Privacy-Preserving Embedding via Look-up Table Evaluation with Fully Homomorphic Encryption

Jae-yun Kim [1] Saerom Park [2] Joohee Lee [3] Jung Hee Cheon [1]

Abstract

In privacy-preserving machine learning (PPML), homomorphic encryption (HE) has emerged as a significant primitive, allowing the use of machine learning (ML) models while protecting the confidentiality of input data. Although extensive research has been conducted on implementing PPML with HE by developing the efficient construction of private counterparts to ML models, the efficient HE implementation of embedding layers for token inputs such as words remains inadequately addressed. Thus, our study proposes an efficient algorithm for privacy-preserving embedding via look-up table evaluation with HE (HELUT) by developing an encrypted indicator function (EIF) that assures high precision with the use of the approximate HE scheme (CKKS). Based on the proposed EIF, we propose the CodedHELUT algorithm to facilitate an encrypted embedding layer for the first time. CodedHELUT leverages coded inputs to improve overall efficiency and optimize memory usage. Our comprehensive empirical analysis encompasses both synthetic tables and real-world large-scale word embedding models. CodedHELUT algorithm achieves amortized evaluation time of 0.018-0.242s for GloVe6B50d, 0.104-01.298s for GloVe42300d, 0.262-3.283s for GPT-2 and BERT embedding layers while maintaining high precision (16 bits).

[1]Department of Mathematical Sciences, Seoul National University, Seoul, South Korea [2]Department of Industrial Engineering, Ulsan National Institute of Science and Technology, Ulsan, South Korea [3]Department of Convergence Security Engineering, Sungshin Women's University, Seoul, South Korea. Correspondence to: Saerom Park <srompark@unist.ac.kr>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1. Introduction

Fully Homomorphic Encryption (FHE) introduced by (Gentry, 2009) is a cryptographic primitive that enables computations over encrypted data without decryption. Machine learning (ML) has permeated various domains with significant privacy implications (Chen et al., 2017; McMahan et al., 2023; Tanuwidjaja et al., 2020). When an outsourced computation is required while maintaining data privacy, building privacy-preserving machine learning (PPML) with FHE facilitates the utilization of sensitive data. However, most FHE schemes only support limited operations such as multiplication, addition, and slotwise rotation with increased memory and computations. Given these limitations, combining PPML with FHE presents a significant challenge as ML models often involve more advanced operations.

Among the diverse range of FHE schemes (Brakerski et al., 2011; 2014; Brakerski, 2012; Fan & Vercauteren, 2012), the CKKS scheme (Cheon et al., 2017) stands out as a prominent candidate for PPML studies (Kim et al., 2018; 2020; Boemer et al., 2019; Park et al., 2019; 2022; 2020; Byun et al., 2023), due to its plaintext space, which is adept at handling large-sized real (or complex) vectors and offers slot-wise approximate computations in a Single Instruction, Multiple Data (SIMD) manners. This suitability has encouraged the widespread adoption of CKKS in PPML, with the development of algorithms for supporting advanced operations such as inversion (Cheon et al., 2017), comparison (Cheon et al., 2020), and statistics for various data types (Lee et al., 2023).

While prior PPML research using FHE has focused on making ML algorithm evaluation HE-compatible, implementing PPML for NLP tasks poses a new challenge. Unlike other unstructured data such as images and audio that can be readily represented as numerical vectors, NLP models often rely on memory-intensive processes to generate word or token vectors, heavily involving Look-up Table (LUT) operations with large-sized inputs. Thus, although we can make the rest of ML evaluations HE-friendly, representing client-provided natural language as HE ciphertexts can lead to impractical ciphertext sizes and execution times. Alternatively, offloading computations to the client by sharing partial model information for numerical embedding calculations is a tempting solution, but it would not resolve the privacy concern on the server's word embedding model.

In this study, we aim to develop an efficient coded Look-up Table evaluation mechanism with FHE (CodedHELUT), which enables enhanced end-to-end encryption for the embedding layer. Thus, our method enables the server to protect their proprietary embeddings as well as the clients to protect their data privacy. Our approach comprises three key components: (i) constructing an Encrypted Indicator Function (EIF) as a foundational element, (ii) developing LUT evaluation using EIF for small-sized tables, and (iii) proposing a coded LUT evaluation method based on compressed coding (Shu & Nakayama, 2018) for handling large-sized tables. This algorithm can serve as an FHE module in any ML model evaluation that necessitates the use of a substantial number of word or token embeddings.

Our main contributions can be summarized as follows:

- We proposed a novel EIF based on the CKKS scheme, incorporating a noise-cleansing technique. We provide a theoretical analysis of the noise bound for the input range and cost budget, demonstrating the effectiveness of our algorithm compared to existing methods. Also, we discussed further application of our novel EIF.
- We developed an efficient privacy-preserving LUT evaluation mechanism utilizing EIF to address the challenge of end-to-end encryption for the embedding layer. Our method enables LUT evaluation for large-sized tables by combining the basic LUT method for small-sized tables with coded-compression techniques.
- We implemented our Coded LUT mechanism, successfully evaluating the embedding layer with CKKS for the first time. To illustrate the efficiency, we implemented Word Embedding of size 1.9M vocabularies with CKKS, achieving 16-bit precision without bootstrapping.

2. Preliminaries

This section provides the necessary background for understanding our study including the CKKS HE scheme and the coded look-up table method to compress the large LUT computation. We provide a brief survey of related work on EIF, LUT, and NLP with CKKS in Appendix C.

For notational convenience, we employ the same function notation for both unencrypted and encrypted cases. For example, consider f as a polynomial function. In this context, $f(x)$ denotes the evaluation of the polynomial in the unencrypted message x, whereas $f(\mathsf{ct})$ represents the polynomial evaluation on the ciphertext ct. Likewise, for a constant c, the expressions $c \pm \mathsf{ct}$ and $c \cdot \mathsf{ct}$ denote the operations of constant addition/subtraction and multiplication with ct, respectively. In Table 3 of Appendix B.1, we provide a comprehensive summary of the notations used for parameters in the CKKS scheme and our algorithms.

2.1. Homomorphic Encryption

Homomorphic Encryption is a cryptographic primitive allowing arithmetic operations over encrypted data. In this paper, we use CKKS scheme (Cheon et al., 2017) for HE, which supports approximate computations over complex numbers such as addition, and Hadamard multiplication. A detailed description of CKKS is given in Appendix B.2, and we briefly introduce the basic notations for further discussion in this subsection.

The CKKS scheme utilizes an encoding scheme $\mathsf{Ecd} : \mathbb{C}^{N/2} \to \mathcal{R}$ which is an (approximate) inverse of the decoding scheme $\mathsf{Dcd} : \mathcal{R} \to \mathbb{C}^{N/2}$ defined by $p(X) \mapsto \Delta^{-1} \cdot \sigma(p(X))$, where $\Delta > 0$ is a positive real number called scaling factor in (Cheon et al., 2017) and σ is the complex canonical embedding map.

- $\mathsf{Enc}(\mathsf{pk}, \mathsf{pt}) \to \mathsf{ct}$: Using public key pk, it encrypts the plaintext pt to a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{pt}$: Using secret key sk, it decrypts a ciphertext to a plaintext pt.
- $\mathsf{Add}(\mathsf{ct}_0, \mathsf{ct}_1) \to \mathsf{ct}_{\mathsf{add}}$, $\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_0, \mathsf{ct}_1) \to \mathsf{ct}_{\mathsf{mult}}$: Outputs a ciphertext which is decrypted to the addition/multiplication of input ciphertexts' messages.
- $\mathsf{Rescale}_{\ell \to \ell'}(\mathsf{ct}) \to \mathsf{ct}'$: it shortens the lower bits after multiplication.
- $\mathsf{Rot}(\mathsf{rk}, \mathsf{ct}, i) \to \mathsf{ct}'$: it cyclic-shifts the slots of a plaintext vector by i positions.

Bit consumption, also known as multiplication depth, must be meticulously monitored during Mult and $\mathsf{Rescale}$ operations because reduced ciphertext bits can hinder subsequent operations. Although bootstrapping (Cheon et al., 2018a) can restore consumed bits, we prioritize stringent control of bit consumption because of its significant computational overhead.

We remark that the noise occurs after each algorithm since CKKS is an approximate HE. To track the noise efficiently, we use two tags v and B with a ciphertext ct, where $\langle \mathsf{sk}, \mathsf{ct} \rangle \bmod q = \mathsf{pt} + e$, $\|\mathsf{pt}\|_\infty^{\mathsf{can}} / \Delta \le v$ and $\|e\|_\infty^{\mathsf{can}} / \Delta \le B$[1], denote (ct, v, B) as valid ciphertext. The computation of noise bound B after each operation is presented in Appendix B.3. For message bound v and noise bound B, we call $\beta = B/v$ *relative noise*. We follow the relative noise argument in (Cheon et al., 2017).

[1]For cannonical embedding map $\sigma : \mathcal{R} \mapsto \mathbb{C}^{N/2}$ and plaintext polynomial $p(X)$, $\|p(X)\|_\infty^{\mathsf{can}}$ be canonical-norm, equal to $\|\sigma(p(X)\|_\infty$.

2.2. Efficient Embedding Through Coded Compression

To enhance the viability of privacy-preserving embedding using FHE, it is significant to reduce the memory footprint. To achieve it, we can use the coded compression technique, as introduced in (Shu & Nakayama, 2018), which facilitates the decomposition of embeddings into more compact compositional code embeddings, significantly minimizing memory requirements.

For positive integers k and d, consider a dictionary of size k and its index set $\mathbb{Z}_k$, and a lookup table $T(i) : \mathbb{Z}_k \to \mathbb{R}^d$ which maps the index of a token to its d-dimensional embedding. Then the compression of T involves the following components:

- **Coded Input:** Let $p < k$ be a positive integer and $l := \lceil \log_p k \rceil$. Each token is represented by a input index $x \in \mathbb{Z}_k$, which can be decomposed to coded input $(x_0, \dots, x_{l-1}) \in \mathbb{Z}_p^\ell$ where $x = \Sigma_{i=0}^{l-1} p^i x_i$.
- **Compressed Embedding:** The compressed embedding $\tilde{T}(x)$ is obtained by summing up $T_i(x_i)$, where $T_i : \mathbb{Z}_p \to \mathbb{R}^d$.

$$T(x) \approx \sum_{i=0}^{l-1} T_i(x_i) := \tilde{T}(\{x_i\}_{i=0}^{l-1}) \tag{1}$$

- **Code and Embedding Learning:** The discrete codes $x \in \mathbb{Z}_k$ and the corresponding codebooks $T_i(\cdot)$ $(i = 0, \dots, l-1)$ are jointly learned by using the Gumbel-softmax trick with simple neural networks and a non-symmetric encode-decode structure. Then this compressed embedding is denoted as $p \times l$ coding.

On this basis, utilizing this high performance with a high compression rate, we leverage the coded compression method to devise a large-scale lookup table evaluation using FHE, enhancing the computational efficiency of HE operations, as discussed in Section 4.3.

3. Encrypted Indicator Function

In this section, we introduce a novel Encrypted Indicator Function (EIF), a fundamental component for facilitating precise encrypted LUT evaluation with CKKS. EIF empowers the processing of encrypted conditional statements, yielding outputs that seamlessly integrate into subsequent computations. In the context of HELUT, EIF plays a critical role in identifying the input value within the table, thereby allowing for retrieval of the corresponding LUT outputs and ultimately yielding the desired results. Nevertheless, non-smooth indicator functions pose significant challenges when combined with the inherent noise associated with the CKKS scheme. To guarantee acceptable precision in encrypted LUT evaluations that utilize one-hot encoded vectors generated by the EIF, meticulous noise control is imperative. Moreover, our EIF is more efficient than other design choices based on (Cheon et al., 2020; Lee et al., 2023), discussed in detail in Appendix D.3.1 and E.4

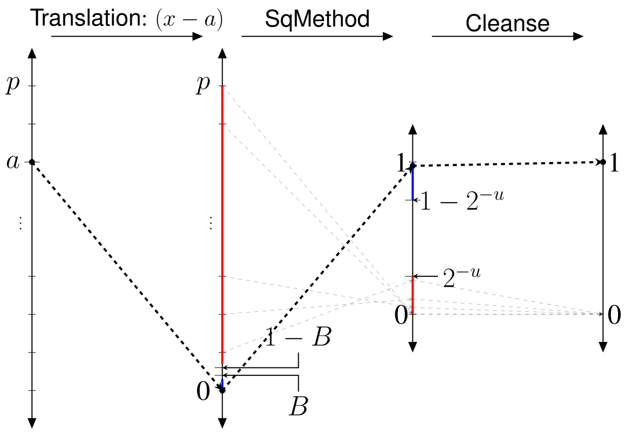

Figure 1. Illustrative Example of Encrypted Indicator Function.

