2025 6th International Conference on Artificial Intelligence and Computer Engineering (ICAICE) | 979-8-3315-7496-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICAICE68195.2025.11382418

Efficient Privacy-Preserving Linear System Solving Based on Homomorphic Encryption

Yang Liu[1], Shuai Wang[1], Jingwei Chen[2,3,*], Wenyuan Wu[2,3], Yong Feng[2,3]
[1]School of Information Science and Engineering, Chongqing Jiaotong University, Chongqing, China
[2]Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China
[3]Chongqing Key Laboratory of Secure Computing for Biology, Chongqing, China
Email: liuyang13@cqjtu.edu.cn, shuaiwang@mails.cqjtu.edu.cn, *chenjingwei@cigit.ac.cn,
wuwenyuan@cigit.ac.cn, yongfeng@cigit.ac.cn

***Abstract*—Solving the linear systems $\boldsymbol{Ax} = \boldsymbol{b}$ is not only the cornerstone of classical scientific computing, but also closely related to artificial intelligence models. As demands for privacy protection of data and models continue to escalate, research on how to solve encrypted linear systems efficiently without leaking any plaintext information has become an increasingly important direction. Homomorphic encryption, a key cryptographic technology, offers one solution for data privacy and secure computation. Focusing on the two-party outsourcing setting, we design a non-interactive linear-system solver that combines homomorphic encryption with the Neumann series. We present an optimized evaluation scheme for matrix polynomials that reduces computational complexity and leverages the Single Instruction Multiple Data (SIMD) technique to enable batch processing of multiple linear systems. Finally, we conduct experiments on matrices with different types across various dimensions. The results demonstrate that our solution not only guarantees privacy and security but also reduces computation time dramatically, opening new possibilities for practical applications of homomorphic encryption.**

***Index Terms*—homomorphic encryption, privacy-preserving, linear system, matrix inversion, Neumann series**

I. Introduction

As the scale of artificial intelligence (AI) models expands exponentially, many core computations in AI (especially in machine learning and deep learning) rely on the solution of linear systems $\boldsymbol{Ax} = \boldsymbol{b}$. For instance, optimization algorithms (such as gradient descent and Newton's method) often require solving linear systems approximately at each step. However, these AI models usually contain some private information or exclusive rights. Consequently, it has gradually become an important research direction to solve encrypted linear systems efficiently without leaking any plaintext information. With the emergence of technologies such as homomorphic encryption [1], secure multi-party computation [2], [3], differential privacy [4], and federated learning [5], privacy-preserving techniques have been widely adopted in real-world applications. Among them, homomorphic encryption rests on established assumptions, offers provable security, and achieves the theoretically optimal round complexity. It is therefore regarded as one of the most promising privacy-preserving technologies. The representative mature schemes include BGV [6], BFV [7], CKKS [8], and TFHE [9].

References [10], [11] solve linear systems using the Paillier additively homomorphic encryption scheme. Reference [10] employs a Cholesky decomposition and additionally introduces Yao's garbled circuits. Reference [11] adopts Gaussian elimination and proposes a new protocol that reduces communication cost. References [12], [13] perform matrix inversion via Newton iteration [14], which consumes a large multiplicative depth. To avoid bootstrapping, the participating parties must interactively encrypt intermediate results and re-encrypt them to eliminate noise. Reference [15] combines homomorphic encryption with random perturbation techniques to design an interactive secure multiplicative inverse protocol for the two-party scenario where both parties share the data. However, it is only applicable to low-dimensional matrices. In summary, although the aforementioned approaches can securely solve linear systems in the multi-user setting where data are distributed among participants, they require multiple rounds of interaction and lack generality.

The main contributions of this paper are as follows:

- For the two-party outsourcing scenario, we propose a non-interactive linear systems solver via the *Neumann series* of the matrix $\boldsymbol{A}$ and present the overhead analysis.
- For matrix polynomial evaluation, we introduce an optimization that reduces computational complexity and enables batch solving of multiple linear systems.
- All methods are implemented with the CKKS scheme [8] in the SEAL library [16]. Experimental results show that the proposal scales well for matrices of various dimensions and achieves speed-ups of up to two orders of magnitude compared with [15].

II. Background

In this section, we first introduce some operations of the CKKS scheme [8]. Then, we recall a state-of-the-art method for ciphertext matrix multiplication [17].

A. The CKKS scheme

For $\boldsymbol{x} = (x_i)_{0 \le i < \ell}$ and $\boldsymbol{y} = (y_i)_{0 \le i < \ell}$, let $\mathsf{ct}.\boldsymbol{x}$ and $\mathsf{ct}.\boldsymbol{y}$ be the ciphertext encrypted by CKKS under the same public key. Besides encryption (Enc) and decryption (Dec), CKKS supports the following basic operations on ciphertexts:

- $\mathsf{Add}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})$: $\mathsf{Dec}(\mathsf{Add}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})) \approx \boldsymbol{x} + \boldsymbol{y}$.

