Amortized Large Look-up Table Evaluation with Multivariate Polynomials for Homomorphic Encryption

Heewon Chung[1], Hyojun Kim[1], Young-Sik Kim[2], and Yongwoo Lee[3]

[1]DESILO Inc., Seoul, Republic of Korea
{heewon.chung, hyojun.kim}@desilo.ai
[2]Daegu Gyeongbuk Institute of Science and Technology, Daegue, Republic of Korea
ysk@dgist.ac.kr
[3]Inha University, Incheon, Republic of Korea
yongwoo@inha.ac.kr

Abstract

We present a new method for efficient look-up table (LUT) evaluation in homomorphic encryption (HE), based on Ring-LWE-based HE schemes, including both integer-message schemes such as Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski/Fan-Vercauteren (BFV), and complex-number-message schemes like the Cheon-Kim-Kim-Song (CKKS) scheme. Our approach encodes bit streams into codewords and translates LUTs into low-degree multivariate polynomials, allowing for the simultaneous evaluation of multiple independent LUTs with minimal overhead. To mitigate noise accumulation in the CKKS scheme, we propose a novel noise-reduction technique, accompanied by proof demonstrating its effectiveness in asymptotically decreasing noise levels.

We demonstrate our algorithm's effectiveness through a proof-of-concept implementation, showcasing significant efficiency gains, including a 0.029ms per slot evaluation for 8-input, 8-output LUTs and a 280ms amortized decryption time for AES-128 using CKKS on a single GPU. This work not only advances LUT evaluation in HE but also introduces a transciphering method for the CKKS scheme utilizing standard symmetric-key encryption, bridging the gap between discrete bit strings and numerical data.

Keywords. Homomorphic encryption (HE), look-up table (LUT), multivariate polynomial, transciphering

1 Introduction

Homomorphic encryption (HE) serves as a crucial tool in secure computation, enabling computations on encrypted data without revealing sensitive information. Typically, HE schemes are classified into two categories: word-wise and bit-wise HE. Brakerski-Gentry-Vaikuntanathan (BGV) [BGV14] / Brakerski-Fan-Vercauteren (BFV) [Bra12, FV12] / Cheon-Kim-Kim-Song (CKKS) [CKKS17] fall under the word-wise category, supporting addition and multiplication with a notable advantage in naturally accommodating Single Instruction, Multiple Data (SIMD) operations. Specifically, BGV and BFV excel in operations within $\mathbb{Z}_t$ with exact precision, making them suitable for applications like private information retrieval. On the other hand, CKKS facilitates operations over

complex (including real) numbers, making it particularly efficient for various machine learning applications. However, a limitation common to BGV/BFV/CKKS is their restriction to basic arithmetic operations, necessitating workarounds such as approximation polynomials for non-arithmetic operations like min/max.

Ducas and Micciancio [DM15] introduced a groundbreaking HE approach that supports arbitrary binary operations, also known as functional bootstrapping, through blind rotation techniques. Subsequent works [CGGI17, CGGI20] have further enhanced this method. DM-like HE is advantageous for its small parameter size and the ability to perform arbitrary binary gate operations, allowing for a wide range of computations within homomorphic encryption. Recent advancements [YXS$^+$21, Zam22, LMP22] aims to extend DM-like HE to support more general operations and handle multiple bits by utilizing larger parameters.

However, DM-like HE faces two primary limitations. Firstly, unlike BGV-like HE, it does not inherently support SIMD operations, leading to lower throughput. Secondly, to support multi-bit plaintext, a quadratic increase in parameter size is required, leading to efficiency loss. According to Bergerat et al. [BBB$^+$23], performance significantly deteriorates when dealing with LUT larger than 8 bits. Consequently, circuit design often involves multiple small LUTs, posing challenges in parameter selection and the tradeoff in the number of gates and performance of HE operations.

This work is motivated by the observation that constructing an efficient algorithm to support an arbitrary circuit with large input and output and SIMD operation simultaneously is important, as it can significantly enrich the application of HE. In particular, our focus is on executing LUTs with large input and output sizes.

1.1 Contribution

The following are two primary contributions of this paper.

Firstly, we propose a novel method for the secure evaluation of LUTs with large inputs and outputs, leveraging RLWE-based HE schemes. This method is characterized by its use of multivariate polynomials to manage substantial input sizes, making it compatible with all RLWE-based HE schemes such as BGV, BFV, and CKKS. A notable advantage of our approach is its reliance on low-degree multivariate polynomials, which offers significant benefits compared to existing solutions. These include the ability to process multiple inputs through a single LUT evaluation, enhancing efficiency; the capability to conduct slot-wise evaluations of distinct LUTs within the same ciphertext without incurring additional costs; and the facilitation of multiple independent LUT evaluations on the same ciphertext with minimal additional computational overhead.

Secondly, we introduce a sophisticated noise-reduction technique that effectively mitigates the inherent noise in the CKKS scheme when applied to our LUT evaluation method. Considering the inherently noisy nature of the CKKS scheme, attributed to its approximate arithmetic, directly diminishing the signal-to-noise ratio (SNR) poses a significant challenge. Inspired by Cheon et al. [CKK20] and Drucker et al. [DMPS22], we address the CKKS scheme's noise by discretizing inputs, proposing an effective noise-reduction algorithm. Our noise-reduction approach employs a low-degree polynomial, requiring less than five multiplications. We also offer proof that this method can diminish the initial noise, measured by SNR $\epsilon < 1$, to a notably reduced level, expressed as $c \cdot \epsilon^2$, with c being a minor constant.

To empirically validate the performance of our proposed scheme, we implemented AES decryption as a proof-of-concept, achieving a decryption time of 330ms per AES ciphertext. This implementation marks the first instance of a transciphering framework that employs the AES stan-

dard over the CKKS scheme. Besides, our technique effectively avoids the Li-Micciancio attack as it ensures the absence of noise in the message after decryption and decoding.

1.2 Technical Overview

For a function $T_{\ell\to\ell} : \{0,1\}^\ell \to \{0,1\}^\ell$ as a LUT, we introduce an encoding algorithm encode to embed a binary vector of length ℓ into a plaintext space HE scheme. In particular, we define

$$\mathsf{encode}(\vec{x}) = \zeta^{\mathsf{int}(\vec{x})},$$

where ζ is a primitive 2^ℓ-th root of unity of $\mathbb{Z}_t$ and $\mathbb{C}$, for BGV/BFV and CKKS, respectively with $\mathsf{int}(\vec{x})$ is a bijection from a ℓ-bit string to an integer $[0, 2^\ell)$. Then, there exists a polynomial of degree $2^\ell - 1$ representing $T_{\ell\to\ell}$ and one can execute it homomorphically.

Our approach is generalized to a LUT with larger input, that is, for $T_{\alpha\ell\to\beta\ell} : \{0,1\}^{\alpha\ell} \to \{0,1\}^{\beta\ell}$. As a result, we can find out that β multivariate polynomials, each with α inputs, can collectively represent T with the same encoding function. Our method involves decomposing $T_{\alpha\ell\to\beta\ell}$ into a set of LUTs, $\{T^{(j)}_{\alpha\ell\to\ell} : \{0,1\}^{\alpha\ell} \to \{0,1\}^\ell\}_{j\in[0,\beta-1]}$, where $T^{(j)}_{\alpha\ell\to\ell}(x)$ is a j-th ℓ-bit fragment of $T_{\alpha\ell\to\beta\ell}$. Rather than interpolating a polynomial for each $T^{(j)}_{\alpha\ell\to\ell}$, we employ a multivariate polynomial.

We also provide a couple of methods for enhancing the efficiency. At first, our approach enables the evaluation of distinct LUTs for individual message slots, while maintaining computational cost. The underlying equation is that, given $f(x) = \sum_{i=0}^{n-1} a_i x^i$ and $g(x) = \sum_{i=0}^{n-1} b_i x^i$,

$$(f(m_0), g(m_1)) = \left(\sum_{i=0}^{n-1} a_i m_0^i, \sum_{i=0}^{n-1} b_i m_1^i\right) = \sum_{i=0}^{n-1} (a_i, b_i) \circ (m_0^i, m_1^i)$$

where $\circ$ denotes the Hadamard product. Deriving from this equation, we find that

$$\mathsf{Enc}(f(m_0), g(m_1)) = \sum_{i=0}^{n-1} \mathsf{encode}(a_i, b_i) \cdot \mathsf{Enc}(m_0, m_1)^i.$$

This implies that some distinct LUTs can be executed on each message slot by $\log n$ squaring ciphertext and $\log n$ ciphertext multiplications, combined with n scalar multiplications. This cost is equal to that of a single LUT evaluation. This observation can be generalized to multivariate polynomials with multiple inputs. Moreover, as a power of ciphertexts can be reused when evaluating other LUTs, it facilitates the efficient evaluation of multiple LUTs with minimal additional computation overhead. Secondly, we introduce a *noise-reduction function* $f : \mathbb{C} \to \mathbb{C}$, charaterized by $f(t) = t$ and $\|f(t+\epsilon) - f(t)\| < \|\epsilon\|$ for $t \in \mathbb{C}$. This function efficiently mitigates noise in the ciphertext, enabling the execution of arbitrary unbounded arithmetic circuits. We prove that $f(x)$ is a noise reduction function if any of its coefficients is zero. Specifically, we demonstrate that for $x = t + \epsilon$,

$$f(t) = -\frac{1}{n} t^{n+1} + \left(1 + \frac{1}{n}\right) t$$

serves as one of the concrete constructions for the noise-reduction function. It is proven in this paper that $\|f(t+\epsilon)\| \le c \cdot \epsilon^2$ for small constant c and $\epsilon \ll 1$.

We apply our method to AES-128 transciphering, breaking down its 10-round process into LUT-representable operations. The AddKey step, involving XOR with round keys, is modeled

as a 4-input/output LUT. SubBytes, executing multiplicative inverses in GF(2^8), translates into an 8-input/output LUT. ShiftRows and MixColumns combine into a single 8-input/output LUT, streamlining the process based on Gentry et al. [GHS12]. This approach demonstrates our method's utility in executing AES within a homomorphic encryption framework, optimizing cryptographic computations.

1.3 Related Work

Multi-bit DM-like HE. Homomorphic operations in DM-like ciphertexts leverage LUTs for gate evaluations, such as NAND, by mapping a set of 2-bit inputs (from $0, 1, 2$, representing the sum of two binary values) to 1-bit outputs. The original DM-like homomorphic encryption (HE) framework is optimized for efficiency due to small parameter sizes, yet it is inherently limited by noise sensitivity and the restriction to binary gates per bootstrapping cycle. This limitation necessitates multiple bootstrapping for complex circuits, with the count increasing with the number of gates.

To enhance the efficiency and reduce the number of required bootstrappings, there is a clear benefit in supporting larger LUTs, albeit at the cost of increased parameter sizes. Modern strategies aim to balance the LUT size with the overall cost by optimizing the message space for different data types, considering both the gate count and bootstrapping requirements. Bergerat et al. [BBB$^+$23] have pinpointed effective parameters for LUTs up to 8 bits within the TFHE framework.

Amortized bootstrapping in DM-like HE. Micciancio and Sorrel [MS18] introduced a groundbreaking approach for bootstrapping multiple DM-like ciphertexts simultaneously, termed amortized bootstrapping, which significantly reduced the asymptotic complexity per ciphertext. Despite its theoretical advancements, practical limitations persisted due to exponential noise growth and the inability to leverage smaller parameters.

Subsequent research, including works by [GPvL23, MKMS23, LW23a, LW23b], has focused on enhancing the efficiency of amortized bootstrapping. Parallelly, an alternative strand of amortization, explored by [CIM19, MKG23], concentrates on applying multiple gates to a single ciphertext, offering performance improvements in scenarios with numerous gates, despite not operating in a SIMD manner.

