

LEAF: A Low-Latency Evaluation Architecture for Feedforward Block in Privacy-Preserving Transformer Inference

Linru Zhang¹, Xiangning Wang¹, Xianhui Lu², Huaxiong Wang¹, and Kwok-Yan Lam¹

¹ Nanyang Technological University, Singapore, Singapore,
linru.zhang@ntu.edu.sg, xiangning.wang@ntu.edu.sg, hxwang@ntu.edu.sg,
kwokyan.lam@ntu.edu.sg

² Chinese Academy of Sciences, Beijing, China, luxianhui@iie.ac.cn

Abstract. Fully homomorphic encryption (FHE) is an appealing and promising solution for privacy-preserving transformer inference to protect users' privacy. However, the huge computational overhead makes it unrealistic to apply FHE in real-world transformers for large language models (LLM). Current FHE-based approaches to secure transformer inference face significant performance challenges, with total latency exceeding 5 hours for 32-input batches.

The feedforward block, comprising a large-scale matrix multiplication followed by a GELU evaluation, is widely recognized as one of the most computationally intensive components in privacy-preserving transformer inference. In the state-of-the-art system NEXUS [49], evaluating the feedforward block incurs a total latency of 5,378 seconds, processing up to 32 inputs per batch.

We aim to reduce the latency and propose LEAF, a low-latency evaluation architecture for the feedforward block. LEAF introduces a novel combination of fast matrix multiplication and an asymptotically efficient algorithm for computing non-polynomial activations. When evaluated on the BERT-base model, LEAF reduces total latency to 53.4 seconds, offering a 100× speedup over the state-of-the-art method in the same environment. Our implementations are available.³

Keywords: Fully homomorphic encryption · Large-language model · Privacy-preserving AI · Non-polynomial function

1 Introduction

Fully homomorphic encryption (FHE), proposed by Gentry [17], enables evaluations of arbitrary functions on encrypted data without decryption. FHE has been widely adopted in AI-as-a-Service scenarios to safeguard user privacy, while achieving practical inference latency on relatively simple models. For example,

³ <https://github.com/zhanglr3/LEAF-HE>

DiNN [4] demonstrates an inference latency of 0.49 seconds on a basic back-propagation network with a single hidden layer of 30 nodes. LoLa [6] further reduces latency to 0.29 seconds for a network with 100 hidden nodes and reports 2.2 seconds for a convolutional neural network with two convolutional layers. Progressing toward deeper architectures, SHE [33] reports a latency of approximately 2 minutes for privacy-preserving inference on ResNet-18 [23].

However, the substantial computational overhead of FHE presents a significant barrier to its application in transformer-based large language models (LLMs), such as GPT [43], BERT [11], and LLaMA [47]. Only one existing work has explored the application of FHE to privacy-preserving transformer inference. NEXUS [49] uses CKKS scheme and reports an total latency exceeding 5 hours for 32-inputs batch in the BERT-base model, obtained by summing the execution time of each individual operation on a 32-core CPU.

The feedforward block is one of the most computationally expensive components in privacy-preserving transformer inference. It comprises two key operations: (1) large-scale plaintext-ciphertext matrix multiplication—of size 768×3072 in BERT-base and 4096×14336 in LLaMA-3-8B—and (2) GELU activation function evaluation, applied to inputs of size 128×3072 in BERT-base or 8×14336 in LLaMA-3-8B. NEXUS incurs a total latency of 5,378 seconds for evaluating the feedforward layer, even when processing a batch of up to 32 inputs.⁴

There exist two main challenges to reduce the latency.

Matrix multiplication. Evaluating plaintext-ciphertext matrix multiplication takes 45% of the runtime of evaluating feedforward block in NEXUS [49]. It uses *slot encoding* to encode vector from $\mathbb{C}^{N/2}$ into one polynomial using the isomorphism between $\mathbb{C}^{N/2}$ and the polynomial ring with degree N . Based on slot encoding, the encrypted matrix is packed into one or more CKKS ciphertexts. The plaintext-ciphertext matrix multiplication is then performed on slots. The performance of the algorithm is hindered by the large RLWE modulus and the costly rotations of slots.

Non-polynomial function evaluation and bootstrapping. In CKKS-based schemes such as NEXUS, non-polynomial functions are typically approximated using polynomials, which are subsequently evaluated across all ciphertext slots. Achieving sufficient approximation accuracy often requires high-degree polynomials. However, evaluating high-degree polynomials introduces significant computation overhead: (1) it increases the multiplicative depth needed, requiring a very large ciphertext modulus and thereby increasing the overall computational overhead; (2) it often requires a bootstrapping step—either before or after the polynomial evaluation—to refresh ciphertexts, further degrading performance. Bootstrapping [17], which uses an encrypted secret key to “refresh” ciphertext with reduced noise and restored multiplicative depth, is an essential part in FHE schemes to achieve the capability of evaluating circuits with arbitrary depth. Due to its high computational cost, bootstrapping is typically performed only when the

⁴ This result is obtained by running NEXUS’s open-source library on an Intel(R) Xeon(R) Platinum 8480+ CPU at 2.0 GHz. Further details are provided in section 4.

remaining multiplicative levels are insufficient to support further homomorphic operations.

1.1 Our contribution

To overcome these limitations, we propose LEAF, a novel low-latency evaluation architecture for feedforward block in privacy-preserving transformer inference.

For the matrix multiplication, LEAF adopts the efficient plaintext-ciphertext matrix multiplication (PCMM) algorithm from [1] which leverages coefficient encoding—a technique that maps each vector element to a polynomial coefficient—to eliminate costly data movement across ciphertext slots. For non-polynomial activation functions, LEAF introduces a new asymptotically efficient evaluation technique that is compatible with the coefficient encoding framework. Our non-polynomial evaluation algorithm achieves an amortized complexity of $\tilde{O}(1)$ polynomial multiplications per input, and support ciphertext refreshing during the evaluation. This eliminates the need for separate bootstrapping procedures. Together, these innovations significantly reduce inference latency and enable practical, scalable privacy-preserving inference for transformer models.

To demonstrate the practicality of our framework, we evaluate it on the BERT-base model. Specifically, our approach takes a 128×768 matrix as input and encrypts it, then performs a matrix multiplication with a 768×3072 plaintext weight matrix, and applies the GELU activation function to 128×3072 numbers outputted by matrix multiplication and refreshes the resulting ciphertexts. This entire pipeline is completed in just 53.4 seconds on a 56-thread server, while NEXUS takes 5378 seconds in the same environment. These results underscore the effectiveness of our solution in reducing the computational overhead in FHE-based privacy preserving inference of transformer models.

Besides the algorithms and framework mentioned above, we also propose some optimizations in each step to improve its efficiency. We implemented our scheme using SEAL [44], and our implementations are available at Anonymous GitHub.⁵

1.2 Technical Overview

Among lots of FHE schemes, the CKKS scheme [8] has emerged as one of the most widely chosen approaches for privacy-preserving transformer inference, due to its support for approximate arithmetic over encrypted complex numbers. A key feature of CKKS is its ability to perform operations in a single-instruction-multiple-data (SIMD) style, by packing multiple inputs into ciphertext slots, enabling efficient computation over a large amount of data. NEXUS [49] and another concurrent work THOR [41] use slot-encoded CKKS scheme in transformer inference. However, their performance is significantly constrained by the use of extremely large ciphertext modulus. NEXUS, for example, employs a 1763-bit modulus and have high cost of frequent bootstrapping operations. As

⁵ <https://anonymous.4open.science/r/HE-based-non-polynomial-eval-25E3>

illustrated in Figure 1(a)⁶, the complete evaluation of a feedforward block using slot-encoded CKKS in NEXUS requires at least 30 multiplicative layers, leading to a ciphertext modulus over one thousand bits and significant computational cost.

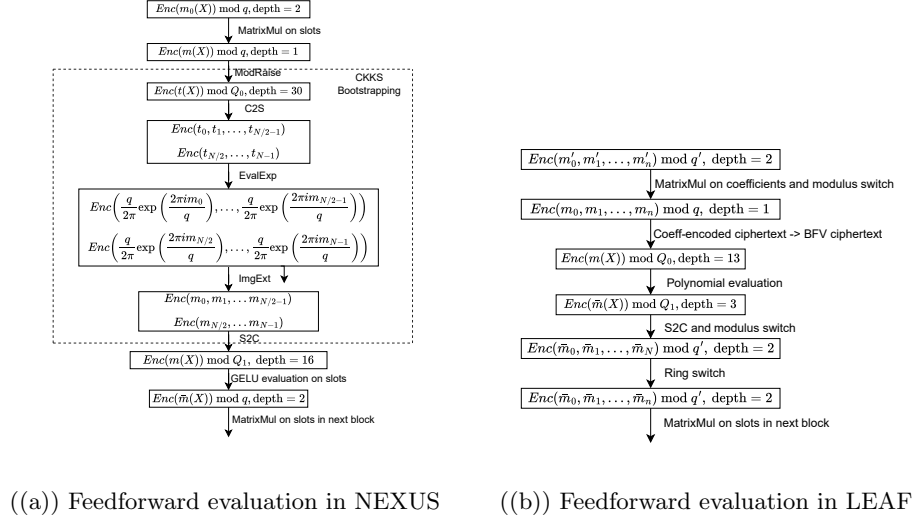


Fig. 1: Comparison between NEXUS and LEAF

As illustrated in Figure 1(b), our scheme LEAF proposes a novel framework to simplify the evaluation process and requires only 13 multiplicative layers—corresponding to a ciphertext modulus of less than 700 bits—thereby significantly reducing computational overhead. The improvements come from two aspects: (1) Using *coefficient encoding* instead of slot encoding, where each element of the matrix is set to one coefficient of polynomial. our LEAF makes use of the efficient matrix multiplication proposed by [1]. It eliminates costly slot rotations compared with matrix multiplications using slot encoding. (2) LEAF draws inspiration from the idea of functional bootstrapping [13, 31, 36]: to combine the refreshing process and the functional evaluation process into one function to be evaluated. Compared with NEXUS and THOR, although we need to move data between coefficients and slots, our method LEAF requires less multiplicative layers and a much smaller ciphertext modulus. This can significantly improve the efficiency of the whole scheme.

⁶ According to Section VI (Evaluation) in NEXUS [49], the plaintext-ciphertext matrix multiplication needs 1 multiplicative layer, the CKKS bootstrapping requires 14 multiplicative layers, and the evaluation of GELU also needs 14 multiplicative layers.

In addition, we significantly extend and optimize the functional bootstrapping framework in [31]. While their method starts from LWE ciphertexts and maps them to slot-encoded BFV ciphertexts for evaluation, and ends by extraction to obtain LWE ciphertexts, we address a different setting and introduce key optimizations: First, we propose a new transformation algorithm that lifts many low-level coefficient encoded RLWE ciphertexts in polynomial ring with small degree (aligned with the output format of [1]) into one high-level slot-encoded BFV ciphertext in ring with larger degree. We then follow the polynomial evaluation and slot-to-coefficient (S2C) procedures in [31], but apply several optimizations to improve computational efficiency. To complete the process, we adopt the ring-switching technique from [2], enabling us to return refreshed, evaluated coefficient encoded RLWE ciphertexts in the original ring with smaller degree.

The transformation algorithm uses the fact that the decryption formula of each element m_i in the low-level coefficient encoded RLWE ciphertext can be written as a linear operation $\bar{b}_i + \langle \bar{\mathbf{a}}_i, \mathbf{sk} \rangle \bmod q$, where \bar{b}_i and vector $\bar{\mathbf{a}}_i$ are extracted from the ciphertext and vector \mathbf{sk} consists of the coefficients of the secret key. By encrypting the RLWE secret key vector to a high-level slot encoded BFV ciphertext, we can homomorphically evaluate the decryption formula to get a high-level slot encoded BFV ciphertext by linear transformation (LT) algorithm, which computes the multiplication between a matrix and an encrypted vector. An important detail in this process is the arrangement of messages within the BFV slots: they must follow an interleaved order. This layout is specifically designed to align with the ring-splitting algorithm, ensuring that for any message m originally in the i -th position of the j -th input RLWE ciphertext, the evaluation result $GELU(m)$ remains in the i -th position of the j -th output RLWE ciphertexts. This positional consistency is crucial for the correctness of subsequent matrix multiplication steps.

Finally, we present several optimizations to improve computational efficiency. We categorize the BFV LT algorithm into two cases. The first is a specialized variant that requires fewer ciphertext rotations and is tailored for matrices with a specific structure—this form is leveraged in our transformation algorithm. The second one supports general matrix structures and is employed during the S2C process. Both algorithms are optimized using the Residue Number System (RNS) to minimize computational latency. During the S2C process, we observe that the decoding matrix can be precomputed and encoded in advance. By incorporating this and other optimizations, we reduce the S2C latency from 137 seconds to 15 seconds compared to [32], under a single-threaded setting on the same machine. We also design a parallel polynomial evaluation algorithm that leverages modern multi-core CPUs to improve both asymptotic complexity and practical performance. This optimization accelerates the evaluation by a factor of 5.5 compared to [32].

1.3 Related works

Matrix multiplication in privacy-preserving transformer inference . NEXUS, based on slot-encoded CKKS, proposes a ciphertext compression and

decompression method in plaintext-ciphertext matrix multiplication to reduce the communication cost, and applies column-packing to organize a batch of inputs into slots of several ciphertexts. A concurrent work, THOR [41] also applies slot-encoded CKKS in privacy-preserving transformer inference. It uses a special “diagonal-packing” to organize the data from one matrix to vectors when performing matrix multiplications. It requires more rotations across slots in plaintext-ciphertext matrix multiplication. Recently, a line of works ([1, 30]) propose more efficient matrix multiplication using *coefficient encoding*. Reported in [1], it takes 0.42s and 1.7s for the product of two 256×256 matrices and two 1024×1024 matrices in single thread. To see how it works, let $M \in \mathbb{Z}_P^{d \times N}$ be the matrix to be encrypted. Based on coefficient encoding, we can encrypt the i -th row of the matrix M into one RLWE ciphertext as two polynomials $(\mathbf{a}_i, \mathbf{b}_i)$, for $i \in [d]$. [1] proposed the construction of matrices $(A, B) \in (\mathbb{Z}_Q^{d \times N})^2$ s.t. the i -th row of A (resp. B) is the coefficients of polynomial \mathbf{a}_i (resp. \mathbf{b}_i). Then given a plaintext matrix W with bounded elements, (WB, WA) is the RLWE ciphertext of WM .

Non-polynomial function evaluation. Evaluating common activations such as ReLU, GELU, Sigmoid and Softmax in HE-encrypted ciphertexts has been studied for many years to improve its efficiency and accuracy.

RLWE-based FHE schemes use polynomials to approximate non-polynomial activations. Although they support SIMD operations which can evaluate lots of inputs simultaneously, lots of efforts are required to improve the accuracy of the approximation. Furthermore, evaluation of high-degree polynomials consumes many multiplication levels, such that the expensive bootstrapping procedure may be required to ensure that there are enough levels for the following evaluations.

Taylor series is a popular way to approximate functions like exponentiation. $\exp(x) \approx (1 + \frac{x}{2^n})^{2^n}$, $x \leq 0$ is widely used in many works [35, 49] as a part of the softmax⁷ evaluation. However, for other popular activations, it may be difficult to find a polynomial to approximate it accurately in a reasonable input range.

For the evaluation of GELU, Puma [12] first utilized the fact that GELU is almost linear for $x > 0$ and $x < -1$, and designed a piece-wise approximation using low-degree polynomials to approximate GELU. It divides GELU into 4 pieces and use a multi-party protocol “less than” [26, 34, 40] to indicate which piece the input falls into. BumbleBee [35] followed Puma’s idea and proposed some optimizations to reduce the communication overhead of the GELU protocol by 5%. NEXUS [49] also used the piece-wise approximation but implements a HE sign evaluation to obtain 4 encrypted bits b_0, b_1, b_2, b_3 where $b_i = 1$ iff x belongs to the i -th segment. It removed the multi-party protocol and achieved non-interactive HE-based evaluation but the input range is limited to $x \in [-8, 8]$, which is usually not enough for the inference of real-world transformers. When the input is > 8 or < -8 , the error will become extremely large and make the

⁷ The definition of softmax function is $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{m-1} e^{x_j}}$ and commonly evaluated using $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum_{j=0}^{m-1} e^{x_j - x_{max}}}$ to achieve numerical stability [28].

evaluation meaningless. A concurrent work, THOR [41] further improved it by using a composition of two polynomials with degree 31 and 27, and leveraged the famous Paterson-Stockmeyer algorithm [42] for efficient polynomial evaluation and lower multiplicative depth.

LWE-based FHE schemes use homomorphic look-up-table (LUT) algorithm to evaluate non-polynomial functions. The core idea of LUT is to encode the “Table” containing the value of the non-polynomial function, so that we can apply “blind rotate” on the ciphertext to locate the position of the desired output. The algorithm is proposed by [39] and improved by many works [3, 4, 27, 36]. This kind of algorithms cannot support SIMD operations, making it inefficient for applications like inference of transformers. Reported in [36], it took about 0.9s for one LUT evaluation in single thread.

2 Preliminary

2.1 Notations

In this paper, we use \mathbb{Z} to denote the set of all integers and use \mathbb{Z}_q to denote the ring $\mathbb{Z}/q\mathbb{Z}$ for some integer $q \in \mathbb{Z}$. $[N]$ denotes the integer set $\{0, 1, \dots, N-1\}$. We use polynomial ring $\mathcal{R}^{(N)} = \mathbb{Z}[X]/(X^N + 1)$ to denote the $2N$ -th cyclotomic ring where N is a power of two. Similarly, let $\mathcal{R}_q^{(N)} = \mathcal{R}^{(N)}/q\mathcal{R}^{(N)}$ for some integer $q \in \mathbb{Z}$.