We consider an indicator function $\delta_a : \mathbb{Z}_p \to \{0, 1\}$ mapping $a \in \mathbb{Z}_p$ to 1 and other values to 0. Our objective is to compute this non-smooth indicator function effectively while managing the associated noise levels. To achieve this, we have developed the EIF $\mathsf{Indicator}_a(x)$, consisting of two sub-algorithms: Cleanse and SqMethod. We devise SqMethod algorithm that maps the input element $x \in \mathbb{Z}_p$ to the vicinity of 0 for $x \neq a$ and 1 for $x = a$ with small approximation error [2], through a series of rescaling and recursive squaring operations, as elaborated in Section 3.1. Subsequently, the sub-algorithm Cleanse applies a noise-aware rounding mechanism to the output of SqMethod, effectively rounding the results from SqMethod. The properties of Cleanse in approximation error reduction are discussed in Section 3.2. As a result, by establishing theoretical approximation error bounds for both sub-algorithms and providing a method for parameter selection, we ensure that the overall noise introduced by the EIF algorithm remains within the targeted precision threshold. Figure 1 illustrates the entire process of EIF.

3.1. Square Method Function

Let $\delta_a : \mathbb{Z}_p \to \{0, 1\}$ denote an indicator function for $a \in \mathbb{Z}_p$. For the case of δ_0, we can construct an approximate indicator function defined on $[0, p)$ as follows:

$$\delta_0(x) \approx (1 - 2 \cdot x/p)^{2^r} \tag{2}$$

[2]The approximation error is defined as the difference between δ_a and its polynomial approximation by CKKS, which is discriminated from CKKS evaluation noise discussed in Section B.3.

where $x \in [0, p)$ and $r \in \mathbb{Z}_+$, and its limit becomes

$$\lim_{r\to\infty} (1 - 2 \cdot x/p)^{2^r} = \begin{cases} 1 & x = 0 \\ 0 & x \in (0, p) \end{cases} = \delta_0(x). \quad (3)$$

However, even when we evaluate this formula using CKKS for a large r, it does not converge to δ_0, primarily due to the noise associated with the encryption of 0. This issue arises because $1 - \mathsf{ct}/p$ is not a ciphertext of exact 1 even if ct is the encryption of 0.

To support δ_a for $a \neq 0$, we construct a modified formula designed to ensure the bounding of the noises. Moreover, the remaining noise will be reduced by the Cleanse function (as detailed in Section 3.2). The SqMethod function is defined as follows:

$$\mathsf{SqMethod}^a_{r,p}(x) = \left(1 - 2 \cdot \frac{(x-a)^2}{p^2}\right)^{2^r}. \quad (4)$$

To ensure an approximation error bounded by 2^{-u} when approximating the indicator function using SqMethod, we establish a condition for selecting the parameter r in (4).

Theorem 3.1 (The approximation error bound of SqMethod). *Let (ct, p, B) be a valid ciphertext of $x \in \mathbb{Z}_p$. Then, there exists $r \in \mathbb{Z}_+$, $u \in \mathbb{R}_+$ satisfying $\|\mathsf{Dec} \circ \mathsf{SqMethod}^a_{r,p}(\mathsf{ct}) - \delta_a(x)\| < 2^{-u}$ if $\max(\mathsf{SqMethod}^0_{r,p}(1-B), 1 - \mathsf{SqMethod}^0_{r,p}(B)) \leq 1 - 4 \cdot 2^r B_*$, where B_* is the noise bound occurred by Mult and Rescale.*

Proof. The proof is deferred to Appendix A.1. □

When we induce the approximation error bound in Theorem 3.1, for the given r the following inequalities are provided:

$$\mathsf{SqMethod}^0_{r,p}(1-B) + 4 \cdot 2^r B_* < 2^{-u} \quad (5)$$
$$1 - \mathsf{SqMethod}^0_{r,p}(B) + 4 \cdot 2^r B_* < 2^{-u} \quad (6)$$

Based on these inequalities, we can determine the parameters (r, u) that satisfy the approximation error bounds (5) and (6), minimizing r to reduce multiplication costs and maximizing u to minimize the approximation error with directly enhancing accuracy. Algorithm 1 searches for the optimal parameters (r, u) while balancing between the approximation error and the computational efficiency of SqMethod.

3.2. Cleansing Function

While SqMethod enables to approximate an indicator function δ_a with the approximation error bounded by 2^{-u}, Cleanse aims to efficiently round the output of SqMethod to 0 or 1. Cleanse can be given by $\mathsf{Cleanse}(x) = -2x^3 + 3x^2$ for $x \in V(u) := (-2^{-u}, 2^{-u}) \cup (1 - 2^{-u}, 1 + 2^{-u})$.

Algorithm 1 Parameter Selection of r, u for SqMethod

```
Parameter $p \in \mathbb{Z}_+$
Input $B$ is the noise bound of the input, $B_*$ is the noise parameter in Rescale ∘ Mult
Output $(r, u)$
Procedure paramSq$(p, B, B_*)$ :
 1: $r' = \mathrm{argmin}_r(\mathsf{SqMethod}^0_{r,p}(1-B) + 4 \cdot 2^r B_*)$
 2: $r' \leftarrow \lceil r' \rceil$
 3: $u_{max} \leftarrow 0$
 4: while $r' \geq 1$ do
 5:     $u_0 = -\log(\mathsf{SqMethod}^0_{r',p}(1-B) + 4 \cdot 2^{r'} B_*)$
 6:     $u_1 = -\log(1 - \mathsf{SqMethod}^0_{r',p}(B) + 4 \cdot 2^{r'} B_*)$
 7:     $u' = \min(u_0, u_1)$
 8:     if $u' \geq u_{max}$ then
 9:         $u_{max} \leftarrow u'$
10:         $r' \leftarrow r' - 1$
11:     else
12:         break
13:     end if
14: end while
15: Return $(r' + 1, u_{max})$
```

Note that $-2x^3 + 3x^2$ is a low-degree polynomial such that it passes through $(0, 0)$ and $(1, 1)$, and fast convergence is guaranteed by its derivative as shown in (Cheon et al., 2020). We provide a theoretical analysis for the rounding property and error reduction of Cleanse.

Theorem 3.2 (The Rounding Property of the Cleanse Function). *Let $(\mathsf{ct}, 1, B)$ be a valid ciphertext of a message $x \in V(n)$. For $n > 2$, the evaluation of Cleanse (ct) can reduce the noise by mapping its message to $V(2n-2)$ with the relative noise $15\beta + 10B_*$, where B_* is the noise occurred by Mult and Rescale.*

Proof. The proof is deferred on Appendix A.2. □

Theorem 3.2 demonstrates the error reduction capability of Cleanse when applied to a valid ciphertext of a message $x \in V(n)$ for $n > 2$. Notably, after a single round of Cleanse, the difference bound $|b - x|$ (where b is the rounded value of x) is tightened from 2^{-n} to 2^{-2n+2}. This suggests that repeated application of Cleanse, followed by the ciphertext bounded by 2^{-u} via the SqMethod $(u > 2)$, can effectively reduce both approximation and evaluation noises, ultimately converging the message value x towards its rounded counterpart.

3.3. Encrypted Indicator Function by Square Method

We can construct EIF by composing SqMethod and Cleanse for the ciphertext ct of $x \in \mathbb{Z}_p$ as the approxi-

Algorithm 2 Parameter Selection of r, u, s for $\mathsf{Indicator}$

```
Parameter $p \in \mathbb{Z}_+$
Input $B$ is the noise bound of the input, $B_*$ is the noise bound parameter in $\mathsf{Rescale} \circ \mathsf{Mult}$
Output $(r, u, s)$
Procedure $\mathsf{paramInd}(p, B, B_*)$:
  1: $(r, u) \leftarrow \mathsf{paramSq}(p, B, B_*)$
  2: $s \leftarrow \left\lceil \log\left(\frac{2-\log(10B_*)}{u-2}\right)\right\rceil$
  3: Return $(r, u, s)$
```

mation of $\delta_a(x)$:

$$\mathsf{Indicator}_a(\mathsf{ct}) = \mathsf{Cleanse}^{(s)} \circ \mathsf{SqMethod}^a_{r,p}(\mathsf{ct}) \quad (7)$$

where s is the number of compositions of $\mathsf{Cleanse}$. As a result, we can build EIF with a multiplication depth of $2 + r + 2s$.

Even if $\mathsf{Cleanse}$ can reduce the approximation error $|\mathsf{Dec} \circ \mathsf{Cleanse}(\mathsf{ct}) - b|$ from 2^{-n} to 2^{-2n+2}, as in Theorem 3.2, when considering the noise bound $15\beta + 10B_*$, the error reduction has its limit. Since we cannot precisely track the real noise, we set the limit of error reduction to $10B_*$. Then, we determine the parameter s that meets this limit.

Proposition 3.3. *Let the limit of error reduction in $\mathsf{Cleanse}$ be $10B_*$ and the error bound of $\mathsf{SqMethod}$ be 2^{-u} where $u > 2$, the parameter s of $\mathsf{Indicator}_a(\mathsf{ct}) = \mathsf{Cleanse}^{(s)} \circ \mathsf{SqMethod}^a_{r,p}(\mathsf{ct})$ satisfies $s \leq \log\left(\frac{2-\log(10B_*)}{u-2}\right)$.*

Proof. The proof is deferred on Appendix A.3. □

As a result, we can select the parameters r, u, s for $\mathsf{Indicator}$ evaluation that ensure bounded noise. Further discussions about parameter selection will be provided in Appendix D.1.

4. Look-Up Table Evaluation with Homomorphic Encryption (HELUT)

This section proposes a LUT evaluation with encrypted data, utilizing EIF as a building block. We first propose the LUT evaluation using the EIF for one-hot encoding (OHE) and OHE basis with coded input and then develop the efficient coded LUT evaluation method for larger tables.

4.1. LUT evaluations with One-Hot Encoding

Let $T : \mathbb{Z}_p \to \mathbb{R}^d$ be a LUT. Then, we can represent the encrypted version of the LUT evaluation for ct that is the encrypted input (index) in $\mathbb{Z}_p$ as follows:

$$\begin{aligned} T(\mathsf{ct}) &= \Sigma_{i \in \mathbb{Z}_p} \mathsf{Indicator}_i(\mathsf{ct}) \cdot T(i) && (8) \\ &= \boldsymbol{E} \cdot \mathsf{OHE}_p(\mathsf{ct}), && (9) \end{aligned}$$

where $\mathsf{Indicator}_i(\mathsf{ct})$ is the EIF evaluation for ct, $\boldsymbol{E} \in \mathbb{R}^{p \times d}$ is an embedding matrix with its i-th column containing the result of $T(i)$, and $\mathsf{OHE}_p(i) \in \{0,1\}^p$ is the p dimensional one-hot encoding of i. While (Badawi et al., 2020) employed (9) from $\mathsf{Enc}(\mathsf{OHE}_p(x))$[3], our approach introduces an efficient algorithm that employs coded input indices to leverage the smaller OHE.

4.2. HELUT with Coded Input

We propose HELUT with coded input to improve the efficiency of HELUT evaluation for large tables. Consider the input message $x \in \mathbb{Z}_k$ rather than $\mathbb{Z}_p$ where $k = p^l$, in order to align with the $\mathsf{Indicator}$ applied to $\mathbb{Z}_p$. For the input message $x \in \mathbb{Z}_k$, we first decompose it into a coded input $\mathsf{Decomp}(x) = (x_0, \ldots, x_{l-1}) \in \mathbb{Z}_{p^l}$ such that $x = \sum_{i=0}^{l-1} x_i \cdot p^i$. Consequently, we assume that clients transmit the encryption of coded input $\{\mathsf{Enc}(x_i)\}_{i=0}^{l-1}$. For instance, consider the parameters $p = 8$ and $l = 4$. In this case, the conventional approach $\mathsf{Enc}(\mathsf{OHE}_k(x))$ would consume the slot size of $k = 2^{12}$ for a single token index. Our method, however, achieves a substantial reduction by requiring a slot size of $l = 4$.