A notable recent development by Liu and Wang [LW23c] reimagines functional bootstrapping through a high-degree polynomial ($t - 1$ in $\mathbb{Z}_t$, where t is the message modulus, such as 65537) evaluation on BFV ciphertexts, achieving amortized efficiency. However, the crux of this method's runtime lies in the evaluation of this high-degree polynomial, necessitating optimizations such as finding polynomials with numerous zeros to enhance practicality.

Drucker et al.[DMPS22] introduced BLEACH, a novel technique utilizing the CKKS scheme for discrete operations, where binary gates are modeled using interpolated polynomials over real numbers, with Cheon et al.'s sign function[CKK20] aiding in noise reduction. This method, however, is confined to binary gates with single-bit inputs. Our approach broadens this concept to support LUTs handling multi-bit inputs, effectively generalizing the foundational work of Drucker et al.

1.4 Organization

In Section 2, we outline the foundational concepts of LUTs and HE schemes. The core algorithm for implementing large LUTs within an approximate HE framework, specifically the CKKS scheme, is detailed in Section 3. Section 4 delves into optimization strategies for managing LUTs of any

size. The adaptation of our principal methodology to the BGV and BFV schemes is explored in Section 5. The efficacy of our approach is evidenced through the evaluation of AES execution performance in Section 6. The paper concludes with final thoughts and remarks in Section 7.

2 Preliminaries

2.1 Notations

All logarithms are base 2 unless otherwise indicated. For $p \in \mathbb{Z}$, $\mathbb{Z}_p$ denotes the ring of integers modulo p. For $\alpha \in \mathbb{N}$, $\mathbb{F}^{\prod_{i=0}^{\alpha-1} n_i}$ denotes the vector space of dimension $n_0 \times \cdots \times n_{\alpha-1}$ over $\mathbb{F}$. An element in $\mathbb{F}^{\prod_{i=0}^{\alpha-1} n_i}$ is denoted by capital letters; for example, $A \in \mathbb{C}^{\prod_{i=0}^{\alpha-1} n_i}$, and each entry of A is represented by $A[i_0, \ldots, i_{\alpha-1}]$ where $i_0, \ldots, i_{\alpha-1} \in [0, n-1]$.

For a power of two N, we denote the $2N$-th cyclotomic ring by $\mathcal{R}_N := \mathbb{Z}[X]/\Phi_{2N}(X)$ and its quotient ring by $\mathcal{R}_{N,Q} := \mathcal{R}_N/Q\mathcal{R}_N$, where $\Phi_{2N}(X) = X^N + 1$. Ring elements in $\mathcal{R}_N$ are indicated in bold, e.g. $\boldsymbol{a} = \boldsymbol{a}(X)$.

We introduce notations for vector operators denoted by bold letters. The i-th element of vector $\vec{a}$ is represented as $a[i]$, and $\vec{a}_{[i:j]}$ denotes the subvector $(a[i], \cdots, a[j])$. The inner product of two vectors $\vec{a}$ and $\vec{b}$ is denoted as $\langle \vec{a}, \vec{b} \rangle$. Additionally, we denote the L2 and infinity norm by $\|\cdot\|_2$, $\|\cdot\|_\infty$ respectively, for a ring element $\boldsymbol{a}$ in $\mathcal{R}_N$ and a vector $\vec{a} \in \mathbb{Z}^n$. The norms of polynomials in $\mathcal{R}$ are the norms of its coefficients vector.

We write the floor, ceiling and round functions as $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ and $\lfloor \cdot \rceil$, respectively. $\lfloor x \rceil_p$ denotes the nearest multiple of p to x. For $q \in \mathbb{Z}$ and $q > 1$, we identify the ring $\mathbb{Z}_q$ with $[-q/2, q/2)$ as the representative interval, and for $x \in \mathbb{Z}$ we denote the centered remainder of x modulo q by $[x]_q \in \mathbb{Z}_q$. We extend these notations to elements of $\mathcal{R}_N$ by applying them coefficient-wise. We use $\boldsymbol{a} \leftarrow \mathsf{S}$ to denote uniform sampling from the set S. We denote sampling according to a distribution χ by $\boldsymbol{a} \leftarrow \chi$.

2.2 Look-up Table

An LUT is a data structure optimized for quick value retrieval, typically organized as a table or array where each entry directly maps a specific input to its corresponding output. This structure is particularly useful for applications necessitating rapid data access, serving as a crucial tool for efficient information retrieval. An n-to-m LUT, denoted as an *n-input, m-output LUT*, facilitates the mapping of n-bit inputs to m-bit outputs. LUTs are fundamental in the operation of Field-Programmable Gate Arrays (FPGAs), enabling them to be programmable and expedite arbitrary circuit functions.

Within HE, LUTs play a pivotal role in boosting computational efficiency. Given RLWE-based HE's inherent support for only addition and multiplication, the range of directly computable functions is limited. LUTs address this limitation by precalculating the outcomes of diverse functions for all possible inputs, thus enabling the efficient homomorphic evaluation of complex functions on encrypted data without engaging in intricate computations. The concept of utilizing LUTs for evaluating arbitrary binary gates in HE was first introduced by Ducas and Micciancio [DM15]. This idea was further expanded by Chillotti et al. [CGGI20], who integrated LUTs with HE to facilitate the homomorphic evaluation of a broader spectrum of functions, significantly enhancing the versatility and efficiency of HE applications.

2.3 Basic Lattice-based Encryption

We rewrite basic lattice-based encryptions following to [BGV14]. For positive integers q and n, basic LWE encryption of $m \in \mathbb{Z}$ under the secret key $\vec{s} \leftarrow \chi_{\mathtt{key}}$ is defined as

$$\mathsf{LWE}_{n,q,\vec{s}}(m) = (b, \vec{a}) = (-\langle \vec{a}, \vec{s} \rangle + e + m, \vec{a}) \in \mathbb{Z}_q^{n+1},$$

where $\vec{a} \leftarrow \mathbb{Z}_q^n$ and error $e \leftarrow \chi_{\mathtt{err}}$. We will occasionally drop subscripts n, q, and $\vec{s}$ in LWE when they are clear from the context.

For a positive integer Q and a power of two N, basic RLWE encryption of $\boldsymbol{m} \in \mathcal{R}$ under the secret key $\boldsymbol{z} \leftarrow \chi_{\mathtt{key}}$ is defined as

$$\mathsf{RLWE}_{N,Q,\boldsymbol{z}}(\boldsymbol{m}) := (\boldsymbol{b}, \boldsymbol{a}) = (-\boldsymbol{a} \cdot \boldsymbol{z} + \boldsymbol{e} + \boldsymbol{m}, \boldsymbol{a}) \in \mathcal{R}_{N,Q}^2,$$

where $\boldsymbol{a} \leftarrow \mathcal{R}_{N,Q}$ and $\boldsymbol{e} \leftarrow \chi_{\mathtt{err}}$. As in LWE, we will occasionally drop subscripts N, Q, and $\boldsymbol{z}$ in RLWE, when they are clear from the context.

2.4 Brief Overview of Homomorphic Encryption Schemes

HE allows for computations to be performed directly on ciphertexts without requiring decryption. A HE scheme can be represented by a quadruple of probabilistic polynomial-time algorithms, denoted by $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, which are defined as follows:

- $\mathsf{KeyGen}(\lambda) \to (pk, sk, evk)$: This algorithm takes a security parameter λ and the circuit depth L, and outputs a public key pk, a secret key sk, and an evaluation key evk.
- $\mathsf{Enc}(pk, m) \to c$: This algorithm takes a plaintext $m \in \mathsf{P}$ and a public key pk as inputs, and outputs a ciphertext $c \in \mathsf{C}$, where P and C denote the plaintext space and the ciphertext space, respectively.
- $\mathsf{Dec}(sk, c) \to m$: This algorithm takes a ciphertext $c \in \mathsf{C}$ and a secret key sk as inputs, and outputs a plaintext $m \in \mathsf{P}$.
- $\mathsf{Eval}(evk, c, f) \to c_f$: This algorithm takes a ciphertext c, a circuit f and an evalaution key evk as inputs, and outputs a ciphertext $c_f \in \mathsf{C}$. The circuit f can be any fan-in-two arithmetic circuit consisting of additions and multiplications.

In this paper, we will omit pk, sk, evk if they are clear from the context. Depending on the support for a circuit in the Eval, HE schemes are broadly classified into bit-wise and word-wise categories. In bit-wise schemes, such as FHEW [DM15] and TFHE [CGGI17, CGGI20], the fundamental circuit involves logic gates. On the other hand, word-wise schemes, including BGV [BGV14], BFV [Bra12, FV12], and CKKS [CKKS17], primarily focus on basic operations like addition and multiplication. For brevity, we omit to describe Setup and KeyGen and only focus on basic algorithms. It is worth noting the existence of variants of operations, particularly in the forms of evk and Mult. Throughout this section, we denote sk, pk as a secret key and a public key, respectively, with N as the dimension of RLWE. Various distributions are distinguished by the subscript of χ.

The BGV/BFV scheme. BGV/BFV schemes have a vector of integers in $\mathbb{Z}_t$ as a message, which is encoded to the message space $\mathcal{R}_t$ for SIMD operations, for message modulus t. BGV and BFV

use almost the same mathematical structure, while BGV stores the message in the least significant bit and BFV stores the message in the most significant bit. The BGV scheme provides the following operations:

for a plaintext modulus t, and a chain of modulus $Q_0, \ldots, Q_L$,

- $\mathsf{Enc}(pk, m(X)) \to \mathtt{ct}$: for $m(X) \in \mathcal{R}_t$, return $\mathtt{ct} = v \cdot pk + (m(X) + t \cdot e_0, t \cdot e_1)$ where $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$.
- $\mathsf{Dec}(sk, \mathtt{ct}) \to m'(x)$: for $\mathtt{ct} = (b, a) \in \mathcal{R}_{Q_\ell}^2$, return $m(X) = [[b - a \cdot sk]_{Q_\ell}]_t$.
- $\mathsf{Add}(\mathtt{ct}_1, \mathtt{ct}_2) \to \mathtt{ct}_{add}$: for $\mathtt{ct}_1, \mathtt{ct}_2 \in \mathcal{R}_{Q_\ell}^2$, return $\mathtt{ct}_{add} = \mathtt{ct}_1 + \mathtt{ct}_2 \mod Q_\ell$.
- $\mathsf{Mult}(evk, \mathtt{ct}_1, \mathtt{ct}_2) \to \mathtt{ct}_{mult}$: for $\mathtt{ct}_1 = (b_1, a_1), \mathtt{ct}_2 = (b_2, a_2) \in \mathcal{R}_{Q_\ell}^2$, return $\mathtt{ct}_{mult} = (d_0, d_1) + [d_2 \cdot evk]_{Q_\ell}$ where $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \mod Q_\ell$.
- $\mathsf{ModSwitch}(\ell, \ell', \mathtt{ct}) \to \mathtt{ct}'$: for $\mathtt{ct} \in \mathcal{R}_{Q_\ell}^2$ at level ℓ, return $\mathtt{ct}' = \lfloor \frac{Q_{\ell'}}{Q_\ell} \cdot \mathtt{ct} \rceil \mod Q_{\ell'}$.

The BFV scheme provides the following operations:

for $\Delta = \lfloor Q/t \rfloor$ with a plaintext modulus t and a ciphertext modulus Q,

- $\mathsf{Enc}(pk, \Delta, m(X)) \to \mathtt{ct}$: for $m(X) \in \mathcal{R}_t$, return $\mathtt{ct} = v \cdot pk + (\Delta m(X) + e_0, e_1)$ where $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$.
- $\mathsf{Dec}(sk, \mathtt{ct}) \to m'(x)$: for $\mathtt{ct} = (b, a) \in \mathcal{R}_{Q_\ell}^2$, return $m(X) = \left\lfloor \frac{t}{Q} [b - a \cdot sk]_Q \right\rceil_t$.
- $\mathsf{Add}(\mathtt{ct}_1, \mathtt{ct}_2) \to \mathtt{ct}_{add}$: for $\mathtt{ct}_1, \mathtt{ct}_2 \in \mathcal{R}_Q^2$, set $\mathtt{ct}_{add} = \mathtt{ct}_1 + \mathtt{ct}_2 \mod Q$.
- $\mathsf{Mult}(evk, \mathtt{ct}_1, \mathtt{ct}_2) \to \mathtt{ct}_{mult}$: for $\mathtt{ct}_1 = (b_1, a_1), \mathtt{ct}_2 = (b_2, a_2) \in \mathcal{R}_{Q_\ell}^2$, return $\mathtt{ct}_{mult} = (d_0, d_1) + [d_2 \cdot evk]_Q$ where $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \mod Q$.