For set S we write $x \xleftarrow{\$} S$ to denote that, x is sampled uniformly random from set S . If \mathcal{D} is a distribution, we write $x \xleftarrow{\$} \mathcal{D}$ to denote that x is sampled from distribution \mathcal{D} . We use bold lowercase letter to denote (row) vector (e.g., $\mathbf{v} \in \mathbb{Z}^n$). Let v_i or $v[i]$ be the i -th element of vector \mathbf{v} , so we write $\mathbf{v} = [v_0, v_1, \dots, v_{n-1}]$. We use notations like $\mathbf{m}, \mathbf{a}, \mathbf{b}$ to denote the element in polynomial ring. The floor function is written as $\lfloor \cdot \rfloor$. The ceiling function is written as $\lceil \cdot \rceil$. The rounding function is written as $\lfloor \cdot \rceil$.

The ring learning with errors (RLWE) problem [38] is widely used to design homomorphic encryption schemes [5, 8, 9, 13]. The secret \mathbf{s} is chosen from polynomial ring $\mathcal{R}_Q^{(N)}$. An RLWE sample $(\mathbf{a}, \mathbf{a}\mathbf{s} + \mathbf{e}) \in (\mathcal{R}_Q^{(N)})^2$ is generated by choosing $\mathbf{a} \xleftarrow{\$} \mathcal{R}_Q^{(N)}$ and the error term \mathbf{e} from the error distribution E . Here $Q \geq 2$ is an integer modulus. The decisional version is to distinguish between the RLWE sample derived from some secret \mathbf{s} and a sample from uniform distribution over $(\mathcal{R}_Q^{(N)})^2$.

RLWE cryptosystem. The RLWE ciphertext of message $\mathbf{m} \in \mathcal{R}_P^{(N)}$ is defined as $\text{RLWE}_{\mathbf{s}, Q}(\mathbf{m}) = (\mathbf{a}, \mathbf{b}) \in (\mathcal{R}_Q^{(N)})^2$ where integer P is plaintext modulus and integer $Q > P$ is ciphertext modulus. We also use $\text{RLWE.ct}(\mathbf{m})$ for short. The polynomial \mathbf{a} is uniformly sampled from polynomial ring $\mathcal{R}_Q^{(N)}$, the secret key polynomial \mathbf{s} and error polynomial \mathbf{e} are small-norm polynomials and are sampled from some specific distributions respectively. For simplicity, this paper presents our scheme using a symmetric key scheme. The proposed approach is directly

applicable in an asymmetric key scheme. RLWE ciphertext (\mathbf{a}, \mathbf{b}) satisfies that $\mathbf{b} + \mathbf{a} \cdot \mathbf{s} = \left\lfloor \frac{Q}{P} \right\rfloor \mathbf{m} + \mathbf{e}$. The decryption of RLWE ciphertext is

$$\text{Dec}(\mathbf{s}, (\mathbf{a}, \mathbf{b})) = \left\lfloor \frac{\mathbf{b} + \mathbf{a} \cdot \mathbf{s} \pmod{Q}}{\lfloor Q/P \rfloor} \right\rfloor, \quad (1)$$

where the rounding function is applied to each coefficient in the polynomial.

2.2 Coefficient encoding and fast plaintext-ciphertext matrix multiplication

A way of encoding a vector from \mathbb{Z}_P^N a polynomial in the plaintext space $\mathcal{R}_P^{(N)}$ is to put all the integers as the coefficients of the polynomial:

$$\text{Ecd}_c(\cdot) : \mathbb{Z}_P^N \rightarrow \mathcal{R}_P^{(N)}; \mathbf{m} = [m_0, m_1, \dots, m_{N-1}] \mapsto \mathbf{m} := \sum_{i=0}^{N-1} m_i x^i.$$

We call it *coefficient encoding*, and we can verify that:

$$\text{Ecd}_c(\mathbf{m}_1 + \mathbf{m}_2) = \text{Ecd}_c(\mathbf{m}_1) + \text{Ecd}_c(\mathbf{m}_2) \pmod{P}.$$

$$\text{Ecd}_c(\beta \cdot \mathbf{m}) = \beta \cdot \text{Ecd}_c(\mathbf{m}) \pmod{P}, \text{ for } \beta \in \mathbb{Z}_P.$$

We also frequently apply its inverse: $\text{Ecd}_c^{-1}(\cdot) : \mathcal{R}_P^{(N)} \rightarrow \mathbb{Z}_P^N$. The inverse generates the coefficient vector of the input polynomial, from low degree term to high degree term. Using coefficient encoding, the line of works [1, 30] proposed a fast plaintext-ciphertext matrix multiplication. Let $M \in \mathbb{Z}_P^{l \times N}$ is a matrix and is encrypted into l RLWE ciphertexts $\text{RLWE}_{\mathbf{s}, Q}(\mathbf{m}_i) = (\mathbf{a}_i, \mathbf{b}_i) \in (\mathcal{R}_Q^{(N)})^2$, $\forall i \in [l]$. Here \mathbf{m}_i is the polynomial whose coefficients are the i -th row of M : $\mathbf{m}_i := \sum_{j=0}^{N-1} m_{ij} x^j$, i.e., applying $\text{Ecd}_c(\cdot)$ on the i -th row of M . Then, $\forall i \in [l]$:

$$\mathbf{b}_i + \mathbf{a}_i \cdot \mathbf{s} = \left\lfloor \frac{Q}{P} \right\rfloor \sum_{j=0}^{N-1} m_{ij} x^j + \mathbf{e}_i \pmod{Q}. \quad (2)$$

Write $\mathbf{s} = \sum_{j=0}^{N-1} s_j x^j$, $\mathbf{b}_i = \sum_{j=0}^{N-1} b_{ij} x^j$ and $\mathbf{a}_i = \sum_{j=0}^{N-1} a_{ij} x^j$. Following the notation in [1], let $\text{Toep}(\mathbf{s}) \in \mathbb{Z}_Q^{N \times N}$ be the matrix whose i -th row is the coefficients of polynomial $x^i \mathbf{s} \pmod{x^N + 1}$:

$$\text{Toep}(\mathbf{s}) = \begin{bmatrix} s_0 & s_1 & \dots & s_{N-1} \\ -s_{N-1} & s_0 & \dots & s_{N-2} \\ \dots & \dots & \dots & \dots \\ -s_1 & -s_2 & \dots & s_0 \end{bmatrix}$$

Then Equation 2 can be rewrite in vector and matrix form (Lemma 3 in [1]), omitting the error term \mathbf{e}_i :

$$[b_{i0}, b_{i1}, \dots, b_{i,N-1}] + [a_{i0}, a_{i1}, \dots, a_{i,N-1}] \text{Toep}(\mathbf{s}) \approx \left\lfloor \frac{Q}{P} \right\rfloor [m_{i0}, m_{i1}, \dots, m_{i,N-1}] \pmod{Q}.$$

Combining $\forall i \in [l]$ we can obtain:

$$B + A \cdot \text{Tope}(\mathfrak{s}) \approx \left\lfloor \frac{Q}{P} \right\rfloor M \pmod{Q}, \quad (3)$$

where the i -th row of $A \in \mathbb{Z}_Q^{l \times N}$ is the coefficients of \mathfrak{a}_i , i.e., $[a_{i0}, a_{i1}, \dots, a_{i,N-1}]$. $B \in \mathbb{Z}_Q^{l \times N}$ and $M \in \mathbb{Z}_P^{l \times N}$ have the same dimensions as A and are constructed in the same way. Then, multiplying a plaintext matrix $W \in \mathbb{Z}_P^{d \times l}$ with M can be obtained by

$$WB + WA \cdot \text{Tope}(\mathfrak{s}) \approx \left\lfloor \frac{Q}{P} \right\rfloor WM \pmod{Q}.$$

Then for $j \in [d]$, the j -th row of WB and WA form the RLWE ciphertext of the j -th row of WM , under the coefficient encoded $\text{Ecd}_c(\cdot)$. That is, (WB, WA) is effectively d RLWE ciphertexts which uses coefficient encoding. The error term is also multiplied by the plaintext matrix W . As stated in [1], both W and M should have entry with small absolute values compared with Q . When dealing with $W \in \mathbb{R}^{d \times l}$ (each entry has a small absolute value), [1] encodes each w_{ij} into \mathbb{Z} by a proper scaling factor.

2.3 Slot encoding and BFV scheme

There exists another way to encode vectors to elements of a polynomial ring. In this subsection we give the brief descriptions of the BFV scheme [14]. It is widely used in homomorphic integer arithmetic. Let P be the plaintext modulus and $Q > P$ be the ciphertext modulus. We apply the BFV setting when N is a power of 2, and use SEAL [44] as the implementation of BFV scheme. Then the plaintext space and ciphertext space are $\mathcal{R}_P^{(N)} = \mathbb{Z}_P[X]/(X^N + 1)$ and $\mathcal{R}_Q^{(N)} = \mathbb{Z}_Q[X]/(X^N + 1)$, respectively. We further require that $P \equiv 1 \pmod{2N}$. It is a variant of RLWE scheme and its ciphertexts have the same form $(\mathfrak{a}, \mathfrak{b}) \in (\mathcal{R}_Q^{(N)})^2$. The encryption/decryption procedures are the same as in the RLWE cryptosystem (Equation 1). We will use $\text{BFV}_{s,Q}(\mathfrak{m})$ to denote BFV ciphertext encrypting polynomial $\mathfrak{m} \in \mathcal{R}_P^{(N)}$ and use $\text{BFV.ct}(\mathfrak{m})$ for short. Below we introduce the main properties and functionalities of the encoding in BFV schemes and refer the details to, e.g., [14], [18, 21].

To encrypt a vector $\mathbf{m} \in [m_0, \dots, m_{N-1}] \in \mathbb{Z}_P^N$, BFV scheme constructs a polynomial $\tilde{\mathbf{m}} \in \mathcal{R}_P^{(N)}$ to encode vector \mathbf{m} .⁸ More concretely, let ζ be the $2N$ -th primitive root of unity of P .⁹ Write $\bar{\zeta} = \zeta^{-1} \pmod{P}$, $\zeta_j := \zeta^{5^j}$ and $\bar{\zeta}_j := \bar{\zeta}^{5^j}$. We set the polynomial $\tilde{\mathbf{m}}$ so that:

$$\forall i \in \{0, 1, \dots, N/2 - 1\}, m_i = \tilde{\mathbf{m}}(\zeta_i).$$

⁸ In this case where each slot only stores integer in \mathbb{Z}_P , [7] defined a simplified version of BFV bootstrapping called *thin bootstrapping*, and we focus on thin bootstrapping in this work. Please refer to [7] and [16] for more general BFV settings.

⁹ That is to say, $\zeta^{2N} = 1 \pmod{P}$ and $\forall 0 < l < 2N, \zeta^l \neq 1 \pmod{P}$.

$$\forall i \in \{N/2, \dots, N-1\}, m_i = \tilde{\mathbf{m}}(\bar{\zeta}_{i-N/2}).$$

We call $\tilde{\mathbf{m}}$ be the *slot encoding* of \mathbf{m} :

$$\text{Ecd}_s(\cdot) : \mathbb{Z}_P^N \rightarrow \mathcal{R}_P^{(N)}; \mathbf{m} = [m_0, m_1, \dots, m_{N-1}] \mapsto \tilde{\mathbf{m}}.$$

We use *plaintext slots* or simply *slots* to represent the entries of vector \mathbf{m} .

SIMD style addition and multiplication using slot encoding. Note that $\text{Ecd}_s(\cdot) : \mathbf{m} \mapsto \tilde{\mathbf{m}}$ is a ring isomorphism. According to Chinese remainder theorem ([45], [16]):

$$\tilde{\mathbf{m}}_1 + \tilde{\mathbf{m}}_2 = \text{Ecd}_s(\mathbf{m}_1 + \mathbf{m}_2) = \text{Ecd}_s([m_{1,0} + m_{2,0}, m_{1,1} + m_{2,1}, \dots, m_{1,N-1} + m_{2,N-1}]),$$

$$\tilde{\mathbf{m}}_1 \times \tilde{\mathbf{m}}_2 = \text{Ecd}_s(\mathbf{m}_1 \otimes \mathbf{m}_2) := \text{Ecd}_s([m_{1,0} \times m_{2,0}, m_{1,1} \times m_{2,1}, \dots, m_{1,N-1} \times m_{2,N-1}]).$$

$\mathbf{m}_1 \otimes \mathbf{m}_2$ stands for the Hadamard product (also called the element-wise product). Further, let ct_i be the ciphertext of polynomial $\tilde{\mathbf{m}}_i$, s.t. $\tilde{\mathbf{m}}_i = \text{Dec}(\text{ct}_i)$, $i = 1, 2$. There exist functions $\text{add}(\cdot, \cdot), \text{mul}(\cdot, \cdot) : (\mathcal{R}_Q^{(N)})^2 \times (\mathcal{R}_Q^{(N)})^2 \rightarrow (\mathcal{R}_Q^{(N)})^2$ which satisfy:

$$\text{Dec}(\text{add}(\text{ct}_1, \text{ct}_2)) = \tilde{\mathbf{m}}_1 + \tilde{\mathbf{m}}_2 = \text{Dec}(\text{ct}_1) + \text{Dec}(\text{ct}_2) = \text{Ecd}_s(\mathbf{m}_1 + \mathbf{m}_2),$$

$$\text{Dec}(\text{mul}(\text{ct}_1, \text{ct}_2)) = \tilde{\mathbf{m}}_1 \times \tilde{\mathbf{m}}_2 = \text{Dec}(\text{ct}_1) \times \text{Dec}(\text{ct}_2) = \text{Ecd}_s(\mathbf{m}_1 \otimes \mathbf{m}_2),$$

where $\text{Dec}(\cdot)$ is Equation 1. This allows Single-Instruction-Multiple-Data (SIMD) style evaluations on multiple encrypted data.

Polynomial evaluation. Let \mathbf{p} be an polynomial in $\mathbb{Z}_P[X]$, we are able to homomorphically evaluate $[\mathbf{p}(m_0), \mathbf{p}(m_1), \dots, \mathbf{p}(m_{N-1})] \pmod{P}$ using the above properties given a BFV ciphertext of $\text{Ecd}_s(\mathbf{m})$. We use $\text{Eval}(\mathbf{p}, \cdot)$ to denote this procedure.

The main difference between coefficient encoding and slot encoding is that SIMD style multiplication is not supported. In BFV scheme, the conversion between slot encoding and coefficient encoding are available. Given a vector \mathbf{m} , we will let \mathbf{m} be the coefficient encoding: $\text{Ecd}_c(\mathbf{m})$, and we will let $\tilde{\mathbf{m}}$ be the slot encoding: $\text{Ecd}_s(\mathbf{m})$.

Slot-to-Coefficient (S2C) Given the ciphertext BFV.ct which encrypts $\tilde{\mathbf{m}}$: the slot encoding of \mathbf{m} , the operation S2C generates another BFV.ct which encrypts the coefficient encoding of \mathbf{m} : i.e., $\text{Ecd}_c(\mathbf{m}) = \mathbf{m}$.

Coefficient-to-Slot (C2S) C2S is the reverse procedure of S2C. Given the ciphertext BFV.ct which encrypts \mathbf{m} : the coefficient encoding of \mathbf{m} , the operation C2S generates another BFV.ct which encrypts the slot encoding of \mathbf{m} : i.e., $\text{Ecd}_s(\mathbf{m}) = \tilde{\mathbf{m}}$.

2.4 Challenges in secure inference of transformer architecture

The transformer architecture [48] is now the most popular AI model architecture. It is composed by matrix multiplications and non-polynomial functions which will be approximated by polynomial. Then there is a problem of choosing coefficient encoding or slot encoding.

As discussed in Section 2.2, using coefficient encoding allows us to evaluate very fast plaintext-ciphertext matrix multiplication (PCMM). Evaluating multiplication between plaintext W and encrypted M can be obtained by two plaintext-plaintext matrix multiplication WB and WA . According to [1], plaintext matrix multiplications can be further accelerated by high-performance Basic Linear Algebra Subprograms (BLAS) libraries. On the contrary, almost all previous works focus on slot encoding and evaluate matrix multiplication on slots (e.g., [18, 19, 21, 25]). Using slot encoding cannot make use of the above fast PCMM. These works use Linear Transformation (LT) which requires frequently "rotating" the data on the encoded vector, and is much slower.

However, using coefficient encoding and fast PCMM has two limitations. First, when the ring degree N is determined, the time performance of this matrix multiplication depends heavily on the ciphertext modulus Q as we are computing WA and WB where $A, B \in \mathbb{Z}_Q^{l \times N}$. When Q is as large as thousands of bits (which is very common in BFV schemes), the computational overhead grows rapidly. Therefore [1] propose the fast matrix multiplication for Q below 80 bits.

Second, when applying the new PCMM to existing slot-encoded CKKS frameworks like NEXUS [49], the latency of matrix multiplication is reduced, but it introduces C2S and S2C to transform data between slot encoding and coefficient encoding. It will bring extra time cost and offset much of the performance gain achieved by the new PCMM. Although the new PCMM is much faster than the traditional slot encoding matrix multiplication, it is difficult to apply it to existing works about privacy-preserving inference of transformers.