To obtain (9) with coded input $\{x_i\}_{i=0}^{l-1}$ of x, we compute $\mathsf{OHE}_p(\mathsf{Enc}(x_i))$ as follows:

$$\begin{aligned} \boldsymbol{w}_i &= \mathsf{OHE}_p(\mathsf{Enc}(x_i)) = (w_{i,0}, \ldots, w_{i,p-1}) && (10) \\ &= (\mathsf{Indicator}_0(\mathsf{Enc}(x_i)), \ldots, \mathsf{Indicator}_{p-1}(\mathsf{Enc}(x_i))). \end{aligned}$$

To improve the efficiency of (9), we devised a memory-efficient approach in Algorithm 3.This algorithm operates on a coded input $\{x_i\}_{i=0}^{l-1}$ of the input message x and yields $\hat{T}(\{x_i\}_{i=0}^{l-1})$ that is the ciphertext of $T(x)$. Algorithm 4 includes a simple technique that dynamically constructs the OHE representation on the fly, leveraging the $\mathsf{OHEbasis} = \{\boldsymbol{w}_{i,j}\}$ to effectively mitigate memory overhead.

In Algorithm 3, we demonstrate the construction of LUT using $p \cdot l$ $\mathsf{Indicator}$ operations, a significant reduction compared to the p^l operations for $\mathsf{OHE}_k(\mathsf{Enc}(x))$. The detailed cost analysis of 9 and Algorithm 3 is included in Appendix D.2. Even if Algorithm 3 has computational gain for large k, it still necessitates $O(k)$ Mult and Add with the noise accumulation. Thus, to address this issue for a very large table such as word embedding, we propose codedHELUT by leveraging the coded compression of the given table (embedding matrix $\boldsymbol{E}$) (Shu & Nakayama, 2018). Even if Algorithm 3 has still smaller computational cost than (9), it still necessitates $O(k)$ multiplications and $O(k)$ additions for table lookup. Also, even with minimal noise growth from individual additions, the collective effect of k addi-

[3]For large p, clients can not encrypt $\mathsf{OHE}_p(x)$ into a single ciphertext. It can exacerbate the ciphertext expansion problem, discussed in Appendix C.4.

Algorithm 3 HELUT with Coded Input(HELUT-CI)

```
Parameter $k = p^l$
Input $\boldsymbol{E} \in \mathbb{R}^{k\times d}$ be an embedding matrix of LUT $T$, $\{\mathsf{Enc}(x_i)\}_{i=0}^{l-1}$ be CKKS ciphertexts of $\mathsf{Decomp}(x) = (x_0, \ldots, x_{l-1}) \in \mathbb{Z}_p^l$ for $x \in \mathbb{Z}_k$
Output $\mathsf{ct}_0, \cdots, \mathsf{ct}_{d-1}$; CKKS ciphertexts of $T(x)$
Procedure $\tilde{T}(\{\mathsf{Enc}(x_i)\}_{i=0}^{l-1})$:
 1: $\boldsymbol{w}_i \leftarrow \mathsf{OHE}_p(\mathsf{Enc}(x_i))$ for $i = 0, \cdots, l-1$
 2: Initialize $\mathsf{ct}_j \leftarrow \mathsf{Enc}(0)$ for $j = 0, \cdots, d-1$
 3: for $0 \le y < k$ do
 4:     $(y_0, \ldots, y_{l-1}) \leftarrow \mathsf{Decomp}(y)$
 5:     for $0 \le i < l$ do
 6:         $w \leftarrow \Pi w_{i,y_i}$
 7:     end for
 8:     for $0 \le j < d$ do
 9:         $\mathsf{ct}_j \leftarrow \mathsf{Add}(\mathsf{ct}_j, \mathsf{Mult}(w, E_{yj}))$
            //* $E_{yj}$ is the $(y, j)$ element of $\boldsymbol{E}$ *//
10:     end for
11: end for
12: Return $\mathsf{ct}_0, \ldots, \mathsf{ct}_{d-1}$
```

tions can amplify noise. If one-hot encoding ciphertext has a noise bound of B (using EIF), then each multiplication results in the noise $B\|\boldsymbol{E}\|_{\max}$ (Cheon et al., 2017). Due to subsequent addition, the LUT's noise bound could reach $kB\|\boldsymbol{E}\|_{\max}$, potentially compromising accuracy. in the following subsection by introducing the coded compression. in this noise to B multiplied by the size of the maximum element in the look-up table, $\|\boldsymbol{E}\|_{\max}$. If one-hot encoding entails noise bounded by B, the noise bound of LUT is up to $k \cdot B$, which could entirely break the result. Note this additional cost and accuracy is shared property with the method in (Badawi et al., 2020), in which the embedding is derived from the multiplication of encrypted one-hot encoding and table matrix. We solve this problem in the following subsection.

4.3. Coded Look-Up Table Evaluation

To address the challenges posed by the large LUTs inherent to embedding layers, such as NLP applications with massive vocabularies (i.e., $100K$), we propose a codedHELUT algorithm. This technique effectively compresses the look-up table T (as detailed in Section 2.2) and leverages Decomp representations of toke indices, substantially reducing memory footprint and computational costs, thereby enabling efficient LUT evaluation even for extensive tokens.

The coded LUT evaluation method is detailed in Algorithm 4. Its main difference from Algorithm 3 lies in the use of compressed embedding with the code embedding LUTs (codebooks) $T_i(x_i)$ that collectively construct the approximate embedding as follows $\tilde{T}(x) = \sum_{i=0}^{l-1} T_i(x_i) \approx T(x)$.

Algorithm 4 CodedHELUT Evaluation

```
Parameter $k = p^l$
Input $\{\boldsymbol{E}^{(i)}\}_{i=0}^{l-1}$ be the embedding matrices of codebooks $\{T_i\}_{i=0}^{l-1}$ for $T$, $\{\mathsf{Enc}(x_i)\}_{i=0}^{l-1}$ be CKKS ciphertexts of $\mathsf{Decomp}(x) = (x_0, \ldots, x_{l-1}) \in \mathbb{Z}_p^l$ for $x \in \mathbb{Z}_k$
Output $\mathsf{ct}_0, \ldots, \mathsf{ct}_{d-1}$; CKKS ciphertext of $\tilde{T}(x)$
Procedure $\tilde{T}(\{\mathsf{Enc}(x_i)\}_{i=0}^{l-1})$:
 1: for $0 \le i < l, 0 \le y < p$ do
 2:     $\boldsymbol{w}_{i,y} \leftarrow \mathsf{Indicator}_y(\mathsf{Enc}(x_i))$
 3: end for
 4: for $0 \le j < d$ do
 5:     Initialize $\mathsf{ct}_j \leftarrow \mathsf{Enc}(0)$
 6:     for $0 \le i < l, 0 \le y < p$ do
 7:         $\mathsf{ct}_j \leftarrow \mathsf{Add}(\mathsf{ct}_j, \mathsf{Mult}(w_{i,y}, E_{yj}^{(i)}))$
            //* $E_{yj}^{(i)}$ is the $(y, j)$ element of $\boldsymbol{E}^{(i)}$ *//
 8:     end for
 9: end for
10: Return $\mathsf{ct}_0, \ldots, \mathsf{ct}_{d-1}$
```

We assume that our algorithm utilizes the compressed embedding (i.e., code and token embedding) learned using the method proposed in (Shu & Nakayama, 2018). Then, we can represent $\tilde{T}(\{x_i\}_{i=0}^{l-1})$ as follows:

$$\tilde{T}(\{x_i\}_{i=0}^{l-1}) = \sum_{i=0}^{l-1} \boldsymbol{E}^{(i)} \mathsf{OHE}_p(x_i) \approx T(x), \tag{11}$$

where $\boldsymbol{E}^{(i)}$ is the embedding matrix of codebook T_i. As a result, Algorithm 4 enables the construction of $\tilde{T}(x)$ from the OHE bases of coded inputs with EIF on $\mathbb{Z}_p$ and codebooks $\{T_i\}_{i=0}^{l-1}$. In terms of the effectiveness of the compressed embedding, Shu & Nakayama (2018) showed that the compressed embedding achieved performance on par with the original embedding. Our empirical experiments also validate this finding, showing that CodedHELUT maintains utility even with the compressed embeddings and approximate computations, as detailed in Table 2. Note that the following accuracy analysis does not consider the effect of embedding compression.

Cost and Accuracy Analysis The coded LUT evaluation in Algorithm 4 yields significant performance and noise reduction benefits. In particular, the multiplication cost is reduced from $O(k) = O(p^l)$ to $O(p \cdot l)$. In addition, the addition noise is reduced from $p^l \cdot B\|\boldsymbol{E}\|_{\max}$ to $pB\sum_{i=0}^{l-1}\|\boldsymbol{E}^{(i)}\|_{\max}$[4]. For instance, when we consider the task of evaluating GloVe (Global Vectors for Word Representation) 6B50d ($k = 400,000$), HELUT evaluations of (9) (Badawi et al., 2020) and Algorithm 3 require k Mult with the worst-case precision loss is $\log 400K = 19$ bits. In contrast, codedHELUT (Algorithm 4) with $p = 64$ and

[4]For $\boldsymbol{E}$, the max norm is defined as $\|\boldsymbol{E}\|_{\max} := \max_{i,j} |E_{ij}|$.

$l = 8$ requires only 512 Mult, resulting in the maximum precision loss of 9 bits. This highlights the advantages of our technique in terms of both computational efficiency and noise resilience.

The illustrative example of this operation is provided in Figure 2. A detailed cost comparison with other methods is provided in Appendix D.2.

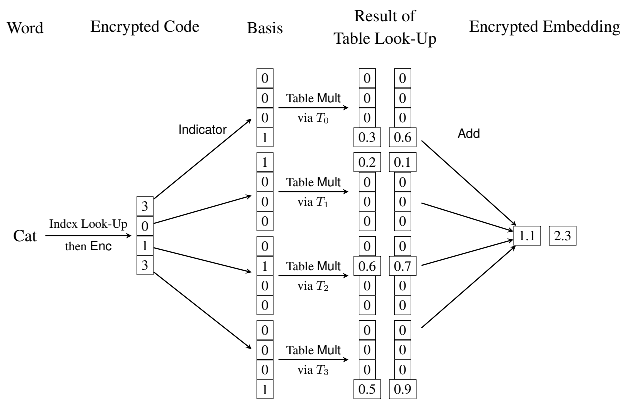

Figure 2. Illustrative Example of Word Embedding with 4 codebooks of dimension 4, to 2-dimensional embedding by 16-slotted ciphertext in **p1**

Parallelization through SIMD To accelerate HELUT, we can leverage the power of SIMD instructions provided by the CKKS scheme with a large slot size (e.g., $n = 2^{16}$). Since SIMD is based on the parallelization of operations, we consider three key opportunities for parallelization (**p1, p2**) within our HELUT evaluation:

- **Codes of Coded Input (p1):** By packing codes of coded inputs into a single ciphertext, we can drastically reduce the cost of OHEbasis with Indicator operation to 1 with pl slots. (Line 1 of Algorithm 4)
- **multiple input indices (p2):** For client requests involving multiple input indices, we can batch $n/(pl)$ input indices in a ciphertext and reduce the overall computational cost.

The illustrative example of ciphertext packing is provided in Figure 2. A detailed cost comparison with other methods is provided in Appendix D.2.

Scalability The scalability of the CodedHELUT evaluation relies on the coded compression parameters (p, l) rather than the original embedding's vocabulary size (k). Since scalability issues in embedding layers primarily stem from the number of tokens, the CodedHELUT method can efficiently manage a large number of tokens. Moreover, while the CodedHELUT evaluation is linearly affected by the output dimension of the embedding vector, each part can be evaluated independently. We will demonstrate the scalability of CodedHELUT with experimental results in Section 5.2.

Scenario Specification Suppose that a server's ML model begins with an embedding layer. Instead of using the trained embedding layer directly, the server locally learns a compressed model specifically for this layer (Shu & Nakayama, 2018). Additionally, the server provides an API that enables mapping tokens to the coded input indices for the obtained compressed model. Clients can then generate a sequence of coded inputs corresponding to their sequence of tokens. To maintain the privacy of the client's data, clients send ciphertexts of these coded inputs. Subsequently, the server can compute the results of the embedding layer using our coded HELUT algorithm and the subsequent ML model using this encrypted data. This process is depicted in Figure 3.

