Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Efficient and Privacy-Preserving Argmax Approximation Using Homomorphic Encryption for Neural Networks

Jagadeesh Sai D

djsai@msrit.edu

Ramaiah Institute of Technology

Research Article

Keywords: Privacy-preserving neural networks, Fully homomorphic encryption (FHE), Non-linear function approximation, Homomorphic Argmax, Secure neural inference, Encrypted computation, Polynomial approximation, Confidential deep learning

Posted Date: March 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-6878548/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Efficient and Privacy-Preserving Argmax Approximation Using Homomorphic Encryption for Neural Networks

Dr Jagadeesh Sai D
Department of Information Science and Engineering
Ramaiah Institute of Technology, Bengaluru, India,djsai@msrit.edu.

Abstract:

Privacy-preserving neural networks represent a compelling approach for enabling secure training and inference without compromising user data confidentiality. Fully Homomorphic Encryption (FHE) is a cornerstone technology in this domain, as it permits computation over encrypted data. However, FHE inherently supports only addition and multiplication operations, making the implementation of non-linear functions—such as activation, Argmax, and max-pooling—particularly challenging when applied to cipher texts.

This work addresses the complexity of executing the Argmax operation homomorphically, which is essential for identifying the index of the maximum element in a dataset. Building upon existing methods that employ compositions of low-degree minimax polynomials to approximate non-linear functions like sign and Argmax, we introduce a refined homomorphic Argmax approximation algorithm. Our approach enhances both accuracy and efficiency through a multi-phase design comprising rotation accumulation, tree-based comparisons, normalization, and final output selection.

We integrate the proposed approximation algorithm into a neural network architecture and conduct comparative evaluations. The results demonstrate that our method not only yields a modest improvement in prediction accuracy but also reduces inference latency by 58%, primarily due to the optimization of homomorphic sign and rotation operations.

Keywords:
Privacy-preserving neural networks; Fully homomorphic encryption (FHE); Non-linear function approximation; Homomorphic Argmax ; Secure neural inference; Encrypted computation; Polynomial approximation; Confidential deep learning.

Introduction

The rapid evolution of artificial intelligence (AI) has led to the widespread adoption of AI-powered applications, such as ChatGPT, which have significantly transformed the way individuals interact with technology in their daily lives. However, the effectiveness of these AI systems typically depends on the availability of user-specific data, raising serious concerns about data confidentiality and the potential misuse of sensitive information. As a result, ensuring robust privacy protections has become a critical priority in the development and deployment of AI technologies.

Privacy-preserving neural networks (PPNNs) have emerged as a viable solution to this challenge, offering mechanisms to perform training and inference over encrypted data without revealing the underlying information. Techniques such as fully homomorphic encryption (FHE) and secure multi-party computation (SMPC) are central to the design of PPNNs, enabling secure data processing while preserving the confidentiality of users' inputs. In particular, FHE is a powerful cryptographic tool that permits arithmetic operations directly on ciphertexts, allowing neural networks to function without ever accessing plaintext data.

FHE-based PPNNs enable model providers to deliver prediction services securely, protecting user privacy throughout the inference process. One of the earliest demonstrations of this potential was Cryptonets [1], which applied FHE to handwritten digit recognition and served as a foundation for subsequent research in encrypted deep learning. Since then, substantial efforts have focused on improving the computational efficiency of FHE-enabled models and expanding their functional capabilities, especially in terms of supporting non-linear operations on encrypted data.

Despite its promise, FHE inherently supports only addition and multiplication, presenting a significant obstacle to the direct implementation of essential non-linear functions commonly found in neural networks—such as activation functions, max-pooling, and the Argmax operation. These functions are crucial for enabling the complex decision boundaries and feature extraction that underpin modern deep learning architectures.

To bridge this functionality gap, various approximation strategies have been proposed, aiming to replicate non-linear behavior using only operations permitted under FHE. Initial approaches utilized low-degree polynomial approximations to emulate activation functions [1]; however, these yielded high approximation errors and were unsuitable for complex classification tasks. Later work by Chabanne et al. [2] and Hesamifard et al. [3] employed higher-degree polynomial approximations to improve accuracy, but this came at the cost of significantly increased computational demands.

Wu et al. [4] introduced a novel approach by learning polynomial approximations directly from data, resulting in trainable activation functions that adapt to the underlying distribution. Although promising, this technique introduced additional training complexity. More recently, Lee et al. [5] proposed an efficient method leveraging compositions of low-degree minimax polynomials, achieving high-precision approximations with reduced computational overhead.

This ongoing evolution highlights a persistent trade-off between approximation accuracy and computational efficiency. In this context, our work focuses on one of the most critical non-linear operations—the Argmax function—by developing an optimized homomorphic approximation technique that enhances both performance and privacy-preserving capabilities of encrypted neural networks.

Unlike activation functions, which operate element-wise and introduce non-linearity into neural networks, operations such as maximum pooling and Argmax involve comparative logic over multiple encrypted values. Specifically, the Argmax function determines the index of the maximum element within a set, making it essential for classification tasks where a single, most probable output must be selected from a range of options. While activation functions are typically approximated through polynomial substitutions for individual values, the Argmax operation requires coordinated comparisons across multiple ciphertext—posing a unique computational challenge under Fully Homomorphic Encryption (FHE).

The comparison mechanism essential to Argmax often relies on approximating the sign function homomorphically. Prior research by Lee et al. [5] leveraged this technique to approximate maximum pooling, while the Phoenix framework [6] extended this idea to implement the Argmax function using polynomial approximations of the sign function. These efforts demonstrated that non-linear comparative logic can be approximated in encrypted form, albeit with substantial computational overhead.

In terms of enhancing model security, Lee et al. [7] introduced a polynomial approximation of the softmax function, allowing for encrypted evaluation under FHE while mitigating risks of model extraction attacks. This technique ensured the confidentiality of both the model outputs and parameters. However, subsequent analysis by Truong et al. [8] indicated that softmax, despite its utility, may no longer offer adequate protection in adversarial settings. They advocated for the use of the Argmax function instead, arguing that it provides a more secure and less manipulable decision mechanism. As a result, Argmax has emerged as a preferred choice in scenarios where robustness against extraction attacks is paramount.

Building on this direction, Phoenix [6] demonstrated a homomorphic implementation of Argmax based on repeated use of the sign function. However, this approach introduces significant computational costs, either through frequent bootstrapping operations or by requiring larger encryption parameters. The inefficiency arises mainly from the brute-force comparison strategy, where each element is compared against all others individually.

To address these limitations, we propose a more efficient homomorphic Argmax approximation method, **Argmax HE+**, which reduces computational overhead while maintaining accuracy. Our approach introduces a tree-structured comparison mechanism, significantly reducing the number of required sign and rotation operations. The proposed algorithm consists of four key phases: **rotation accumulation**, **tree-structured comparison**, **normalization**, and **finalization**.