The CKKS scheme. The CKKS scheme has a vector of $\mathbb{C}$ as a message, and using the canonical embedding, it is mapped to the plaintext space as $\mathcal{R}$. The CKKS scheme provides the following operations:

for a factor Δ, a chain of modulus $Q_0, \cdots, Q_L$, and an auxiliary modulus P,

- $\mathsf{Enc}(pk, m(X)) \to \mathtt{ct}$: for $m(X) \in \mathcal{R}$, return $\mathtt{ct} = v \cdot pk + (m(X) + e_0, e_1)$ where $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi$.
- $\mathsf{Dec}(sk, \mathtt{ct}) \to m'(x)$: for $\mathtt{ct} = (b, a) \in \mathcal{R}_{Q_\ell}^2$, return $m(X) = b - a \cdot sk \mod Q_\ell$.
- $\mathsf{Add}(\mathtt{ct}_1, \mathtt{ct}_2) \to \mathtt{ct}_{add}$: for $\mathtt{ct}_1, \mathtt{ct}_2 \in \mathcal{R}_{Q_\ell}^2$, set $\mathtt{ct}_{add} = \mathtt{ct}_1 + \mathtt{ct}_2 \mod Q_\ell$.
- $\mathsf{Mult}(evk, \mathtt{ct}_1, \mathtt{ct}_2) \to \mathtt{ct}_{mult}$: for $\mathtt{ct}_1 = (b_1, a_1), \mathtt{ct}_2 = (b_2, a_2) \in \mathcal{R}_{Q_\ell}^2$, return $\mathtt{ct}_{mult} = (d_0, d_1) + \lfloor P^{-1} \cdot d_2 \cdot evk \rceil \mod Q_\ell$ where $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \mod Q_\ell$.
- $\mathsf{RS}(\ell, \ell', \mathtt{ct}) \to \mathtt{ct}'$: for $\mathtt{ct} \in \mathcal{R}_{Q_\ell}^2$ at level ℓ, set $\mathtt{ct}' = \lfloor P^{\ell' - \ell} \cdot \mathtt{ct} \rceil \mod Q_{\ell'}$.

The DM-like HE schemes. DM-like HE scheme performs LUTs on every operation, so-called a functional bootstrapping. The DM-like scheme provides the following operations:

for LWE dimension n and LWE and RLWE ciphertext modulus q and Q,

- $\mathsf{KeyGen}(\lambda) \to (sk, evk)$: return a secret key, public key, and evaluation keys that consists of the bootstrapping key brk and key switching key ksk. The constants d and B are associated with performance. The bootstrapping key brk generation varies across different bootstrapping methods.
  - return $s, z \leftarrow \chi_{key}$, where $s \in \mathbb{Z}^n$ and $z \in \mathcal{R}$.
  - return $brk = \{brk_i = (\{\mathsf{RLWE}_z(g_k \cdot s_i)\}_k, \{\mathsf{RLWE}_z(g_k \cdot zs_i)\}_k)\}_i \in \mathcal{R}_Q^{2dn}$.
  - return $ksk = \{\mathsf{LWE}_s\left(B^j z_i\right)\}_{i,j}$.
- $\mathsf{Enc}(pk, m) \to \mathtt{ct}$: for $m \in \{0,1\}^t$, return $\mathtt{ct} = \mathsf{LWE}_z(m \cdot q/t)$.
- $\mathsf{Dec}(sk, \mathtt{ct}) \to m'$: for $\mathtt{ct} \in \mathbb{Z}_q^{n+1}$, return $m = \lfloor (b - \langle a, s\rangle) \cdot t/q \rceil$.
- $\mathsf{Boot}(\mathtt{ct}, evk) \to \mathtt{ct}'$: find $\mathsf{RLWE}_z\left(f \cdot X^{\mathsf{Dec}(sk,\mathtt{ct})}\right)$ using evk, extract its free coefficient, and key switch to s. In this algorithm, f is a polynomial in $\mathcal{R}_Q$ representing a LUT.

3 Look-up Table Evaluation with Approximate HE

Our goal is to execute LUT operations effectively through homomorphic encryption. We initially concentrate on the CKKS scheme [CKKS17], chosen for its straightforward matrix operations within the complex domain. Subsequently, we extend our approach to the BGV [BGV14] and BFV [Bra12, FV12] schemes, detailed in Section 5.

3.1 Basic Instance: an ℓ-to-ℓ LUT over CKKS

We denote a ℓ-to-ℓ LUT as a function $T_{\ell\to\ell} : \{0,1\}^\ell \to \{0,1\}^\ell$. We introduce the encoding algorithm encode to map binary strings to complex numbers, and conversely, the decoding algorithm decode to revert a complex number to a binary string. The encoding function encode is a bijection that correlates each binary sequence of length ℓ with a distinct codeword in $\mathcal{C}$, a subset of $\mathbb{C}$ with cardinality 2^ℓ. However, due to CKKS's inherent noise, the output may not precisely fall within $\mathcal{C}$. Therefore, decode aims to identify the nearest codeword, effectively inverting encode:

$$\mathsf{encode} : \{0,1\}^\ell \to \mathcal{C}$$
$$\mathsf{decode} : \mathbb{C} \to \{0,1\}^\ell$$

Importantly, $\mathcal{C}$ can be chosen such that decoding is feasible in $O(1)$ time, enhancing efficiency.

Concrete construction of codes. We choose the code as a set of primitive n-th roots of unity, where $n = 2^\ell$. In particular, we choose $\mathcal{C} = \{\zeta^j\}_{j=0}^{n-1}, \zeta = \exp(\frac{-2\pi i}{n})$. Thus, the encoding is defined by

$$\mathsf{encode}(\vec{x}) = \zeta^{\mathsf{int}(\vec{x})},$$

where $\mathsf{int}(\cdot)$ maps ℓ-bit string to an integer in $[0, n)$. The decoding algorithm simply finds the nearest point in $\mathcal{C}$ and performs the inverse of the exponent.

It is important to highlight that the applicability of the proposed construction is not confined to the particular choice of the code presented herein; it can applied to any arbitrary fixed-size set. Nevertheless, the adoption of the above code $\mathcal{C}$ in our scheme notably enhances computational efficiency for reasons that will be detailed in Section 3.4.

For the sake of brevity, this discussion is limited to cases where $T = \{0,1\}^\ell$, meaning that $|T|$ is a power of two, and thus, T corresponds to a sequence of bits. It is important to note, however, that the proposed scheme is versatile and can be adapted to any finite T, eliminating the requirement for a power-of-two computational setting.

LUT as an interpolation polynomial. Considering that both the input and output are constrained to a specific code, it is feasible to construct an interpolating polynomial of degree $2^\ell - 1$, which accurately represents the input-output relationship of the transformation $T_{\ell\to\ell}$. Consequently, this transformation can be implemented within the CKKS framework via the aforementioned polynomial. The merit of maintaining identical domains and ranges for the LUT lies in the facilitation of consecutive LUT operations, thereby obviating the need for auxiliary functions.

The interpolation polynomial in the complex domain is expressed as a polynomial of degree $n-1$:

$$f(u) = \sum_{i=0}^{n-1} a_i u^i,$$

where the coefficients a_i are elements of the complex numbers $\mathbb{C}$. Although various methods exist for interpolation, in this context, the polynomial can be efficiently determined by solving a system of linear equations:

$$\begin{bmatrix} a_0 & \dots & a_{n-1} \end{bmatrix} \cdot U = \begin{bmatrix} d_0 & \dots & d_{n-1} \end{bmatrix}, \tag{1}$$

where the basis matrix U, corresponding to code $\mathcal{C} = \{c_0, \dots, c_{n-1}\}$, is defined as:

$$U = \begin{bmatrix} c_0^0 & c_1^0 & \dots & c_{n-1}^0 \\ c_0^1 & c_1^1 & & c_{n-1}^1 \\ & & \ddots & \\ c_0^{n-1} & c_1^{n-1} & \dots & c_{n-1}^{n-1} \end{bmatrix}, \tag{2}$$

with each $d_i = f(c_i)$ representing the polynomial evaluated at c_i. Notably, each d_i also constitutes a codeword in $\mathcal{C}$, and $\mathsf{decode}(d_i)$ yields the result of LUT operation $T_{\ell\to\ell}(\mathsf{decode}(c_i))$, correlating the input c_i to its encoded output d_i. Once the coefficients are determined, the interpolation polynomial can be efficiently evaluated within the CKKS domain. Consequently, this facilitates the execution of LUT operations using CKKS schemes.

The inherent noise within the CKKS scheme leads to an accumulation of noise when evaluating the interpolation polynomial. Nonetheless, we demonstrate in a subsequent section that the increase in noise, consequent to the evaluation of LUT polynomials, is confined within a reasonably small bound. Furthermore, we introduce a noise reduction technique that effectively diminishes the noise present in the plaintext, leveraging the discrete characteristics of the code. A comprehensive proof, establishing that the proposed noise reduction polynomial eliminates the dominant noise term, is presented in Section 4.1.

Our approach can be naturally extended to the case where $T_{\ell\to\beta\ell} : \{0,1\}^\ell \to \{0,1\}^{\beta\ell}$ by splitting into the number of β small LUTs, denoted as $\{T^{(j)}_{\ell\to\ell} : \{0,1\}^\ell \to \{0,1\}^\ell\}_{j\in[0,\beta-1]}$. Each $T^{(j)}_{\ell\to\ell}$ denotes the j-th ℓ-bit segment of $T_{\ell\to\beta\ell}$, i.e., $T^{(j)}_{\ell\to\ell}(\vec{x}) = T_{\ell\to\beta\ell}(\vec{x})_{[j\ell,(j+1)\ell-1]}$, that is, each covers a subset of the overall computation and has the same domain and range. Then, for each $j \in [0, \beta-1]$, there is a distinct polynomial representing $T^{(j)}_{\ell\to\ell}$ and one can obtain a set of polynomials representing $T_{\ell\to\beta\ell}$. By evaluating each polynomial individually, $T_{\ell\to\beta\ell}$ can be computed.

Figure 1: Brief overview of our idea

In contrast to DM-like HE schemes as outlined in [DM15, CGGI17, LMK+23], our proposed approach eliminates the need for costly blind rotations. Furthermore, when compared to the recent contributions by Liu and Wang [LW23c], our method employs polynomials of significantly lower degree, enhancing efficiency.

However, the scenario at hand is not without its drawbacks. The polynomial degree's dependency on 2^ℓ poses a significant scalability challenge. Additionally, the scheme's capability is confined to single-input operations, requiring precomputational integration of multiple input bit strings into one for small LUT. Those are typical characteristics of LUT-based HE schemes. The next section will address these challenges by employing *multivariate polynomials*, which reduces the polynomial degree to $\alpha 2^\ell$ for $\alpha\ell$-bit input LUTs, thus offering a resolution to the scalability issue. Furthermore, this approach also introduces segmenting bit strings in HE ciphertexts, which enhances the efficiency of input assembly for LUT evaluations.

3.2 Extension to an $\alpha\ell$-to-$\beta\ell$ LUT

This section expands the look-up table (LUT) framework to include operations with multiple precision inputs and outputs. It covers both fundamental LUT functionalities and those involving multiple inputs. This broader scope is crucial for scenarios demanding high precision across various inputs and outputs, significantly enhancing the framework's versatility and application range.