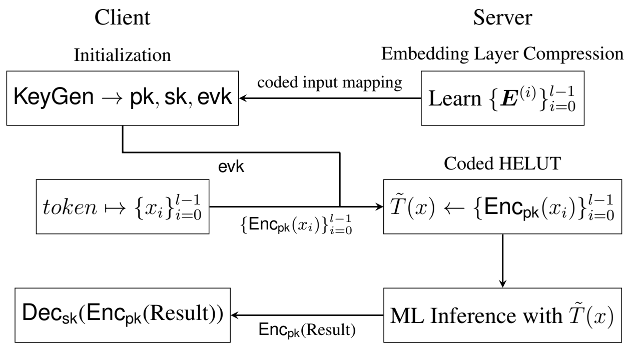
Client
Server
Initialization
Embedding Layer Compression
KeyGen → pk, sk, evk
coded input mapping
Learn $\{\boldsymbol{E}^{(i)}\}_{i=0}^{l-1}$
evk
Coded HELUT
$token \mapsto \{x_i\}_{i=0}^{l-1}$
$\{\mathsf{Enc}_{\mathsf{pk}}(x_i)\}_{i=0}^{l-1}$
$\tilde{T}(x) \leftarrow \{\mathsf{Enc}_{\mathsf{pk}}(x_i)\}_{i=0}^{l-1}$
$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(\text{Result}))$
$\mathsf{Enc}_{\mathsf{pk}}(\text{Result})$
ML Inference with $\tilde{T}(x)$

Figure 3. Scenario Specification of PPML with CodedHELUT

Limitations due to compressed embeddings Our CodedHELUT method significantly improves efficiency by leveraging coded input. In particular, to integrate our algorithm with conventional embedding layers, we utilized compressed embeddings based on the method proposed in (Shu & Nakayama, 2018) and demonstrated its effectiveness through experiments in Section 5.2. However, the reliance on the specific method (Shu & Nakayama, 2018) for compressed embeddings can be a limitation of our study. Thus, we discuss this limitation in detail and present future research directions concerning it in Section D.4.

5. Experiments

In this section, we demonstrate the efficiency of HELUT by using the synthetic table and large-scale real-world word embeddings.

Experimental Setting We implemented HELUT with linear transformation (HELUT-LT) method in (9), HELUT with coded inputs (HELUT-CI) in Algorihtm 3, and codedHELUT in Algorithm 4 with SIMD parallelization **p1**. We also applied SIMD **p2** for all HELUT methods for multiple inputs (token indices). For comparison, we utilized synthetically generated table ($T : \mathbb{Z}_{64} \rightarrow \mathbb{R}^{16}$) and the real-world large-scale NLP embedding GloVe 6B50d, GloVe 42B300d (Pennington et al., 2014a;b), BERT(Bidirectional Encoder

Representations from Transformers) (Pires et al., 2019) and GPT-2(Generative pre-trained transformer) (Radford et al., 2019). For encrypted embedding layers, we only used codedHELUT as other methods are not practically viable. To assess the efficiency, we repeated for 8 independent trials and measured the average and standard deviation of running times, where amortized running time corresponds to the total running time divided by the number of inputs.

CKKS Parameters We used the quotient polynomial $\mathbb{Z}_q[X]/(X^N+1)$ with $\log N = 17$ and the scaling factor size $\log \Delta = 35$. Thus, the slot size for SIMD becomes $n = 2^{16}$, and the hamming weight of the secret key is 128. The size of q varies by experiment and is described in each table. These parameters satisfy 128-bit security according to (Jung Hee et al., 2022). In Appendix E.3, we also provide the parameters (r, u, s) for $\mathsf{Indicator}$.

Implementation Details Our experiments were conducted on the server with Intel Xeon 6426Y at 2.5GHz. Our HE implementation was based on OpenFHE (Al Badawi et al., 2022). For embedding compression, we used the public code of (Shu & Nakayama, 2018; Kim, 2018) with Pytorch 1.10.0 (Python 3) for the pre-trained embeddings.

5.1. HELUT on Synthetic Table

To compare the efficiency of HELUT, we measured the amortized running time for $\mathsf{Indicator}$, Linear Transformation (LinearTrans), and the entire HELUT process (Total). Table 1 shows the results of evaluating LUT for 2^{12} input indices with four HELUT implementations: HELUT-LT, HELUT-CI, CodedHELUT, and CodedHELUT with parallelization **p1**. For all HELUT methods, we included the SIMD parallelization **p2** for 2^{12} inputs.

HELUT-LT, which does not use coded input, requires parameters $p = 64$ where $k = p = 64$ represents the number of rows in the synthetic table. This configuration results in high evaluation time due to $\mathsf{Indicator}_{64}$ operations. HELUT-CI, on the other hand, achieves a faster evaluation time by reducing the number of $\mathsf{Indicator}$ operations to $pl = 16$ in addition to smaller $p = 16$, where the reduction in p leads to fewer $\mathsf{SqMethod}$ and $\mathsf{Cleanse}$ rounds with decreased cost, as detailed in Appendix E.3. This reduction overweighs the increased LinearTrans running time because of the construction of w in Algorithm 3. Comparing them, CodedHELUT exhibits a similar $\mathsf{Indicator}$ evaluation time as HELUT-CI but benefits from a shorter LinearTrans running time. In addition, CodedHELUT can achieve a higher precision, surpassing HELUT-LT and HELUT-CI than HELUT-LT and HELUT-CI. Through the SIMD parallelization with **p1**, CodedHELUT significantly reduces the evaluation time and facilitates HELUT evaluations using one packed ciphertext.

5.2. HELUT on Word Embedding

We applied CodedHELUT with parallelization **p1** and **p2** to the word embedding layers of GloVe 6B50d (400K vocabularies, $d = 50$), 42B300d (1.9M vocabularies, $d = 300$), and GPT-2 (50K vocabularies, $d = 768$) models using the compressed coded embedding obtained according to (Shu & Nakayama, 2018), as detailed in Appendix E.1. The CodedHELUT evaluation of BERT and GPT-2 for the same (p, l) has the same computational cost because they have the same output dimension $d = 768$. Table 2 shows the amortized running times of HELUT evaluation, as well as the compressed embedding performance in terms of MSE (mean squared error between original and compressed embeddings). We can pack n/pl word indices per single ciphertext (words/ct). Additionally, we conducted an experiment to demonstrate the effectiveness of the compressed embedding for the downstream sentiment analysis task on the IMDB dataset (Maas et al., 2011). The results of this experiment are summarized in Appendix E.2.

Our results demonstrate that smaller pl leads to lower HELUT evaluation costs because of the increased word packing and the reduced Rot and Add costs in LinearTrans. However, this compression comes at the expense of increased MSE and reduced sentiment classification accuracy. In addition, for $pl = 512$, CodedHELUT with $p = 8, l = 64$ offers better performance than $p = 16, l = 32$ in terms of HELUT evaluation cost and MSE with similar accuracy since smaller p results in a more efficient $\mathsf{Indicator}_p$, reducing the bit consumption in FHE operations (535 for $p = 8$ and 675 for $p = 16$). Moreover, it is noteworthy that the worst precision was 16 bits on all settings, which was the maximum precision we designed for our algorithm. Consequently, performance metrics like MSE and accuracy remain nearly identical to their non-encrypted counterparts.

Note on Vocabulary size The computation cost of CodedHELUT directly depends on the size of pl and d, not the size of the original vocabulary. For relatively small vocabularies such as BERT and GPT-2, the compressed embedding achieves lower MSE even with a small pl. This allows for a higher compression rate by using even smaller pl values, leading to a more efficient and lightweight CodedHELUT evaluation.

6. Conclusion

In this study, we introduce a novel efficient LUT evaluation algorithm with high precision based on the CKKS scheme, utilizing the proposed EIF with bounded noise. We provide theoretical analysis for the precision guarantee of our EIF construction and empirically validate the practicality of the CodedHELUT algorithm in large-scale word embedding evaluation for GloVe, BERT, and GPT-2, without the need

Table 1. Average amortized running time with standard deviation (ms), and precision (bits) of various LUT evaluation methods for 2^{12} inputs. $\log q = 850$ for each experiment.

Method	p	l	Indicator	LinearTrans	Total	Precision
HELUT-LT	64	1	71.05 ± 1.32	4.10 ± 0.09	81.16 ± 1.3	14
HELUT-CI	8	2	13.53 ± 0.11	9.44 ± 0.32	22.98 ± 0.35	14
CodedHELUT	8	2	13.80 ± 0.33	1.58 ± 0.12	15.39 ± 0.37	14.25
CodedHELUT+**p1**	8	2	0.76 ± 0.03	3.06 ± 0.21	3.83 ± 0.22	15

Table 2. Average amortized running time with standard deviation (ms) of CodedHELUT(with **p1** parallelization) for Compressed Word Embedding with GloVe 6B50d, GloVe 42B300d, GPT-2, BERT and MSE loss of their compression. $\log q = 535$ for $\log p = 8$, $\log q = 675$ for $\log p = 16$

Setting		HELUT evaluation			MSE loss			
p	l	6B50d	42B300d	BERT/GPT-2	6B50d	42B300d	BERT	GPT-2
8	8	0.0182 ± 0.00063	0.10423 ± 0.00838	0.2621 ± 0.02266	0.180	0.069	0.001	0.013
8	16	0.04028 ± 0.0007	0.23359 ± 0.01905	0.59094 ± 0.05402	0.103	0.065	0.001	0.012
8	32	0.09192 ± 0.00584	0.51627 ± 0.04144	1.31364 ± 0.1155	0.063	0.055	0.001	0.012
8	64	0.20412 ± 0.0108	1.14468 ± 0.09586	2.88359 ± 0.24121	0.030	0.041	0.001	0.010
16	32	0.24174 ± 0.01235	1.2981 ± 0.0884	3.28315 ± 0.23777	0.050	0.052	0.001	0.010

for bootstrapping. Furthermore, our framework enables the end-to-end HE evaluation of the embedding layer with small input ciphertexts by clients in PPML scenarios. Although we demonstrate the effectiveness of our algorithm through word embedding experiments, our algorithm is applicable to any embedding layer that requires a look-up evaluation (multiplication with one-hot representation) for a large embedding matrix, such as a recommendation system (i.e., user/item embedding) and graph neural networks (i.e., node/edge embeddings) (Gao et al., 2023; Barkan & Koenigstein, 2016). Consequently, our study has a significant potential to advance the development of various PPML applications with embedding layers, particularly by combining more complex deep learning models.

Acknowledgements

We first appreciate reviewers who gave useful comments. This work was supported by the National Research Foundation of Korea (NRF) and Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT)(No. NRF-2022R1F1A1065171, No. RS-2020-II201336, Artificial Intelligence Graduate School Program(UNIST), No. RS-2024-00398360, Development of a user-friendly and efficiency-optimized real-time homomorphic statistical analysis processing platform). This work was also supported by Korea Institute of Information Security & Cryptology (KIISC).

Impact Statement

CodedHELUT presents significant improvement for privacy-preserving NLP tasks utilizing HE in terms of privacy and efficiency. Our method enables secure and efficient processing of NLP workloads while protecting sensitive data, and addressing key privacy and efficiency issues. Traditional NLP with CKKS faces privacy challenges during text-to-number conversion. Prior solutions expose the server's pre-trained model, posing privacy risks. CodedHELUT ensures end-to-end HE evaluation, safeguarding both the server's model and the client's input data. In addition, HE in PPML struggles with ciphertext expansion, especially in models with large embedding layers. Our method compresses embeddings within the encrypted domain, facilitating efficient processing and making HE more practical for PPML. Furthermore, our solution supports efficient look-ups for embedding layers across various machine-learning domains. As a result, our work enhances the privacy and efficiency of NLP workflows in PPML, contributing to the broader adoption and practicality of homomorphic encryption in machine learning.

References

Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D. B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., and Zucca, V. Openfhe: Open-source fully homomorphic encryption library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, pp. 53–63, New York, NY, USA, 2022. Association for Computing Machinery. doi: 10.1145/3560827.3563379. URL https://doi.org/10.1145/3560827.3563379.

Badawi, A. A., Hoang, L., Mun, C. F., Laine, K., and Aung, K. M. M. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access*, 8:226544–226556, 2020. doi: 10.1109/ACCESS.2020.3045465.

Barkan, O. and Koenigstein, N. Item2vec: neural item embedding for collaborative filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2016.

Boemer, F., Lao, Y., Cammarota, R., and Wierzynski, C. Ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF '19, pp. 3–13, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366854. doi: 10.1145/3310273.3323047. URL https://doi.org/10.1145/3310273.3323047.

Brakerski, Z. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pp. 868–886. Springer, 2012.

Brakerski, Z., Gentry, C., and Vaikuntanathan, V. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. https://eprint.iacr.org/2011/277.

Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Brüggemann, A., Hundt, R., Schneider, T., Suresh, A., and Yalame, H. Flute: Fast and secure lookup table evaluations (full version). Cryptology ePrint Archive, Paper 2023/499, 2023. URL https://eprint.iacr.org/2023/499. https://eprint.iacr.org/2023/499.

Byun, J., Park, S., Choi, Y., and Lee, J. Efficient homomorphic encryption framework for privacy-preserving regression. *Applied Intelligence*, 53(9):10114–10129, 2023.

Chen, M., Hao, Y., Hwang, K., Wang, L., and Wang, L. Disease prediction by machine learning over big data from healthcare communities. *IEEE Accees*, 5:8869–8879, 2017.

Chen, T., Bao, H., Huang, S., Dong, L., Jiao, B., Jiang, D., Zhou, H., Li, J., and Wei, F. The-x: Privacy-preserving transformer inference with homomorphic encryption, 2022.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 409–437. Springer, 2017.

Cheon, J. H., Han, K., Kim, A., Kim, M., and Song, Y. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 360–384. Springer, 2018a.

Cheon, J. H., Han, K., Kim, A., Kim, M., and Song, Y. A full rns variant of approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2018/931, 2018b. URL https://eprint.iacr.org/2018/931. https://eprint.iacr.org/2018/931.

Cheon, J. H., Kim, D., and Kim, D. Efficient homomorphic comparison methods with optimal complexity. In Moriai, S. and Wang, H. (eds.), *Advances in Cryptology – ASIACRYPT 2020*, pp. 221–256, Cham, 2020. Springer International Publishing. ISBN 978-3-030-64834-3.

Chillotti, I., Gama, N., Georgieva, M., and Izabachene, M. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security*, pp. 3–33. Springer, 2016.

Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

Fan, J. and Vercauteren, F. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. URL https://eprint.iacr.org/2012/144.

Gao, C., Zheng, Y., Li, N., Li, Y., Qin, Y., Piao, J., Quan, Y., Chang, J., Jin, D., He, X., and Li, Y. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Trans. Recomm. Syst.*, 1

(1), mar 2023. doi: 10.1145/3568022. URL https://doi.org/10.1145/3568022.

Gentry, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

Guimarães, A., Borin, E., and Aranha, D. F. Revisiting the functional bootstrap in tfhe. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):229–253, Feb. 2021. doi: 10.46586/tches.v2021.i2.229-253. URL https://tches.iacr.org/index.php/TCHES/article/view/8793.

Jung Hee, C., Yongha, S., and Donggeon, Y. Practical fhe parameters against lattice attacks. *Journal of the Korean Mathematical Society*, 59(1):35–51, 2022. doi: 10.4134/JKMS.j200650.

Kim, A., Song, Y., Kim, M., Lee, K., and Cheon, J. H. Logistic regression model training based on the approximate homomorphic encryption. *BMC Med Genomics*, 11(83), 2018.

Kim, M. Compressing word embeddings via deep compositional code learning (pytorch implementation). https://github.com/mingu600/compositional_code_learning, 2018.

Kim, M., Song, Y., Li, B., and Micciancio, D. Semi-parallel logistic regression for gwas on encrypted data. *BMC Medical Genomics*, 13(7):99, 2020. doi: 10.1186/s12920-020-0724-z. URL https://doi.org/10.1186/s12920-020-0724-z.

Launchbury, J., Diatchki, I. S., DuBuisson, T., and Adams-Moran, A. Efficient lookup-table protocol in secure multiparty computation. In *Proceedings of the 17th ACM SIGPLAN international conference on Functional programming*, pp. 189–200, 2012.

Lee, G., Kim, M., Park, J. H., Hwang, S.-w., and Cheon, J. H. Privacy-preserving text classification on BERT embeddings with homomorphic encryption. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V. (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3169–3175, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.231. URL https://aclanthology.org/2022.naacl-main.231.

Lee, Y., Seo, J., Name, Y., Chae, J., and Cheon, J. H. Heaan-stat: a privacy-preserving statistical analysis toolkit for large-scale numerical, ordinal, and categorical data. *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2023. doi: 10.1109/TDSC.2023.3275649.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data, 2023.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Park, S., Kim, J., Lee, J., Byun, J., Cheon, J. H., and Lee, J. Security-preserving support vector machine with fully homomorphic encryption. *SafeAI@ AAAI*, 2301, 2019.

Park, S., Byun, J., Lee, J., Cheon, J. H., and Lee, J. He-friendly algorithm for privacy-preserving svm training. *IEEE Access*, 8:57414–57425, 2020.

Park, S., Byun, J., and Lee, J. Privacy-preserving fair learning of support vector machine with homomorphic encryption. In *Proceedings of the ACM Web Conference 2022*, pp. 3572–3583, 2022.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014a. URL http://www.aclweb.org/anthology/D14-1162.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/, 2014b. Stanford NLP Group.

Pires, T., Schlinger, E., and Garrette, D. How multilingual is multilingual bert?, 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Shu, R. and Nakayama, H. Compressing word embeddings via deep compositional code learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=BJRZzFlRb.

Tanuwidjaja, H. C., Choi, R., Baek, S., and Kim, K. Privacy-preserving deep learning on machine learning as a service—a comprehensive survey. *IEEE Access*, 8:167425–167447, 2020. doi: 10.1109/ACCESS.2020.3023084.

A. Theoretical Proofs

A.1. Proof of Theorem 3.1

Lemma A.1 (Noise Bound of Power of Two Square (Cheon et al., 2017)). *For power of two integer $deg = 2^i$, deg-squares and rescale of $(c_0, 1, \beta)$ makes ciphertext $(c_i, 1, (deg \cdot \beta + (deg-1)\beta_*))$ where β_* is the relative noise of $B_* = B_{mult} + B_{rescale}$.*

Lemma A.2 (Noise Bound of $\mathsf{SqMethod}$). *For power of two integer 2^r, if $(\mathsf{ct}, p, p\beta)$ is a valid ciphertext of $x \in \mathbb{Z}_p$, evaluation of $\mathsf{SqMethod}(\mathsf{ct})^a_{r,p}$ has noise bound $4 \cdot 2^r\beta + (3 \cdot 2^r - 1)\beta_*$*

Proof. $(\mathsf{ct}/p, 1, \beta)$ is a valid ciphertext, and square yields $((\mathsf{ct}/p)^2, 1, 2\beta + \beta_*)$. Then 2^r-squares of $(1 - 2(\mathsf{ct}/p)^2, 1, 4\beta + 2\beta_*)$ yields noise bound

$$2^r(4\beta + 2\beta_*)) + (2^r - 1)\beta_*$$
$$= 4 \cdot 2^r\beta + (3 \cdot 2^r - 1)\beta_*$$

□

Based on the noise bound in Lemma A.2, we can obtain the following parameter selection condition of r, u.

Theorem 3.1 (The approximation error bound of $\mathsf{SqMethod}$). *Let (ct, p, B) be a valid ciphertext of $x \in \mathbb{Z}_p$. Then, there exists $r \in \mathbb{Z}_+$, $u \in \mathbb{R}_+$ satisfying $\|\mathsf{Dec} \circ \mathsf{SqMethod}^a_{r,p}(\mathsf{ct}) - \delta_a(x)\| < 2^{-u}$ if $\max(\mathsf{SqMethod}^0_{r,p}(1-B), 1 - \mathsf{SqMethod}^0_{r,p}(B)) \leq 1 - 4 \cdot 2^r B_*$, where B_* is the noise bound occurred by Mult and $\mathsf{Rescale}$.*

Proof of Theorem 3.1. Since a in $\mathsf{SqMethod}^a_{r,p}$ works as a permutation in $\mathbb{Z}_p$ (i.e., $(x-a)^2$), without loss of generality, we consider $a = 0$.

Note that the noise and the plaintext for the ciphertext ($\mathsf{pt}+e$) are inseparable. For simplicity, we can re-define the addition of message $x \in \mathbb{Z}_p$ noise $e \leq B$ as message $x' \in \mathcal{N}_B(x) := \{x' : |x - x'| \leq B, x \in \mathbb{Z}_p\}$ and small encoding noise.

For ct' be the ciphertext of x', we have the approximation bound condition of r, u satisfying:

$$\|\mathsf{Dec} \circ \mathsf{SqMethod}^0_{r,p}(\mathsf{ct}') - \delta_0(\lfloor x' \rceil)\| < 2^{-u}$$

Then by Lemma A.2, we get noise bound $4 \cdot 2^r\beta + (3 \cdot 2^r - 1)\beta_*$. We rearranged message x and noise e bounded by B to the converted message $x' = x + e'$, and then the noise is reset to encoding noise, bounded by N/Δ which is a rounding noise of canonical embedding. The size of this new noise bound β' of x' can be expressed as $\beta' = N/\Delta \leq 1/4 \cdot B_{scale}/\Delta \leq 1/4 \cdot \beta_*$, since scaling noise bound $B_{scale}N/3 \cdot (3 + 8\sqrt{h})$ with hamming weight of secret key $h \geq 64$. Then the noise bound of $\mathsf{SqMethod}$ is rearranged to $4 \cdot 2^r\beta_*$. The input of $\mathsf{SqMethod}$ is normalized by initial division by p, so we can assume $\beta_* = B_*$.

Since $\mathsf{SqMethod}$ is the decreasing function, the maximum approximation error is attained by either $x' = B \in \mathcal{N}_B(0)$ or $x' = 1 - B \in \mathcal{N}_B(1)$. Thus, if the approximation error is bounded by 2^{-u}, we can obtain the inequalities:

$$\mathsf{SqMethod}^0_{r,p}(1-B) + 4 \cdot 2^r B_* < 2^{-u}$$

$$\mathsf{SqMethod}^0_{r,p}(B) - 4 \cdot 2^r B_* > 1 - 2^{-u}$$

The second inequality can be rearranged to

$$1 - \mathsf{SqMethod}^0_{r,p}(B) + 4 \cdot 2^r B_* < 2^{-u}$$

If $\max(\mathsf{SqMethod}^0_{r,p}(1-B), 1 - \mathsf{SqMethod}^0_{r,p}(B)) \leq 1 - 4 \cdot 2^r B_*$, there exists $r \in \mathbb{Z}_+$ and $u \in \mathbb{R}_+$ to bound the approximation error of $\mathsf{SqMethod}_{r,p}$. □

A.2. Proof of Theorem 3.2

Lemma A.3 (Noise Estimation of Polynomial Evaluation (Cheon et al., 2017)). *Let $f(x) = \Sigma_{0 \leq i \leq d} a_i x^i$ be polynomial with nonzero coefficient $a_d \in \mathbb{R}$. Let $(\mathsf{ct}, 1, \beta)$ be valid ciphertext, then valid encryption of the evaluation of ct on $f(x)$ becomes $(\mathsf{ct}', M_f, \beta_d M_f)$ for $M_f = \Sigma_{0 \leq i \leq d}|a_i|$ and $\beta_d \leq d\beta + (d-1)\beta_*$ where β_* is the relative noise of $B_* = B_{mult} + B_{rescale}$.*

Theorem 3.2 (The Rounding Property of the $\mathsf{Cleanse}$ Function). *Let $(\mathsf{ct}, 1, B)$ be a valid ciphertext of a message $x \in V(n)$. For $n > 2$, the evaluation of $\mathsf{Cleanse}$ (ct) can reduce the noise by mapping its message to $V(2n-2)$ with the relative noise $15\beta + 10B_*$, where B_* is the noise occurred by Mult and $\mathsf{Rescale}$.*

Proof. For simplicity, we define the tuple of variables (b, x_b) as follows:

$$(b, x_b) = \begin{cases} (0, -x) & \text{if } x \in (-2^{-n}, 2^{-n}) \\ (1, 1-x) & \text{if } x \in (1 - 2^{-n}, 1 + 2^{-n}) \end{cases}$$

where b is the rounded value (either 0 or 1), and $x_b = b - x$ is the difference between b and x. Then, we can assure $|x_b| < 2^{-n}$ as $x \in V(n)$.