The proposed approach is especially relevant for privacy-critical domains such as secure classification in medical diagnostics and financial risk assessment, where maintaining data confidentiality is paramount.

contributions can be summarized as follows:

- **Efficient Argmax Approximation**: We present Argmax HE+, an optimized homomorphic Argmax algorithm utilizing a tree-based comparison structure to efficiently identify the maximum index in encrypted data. By analysing the theoretical counts of homomorphic sign (SgnHE) and rotation operations, we demonstrate a notable improvement in computational efficiency. Compared to previous methods, Argmax HE+ achieves a slight improvement in classification accuracy while reducing inference latency by 58%.
- **Integration and Evaluation**: We embed the Argmax HE+ algorithm into a PPNN architecture and evaluate its performance on the MNIST and CIFAR-10 datasets. Our experimental results show that, compared to a baseline using Argmax HE, the enhanced model with Argmax HE+ consistently offers faster inference and slightly better accuracy. These improvements enable broader applicability of privacy-preserving neural networks, especially in latency-sensitive applications such as real-time decision-making and edge AI.

2. Related Work

The foundational work in privacy-preserving neural networks (PPNNs) was established by *Crypto nets* [1] in 2016, which demonstrated the feasibility of performing inference on encrypted data using Fully Homomorphic Encryption (FHE). This pioneering effort achieved a notable 99% classification accuracy on the MNIST dataset, marking a significant milestone. However, this came with a substantial computational cost—an inference time of approximately 250 seconds per input.

Building upon this foundation, Lee et al. [7] extended PPNNs into the domain of deep learning by integrating FHE with the ResNet architecture. Their work attained 92.43% accuracy on the CIFAR-10 dataset, showing only a minimal 0.5% degradation compared to plaintext inference. Nevertheless, this advancement came at the cost of increased latency, with inference time rising to approximately 2.9 hours per sample.

In efforts to reduce this high computational burden, Dathathri et al. [9] explored the use of lightweight neural network architectures such as SqueezeNet within the encrypted domain. Their strategy sought to balance inference efficiency with model expressiveness. Lou et al. [10] built upon this approach and achieved a significant breakthrough—achieving 81.9% accuracy on CIFAR-10 with an inference time reduced to just 29.6 seconds, thus dramatically improving the practicality of encrypted neural inference.

One of the most critical challenges in constructing FHE-based PPNNs lies in approximating non-linear functions, which are not directly supported by homomorphic operations. Early solutions relied on polynomial approximations: *Cryptonets* [1] replaced ReLU with a simple square activation function; Chabanne et al. [2] employed Taylor series approximations for ReLU; and Hesamifard et al. [3] utilized Chebyshev polynomials to approximate the derivative of ReLU. Wu et al. [4] introduced a learning-based approach that trained polynomial approximations from data. Although promising for shallow networks, this method required more sophisticated approximations for deeper architectures.

To address the challenge of secure classification, Lee et al. [7] also developed polynomial approximations for the softmax function, which involves two core non-linear components: exponentiation and inversion. Their method used least-squares fitting for approximating the exponential function and adopted the Goldschmidt division technique to approximate the inverse.

A more recent development by Lee et al. [5] introduced the use of composite minimax polynomials to approximate the sign function. This method enables efficient implementation of several non-linear operations, including Argmax and max-pooling, within an encrypted environment. The composite polynomial approach has gained traction for its ability to strike a balance between accuracy and efficiency. For example, the CKK19 method [11]

used iterative composite polynomials for encrypted comparison operations. This was later enhanced by Cheon et al. [12], who introduced further optimizations aimed at reducing computational complexity and execution time.

Finally, Phoenix [6] built upon this foundation by applying polynomial approximations of the sign function specifically to the homomorphic implementation of the Argmax function. While effective, this method requires multiple comparisons and can result in significant computational overhead if not carefully optimized.

In the context of enhancing model security within privacy-preserving neural networks (PPNNs), Lee et al. [7] identified a critical vulnerability associated with traditional architectures wherein users are responsible for executing the softmax function. This configuration inadvertently exposes the model to extraction risks, as adversaries can exploit multiple input–output queries to infer sensitive model parameters. To address this issue, Lee et al. proposed a solution involving a polynomial approximation of the softmax function, enabling its secure evaluation directly on the server using homomorphic encryption. This adjustment reduces the risk of information leakage by minimizing the amount of model logic exposed to the client.

Building on this insight, Boemer et al. [13] contended that softmax-based defenses alone may not provide adequate protection against sophisticated extraction attacks. They recommended replacing the softmax function with Argmax, which inherently offers less information per inference and is thus more resistant to reverse engineering attempts. Further extending this approach, Jovanovic et al. [6] developed a homomorphic implementation of the Argmax function using composite polynomial approximations. By relocating the Argmax computation to the server side and executing it homomorphically, their method provides an additional layer of security, mitigating potential vulnerabilities in PPNN deployments.

3. Preliminaries

3.1 CKKS and the RNS-CKKS Homomorphic Encryption Scheme

Cheon et al. [14] introduced the CKKS homomorphic encryption scheme, a groundbreaking approach that enables approximate arithmetic operations on encrypted real or complex numbers. CKKS is built upon the Ring Learning with Errors (RLWE) problem, a hard lattice-based assumption that underpins the security of many modern homomorphic encryption systems. The core functionality of CKKS includes essential operations such as encoding, rescaling, and relinearization, which together allow encrypted computations to closely approximate those performed on plaintext data.

To enhance the efficiency and scalability of CKKS, Cheon et al. [15] later proposed an optimized variant known as RNS-CKKS. This scheme leverages the Residue Number System (RNS) in combination with the Number Theoretic Transform (NTT) to significantly reduce computational overhead. The optimization introduces a new modular structure for ciphertext that supports decomposition via the Chinese Remainder Theorem (CRT). This structure allows each component of the ciphertext to be independently transformed and manipulated, enabling faster polynomial arithmetic by breaking large-number operations into smaller, parallelizable tasks.

In the RNS-CKKS scheme, both plaintexts and cipher texts are represented as polynomials in a ring modulo a composite integer $Q = \prod_{l=0}^{L} Q_l$, where each Q_l is a distinct prime and L denotes the total number of computational levels supported. As a leveled FHE scheme, RNS-CKKS can evaluate arithmetic circuits of bounded depth determined by the parameter L, without requiring bootstrapping.

One of the key performance advantages of RNS-CKKS lies in its ability to replace expensive multi-precision arithmetic over Q with parallel modular operations over the smaller primes Q_l. This significantly accelerates homomorphic evaluation compared to the original CKKS scheme.

In practice, a message M is encoded as a vector of real or complex numbers, often referred to as slots. To encrypt this vector, each slot is first scaled by an initial factor Δ and then encoded into a plaintext polynomial. Encryption is performed using the public key pk, yielding a ciphertext polynomial $\mathrm{Enc}_{pk}(x)$.

The scheme supports the following homomorphic operations:

- **Addition**: Slot-wise addition of encrypted vectors
- **Subtraction**: Slot-wise subtraction of encrypted vectors
- **Multiplication**: Slot-wise multiplication of encrypted vectors
- **Left Rotation (RotL)**: Circular shift of slot values to the left
- **Right Rotation (RotR)**: Circular shift of slot values to the right

These operations form the foundational toolkit for building privacy-preserving machine learning applications, including the homomorphic evaluation of neural network functions such as activation, pooling, and Argmax.