- $\mathsf{Mul}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})$: $\mathsf{Dec}(\mathsf{Mul}(\mathsf{ct}.\boldsymbol{x}, \mathsf{ct}.\boldsymbol{y})) \approx \boldsymbol{x} \odot \boldsymbol{y}$, where $\odot$ is for component-wise multiplication.
- $\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})$: $\mathsf{Dec}(\mathsf{CMul}(\boldsymbol{m}, \mathsf{ct}.\boldsymbol{x})) \approx \boldsymbol{m} \odot \boldsymbol{x}$, where $\boldsymbol{m}$ is a message in $\mathbb{C}^\ell$.
- $\mathsf{Rot}_k(\mathsf{ct}.\boldsymbol{x})$ convert $\mathsf{ct}.\boldsymbol{x} = \mathsf{Enc}(x_0, \ldots, x_{\ell-1})$ into a new ciphertext that encrypts $(x_k, \ldots, x_{\ell-1}, x_0, \ldots, x_{k-1})$.

The semantic security of CKKS is based on the RLWE assumption [18]. Meanwhile, there are corresponding countermeasures [19] against Li-Micciancio's attack [20].and complete correctness verification in the CKKS scheme [8].

B. Encrypted Matrix Multiplication

Encrypted matrix multiplication is a highly active research area, with recent progress including [21]–[23]. Among these, Park's algorithm [21] is effective for cases with very high dimensions, Zheng et al.'s algorithm [22] is currently only applicable to the BGV scheme [6], and Chen et al.'s algorithm [23] works well for matrices with dimensions being coprime. The matrix inversion algorithm presented in this paper relies on the multiplication of general square matrices. Compared to these recent works, the JKLS algorithm proposed in [17] is more suitable. In particular, JKLS utilizes the diagonal encoding and the baby-step giant-step technique proposed by Halevi and Shoup in [24], [25] for handling linear transformations.

The total cost of JKLS for a square matrix multiplication of dimension n includes $3n+5\sqrt{n}$ Rots, $5n-1$ CMuls and n Muls, requiring 3 multiplicative depth (1 Mul and 2 CMuls).

III. Encrypted Matrix Inversion

In this section, we present a matrix inversion method based on the Neumann series, reducing matrix inversion to matrix multiplication and enabling the use of encrypted matrix multiplication algorithms, e.g., JKLS.

A. Neumann Series

The Neumann series was introduced by Carl Neumann in 1877 in the context of potential theory [26]. For a square matrix $\boldsymbol{A}$, the *Neumann series* is defined as $\sum_{k=0}^{\infty} \boldsymbol{A}^k$, where $\boldsymbol{A}^0 = \boldsymbol{I}$, i.e., the identity matrix. Then we have

Lemma III.1. *If there exists a consistent*[1] *matrix norm* $\|\cdot\|$ *such that* $\|\boldsymbol{A}\| < 1$*, then the Neumann series converges, and we have:*

$$(\boldsymbol{I} - \boldsymbol{A})^{-1} = \sum_{i=0}^{\infty} \boldsymbol{A}^i.$$

The proof of Lemma III.1 can be found in, e.g., [27, Satz II.1.11]. It follows from Lemma III.1 that for an invertible matrix $\boldsymbol{A}$, we have $\boldsymbol{A}^{-1} = \sum_{i=0}^{\infty} (\boldsymbol{I} - \boldsymbol{A})^i$ if $\|\boldsymbol{I} - \boldsymbol{A}\| < 1$.

For an arbitrary invertible matrix $\boldsymbol{A}$, the condition $\|\boldsymbol{I} - \boldsymbol{A}\| < 1$ is not necessarily satisfied. However, we can still transform $\boldsymbol{A}$ into a matrix that meets this condition through a suitable preprocessing step. The basic idea is observed from

$$\boldsymbol{A}^{-1} = \alpha(\alpha\boldsymbol{A})^{-1} = \alpha \sum_{i=0}^{\infty} (\boldsymbol{I} - \alpha\boldsymbol{A})^i \tag{1}$$

[1]We say a matrix norm $\|\cdot\|$ is consistent if $\|\boldsymbol{AB}\| \le \|\boldsymbol{A}\| \cdot \|\boldsymbol{B}\|$ for all $\boldsymbol{A}$ and $\boldsymbol{B}$. For instance, $\|\cdot\|_1$, $\|\cdot\|_2$, and $\|\cdot\|_\infty$ are all consistent.

if

$$\|\boldsymbol{I} - \alpha\boldsymbol{A}\| \le \delta < 1. \tag{2}$$

Therefore, if for a given invertible matrix $\boldsymbol{A}$, one can find a scaling factor α such that (2) holds, then it is possible to compute an ϵ-approximation of $\boldsymbol{A}^{-1}$ using the truncated Neumann series ((1)) at

$$d \ge \left\lceil \log_{\frac{1}{\delta}} \frac{\alpha}{(1-\delta)\epsilon} \right\rceil - 1. \tag{3}$$

In this way, we reduce the problem of inverting a matrix to a problem of computing a matrix polynomial.