3 Full flow of LEAF

In this section, we introduce the full flow of LEAF for feedforward block in privacy-preserving transformer inference. LEAF consists of a fast coefficient encoded matrix multiplication algorithm, and an asymptotically efficient non-polynomial function evaluation algorithm which is compatible with the coefficient encoding framework. This novel framework (1) removes costly data movement across ciphertext slots, (2) Combines function evaluation with bootstrapping, reducing the size of ciphertext modulus and eliminating the need for separate bootstrapping procedure. Together, these innovations enable a $100\times$ improvement in total latency over the state-of-the-art implementation (NEXUS [49]) in the same environment.

Our matrix multiplication component adopts the fast PCMM proposed in [1], which was briefly reviewed in subsection 2.2. For a comprehensive description of the algorithm and its performance characteristics, we refer the reader to [1]. In this section, we focus on presenting our evaluation algorithm for non-polynomial activation functions, and how it integrates with the fast PCMM from [1], which outputs coefficient encoded RLWE ciphertexts.

We use lowercase letters p, q to denote the plaintext modulus, ciphertext modulus respectively in the RLWE scheme which uses coefficient encoding. We use n to denote the degree of polynomial ring in the RLWE scheme. We use

uppercase letters $P, Q > q, N > n$ denote the plaintext modulus, ciphertext modulus and degree of polynomial ring respectively in the BFV scheme which uses slot encoding. Let $\underline{s} \in \mathbb{Z}_q[X]/(X^n + 1)$ be the secret key of the RLWE scheme. Let $\mathfrak{s} \in \mathbb{Z}_Q[X]/(X^N + 1)$ be the secret key of the BFV scheme.

3.1 Overview

Our non-polynomial evaluation algorithm takes d low-level coefficient encoded RLWE ciphertexts procedured by PCMM [1] as input, and applies modulus switching to switch the ciphertext modulus to a small q :

$$i \in [d], \text{RLWE}_{\underline{s}, q}(\text{Ecd}_c(\mathbf{m}_i = [m_{i0}, \dots, m_{in}])) ,$$

and then evaluate a non-polynomial activation and refresh ciphertexts. Here $g(\cdot) : \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ is the function we want to evaluate on each entry in \mathbf{m}_i . Finally we output $\text{ct}'_i = \text{RLWE}_{\underline{s}, q}(\text{Ecd}_c([g(m_{i0}), \dots, g(m_{in})])), i \in [d]$ where ct'_i is also a coefficient encoded ciphertext and can be input to the next matrix multiplication.

Main idea. The idea comes from two observations. First, the decryption of one element in the message vector \mathbf{m}_i can be written as a mod- q linear operation between a vector extracted from the ciphertext polynomials and a vector consists of the coefficients of the secret key. If we set q to the plaintext modulus of BFV, i.e., $q = P$, and encrypt the “RLWE secret key coefficient vector” (from $\text{Ecd}_c^{-1}(\mathfrak{s})$) by BFV to get a high-level slot encoding BFV ciphertext, then we can homomorphically evaluate decryption formulas to get a high-level slot encoding BFV ciphertext of the noisy message by Linear Transformation (LT). In brief, LT [18, 21] evaluates the multiplication between a matrix C and an encrypted column vector \mathbf{x}^T . The output is a BFV ciphertext encrypting $\text{Ecd}_s(C\mathbf{x}^T)$.

Second, when evaluating a $P - 1$ degree polynomial derived through interpolation (e.g., [24, 31]) of a function f from $\mathbb{Z}_P \rightarrow \mathbb{Z}_P$ which maps the noisy message to a refreshed message, we can reduce the noise in all slots simultaneously by the SIMD property of BFV ciphertext. More concretely, according to the decryption algorithm of RLWE (Equation 1), the noise term is removed by a “mod-then-floor” procedure, which can be seen as a function from $\mathbb{Z}_P \rightarrow \mathbb{Z}_P$. Furthermore, since f can be an arbitrary function from $\mathbb{Z}_P \rightarrow \mathbb{Z}_P$, we can define f that takes the noisy message as input and returns the result of an arbitrary function $g(\cdot)$ of the refreshed message. We will elaborate the details of this in Section 3.3.

In this section, we will introduce the details and also the optimizations we proposed to improve the performance. Our algorithm consists of the following steps:

- (*Coefficient encoded RLWE ciphertexts \rightarrow slot encoded BFV ciphertext*)
This step switches N/n low-level coefficient encoded RLWE ciphertexts to one high-level slot encoded BFV ciphertexts. Notice that this step is different with the C2S procedure. Since the parameters of the RLWE scheme and BFV scheme are different and the RLWE ciphertexts are in lower level, we cannot directly apply C2S of BFV here. We propose the method in Section 3.2.

- (Evaluating arbitrary function on slot encoded BFV ciphertext)
It is evaluating a $P - 1$ degree polynomial $fpoly(\cdot)$ on BFV ciphertext, where P is the BFV plaintext space. $fpoly(\cdot)$ is constructed according to the arbitrary function $g(\cdot)$ that we want to evaluate, such as the non-polynomial activation function. BFV ciphertext is in slot encoding so it supports SIMD style polynomial evaluations. We will elaborate the steps and constructions of $fpoly(\cdot)$ in Section 3.3.
- (Slot encoded BFV ciphertext \rightarrow coefficient encoded BFV ciphertexts)
In Section 3.4, we perform S2C on the slot encoded BFV ciphertext to get one coefficient encoded BFV ciphertext.
- (Coefficient encoded BFV ciphertext \rightarrow coefficient encoded RLWE ciphertexts)

By performing modulus switching and ring switching we obtain N/n coefficient encoded RLWE ciphertexts whose parameters are the same as the input RLWE ciphertexts. After that, these RLWE ciphertexts are ready for the following plaintext-ciphertext matrix multiplication or other HE evaluations. The details are in Section 3.5.

Figure 2 shows the flow of non-polynomial evaluation in LEAF.¹⁰ And the detailed algorithm is included in Algorithm 4.

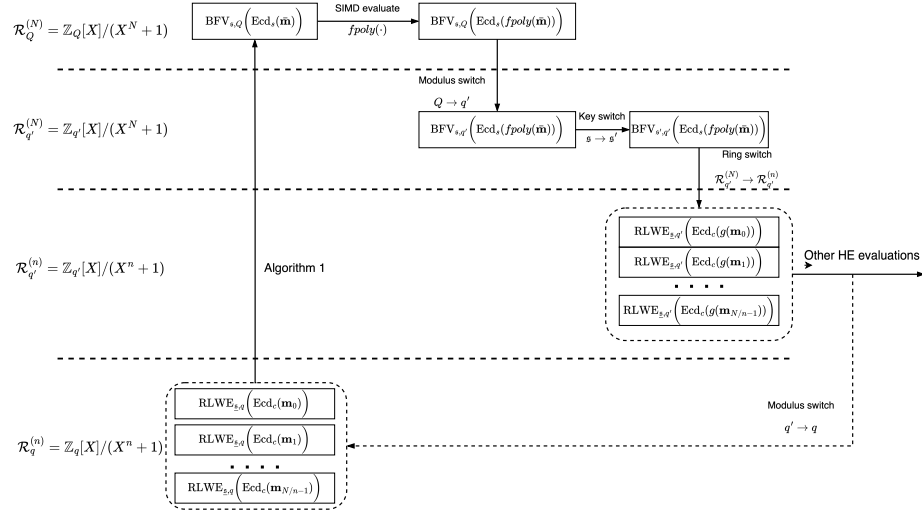


Fig. 2: Non-polynomial evaluation of LEAF. The ciphertexts are refreshed and evaluated by function $g(\cdot) : \mathbb{Z}_P \rightarrow \mathbb{Z}_P$.

Noise analysis. According to [36], the analysis follows the widely used heuristic assumption that the coefficients of each polynomial behave like independent

¹⁰ Other HE evaluations can include PCMM, addition, multiplication, etc.

zero-mean random variables of the same variance [9], and central limit heuristic [13]. We will include a lemma about the *variance* of (the coefficients of) the noise term in each subsection. And we refer the detailed proof of these lemmas to Appendix B. This variance can be used in *Chebyshev's inequality*: $\Pr[|X - EX| \geq b] \leq \frac{\sigma^2}{b^2}$ to bound noise term. For correct decryption with overwhelming probability, the noise term must be bounded by $\left\lfloor \frac{Q}{2P} \right\rfloor$ given ciphertext modulus Q and plaintext modulus P . Therefore, the variance of the noise term should be bounded by

$$\sigma^2 \ll \left\lfloor \frac{Q}{2P} \right\rfloor^2 \approx \frac{Q^2}{4P^2} . \quad (4)$$

3.2 From coefficient encoded RLWE to slot encoded BFV

In this section, we show the algorithm that converts low-level coefficient encoded RLWE ciphertexts to high-level slot encoded BFV ciphertexts.

Let a coefficient encoded RLWE ciphertext $(\mathbf{a}, \mathbf{b}) \in (\mathcal{R}_q^{(n)})^2$ encrypt a polynomial $\text{Ecd}_c(\mathbf{m})$ under secret key $\underline{\mathbf{s}}$ and let the noise term be \mathbf{e} . let vector $\mathbf{a} := \text{Ecd}_c^{-1}(\mathbf{a}) \in \mathbb{Z}_q^n$. Similarly we can have $\mathbf{e}, \mathbf{b}, \underline{\mathbf{s}} \in \mathbb{Z}_q^n$. The i -th element of vector \mathbf{m} is m_i and we have $\alpha m_i + e_i = b_i + \text{Ecd}_c^{-1}(\mathbf{a}\underline{\mathbf{s}})[i] \bmod q$, where $\alpha = \lfloor \frac{q}{p} \rfloor$. Notice that $\mathbf{a}\underline{\mathbf{s}}$ is a polynomial multiplication in the polynomial ring $\mathcal{R}_q^{(n)}$, so the coefficient of x^i in $\mathbf{a}\underline{\mathbf{s}}$ is $\sum_{j=0}^i (a_{i-j}\underline{s}_j) + \sum_{j=i+1}^{n-1} (-a_{n+i-j}\underline{s}_j) \bmod q$. If we take it as an inner product, we get

$$\alpha m_i + e_i = b_i + \langle \mathbf{a}^{(i)}, \underline{\mathbf{s}} \rangle \bmod q, \quad i \in [n], \quad (5)$$

where $\mathbf{a}^{(i)} = [a_i, a_{i-1}, \dots, a_0, -a_{n-1}, -a_{n-2}, \dots, -a_{i+1}]$ can be obtained by the permutation of coefficients in polynomial \mathbf{a} .

Our packing technique is from the observation that $b_i + \langle \mathbf{a}^{(i)}, \underline{\mathbf{s}} \rangle$ is a linear evaluation. If vector $\underline{\mathbf{s}}$ is slot encoded and is encrypted to a BFV ciphertext, then $b_i + \langle \mathbf{a}^{(i)}, \underline{\mathbf{s}} \rangle$ can be evaluated in slots. Note that the length of $\underline{\mathbf{s}}$ is n , the dimension in RLWE cryptosystem, which is below the degree N in BFV cryptosystem.¹¹ As both n and N we used are power-of-2, so $n|N$.

Now we proceed to extend Equation 5 to multiple RLWE ciphertexts. Define the packing key $\text{BFV.ct}_{\text{packing}} := \text{BFV}_{\mathbf{s}, Q}(\text{Ecd}_s([\underline{\mathbf{s}}, \underline{\mathbf{s}}, \dots, \underline{\mathbf{s}}]))$ by repeating $\underline{\mathbf{s}}$ for N/n times in N plaintext slots, where \mathbf{s} is the BFV secret key. Algorithm 1 we show how to pack N/n RLWE ciphertexts $\{\text{RLWE}_{\underline{\mathbf{s}}, q}(\text{Ecd}_c(\mathbf{m}_t))\}_{t \in [N/n]}$ into one BFV ciphertext.

It is noteworthy to point out that A is packing data in a specific order. For $i \in [N/n]$, the sub-block $A_i \in \mathbb{Z}_P^{n \times n}$ is obtained by collecting $\mathbf{a}^{(i)}$ in Equation 5 from all the N/n input RLWE ciphertexts. The purpose of this order is fit the ring splitting algorithm, as elaborated in Section 3.5. Figure 3 shows how to pack $\{\mathbf{m}_t\}_{t \in [N/n]}$ in one N -dimension vector.

¹¹ For example, n is assigned to be at most 2^{13} in [1], while popular parameter settings of BFV usually choose $N = 2^{15}$ or 2^{16} .

Algorithm 1 Pack N/n RLWE ciphertexts to 1 BFV ciphertext.

Input:

N/n RLWE ciphertexts $\{\text{RLWE}_{\underline{s},q}(\text{Ecd}_c(\mathbf{m}_t)) = (\mathbf{a}_t, \mathbf{b}_t)\}_{t \in [N/n]}$.
Packing key $\text{BFV.ct}_{\text{packing}} := \text{BFV}_{\underline{s},Q}(\text{Ecd}_s([\underline{s}, \underline{s}, \dots, \underline{s}]))$.

Output: BFV ciphertext $\text{BFV}_{\underline{s},Q}(\text{Ecd}_s(\bar{\mathbf{m}}))$ for some $\bar{\mathbf{m}} \in \mathbb{Z}_P^N$.

1: Construct matrix:

$$A \in \mathbb{Z}_P^{N \times N}, A := \begin{bmatrix} A_0 & 0 & \dots & 0 \\ 0 & A_1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & A_{N/n-1} \end{bmatrix}, \quad (6)$$

where for $i \in [N/n]$, each $A_i \in \mathbb{Z}_P^{n \times n}$, and the t -th row in A_i is $\mathbf{a}_t^{(i)} = [a_{ti}, a_{t,i-1}, \dots, a_{t0}, -a_{t,n-1}, -a_{t,n-2}, \dots, -a_{t,i+1}]$ derived from \mathbf{a}_t in the t -th input RLWE ciphertext.

2: Construct vector $\bar{\mathbf{b}} \in \mathbb{Z}_P^N$: $\bar{\mathbf{b}} := [\bar{\mathbf{b}}_0, \bar{\mathbf{b}}_1, \dots, \bar{\mathbf{b}}_{N/n-1}]$ s.t. the t -th entry in $\bar{\mathbf{b}}_i$ is the i -th coefficient of \mathbf{b}_t .

3: Return $\text{BFV}_{\underline{s},Q}(\text{Ecd}_s(\bar{\mathbf{m}})) = \text{Ecd}_s(\bar{\mathbf{b}}) + LT(A, \text{BFV.ct}_{\text{packing}})$.

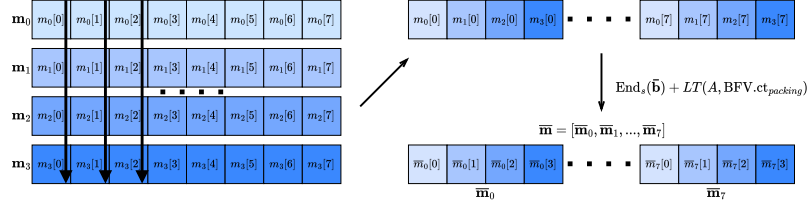


Fig. 3: The relationship between $\{\mathbf{m}_t\}_{t \in [N/n]}$ and $\bar{\mathbf{m}}$.

The *linear transformation*, $LT(A, \cdot)$, generates a BFV ciphertext encrypting the multiplication between matrix A and the input encrypted vector, using slot encoding. Therefore, if we write $\bar{\mathbf{m}} = [\bar{\mathbf{m}}_0, \bar{\mathbf{m}}_1, \dots, \bar{\mathbf{m}}_{N/n-1}]$, then we have $\bar{\mathbf{m}}_i = \bar{\mathbf{b}}_i + \underline{s}A_i \pmod q$ on slots. And the t -th entry of $\bar{\mathbf{m}}_i$ is:

$$\bar{m}_i[t] = \bar{b}_i[t] + \langle \mathbf{a}_t^{(i)}, \underline{s} \rangle = \alpha m_t[i] + e_t[i] \pmod q. \quad (7)$$

A key observation of accelerating LT is that the square matrix A in Equation 6 has a special structure: $A = \begin{bmatrix} A_{00} & 0 \\ 0 & A_{11} \end{bmatrix}$. It requires much fewer rotation processes, compared the case of general matrix $A' \in \mathbb{Z}_P^{N \times N}$. We refer our detailed implementations of linear transformation to Appendix A, Algorithm 5.

The analysis of the noise term is in Lemma 1.

Lemma 1. Let $\sigma_{\text{packing}K}^2$ be the variance of noise term in generating the packing key. Let $\sigma_{\text{rot}K}^2$ be the variance of noise term in generating the rotation key. Then the variance of noise term in the output of Algorithm 1 ($\text{BFV}_{\underline{s},Q}(\text{Ecd}_s(\bar{\mathbf{m}}))$),

denoted as σ_1^2 , is bounded by:

$$\sigma_1^2 \leq N^2 P^2 \sigma_{\text{packing}K}^2 + \frac{B^2}{12} N^3 P^2 \lfloor \log_B Q \rfloor \sigma_{\text{rot}K}^2 .$$

RNS form and NTT algorithm. We use Residual Numbering System (RNS) [18, 29] and Number Theoretic Transform (NTT) [22] to optimize the polynomial operations over the polynomial ring in LT as follows. RNS can handle the integer arithmetic that exceed the machine word size (e.g., over 64 bits). It is based on the Chinese remainder theorem and it decompose huge integers into smaller integers within 64 bits. NTT can help to convert the expensive polynomial multiplications to fast element-wise vector multiplications.