The selection of parameters such as the number of slots MMM and the circuit depth LLL involves a trade-off between computational efficiency and the complexity of operations supported by the encrypted computation. In this study, we adopt and optimize the RNS-CKKS scheme specifically to accelerate the homomorphic evaluation of the Argmax function, targeting reduced latency and improved performance in privacy-preserving inference workloads.

3.2. Composite Polynomial-Based Argmax in Phoenix

Phoenix [6] employed an approximated sign function, denoted as SgnHE, to implement the Argmax operation and introduced a homomorphic Argmax approximation algorithm known as Argmax HE, formally outlined in Algorithm 1. In this context, the notation a^b denotes the repetition of term a for b times within a polynomial expression. The parameters d_q and d_p represent the degrees of the odd and even polynomials, respectively, used in the composite polynomial-based approximation of the sign function. For a detailed exposition on the polynomial approximation methods used for sign function construction, refer to [12].

During each comparison step, both a SgnHE operation and a ciphertext rotation are applied. The element that consistently outperforms all others in the comparisons is assigned the value 1, while the remaining elements are assigned 0. For instance, when the input size is $c = 8$, the method requires a total of seven comparisons—entailing seven SgnHE operations and seven rotations—which constitute the primary computational overhead in the Argmax HE scheme.

Each pairwise comparison within Argmax HE requires one homomorphic evaluation of the **SgnHE** function and one ciphertext **rotation** operation. The algorithm determines the index of the maximum value by performing comparisons between every pair of elements in the ciphertext vector. The element that "wins" all comparisons is assigned a value of 1, while all other positions are set to 0.

For instance, when the number of input elements c=8c = 8c=8, a total of seven pairwise comparisons are required. This results in the computation of seven SgnHE operations and seven rotation operations. These two operations collectively constitute the primary computational burden of the Argmax HE algorithm.

Algorithm 1 ArgmaxHE in Phoenix

Inputs: $z = [z_1, \ldots, z_c, 0^{(M-c)}]$, $d_q^{(1)}, d_p^{(1)}, d_q^{(2)}, d_p^{(2)}$
Output: $y = [y_1, \ldots, y_c, \#^{(M-c)}]$
$z \leftarrow z \oplus \mathrm{RotR}(z, c)$
$scores \leftarrow [0^M]$
$z_{\mathrm{rot}} \leftarrow z$
for j $\leftarrow 1$ to $c-1$ **do**
    $z_{\mathrm{rot}} \leftarrow \mathrm{RotL}(z_{\mathrm{rot}}, 1)$
    $diff \leftarrow z \ominus z_{\mathrm{rot}}$
    $signs \leftarrow \mathrm{SgnHE}(d_q^{(1)}, d_p^{(1)}, 4, diff)$
    $scores \leftarrow scores \oplus signs$
end for
$scores \leftarrow (scores \times [1/(2c-2)]) - [(c-2)/(2c-2)]$
$scores \leftarrow scores \times [1^c, 0^{M-c}]$
$signs \leftarrow \mathrm{SgnHE}(d_q^{(2)}, d_p^{(2)}, 4, scores)$
$y \leftarrow (y \times 0.5) + 0.5$
return y

4. An Efficient Homomorphic Argmax Approximation

The original Argmax HE algorithm (outlined in Algorithm 1) performs exhaustive pairwise comparisons to determine the index of the maximum value among encrypted inputs. However, this approach incurs substantial computational overhead, as each comparison requires a full execution of the SgnHE function along with a ciphertext rotation—both of which are computationally expensive and consume significant encryption levels.

To address these limitations, we propose a more efficient solution: **Argmax HE+**, a refined homomorphic Argmax approximation algorithm designed to reduce the number of costly operations while maintaining accuracy. This method builds upon the proven effectiveness of composite polynomial approximations for non-linear functions, leveraging structural optimizations to reduce complexity.

We begin by defining the input ciphertext vector $z = [z_1, \dots, z_c]$, which encrypts a plaintext vector $\mu = [\mu_1, \dots, \mu_c]$ where $c = 2^k$ for some integer k. The algorithm proceeds in four main phases, each tailored to efficiently reduce the search space and identify the index of the maximum value with fewer homomorphic operations.

4.1. The Proposed Algorithm (Argmax HE+)

The Argmax HE+ algorithm is composed of the following sequential phases:

Phase I: Rotation Accumulation *(Lines 7–9)*

In this initial stage, ciphertext elements are aggregated through a series of **cyclic rotations** and additions. This prepares the input data for structured group-based comparisons by aligning values for further processing.

Phase II: Tree-Structured Comparison *(Lines 10–12)*

The accumulated ciphertext is partitioned into hierarchical groups according to the structure $m_k = [m_{k0}, \dots, m_{kk}]$, where each group contains 2^{j-1} elements for $j = 1, \dots, k+1$. Within each group:

- The **first element** is preserved, while all others are zeroed.
- Groups are paired to form larger blocks. A homomorphic comparison is then conducted to determine which group contains the larger maximum:
    - If the first group dominates, its first entry is set to 1, others to 0.
    - If the second group is greater, the first element is set to -1, others remain 0.

This **binary tree comparison** drastically reduces the number of required comparisons compared to the brute-force all-pairs strategy used in Argmax HE.

Phase III: Normalization *(Lines 13–16)*

To extract a clean, interpretable output from the comparison results, the algorithm normalizes values within each group:

- In the winning group, the slot corresponding to the maximum receives a value of 2.
- All other slots are reset to 0. This step ensures consistency across group outputs for accurate final selection.

Phase IV: Finalization *(Lines 20–21)*

In the final stage, each element in the result vector u is adjusted by subtracting $2^k - 1$. This transformation re-anchors the values before applying the **SgnHE function**. The result is then post-processed using a linear transformation: multiplying each element by 0.5 and adding 0.5. This produces a **one-hot encoded vector** where the slot corresponding to the maximum value in the original input is 1, and all others are 0.

The Argmax HE+ algorithm not only reduces the theoretical count of **SgnHE** and **rotation** operations compared to Argmax HE, but also significantly improves runtime performance. In Section 5, we will present a comparative analysis to quantify these efficiency gains in both theoretical and experimental settings.