For now, we discuss two important special cases, for which a scaling factor α satisfying (2) can always be constructed.

a) Diagonal Dominant Matrices: Without loss of generality, we restrict ourselves to row diagonally dominant matrices. Recall that an $n \times n$ matrix $\boldsymbol{A} = (a_{i,j})$ is *strictly row-diagonally dominant* if for each row i, $|a_{i,i}| > \sum_{j \ne i} |a_{i,j}|$. Furthermore, we assume that all diagonal entries are positive, i.e., $a_{i,i} > 0$ for all i. Now we can choose α satisfying

$$0 < \alpha < \min\left\{ \frac{1}{a_{i,i}} \right\}. \tag{4}$$

Then for the i-th row of $\boldsymbol{I} - \alpha\boldsymbol{A}$ we have the ∞-norm is

$$|1 - \alpha \cdot a_{i,i}| + \alpha \sum_{j \ne i} |a_{i,j}| = 1 + \alpha \left(\sum_{j \ne i} |a_{i,j}| - a_{i,i} \right) < 1.$$

Since the index i is arbitrary, it follows that $\|\boldsymbol{I} - \alpha\boldsymbol{A}\|_\infty < 1$, i.e., (2) holds under the ℓ_∞-norm.

b) Real Symmetric Matrices: We now consider the case that $\boldsymbol{A}$ is real symmetric (or Hermitian in the complex case). Without loss of generality, we only discuss positive-definite matrices. In this case, we choose α satisfying

$$0 < \alpha < \frac{2}{\lambda_{\max}(\boldsymbol{A})}, \tag{5}$$

where $\lambda_{\max}(\boldsymbol{A})$ is the maximal eigenvalue of the given real positive-definite symmetric matrix $\boldsymbol{A}$. To estimate $\lambda_{\max}$, one can use, e.g., Gershgorin's circle theorem [28]. From (5), we have

$$\|\boldsymbol{I} - \alpha\boldsymbol{A}\|_2 = \max_i |1 - \alpha\lambda_i(\boldsymbol{A})| < 1,$$

where $\lambda_i(\boldsymbol{A})$ is the i-th eigenvalue of $\boldsymbol{A}$. This implies that (2) holds under the ℓ_2-norm.

B. Evaluating Matrix Polynomials

Given a matrix $\boldsymbol{A}$, a scaling factor α satisfying (2), and an error tolerance ϵ, we can truncate the Neumann series in (1) at some d that satisfies (3), yielding an ϵ-approximation of $\boldsymbol{A}^{-1}$:

$$P(\boldsymbol{X}) = \alpha(\boldsymbol{I} + \boldsymbol{X} + \boldsymbol{X}^2 + \ldots + \boldsymbol{X}^d), \tag{6}$$

where $\boldsymbol{X} = \boldsymbol{I} - \alpha\boldsymbol{A}$. We present an efficient evaluation method specifically tailored for this matrix polynomial.

For simplicity, we assume that $\alpha = 1$ and $d = 2^k$ for some positive integer k in (6). For an integer k, let $P_k(\boldsymbol{X}) =$

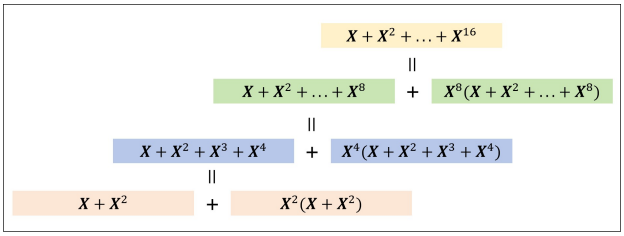

Fig. 1. Matrix polynomial evaluation method for $d = 16$.

TABLE I
METHODS FOR EVALUATING THE MATRIX POLYNOMIAL.

Method	Matrix multiplications	Matrix multiplications depth
Textbook	$O(d^2)$	$O(d^2)$
Horner	$O(d)$	$O(d)$
Paterson–Stockmeyer	$O(\sqrt{d})$	$O(\log d)$
Ours	$O(\log d)$	$O(\log d)$

$\boldsymbol{X} + \boldsymbol{X}^2 + \cdots + \boldsymbol{X}^{2^k}$. Then we have $P(\boldsymbol{X}) = \boldsymbol{I} + P_k(\boldsymbol{X})$. Thus, we focus on the evaluation of $P_k(\boldsymbol{X})$. In fact,

$$P_k(\boldsymbol{X}) = P_{k-1}(\boldsymbol{X}) + \boldsymbol{X}^{2^{k-1}} \cdot P_{k-1}(\boldsymbol{X}). \tag{7}$$

To compute $P_{k-1}(\boldsymbol{X})$, we can continue the recursive procedure as defined in (7), ultimately reaching the leaf node $P_1(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{X}^2$. Fig. 1 illustrates the process for $d = 16$.