- We transform each polynomial encoding a vector from matrix A to NTT domain by David Harvey’s NTT algorithm [22].
- In baby steps, we transfer the rotated ciphertexts to NTT domain by David Harvey’s NTT algorithm.
- Therefore, the computations in giant step are in NTT domain, so the multiplications can be done in NTT domain directly, which is much faster than in polynomial domain.
- After computing multiplications in giant step, we transfer ciphertexts back from NTT domain to polynomial domain by applying the inverse of David Harvey’s NTT. Rest computations are in the polynomial domain and it returns the result ciphertext in polynomial form.

Details of optimized LT algorithm is shown in Appendix A, Algorithm 6. This optimization saves 40% time compared to Algorithm 5.

To further accelerate LT, we implement a *multi-thread enabled LT* to make use of powerful modern multi-core CPU, and the experimental results show that at least 72% time is saved. It is noteworthy to point out that all the above optimizations will not change the variance of the noise term.

3.3 SIMD polynomial evaluation on slot encoded BFV ciphertext

In this section, we first present the construction of the polynomial $fpoly(x)$, then propose our optimized homomorphic polynomial evaluation algorithm, together with its asymptotic analysis.

Polynomial construction The purpose of bootstrapping in homomorphic encryption schemes is to refresh the ciphertext and reduce the noise scale, so that it is feasible to perform more homomorphic evaluations on the ciphertext. In general, the bootstrapping procedure is done by homomorphically evaluate the decryption algorithm. According to the coefficient encoded RLWE ciphertext, the decryption algorithm of each $\bar{m}_i[t] \in \mathbb{Z}_p$ in Equation 7 is:

$$\left\lfloor \frac{\bar{m}_i[t]}{\alpha} \right\rfloor = \left\lfloor \frac{\alpha m_t[i] + e_t[i]}{\alpha} \right\rfloor = \lfloor m_t[i] + e_t[i]/\alpha \rfloor , \quad (8)$$

where $\alpha = \lfloor q/p \rfloor$ and the decryption is correct when $|e_t[i]| < \alpha/2 = \lfloor \frac{q}{2p} \rfloor$.

Therefore, our main task of bootstrapping is to homomorphically evaluate Equation 8, on the BFV ciphertext encrypting $(\text{Ecd}_s(\tilde{\mathbf{m}}))$. [32] proposes a special polynomial $f_{poly}(x)$ on \mathbb{Z}_P s.t. $f_{poly}(\tilde{m}_i) = \alpha m_i$, and sets $P = q$, i.e. the plaintext modulus of BFV is the same as the ciphertext modulus of RLWE. Such a polynomial $f_{poly}(x)$ always exists on $\mathcal{R}_P^{(N)} = \mathbb{Z}_P[X]/(X^N + 1)$ for prime P , as shown in Lemma 2.

Lemma 2 ([24]). *Let P be a prime. For any function $f : \mathbb{Z}_P \rightarrow \mathbb{Z}_P$, one can construct a polynomial $f_{poly}(x) \in \mathbb{Z}_P[X]$ s.t. $\forall x \in \mathbb{Z}_P, f(x) = f_{poly}(x)$:*

$$f_{poly}(x) := f(0) - \sum_{i=1}^{P-1} x^i \sum_{a=0}^{P-1} f(a) a^{P-1-i}.$$

By choosing $f(x) = \alpha \lfloor \frac{x}{\alpha} \rfloor$ and then the related $f_{poly}(x)$ satisfies $f_{poly}(\tilde{m}_i[t]) = \alpha m_t[i]$. Given $|e_t[i]| < \lfloor \frac{q}{2p} \rfloor$, the noise term e_i is removed and thus the ciphertext is refreshed.

Note that $f(x)$ can be arbitrary function from \mathbb{Z}_P to \mathbb{Z}_P . Thus, for any function $g(x)$ from \mathbb{Z}_p to \mathbb{Z}_p , we can choose $f(x) = \alpha g(\lfloor x/\alpha \rfloor)$ and use Lemma 2 to construct $f_{poly}(x)$. Then we have

$$f_{poly}(\tilde{m}_i[t]) = f(\alpha m_t[i] + e_t[i]) = \alpha g\left(\left\lfloor \frac{\alpha m_t[i] + e_t[i]}{\alpha} \right\rfloor\right) = \alpha g(m_t[i]),$$

given $|e_t[i]| < \lfloor \frac{q}{2p} \rfloor$.

Real value functions Further, let $r(x) : [-B, B] \rightarrow [-B, B]$ with $\Delta B \leq P/2$, and let \mathbb{Z}_p contains $-p/2, \dots, 0, 1, \dots, p/2 - 1$. We can construct $g(x) : \mathbb{Z}_p \rightarrow \mathbb{Z}_p$, $x \mapsto \lfloor \Delta r(x/\Delta) \rfloor$. This gives an evaluation of real function $r(x)$ on discrete points.

Optimized homomorphic polynomial evaluation

Paterson-Stockmeyer algorithm [42]. Recall that in subsection 2.3, we are able to homomorphically evaluate any polynomial $\mathbf{p} \in \mathbb{Z}_P[X]$ on each plaintext slot, using the SIMD style $\text{add}(\cdot, \cdot)$ and $\text{mul}(\cdot, \cdot)$ of BFV scheme. However, the homomorphic multiplication function $\text{mul}(\cdot, \cdot)$ between two ciphertexts is slow and is one of the main bottleneck of FHE schemes. As a result, instead of naively evaluating an polynomial, it is important and necessary to reduce the number of *ciphertext-ciphertext* multiplications. The well-known Paterson-Stockmeyer algorithm [42] has the desired property. It was designed to accelerate the computation of matrix polynomial by reducing the number of the expensive non-scalar multiplication (e.g., matrix multiplication). It needs only $O(\sqrt{d})$ non-scalar multiplications to evaluate a polynomial with degree d . In the homomorphic evaluation scenario, Paterson-Stockmeyer only needs $O(\sqrt{d})$ ciphertext-ciphertext

multiplications. Thus it is widely used in homomorphic schemes to evaluate polynomial with encrypted input (e.g., [32, 36]). For completeness, we include Paterson-Stockmeyer algorithm in Algorithm 2.

Algorithm 2 Paterson-Stockmeyer algorithm for polynomial evaluation [42].

- 1: **Input:** A degree- $(d-1)$ polynomial $\mathbf{p} = a_{d-1}X^{d-1} + \dots + a_1X + a_0$. Data x to be evaluated.
- 2: **Output:** $\mathbf{p}(x)$.
- 3: Let $k := \lceil d \rceil$. Compute $\{x, x^2, \dots, x^k\}$.
- 4: Use x^k to compute $\{x^{2k}, x^{3k}, \dots, x^{k^2-k}\}$.
- 5: Rewrite $\mathbf{poly}(x)$ in the following form using $\{x, x^2, \dots, x^k\} \cup \{x^{2k}, x^{3k}, \dots, x^{k^2-k}\}$:

$$\begin{aligned} \mathbf{p}(x) = & (a_0 + a_1x + \dots + a_{k-1}x^{k-1}) \\ & + (a_k + a_{k+1}x + \dots + a_{2k-1}x^{k-1})x^k \\ & + \dots + (a_{d-k} + a_{d-k+1}x + \dots + a_{d-1}x^{k-1})x^{k^2-k} . \end{aligned}$$

- 6: Evaluate each line of above and return the summation of all the results.
-

Paralleled polynomial evaluation algorithm. Note that in Algorithm 2, most non-scalar multiplications are in line 3 and line 4 and they have a large portion of the time cost of Algorithm 2. Current implementations (e.g., [32]) do not apply optimization on them and only naively compute $\{x, x^2, \dots, x^k\}$ and $\{x^{2k}, x^{3k}, \dots, x^{k^2-k}\}$. We observe that if we can implement these two steps in multi-thread, the efficiency of homomorphic polynomial evaluation can be hugely improved. The concrete algorithm is proposed in Algorithm 3. Take $k = 256$ as a

Algorithm 3 $Power(x, k)$: Multi-thread computation of $\{x, x^2, \dots, x^k\}$.

- 1: **Input:** Positive integer k , data x .
 - 2: **Output:** $\{x, x^2, \dots, x^k\}$.
 - 3: Define set $D_l = \{x, x^2, \dots, x^l\}$. Initially $D_1 = \{x\}$.
 - 4: **while** $l < k$ **do**
 - 5: $l := l \times 2$.
 - 6: Use multi-thread to compute $D_l = (D_{l/2} \times x^{l/2}) \cup D_{l/2}$, where $D_{l/2} \times x^{l/2}$ is to apply multiplication on each element of $D_{l/2}$.
 - 7: **end while**
 - 8: Return $\{x, x^2, \dots, x^k\} \subseteq D_{2^{\lceil \log k \rceil}}$.
-

toy example in $Power(x, k)$. At first we have $D_1 = \{x\}$, $D_2 = \{\{x\} \times x\} \cup \{x\} = \{x^2, x\}$. Moving forward we can obtain $D_4 = \{x, x^2, x^3, x^4\}$, ..., and finally $\{x, x^2, \dots, x^{256}\}$.

Asymptotic analysis. Using Algorithm 2 can homomorphically evaluate degree $(d-1)$ -polynomial using only $O(\sqrt{d})$ ciphertext multiplications. Previous implementations of line 3 and line 4 in Algorithm 2 (e.g., [32,36]) compute x, x^2, \dots, x^k sequentially, and do not make use of parallel processing. One core has to perform $O(\sqrt{d})$ ciphertext multiplications.

We propose Algorithm 3 that optimizes computing line Algorithm 3 and line 4 in Algorithm 2. Assume that we have a t -core CPU, then each core only needs to do $O(\log t + k/t) = O(\log t + \sqrt{d}/t)$ ciphertext multiplications.¹²

The analysis of the noise term is in Lemma 3.

Lemma 3. *Let the input of our Paterson-Stockmeyer algorithm (Algorithm 2) be from Algorithm 1, whose noise term has variance σ_1^2 . Let the polynomial being evaluated has degree $P-1$. Then the variance of noise term in the output of Algorithm 2, denoted as σ_2^2 , can be bounded by $o(\frac{Q^2}{4P^2})$ given Q supports $\lceil \log \sqrt{P} \rceil + 3$ multiplication level.*

3.4 From Slot-based BFV ct to Coeff-based BFV ct

The process Slot-to-Coefficient (S2C) is widely used in different FHE schemes (e.g., [15,16]). Given a slot encoded BFV ciphertext of $\text{Ecd}_s(\mathbf{m} = [m_0, m_1, \dots, m_{N-1}]^T)$, S2C process outputs a BFV ciphertext encrypting $\text{Ecd}_c(\mathbf{m}) = \mathbf{m} = \sum_{i=0}^{N-1} m_i x^i$. It moves all the elements in plaintext slots to polynomial coefficients. Recall the definition of $\text{Ecd}_s(\mathbf{m}) = \tilde{\mathbf{m}} = \sum_{i=0}^{N-1} \tilde{m}_i x^i$, we have for $r \in [N/2]$:

$$m_r = \tilde{\mathbf{m}}(\zeta_r) = \sum_{i=0}^{N-1} \tilde{m}_i \zeta_r^i, \quad m_{r+N/2} = \tilde{\mathbf{m}}(\bar{\zeta}_r) = \sum_{i=0}^{N-1} \tilde{m}_i \bar{\zeta}_r^i. \quad (9)$$

We can rewrite Equation 9 to $\mathbf{m} = U[\tilde{m}_0, \tilde{m}_1, \dots, \tilde{m}_{N-1}]^T$ by setting:

$$U = \begin{bmatrix} 1 & \zeta_0 & \dots & \zeta_0^{N-1} \\ 1 & \zeta_1 & \dots & \zeta_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \dots & \zeta_{N/2-1}^{N-1} \\ 1 & \bar{\zeta}_0 & \dots & \bar{\zeta}_0^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \bar{\zeta}_{N/2-1} & \dots & \bar{\zeta}_{N/2-1}^{N-1} \end{bmatrix} \in \mathbb{Z}_P^{N \times N}. \quad (10)$$

¹² We can verify this by checking each iteration in the while loop. For some l , in that round $l/2$ ciphertext multiplications are required, and they can be evaluated independently. In the round where $l/2 \leq t$, each core performs at most one ciphertext multiplication. In the round where $l/2 > t$, each core performs $O(\frac{l}{2t})$ ciphertext multiplications. By summing up, we obtain a total number of ciphertext multiplications $O(\log t) + O(k/t) = O(\log t + \sqrt{d}/t)$ as by definition $k = \lceil \sqrt{d} \rceil$.

Then the S2C process outputs a slot encoded BFV ciphertext encrypting $\text{Ecd}_s(U\mathbf{m}^T)$. This can be done by LT using $\text{BFV.ct}(\text{Ecd}_s(\mathbf{m}))$ and plaintext matrix U . Instead of storing U in matrix form, we can pre-compute the encoding of U , which can save a lot of time in the LT algorithm. We refer the detailed slot encoding algorithm of U to Algorithm 8 in Appendix A.3. Let the result of Algorithm 8 be $\text{Ecd}_{LT}(U)$, which contains N BFV ciphertext encrypting the (upper) diagonals of U in slot encoding.

Combining all the optimizations, our optimized linear transformation algorithm for S2C process is shown in Algorithm 9 in Appendix A.3. Compared with the state-of-the-art S2C implementation [31], our optimized S2C process reduced the latency from 137s to 15s when running in the same machine with same parameters (Both are evaluated in single-thread setting).

The analysis of the noise term is in Lemma 4.

Lemma 4. *Let the input of our S2C process be from Algorithm 2, whose noise term has variance σ_2^2 . The variance of noise term in the output of S2C, denoted as σ_3^2 , is bounded by:*

$$\sigma_3^2 := N^2 P^2 \sigma_2^2 + (\sqrt{N/2} + N^2 P^2) \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma_{rotK}^2 .$$

3.5 From one coefficient encoded BFV to N/n coefficient encoded RLWE ct

After the S2C process, we obtain a BFV ciphertext in coefficient encoding. Let this BFV ciphertext be $\text{BFV}_{\mathbf{s},Q}(\text{Ecd}_c(\mathbf{m}))$ for some vector $\mathbf{m} \in \mathbb{Z}_P^N$. In this section we split $\text{BFV}_{\mathbf{s},Q}(\text{Ecd}_c(\mathbf{m}))$ into N/n coefficient encoded RLWE ciphertext $\{\text{RLWE}_{\underline{\mathbf{s}},q}(\text{Ecd}_c(\mathbf{m}_t))\}_{t \in [N/n]}$. In brief, the flow in this section is:

$$\begin{aligned} \text{BFV}_{\mathbf{s},Q}(\text{Ecd}_c(\mathbf{m})) &\rightarrow \text{BFV}_{\mathbf{s},q'}(\text{Ecd}_c(\mathbf{m})) \rightarrow \\ &\text{BFV}_{\mathbf{s}',q'}(\text{Ecd}_c(\mathbf{m})) \rightarrow \{\text{RLWE}_{\underline{\mathbf{s}},q'}(\text{Ecd}_c(\mathbf{m}_t))\}_{t \in [N/n]} . \end{aligned}$$

Modulus switch. The first arrow, modulus switch, converts a BFV ciphertext (\mathbf{a}, \mathbf{b}) with ciphertext modulus Q into another one with ciphertext modulus $q' < Q$. The modulus switch is simply done by $(\mathbf{a}, \mathbf{b}) \mapsto (\lfloor q'\mathbf{a}/Q \rfloor, \lfloor q'\mathbf{b}/Q \rfloor)$.

Ring Switch. The second arrow is key switch and ring switch from [2]. Define the *key switching key* $ks_{\mathbf{s} \rightarrow \mathbf{s}'} := \text{BFV.ct}_{\mathbf{s}',Q}(\mathbf{s})$, i.e., the BFV ciphertext encrypting the previous secret key $\mathbf{s} = \sum_{i=0}^{N-1} s[i]x^i \in \mathcal{R}_P^{(N)}$ using the new secret key. The new secret key, $\mathbf{s}' := \sum_{i=0}^{N-1} s'[i]x^i \in \mathcal{R}_P^{(N)}$, is derived from the RLWE secret key $\underline{\mathbf{s}} = \sum_{i=0}^{n-1} \underline{s}[i]x^i \in \mathcal{R}_{q'}^{(n)}$. \mathbf{s}' has coefficients that satisfy:

$$s'[i] = \underline{s}[i/(N/n)], \text{ for } i \bmod N/n = 0; \quad s'[i] = 0, \text{ otherwise.}$$

The key switching key is pre-generated in the set-up process. Using the key switching algorithm, we can convert $\text{BFV}_{\mathbf{s},Q}(\text{Ecd}_c(\mathbf{m}))$ to $\text{BFV}_{\mathbf{s}',Q}(\text{Ecd}_c(\mathbf{m}))$. We refer the detailed key switching algorithm to [2].

For the last arrow, we have to convert the ciphertext (\mathbf{a}, \mathbf{b}) from $(\mathcal{R}_{q'}^{(N)})^2$ to N/n RLWE ciphertexts $\{(\mathbf{a}_t, \mathbf{b}_t)\}_{t \in [N/n]}$ from $(\mathcal{R}_{q'}^{(n)})^2$. In brief, the ring switching algorithm is re-arranging all the coefficients in (\mathbf{a}, \mathbf{b}) to the coefficients of $\{(\mathbf{a}_t, \mathbf{b}_t)\}_{t \in [N/n]}$. Let $\mathbf{a} := \sum_{i=0}^{N-1} a[i]x^i$. We can arrange the coefficients of $\mathbf{a}_t := \sum_{i=0}^{n-1} a_t[i]x^i$ as $a_t[i] = a[iN/n + t]$. $\mathbf{b}_t := \sum_{i=0}^{n-1} b_t[i]x^i, \forall t \in [N/n]$ is derived from \mathbf{b} in the same way.