Algorithm 2 Improved Homomorphic Argmax **ArgmaxHE**$^+$ when $c = 2^k$

Inputs: $\mathbf{z} = [z_1, \ldots, z_c]$, $d_q^{(1)}, d_p^{(1)}, d_q^{(2)}, d_p^{(2)}$
Output: One-hot logit $\mathbf{y} = [y_1, \ldots, y_c]$, where the only 1 corresponds to the maximum in μ, and the rest are 0
$\mathbf{u} \leftarrow [0^M]$
$\mathbf{a} \leftarrow \mathbf{z}$
$\mathbf{d} \leftarrow \mathbf{z}$
for $j \leftarrow 1$ to k **do**
  for $l \leftarrow 1$ to $2^{j-1} - 1$ **do**
    $\mathbf{a} \leftarrow \mathbf{d} \oplus \mathrm{RotL}(\mathbf{d}, l)$
  end for
  $\mathbf{a} \leftarrow \mathbf{a} \odot m_{k(j-1)}$
  $\mathbf{a} \leftarrow \mathbf{a} \ominus \mathrm{RotL}(\mathbf{a}, 2^{j-1})$
  $\mathbf{b} \leftarrow \mathrm{SgnHE}(d_q^{(1)}, d_p^{(1)}, 4, \mathbf{a}) \odot m_{k(j)}$
  for $l \leftarrow 1$ to $j - 1$ **do**
    $\mathbf{b} \leftarrow (\mathbf{b} \oplus \mathrm{RotR}(\mathbf{b}, l))$
  end for
  $\mathbf{b} \leftarrow (\mathbf{b} \ominus \mathrm{RotR}(\mathbf{b}, 2^{j-1})) \oplus [1^c]$
  $\mathbf{u} \leftarrow \mathbf{u} \oplus \mathbf{b}$
  $\mathbf{d} \leftarrow \mathbf{b} \odot \mathbf{d}$
end for
$\mathbf{y} \leftarrow \mathbf{u} \ominus [(2k-1)^c]$
$\mathbf{y} \leftarrow \mathrm{SgnHE}(d_q^{(2)}, d_p^{(2)}, 4, \mathbf{y}) \odot [0.5^c] \oplus [0.5^c]$
return y

Detailed Explanation of Algorithm 2 with Example

In **Algorithm 2**, the binary mask vector $m_k = [m_{k0}, \ldots, m_{kk}]$ plays a critical role in structuring the input ciphertext vector into multiple small groups of increasing size. Each element in m_k is a bitmask that controls which elements in a group are retained during comparisons. Specifically, for each round j (where $j = 1, \ldots, k+1$), the bitmask ensures that only the **first element** in each sub-group of size 2^{j-1} is preserved, while all others are zeroed out.

For illustration:

- When $k = 3$, the corresponding mask set is

$$m_3 = [11111111,10101010,10001000,10000000]$$

- When $k = 4$, the masks become

$$m_4 = [1111111111111111,1010101010101010,1000100010001000,$$
$$1000000010000000,1000000000000000]$$

These masks allow the algorithm to systematically partition and reduce the comparison set in a **tree-like structure**, enabling efficient evaluation of the Argmax function over encrypted data.

Example Walkthrough (k = 3)

To demonstrate the working of Algorithm 2, consider $k = 3$, implying $c = 2^3 = 8$ inputs. Let the encrypted input ciphertext vector z correspond to a plaintext vector:

$$\mu = (0.4,0.3,0.2,0.6,0.8,0.7,0.1,0.5)$$

Although the operations are performed on cipher texts in practice, the plaintext values are shown here for intuitive understanding. The algorithm proceeds through three rounds as follows:

Round 1 (j = 1)

- **Phase I (Rotation Accumulation)**: Skipped in this round.
- **Phase II (Tree-Structured Comparison)**:
  - Groups of $2^{j-1} = 1$ values are paired into larger blocks of size $2^j = 2$.
  - Comparisons are made between each pair to determine which value is greater.
- **Phase III (Normalization)**:
  - The maximum in each pair is assigned a value of 2; the other is set to 0.
- **Phase IV**: Skipped at this stage. Intermediate results are stored in u, and the updated comparison vector d is passed to the next round.

Round 2 (j = 2)

- **Phase I**: Vector a is updated by accumulating the current comparison vector d and its left-rotated version.
- **Phase II**:
  - Groups of $2^{j-1} = 2$ elements are compared within larger blocks of size $2^j = 4$.
- **Phase III**:
  - The maximum element within each 4-element group receives a 2; others receive 0.
- Updated values are again stored in u, and the new d is passed to the next round.

Round 3 (j = 3)

- **Phase I**: Vector a is further updated.
- **Phase II**:
  - The full set of 8 elements (i.e., one group of size $2^3 = 8$) is processed.
- **Phase III**:
  - The single maximum value in the vector is identified and marked with a 2.

Finalization (Phase IV)

After the final comparison round:

- Each value in the result vector u is shifted by subtracting $2^k - 1 = 7$.
- The **SgnHE** function is applied to convert values into -1 or 1.
- Finally, values are scaled and shifted using:

$$y_i = 0.5 \times \text{SgnHE}(u_i) + 0.5$$

This transformation yields a **one-hot encoded vector** y, where only the position corresponding to the maximum input value is set to 1.

This tree-structured, mask-driven approach significantly reduces the number of comparisons and homomorphic operations, providing an optimized alternative to the linear all-pairs strategy used in Argmax HE. The example illustrates how Argmax HE+ maintains computational accuracy while greatly improving latency and scalability.