As opposed to the previous work, we employ a multivariate polynomial in $\mathbb{C}$. Due to multivariate polynomials, We can use multiple inputs with small disadvantage in noise only. Consider $T_{\alpha\ell\to\beta\ell} : \{0,1\}^{\alpha\ell} \to \{0,1\}^{\beta\ell}$ for $\alpha, \beta \in \mathbb{N}$ as a LUT. Our proposal starts by partitioning $T_{\alpha\ell\to\beta\ell}$ into a set of LUT $\left\{T^{(j)}_{\alpha\ell\to\ell} : \{0,1\}^{\alpha\ell} \to \{0,1\}^{\ell}\right\}_{j\in[0,\beta)}$ such that $T^{(j)}_{\alpha\ell\to\ell}(\vec{x}) = T_{\alpha\ell\to\beta\ell}(\vec{x})_{[j\ell,(j+1)\ell-1]}$. To handle large input efficiently, we employ a multivariate polynomial $f_j : \mathcal{C}^\alpha \to \mathcal{C}$ such that

$$f_j(u_0, \ldots, u_{\alpha-1}) = v_j,$$

where $(\vec{x}, T^{(j)}_{\alpha\ell\to\ell}(\vec{x})) \in \{0,1\}^{\alpha\ell} \times \{0,1\}^{\ell}$, $u_k = \mathsf{encode}(\vec{x}_{[k\ell,(k+1)\ell-1]})$, and $v_j = \mathsf{encode}(T^{(j)}_{\alpha\ell\to\ell}(\vec{x}))$ for $k \in [0, \alpha-1]$. A set of polynomials collectively represents $T_{\alpha\ell\to\beta\ell}$, enabling its computation using CKKS scheme. The remaining step involves demonstrating the existence and algorithm to deteremine a set of multivariate polynomials for any $T_{\alpha\ell\to\beta\ell}$.

Existence of multivariate polynomials. For any $j \in [0, \beta-1]$, f_j can be expressed as a combination of powers of u_i's and coefficients. Since determining f_j is equivalent to determine a coefficient along with a set of exponents, define $C_j \in \mathbb{C}^{n^\alpha}$ whose entries represent the coefficients

of f_j and a matrix $D_j \in \mathcal{C}^\beta$ whose entries correspond to evaluation points of $f_j(u_0, \ldots, u_{\alpha-1})$ for all possible inputs. It leads to the following equation:

$$f_j(u_0, \ldots, u_{\alpha-1}) = \sum_{i_0=0}^{n-1} \cdots \sum_{i_{\alpha-1}=0}^{n-1} C_j[i_0, \cdots, i_{\alpha-1}] \cdot u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}}.$$

To describe the process of determining C_j, we introduce some helpful operations.

Definition 1. *For $k, \alpha \in \mathbb{N}$ with $k < \alpha$, let $C \in \mathbb{C}^{\prod_{i=0}^{\alpha-1} n_i}$ be a multi-dimensional matrix and a vector $\vec{u} \in \mathbb{C}^{n_k}$. A function*

$$\boxdot_k : \mathbb{C}^{\prod_{i=0}^{\alpha-1} n_i} \times \mathbb{C}^{n_k} \longrightarrow \mathbb{C}^{\prod_{i=0}^{\alpha-1} n_i/n_k}$$

is defined by $C \boxdot_k \vec{u} = B$, where

$$B[i_0, \cdots, i_{k-1}, i_{k+1}, \cdots, i_{\alpha-1}] = \sum_{j=0}^{n-1} C[i_0, \cdots, i_{k-1}, j, i_{k+1}, \cdots i_{\alpha-1}] \cdot u[j] \tag{3}$$

We call $\boxdot_k$ is said to be a vector multiplication in k-axis.

Definition 1 can be extended to define a new operator for the matrix multiplication between a multi-dimensional matrix and a matrix.

Definition 2. *For $k, \alpha \in \mathbb{N}$ with $k < \alpha$, let $U \in \mathbb{C}^{n_k \times n_k}$. A function*

$$\boxtimes_k : \mathbb{C}^{\prod_{i=0}^{\alpha-1} n_i} \times \mathbb{C}^{n_k \times n_k} \longrightarrow \mathbb{C}^{\prod_{i=0}^{\alpha-1} n_i}$$

is defined by $C \boxtimes_k U = D$, where

$$D[i_0, \cdots, i_{\alpha-1}] = \sum_{j=0}^{n-1} C[i_0, \cdots, i_{k-1}, j, i_{k+1}, \cdots, i_{\alpha-1}] \cdot U[j, i_k] \tag{4}$$

We call $\boxtimes_k$ is said to be a matrix multiplication in k-axis.

In similar to matrix multiplication, performing a matrix multiplication in the k-axis between A and its inverse matrix results in an identity matrix for all k.

Lemma 1. *For $k, \alpha \in \mathbb{N}$ with $k < \alpha$, let $C \in \mathbb{C}^{\prod_{i=0}^{\alpha-1} n_i}$ and $U \in \mathbb{C}^{n_k \times n_k}$ C nonsingular matrix. We have $C \boxtimes_k U \boxtimes_k U^{-1} = A$, where U^{-1} is an inverse matrix of U.*

Proof. By Definition 2, we have

$$\begin{aligned} C \boxtimes_k U \boxtimes_k U^{-1} &= \left(\sum_{j=0}^{n-1} C[i_0, \ldots, i_{k-1}, j, i_{k+1}, \cdots, i_{b-1}] \cdot U[j, i_k] \right) \boxtimes_k U^{-1} \\ &= \sum_{r=0}^{n-1} \sum_{j=0}^{n-1} C[i_0, \ldots, i_{k-1}, j, i_{k+1}, \cdots, i_{b-1}] \cdot U[j, r] \cdot U^{-1}[r, i_k] \end{aligned}$$

Since

$$U[j,r]\cdot U^{-1}[r,i_k]=\begin{cases}1 & \text{if } j=i_k\\ 0 & \text{otherwise}\end{cases}$$

due to the definition of U^{-1}, it is clear that $C\boxtimes_k U\boxtimes_k U^{-1}=C$. □

We denote $\vec{u}_i=(1,u_i,u_i^2,\ldots,u_i^{n-1})$. The evaluation of multivariate polynomial f_j can be represented using a vector multiplication in all axis defined in Equation (4):

$$f_j(u_0,u_1,\ldots,u_{\alpha-1})=C_j\boxdot_0\vec{u}_0\boxdot_1\vec{u}_1\cdots\boxdot_{\alpha-1}\vec{u}_{\alpha-1}$$

Note that $|\mathcal{C}|=n$ and $(u_0,u_1,\ldots,u_{\alpha-1})$ has n^α possible combinations. As in Equation (2), we define a basis matrix for code $\mathcal{C}=\{c_0,\ldots,c_{n-1}\}$. In the general context, we denote an element of $\mathcal{C}$ as c_j and define a matrix U employing these elements c_j. Specifically, c_j corresponds to the j-th power of $\zeta=\exp\left(-\frac{2\pi i}{n}\right)$ throughout this paper. The output matrix D can be represented by consecutive matrix multiplication in all axis of U to C. In other words,

$$D=C\boxtimes_0 U\boxtimes_1 U\cdots\boxtimes_{\alpha-1}U. \tag{5}$$

By **??** 1, we can have

$$C=D\boxtimes_{\alpha-1}U^{-1}\boxtimes_{\alpha-2}U^{-1}\cdots\boxtimes_0 U^{-1}. \tag{6}$$

Therefore, it takes $O(\alpha n^2)$ time to find C for any given LUT. There are β such polynomials, so the total time complexity to find is $O(\alpha\beta n^2)$, which is exactly the size of the given table.

We can see that in Equation (5), we set the maximum degree of each u_i to n, rather than set the maximum degree of the multivariate polynomial f. To reduce the multiplicative depth, it would be better to fix the maximum degree of the multivariate polynomial in general. However, it is worth noting that in our choice of the code $\mathcal{C}$, there exists an overlap in high-order and low-order basis, i.e., $u^n=1$, so we set the maximum degree of u_i for each i to $n-1$. Moreover, we can reduce the degree by half using conjugate as will be mentioned later (in Section 3.4), and thus, we can reduce the depth.

3.3 Error Analysis

The goal of this section is analyzing error for our proposed approach.

Proposition 1. *Given $k\in[0,\alpha-1]$ and a multi-dimensional matrix $D\in\mathbb{C}^{n^\alpha}$, each element of $C=D\boxtimes_k U^{-1}$, where $U\in\mathbb{C}^{n\times n}$ is defined as above, satisfies*

$$\|C\|_\infty\le\|D\|_\infty$$

Proof. Remark that for $\zeta=\exp(-2\pi i/2^\ell)$, a matrix U is a discrete Fourier transform matrix and thus there always exists an inverse matrix of U. By Definition 2, we can derive that

$$\begin{aligned}\|C\|_\infty&=\left\|\sum_{j=0}^{n_k-1}D[i_0,\cdots,i_{k-1},j,i_{k+1},\cdots,i_{\alpha-1}]\cdot U^{-1}[j,i_k]\right\|_\infty\\&\le\|D\|_\infty\cdot\left\|\sum_{j=0}^{n_k-1}U^{-1}[j,i_k]\right\|\le\|D\|_\infty.\end{aligned}$$

The last inequality is derived from the fact that $U^{-1}[j, i_k] \in \mathcal{C}$, and since any norm of an element in $\mathcal{C}$ is one. □

By iteratively applying Proposition 1 to Equation (6) for all potential values of k, we establish that the magnitude of each element in C is smaller than that of D. This also implies that the noise associated with any multivariate polynomial remains manageable. As a result, by selecting a suitable scaling factor, it becomes feasible to compute the multivariate interpolation polynomial corresponding to a LUT. Additionally, this framework allows for the evaluation of multiple consecutive LUTs, provided that noise within the ciphertext can be effectively minimized, a concept further elaborated in Section 4.2.

Theorem 1. *Let $f : \mathcal{C}^\alpha \to \mathbb{C}$ be a mutlivariate polynomial whose coefficient multi-dimensional matrix is $C = D \boxtimes_{\alpha-1} U^{-1} \boxtimes_{\alpha-2} U^{-1} \cdots \boxtimes_0 U^{-1} \in \mathbb{C}^{n^\alpha}$. If $\|\epsilon_i\| \ll 1$, it satisfies that*

$$\|f(u_0 + \epsilon_0, \ldots, u_{\alpha-1} + \epsilon_{\alpha-1}) - f(u_0, \ldots, u_{\alpha-1})\| \approx O\left(\epsilon \|D\|_\infty n^\alpha\right)$$

for some constant c where $\epsilon = \|(\epsilon_0, \ldots, \epsilon_{\alpha-1})\|_\infty$.