The analysis of the noise term is in Lemma 5.

Lemma 5. *Let the BFV ciphertext from S2C have noise term with variance σ_3^2 . The variance of noise term after ring switching, denoted as σ_4^2 , is bounded by:*

$$\sigma_4^2 := \sigma_3^2 \frac{(q')^2}{Q^2} + \sigma_{ks}^2,$$

where σ_{ks}^2 is the variance generated by key switching process.

3.6 Full Flow

In this section, we present the full flow of LEAF in Algorithm 4.

Correctness analysis. Correctness of our scheme requires that the decryption of k -th RLWE ciphertext output by Algorithm 4 equals to

$$\text{Ecd}_c([g((WM)_k[0]), \dots, g((WM)_k[n-1])]) ,$$

i.e., the j -th coefficient in the decrypted message is $g((WM)_k[j]), j \in [n], k \in [d]$. The correctness of matrix multiplication is proved in [1]. The correctness of Equation 7 is guaranteed by the LT algorithm in Algorithm 6. The correctness of the polynomial construction is given by Lemma 2. The correctness of S2C can be derived by Equation 9 and the correctness of the LT algorithm in Algorithm 9. The correctness of remaining steps like modulus switch and ring switch is guaranteed by the correctness of BFV scheme. Please refer to [2] for more details. By ensuring the correctness of each component, we show that Algorithm 4 will generate the correct output. We conclude that our failure rate of decryption is negligible by our choice of parameters as in Section 4.

Efficiency analysis. In this section, we focus on analyzing the amortized number of *polynomial multiplications* per non-polynomial function evaluation for one input. For the complexity analysis of matrix multiplication, no polynomial multiplication is needed. Please refer to [1, 30] for more details. Algorithm 1 involves $O(\sqrt{N})$ rotations and $O(N)$ plaintext-ciphertext multiplications. Algorithm 9 has the same asymptotic analysis with Algorithm 1. Algorithm 2 involves $O(\sqrt{P})$ ciphertext-ciphertext multiplication and $O(P)$ plaintext-ciphertext multiplications. In the full flow, all three algorithms are performed z times together with z ring switches to evaluate GELU on $d \times n$ inputs.

Rotation is done via Galois automorphism which takes $\text{poly}(L)$ polynomial multiplications, where L is the multiplicative depth. Each ring switch needs

Algorithm 4 Full flow of our scheme LEAF.

Input:

l coefficient encoded RLWE ciphertexts $\{\text{RLWE.ct}_i\}_{i \in [l]}$ each encrypting the i -th row \mathbf{m}_i of matrix $M \in \mathbb{R}^{l \times n}$, where $\text{RLWE.ct}_i = (\mathbf{a}_i, \mathbf{b}_i) \in \text{RLWE}_{\underline{s}, q'}(\text{Ecd}_c(\mathbf{m}_i := [m_{i1}, \dots, m_{in}])) \in (\mathcal{R}_{q'}^{(n)})^2$.
 Packing key: $\text{BFV.ct}_{\text{packing}} \in \text{BFV}_{s, Q}(\text{Ecd}_s([\underline{s}, \underline{s}, \dots, \underline{s}]))$.
 The non-polynomial function being evaluated $g(\cdot) : \mathbb{Z}_p \rightarrow \mathbb{Z}_p$.
 Plaintext matrix $W \in \mathbb{Z}_p^{d \times l}$ obtained by encoding weight matrix to integers.

Output:

d refreshed RLWE ciphertexts in coefficient encoding, $(WM)_k$ is WM 's k -th row:
 $\text{RLWE}_{\underline{s}, q'}(\text{Ecd}_c(g((WM)_k)))$, where $g((WM)_k) := [g((WM)_k[0]), \dots, g((WM)_k[n-1])]$.

- 1: Perform matrix multiplication on $\{\text{RLWE.ct}_i\}_{i \in [l]}$ and W to obtain d RLWE.ct $_k$, where $\text{RLWE.ct}_k \in \text{RLWE}_{\underline{s}, q'}(\text{Ecd}_c((WM)_k)) \in (\mathcal{R}_{q'}^{(n)})^2, k \in [d]$.
 - 2: Perform modulus switch on d RLWE.ct $_k$ from input ciphertext modulus q' to q , where we have $q = P$, the plaintext modulus of the BFV scheme.
 - 3: Let $z = \lceil d/(N/n) \rceil$.
 - 4: Apply Algorithm 1 z times to pack d coefficient encoded RLWE ciphertexts into z slot encoded BFV ciphertexts: $\{\text{BFV}_{s, Q}(\text{Ecd}_s(\tilde{\mathbf{m}}_j))\}_{j \in [z]}$.
 - 5: Construct $f_{\text{poly}}(x)$ using Lemma 2 with $f(x) := \alpha g(\lfloor \frac{x}{\alpha} \rfloor)$.
 - 6: Evaluate $f_{\text{poly}}(x)$ on $\text{BFV}_{s, Q}(\text{Ecd}_s(\tilde{\mathbf{m}}_j))$ using Algorithm 2 for $j \in [z]$. If we write $g(\cdot)$ with vector input: $g(\mathbf{x}) := [g(x_0), g(x_1), \dots, g(x_{N-1})]$, the result is $\text{BFV}_{s, Q'}(\text{Ecd}_s(\alpha g(\lfloor \tilde{\mathbf{m}}_j / \alpha \rfloor))), j \in [z]$.
 - 7: Perform S2C process from Algorithm 9 z times to obtain z coefficient encoded $\text{BFV.ct}_{\text{coeff}, j} := \text{BFV}_{s, Q'}(\text{Ecd}_c(\alpha g(\lfloor \tilde{\mathbf{m}}_j / \alpha \rfloor))), j \in [z]$.
 - 8: Perform modulus switch on z $\text{BFV.ct}_{\text{coeff}, j}$ from Q' to q' . The result is $\text{BFV.ct}_j \in \text{RLWE}_{\underline{s}, q'}(\text{Ecd}_c(\alpha g(\lfloor \tilde{\mathbf{m}}_j / \alpha \rfloor))), j \in [z]$.
 - 9: Perform z ring switch on $\{\text{BFV.ct}_{\text{coeff}, j}\}_{j \in [z]}$ from $(\mathcal{R}_{q'}^{(N)})^2$ to $(\mathcal{R}_{q'}^{(n)})^2$. Each ring switch will output N/n coefficient encoded RLWE ciphertexts, so there are totally d RLWE ciphertexts. The result is $\text{RLWE}_{\underline{s}, q'}(\text{Ecd}_c(g((WM)_k))), k \in [d]$.
 - 10: Return d refreshed RLWE ciphertexts, each encrypting $\text{Ecd}_c(g((WM)_k)), k \in [d]$ with RLWE secret key \underline{s} and ciphertext modulus q' .
-

a key switch which also takes $\text{poly}(L)$ polynomial multiplications. Ciphertext-ciphertext multiplication requires a tensor product between two ciphertexts followed by a *relinearization* process, which also takes $\text{poly}(L)$ polynomial multiplications in total ([14]). Plaintext-ciphertext multiplication is much simpler and only needs $O(L)$ polynomial multiplications. Regarding to parameters, we have $P > 2N + 1$ to guarantee that there is a primitive $2N$ -th root of unity, and $O(L) = O(\log P)$ to guarantee that there are enough depth for evaluating polynomial and performing LTs. Therefore, the total number of polynomial multiplication to evaluate $d \times n$ inputs is in Equation 11. Thus, $\tilde{O}(dn)$ polynomial multiplications are required to evaluate non-polynomial function on dn inputs. The amortized cost is quasi-constant number $\tilde{O}(1)$ of polynomial multiplications.

Security analysis. Since we do not modify the key generation algorithms and encryption algorithms for RLWE cryptosystem and BFV scheme, the security of our scheme can be easily derived from the semantic security of RLWE cryptosystem and BFV scheme.

$$\begin{aligned}
\text{TotalNumber} &= z \times (O(\sqrt{N}) \cdot O(\text{poly}(l)) + O(N) \cdot O(L)) \times 2 \\
&\quad + z \times (O(\sqrt{P}) \cdot O(\text{poly}(L)) + O(P) \cdot O(L)) + z \times O(\text{poly}(L)) \\
&= z \times (O(\sqrt{N}) \cdot O(\text{poly}(\log N)) + O(N) \cdot O(\log N)) \times 2 \\
&\quad + z \times (O(\sqrt{N}) \cdot O(\text{poly}(\log N)) + O(N) \cdot O(\log N)) + z \times O(\text{poly}(\log N)) \\
&= z \times ((O(\sqrt{N} \text{poly}(\log N)) + O(N \log N)) \times 3 + O(\text{poly}(\log N))) \\
&= O(z(\sqrt{N} \text{poly}(\log N) + N \log N)) = \tilde{O}(zN) = \tilde{O}(dn)
\end{aligned} \tag{11}$$

4 Evaluation results

In this section, we report our evaluation results. We implemented our algorithm in C++. We use SEAL [44] for BFV scheme, OpenSSL [46] and LAC [37] for secure random number generation, OpenBLAS¹³ for implementing matrix multiplication algorithm in [1] and OpenMP [10] for multi-thread programming. We benchmark our scheme on two platforms: Macbook Pro with Apple M3 Max chip and Intel(R) Xeon(R) Platinum 8480+ at 2.0 GHz.

Parameter setting. In our evaluations, we choose coeff-encoded RLWE parameters of non-polynomial function evaluation as follows: Degree of polynomial ring in RLWE $n = 1024$, plaintext modulus $p = 512$, and ciphertext modulus $q = 65537$. Notice that in real world applications, if the input coeff-encoded RLWE ciphertexts are in larger ciphertext modulus and higher polynomial ring, we can apply modulus switch and ring switch first. And BFV parameters as follows: degree of polynomial ring $N = 32768$, plaintext modulus = LWE ciphertext modulus $P = q = 65537$, and ciphertext modulus $\log Q = 669$.¹⁴ For the parameter setting of the matrix multiplication, we refer the reader to [1] for more details.

Failure rate. Our choice of parameters satisfies that output RLWE ciphertexts can be correctly decrypted, which requires the error term in ciphertexts is larger than $\left\lfloor \frac{q}{2p} \right\rfloor$ with tiny probability. The error term in output ciphertexts is a Gaussian distribution [36], so we can analyze the bound of error terms in output ciphertexts by their standard deviation. By running our full scheme with

¹³ <https://github.com/OpenMathLib/OpenBLAS>

¹⁴ Q consists of a set of at most 63 bits primes generated by SEAL library

3200 coeff-encoded RLWE ciphertexts (encrypts 3,276,800 messages in total), the standard deviation of error terms in output ciphertexts is in $(7, 10)$. Thus, we choose q and p such that $\left\lfloor \frac{q}{2p} \right\rfloor = \left\lfloor \frac{65537}{2 \times 512} \right\rfloor > 64$, which ensures that the probability of the event that the error term is larger than $\left\lfloor \frac{q}{2p} \right\rfloor$ is less than 2^{-30} .

4.1 Experimental setup

We primarily compare our work with two open-sourced works: NEXUS [49] and PEGASUS [36]. The concurrent work, THOR [41], also implements FHE-based transformer inference by slot-encoded CKKS scheme, but they do not yet have open source codes and the results in their paper were run in a GPU environment. Therefore, we do not include the comparison with THOR in this section.

NEXUS¹⁵ evaluates all components (such as matrix multiplication, non-linear activations, and bootstrapping) of a transformer in slot-encoded CKKS scheme. PEGASUS¹⁶ evaluates linear functions in slot-encoded CKKS and non-polynomial functions in FHEW-type ciphertext by homomorphic look-up-table (LUT) algorithm. It switches the slot-encoded CKKS ciphertext to a set of FHEW ciphertexts first, then uses LUT to do the evaluation, and repacks to slot-encoded CKKS ciphertext at the end. Both the CPU implementations of NEXUS and the implementation of PEGASUS use SEAL library, which is the same as our implementation. For the choice of non-polynomial function, we choose GELU function in all implementations, which is one of the most popular activation in transformers and also other neural networks.

The results of microbenchmarks are run in Macbook Pro with Apple M3 Max chip and focus on single-thread setting. The results of evaluating feedforward layer in BERT base are run in Intel(R) Xeon(R) Platinum 8480+ at 2.0 GHz, and we set the number of threads to 56. All results are averaged over 10 runs.

4.2 Microbenchmarks

In this section, we report the performance of GELU evaluation in NEXUS, PEGASUS and our non-polynomial evaluation scheme. The flows of evaluation algorithms are shown in Figure 4. NEXUS’s GELU evaluation follows the classic slot-encoded CKKS evaluation algorithm, which consists of a polynomial evaluation process to approximate GELU function and a bootstrapping process to ensure the ciphertext can be used in further computations. Notice that the bootstrapping process can be placed either before or after the polynomial evaluation process, depending on the use case, and the time costs are same in this microbenchmark. PEGASUS’s GELU evaluation pre-computes a “table” containing the value of non-polynomial function, then applies the LWE-based LUT algorithm to evaluate the non-polynomial function and refresh it. Our GELU evaluation also pre-computes a “table” of non-polynomial function, but we encode it into a polynomial and use BFV to evaluate the polynomial in all slots.

¹⁵ <https://github.com/zju-abclab/NEXUS>

¹⁶ <https://github.com/Alibaba-Gemini-Lab/OpenPEGASUS>

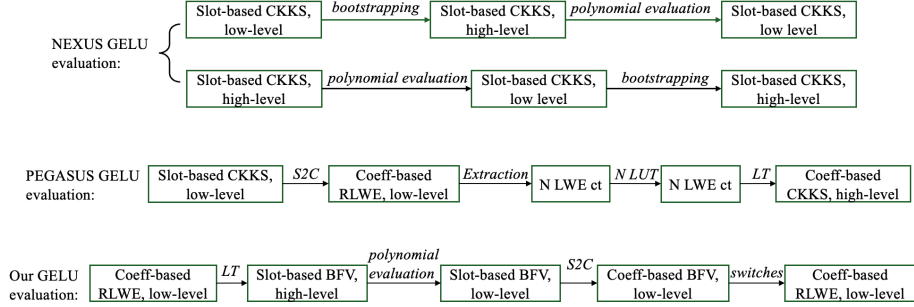


Fig. 4: GELU evaluation algorithms

Accuracy comparison. We set the rounding parameter $s = 16$ in our scheme, so that the input range of GELU function is from $\frac{-256}{s} = -16$ to $\frac{255}{s} = 15.9375$. Figure 5 shows comparisons among real GELU output (red line), NEXUS GELU output (blue line) and our GELU output (orange line) in the input range $[-16, 16]$ and comparisons between absolute errors of NEXUS’s GELU evaluation (blue line) and our GELU evaluation (orange line).¹⁷

Outputs of our GELU evaluation are close to real GELU values for all inputs (the average absolute error of our GELU evaluation is 0.025), while the output of NEXUS’s GELU evaluation introduces a huge error when $x < -8$ or $x > 8$. One key advantage of our GELU evaluation algorithm is its ability to flexibly expand the input range by selecting smaller value of s . This adjustment results in only a slight increase in error, avoiding the significant inaccuracies observed with the approximation method used by NEXUS.

Efficiency comparison. We take one CKKS/BFV ciphertext with 32768 slots as input and report the performance in Table 1¹⁸. All results are in the same single-thread environment. Besides the improvement on accuracy and the flexibility of expand the input range, our algorithm is also 3.7 times faster than NEXUS’s classic slot-encoded CKKS evaluation algorithm. PEGASUS is able to achieve the same accuracy and input range with ours, but it is more than 300 times slower than ours, i.e., the LWE-base LUT algorithm, which does not support SIMD operations, is inefficient for the use case with large number of inputs.

In order to show the optimizations we proposed in section subsection 3.2 Algorithm 3, we also tested our scheme in 20-thread setting and get **0.6ms** as the amortized time per input.

– Total time: 19.6s. Number of input: 32768

¹⁷ PEGASUS’s GELU evaluation has similar accuracy result with ours since we both use a pre-computed table to store the desired output value but use different way to “look up” the table.

¹⁸ PEGASUS’s implementation uses 256 as the default input size.

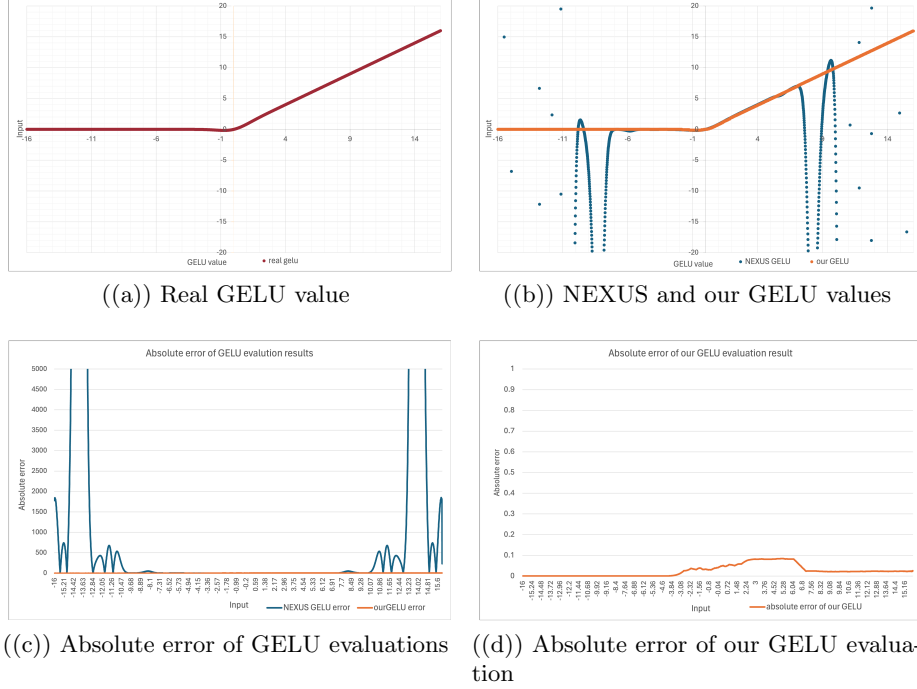


Fig. 5: Output of GELU values and error analysis

- LT time: 2.2s. Polynomial evaluation time: 13.1s. S2C time: 4.3s. Switches time: 0.005s

Notice that this result only considers the non-polynomial evaluations. Our algorithm can take the output of the faster coeff-encoded RLWE matrix multiplication scheme [1] as input directly, making it more suitable for evaluating “matrix multiplication + non-polynomial activation”, which is the common case in secure transformer inference.