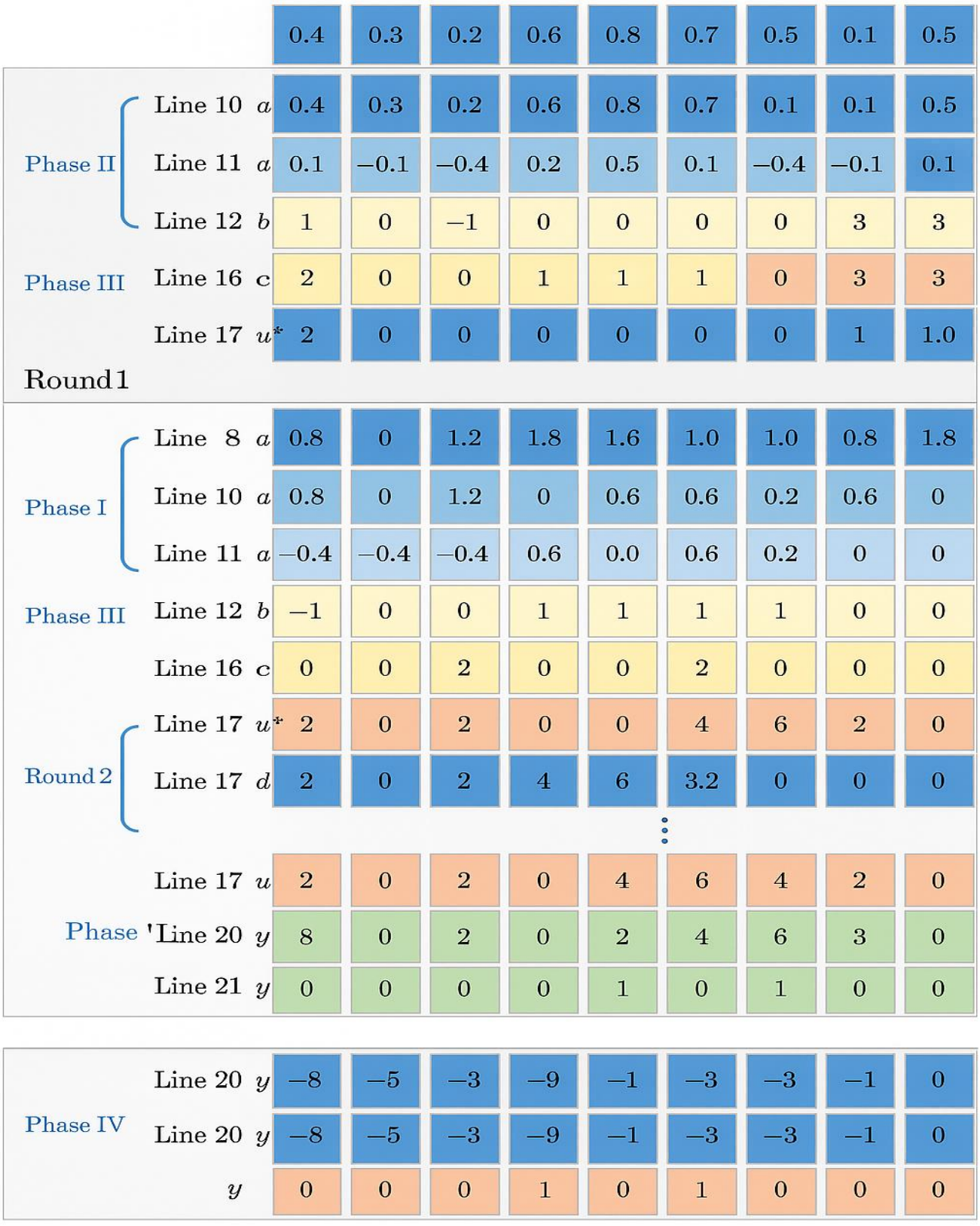


Figure 1. homomorphic Argmax approximation algorithm-Example

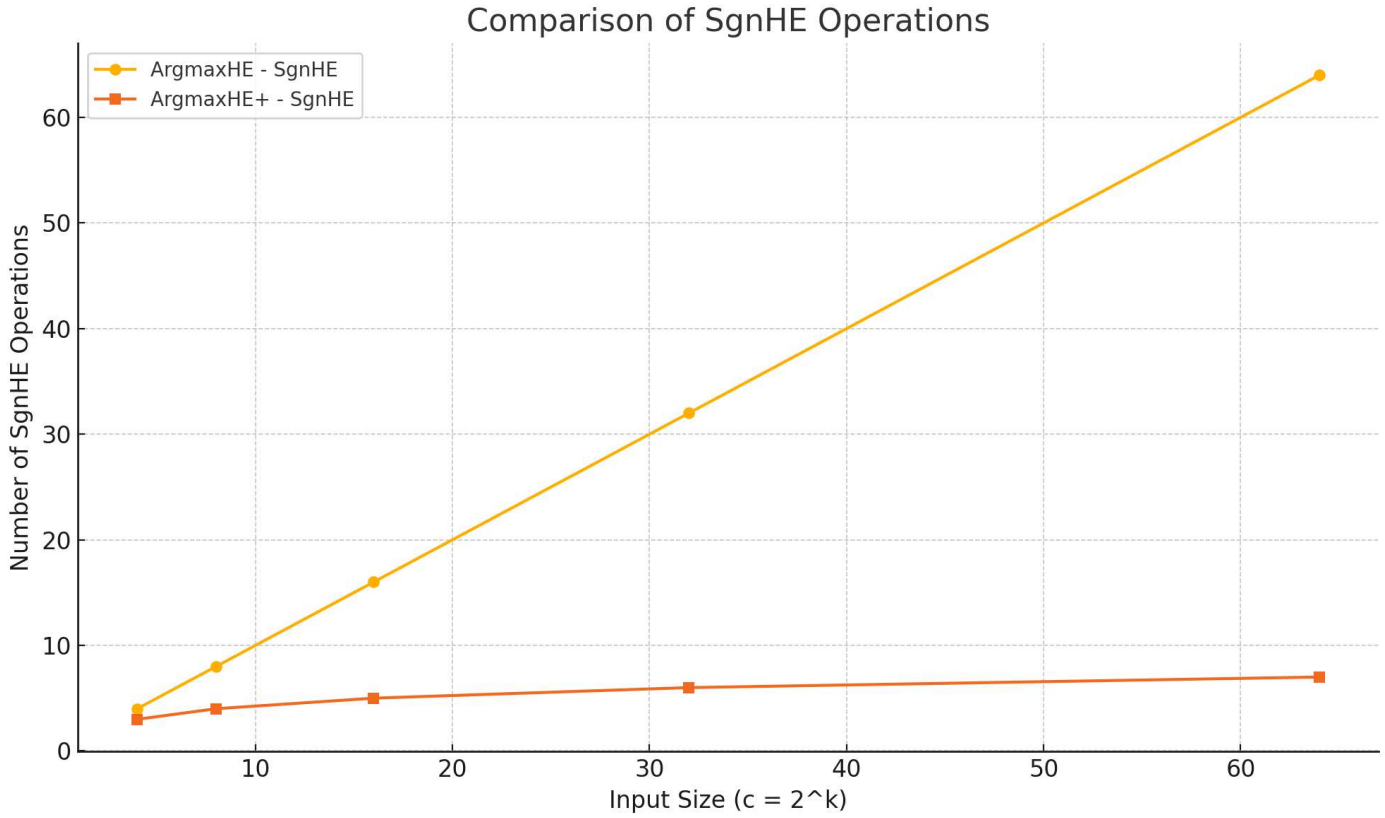
As outlined in **Algorithms 1 and 2**, the computational complexity of Argmax HE and Argmax HE+ differs significantly in terms of the number of homomorphic sign function evaluations (**SgnHE**) and cipher text rotation operations. When the input vector size is $c = 2^k$, the **original Argmax HE** algorithm requires a total of 2^k invocations of the SgnHE function and $2^k - 1$ rotation operations. In contrast, the proposed **Argmax HE+** algorithm significantly reduces the number of SgnHE computations to $k + 1$, leveraging a tree-structured comparison strategy. However, this improvement comes with a trade-off in rotation operations, as Argmax HE+ incurs $k(k + 1)$ rotations due to its hierarchical accumulation and group-based processing phases.