Proof. By definition of a multivariate polynomial and Proposition 1,

$$\begin{aligned}
&\|f(u_0 + \epsilon_0, \cdots, u_{\alpha-1} + \epsilon_{\alpha-1}) - f(u_0, \ldots, u_{\alpha-1})\| \\
&= \left\| \sum_{i_0=0}^{n_0-1} \cdots \sum_{i_{\alpha-1}=0}^{n_{\alpha-1}-1} C[i_0, \cdots, i_{\alpha-1}] \cdot \left((u_0 + \epsilon_0)^{i_0} \cdots (u_{\alpha-1} + \epsilon_{\alpha-1})^{i_{\alpha-1}} - u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}} \right) \right\| \\
&\leq \|D\|_\infty \cdot \sum_{i_0=0}^{n_0-1} \cdots \sum_{i_{\alpha-1}=0}^{n_{\alpha-1}-1} \left\| \left((u_0 + \epsilon_0)^{i_0} \cdots (u_{\alpha-1} + \epsilon_{\alpha-1})^{i_{\alpha-1}} - u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}} \right) \right\|.
\end{aligned}$$

By Proposition 1, we can derive the inequality such that

$$\begin{aligned}
&\left\| \sum_{i_0=0}^{n_0-1} \cdots \sum_{i_{\alpha-1}=0}^{n_{\alpha-1}-1} C[i_0, \cdots, i_{\alpha-1}] \cdot \left((u_0 + \epsilon_0)^{i_0} \cdots (u_{\alpha-1} + \epsilon_{\alpha-1})^{i_{\alpha-1}} - u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}} \right) \right\| \\
&\leq \|D\|_\infty \cdot \sum_{i_0=0}^{n_0-1} \cdots \sum_{i_{\alpha-1}=0}^{n_{\alpha-1}-1} \left\| \left((u_0 + \epsilon_0)^{i_0} \cdots (u_{\alpha-1} + \epsilon_{\alpha-1})^{i_{\alpha-1}} - u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}} \right) \right\|.
\end{aligned}$$

Since $u_i \in \mathcal{C}$, we also have

$$\begin{aligned}
&\left\| \left((u_0 + \epsilon_0)^{i_0} \cdots (u_{\alpha-1} + \epsilon_{\alpha-1})^{i_{\alpha-1}} - u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}} \right) \right\| \\
&= \left\| u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}} \left((1 + \epsilon_0/u_0)^{i_0} \cdots (1 + \epsilon_{\alpha-1}/u_{\alpha-1})^{i_{\alpha-1}} - 1 \right) \right\| \\
&\leq \left\| u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}} \right\| \cdot \left\| \left((1 + \epsilon_0/u_0)^{i_0} \cdots (1 + \epsilon_{\alpha-1}/u_{\alpha-1})^{i_{\alpha-1}} - 1 \right) \right\| \\
&= \left\| (1 + \epsilon_0/u_0)^{i_0} \cdots (1 + \epsilon_{\alpha-1}/u_{\alpha-1})^{i_{\alpha-1}} - 1 \right\|
\end{aligned}$$

Substituting ϵ_i/u_i by t_i, there exists a set of constant c_j such that

$$\left\| \left((1 + t_0)^{i_0} \cdots (1 + t_{\alpha-1})^{i_{\alpha-1}} - 1 \right) \right\| \leq \sum_{j_0, \ldots, j_{\alpha-1}} \left\| c_{j_0, \cdots, j_{\alpha-1}} t_0^{j_0} \cdots t_{\alpha-1}^{j_{\alpha-1}} \right\|$$

due to triangle inequality. As $\|t_i\| \leq \|\epsilon_i\|$ and $\|\epsilon_i\| \ll 1$,

$$\begin{aligned}\sum_{j_0,\cdots,j_{\alpha-1}} \left\| c_{j_0,\cdots,j_{\alpha-1}} t_0^{j_0} \cdots t_{\alpha-1}^{j_{\alpha-1}} \right\| &\leq \max_{j_0,\cdots,j_{\alpha-1}} (\left\| c_{j_0,\cdots,j_{\alpha-1}} \right\|) \sum_{j_0,\cdots,j_{\alpha-1}} \|t_0\|^{j_0} \cdots \|t_{\alpha-1}\|^{j_{\alpha-1}} \\ &\leq \max_{j_0,\cdots,j_{\alpha-1}} (\left\| c_{j_0,\cdots,j_{\alpha-1}} \right\|) \sum_{j_0,\cdots,j_{\alpha-1}} \epsilon^{j_0+\cdots+j_{\alpha-1}} \\ &\leq c \cdot \epsilon\end{aligned}$$

for some constant c and $\epsilon = \|(\epsilon_0, \ldots, \epsilon_{\alpha-1})\|_\infty$. Finally, we can derive that

$$\|f(u_0 + \epsilon_0, \ldots, u_{\alpha-1} + \epsilon_{\alpha-1}) - f(u_0, \ldots, u_{\alpha-1})\| \approx O\left(\epsilon \|D\|_\infty \prod_{i=0}^{\alpha-1} n_i\right)$$

□

3.4 Choice of Code

The following are the reasons why we employ the code $\mathcal{C} = \{\zeta^j\}_{j=0}^{n-1}$, where $\zeta = \exp(-2\pi i/2^\ell)$, in the CKKS scheme.

Better signal-to-noise ratio in polynomial basis. There are two types of noise. First one the noise propagates along with operations, which proportional to the norm of message. The second one is the rescaling error, the fundamental additive noise. A power basis is not suitable for polynomial evaluation unless the degree is very small, as it exponentially diverges or converges to zero. When it diverges, the first kind of noise overhelms lower degree terms, and when it converges to zero, the noise overwhelms the message. In case of bootstrapping, a high degree approximate polynomial is used, and thus we use Chebyshev basis instaed of power basis and various techniques to control the polynomial basis not to diverge or converge [CCS19, LLK+22].

Reducing depth and runtime using conjugation. When we use $\mathcal{C} = \{\zeta^j\}_{j=0}^{n-1}$, it is always true that $x^{n-j} = \overline{x^j}$ for all $x \in \mathcal{C}$. Thus, we can find higher degree term simply by conjugation of lower degree therm. It is noteworthy that a multiplication consumes level of ciphertext, but the conjugation does not. Also, as the conjugtation is done by Galois operation, it only has additive key switching noise unlike ciphertext-ciphertext multiplication. It also has less computational complexity as it does not involves computation among ciphertexts.

Fast polynomial finding algorithm using fast Fourier transform. It is noticed that the matrix U is a transposed *Vandermonde matrix.* Especially, when $c_k = \zeta^k$, it is a discrete Fourier transform (DFT) matrix. Obviously its inverse is inverse DFT matrix $U^{-1} = \frac{1}{n}U^*$. Hence, when we find the coefficients in Equation (6), the multiplications $\boxtimes_k U^{-1}$ can be done by *inverse fast Fourier transform* (FFT) in k-axis by definition. Hence it takes $O(\alpha n \log n)$ times, to find C, rather than $O(\alpha n^2)$.

Application to the BGV/BFV scehems. As noted in previous paragraph, by selecting the proper code, we can let U be a DFT matrix. Interesting fact here is that we now can extend the proposed algorithm to the BGV/BFV schemes. Let t be a message modulus in BGV/BFV schemes, where the n-th root of unity ζ exists in $\mathbb{Z}_t$. We extend the definitions of *vector multiplication in*

(a) Naive code (b) Proposed code

Figure 2: Comparing naive code (left) and proposed code (right) for the same noise.

k-axis and *matrix multiplication in k-axis* in Definition 1 and Definition 2, respectively, to $\mathbb{Z}_t$. Then, the matrix

$$U = \begin{bmatrix} 1 & \zeta^{1\cdot 0} & \dots & \zeta^{(n-1)\cdot 0} \\ 1 & \zeta^{1\cdot 1} & & \zeta^{(n-1)\cdot 1} \\ & & \ddots & \\ 1 & \zeta^{1\cdot (n-1)} & \dots & \zeta^{(n-1)\cdot (n-1)} \end{bmatrix} \in \mathbb{Z}_t^{n\times n}$$

is an transposed number theoretic transform (NTT) matrix, and its inverse is inverse NTT matrix.

Noise bound in LUT polynomial evaluation. While polynomial evaluation in CKKS, the noise in message is multiplied to the noise, and thus a polynomial with small coefficient is preferred in terms of noise growth [LLK+22]. It is noted that finding polynomial in Equation (6) is done by inverse Fourier transform, which is a *norm-preserving* operation. Each element of D is also codeword, and thus its absolute value is one. Hence, each coefficient in C is small, and the noise after the polynomial evaluation just proportional to number of elements.

Less decoding failure probability. Our code is chosen n equally distanced points in unit circle in $\mathbb{C}$. A naive solution can be put n real numbers in $[-1, 1]$, as noted in above paragraph, increasing the range to larger domain $[-A, A], A > 1$ do not help in SNR. Now calculate the failure probability, i.e., $\mathsf{encode}(i)$ is decoded to other numbers due to the noise. Figure 2 shows a naive selection of the code and the proposed method for Gaussian noise of same variance.

Let the failure probability for each case P_{naive} and $P_{proposed}$, and the random variable e represents noise in CKKS message, which can be considered as a additive *complex* Gaussian noise. In the first case, the noise in imaginary part do not affect the decoding failure, thus we have $P_{naive} = Pr\left[|Real(e)| > \frac{1}{(n-1)}\right]$. As in [DM15], the failure probability can be estimated as $P_{naive} = 1 - \mathrm{erf}\left(\frac{1}{(n-1)\cdot\sqrt{2}\sigma}\right)$, where σ is the standard deviation of e. In our case, the noise happens when the noise is outsize the circular sector of central angle $2\pi/n$. We can find a loose upper bound of the noise $P_{proposed} < Pr\left[|e| > \sin(\frac{\pi}{n})\right] = \exp(-\sin^2(\frac{\pi}{n})/(2\sigma^2))$, as CDF of the Rayleigh distribution is $1 - \exp(-x^2/(2\sigma^2))$. For example, when we have noise of $\sigma = \frac{1}{128}$ and $n = 16$, we have $P_{naive} = 2^{-56.0}$ and $P_{proposed} < 2^{-449.8}$.

Simple and fast noise-reduction polynomials. As described later in Section 4.1, our code has simple noise-reduction polynomial of form $-\frac{1}{n}x^{n+1} + \frac{n+1}{n}x$. It can be evaluated only using $\log n + 1$ ciphertext-ciphertext multiplications.

4 Advanced Operations for Efficiency

4.1 Noise Reduction Algorithm

The core concept of our approach lies in utilizing a polynomial function, $f : \mathbb{C} \to \mathbb{C}$, specifically designed to play a pivotal role in noise mitigation. This becomes especially critical when dealing with operations within an *unbounded* arithmetic circuit over complex numbers, where errors inherently tend to accumulate with each gate operation in ciphertext processing. We refer to such a function f as an error-reduction function. Formally, we can define an error-reduction function as follows.

Definition 3. *Let $f : \mathbb{C} \to \mathbb{C}$ be a function and $\mathcal{C} = \{\zeta^j\}_{j=0}^{n-1}$, where $\zeta = \exp(-2\pi i/2^\ell)$. We say f is a noise-reduction function if the following two properties are hold: for $t \in \mathcal{C}$ and $\|\epsilon\| \leq \delta \in \mathbb{R}$,*

- $f(t) = t$
- $\|f(t+\epsilon) - f(t)\| < \|\epsilon\|$

The first condition asserts that a message without error remains unchanged after applying the function f. The second condition indicates, for a small error ϵ, the function f should alleviate the error. This dual criterion characterizes the essential features of a noise-reduction function, emphasizing both the preservation of the original message and the mitigation of errors through the function f.

We propose a valuable proposition to facilitate the construction of such a function. This proposition signifies that when one can ascertain a function f that meets the condition $f(t) = t$ for $t \in X$ and concurrently satisfies a coefficient of ϵ in $f(t+\epsilon)$ is zero, then this function f can be considered as a valid error-reduction function.

Theorem 2. *For $t \in \mathcal{C}$, let f be a polynomial of ϵ, and $f(t+\epsilon) = \sum_{i=0}^{d-1} c_i \epsilon^i$. If $c_1 = 0$, then*

$$\|f(t+\epsilon) - f(t)\| < \|\epsilon\|$$

where $\|\epsilon\| \leq 1/(1+C)$ and $C = \max\{\|c_2\|, \ldots, \|c_{n-1}\|\}$.

Proof. Note that a constant term of $f(t+\epsilon)$ is $c_0 = f(t)$. Suppose $c_1 = 0$. Then,

$$\|f(t+\epsilon) - f(t)\| = \left\| \sum_{i=2}^{n-1} c_i \epsilon^i \right\| \leq C \sum_{i=2}^{n-1} \|\epsilon\|^i < C \cdot \frac{\|\epsilon\|^2}{1 - \|\epsilon\|}$$

Since $\|\epsilon\| \leq \frac{1}{1+C}$,

$$C \cdot \frac{\|\epsilon\|^2}{1 - \|\epsilon\|} \leq C \cdot \frac{1+C}{C} \cdot \|\epsilon\|^2 \leq \|\epsilon\|$$

□

This theorem implies that there exists a function polynomial f that makes the noise asymptotically smaller. Given ϵ, we know that $\|f(t+\epsilon) - f(t)\| = \mathsf{poly}(\epsilon)$. When $\epsilon \ll 1$, the first-order term dominates. It has been shown that we can eliminate the first order term and make the noise asumptotically less than or equal to ϵ^2.