In addition, we can obtain

$$|\mathsf{Cleanse}(x) - b| \leq 2|x_b|^3 + 3x_b^2 \quad (12)$$

because $\mathsf{Cleanse}(x) = 2x_0^3 + 3x_0^2$ for $b = 0$, and $\mathsf{Cleanse}(x) = 1 - 3x_1^2 + 2x_1^3$ for $b = 1$. As a result, as $|x_b| < 2^{-n}$ $(n \geq 1)$,

$$|\mathsf{Cleanse}(x) - b| \leq (2^{1-n} + 3)x_b^2. \tag{13}$$

For Cleanse to work as error reduction, $(2^{1-n} + 3)x_b^2 < |x_b|$ must be hold to get strictly decreasing property. This inequality can be represented as $|x_b| < 1/(2^{1-n}+3)$. Thus, when $2^{-n} < 1/(2^{1-n} + 3)$, the error can be reduced. By solving the inequality with respect to $z = 2^{-n}$, we can obtain the following condition:

$$0 < z \leq \frac{-3 + \sqrt{17}}{4}. \tag{14}$$

Because $1/4 < \frac{-3+\sqrt{17}}{4} < 1/2$, $n > 2$ is required. As a result, for the approximation error bound (13) $4x_b^2 < 2^{-2n+2}$ holds.

In addition, the relative noise bound due to Cleanse evaluation becomes $15\beta + 10B_*$ because $d = 3$ and $M_f = 5$ via Lemma A.3, and β_* in the lemma corresponds to B_* since the bound of input is 1.

□

A.3. Proof of Proposition 3.3

Proposition 3.3. *Let the limit of error reduction in Cleanse be $10B_*$ and the error bound of SqMethod be 2^{-u} where $u > 2$, the parameter s of $\mathsf{Indicator}_a(\mathsf{ct}) = \mathsf{Cleanse}^{(s)} \circ \mathsf{SqMethod}^a_{r,p}(\mathsf{ct})$ satisfies $s \leq \log\left(\frac{2-\log(10B_*)}{u-2}\right)$.*

Proof. According to Theorem 3.2, if the message of ciphertext ct is in $V(n)$, then $\min_{b\in\{0,1\}} |\mathsf{Dec} \circ \mathsf{Cleanse}(\mathsf{ct}) - b|$ is less then 2^{-2n+2} with noise bound $15\beta + 10B_*$. Even n could be increased by a round of Cleanse, if it is dominated by the noise, the reduction has no effect, and further application of Cleanse can not ensure further convergence to $0, 1$. Assuming conservative noise bound, the convergence of Cleanse cannot surpass $10B_*$. By this limit, we can calculate the upper bound of the number of Cleanse rounds (s).

Assuming a conservative noise bound, we calculate s by iteratively updating n until it reaches the limit of $10B_*$. The updated formula for n is $n_i \leftarrow 2n_{i-1} - 2$, with an initial value of $n_0 = u$. This leads to $n_i = 2^i u - (2^{i+1} - 2)$. Setting $n_s = 2^s u - (2^{s+1} - 2) \geq -\log(10B_*)$ and solving for s, we obtain $s \geq \log\left(\frac{2-\log(10B_*)}{u-2}\right)$, providing the maximum value of s.

□

Table 3. Notation Summary of Parameters

Symbol	Description
N	degree of ciphertext space
q	ciphertext modulus
Δ	scaling factor
B	noise bound of a ciphertext
v	plaintext bound of a ciphertext
β	relative noise, B/v
p	bound of EIF domain
l	number of codebooks
k	size of a large LUT : p^l

B. Additional Backgrounds

B.1. Notation Summary of Parameters

We provide comprehensive notations for parameters in the CKKS scheme and our algorithms in Table 3.

B.2. Homomorphic Encryption

Homomorphic Encryption is a cryptographic primitive allowing arithmetic operations over encrypted data. In this paper, we use CKKS scheme (Cheon et al., 2017) for HE, which supports approximate addition, and Hadamard multiplication over complex numbers.

For a power-of-two integer M and $N = M/2$, we use the notation $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$, where q is a positive integer.

The CKKS scheme utilizes an encoding scheme $\mathsf{Ecd} : \mathbb{C}^{N/2} \to \mathcal{R}$ which is an (approximate) inverse of the decoding scheme $\mathsf{Dcd} : \mathcal{R} \to \mathbb{C}^{N/2}$ defined by $p(X) \mapsto \Delta^{-1} \cdot \sigma(p(X))$, where $\Delta > 0$ is a positive real number called scaling factor in (Cheon et al., 2017) and σ is the complex canonical embedding map. The leveled FHE scheme CKKS of depth $L > 0$ is as follows.

- $\mathsf{KeyGen}\ (1^\lambda) \to (\mathsf{sk}, \mathsf{pk}, \mathsf{evk})$: From security parameter λ, returns a secret key $\mathsf{sk} = (s, 1) \in \mathcal{R}^2$, a public key $\mathsf{pk} = (a, -a \cdot s + e) \in \mathcal{R}_q^2$ and evaluation keys evk and rk for multiplication and rotation, respectively.
- $\mathsf{Enc}(\mathsf{pk} = (a, b), \mathsf{pt}) \to \mathsf{ct}$: Samples $\vec{r} \in \mathcal{R}^2$, and $e_0, e_1 \leftarrow \mathcal{DG}_{q_L}$. Returns a ciphertext $\mathsf{ct} \leftarrow \vec{r} \cdot \mathsf{pk} + (e_0, \mathsf{pt} + e_1)$.
- $\mathsf{Dec}(\mathsf{sk} = (s, 1), \mathsf{ct} = (c_0, c_1)) \to \mathsf{pt}$: Outputs $\mathsf{pt} \leftarrow c_1 + c_0 \cdot s \mod q_\ell$.
- $\mathsf{Add}(\mathsf{ct}_0, \mathsf{ct}_1) \to \mathsf{ct}_{\mathsf{add}}$: Outputs $\mathsf{ct}_{\mathsf{add}} \leftarrow \mathsf{ct}_0 + \mathsf{ct}_1 \mod q_\ell$. In its simplified notation, we express $\mathsf{ct}_0 + \mathsf{ct}_1$ as $\mathsf{Add}(\mathsf{ct}_0, \mathsf{ct}_1)$. Note we can add plaintext or a constant

to a ciphertext in a natural way.

- $\mathsf{Mult}(\mathsf{evk}, \mathsf{ct}_0, \mathsf{ct}_1) \to \mathsf{ct}_{\mathsf{mult}}$: Parses $\mathsf{ct}_0 = (c_0, c_1)$ and $\mathsf{ct}_1 = (c'_0, c'_1)$. Computes $(d_0, d_1, d_2) \leftarrow (c_0 \cdot c'_0, c_0 \cdot c'_1 + c'_0 \cdot c_1, c_1 \cdot c'_1) \in \mathcal{R}^3_{q_\ell}$. Returns $\mathsf{ct}_{\mathsf{mult}} \leftarrow (d_0, d_1) + \lfloor \frac{1}{P} \cdot (d_2 \cdot \mathsf{evk} \mod P \cdot q_\ell) \rceil \in \mathcal{R}^2_{q_\ell}$.
- $\mathsf{Rescale}_{\ell \to \ell'}(\mathsf{ct}) \to \mathsf{ct}'$: For an input ct a level ℓ ciphertext, returns $\mathsf{ct}' \leftarrow \lfloor \frac{q_{\ell'}}{q_\ell} \cdot \mathsf{ct} \rceil \in \mathcal{R}^2_{q_{\ell'}}$ at level ℓ'.
- $\mathsf{Rot}(\mathsf{rk}, \mathsf{ct}, i) \to \mathsf{ct}'$: For an input rotation key rk, rotates the plaintext vector of ct by i positions and return ct', where right-shift is represented as $i > 0$ and left-shift is represented as $i < 0$.
- $\mathsf{Boot}_{\mathsf{evk}}(\mathsf{ct}) \to \mathsf{ct}'$: In leveled HE, level refers to the remaining number of further multiplication. The consumption of the level is denoted as depth. After a few multiplications, the number should be refreshed by the bootstrapping algorithm, which allows further multiplications (Cheon et al., 2018a). Note that bootstrapping is a very heavy operation, and our goal is the construction of a large-sized LUT with low depth, without bootstrapping.

B.3. Precision Estimation of CKKS

Since CKKS is an approximate HE, the noise occurs after each algorithm for the supported operations in the scheme. For example, for ct is encryption of pt, $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}) - \mathsf{pt}$ could be nonzero. Thus, we inspect the precision loss of CKKS basic operations from the perspective of the size bound of the noise.

To track the noise efficiently, we will use two tags v and B with a ciphertext ct. For $\langle \mathsf{sk}, \mathsf{ct} \rangle \mod q = \mathsf{pt} + e$, $\|\mathsf{pt}\|^{\mathsf{can}}_\infty / \Delta \le v$ and $\|e\|^{\mathsf{can}}_\infty / \Delta \le B$ [5], denote (ct, v, B) as valid ciphertext. Note this notation is for dropping the effect of scaling factor, so for example, if Δ varies to Δ^2 after Mult, v and B is evaluated with division by Δ^2 to the canonical norm of pt and e.

The following are noise bounds after entailed for CKKS scheme algorithm:

- $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{pt}) \to \mathsf{ct}$: The noise bound of an encryption is less than B_{clean} in overwhelming probability.
- $\mathsf{Add}(\mathsf{ct}_0, \mathsf{ct}_1) \to \mathsf{ct}$: The addition of $(\mathsf{ct}_i, v_i, B_i)$ results $(\Sigma \mathsf{ct}_i, \Sigma v_i, \Sigma B_i)$
- $\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_0, \mathsf{ct}_1) \to \mathsf{ct}$: The multiplication of $(\mathsf{ct}_i, v_i, B_i)$ results $(\mathsf{ct}, v_0 v_1, B_0 B_1 + v_0 B_1 + v_1 B_0 + B_{mult})$. B_{mult} varies by the level of ciphertext and it decreases linearly as ciphertext modulus decreases by rescaling. For ease of computation, we fix B_{mult} of the highest level at computation omitting the level.
- $\mathsf{Mult}(c, \mathsf{ct}) \to c \cdot \mathsf{ct}$: The constant scalar multiplication results in the valid ciphertext $(c \cdot \mathsf{ct}, cv, cB)$, and constant vector multiplication results in the valid ciphertext $(c \cdot \mathsf{ct}, \|c\|_{max} v, \|c\|_{max} B)$.
- $\mathsf{Rescale}(\mathsf{ct}) \to \mathsf{ct}'$: For scaling factor Δ, the result of rescaling is $(\mathsf{ct}, v, B + B_{rescale})$. Since we defined v and B as scaling factor free notation, they varies by $B_{rescale}$ only.

[5]For cannonical embedding map $\sigma : \mathcal{R} \mapsto \mathbb{C}^{N/2}$ and plaintext polynomial $p(X)$, $\|p(X)\|^{\mathsf{can}}_\infty$ be canonical-norm, equal to $\|\sigma(p(X)\|_\infty$ (Cheon et al., 2017).

$\mathsf{Rescale}$ is usually operated after each Mult operation and summed up noise bound of $\mathsf{Rescale}(\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_0, \mathsf{ct}_1)) \to \mathsf{ct}$ is derived as $v_0 v_1, B_0 B_1 + v_0 B_1 + v_1 B_0 + B_{mult} + B_{rescale}$. For convenience of notation, We will denote $B_{mult} + B_{rescale}$ as B_*. Note this value is determined by the scheme parameter, not the message and noise bound of an involved ciphertext. So we can assume this bound B_* as a fixed constant over noise estimation.

For the detailed derivations of the noise bounds mentioned above, please refer to (Cheon et al., 2017). Due to accumulated noise, CKKS ciphertext entails inseparable noise, and improvement of precision meets the limit at a certain point. When we design an algorithm based on CKKS, the obtained precision loss of the entire algorithm implies how accurately the algorithm works. Therefore, we will analyze the noise bound of our proposed algorithm to demonstrate its viability.

C. Related Works

This study proposed a novel privacy-preserving and efficient LUT evaluation method using CKKS by constructing the EIF, demonstrating its effectiveness for embeddings. In the following subsections, we provide a comprehensive review of the related works of EIF, LUT, and NLP with CKKS. Also, we provide a comparison between our method and (Lee et al., 2022) in terms of the ciphertext expansion in Section C.4.

C.1. Encrypted Indicator Function

In the context of CKKS, a comparison function (Cheon et al., 2020) could be a component of the equality test and indicator function. However, this method is less efficient for equality-only tests due to its higher computational cost to achieve comparable precision. Another possibility involves using a Sinc function approximation to map discrete values to zero, with only zero mapped to 1 (Lee et al., 2023). This mechanism is also less efficient in terms of cost. Detailed discussion will be presented in Section D.3.1.

C.2. Look-up Table

A possible LUT construction was proposed in (Lee et al., 2023) by using their Sinc approximation. However, this method requires iterating over the Sinc approximation function as many times as the size of the LUT domain, as the strawman solution in Section 4.1. Even if the construction of LUT from EIF is essentially equivalent to our basic version LUT method, the overall computational cost is less efficient than ours because of the heavier EIF using Sinc.