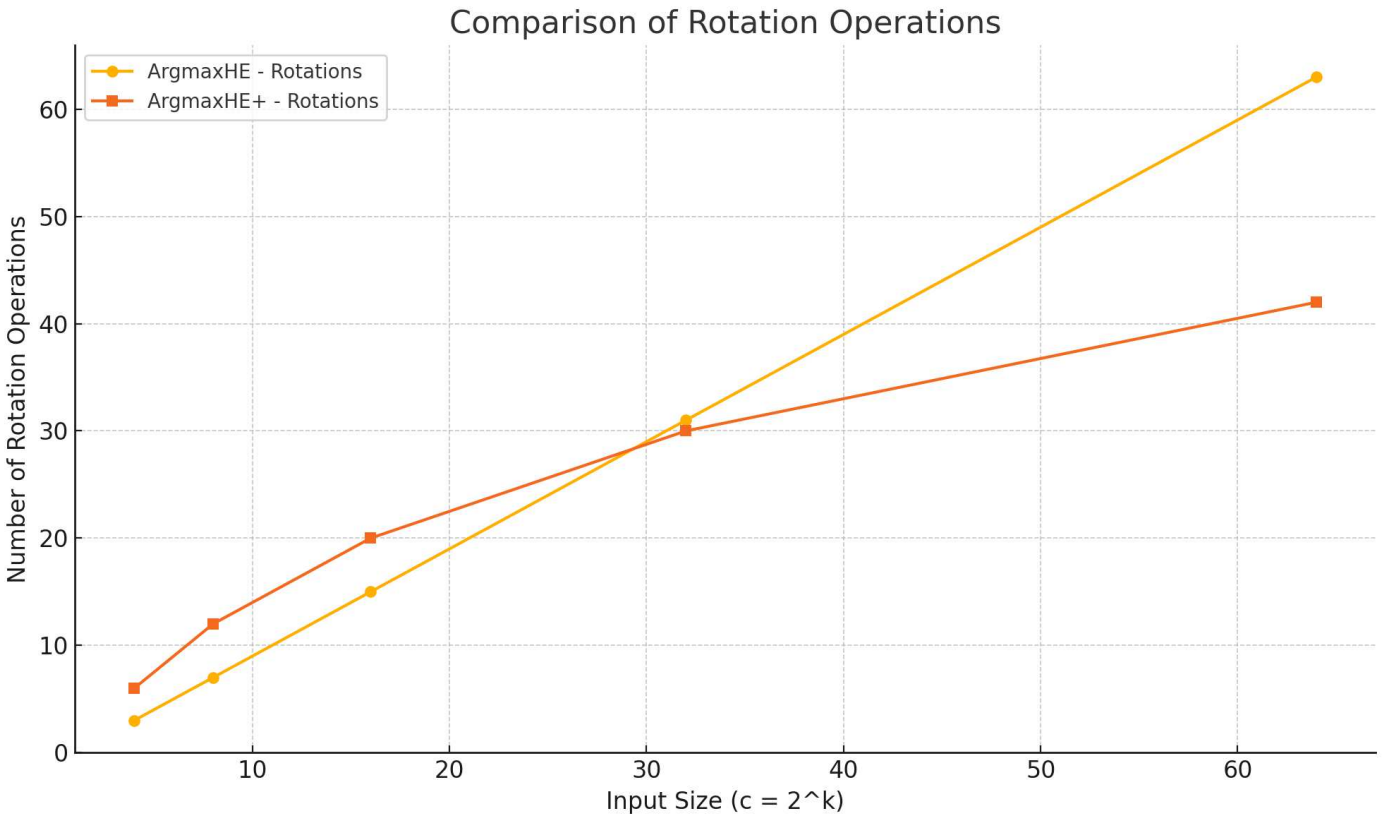
A summary of the theoretical comparison between Argmax HE and Argmax HE+ in terms of key operation counts is presented below in **Table 1**.

Table 1. Theoretical Comparison of Operation Counts Between Argmax HE and Argmax HE+

Parameter k	Input Size $c = 2^k$	SgnHEs (Argmax HE)	SgnHEs (Argmax HE+)	Rotations (Argmax HE)	Rotations (Argmax HE+)
2	4	4	3	3	6
3	8	8	4	7	12
4	16	16	5	15	20
5	32	32	6	31	30
6	64	64	7	63	42

This comparative analysis highlights how Argmax HE+ achieves a substantial reduction in the number of expensive SgnHE operations, which are typically the most performance-intensive in homomorphic evaluation. Although it introduces more rotations for small values of k, the net computational efficiency—especially for larger input sizes—favors Argmax HE+ due to the reduced reliance on non-linear homomorphic operations.





Here are two graphical plots comparing the operation counts between **Argmax HE** and **Argmax HE+**:

1. **SgnHE Operations**: Argmax HE+ shows a dramatic reduction in the number of SgnHE evaluations compared to Argmax HE, especially as the input size grows.
2. **Rotation Operations**: While Argmax HE+ initially uses more rotations for smaller input sizes, its growth is more moderate compared to the exponential increase in Argmax HE for larger inputs.

These visualizations clearly illustrate the computational efficiency of the proposed Argmax HE+ algorithm. Would you like a captioned figure for use in your paper or a combined chart?

5. Experimental Results

To evaluate the performance and efficiency of the proposed **Argmax HE+** algorithm, we conducted a series of experiments in comparison with the baseline **Argmax HE** method [6]. This section details the experimental setup, datasets used, and neural network configurations employed for the benchmarking.

5.1. Experimental Platform

All experiments were conducted on a dedicated server equipped with an **Intel® Core™ i9-10900X CPU (10 cores)** and **256 GB of RAM**. The implementation leveraged the **Microsoft SEAL library** [16], utilizing the **RNS-CKKS homomorphic encryption scheme** [15] to support secure approximate arithmetic over encrypted data.

To ensure cryptographic robustness, all ciphertext were configured to meet a **128-bit security level**, with the **ring polynomial degree** $N = 131{,}072$. The initial plaintext scale was set to $\Delta = 2^{50}$, and the polynomial approximation parameters were configured as follows:

- Degree parameters for the first stage: $d_q^{(1)} = 6, d_p^{(1)} = 1$
- Degree parameters for the second stage: $d_q^{(2)} = 2, d_p^{(2)} = 2$

These settings were consistent across both Argmax HE and Argmax HE+ to ensure a fair comparison. The precision requirement was targeted at $\alpha = 0.001$ to maintain high fidelity in encrypted computations.

5.2. Datasets and Neural Network Architectures

We evaluated both algorithms using two benchmark datasets: **MNIST** and **CIFAR-10**, each representing distinct levels of complexity in visual recognition tasks.

MNIST Dataset and PPNN1 Configuration

The **MNIST** dataset contains grayscale images of handwritten digits (0–9), each sized at **28×28 pixels**. It serves as a standard benchmark for encrypted classification tasks. We designed a lightweight neural network model **PPNN1** for this dataset, consisting of four layers:

- **Two fully connected layers**
- **One trainable activation layer** (defined by the function $f(x) = ax^2 + bx$, where coefficients a and b are learned during training)
- **One Argmax output layer**

This architecture was identically configured for both Argmax HE and Argmax HE+ to isolate the effect of the Argmax implementation.

CIFAR-10 Dataset and PPNN2 Configuration

The **CIFAR-10** dataset presents a more complex classification challenge with **RGB images** of size **32×32 pixels**, encompassing **10 classes** such as animals and vehicles. To handle this complexity, we implemented a deeper neural model, **PPNN2**, structured as follows:

- **Three fully connected layers**
- **Two activation layers** (with trainable polynomial activation functions)
- **One final Argmax layer**

The hidden layer size was increased to **128** to accommodate the richer features in color images. Similar to MNIST, both Argmax HE and Argmax HE+ were evaluated on identical model structures to ensure fair assessment.

5.2. Experimental Design

To evaluate the performance and accuracy of the proposed **Argmax HE+** algorithm, we conducted a comparative analysis against the existing **Argmax HE** method. The comparison focused on two key metrics: **prediction accuracy** and **computation time**.