4.3 Feed forward layer in BERT base

In this section, we report the performance of using our scheme in the feed forward layer of BERT base model inference. The feed forward layer takes matrix as input. It first multiply a matrix to the input, and then evaluate the activation on every entry.

In order to measure the total time cost, we implement [1]’s matrix multiplication algorithm by using OpenBLAS library. The size of input to the feedforward layer is 128×768 . [1]’s matrix multiplication algorithm will first encrypt it to 768 coeff-encoded RLWE ciphertexts where i -th ciphertext encrypts the i -th column of the input. In order to make better use of slots in these RLWE ciphertexts, [1]’s

Algorithm	NEXUS [49]	PEGASUS [36]	Our scheme
Amortized time per input	8.5ms	749ms	2.3ms
Total time	277.8s	191.74s	76.6s
Number of inputs	32768	256	32768
Breakdown time cost	Bootstrapping: 262s	S2C: 0.54s	LT: 7.6s
	Polynomial eval: 15.8s	Extraction: 6.15s	Polynomial eval: 54s
		LUT: 162.7s	S2C: 15s
		LT: 22.35s	Switches: 0.01s

Table 1: Microbenchmark: Performance of NEXUS, PEGASUS and our non-polynomial evaluation algorithms.

algorithm may batch some inputs together. The batch size does not affect the performance of our scheme since our algorithm always makes use of all BFV plaintext slots during the evaluation.

After evaluating the matrix multiplication, [1]’s algorithm will output 3072 coeff-encoded RLWE ciphertexts, each of which encrypts the i -th column of the multiplication result. The algorithm flow and breakdown time costs are listed in Table 2. The algorithm takes 53.4 seconds in total to evaluate the matrix multiplication, GELU function and refresh the ciphertexts in real BERT base model.

Procedure	Ciphertext	Time cost
Input	$\text{RLWE.ct}_i \in \text{RLWE}_{\mathbb{S},q'}(\text{Ecd}_c(\mathbf{m}_i)), i \in [768],$ $W \in \mathbb{R}^{3072 \times 768}$, and $f_{poly}(x)$ constructed by Lemma 2	-
Pt-ct matrix multiplication [1]	$\text{RLWE.ct}_i \in \text{RLWE}_{\mathbb{S},q'}(\text{Ecd}_s((WM)_i)), i \in [3072]$	0.18s
Mod Switch $q' \rightarrow q$	$\text{RLWE.ct}_i \in \text{RLWE}_{\mathbb{S},q}(\text{Ecd}_s((WM)_i)), i \in [3072]$	0.0025s
Coeff-encoded RLWE to slot-encoded BFV	$3072 \times 128/32768 = 12$ BFV ct, where $\text{BFV.ct}_i \in \text{BFV}_{\mathbb{S},Q}(\text{Ecd}_s(\tilde{\mathbf{m}}_i)), i \in [12]$	6.4s
Polynomial evaluation	$\text{BFV.ct}_i \in \text{BFV}_{\mathbb{S},Q'}(\text{Ecd}_s(\text{GELU}(\tilde{\mathbf{m}}_i))), i \in [12]$	42.8s
Slot-encoded BFV to coeff-encoded BFV	$\text{BFV.ct}_i \in \text{BFV}_{\mathbb{S},Q'}(\text{Ecd}_c(\text{GELU}(\tilde{\mathbf{m}}_i))), i \in [12]$	4.02s
Switches	$\text{RLWE.ct}_i \in \text{RLWE}_{\mathbb{S},q}(\text{Ecd}_c(\text{GELU}((WM)_i))), i \in [3072]$	0.0038s

Table 2: Evaluating feedforward layer of BERT base by our scheme

For comparison, we also report the performance of evaluating feedforward layer of BERT base by NEXUS. NEXUS supports to batch up to 32 inputs in one evaluation process.¹⁹ Before entering the feedforward layer, ciphertexts outputted by the layernorm are in low levels. In order to have enough multiplicative levels to support GELU evaluation (consumes 10+ levels), bootstrapping procedure is needed either at the start of feedforward evaluation or after the

¹⁹ More details of the parameter setting can be found in Section VI of NEXUS [49].

first matrix multiplication in feedforward evaluation. According to Table IV in NEXUS [49], if we put the bootstrapping at the start of feedforward evaluation, then it needs to bootstrap 768×128 slots for each input, but the matrix multiplication has to be done in higher level which is slower than evaluating in lower level. Otherwise, if we put the bootstrapping after the matrix multiplication, although the matrix multiplication can be evaluated in low level, it needs to bootstrap 3072×128 slots. Bootstrapping is considered to be slower than the matrix multiplication, therefore we will follow NEXUS’s framework that bootstrapping the output of layernorm first then evaluating the matrix multiplication and GELU function.

The evaluation flow and breakdown time costs are listed in Table 3. Evaluating the bootstrapping, matrix multiplication, and GELU activation in the same model using NEXUS requires a total of 5,378 seconds, which is over $100\times$ slower than our approach. All experiments were conducted on the same machine using 56 threads. While NEXUS supports batching of up to 32 inputs, the total latency remains constant at 5,378 seconds regardless of the batch size, resulting in an amortized latency of at least 168 seconds per input. Even under its most favorable configuration (i.e., full-batch mode), our method remains over $3\times$ faster than NEXUS in feedforward block evaluation within a real BERT model.

Procedure	Ciphertext	Time cost
Input	768 CKKS $ct_i \in \text{CKKS}(\text{Ecd}_s(\mathbf{m}_i)), i \in [768], W \in \mathbb{R}^{3072 \times 768}$	-
Bootstrapping	768 CKKS $ct_i \in \text{CKKS}(\text{Ecd}_s(\mathbf{m}_i)), i \in [768]$	2145s
Pt-ct matrix multiplication	768 CKKS $ct_i \in \text{CKKS}(\text{Ecd}_s((WM)_i)), i \in [3072]$	2435s
GELU evaluation	768 CKKS $ct_i \in \text{CKKS}(\text{Ecd}_s(\text{GELU}((WM)_i))), i \in [3072]$	798s

Table 3: Evaluating feedforward layer of BERT base by NEXUS

5 Conclusion and discussion

LEAF significantly reduces total latency by leveraging coefficient encoding in two key ways. First, it adopts the efficient coefficient-encoded matrix multiplication technique from [1], which eliminates costly data movement across ciphertext slots typically associated with slot-encoded matrix operations. Second, LEAF integrates ciphertext refreshing directly into the non-polynomial evaluation phase by utilizing the BFV scheme to evaluate an interpolated polynomial over a pre-computed function value table. This approach avoids the overhead of a separate bootstrapping procedure, resulting in a more streamlined and efficient computation pipeline.

An interesting direction for future work is exploring (1) the transformation of low-level coefficient encoding RLWE ciphertexts into high-level slot encoding CKKS ciphertexts and (2) the choice of a lower-degree polynomial to evaluate non-polynomial activations and refresh ciphertexts more efficiently and accurately. This approach may further improve overall performance in privacy-preserving transformer inference since evaluating a low-degree polynomial in

CKKS is considered faster than evaluating a $P - 1$ degree polynomial in BFV, where P is the plaintext modulus of BFV. The main challenge of this potential solution is to ensure the accuracy in a large input range and the ability of refresh the input RLWE ciphertexts.

Appendix

A Linear Transformation (LT) in BFV

A.1 Rotation

This procedure "rotates" the plaintext slots in vector \mathbf{m} . This allows integer arithmetic across different plaintext slots. More concretely, we first rewrite row vector $\mathbf{m} = [m_0, m_1, \dots, m_{N-1}] \in \mathbb{Z}_P^N$ in two row vectors with same length:

$$\mathbf{m} := \begin{bmatrix} \mathbf{m}_1 := [m_0, m_1, \dots, m_{N/2-1}] \\ \mathbf{m}_2 := [m_{N/2}, m_{N/2+1}, \dots, m_{N-1}] \end{bmatrix}.$$

We divide \mathbf{m} into two parts because the "rotations" are happening inside \mathbf{m}_1 or \mathbf{m}_2 , instead of rotating all the N elements m_0, m_1, \dots, m_{N-1} as a loop.²⁰ Most previous works do not make full use of plaintext slots (as described in [32]). For example, when choosing $\mathbf{m}_2 = \mathbf{0}$, they only need to care about half part of \mathbf{m} (i.e., \mathbf{m}_1). Then the algorithms of rotation process and linear transformation process are much simpler, because the rotations are effectively rotating $m_0, m_1, \dots, m_{N/2-1}$ as a loop. However, half of the slots are wasted.

In this work, we manage to make use of all plaintext slots in BFV scheme, and propose the corresponding algorithms for rotation process and linear transformation process.

Let $l \in [\frac{N}{2}]$. Define $\phi_l : \mathcal{R} \rightarrow \mathcal{R}, X \mapsto X^{5^l}$ for some polynomial ring \mathcal{R} . Then we can verify that

$$\phi_l(\text{Ecd}_s(\mathbf{m})) = \text{Ecd}_s \left(\begin{bmatrix} \mathbf{m}_1^{\leftarrow l} := [m_l, m_{l+1}, \dots, m_{N/2-1}, m_0, \dots, m_{l-1}] \\ \mathbf{m}_2^{\leftarrow l} := [m_{l+N/2}, m_{l+N/2+1}, \dots, m_{N-1}, m_{N/2}, \dots, m_{l+N-1}] \end{bmatrix} \right)$$

according to our definition of $\text{Ecd}_s(\cdot)$. For simplicity we define function $\text{rot}_l^{\text{vec}}(\cdot)$ on vector space: $\mathbb{Z}_P^N \rightarrow \mathbb{Z}_P^N$ s.t.

$$\text{rot}_l^{\text{vec}}(\mathbf{m}) = \begin{bmatrix} \mathbf{m}_1^{\leftarrow l} := [m_l, m_{l+1}, \dots, m_{N/2-1}, m_0, \dots, m_{l-1}] \\ \mathbf{m}_2^{\leftarrow l} := [m_{l+N/2}, m_{l+N/2+1}, \dots, m_{N-1}, m_{N/2}, \dots, m_{l+N-1}] \end{bmatrix}.$$

Next we can define rotation to be a function on BFV ciphertext space $(\mathcal{R}_Q^{(N)})^2$. Let $\text{ct} = (\mathbf{a}, \mathbf{b})$ be the BFV ciphertext of $\text{Ecd}_s(\mathbf{m})$. Define $\text{rot}_l : (\mathcal{R}_Q^{(N)})^2 \rightarrow (\mathcal{R}_Q^{(N)})^2$ and it first maps (\mathbf{a}, \mathbf{b}) to $(\phi_l(\mathbf{a}), \phi_l(\mathbf{b}))$. However, it is worthy to notice that $(\phi_l(\mathbf{a}), \phi_l(\mathbf{b}))$ is a BFV ciphertext of polynomial $\phi_l(\text{Ecd}_s(\mathbf{m}))$ under secret key $\phi_l(\mathbf{s})$ instead of \mathbf{s} , so it remains to add a key switch process on $(\phi_l(\mathbf{a}), \phi_l(\mathbf{b}))$ at the end of rotation function $\text{rot}_l(\text{ct})$.

Let $\text{ct} = (\mathbf{a}, \mathbf{b}) = \text{BFV}_{\mathbf{s}, Q}(\text{Ecd}_s(\mathbf{m}))$. In this paper we frequently apply the following kinds of rotations:

²⁰ Briefly speaking, this is because that multiplicative group \mathbb{Z}_{2N}^* is not a cyclic group for power-of-2 integer $N > 2$. In general, the rotation process of BFV scheme can be seen as rotation of slots on hypercube. Please refer to [16, 20, 21] for the math details.

- For $l \in [\frac{N}{2}]$, we construct procedure $\text{rot}_l(\cdot)$ to generate a BFV ciphertext $\text{rot}_l(\text{ct})$ encrypting:

$$\begin{aligned} & \text{Ecd}_s \left(\begin{bmatrix} \mathbf{m}_1^{\leftarrow l} := [m_l, m_{l+1}, \dots, m_{N/2-1}, m_0, \dots, m_{l-1}]^T \\ \mathbf{m}_2^{\leftarrow l} := [m_{l+N/2}, m_{l+N/2+1}, \dots, m_{N-1}, m_{N/2}, \dots, m_{l+N-1}]^T \end{bmatrix} \right) \\ &= \text{Ecd}_s(\text{rot}_l^{\text{vec}}(\mathbf{m})) . \end{aligned} \quad (12)$$

- For $l > \frac{N}{2}$, perform $\text{rot}_{l \bmod N/2}(\cdot)$.
- Let $\phi_{-1}(\cdot) : X \mapsto X^{-1}$. Define $\text{rot}_{-1}(\cdot)$ similarly as above. We can verify that $\text{rot}_{-1}(\text{ct})$ is encrypting:

$$\text{Ecd}_s \left(\begin{bmatrix} \mathbf{m}_2 := [m_{N/2}, m_{N/2+1}, \dots, m_{N-1}]^T \\ \mathbf{m}_1 := [m_0, m_1, \dots, m_{N/2-1}]^T \end{bmatrix} \right) .$$

Further, we can define $\text{rot}_{-l} = \text{rot}_l \circ \text{rot}_{-1}$, for $l \in [\frac{N}{2}]$.

A.2 Linear transformation

Rotation procedures make it possible to evaluate functions across different plaintext slots in \mathbf{m} , and thus it is possible to homomorphically evaluate multiplication between a plaintext matrix A and vector \mathbf{m} . While previous works does not specify how to make full use of plaintext slots, we propose new linear transformation algorithm for this case.

We first focus on a special family of square matrices, and propose a faster “1-baby-step 1-giant-step” linear transformation for these matrices in Algorithm 5. Define a family of square matrices as

$$\mathcal{A} := \left\{ A \in \mathbb{Z}_P^{N \times N} \mid A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}, A_1, A_2 \in \mathbb{Z}_P^{N/2 \times N/2} \right\} . \quad (13)$$

Define vectors $\alpha_r \in \mathbb{Z}_P^N$ for $r \in [N]$ s.t. its j -th entry is $\alpha_r[j] = A[j, r + j \pmod{N}]$. Then it can be verified that when $A \in \mathcal{A}$:

$$A\mathbf{m}^T = \sum_{r=0}^{N-1} \alpha_r \otimes \text{rot}_r^{\text{vec}}(\mathbf{m}) .$$

Note that $A\mathbf{m}^T \pmod{P}$ is a valid input to $\text{Ecd}_s(\cdot)$. If we compute the above in $\mathcal{R}_P^{(N)}$:

$$\text{Ecd}_s(A\mathbf{m}^T) = \sum_{r=0}^{N-1} \text{Ecd}_s(\alpha_r) \times \text{Ecd}_s(\text{rot}_r^{\text{vec}}(\mathbf{m})) . \quad (14)$$

Again, let ct be the BFV ciphertext of $\text{Ecd}_s(\mathbf{m})$. Combine Equation 12 and Equation 14, we can homomorphically evaluate $A\mathbf{m}^T$ given input ct . Therefore, for $A \in \mathcal{A}$ there exists a procedure $LT_A(A, \cdot)$ defined on $(\mathcal{R}_Q^{(N)})^2$ s.t. $LT_A(A, \text{ct})$ is encrypting $\text{Ecd}_s(A\mathbf{m}^T)$.

Homomorphic rotation process $\text{rot}_l(\cdot)$ on ciphertext is expensive. It is a standard technique to apply baby-step-giant-step in linear transformation. The number of homomorphic rotation process is reduced from N to $O(\sqrt{N})$ (e.g. [36]). We first include the “1-baby-step 1-giant-step” linear transformation algorithm $LT_{\mathcal{A}}^{1-bs-1-gs}$ in Algorithm 5 and its RNS optimization version in Algorithm 6. For the sake of simplicity, throughout this paper we focus on linear transformation on square matrix. Our linear transformation process can be applied to general matrix as well (e.g., [36]).