Other FHE schemes such as TFHE (Chillotti et al., 2016; 2020) can be used to construct LUT (Guimarães et al., 2021) utilizing their operation on boolean circuits. However, they have a much smaller slot size than CKKS, which induces inefficiency of solution in terms of amortized running time to deal when requiring large messages such as NLP applications. Other privacy-preserving LUT operations based on non-HE primitives, like Multi-Party Computation (Launchbury et al., 2012; Brüggemann et al., 2023), are available. However, because our approach primarily targets minimizing communication overhead, operating with a single server model, and offloading computationally intensive tasks to the server, we focus on HE-based solutions.

C.3. NLP with FHE

Building ML-based language models necessitates converting natural language to numerical vectors like TF-IDF or word/token embeddings (e.g., word2vec, GloVe, Bert) (Mikolov et al., 2013; Pennington et al., 2014a; Pires et al., 2019). Constructing these vectors inherently involves managing large word/token dictionaries, which can pose significant bandwidth challenges, especially in the context of developing PPML with FHE. Notably, recent language models (Brown et al., 2020; Pires et al., 2019) rely on a word/token embedding layer that can be implemented through LUT evaluation. However, while CKKS might offer advantages for PPML models, its inability to handle exact discrete values makes it ill-suited to deal with LUT operations for the embedding layer.

The existing NLP research using CKKS avoids this problem by following two methods. The first approach is sending the encrypted one-hot vectors (Badawi et al., 2020). Since an embedding consists of many vocabularies, the client suffers from the high dimensionality of one-hot embedding. A detailed description of the ciphertext expansion by one-hot embedding is on the section C.4. The second approach is performing an embedding layer in plaintext on the client side (Lee et al., 2022; Chen et al., 2022). This method requires the assumption that the client knows the word/token embeddings (part of the language model) (Lee et al., 2022). However, the servers are owned by a service provider such as a commercial company, so the embedding layer is likely to become a private asset to the service provider. Therefore, revealing the embedding model to the client would pose serious privacy or security threats to the service provider.

With these reasons, the necessity of large-sized LUT evaluation by CKKS arises. If word/token embedding is executed by the server and the client sends ciphertext of words or their indices, it can ease the bandwidth problem and the privacy issue of the table.

C.4. Comparison with (Lee et al., 2022) on the Ciphertext Expansion

In this section, we discuss the effectiveness of our coded LUT algorithm with respect to the ciphertext expansion. In (Lee et al., 2022), they consider an approach to delegate the word embedding computations to the client and compute the rest of the encrypted NLP task using the CKKS FHE scheme. We show that our method provides better efficiency compared to that in (Lee et al., 2022) in terms of ciphertext expansion, analyzing and comparing how many message slots a word occupies in the respective cases since it is asymptotically proportional to the ciphertext expansion for the HELUT process.

As analyzed in (Badawi et al., 2020), a word is firstly transformed into a one-hot encoding before being processed further. Hence, it requires a word to be transformed into a vector size that is equal to the input size of LUT. For example, if we use GloVe 6B50d, one word is expanded into a vector of dimension $400K$. This implies that if we encrypt it after transforming it into one-hot encoding, one word occupies $400K$ slots which is infeasible for the practical usages. If the embedding layer is delegated to the client as in (Lee et al., 2022), a word occupies as many slots as the number of features, *e.g.,* 50 slots per word to use GloVe 6B50d, and 300 slots per word for GloVe 42B300d.

In our solution, a word is transformed into a code of size l, the number of codebooks, so one word occupies l slots. Hence, the ciphertext expansion rate depends neither on the feature dimension of the embedding nor on the size of the LUT. According to Table 2, if we use 16×32 coding, *i.e.,* $l = 32$, one word occupies 32 slots for GloVe 42B300d which is about 9.3x reduced compared to 300 slots in case the word/token embedding layer is delegated to the client as in (Lee et al., 2022).

D. Discussion

D.1. Further Adjustment of Parameter Selection for EIF

We propose a parameter selection algorithm for EIF in Algorithm 2. If the noise bound B of the input ciphertext and parameter B_* is given, we can select r, the round number of SqMethod, and u, so that the message range of SqMethod is obtained by Theorem 3.1. Then we can obtain s, the round number of Cleanse by predicting the precision improvement of Cleanse.

Further adjustment of r, u, s **:** The number obtained by s can be non-optimal since the noise calculation of Theorem 3.3 gives an upper bound of s. Thus, further optimization is required by experiment to reduce s for the given

Algorithm 5 Precision Evaluator by r, u, s for Indicator

Parameter $p \in \mathbb{Z}_+$, N : degree of CKKS ciphertext dimension.
Input EIF parameter (r, u, s)
Output precision prec
Procedure :

```
1: Generate random $x \leftarrow Z_p^{(N/2)}$
2: prec $\leftarrow -\log \|\mathsf{Dec} \circ \mathsf{Indicator}(\mathsf{Enc}(x)) - \delta(x)\|$
3: Return $(r, u, s)$
```

Algorithm 6 Optimized Parameter Selection of r, u, s for Indicator

Parameter $p \in \mathbb{Z}_+$
Input CKKS parameter $(N, \Delta, h, \sigma, P, q)$. N is degree of ciphertext space, Δ is scaling factor, h is hamming weight of secret key, σ is standard deviation of noise sampler, P is evaluation key parameter, q is ciphertext modulus.
Output (r, u, s)
Procedure :

```
1: $B_{clean} \leftarrow 8\sqrt{2\sigma N} + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}$
2: $B_{ks} \leftarrow 8\sigma N/\sqrt{3}$
3: $B_{scale} \leftarrow N/3 \cdot (3 + 8\sqrt{h})$
4: $B_{mult} \leftarrow P^{-1}qB_{ks} + B_{scale}$
5: $B_* \leftarrow B_{mult}/\Delta + B_{scale}$
6: $(r, u, s) \leftarrow \mathsf{paramInd}(p, B, B_*)$
7: while $True$ do
8:     if Precision$(r, u, s)$ < Precision$(r, u, s+1)$ then
9:         $s \leftarrow s + 1$
10:    else
11:        break
12:    end if
13: end while
14: while $True$ do
15:     if Precision$(r-1, u, s)$ < Precision$(r, u, s)$ then
16:         $r \leftarrow r - 1$
17:     else
18:         break
19:     end if
20: end while
21: Return $(r, u, s)$
```

parameter of CKKS.

Also, focusing on the optimization of cost, maximizing u is not always the best choice, since the error that occurs from the evaluation of Cleanse is lower-bounded. For instance, even if we have sufficient noise budgets to set r to achieve u up to 12, if Cleanse can improve the precision by up to 18 bits at maximum, setting u by 10 is enough to achieve the same precision while less computation is required.

Algorithm 6 provides further optimization of r, u, s for parameter selection, which is based on the precision evaluation given in Algorithm 5. For enhanced accuracy in precision evaluation, Algorithm 5 can be executed multiple times. The CKKS parameter computation in lines 1-5 of Algorithm 6 is based on (Cheon et al., 2017). However, these computations can be adjusted if other variants such as RNS (Cheon et al., 2018b) are used.

D.2. Cost Analysis and Comparison of the Possible HELUT Solutions

In this section, we discuss and compare the cost of various evaluation methods of HELUT from strawman solutions to coded embedding. We summarize the comparison of the costs in Table 4.

We remark that the HELUT can be evaluated using a table with both encrypted and unencrypted entries. The Mult column in Table 4 refers to either the number of multiplications between ciphertexts in the respective HELUT methods when using the encrypted table or that of multiplication between plaintext and ciphertext when using the unencrypted table.

HELUT-LT. HELUT-LT refers to the naive method to apply Indicator for $k = p^l$ times, which is the size of LUT inputs, and multiply the k results with the output values of the table. Adding up this value will result in the ciphertext of the evaluation result, as in Equation (8) and Equation (9). This procedure costs $k = p^l$ multiplications and additions for each entry of the embedding vector, so the total cost is multiplied by d, the embedding dimension. Addition of p^l ciphertexts induces noise bound $p^l \cdot B\|\boldsymbol{E}\|_{\max}$, $\boldsymbol{E}$ denoting embedding matrix from LUT. Note the evaluation depth is the depth of Indicator plus one.

This method is different from (Badawi et al., 2020) by Indicator application part, since in (Badawi et al., 2020), the client encrypts the message in OHE_k form. (Badawi et al., 2020) requires excessive memory due to OHE_k formed message. One-hot-encoding of an index consists of k binary numbers, which occupy k slots of certain ciphertexts. If the number of tokens is large as in the LLM model, one token occupies multiple ciphertexts. For instance, in the case of GloVe 42B300d, OHE of a token occupies $1.9M$ slots. If the slot size of CKKS is 65536, 29 ciphertexts are required for a single token. However, our method does not require that much ciphertext to transfer, so it solves excessive bandwidth problem.

HELUT-CI. HELUT-LT requires Indicator to check every candidate in $\mathbb{Z}_k$ for each input, where k can exceed a million in the LLM cases. We can reduce the number of Indicator operations by decomposing OHE_k into OHEbasis by formula (10). In this way, input is decomposed into l codes and encrypted, so it requires performing l Indicator evaluations on the ciphertexts with small domain $\mathbb{Z}_p$ and construct OHEbasis. Then we obtain each entry of

Table 4. Cost Analysis of HELUT. The second column represents slot occupation per the required number of input ciphertexts for a single token.

Method	Active slots per token	Indicator	Mult	Add	Rot	Noise bound
HELUT-LT	1 slot per ciphertext	p^l	dp^l	dp^l	0	$p^l \cdot B\|\boldsymbol{E}\|_{\max}$
HELUT-CI	1 slot per l ciphertexts*	pl	dp^l, lp^{l**}	dp^l	0	$p^l \cdot B\|\boldsymbol{E}\|_{\max}$
CodedHELUT	1 slot per l ciphertexts*	pl	dpl	dpl	0	$pl \cdot B\|\boldsymbol{E}\|_{\max}$
CodedHELUT+**p1**	pl slots per 1 ciphertext	1	d	$d \log pl$	$d \log pl$	$pl \cdot B\|\boldsymbol{E}\|_{\max}$

* input codes of dimension l are assumed to packed in each of l independent ciphertexts, so number of input ciphertext is l at minimum.

** This term is the cost of getting each entries of OHE_k from OHEbasis, which is inevitably Mult between ciphertexts.

OHE_k from OHEbasis by l ciphertext Mult, and total cost of this construction is lp^l Different from other Mult, this Mult is inevitably multiplication between ciphertexts since it is multiplication between OHEbasis.

This method reduces the number of Indicator operations from p^l to pl. Note that Indicator of former is on $\mathbb{Z}_k$ and the latter uses Indicator on $\mathbb{Z}_p$, so not only the number of Indicator but also the evaluation cost and depth of each Indicator is reduced.

CodedHELUT. Even if we use the HELUT-CI method, we should combine basis and construct OHE_k and multiply, add k times to evaluate each of d result entries. However, as in Algorithm 4, evaluation of codedHELUT requires evaluation of small LUT of size p, then simply adding l results outputs an approximate vector equivalent to embedding obtained by large LUT of size k. So this method reduces Mult, Add costs from dp^l to dpl. Also, since the number of addition is reduced, the noise bound is also reduced to $pl \cdot B\|\boldsymbol{E}\|_{\max}$.

Parallelization. We can amortize the overall communication and computation costs utilizing the large slot size of CKKS. We can parallelize the computation in the following way denoted as **p1** and **p2**.

- **p1** represents a ciphertext packing method for parallel computation of line 1 in Algorithm 4. More precisely, when applying Indicator on each code, we can copy the code p times and evaluate single Indicator operation, one Mult, $\log p$ Rot, and Add to obtain the result of single Indicator on $\mathbb{Z}_p$. Packing multiple codes in one ciphertext utilizes l slots per input, and reduces Indicator cost by l times. Then simultaneously Mult table, then summing up l outputs requires $\log l$ Rot and Add operations. In total, this SIMD computation of coded input evaluation reduces the number of Indicator operations by pl times, then requires single table Mult and Rot and Add $\log pl$ to sum up. Note this procedure requires pl slots per input.
- **p2** is achieved after fixing the occupation cost for one input indices, as in slot per token column in Table 4. Then we can parallelize the whole procedure by taking multiple input indices in one ciphertext. If we set CKKS ciphertext slot size as n and assuming **p1**, one ciphertext can contain $n/(pl)$ input indices and operate Algorithm 4 in parallel.