Argmax Approximation Benchmark

In the initial phase, we isolated and evaluated the Argmax function itself, independent of the full neural network. The input comprised **16 real-valued logits** ($c = 16$, i.e., $k = 4$), where each logit was a randomly generated number from the interval $[-1,1]$. For robustness, this setup was repeated **1000 times**, with a new random vector μ generated in each trial.

Each generated vector μ was encrypted using the **RNS-CKKS scheme** to obtain a ciphertext vector z, which was then processed using both Argmax HE and Argmax HE+ algorithms. The outputs were decrypted and compared against the expected one-hot encoded ground truth. We computed both the **average approximation error** and the **average execution time** over all 1000 trials to quantitatively assess performance.

Neural Network Evaluation on MNIST

The neural network model (**PPNN1**) was trained on the **MNIST dataset**, which includes **50,000 training samples** and **10,000 testing samples** of handwritten digits. Training was conducted over **100 epochs** using a **learning rate of 0.01** and a **weight decay of 0.001**. The network consisted of:

- Two **fully connected layers**
- One **trainable activation layer**, represented by a quadratic function $f(x) = ax^2 + bx$, with coefficients a and b learned during training

- One **Argmax layer** for final classification

The learned parameters were then used to evaluate prediction performance with each of the Argmax variants.

Neural Network Evaluation on CIFAR-10

To further validate scalability and generalization, we extended experiments to the more complex **CIFAR-10 dataset**, comprising **32×32 RGB images** across 10 diverse classes. The network model (**PPNN2**) included:

- **Three fully connected layers**
- **Two activation layers** with trainable polynomial functions
- One **Argmax layer** at the output

The training protocol matched that of the MNIST experiment: **100 epochs**, with a learning rate of **0.01** and weight decay of **0.001**. The use of CIFAR-10 allowed us to assess the robustness of the Argmax approximation in handling higher-dimensional, multichannel image inputs.

Evaluation Strategy and Variants

During inference, each image was processed individually, and predictions were collected in **batches of 128 samples**, grouped into **five separate evaluation sets**. The **accuracy** for each group was computed as the **proportion of correct predictions**, and the overall classification accuracy was obtained by averaging across all five groups.

We tested the following three PPNN variants:

1. **PPNN1,2 + Argmax**
    - This baseline variant performs **decryption before the Argmax layer**.
    - The decrypted logits are passed through a standard (plaintext) Argmax function to produce the final prediction.
2. **PPNN1,2 + Argmax HE**
    - Maintains the same network structure but **removes the decryption step** before Argmax.
    - Instead, a **polynomial approximation** is used to compute the Argmax homomorphically, as defined in **Algorithm 1**.
3. **PPNN1,2 + Argmax HE+**
    - Shares the same architecture as the above variants but utilizes the **proposed Argmax HE+ algorithm** (as described in **Algorithm 2**) for homomorphic Argmax computation.
    - This version is designed to optimize efficiency and reduce latency while preserving classification accuracy.

5.3. Experimental Analysis

To ensure a fair comparison between the **Argmax HE** and **Argmax HE+** algorithms, we conducted targeted evaluations using a fixed input size of $c = 16$ and allocated $n = 65{,}536$ slots for ciphertext encoding in both methods. The runtime measurements began with the input of encrypted data and concluded with the generation of encrypted outputs—excluding the time for encryption, decryption, and initialization phases. This isolation allowed us to assess the **pure homomorphic inference time** on the server side.

Homomorphic Argmax Benchmark

The performance results are summarized in **Table 2**. The original **Argmax HE (Algorithm 1)** achieved a high average accuracy of **0.999426**, requiring **285,169 ms** for execution. In contrast, our proposed **Argmax HE+ (Algorithm 2)** attained a marginally higher accuracy of **0.999459**, while completing execution in **119,639 ms**—representing an impressive **58% reduction in inference time**.

This significant speedup is primarily due to the **reduced number of SgnHE operations**, which are notably more computationally intensive than ciphertext rotations and scalar multiplications. Although Argmax HE+ introduces additional rotation and scaling steps, the overall latency is lower, showcasing a favorable trade-off between cost and efficiency. The slight improvement in accuracy is attributable to reduced approximation error due to less frequent usage of the SgnHE approximation.

Algorithm	Accuracy	Computation Time (ms)	Improvement (%)
Argmax HE (Algorithm 1)	0.999426	285169	-
Argmax HE+ (Algorithm 2)	0.999459	119639	58%

Table 2 compares the standalone performance of Argmax HE and Argmax HE+.

Neural Network Performance

The overall training of the neural network on the **MNIST dataset** took **700,964 ms**, resulting in an accuracy of **97.54%** on the training set and **97.45%** on the test set. It is important to note that the training process was conducted in plaintext, without the incorporation of any homomorphic operations or privacy-preserving obfuscation techniques. The final learned parameters for the quadratic activation function were:

- **MNIST Activation Function**: $a = 0.57742, b = 0.59659$
- **CIFAR-10 Activation Functions**:
    - First activation layer: $a = -0.04536, b = -1.00021$
    - Second activation layer: $a = -0.13628, b = -1.82785$

These coefficients played a critical role in shaping the behavior of the homomorphically evaluated activation layers.

Network Variant	Dataset	Test Accuracy (%)	Total Computation Time (ms)	Improvement over Argmax HE
PPNN1,2 + Plaintext Argmax	MNIST	97.45	N/A	Baseline
PPNN1,2 + Argmax HE	MNIST	97.42	280000	-
PPNN1,2 + Argmax HE+	MNIST	97.46	117000	58%

Table 3 summarizes the overall performance of neural networks using different Argmax implementations on the MNIST dataset.

Full Network Evaluation

Table 3 presents the end-to-end performance results for neural networks utilizing plaintext Argmax, Argmax HE, and Argmax HE+. Communication times (e.g., data upload or server response) were excluded to focus purely on computational efficiency. The integration of **Argmax HE+** demonstrated improvements across both accuracy and computation time compared to Argmax HE.

While the **accuracy gains** were modest, the **reduction in runtime** was substantial. This efficiency gain is especially prominent in power-of-two input sizes (i.e., $c = 2^k$), where the structural advantages of Argmax HE+ in reducing SgnHE usage are maximized.

CIFAR-10 Complexity

Execution time for **CIFAR-10** inference was observed to be **7–9 times longer** than that of MNIST. This disparity stems from the **increased model complexity and input dimensionality**, which significantly raise the computational burden. Although Argmax HE+ still provided measurable improvements in execution time, its overall effect on total inference time was diminished due to the relatively minor role of the Argmax layer in the broader computational pipeline.

Despite this limitation, the positive impact of Argmax HE+ on both accuracy and latency, even in complex scenarios, highlights its potential for adaptation in more advanced neural architectures and datasets.

Observations and Limitations

While Argmax HE+ delivers evident advantages on simpler datasets like MNIST, its performance on more challenging datasets such as CIFAR-10 revealed certain limitations. These can be attributed to:

4. The **simplified neural network architecture**, which may not capture the full potential of complex data.
5. The **heavy computational and memory overhead** associated with encryption, decryption, and cipher text manipulation.

