory of computing*. ACM. 2009, pp. 169–178.

[16] Suyog Gupta et al. “Deep learning with limited numerical precision”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 1737–1746.

[17] Shai Halevi, Yuriy Polyakov, and Victor Shoup. “An improved RNS variant of the BFV homomorphic encryption scheme”. In: *Journal of Cryptology* 32.1 (2019), pp. 36–66.

[18] Shai Halevi and Victor Shoup. “Algorithms in HElib”. In: *CRYPTO 2014* (2014), pp. 554–571.

[19] Seungwan Hong et al. “Efficient Sorting of Homomorphic Encrypted Data With k-Way Sorting Network”. In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 4389–4404. ISSN: 1556-6021. DOI: 10.1109/tifs.2021.3106167. URL: http://dx.doi.org/10.1109/TIFS.2021.3106167.

[20] Eunsang Lee et al. “Minimax Approximation of Sign Function by Composite Polynomial for Homomorphic Comparison”. In: *IEEE Transactions on Dependable and Secure Computing* 19.6 (2022), pp. 3711–3727. DOI: 10.1109/TDSC.2021.3105111.

[21] Eunsang Lee et al. “Optimization of Homomorphic Comparison Algorithm on RNS-CKKS Scheme”. In: *IEEE Access* 10 (2022), pp. 26163–26176. ISSN: 2169-3536. DOI: 10.1109/access.2022.3155882. URL: http://dx.doi.org/10.1109/ACCESS.2022.3155882.

[22] Eunsang Lee et al. “Optimization of Homomorphic Comparison Algorithm on RNS-CKKS Scheme”. In: *IEEE Access* 10 (2022), pp. 26163–26176. DOI: 10.1109/ACCESS.2022.3155882.

[23] Wen-jie Lu et al. “PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 1057–1073. DOI: 10.1109/SP40001.2021.00043.

[24] Federico Mazzone et al. *Efficient Ranking, Order Statistics, and Sorting under CKKS*. 2024. DOI: 10.48550/ARXIV.2412.15126. URL: https://arxiv.org/abs/2412.15126.

[25] Theodore J. Rivlin. *The Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory*. New York: Wiley, 1974.

[26] Abiy Tasissa. *Function Approximation and the Remez Algorithm*. https://abiy-tasissa.github.io/remez.pdf. Accessed: 2025-01-06. 2023.

Appendix

Multi-interval Remez Approximation

Below you can find pseudocode for the full implementation of the multi-interval Remez approximation algorithm used in this thesis. You might note that some algorithms below some of the lines have the comment {library}. This is used to indicate that an explanation of how to do this exactly is not given, but instead, we used a library to do this. For example, in Algorithm 7 we solve the linear system of equations $A \cdot l = b$, and for this we use one of the solvers given by the mp math library.

Algorithm 7 Remez Algorithm for Function Approximation

Require: Function func: $X \to \mathbb{R}$, polynomial degree n_{degree}, intervals $D = [(a_1, b_1), (a_2, b_2), \dots]$, approximation parameter *approximationParam*

Ensure: Polynomial coefficients coeffs for a minimax polynomial with maximum error $e_{\max}$

Initialize $n = n_{\text{degree}} + 1$
Generate initial Tchebycheff points x_{points} = `generatePointsInRanges`(D, $n+1$)
maxError = 0;
while true **do**
    A = matrix of size $(n+1) \times (n_{\text{degree}} + 2)$
    for $i = 0$ to n **do**
        $A[i, n_{\text{degree}} + 1] = (-1)^{i+1}$
    end for
    vander = Tchebycheff Vandermonde matrix for x_{points} and n_{degree} ▷ < libary
    for $i = 0$ to n **do**
        for $j = 0$ to n_{degree} **do**
            $A[i, j]$ = vander$[i, j]$
        end for
    end for
    vector b = [func(x) | $x \in x_{\text{points}}$]
    det(A) = determinant of matrix A ▷ < libary
    if det(A) = 0 **then**
        Regularize A using Tikhonov regularization ▷ < libary
    end if
    l = solution the linear system $A \cdot l = b$ ▷ < libary
    coefficients = $l[:-1]$
    $p(x)$ = TchebycheffEvaluate(coeffs, x) ▷ < libary
    Define error function $r(x) = p(x) - \text{func}(x)$
    Define vector extremePoints
    secondDerivRx = second derivative of $r(x)$ ▷ < libary
    for (lower, upper) in D **do**
        unFilteredExtremePoints = `findExtremePointsSetup`(r(x), lower, upper)
        filteredExtremePoints = `filterPosMinNegMax`(unFilteredExtremePoints, secondDerivRx, r(x))
        extremePoints.append(filteredExtremePoints)
    end for
    optimalExtremePoints = `findOptimalExtremes`(extremePoints, n_{degree}, $r(x)$)
    errors = [$|r_i(x)|$ | $x \in$ OptimalExtremePoints]
    $e_{\max}$ = max(errors)
    $e_{\min}$ = min(errors)
    $e_{\text{condit}} = \frac{e_{\max} - e_{\min}}{e_{\min}}$
    if $e_{\text{condit}} <$ approximationParam **then**
        break
    end if
    x_{points} = optimal_extremes
end while
return coeffs, $e_{\max}$

Algorithm 8 Generate Tchebycheff Points, generatePointsInRanges

Require: lower limit *lower*, upper limit *upper*, Number of points n
Ensure: List of n Tchebycheff points in the range [lower, upper]
range = |upper − lower|
index = $[1, 2, \ldots, n]$
points = []
for i in index **do**
    $x = \frac{1}{2}\left(\cos\left(\frac{2i-1}{2n}\pi\right) + 1\right) \cdot \text{range} + \text{lower}$
    Append x to points
end for
return points