D.3. Other Design Choices

D.3.1. Construction of EIF

In this section, we discuss the alternative approaches to constructing EIF. We will then compare the efficiency and precision of these methods against our proposed EIF method with experimental results in Appendix E.4

Lagrange Interpolation As a straw man solution, simple Lagrange Interpolation can be considered as EIF for domain $\mathbb{Z}_p$ as follows:

$$\Pi_{i \in \mathbb{Z}_p \setminus \{a\}} \frac{(x-i)}{(a-i)}.$$

This method is inefficient when applied to large input domains (large p) due to its multiplicative cost scaling with $p-1$.

Comparison Function There is a step function approximation that could be used as a comparison function using CKKS (Cheon et al., 2020). The step function introduced in the paper maps 0 to 0, and other values to ± 1. We can square the result and subtract from 1 to build an EIF. However, it requires more computational cost and depth to reach high precision compared to the square method in Section 3.1. Thus, we recommend the square method over the step function from (Cheon et al., 2020) for the equality test.

Sinc Approximation An approximation of Sinc with certain noise cleansing functions is suggested as a candidate for the indication function (Lee et al., 2023), since if the domain is restricted to integer points, $\mathsf{Sinc}(\pi x) = \delta_0(x)$

holds.

D.3.2. Construction of LUT

In this section, we discuss the alternative solutions of HELUT and analyze their efficiency compared to our method.

LUT by Comparison Function We can detect the number in ciphertext by binary search with comparison function in (Cheon et al., 2020), operating $\log p$ times recursively. However, this method consumes depth rapidly, since the comparison operation should be operated on the result of the former comparison iteration, so the depth accumulates by using the comparison function iteratively. As a result, this method requires bootstrapping, which leads to an inefficient computation cost than our low-depth, no bootstrapping construction.

LUT by Lagrange Interpolation Instead of applying the indicator multiple times and checking whether the input value matches each input candidate, we can construct a polynomial approximation of LUT. However, since the input of interpolation is not the exact value in discrete input space, the output of the interpolation might contain amplified noise due to both input difference and FHE evaluation noise. In this case, we can not cleanse the noise since the output and the noise might have no consistency so we can not construct an efficient cleansing function.

D.4. Limitations and Future Work

This section discusses the limitations of our current approach and outlines potential directions for future research to address these challenges and further enhance the applicability of our method.

Generalizability of LUT evaluation Our method proposes a HE-friendly algorithm for evaluating look-up tables in embedding layers, where the tables contain token embeddings. This method relies on the compression learning method (Shu & Nakayama, 2018), which utilizes the semantic relationships between embeddings. Thus, this method can be extended to other structured large dataset, such as a recommendation system (i.e., user/item embedding) and graph neural networks (i.e., node/edge embeddings) (Gao et al., 2023; Barkan & Koenigstein, 2016). However, this method is difficult to generalize to the lookup tasks for arbitrary tables.

Dependence on Compression Strategy The efficacy of current CodedHELUT is influenced by the performance of the compression learning method. This limitation also stems from the assumption that our method aims to evaluate the embedding layer of an already trained model, utilizing the compression learning method by (Shu & Nakayama, 2018) as an off-the-shelf method. Thus, it is required to explore methods to improve compression learning, such as fine-tuning compressed embedding layers or training models with coded input tokens from the beginning of training in order to enhance the overall efficiency of CodedHELUT.

E. Additional Experiments

E.1. Experimental Details

Coded Compression Detail: Our compression method follows the setting of (Shu & Nakayama, 2018). We trained the model with the code embedding matrices and the discrete codes that are constructed by a neural network with the Gumbel-Softmax trick, and Adam optimizer with a learning rate 0.001. The original implementation used the dataset of vocabularies from the intersection of GloVe and IMDB data set, but our compression includes the entire vocabulary from GloVe. We conducted the training for $200K$ iterations and evaluated MSE in every 1000 iterations. We selected a parameter with the lowest MSE. This compression is conducted in plaintext. Note that the model size is significantly reduced by the compression. While the size of the original embedding is 271.3MB for GloVe 6B50d and 5.03GB for GloVe 42B300d, respectively, the codebooks of GloVe 6B50d and 42B300d with $pl = 512$ become 391KB and 2.4MB, respectively.

E.2. IMDB Sentimental Analysis with Compressed Embeddings

To demonstrate the effectiveness of compressed embeddings for a downstream task, we evaluated their performance on the sentiment analysis task using the IMDB movie review dataset (Maas et al., 2011) as in (Shu & Nakayama, 2018). We conducted four independent trials and measured the average accuracy. The IMDB dataset contains 25,000 reviews for both the training and validation sets. We truncated each review to a maximum length of 400 words. Our baseline was set using the original embeddings, against which we compared the performance of the compressed embeddings. The classifier was trained for up to 50 epochs using GloVe 42B300d embeddings and up to 100 epochs using GloVe 6B50d embeddings. The results are summarized in Table 5.

E.3. Implementation of Indicator with Various p and Δ

To demonstrate the efficiency of our EIF, we evaluated it across various input domains, $\mathbb{Z}_p$ with $p = 2^i, 2 \leq i \leq 6$ for $\Delta = 35$ and $2 \leq i \leq 8$ for $\Delta = 50$[6]. Figure 4 and Figures 5 illustrate the amortized times for a ciphertext with 2^{16} slots and the total consumed depth due to the rescaling in CKKS computation. In Figures 4 and 5, we can observe

[6]The evaluation results of $p = 128, 256$ for $\log \Delta = 35$ are absent since Algorithm 1 outputs $u < 2$, and the application condition for Cleanse is not satisfied.

Table 5. Experimental result on IMDB sentimental analysis task with compressed embedding of GloVe 6B50d, GloVe 42B300d

Setting		**Accuracy**	
p	l	**6B50d**	**42B300d**
8	8	69.32±4.49	71.63±2.53
8	16	76.65±3.05	80.57±1.69
8	32	79.14±4.28	82.93±0.96
8	64	79.49±3.64	85.27±0.63
16	32	79.56±1.03	85.18±1.54
Original		79.64 ± 2.10	85.74 ± 1.04

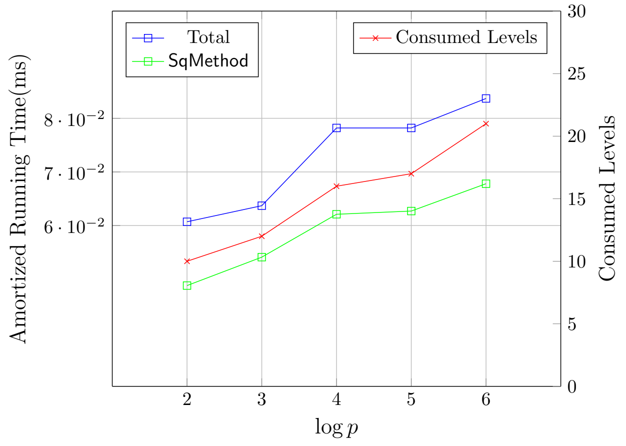

Figure 4. Amortized time(ms) and consumed levels of EIF for $2 \le \log p \le 6$ with $\log \Delta = 35$, $\log q = 955$

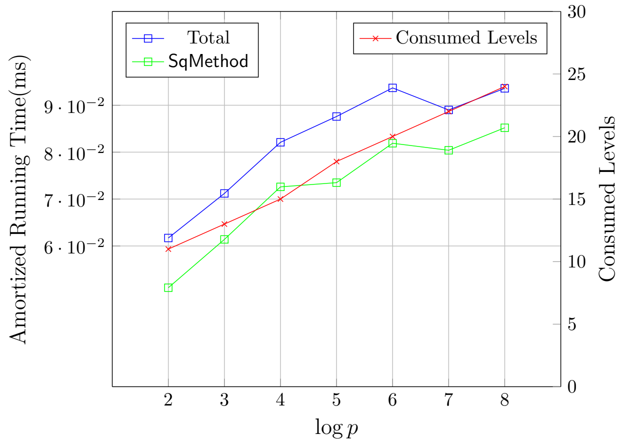

Figure 5. Amortized time(ms) and consumed levels of EIF for $2 \le \log p \le 8$ with $\log \Delta = 50$, $\log q = 955$

Table 6. Suggested Parameter of EIF on Fresh Ciphertext, $\log N = 17$, $\log \Delta = 35$ for 15 bit precision

p	**4**	**8**	**16**	**32**	**64**
r	6	8	10	11	13
u	11	9	7	5	4
s	1	1	2	2	3
Depth	8	10	14	15	19

Table 7. Suggested Parameter of EIF on Fresh Ciphertext, $\log N = 17$, $\log \Delta = 50$ for 30 bit precision

p	**4**	**8**	**16**	**32**	**64**	**128**	**256**
r	7	9	11	12	14	16	18
u	24	23	21	11	11	11	11
s	1	1	1	2	2	2	2
Depth	9	11	13	16	18	20	22

that the amortized running time is approximately linearly proportional to $\log p$, as multiplication cost increases with additional rounds of SqMethod. The difference between total cost and SqMethod represents the cost of Cleanse, and the difference remains stable if s is the same. For each p, Tables 6 and 7 provide the parameter sets (r, u, s) obtained by Algorithm 6, where the consumed level is calculated as $2 + r + 2s$. We found that with a larger scaling factor Δ (Δ=50), SqMethod reaches larger values of u for similar r, and Cleanse achieves a higher precision limit with smaller s.

Precision As we discussed in Proposition 3.3, the precision limit is defined by the size of B_*. In our experiment, we obtained uniform precision regardless of the domain size p. The worst precision limit is 15 for $\log \Delta = 35$, and 30 for $\log \Delta = 50$.

E.4. Cost and Precision Analysis of Other Design Choices for EIF

We compared our proposed EIF with three other methods presented in Section D.3.1: EIF using Lagrangian interpolation (Lagrange), Comprarison-based EIF (Cheon et al., 2020), and EIF using Sinc (Lee et al., 2023). Tables 8-11 prenet the consumed depth, the number of Mult and constant Mult operations, the precision bits, and the average amortized time for 8 repetitions across different configurations of $\log \Delta = 35, 50$ and $p = 8, 64$.

For $p = 8$, we observed that Lagrange interpolation requires the least depth, but its multiplication cost increases linearly with p, while the costs of the other algorithms scale logarithmically with p. In addition, for $p > 10$, the coefficients of the interpolation polynomial grow excessively large, leading to a collapse in the evaluation results. Consequently, Lagrange interpolation failed to evaluate EIF for $p = 64$. Our proposed EIF achieved the highest precision with minimal cost across all cases except for Lagrange. Although the EIF using Sinc (Lee et al., 2023) requires less depth, the computational cost is inferior to ours as it involves computations of Sinc and cosine, which are computationally intensive.

Table 8. Cost and precision comparison of other design choices of EIF with $\log \Delta = 35$ and $\log p = 8$

Method	**Depth**	Mult	**const** Mult	**Precision**	**Cost**
Lagrange	6	9	7	7 bits	0.0657 ms
(Cheon et al., 2020)	18	21	21	14 bits	0.1499 ms
(Lee et al., 2023)	11	22	19	11 bits	0.1462 ms
Proposed EIF	12	11	1	16 bits	0.0762 ms

Table 9. Cost and precision comparison of other design choices of EIF with $\log \Delta = 35$ and $\log p = 64$

Method	**Depth**	Mult	**const** Mult	**Precision**	**Cost**
(Cheon et al., 2020)	18	21	21	12 bits	0.1735 ms
(Lee et al., 2023)	14	28	19	5 bits	0.1867 ms
Proposed EIF	21	20	1	16 bits	0.1087 ms

Table 10. Cost and precision comparison of other design choices of EIF with $\log \Delta = 50$ and $\log p = 8$

Method	**Depth**	Mult	**const** Mult	**Precision**	**Cost**
Lagrange	6	9	7	31 bits	0.0698 ms
(Cheon et al., 2020)	18	21	21	22 bits	0.1629 ms
(Lee et al., 2023)	11	22	19	25 bits	0.1588 ms
Proposed EIF	13	12	1	31 bits	0.0832 ms

Table 11. Cost and precision comparison of other design choices of EIF with $\log \Delta = 50$ and $\log p = 64$

Method	**Depth**	Mult	**const** Mult	**Precision**	**Cost**
Comparison (Cheon et al., 2020)	18	21	21	22 bits	0.1822 ms
Sinc (Lee et al., 2023)	14	28	19	21 bits	0.1900 ms
Proposed EIF	20	19	1	31 bits	0.1113 ms