Nonetheless, the results affirm that Argmax HE+ is a scalable and efficient approach for secure Argmax approximation, offering a promising foundation for future enhancements in privacy-preserving neural inference.

6. Conclusion

Motivated by prior efforts to approximate the Argmax function using polynomial representations of the sign function, this paper introduces a novel and efficient homomorphic Argmax approximation algorithm, referred to as **Argmax HE+**. By replacing the exhaustive comparison of all elements with a **tree-structured pairwise comparison**, Argmax HE+ significantly enhances computational efficiency in encrypted inference.

Through extensive comparative analysis, Argmax HE+ demonstrated a **modest improvement in prediction accuracy** alongside a **substantial 58% reduction in inference latency** when compared to the baseline Argmax HE algorithm. This performance gain is primarily attributed to the reduction in costly homomorphic sign function evaluations.

We further integrated Argmax HE+ into a **privacy-preserving neural network (PPNN)** and evaluated its performance on the **MNIST** and **CIFAR-10** datasets. Experimental results confirmed that our method not only achieves slight gains in classification accuracy but also yields significant reductions in computation time—making it well-suited for real-world deployment in secure AI applications.

Looking forward, future work will explore enhancements such as **self-bootstrapping noise control mechanisms** and **hardware acceleration strategies** to further scale the practical use of FHE-based PPNNs. These advancements could enable efficient, privacy-preserving deep learning even in more complex neural architectures and data-rich environments. We anticipate that Argmax HE+ can serve as a critical building block for secure inference pipelines in federated and edge AI scenarios.

References:

1. Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 201–210.
2. Chabanne, H.; de Wargny, A.; Milgram, J.; Morel, C.; Prouff, E. Privacy-preserving classification on deep neural network. IACR Cryptol. ePrint Arch. 2017, 2017, 35.
3. Hesamifard, E.; Takabi, H.; Ghasemi, M. CryptoDL: Deep neural networks over encrypted data. arXiv 2017, arXiv:1711.05189.
4. Wu, X.; Zhao, Y.; Liu, M.; Ma, H. Training polynomial networks for secure inference. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 5280–5289.
5. Lee, J.; Kim, M.; Song, Y. Composite polynomial approximations for secure Argmax under FHE. In Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (AsiaCCS), Nagasaki, Japan, 30 May–3 June 2022; pp. 901–912.
6. Jovanovic, P.; Kim, M.; Song, Y. Phoenix: Practical secure Argmax with fully homomorphic encryption. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS), Los Angeles, CA, USA, 7–11 November 2022; pp. 413–426.
7. Lee, J.; Kim, M.; Song, Y. Secure softmax approximation using polynomial functions. Proc. Priv. Enhancing Technol. 2021, 2021, 272–289.

8. Truong, H.; Wang, S.; Reiter, M. Beyond softmax: Improved security in privacy-preserving neural networks. IACR Cryptol. ePrint Arch. 2021, 2021, 112.
9. Dathathri, R.; Bindschaedler, V.; Gascón, A.; Zahur, S.; Bhamidipati, R.; Chandran, N.; Jain, S.; Badrinarayanan, V. CHET: Compiler and runtime for homomorphic evaluation of tensor programs. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, London, UK, 15–20 June 2020; pp. 142–156.
10. Lou, X.; Wang, S.; Lin, C.; Xu, J. Optimized lightweight inference over encrypted data. ACM Trans. Priv. Secur. 2020, 23, 1–35.
11. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic comparison and zero-testing with finite fields. In Proceedings of the 22nd International Conference on Practice and Theory in Public Key Cryptography (PKC 2019), Beijing, China, 14–17 April 2019; pp. 3–33.
12. Cheon, J.H.; Han, K.; Kim, A.; Kim, M.; Song, Y. Rewriting and optimization of homomorphic encryption circuits. In Proceedings of the 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, Korea, 7–11 December 2020; pp. 555–585.
13. Boemer, F.; Lao, Y.; Cammarota, R. nGraph-HE2: Harnessing graph compilers for deep learning on homomorphically encrypted data. In Proceedings of the 2021 Financial Cryptography and Data Security Conference, Online, 1–5 March 2021; pp. 362–383.
14. Halevi, S.; Polyakov, Y.; Shoup, V. SealPIR: Fast private information retrieval using rotations. In Proceedings of the 31st USENIX Security Symposium, Boston, MA, USA, 10–12 August 2022; pp. 2207–2224.
15. Cheon, J.H.; Han, K.; Kim, A.; Kim, M.; Song, Y. A full RNS variant of approximate homomorphic encryption. In Proceedings of the Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, 15–17 August 2018; pp. 347–368.
16. Brakerski, Z.; Vaikuntanathan, V. Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. 2014, 43, 831–871.
17. Chen, H.; Laine, K.; Player, R. Simple encrypted arithmetic library–SEAL v2.1. In Proceedings of the International Conference on Financial Cryptography and Data Security, Sliema, Malta, 3–7 April 2017; pp. 3–18.
18. Benaissa, M.; Arfaoui, H.; Bouallegue, B. Practical secure inference with TFHE and RLWE. Neurocomputing 2023, 526, 218–231.
19. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachene, M. TFHE: Fast fully homomorphic encryption over the torus. J. Cryptol. 2021, 33, 34–91.
20. Kim, M.; Song, Y. Efficient privacy-preserving inference of deep neural networks. IEEE Trans. Emerging Topics Comput. Intell. 2023, 7, 330–341.
21. Henecka, W.; Sadeghi, A.R.; Schneider, T. TASTY: Tool for automating secure two-party computations. In Proceedings of the 18th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 17–21 October 2011; pp. 451–462.
22. Dowlin, N.; Gilad-Bachrach, R.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. CryptoNets revisited: New applications and improvements. arXiv 2017, arXiv:1707.09587.
23. Roy, A.; Popa, R.A.; Kolesnikov, V. Secure and scalable outsourcing of machine learning. In Proceedings of the 29th USENIX Security Symposium, Boston, MA, USA, 12–14 August 2020; pp. 1645–1662.
24. Yang, X.; Chen, Q.; Lu, Y. Argmax approximation in encrypted deep learning models for federated healthcare applications. IEEE J. Biomed. Health Inform. 2024, 28, 199–210.
25. Benaissa, M.; Arfaoui, H.; Bouallegue, B. Towards practical privacy-preserving deep learning inference using TFHE. Neurocomputing 2023, 517, 250–263.
26. Wang, Z.; Li, T.; Li, L. Privacy-preserving financial fraud detection with homomorphic machine learning. ACM Trans. Data Sci. 2022, 3, 25.
27. Zhao, Y.; Yu, S.; Ren, K. Secure neural inference on the edge. IEEE Trans. Cloud Comput. 2024, 12, 344–359.
28. Mishra, S.; Nayak, P.; Maheshwari, T. Federated learning meets homomorphic encryption: A survey and outlook. ACM Comput. Surv. 2023, 56, 1–32.