We can identify a specific degree d that not only guarantees the existence of function f but also allows us to establish its uniqueness. Denote $f(x) = \sum_{i=0}^{d-1} a_i x^i$ and $c_1 = a_1 + 2a_2 x + \cdots (d-1)a_{d-1}x^{d-2}$. To satisfy both $f(t) = t$ and $c_1 = 0$, we formulate a comprehensive system of equations:

$$\begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{d-1} \\ & & \vdots & & \\ 1 & t_n & t_n^2 & \cdots & t_n^{d-1} \\ 0 & 1 & 2t_1 & \cdots & (d-1)t_1^{d-2} \\ & & \vdots & & \\ 0 & 1 & 2t_n & \cdots & (d-1)t_n^{d-2} \end{bmatrix} \begin{bmatrix} a_0 \\ \\ \vdots \\ \\ a_{d-1} \end{bmatrix} = \begin{bmatrix} t_1 \\ \vdots \\ t_n \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Let A be a leftmost matrix with dimensions $2n \times d$. It is evident that we can ascertain the coefficients of $f(x)$ whenver d is equal to the rank of A. Fortunately, A is non-singular, ensuring the existence of f when $d = 2n$. Moreover, the matrix has an inverse, implying that $f(x)$ can be uniquely determined through this system of equations. As we mentioned above, we assume that $c_1 = 0$, but it can be also generalized to arbitrary index. Specifically, in our implementation, we demonstrate that for $x = t + \epsilon$,

$$f(t) = -\frac{1}{n}t^{n+1} + \left(1 + \frac{1}{n}\right)t$$

serves as one of the concrete constructions for the noise-reduction function.

4.2 High Precision Look-up Table in Batched Manner

Interstingly, our approach allows the concurrent evaluation of distinct LUTs for individual message slots, all while maintaining computational cost. For instance, consider a ciphertext $\mathsf{Enc}(m_0, m_1)$ underlying message is (m_0, m_1) and sets of multivariate polynomials $\{f_j\}_{j\in[0,\beta-1]}$ and $\{h_j\}_{j\in[0,\beta-1]}$ representing LUT H and T, respectively. Applying our approach, one can obtain a ciphertext of $\{(f_j(m_0), h_j(m_1))\}_{j\in[0,\beta-1]}$, thereby representing a set of ciphertexts indicating $H(m_0)$ and $T(m_1)$.

The naïve approach is that independently executing $\mathtt{ct}_{1,j} = f_j(\mathsf{Enc}(m_0, m_1))$ and $\mathtt{ct}_{2,j} = h_j(\mathsf{Enc}(m_0, m_1))$ for all j. The desired result can be obtained by computing $(1,0)\cdot\mathtt{ct}_{1,j} + (0,1)\cdot\mathtt{ct}_{2,j}$. However, its cost totally depends on the number of executed LUTs, leading to inefficiency when the number of LUTs is large.

The fundamental equation involves that, given $f(x) = \sum_{i=0}^{n-1} a_i x^i$ and $g(x) = \sum_{i=0}^{n-1} b_i x^i$,

$$(f(m_0), g(m_1)) = \left(\sum_{i=0}^{n-1} a_i m_0^i, \sum_{i=0}^{n-1} b_i m_1^i\right) = \sum_{i=0}^{n-1} (a_i, b_i) \circ (m_0^i, m_1^i) \tag{7}$$

where $\circ$ denotes the Hadamard product. Applying HE to Equation (7), we straightforwardly obtain that

$$\mathsf{Enc}(f(m_0), g(m_1)) = \sum_{i=0}^{n-1} \mathsf{encode}(a_i, b_i) \cdot \mathsf{Enc}(m_0, m_1)^i \tag{8}$$

The computation cost is that it takes only $O(\log n)$ squaring ciphertext and $O(\log n)$ ciphertext multiplications, together with $O(n)$ scalar multiplications, which is comparable to that of a single LUT evaluation. Moreover, Equation (8) can be generalized to support multiple message slots and extended to multivariate polynomials.

The brief construction of our apporach is described as follows. For the simplicity, we assume that all LUTs have the consistent input and output dimensions. Let m denote the number of LUTs, each associated with distinct a set of multivariate polynomials of degree n. Let $\mathtt{ct}$ represent a ciphertext with an underlying message serving as an input. Depending on the application, there could be multiple ciphertexts involved. The operation $\mathtt{ct}^i$ is executed leveraging the square-and-multiply technique. To facilitate the desired output, a fresh vector $\vec{v}_{i_0,\ldots,i_{\alpha-1}}$ of length n is generated for $i_0,\ldots,i_{\alpha-1} \in [0, n-1]$. Each entry of $\vec{v}_{i_0,\ldots,i_{\alpha-1}}$ is derived from the coefficients of the corresponding multivariate polynomial at the degree of $u_0^{i_0}\cdots u_{\alpha-1}^{\alpha-1}$. The methodology for selecting these coefficients depends on the targeted LUTs. Then, a new multivariate polynomial h_j can be generated through $\mathsf{encode}(\vec{v}_{i_0,\ldots,i_{\alpha-1}})$, that is,

$$h_j(u_0,\ldots,u_{\alpha-1}) = \sum_{i_0=0}^{n-1}\cdots\sum_{i_{\alpha-1}=0}^{n-1} \mathsf{encode}(\vec{v}_{i_0,\ldots,i_{\alpha-1}})\cdot u_0^{i_0}\cdots u_{\alpha-1}^{i_{\alpha-1}}.$$

Consequently, the desired outcome is achieved by evaluating h_j with precomputed $\mathtt{ct}^i$. To establish the comparability of noise upper bounds for this approach, we introduce an useful proposition.

Proposition 2. *Let m represent the number of LUTs, each associated with a distinct multivariate polynomial set $\{f_{i,j}\}_{i\in[0,m-1],j\in[0,\alpha-1]}$. Let $\{h_j\}$ be a set of multivarate polynomials generated by the our proposal. Then, for all j, there exists $c_j \in \mathcal{C}$ such that*

$$h_j(u_0,\ldots,u_{\alpha-1}) = \sum_{i=0}^{m-1} c_j f_{i,j}(u_0,\ldots,u_{\alpha-1})$$

where $u_i \in \mathcal{C}$.

Proof. By the construction of h_j, each coefficient is $\mathsf{encode}(\vec{v}_{i_0,\ldots,i_{\alpha-1}})$, where each entry of $\vec{v}_{i_0,\ldots,i_{\alpha-1}}$ represents a coefficient of $u_0^{i_0}\cdots u_{\alpha-1}^{\alpha-1}$ in $f_{i,j}$. It is entirely deteremined by fixed target evaluation function. Consequently, there exists a binary vector $\vec{b}_j$ of length n that indicates a target and then $\vec{v}_{i_0,\ldots,i_{\alpha-1}}$ can be represented as a Hadamard product of a coefficient vector of $f_{i,j}$ and $\vec{b}_j$ for all j. Since $\mathcal{C}$ is closed under multiplication, $c_j = \mathsf{encode}(\vec{b}_j)$. □

Due to Proposition 2, it is evident that the norm of h_j is equivalent to that of $f_{i,j}$ since the absolute value of c_j is one. This implies that the noise upper bound of the evaluation of h_j is comparable to that of the evaluation of $f_{i,j}$ and, consequently, similar to that of a single LUT.

Theorem 1 implies that a noise upper bound of any given multivariate polynomial is not huge. Therefore, we can find and set the corresponding scaling factor to evaluate the multivariate interpolation polynomial for a LUT. Furthermore, we can evaluate more than one LUTs if we can reduce reduce noise in ciphertext, which is to be described in the following subsection.

5 Look-up Table Evaluation with BGV and BFV

Building on the foundational discussion, it's important to note that our proposed technique is not confined to the CKKS scheme but is also compatible with BGV and BFV schemes. In the following, we demonstrate the implementation of the proposed method within the BGV and BFV contexts, as initially described in Section 3.

Our approach begins by decomposing a $\alpha\ell$-to$\beta\ell$ LUT, $T_{\alpha\ell\to\beta\ell}$ into smaller $\alpha\ell$-toβ LUTs denoted as $\left\{T^{(j)}_{\alpha\ell\to\ell} : \{0,1\}^{\alpha\ell} \to \{0,1\}^{\ell}\right\}_{j\in[0,\beta)}$, where $T^{(j)}_{\alpha\ell\to\ell}(\vec{x})$ extracts the jth ℓ-bit segment from $T_{\alpha\ell\to\beta\ell}$. To process inputs effectively, we introduce a multivariate polynomial $f_j : \mathcal{C}^{\alpha} \to \mathcal{C}$ such that

$$f_j(u_0, \ldots, u_{\alpha-1}) = v_j,$$

where for each $k \in [0, \alpha-1]$, $u_k = \mathsf{encode}(\vec{x}_{[k\ell,(k+1)\ell-1]})$ corresponds to the encoded input segments and $v_j = \mathsf{encode}(T^{(j)}_{\alpha\ell\to\ell}(\vec{x}))$ represents the encoded output of $T^{(j)}$. These polynomials, in aggregation, represent $T_{\alpha\ell\to\beta\ell}$, facilitating its execution within the BGV/BFV framework. A key adaptation in our scheme is that the code $\mathcal{C}$ is a subset of $\mathbb{Z}_t$, not $\mathbb{C}$, and the function encode is now a bijection from ℓ-bit strings to $\mathcal{C} \subset \mathbb{Z}_t$.

Especially, we propose the use of code $\mathcal{C} = \{\zeta^j\}_{j=0}^{n} \subset \mathbb{Z}_t$, where ζ here is a n-th primitive root of unity within $\mathbb{Z}_t$. This particular choice of code seamlessly facilitates the adaptation of the techniques outlined in Section 3 to the BGV and BFV schemes, making the process straightforward. Then the encoding and decoding are defined as follows

$$\begin{aligned}\mathsf{encode}(\vec{x}) &= \zeta^{\mathsf{int}(\vec{x})}\\ \mathsf{decode}(u) &= \mathsf{int}^{-1}(\log_{\zeta}(u)),\end{aligned}$$

where int^{-1} is an inverse mapping of $\int$ and $\log_{\zeta}$ is a discrete logarithm[1].

The definition of *vector multiplication in k-axis* and *matrix multiplication in k-axis* are extended to $\mathbb{Z}_t$, where t denotes the plaintext modulus of BGV or BFV scheme. process for identifying the multivariate interpolation polynomial, as delineated in Equation (6), involves the construction of the Vandermonde matrix U and its inverse U^{-1}, expressed as:

$$U = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & \zeta^{1\cdot 1} & & \zeta^{(n-1)\cdot 1} \\ & & \ddots & \\ 1 & \zeta^{1\cdot(n-1)} & \ldots & \zeta^{(n-1)\cdot(n-1)} \end{bmatrix}, U^{-1} = n^{-1} \cdot \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & \zeta^{-1\cdot 1} & & \zeta^{-(n-1)\cdot 1} \\ & & \ddots & \\ 1 & \zeta^{-1\cdot(n-1)} & \ldots & \zeta^{-(n-1)\cdot(n-1)} \end{bmatrix} \in \mathbb{Z}_t^{n\times n},$$

where n^{-1} is the multiplicative inverse of n in $\mathbb{Z}_t$. Importantly, applying U^{-1} through matrix multiplication in the k-axis, $\boxtimes_k U^{-1}$, is equal to applying inverse NTT on k-axis rows, and thus can be done efficiently as in CKKS using FFT.

Subsequently, the components of the multi-dimensional matrices C, as determined by Equation (6), serve as the coefficients for the interpolation polynomial, which in turn is represented by the multi-dimensional matrix D. Say,

$$f_j(u_0, \ldots, u_{\alpha-1}) = \sum_{i_0=0}^{n-1} \cdots \sum_{i_{\alpha-1}=0}^{n-1} C_j[i_0, \cdots, i_{\alpha-1}] \cdot u_0^{i_0} \cdots u_{\alpha-1}^{i_{\alpha-1}}.$$

is homomorphically evaluated.