Algorithm 5 1-baby-step 1-giant-step linear transformation for matrix family \mathcal{A} in BFV/BGV

- 1: **Input:** Let ct be the BFV ciphertext encrypting $\text{Ecd}_s(\mathbf{m})$. A plaintext matrix $A \in \mathcal{A}$.
- 2: **Output:** Let $LT_{\mathcal{A}}(A, \text{ct})$ be the BFV ciphertext encrypting $\text{Ecd}_s(A\mathbf{m}^T)$.
- 3: Define vectors $\alpha_r \in \mathbb{Z}_P^N$ for $r \in [N]$ s.t. its j -th entry is

$$\alpha_r[j] = A[j, r + j \pmod{N}].$$

- 4: **Baby-step:** Let $B = \lceil \sqrt{N} \rceil$. Compute $\text{rot}_b(\text{ct})$ for $b \in [B]$.
- 5: **Giant-step:** Let $G = \lceil N/B \rceil$. Compute

$$\tilde{\text{ct}} = \sum_{g \in [G]} \text{rot}_{Bg} \left(\sum_{b \in [B]} \text{Ecd}_s(\text{rot}_{N-Bg}^{vec}(\alpha_{Bg+b})) \cdot \text{rot}_b(\text{ct}) \right)$$

- 6: Return $\tilde{\text{ct}}$.
-

Then we show the “2-baby-step 1-giant-step” linear transformation algorithm $LT^{2-bs-1-gs}$ for general matrices in Algorithm 7. The RNS optimization version of it will be discussed in subsection A.3 and used in our S2C process.

A.3 Pre-computation of S2C

The output of S2C process is also a coefficient encoded BFV ciphertext encrypting $\text{Ecd}_c(\mathbf{m})$. In fact, the S2C process is a linear transformation between matrix U and encrypted vector \mathbf{m} . Note that the output of S2C process is a BFV ciphertext of polynomial $\sum_{i=0}^{N-1} m_i x^i$, which slot-encodes vector $U\mathbf{m}^T$ by the definition of $\text{Ecd}_s(\cdot)$. Let ct be this BFV ciphertext. Then the linear transformation $LT(U, \text{Ecd}_s(\mathbf{m}))$ outputs exactly the desired result of S2C process. Notice that matrix U do not satisfies Equation 13, therefore the “1-baby-step 1-giant-step” LT (Algorithm 5) cannot be applied to obtain BFV ciphertext of $\text{Ecd}_s(U\mathbf{m}^T)$.

Fortunately, U is a pre-known matrix. Instead of storing U in a matrix form, we pre-compute vectors $\alpha_1, \dots, \alpha_N$ according to line 3 of Algorithm 5, and rotate

Algorithm 6 Optimized Linear Transformation with 1-baby-step 1-gaint-step for matrix family \mathcal{A} in BFV/BGV

- 1: **Input:** Let ct be the BFV ciphertext of $\text{Ecd}_s(\mathbf{m})$. A plaintext matrix $A \in \mathcal{A}$.
- 2: **Output:** Let $LT_{\mathcal{A}}(A, \text{ct})$ be the BFV ciphertext of $\text{Ecd}_s(A\mathbf{m}^T)$.
- 3: Define vectors $\alpha_r \in \mathbb{Z}_P^N$ for $r \in [N]$ s.t. its j -th entry is

$$\alpha_r[j] = A[j, r + j \pmod{N}].$$

- 4: **Baby-step:** Let $B = \lceil \sqrt{N} \rceil$. Compute $\text{rot}_b(\text{ct})$ for $b \in [B]$.
- 5: Compute $NTT(\text{rot}_b(\text{ct}))$ for $b \in [B]$.
- 6: **Giant-step:** Let $G = \lceil N/B \rceil$. Compute

$$\tilde{\text{ct}} = \sum_{g \in [G]} \text{rot}_{Bg} \left(INTT \left(\sum_{b \in [B]} NTT(\text{Ecd}_s(\text{rot}_{N-Bg}^{vec}(\alpha_{Bg+b}))) \otimes NTT(\text{rot}_b(\text{ct})) \right) \right)$$

- 7: Return $\tilde{\text{ct}}$.
-

Algorithm 7 2-baby-step 1-gaint-step linear transformation for general matrix in BFV/BGV

- 1: **Input:** Let ct be the BFV ciphertext of $\text{Ecd}_s(\mathbf{m})$. A plaintext matrix $A \in \mathcal{A}$.
- 2: Compute $\{\text{Ecd}_s(\mathbf{b}_i)\}_{i \in [N]} \leftarrow \text{Ecd}_{LT}(A)$ in Algorithm 8.
- 3: **Output:** Let $LT_{\mathcal{A}}(A, \text{ct})$ be the BFV ciphertext of $\text{Ecd}_s(A\mathbf{m}^T)$.
- 4: Define $d = \lceil \sqrt{N/2} \rceil$.
- 5: **Baby-step-1:** Compute $\text{rot}_{id}(\text{ct})$ for $i \in [d]$.
- 6: **Baby-step-2:** Compute $\text{rot}_{-id}(\text{ct})$ for $i \in [d]$.
- 7: Define $G = \lceil \sqrt{N/2} \rceil$, $H = \lceil N/(2G) \rceil$.
- 8: **Giant-step:** Compute

$$\tilde{\text{ct}} = \sum_{g \in [G]} \text{rot}_{Hg} \left(\sum_{h \in [H]} \text{Ecd}_s(\mathbf{b}_{2Hg+h}) \cdot \text{rot}_b(\text{ct}) + \text{Ecd}_s(\mathbf{b}_{2Hg+H+h}) \cdot \text{rot}_{-b}(\text{ct}) \right)$$

- 9: Return $\tilde{\text{ct}}$.
-

then encode them by line 5 of Algorithm 5. Therefore, we first encode the matrix U by Algorithm 8 and get $\text{Ecd}_{LT}(U)$, then apply the “2-baby-step 1-giant-step” LT algorithm (Algorithm 7) for general matrix here.

Algorithm 8 Matrix encoding algorithm for LT in BFV ($\text{Ecd}_{LT}(\cdot)$)

```

1: Input: A plaintext matrix  $U \in \mathbb{Z}_P^{N \times N}$ , where  $P$  is the plaintext modulus and  $N$ 
   is the degree of the polynomial ring.
2: Output: A set of encoded vectors  $\{\text{Ecd}_s(\mathbf{b}_i)\}_{i \in [N]}$ .
3: Define  $d = \lceil \sqrt{N/2} \rceil$ .
4: for each  $i \in [d]$  do
5:   for each  $j \in [2d]$  do
6:     Define vector  $\mathbf{b}_{i*2d+j} \in \mathbb{Z}_P^N$  s.t. its  $k$ -th entry :
7:     for each  $k \in [N]$  do
8:       Define  $row_{ind} = (k - i) \bmod N/2 \in [-N/4, N/4]$ .
9:       Compute  $row_{ind} + = N/2$  if  $row_{ind} < 0$ . and  $row_{ind} + = N/2$  if  $k \geq N/2$ .
10:      Define  $col_{ind} = k + jd \bmod N/2 \in [-N/4, N/4]$ .
11:      Compute  $col_{ind} + = N/2$  if  $(k \geq N/2) \wedge (j < d)$ . and  $col_{ind} + = N/2$  if
          $(k < N/2) \wedge (j \geq d)$ .
12:      The  $k$ -th entry is then:
           
$$\mathbf{b}_{i*2d+j}[k] = U[row_{ind}][col_{ind}]$$

13:     end for
14:     Compute  $\text{Ecd}_s(\mathbf{b}_{i*2d+j})$ .
15:   end for
16: end for
17: Return  $\{\text{Ecd}_s(\mathbf{b}_i)\}_{i \in [N]}$ 

```

Combining all the optimizations, our optimized linear transformation algorithm for S2C process is shown in Algorithm 9. Compared with the state-of-the-art S2C implementation [31], our optimized S2C process reduced the latency from 137s to 15s when running in the same machine with same parameters.

B Noise analysis

In this section we analysis the noise scale of rotation and linear transformation in the form of variance, which together can provide the correctness of our full flow in Algorithm 4. The variance can be used in Chebyshev’s inequality: $\Pr[|X - EX| \geq b] \leq \frac{\sigma^2}{b^2}$ to bound noise term or error term. It is required that $|err| \leq \lfloor \frac{Q}{2P} \rfloor$ for ciphertext modulus Q . Therefore, the variance of the error term should be bounded by

$$\sigma^2 \ll \left\lfloor \frac{Q}{2P} \right\rfloor^2 \approx \frac{Q^2}{4P^2} . \quad (15)$$

To begin with, we include the following lemma from [36].

Algorithm 9 Optimized Linear Transformation with 2-baby-step 1-gaint-step for general matrix in BFV/BGV.

- 1: **Input:** Let ct be the BFV ciphertext of $\text{Ecd}_s(\mathbf{m})$. A matrix $U \in \mathbb{Z}_P^{N \times N}$.
- 2: Pre-compute $\{\text{Ecd}_s(\mathbf{b}_i)\}_{i \in [N]} \leftarrow \text{Ecd}_{LT}(U)$ in Algorithm 8.
- 3: Pre-compute $NTT(\text{Ecd}_s(\mathbf{b}_i))$ for $i \in [N]$.
- 4: **Output:** Let $LT(U, \text{ct})$ be the BFV ciphertext of $\text{Ecd}_s(U\mathbf{m}^T)$.
- 5: Define $d = \sqrt{N/2}$.
- 6: **Baby-step-1:** Compute $\text{rot}_{id}(\text{ct})$ for $i \in [d]$.
- 7: Compute $NTT(\text{rot}_{id}(\text{ct}))$ for $i \in [d]$.
- 8: **Baby-step-2:** Compute $\text{rot}_{-id}(\text{ct})$ for $i \in [d]$.
- 9: Compute $NTT(\text{rot}_{-id}(\text{ct}))$ for $i \in [d]$.
- 10: Define $G = \lceil \sqrt{N/2} \rceil$, $H = \lceil N/(2G) \rceil$.
- 11: **Giant-step:** Compute

$$\tilde{\text{ct}} = \sum_{g \in [G]} \text{rot}_{Hg} \left(INTT \left(\sum_{h \in [H]} NTT(\text{Ecd}_s(\mathbf{b}_{2Hg+h})) \otimes NTT(\text{rot}_b(\text{ct})) + \right. \right. \\ \left. \left. NTT(\text{Ecd}_s(\mathbf{b}_{2Hg+H+h})) \otimes NTT(\text{rot}_{-b}(\text{ct})) \right) \right)$$

- 12: Return $\tilde{\text{ct}}$.
-

Lemma 6 (Appendix D in Pegasus [36]).

1. Let polynomials \mathbf{a}, \mathbf{b} have degree n , assuming whose coefficients are iid respectively. The variances of the coefficients are σ_a^2 and σ_b^2 respectively. Then the variance of $\mathbf{a} + \mathbf{b}$ is $\sigma_a^2 + \sigma_b^2$. The variance of $\mathbf{a}\mathbf{b}$ is $n\sigma_a^2\sigma_b^2$.
2. If the coefficient is uniformly random over $[0, B)$, then its variance is $B^2/12$.

Proof. According to [36], the analysis follows the widely used heuristic assumption that the coefficients of each polynomial behave like independent zero-mean random variables of the same variance [9], and central limit heuristic [13]. $\mathbf{a} + \mathbf{b}$ case is trivial. If we write $\mathbf{c} := \mathbf{a}\mathbf{b}$ into:

$$\mathbf{c} = \sum_{i=0}^{n-1} c_i x^i = \sum_{i=0}^{n-1} a_i x^i \sum_{i=0}^{n-1} b_i x^i \bmod (x^n + 1),$$

we have $c_k = \sum_{i+j=k} a_i b_j - \sum_{i+j=k+n} a_i b_j$. The variance of c_k for every k is $n\sigma_a^2\sigma_b^2$. The second part follows directly by uniform distribution, and is widely used in ciphertext error variance.

□

B.1 Rotation

Given BFV ciphertext $(\mathbf{a}, \mathbf{b}) \in (\mathcal{R}_Q^{(N)})^2$ under secret key \mathbf{s} , the rotation process first computes $(\phi_l(\mathbf{a}), \phi_l(\mathbf{b}))$, which is a BFV ciphertext under secret key $\phi_l(\mathbf{s})$. Therefore, it remains to apply a key switch process on $(\phi_l(\mathbf{a}), \phi_l(\mathbf{b}))$. For completeness, we include the building blocks of key switch process here.

Extended RLWE. For plaintext message $\mathbf{m} \in \mathcal{R}_P^{(N)}$ and a base B , define the extended RLWE ciphertext under secret key \mathbf{s} :

$$\widetilde{\text{RLWE}}(\mathbf{m}) := \{\text{RLWE}_{\mathbf{s},Q}(B^d \mathbf{m}) := (\mathbf{a}_d, \mathbf{b}_d)\}_{d \in [\lfloor \log_B Q \rfloor]} .$$

Here we point out that $(\mathbf{a}_d, \mathbf{b}_d)$ satisfies

$$\mathbf{a}_d \mathbf{s} + \mathbf{b}_d = B^d \mathbf{m} + \mathbf{e}_d \pmod{Q} .$$

Extended RLWE ciphertexts can help to reduce the noise scale in the multiplication between plaintext and ciphertext: e.g., [21] and Algorithm 10 from [36].

Algorithm 10 Multiplication between plaintext and extend RLWE ciphertext.
(\diamond)

- 1: **Input:** Plaintext polynomial $\mathbf{c} \in \mathcal{R}_Q^{(N)}$. Extended RLWE ciphertext $\widetilde{\text{RLWE}}(\mathbf{m})$ for polynomial \mathbf{m} . Base B .
- 2: **Output:** RLWE ciphertext $\text{RLWE}_{\mathbf{s},Q}(\mathbf{cm})$. We write $\text{RLWE}_{\mathbf{s},Q}(\mathbf{cm}) = \mathbf{c} \diamond \widetilde{\text{RLWE}}(\mathbf{m})$.
- 3: Write polynomial $\mathbf{c} \in \mathcal{R}_Q^{(N)}$ to

$$\mathbf{c} = \sum_{d=0}^{\lfloor \log_B Q \rfloor} B^d \cdot \mathbf{c}_d ,$$

where each polynomial \mathbf{c}_d has B -bounded coefficients.

- 4: Let $\widetilde{\text{RLWE}}(\mathbf{m}) := \{\text{RLWE}_{\mathbf{s},Q}(B^d \mathbf{m})\}_{d \in [\lfloor \log_B Q \rfloor]}$, in which we write $(\mathbf{a}_d, \mathbf{b}_d) \in \text{RLWE}_{\mathbf{s},Q}(B^d \mathbf{m})$.
- 5: Return

$$\left(\sum_{d=0}^{\lfloor \log_B Q \rfloor} \mathbf{a}_d \mathbf{c}_d, \sum_{d=0}^{\lfloor \log_B Q \rfloor} \mathbf{b}_d \mathbf{c}_d \right) .$$

For rotation process $\text{rot}_l(\cdot)$, define rotation key to be the extended RLWE ciphertext $\widetilde{\text{RLWE}}(\phi_l(\mathbf{s}))$ under secret key \mathbf{s} . The rotation process $\text{rot}_l(\cdot)$ on RLWE ciphertext $(\mathbf{a}, \mathbf{b}) \in (\mathcal{R}_Q^{(N)})^2$ finally outputs $(0, \phi_l(\mathbf{b})) + \phi_l(\mathbf{a}) \diamond \widetilde{\text{RLWE}}(\phi_l(\mathbf{s}))$.

Lemma 7. *Let σ^2 be the variance of error in generating rotation key $\widetilde{\text{RLWE}}(\phi_l(\mathbf{s}))$. Let ct be a RLWE ciphertext whose error term has variance σ_{in}^2 . Then the variance of error term in $\text{rot}_l(ct)$ is $\sigma_{in}^2 + \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma^2$.*

Proof. Let $ct = (\mathbf{a}, \mathbf{b})$ encrypting \mathbf{m} and set $D = \lfloor \log_B Q \rfloor$. ct satisfies $\mathbf{a}\mathbf{s} + \mathbf{b} = \mathbf{m} + \mathbf{e} + \mathbf{h}Q$. Here \mathbf{h} is some polynomial and \mathbf{e} is the error term. By definition we can write $\widetilde{\text{RLWE}}(\phi_l(\mathbf{s})) = \{\text{RLWE}_{\mathbf{s},Q}(B^d \phi_l(\mathbf{s}))\}_{d \in [D]} = \{(\mathbf{a}_d, \mathbf{b}_d)\}_{d \in [D]}$. And by the definition of extended RLWE scheme,

$$\mathbf{a}_d \mathbf{s} + \mathbf{b}_d = B^d \phi_l(\mathbf{s}) + \mathbf{e}_d + \mathbf{h}_d \cdot Q, \quad \forall d \in [D] .$$

Here \mathfrak{h}_d is some polynomial, and \mathfrak{e}_d is the error term with variance σ^2 .