Algorithm 9 Find Extreme Points Setup findExtremePointsSetup

Require: Residual function derivative $r_i'(x)$, lower bound *lower*, upper bound *upper*
Ensure: List of extreme points *extremePoints*
Define step size sc = $\frac{|\text{upper}-\text{lower}|}{6000}$
Define precision $l = 150$ (bits)
extremePoints = `findExtremePoints`$(r_i'(x), \text{lower}, \text{upper}, \text{sc}, l)$
return extremePoints

Algorithm 10 Find Extreme Points findExtremePoints

Require: Residual function derivative $r_i'(x)$, interval $[start, end]$, step size *sc*, precision l
Ensure: List of extreme points *extremePoints*
Initialize extremePoints = []
for $x \in [\text{start}, \text{end}]$ with step size sc **do**
    if `isExtremePoint`$(r_i'(x), x, \text{sc})$ **then**
        x_l = `refineExtremePoint`$(r_i'(x), x, \text{sc}, l)$
        Append refined x_l to extremePoints
    end if
end for
return extreme_points

Algorithm 11 Check if Point is an Extreme Point isExtremePoint

Require: Residual function derivative $r_i'(x)$, point x, step size *sc*
Ensure: Boolean indicating whether x is an extreme point
Compute mult = $(r_i'(x) - r_i'(x - \text{sc})) \cdot (r_i'(x + \text{sc}) - r_i'(x))$
return mult ≤ 0

Algorithm 12 Refine Extreme Point to l-bit Precision refineExtremePoint

Require: Residual function derivative $r_i'(x)$, initial point x, step size *sc*, precision l
Ensure: Refined extreme point x_i
Initialize $x_i = x$
for $k = 1$ to l **do**
    Define search range $[x_i - \frac{\text{sc}}{2^k}, x_i, x_i + \frac{\text{sc}}{2^k}]$
    Update x_i as the argument that maximizes $|r_i'(x)|$ within the search range using `arg_max`
end for
return x_i

Algorithm 13 Find Maximum Argument in a Range

Require: Search range *searchRange*, function $f(x)$
Ensure: Point *best_x* that maximizes $f(x)$
Initialize best_x = search_range[0]
Initialize maxVal = f(best_x)
for $x \in$ searchRange **do**
    Compute val = $f(x)$
    if val > maxVal **then**
        Update maxVal = val
        Update best_x = x
    end if
end forreturn best_x

Algorithm 14 Filter Positive Minimum and Negative Maximum Points filterPosMinNegMax

Require: Set of extreme points *extremes*, second derivative *secDerivative*, error function diff(x)
Ensure: Filtered set of extreme points *filteredExtremes*
Initialize filteredExtremes = []
for extreme $\in$ extremes **do**
    Compute concav = `concavity`(secDerivative, extreme)
    if concav $\cdot$ signFunction(diff(extreme)) = 1 **then**
        Append extreme to filteredExtremes
    end if
end forreturn filteredExtremes

Algorithm 15 Determine Concavity

Require: Second derivative sec_derivative, point x
Ensure: Concavity indicator (1 for concave, -1 for convex, 0 else)
Compute secondDerivativeAtX = secDerivative(x)
if secondDerivativeAtX < 0 **then return** 1 ▷ Concave
else if secondDerivativeAtX > 0 **then return** -1 ▷ Convex
elsereturn 0 ▷ Inflection point
end if

Algorithm 16 Choose New Extreme Points findOptimalExtremes

Require: Set of extreme points B, polynomial degree d, error function $r(x)$
Ensure: Refined set of points B satisfying alternating and maximum absolute sum conditions

Initialize $i = 0$
while $i < |B| - 1$ **do**
    if sign_func$(r(B[i])) =$ sign_func$(r(B[i+1]))$ **then**
        if $|r(B[i])| < |r(B[i+1])|$ **then**
            Remove $B[i]$ from B
        else
            Remove $B[i+1]$ from B
        end if
    else
        Increment i by 1
    end if
end while
if $|B| > d + 3$ **then**
    Compute $T =$ sorted$([(|r(B[i])| + |r(B[i+1])|, i, i+1)$ for $i = 0, \ldots, |B| - 2])$
end if
while $|B| > d + 2$ **do**
    if $|B| = d + 3$ **then**
        if $|r(B[0])| < |r(B[-1])|$ **then**
            Remove $B[0]$ from B
        else
            Remove $B[-1]$ from B
        end if
    else if $|B| = d + 4$ **then**
        Add $(|r(B[0])| + |r(B[-1])|, 0, |B| - 1)$ to T and sort T
        Let $(v, i_1, i_2) = T[0]$
        Remove $B[\max(i_1, i_2)]$ and $B[\min(i_1, i_2)]$ from B
    else
        if $T[0][1] = 0$ or $T[0][2] = |B| - 1$ **then**
            if $T[0][1] = 0$ **then**
                Remove $B[0]$ from B
            else
                Remove $B[-1]$ from B
            end if
        else
            Let $(v, i_1, i_2) = T[0]$
            Remove $B[\max(i_1, i_2)]$ and $B[\min(i_1, i_2)]$ from B ▷ < Library
        end if
    end if
    Recompute $T =$ sorted$([(|r(B[i])| + |r(B[i+1])|, i, i+1)$ for $i = 0, \ldots, |B| - 2])$ ▷ < libary
end while
return B