In contrast to the CKKS scheme, which inherently incorporates noise within its plaintexts, the BGV/BFV schemes ensure that plaintexts remain devoid of noise. This obviates the necessity for additional noise analysis or reduction algorithms beyond what is standardly provided. The proposed scheme allows for the efficient utilization of LUTs based on BGV and BFV with low-degree polynomials, without necessitating modifications to the underlying HE schemes, thereby enhancing usability.

[1]As the search space is very small, solving the discrete logarithm can be done very efficiently

6 Experimental Results

In this section, we demonstrate the effectiveness of the proposed method through implementation. We employ AES-128 decryption as a benchmark operation within the CKKS scheme.

AES serves as a valuable metric for evaluating performance and is practically indispensable. Transciphering is a technique aimed at reducing communication/computation overhead on the client side. Numerous works in the literature have focused on transciphering. One line of research advocates for HE-friendly symmetric key cryptosystems. Considering the substantial challenges in migrating cryptosystems (even the transition from DES to AES required considerable time and effort following the discovery of DES vulnerabilities, and migrating to Post-Quantum Cryptography is a government-scale endeavor), it remains uncertain whether such HE-friendly cryptosystems can be feasibly adapted for real product use in transciphering applications.

It is worth noting that due to its approximate computation nature, achieving the original AES over CKKS has been considered challenging, and the performance of its homomorphic evaluation was not satisfactory [HKL+22].

Another line of work involves implementing existing trustworthy cryptosystems, the AES, efficiently on top of HE schemes [GHS12, TCBS23] (and other references from TCBS23). However, the TFHE/FHEW-based implementation [TCBS23] exhibits low throughput since FHEW/TFHE lacks support for SIMD. The well-known implementation by Gentry, Halevi, and Smart (GHS) [GHS12] demonstrates good performance, considering advancements in HE algorithms/libraries and computing speed over the years (although it was published more than a decade ago). However, it serves more as a benchmark than as an AES transciphering solution, as in their setting, the BGV encryption is configured to support operations on a Galois field, which is not frequently used in general computations. Moreover, it is designed without allowing further operations of the final ciphertext. Therefore, we believe that the proposed improvement serves as a robust benchmark to showcase performance and represents a practical transciphering proposal.

6.1 A Brief Overview

The AES-128 consists of 10 rounds of the same operations with different round keys. Each round is operated on 4×4 matrix of bytes, where each bytes are considered as an element of a Galois field $\mathrm{GF}(2^8)$. We use $n = 2^4$ as the fact that 4 divides 8 makes the algorithm easier to understand and implement.

Each round function of AES consists of four operations: AddKey ,SubBytes, ShiftRows, and MixColumns. AddKey is simply XOR of the 16-byte round key and current state, as XOR is a bit-wise operation, we can evaluate this with simple 4-to-4 LUT. Besides, it is observed that LUT for XOR is sparse, i.e., the LUT polynomial has many zeros in its coefficients, and thus we can skip some ciphertext-ciphertext multiplications. SubBytes is S-box look up which is finding α^{-1} for $\alpha \in \mathrm{GF}(2^8)$, considering each byte in the state as element of $\mathrm{GF}(2^8)$. This is naturally done by the proposed method using 8-to-8 LUT evaluation. In ShiftRows step, we consider the state as 4×4 matrix and shift each row by some amount. It can be done by rotation operation and inner product with indicator plaintext vectors, which have only one and zeros as elements. MixCol multiplies the state 4×4 matrix by a fixed 4×4 matrix in $\mathrm{GF}(2^8)^{4\times 4}$. The multiplication in $\mathrm{GF}(2^8)$ is done by an 8-to-8 LUT evaluation, where we find the LUT evaluation polynomial for multiplication by given

amount[2]. As in [GHS12], we merge ShiftRows and MixColumns steps and reduce the computation.

Parameter selection and implementation details. To meet 128-bit security, we use $N = 2^{16}$, $Q = 2^{1658}$, and scaling factor $\sim 2^{59}$. For ease of implementation, we do bootstrapping after each LUT evaluation, and each LUT has depth 5. There are 10 rounds in AES-128, and we do noise reduction every round for easy implementation. We use Liberate.FHE library [DES23] on a single NVIDIA A100 GPU for implementation. Hence, it might be hard to do a fair comparison with previous works considering the difference in environment.

6.2 Homomorphic Evaluation of Basic Operations

Representation of AES state. We divide a byte into two 4-bit numbers, the left half and the right half. A half byte is encoded into $\mathcal{C} = \{\zeta^j\}_{j=0}^{15}$, where $\zeta = \exp(\frac{-2\pi i}{16})$ thus each slot has a codeword $c \in \mathcal{C}$. For ease of understanding and implementation, we use separate ciphertexts for left half bytes and right half bytes. We choose a CKKS parameter with ring dimension $N = 2^{16}$, whose maximum number of slots is 2^{15}. As the state requires 16 bytes, we put $2^{15}/16 = 2048$ ciphertext batched in two ciphertexts (each slot in the same position represents the left and right half of the same bits). Following Gentry-Halevi-Smart [GHS12], we place the 16 bytes of the AES state in plaintext slots using column-first ordering, namely, we have

$$\alpha = [\alpha_{00}, \alpha_{10}, \alpha_{20}, \alpha_{30}, \alpha_{01}, \alpha_{11}, \alpha_{21}, \alpha_{31}, \alpha_{02}, \alpha_{12}, \alpha_{22}, \alpha_{32}, \alpha_{03}, \alpha_{13}, \alpha_{23}, \alpha_{33},],$$

representing the input plaintext matrix

$$A = (\alpha_{ij})_{i,j} = \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix}.$$

The batching is done by placing elements of α by the gap of $2^{15}/16 = 2048$. For example, let β and γ are other AES states we are batching. Then the coefficient slot will look like the following array

$$\overbrace{[\underbrace{\alpha_{00}, \beta_{00}, \gamma_{00}, \ldots}_{2048}, \underbrace{\alpha_{10}, \beta_{10}, \gamma_{10}, \ldots}_{2048}, \underbrace{\alpha_{20}, \beta_{20}, \gamma_{20}, \ldots}_{2048}, \ldots \underbrace{\alpha_{33}, \beta_{33}, \gamma_{33}, \ldots}_{2048}]}^{2^{16}}$$

By doing this, we can rotate each state by r independently by applying a rotation of $2048r$. Finally, we encode each element with encode and encrypt it.

AddKey. Here, we assume the AES round key is given as a CKKS ciphertext[3]. AddKey is simply an XOR with AES state and encryption key, it can be done by 8-to-4 LUT. In our experiment, we used two ciphertexts, which require two executions of 8-to-4 LUTs, and it took 1.63s. Note that

[2]In encryption of AES, we only multiply by two or three, which can also be done by XORs, and cheaper. However, in this paper, we did not apply such optimization as the goal of this section is to provide a reference usage of the proposed method.

[3]The proposed method supports arbitrary LUT operations, the key expansion can also be done when the 128-bit secret key is given as encrypted form in HE.

the polynomials for XOR have many zeros in them and only a quarter of them are non-zeros, thus, the homomorphic evaluation of the polynomial can be done very fast.

SubBytes. SubBytes and its inverse InvSubBytes can be implemented as 8-to-8 LUT. An 8-to-8 LUT is done by two 8-to-4 LUTs. As those two LUTs share the inputs, we can calculate the power basis and reuse it. Thus, it took 0.94s in our experiment, which is almost half of AddKey. We do the same LUTs on each slot.

ShiftRows and MixColumns. As in [GHS12], we can merge the ShiftRows and MixColumns steps. We explain the AES decryptions in detail, but the encryption can also be done similarly. In InvMixColumns, we calculate the following matrix multiplication in $\mathrm{GF}(2^8)$

$$A' = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \times A,$$

where A and $A' \in \mathrm{GF}(2^8)^{4\times 4}$ represents the input and output AES states of InvMixColumns, respectively. For given input state A, the output of InvShiftRows is A' as given below

$$A = \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix}, A' = \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{13} & \alpha_{10} & \alpha_{11} & \alpha_{12} \\ \alpha_{22} & \alpha_{23} & \alpha_{20} & \alpha_{21} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{30} \end{pmatrix}.$$

To merge those steps, we first do a Hadamard multiplication of the input state A, and each following matrix

$$X_0 = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 14 & 11 & 13 & 9 \\ 14 & 11 & 13 & 9 \\ 14 & 11 & 13 & 9 \end{pmatrix}, X_1 = \begin{pmatrix} 9 & 14 & 11 & 13 \\ 9 & 14 & 11 & 13 \\ 9 & 14 & 11 & 13 \\ 9 & 14 & 11 & 13 \end{pmatrix}, X_2 = \begin{pmatrix} 13 & 9 & 14 & 11 \\ 13 & 9 & 14 & 11 \\ 13 & 9 & 14 & 11 \\ 13 & 9 & 14 & 11 \end{pmatrix}, X_3 = \begin{pmatrix} 11 & 13 & 9 & 14 \\ 11 & 13 & 9 & 14 \\ 11 & 13 & 9 & 14 \\ 11 & 13 & 9 & 14 \end{pmatrix}.$$

This step requires an 8-to-32 LUT, this can be done easily as the proposed scheme supports independent LUTs on each slot. Then, we XOR each row put the value in the first column and the other three columns become zeros, we call these matrices B_0, B_1, B_2, and B_3. In this step, we rotate the ciphertext to the left by two and then do XOR with the ciphertext before rotation. Then rotate again by one, and then do XOR again. In the final XOR, to make the columns except the first to zero, we do independent LUTs. For the 0-th, 4-th, 8-th, and 12-nd slots, we apply coefficients of polynomials for XOR, but for the other slots, we put zeros on all coefficients. By doing this, we can save a multiplication by $(1, 0, 0, 0, 1, \dots)$ without additional cost. Finally, we find $B_0 + B_1 \gg 5 + B_2 \gg 10 + B_3 \gg 15$ which gives us the final result of InvMixColumns and InvShiftRows. This step takes 41.1s, the major amount of runtime.

6.3 Homomorphic AES Evaluation Performance

Our AES decryption experiment on a single NVIDIA A100 GPU was completed in 573.83 seconds, as shown in Table 1 alongside previous studies. Unlike most prior work that utilized DM/CGGI HE schemes due to their inherent LUT support, our method and [GHS12] utilizes amortized HE. DM/CGGI relies on *functional bootstrapping* for LUT evaluation, a necessity in these schemes. The

Table 1: Transciphering Comparison with Previous Work: While the values in this table are borrowed from their manuscript and thus may not facilitate a fair comparison due to distinct operational environments, our analysis on a single NVIDIA A100 GPU is believed to be the most advantageous. Despite these considerations, our approach demonstrates substantial amortization and enhanced performance.

	GHS12[GHS12]	TCBS23[TCBS23]	WWL+23[WWL$^+$23]	ours
Symm. ciphier	AES-128	AES-128	AES-128	AES-128
Total Runtime (sec)	245.1/394.3	28.73	9	573.83
# of ciphertext amortized	120	1	1	2048
Amortized runtime (sec)	2.04/3.29	28.73	9	0.28
Base scheme	BGV	CGGI	CGGI	CKKS*
Further operation	No (Yes)	Yes	Yes	Yes
Requires bootstrapping	No	Yes	Yes	Yes

*Also supports BGV/BFV but not implemented.

approach in [GHS12] wasn't tailored for transciphering, as it does not consider the levels after AES evaluation. However, recent advancements, such as those by Lee-Min-Song [LMS24], leverage BGV bootstrapping to enable further computations after bootstrapping, and thus we can use [GHS12] as transciphering.