To apply Algorithm 10, we write $\phi_l(\mathbf{a}) = \sum_{d \in [D]} \bar{\mathbf{a}}_d B^d$. Then Algorithm 10 outputs

$$\left(\sum_{d=0}^D \mathbf{a}_d \bar{\mathbf{a}}_d, \sum_{d=0}^D \mathbf{b}_d \bar{\mathbf{a}}_d \right).$$

Let us say applying \mathbf{s} to ciphertext (\mathbf{a}, \mathbf{b}) outputs $\mathbf{a}\mathbf{s} + \mathbf{b}$. By applying \mathbf{s} to $(0, \phi_l(\mathbf{b})) + \phi_l(\mathbf{a}) \diamond \widetilde{\text{RLWE}}(\phi_l(\mathbf{s}))$ we have

$$\begin{aligned} \left(\sum_{d=0}^D \mathbf{a}_d \cdot \bar{\mathbf{a}}_d \right) \mathbf{s} + \sum_{d=0}^D b_d \cdot \bar{\mathbf{a}}_d + \phi_l(\mathbf{b}) &= \sum_{d=0}^D (\mathbf{a}_d \mathbf{s} + b_d) \cdot \bar{\mathbf{a}}_d + \phi_l(\mathbf{b}) \\ &= \sum_{d=0}^D (\bar{\mathbf{a}}_d B^d \phi_l(\mathbf{s}) + \bar{\mathbf{a}}_d \mathfrak{e}_d + \bar{\mathbf{a}}_d \mathfrak{h}_d Q) + \phi_l(\mathbf{b}) \\ &= \phi_l(\mathbf{a})\mathbf{s} + \phi_l(\mathbf{b}) + \sum_{d=0}^D (\bar{\mathbf{a}}_d \mathfrak{e}_d + \bar{\mathbf{a}}_d \mathfrak{h}_d Q) \\ &= \phi_l(\mathbf{m}) + \phi_l(\mathfrak{e}) + \phi_l(\mathbf{h})Q + \sum_{d=0}^D (\bar{\mathbf{a}}_d \mathfrak{e}_d + \bar{\mathbf{a}}_d \mathfrak{h}_d Q). \end{aligned}$$

After $\bmod Q$ on both sides, it becomes $\phi_l(\mathbf{m}) + \phi_l(\mathfrak{e}) + \sum_{d=0}^D (\bar{\mathbf{a}}_d \mathfrak{e}_d)$. Finally, by the heuristic assumption that \mathbf{a} has uniformly-random coefficient on \mathbb{Z}_Q , $\bar{\mathbf{a}}_d$ has uniformly-random coefficient in $[0, B)$. By applying Lemma 6, the variance of $\bar{\mathbf{a}}_d$ is $B^2/12$. Note that $\phi_l(\cdot)$ will not change variance. So $\phi_l(\mathfrak{e}) + \sum_{d=0}^D (\bar{\mathbf{a}}_d \mathfrak{e}_d)$ has variance $\sigma_{in}^2 + \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma^2$. \square

We can set $\sigma_{rot}^2 := \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma^2$.

B.2 Linear transformation

We first analyze the variance of error in the Linear transformation Algorithm 5.

Lemma 8. *Let σ^2 be the variance of error in generating rotation key $\widetilde{\text{RLWE}}(\phi_l(\mathbf{s}))$. Let ct be a RLWE ciphertext whose error term has variance σ_{in}^2 . Then the variance of error term in $LT_{\mathcal{A}}(A, ct)$ is at most $N^2 P^2 \sigma_{in}^2 + \frac{B^2}{12} N^3 P^2 \lfloor \log_B Q \rfloor \sigma^2$, if $A \in \mathbb{Z}_P^{N \times N}$.*

Proof. 1. Input: We use σ_{in}^2 to represent the variance of error in input ciphertext.
2. Baby-step: Rotations are applied to input ciphertext. By Lemma 7, each $\text{rot}_d(ct)$ has error term with variance $\sigma_{in}^2 + \sigma_{rot}^2$.

3. Giant-step: Assume \sqrt{N} is integer. From Algorithm 5, the function is composed by two rotations:

$$\tilde{\text{ct}} = \sum_{g \in [\sqrt{N}]} \text{rot}_{\sqrt{N}g} \left(\sum_{b \in [\sqrt{N}]} \text{Ecd}_s(\text{rot}_{N-\sqrt{N}g}^{\text{vec}}(\alpha_{\sqrt{N}g+b})) \cdot \text{rot}_b(\text{ct}) \right)$$

First, the inner rotation $\text{rot}_b(\text{ct})$ adds an error term with variance σ_{rot}^2 . Next, multiplying polynomial $\text{Ecd}_s(\text{rot}_{N-\sqrt{N}g}^{\text{vec}}(\alpha_{\sqrt{N}g+b}))$ from $\mathcal{R}_P^{(N)}$ leads to a factor of NP^2 to the variance of error term. The summation leads to another factor of \sqrt{N} . Finally, the outside rotation $\text{rot}_{\sqrt{N}g}(\cdot)$ adds another error term with variance σ_{rot}^2 , and the outermost summation leads to a factor of \sqrt{N} .

In summary, the total variance increased is $\sqrt{N} \left(\sigma_{\text{rot}}^2 + \sqrt{N} \cdot NP^2(\sigma_{\text{in}}^2 + \sigma_{\text{rot}}^2) \right)$. By computing σ_{rot}^2 from Lemma 7, the variance of error term after linear transformation $LT_A(A, \text{ct})$ is bounded by

$$\sqrt{N} \left(\sigma_{\text{rot}}^2 + \sqrt{N} \cdot NP^2(\sigma_{\text{in}}^2 + \sigma_{\text{rot}}^2) \right) = N^2 P^2 \sigma_{\text{in}}^2 + (N^2 P^2 + \sqrt{N}) \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma^2 .$$

□

Similarly, we can compute the variance of error term after linear transformation for general square matrix e.g., $LT(U, \text{ct})$ in Algorithm 7. Recall that from Algorithm 7, the main differences are there are more baby-steps, and the computations in giant-step are changed:

$$\tilde{\text{ct}} = \sum_{g \in [G]} \text{rot}_{Hg} \left(\sum_{h \in [H]} \text{Ecd}_s(\mathbf{b}_{2Hg+h}) \cdot \text{rot}_b(\text{ct}) + \text{Ecd}_s(\mathbf{b}_{2Hg+H+h}) \cdot \text{rot}_{-b}(\text{ct}) \right) \quad (16)$$

Then the total variance increased is $\sqrt{N/2} \left(\sigma_{\text{rot}}^2 + 2\sqrt{N/2} \cdot NP^2(\sigma_{\text{in}}^2 + \sigma_{\text{rot}}^2) \right)$.

Lemma 9. *Let σ^2 be the variance of error in generating rotation key $\widetilde{\text{RLWE}}(\phi_l(\mathbf{s}))$. Let ct be a RLWE ciphertext whose error term has variance σ_{in}^2 . Then the variance of error term in $LT'(A, \text{ct})$ is at most*

$$\sqrt{N/2} \left(\sigma_{\text{rot}}^2 + 2\sqrt{N/2} \cdot NP^2(\sigma_{\text{in}}^2 + \sigma_{\text{rot}}^2) \right) = N^2 P^2 \sigma_{\text{in}}^2 + (\sqrt{N/2} + N^2 P^2) \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma^2 .$$

By Chebyshev's inequality and Equation 15, the output ciphertext can be decrypted with overwhelming probability if

$$N^2 P^2 \sigma_{\text{in}}^2 + (N^2 P^2 + \sqrt{N/2}) \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma^2 \ll \frac{Q^2}{4P^2} . \quad (17)$$

B.3 Noise analysis of Algorithm 4

Now we are ready to provide the noise analysis of Algorithm 4 in the form of variance, and finish the proofs of Lemma 2,3,4 and 5. For the sake of simplicity we set $d = N/n$ as Algorithm 4 is the same in $d/(N/n)$ parts.

- (*Coefficient encoded RLWE ciphertexts \rightarrow slot encoded BFV ciphertext*)
 Let the input RLWE ciphertexts be $(\mathbf{a}_i, \mathbf{b}_i) \in \text{RLWE}_{\mathbf{s}, q'}(\text{Ecd}_c([m_{i1}, \dots, m_{in}])) \in (\mathcal{R}_{q'}^{(n)})^2$, for $i \in [N/n]$. After Algorithm 1 the noise terms of these RLWE ciphertexts will be moved to BFV slots, and later will be cleared by the SIMD evaluation of $f_{\text{poly}}(\cdot)$. Therefore, the noise term in the output BFV ciphertext of Algorithm 1 comes from the linear transformation. By Lemma 8, the variance of the noise term is $\sigma_1^2 \leq N^2 P^2 \sigma_{\text{packing}K}^2 + \frac{B^2}{12} N^3 P^2 \lfloor \log_B Q \rfloor \sigma_{\text{rot}K}^2$, where $\sigma_{\text{packing}K}^2$ is the variance in generating the packing key, and $\sigma_{\text{rot}K}^2$ is the variance in generating the rotation key.
- (*Evaluating arbitrary function on slot encoded BFV ciphertext*)
 In this step, $f_{\text{poly}}(\cdot)$, which is a $P - 1$ degree polynomial, is evaluated in HE. The multiplication depth to evaluating it, according to the Paterson-Stockmeyer algorithm, is $\lceil \log \sqrt{P} \rceil + 2$. Let the noise variance in the output be σ_2^2 . Then we must have $\sigma_2^2 \ll \frac{Q^2}{4P^2}$ according to Equation 15. More concretely, we will choose a larger Q s.t. $\sigma_2^2 \ll \frac{Q^2}{8P^2 P^2 N^2}$. This will only increase the bit-length of Q by several bits. In fact, we are increasing Q so that two more levels are supported, for the previous LT and the later S2C. According to the instruction in BFV [14], SEAL [44] and σ_1^2 , the result can be correctly decrypted for Q s.t. $\log Q = 669$, supporting 13 levels.
- (*Slot encoded BFV ciphertext \rightarrow coefficient encoded BFV ciphertexts*)
 We perform S2C to the previous output. By Lemma 9, the noise variance in the output is

$$\sigma_3^2 := N^2 P^2 \sigma_2^2 + (\sqrt{N/2} + N^2 P^2) \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma_{\text{rot}K}^2.$$

- (*Coefficient encoded BFV ciphertext \rightarrow coefficient encoded RLWE ciphertexts*)
 This part contains three steps. First is modulus switching from Q to q' . The noise variance is from σ_3^2 to $\sigma_3^2(q')^2/Q^2$. Next is key switching from \mathbf{s} to \mathbf{s}' . This step adds noise of σ_{ks}^2 to $\sigma_3^2(q')^2/Q^2$, where σ_{ks}^2 is the variance in from key switching. The last step is ring switching, which is a re-arrangement of coefficients and it does not affect variance.

In conclusion, using our choice of σ_2^2 , the variance in the noise terms in the final output ciphertexts is

$$\begin{aligned}\sigma_3^2(q')^2/Q^2 + \sigma_{ks}^2 &= \frac{(q')^2}{Q^2} N^2 P^2 \sigma_2^2 + \frac{(q')^2}{Q^2} (\sqrt{N/2} + N^2 P^2) \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma_{rotK}^2 + \sigma_{ks}^2 \\ &\ll \frac{(q')^2}{Q^2} \frac{Q^2}{8p^2} + \frac{(q')^2}{Q^2} (\sqrt{N/2} + N^2 P^2) \frac{B^2}{12} N \lfloor \log_B Q \rfloor \sigma_{rotK}^2 + \sigma_{ks}^2 \\ &\leq \frac{(q')^2}{4p^2} .\end{aligned}$$

σ_{ks}^2 is much smaller than $\frac{(q')^2}{4p^2}$. Note that the final output ciphertexts have plaintext modulus p and ciphertext modulus q' , so it can be correctly decrypted.

When evaluating real value function $r(x) : [-B, B] \rightarrow [-B, B]$ where $\Delta B \leq P/2$, there exists another additive error $\max_{x \in [-B, B]} |r(\frac{\lfloor \Delta x \rfloor}{\Delta}) - r(x)|$. This error is from the approximation of $r(x)$ using $f_{poly}(x)$. If $r(x)$ is an \mathcal{L} -Lipschitz function, then this additive error is at most $\mathcal{L}/(2\Delta)$.

References

1. Bae, Y., Cheon, J.H., Hanrot, G., Park, J.H., Stehlé, D.: Plaintext-ciphertext matrix multiplication and the bootstrapping: Fast and fused. In: Annual International Cryptology Conference. pp. 387–421. Springer (2024)
2. Bae, Y., Cheon, J.H., Kim, J., Park, J.H., Stehlé, D.: Hermes: Efficient ring packing using mlwe ciphertexts and application to transciphering. In: Annual International Cryptology Conference. pp. 37–69. Springer (2023)
3. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology* **14**(1), 316–338 (2020)
4. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Annual International Cryptology Conference. pp. 483–512. Springer (2018)
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* **6**(3), 1–36 (2014)
6. Brutzkus, A., Gilad-Bachrach, R., Elisha, O.: Low latency privacy preserving inference. In: International Conference on Machine Learning. pp. 812–821. PMLR (2019)
7. Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 315–337. Springer (2018)
8. Cheon, J.H., Kim, A., Kim, M., 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)
9. Costache, A., Smart, N.P.: Which ring based somewhat homomorphic encryption scheme is best? In: Cryptographers’ Track at the RSA Conference. pp. 325–340. Springer (2016)

10. Dagum, L., Menon, R.: Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering* **5**(1), 46–55 (1998)
11. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. pp. 4171–4186 (2019)
12. Dong, Y., Lu, W.j., Zheng, Y., Wu, H., Zhao, D., Tan, J., Huang, Z., Hong, C., Wei, T., Chen, W.: Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533* (2023)
13. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: *Annual international conference on the theory and applications of cryptographic techniques*. pp. 617–640. Springer (2015)
14. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012)
15. Geelen, R.: Revisiting the slot-to-coefficient transformation for bgv and bfv. *Cryptology ePrint Archive* (2024)
16. Geelen, R., Vercauteren, F.: Bootstrapping for bgv and bfv revisited. *Journal of Cryptology* **36**(2), 12 (2023)
17. 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)
18. Halevi, S., Shoup, V.: Algorithms in helib. In: *Annual Cryptology Conference*. pp. 554–571. Springer (2014)
19. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in helib. In: *Annual International Cryptology Conference*. pp. 93–120. Springer (2018)
20. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in HELib. pp. 93–120 (2018). https://doi.org/10.1007/978-3-319-96884-1_4
21. Halevi, S., Shoup, V.: Design and implementation of helib: a homomorphic encryption library. *Cryptology ePrint Archive* (2020)
22. Harvey, D.: Faster arithmetic for number-theoretic transforms. *Journal of Symbolic Computation* **60**, 113–119 (2014)
23. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778 (2016)
24. Iliashenko, I., Negre, C., Zucca, V.: Integer functions suitable for homomorphic encryption over finite fields. In: *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. pp. 1–10 (2021)
25. Jiang, X., Kim, M., Lauter, K., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. pp. 1209–1222 (2018)
26. Keller, M.: Mp-spdz: A versatile framework for multi-party computation. In: *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. pp. 1575–1590 (2020)
27. Lam, K.Y., Lu, X., Zhang, L., Wang, X., Wang, H., Goh, S.Q.: Efficient fhe-based privacy-enhanced neural network for trustworthy ai-as-a-service. *IEEE Transactions on Dependable and Secure Computing* pp. 1–18 (2024). <https://doi.org/10.1109/TDSC.2024.3353536>
28. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *nature* **521**(7553), 436–444 (2015)

29. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40. pp. 618–647. Springer (2021)
30. Liu, J., Zhang, L.F.: Privacy-preserving and publicly verifiable matrix multiplication. *IEEE Transactions on Services Computing* **16**(3), 2059–2071 (2022)
31. Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7 ms, with $\tilde{O}(1)$ polynomial multiplications. pp. 101–132 (2023). https://doi.org/10.1007/978-981-99-8736-8_4
32. Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7 ms, with $o(1)$ polynomial multiplications. In: *International Conference on the Theory and Application of Cryptology and Information Security*. pp. 101–132. Springer (2023)
33. Lou, Q., Jiang, L.: She: A fast and accurate deep neural network for encrypted data. *Advances in Neural Information Processing Systems* **32** (2019)
34. Lu, W.j., Fang, Y., Huang, Z., Hong, C., Chen, C., Qu, H., Zhou, Y., Ren, K.: Faster secure multiparty computation of adaptive gradient descent. In: *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*. pp. 47–49 (2020)
35. Lu, W.j., Huang, Z., Gu, Z., Li, J., Liu, J., Hong, C., Ren, K., Wei, T., Chen, W.: Bumblebee: Secure two-party inference framework for large transformers. *Cryptology ePrint Archive* (2023)
36. Lu, W.j., Huang, Z., Hong, C., Ma, Y., Qu, H.: Pegasus: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: *2021 IEEE Symposium on Security and Privacy (SP)*. pp. 1057–1073. IEEE (2021)
37. Lu, X., Liu, Y., Zhang, Z., Jia, D., Xue, H., He, J., Li, B., Wang, K.: Lac: Practical ring-lwe based public-key encryption with byte-level modulus. *Cryptology ePrint Archive* (2018)
38. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: *Annual international conference on the theory and applications of cryptographic techniques*. pp. 1–23. Springer (2010)
39. Micciancio, D., Polyakov, Y.: Bootstrapping in fhe-like cryptosystems. In: *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. pp. 17–28 (2021)
40. Mohassel, P., Rindal, P.: ABy3: A mixed protocol framework for machine learning. In: *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. pp. 35–52 (2018)
41. Moon, J., Yoo, D., Jiang, X., Kim, M.: THOR: Secure transformer inference with homomorphic encryption. *Cryptology ePrint Archive*, Paper 2024/1881 (2024), <https://eprint.iacr.org/2024/1881>
42. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing* **2**(1), 60–66 (1973)
43. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
44. Microsoft SEAL (release 4.1). <https://github.com/Microsoft/SEAL> (Jan 2023), microsoft Research, Redmond, WA.
45. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. *Designs, codes and cryptography* **71**, 57–81 (2014)
46. The OpenSSL Project: OpenSSL: The open source toolkit for SSL/TLS (April 2003), www.openssl.org

- 47. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
- 48. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. *Advances in neural information processing systems* **30** (2017)
- 49. Zhang, J., Yang, X., He, L., Chen, K., Lu, W.j., Wang, Y., Hou, X., Liu, J., Ren, K., Yang, X.: Secure transformer inference made non-interactive. *Cryptology ePrint Archive* (2024)