As a proof of concept, this implementation opens the door to further optimizations. Adjusting parameters could yield improvements; our current scaling factor, 2^{59} (normalized by scaling factor), is overly precise for the code $|\mathcal{C}|= 2^4$, leading to noise after AES decryption has a standard deviation of $2^{-37.4}$, which is superfluous. Refining parameter choices may cut down on the frequent bootstrapping after every LUT, a process that dominates runtime. There are a series of ongoing efforts in improving CKKS bootstrapping [CHK$^+$18, CCS19, HK20, BMTH21, LLL$^+$21, JKA$^+$21, LLK$^+$22, BCC$^+$22], and the improvement of CKKS bootstrapping will improve the performance of the proposed method. Using a CKKS-encrypted AES state instead of conventional plaintext extends the duration of the first AddRoundKey step. Combining invSubBytes with AddKey into a LUT presents additional optimization opportunities. This list is not exhaustive; other optimization avenues remain unexplored.

6.4 Runtime Results of LUT Evaluation

Tables 2 and 3 present the runtime results for evaluating a single LUT, including comparative analyses with prior studies. In the CKKS case, we adopted identical parameters to those in the AES experiment, with $|\mathcal{C}|= 2^4 = 16$ utilized. We use Liberate.FHE library [DES23] on a single NVIDIA A100 GPU. In Table 2, we have the runtime result for the proposed scheme based on BFV and BGV schemes. Those are tested with parameter $t = 65537$, $q = 2^{842}$, and $N = 2^{15}$. The running environment is equal to Liu-Wang [LW23c], Google Compute Cloud e2-standard-4, 16GB, implemented using OpenFHE v1.1.2 [Ope22].

Our scheme's inherent adaptability is highlighted by the wide range of parameter choices available, including the CKKS scaling factor and code size. For example, a configuration space of $|\mathcal{C}|= 2^6 = 64$ might be beneficial for a 12-to-12 LUT. Nevertheless, given the extensive possible outcomes, this paper limits its focus to the more general implementation aspects of our scheme, rather than detailing every potential result.

Table 2: Comparison with previous work for smaller than 10-Input LUTs: This table integrates findings from [LW23c], the recognized state-of-the-art, with our results. It's important to clarify that discrepancies in testing environments The comparison may not be entirely equitable due to differences in computational environments. Despite these variations, our method shows significant speed advantages.

LUT space	3-to-3	7-to-7		8-to-8			9-to-9	8-to-8		8-to-12
Scheme	[LMP22]	[GPvL23]	[KS21]	[GBA21]	[LMP22]	[LHH$^+$21]	[LW23c]	Ours-BFV (BGV)	Ours-CKKS*	
Amortized time per LUT(ms)	1192	1205	35169	2203	1793	3476	**6.7**	**1.01(0.996)**	**0.029 (0.15)**	**0.042 (0.17)**
Total time (sec)	1.192	1234	35.169	2.203	1.793	3.476	**220**	**33.26(32.64)**	**0.94 (4.94)**	**1.37 (5.43)**
# of LUTs per amortized	1	1024	1	1	1	1	**32768**	**32768**	**32768**	**32768**
Input assumption	No	No	No	Yes	Yes	Yes	**No**	**No**	**No**	**No**

*values in parenthesis is runtime including a CKKS bootstrapping.

In Tables Table 2 and Table 3, we apply AES s-box for the 8-to-8 Look-Up Table (LUT) and deploy arbitrary dense LUTs in other contexts. The performance remains largely unaffected by the specific entries in the LUTs, owing to the absence of optimization techniques for sparse LUT polynomials. Despite the advantage of operating in a superior computational setting in the CKKS case, our scheme exhibits enhanced performance when compared with the leading method introduced by Liu and Wang [LW23c]. Under identical environments and parameter settings, our approach outperforms that of [LW23c] by a factor of six. In our framework, it is possible to process four consecutive LUTs, followed by a necessary bootstrapping for further computation. On the other hand, the method in [LW23c] permits only a single LUT execution, which inherently fulfills the role of bootstrapping, thereby negating the need for an additional bootstrapping phase. This implies that our method maintains its superior speed advantage as long as the BFV bootstrapping time remains below 752 seconds. This performance improvement stems from our approach's utilization of low-degree multivariate polynomials—with degrees set at 16 or 24—and our methodical reuse of polynomial bases for producing larger outputs, as opposed to the reliance on high-degree sparse polynomials (e.g., degrees of 65536) in the methodology proposed in [LW23b].

7 Conclusion

In this work, we introduce an efficient method for evaluating look-up tables (LUTs) within homomorphic encryption (HE) frameworks, specifically targeting Ring-LWE-based schemes like BGV, BFV, and CKKS. Our approach, which encodes bit streams into codewords and transforms LUTs into low-degree multivariate polynomials, allows for the simultaneous evaluation of multiple LUTs on a single encrypted message, significantly reducing computational overhead in the context of amortization. To alleviate noise accumulation in the CKKS scheme, we also propose a novel noise-reduction polynomial, accompanied by proof demonstrating its effectiveness in asymptotically decreasing noise levels. The practicality of our method is demonstrated through a proof-of-concept implementation, showcasing notable improvements in the efficiency of LUT operations and AES-128 decryption times. Moreover, we put forward a novel transciphering technique within the CKKS scheme, leveraging the strength of symmetric encryption to bridge discrete and numerical data in

Table 3: Comparison with prior arts for larger than 10-input LUTs. The same environment and the same parameter as in Table 2 is used. Hence, we exploit polynomials with three variables for LUT evaluation.

LUT space	10-to-10	12-to-12				12-to-16
Scheme	[GBA21]	[LMP22]	[GBA21]	[LW23c]	Ours*	
Amortized time per LUT(ms)	23667	3998	51085	**39.1**	**0.58 (0.77)**	**0.78 (0.96)**
Total time (sec)	23.667	3.998	51.085	**1280**	**19.1 (25.13)**	**25.4 (31.45)**
# of LUTs per amortized	1	1	1	**32768**	**32768**	**32768**
Input assumption	Yes	Yes	Yes	**No**	**No**	**No**

*values in parenthesis is runtime including a CKKS bootstrapping.

encrypted formats.

The proposed scheme's adaptability paves the way for numerous intriguing future research directions. In this work, we adopted a consistent codeword set, $\mathcal{C}$, throughout the computations. However, the use of interpolation polynomials offers the flexibility to transition between different codeword sets, enabling a ciphertext encoding a bitstream in $\mathcal{C}$ to be seamlessly converted to another set $\mathcal{C}'$, even when $|\mathcal{C}| \neq |\mathcal{C}'|$. This capability extends to the conversion between codewords and complex numbers, enhancing the versatility of our approach. For the CKKS scheme, there's a promising opportunity to optimize by integrating the error reduction function directly with the polynomials used for LUT evaluation. Such an integration could significantly reduce the computational requirements and execution time for LUT operations. Our current AES implementation, while serving as a solid reference, presents numerous possibilities for optimization. Future efforts could focus on refining the AES algorithm, potentially by merging LUTs used in consecutive rounds or by adjusting the size of $\mathcal{C}$ to optimize performance, taking full advantage of our scheme's ability to accommodate arbitrary LUTs in HE. These potential enhancements stand to streamline encrypted computations further, broadening the scope of our method's application in ensuring secure and efficient data processing.

References

[BBB+23] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (t)fhe. *Journal of Cryptology*, 36, 2023.

[BCC+22] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. Meta-bts: Bootstrapping precision beyond the limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 223–234, 2022.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BMTH21] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Advances in Cryptology – EUROCRYPT 2021*. Springer, 2021.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, pages 868–886. Springer, 2012.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2019*, pages 34–54. Springer, 2019.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology – ASIACRYPT 2017*, pages 377–408. Springer, 2017.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, pages 34–91, 2020.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2018*, pages 360–384. Springer, 2018.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New techniques for multi-value input homomorphic evaluation and applications. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 106–126, Cham, 2019. Springer International Publishing.

[CKK20] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology - ASIACRYPT 2020*, pages 221–256. Springer, 2020.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437. Springer, 2017.

[DES23] DESILO. Liberate.FHE: A New FHE Library for Bridging the Gap between Theory and Practice with a Focus on Performance and Accuracy, 2023. `https://github.com/Desilo/liberate-fhe`.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015*, pages 617–640. Springer, 2015.

[DMPS22] Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. Bleach: Cleaning errors in discrete computations over ckks. Cryptology ePrint Archive, Paper 2022/1298, 2022. `https://eprint.iacr.org/2022/1298`.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[GBA21] Antonio Guimarães, Edson Borin, and Diego F Aranha. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 229–253, 2021.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology – CRYPTO 2012*, pages 850–867. Springer, 2012.

[GPvL23] Antonio Guimarães, Hilder V. L. Pereira, and Barry van Leeuwen. Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. Cryptology ePrint Archive, Paper 2023/014, 2023.

[HK20] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Topics in Cryptology – CT-RSA 2020*, pages 364–390. Springer, 2020.

[HKL+22] Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption. In *Advances in Cryptology - EUROCRYPT 2022*, pages 581–610. Springer, 2022.

[JKA+21] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:114–148, 2021.

[KS21] Kamil Kluczniak and Leonard Schild. FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. *arXiv preprint arXiv:2109.02731*, 2021.

[LHH+21] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE symposium on Security and Privacy (S&P)*, pages 1057–1073. IEEE, 2021.

[LLK+22] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In *Advances in Cryptology - EUROCRYPT 2022*, pages 551–580. Springer, 2022.

[LLL+21] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In *Advances in Cryptology – EUROCRYPT 2021*. Springer, 2021.

[LMK+23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2023*, pages 227–256. Springer, 2023.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using fhew/tfhe bootstrapping. In *Advances in Cryptology–ASIACRYPT 2022*, pages 130–160. Springer, 2022.

[LMS24] Dongwon Lee, Seonhong Min, and Yongsoo Song. Functional bootstrapping for fv-style cryptosystems. Cryptology ePrint Archive, Paper 2024/181, 2024. https://eprint.iacr.org/2024/181.

[LW23a] Feng-Hao Liu and Han Wang. Batch bootstrapping i: A new framework for simd bootstrapping in polynomial modulus. In *Advances in Cryptology – EUROCRYPT 2023*, page 321–352. Springer-Verlag, 2023.

[LW23b] Feng-Hao Liu and Han Wang. Batch bootstrapping ii: Bootstrapping in polynomial modulus only requires o(1) FHE multiplications in amortization. In *Advances in Cryptology – EUROCRYPT 2023*, page 353–384. Springer-Verlag, 2023.

[LW23c] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{o}(1)$ polynomial multiplications. In *Advances in Cryptology - ASIACRYPT 2023*, pages 101–132. Springer, 2023.

[MKG23] Johannes Mono, Kamil Kluczniak, and Tim Güneysu. Improved circuit synthesis with amortized bootstrapping for fhew-like schemes. Cryptology ePrint Archive, Paper 2023/1223, 2023. https://eprint.iacr.org/2023/1223.

[MKMS23] Gabrielle De Micheli, Duhyeong Kim, Daniele Micciancio, and Adam Suhl. Faster amortized fhew bootstrapping using ring automorphisms. Cryptology ePrint Archive, Paper 2023/112, 2023. https://eprint.iacr.org/2023/112.

[MS18] Daniele Miccianco and Jessica Sorrell. Ring packing and amortized FHEW bootstrapping. In *45th International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[Ope22] OpenFHE. Open-Source Fully Homomorphic Encryption Library. https://github.com/openfheorg/openfhe-development, 2022.

[TCBS23] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. A homomorphic AES evaluation in less than 30 seconds by means of TFHE. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 79–90. ACM, 2023.

[WWL+23] Benqiang Wei, Ruida Wang, Zhihao Li, Qinju Liu, and Xianhui Lu. Fregata: Faster homomorphic evaluation of aes via tfhe. In Elias Athanasopoulos and Bart Mennink, editors, *Information Security*, pages 392–412, Cham, 2023. Springer Nature Switzerland.

[YXS+21] Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. TOTA: fully homomorphic encryption with smaller parameters and stronger security. *IACR Cryptol. ePrint Arch.*, page 1347, 2021.

[Zam22] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.