In summary, evaluating (7) consumes a matrix multiplication depth of $k = O(\log d)$ and a total of $2k - 2 = O(\log d)$ matrix multiplications. As shown in Table I, this approach significantly outperforms both Horner's rule and the Paterson–Stockmeyer method [29].

C. Encrypted Matrix Inversion

By combining the Neumann-based approach described in Section III-A with the matrix polynomial evaluation method presented in Section III-B, we directly arrive at the following encrypted matrix inversion algorithm, named Neumann.

The correctness of the algorithm follows directly from the analysis in the previous two subsections and is thus omitted here. From Algorithm 1, it is evident that its primary computational overhead comes from $2(\log d - 1)$ calls to the JKLS algorithm, with an encrypted matrix multiplicative depth $\log d$. Recalling the overhead of JKLS (Section II-B), Algorithm 1 requires $2(3n + 5\sqrt{n})(\log d - 1) = O(n \log d)$ Rots, $2(5n - 1)(\log d - 1) + 2 = O(n \log d)$ CMuls and $2n(\log d - 1) = O(n \log d)$ Muls, with $3 \log d + 2 = O(\log d)$ multiplicative depth ($\log d + 2$ Muls and $2 \log d$ CMuls).

D. Further Considerations

Here, we propose two extensions of Algorithm 1. First, we can directly adopt the block matrix multiplication commonly found in textbooks to invert high-dimensional matrices while avoiding the large memory requirements and high multiplicative depth of fast algorithms for plain matrix multiplication.

Algorithm 1 (Neumann)

Input: A ciphertext $\mathsf{ct}.\boldsymbol{A}$ that encrypts an $n \times n$ matrix $\boldsymbol{A}$, a scaling factor α satisfying (2), and an error tolerance $\epsilon \in (0, 1)$.

Output: A ciphertext $\mathsf{ct}.\boldsymbol{P}$ of a $n \times n$ matrix $\boldsymbol{P}$ such that $\left\|\boldsymbol{A}^{-1} - \boldsymbol{P}\right\| < \epsilon$.

1: Determine the least power-of-two integer d satisfying (3).
2: $\mathsf{ct}.\boldsymbol{A} \leftarrow \mathsf{Add}(\mathsf{ct}.\boldsymbol{I}, \mathsf{CMul}(\alpha, \mathsf{ct}.\boldsymbol{A}))$. ▷ Compute $\boldsymbol{I} - \alpha\boldsymbol{A}$.
3: Initialize $\mathsf{ct}.\boldsymbol{P} \leftarrow \mathsf{ct}.\boldsymbol{A}$ and $\mathsf{ct}.\boldsymbol{Q} \leftarrow \mathsf{ct}.\boldsymbol{A}$. ▷ $\mathsf{ct}.\boldsymbol{Q}$ is for powers of $\boldsymbol{A}$.
4: **for** $0 \le i < \log d$ **do**
5: $\quad \mathsf{ct}.\boldsymbol{T} \leftarrow \mathsf{JKLS}(\mathsf{ct}.\boldsymbol{P}, \mathsf{ct}.\boldsymbol{Q})$. ▷ See Sec. II-B for JKLS.
6: $\quad \mathsf{ct}.\boldsymbol{P} \leftarrow \mathsf{Add}(\mathsf{ct}.\boldsymbol{P}, \mathsf{ct}.\boldsymbol{T})$.
7: $\quad$ **if** $i = 0$ **then**
8: $\quad\quad \mathsf{ct}.\boldsymbol{Q} \leftarrow \mathsf{ct}.\boldsymbol{T}$.
9: $\quad$ **if** $1 \le i < \log d - 1$ **then**
10: $\quad\quad \mathsf{ct}.\boldsymbol{Q} \leftarrow \mathsf{JKLS}(\mathsf{ct}.\boldsymbol{Q}, \mathsf{ct}.\boldsymbol{Q})$.
11: $\mathsf{ct}.\boldsymbol{P} \leftarrow \mathsf{Add}(\mathsf{ct}.\boldsymbol{P}, \mathsf{ct}.\boldsymbol{I})$.
12: **return** $\mathsf{ct}.\boldsymbol{P} \leftarrow \mathsf{CMul}(\alpha, \mathsf{ct}.\boldsymbol{P})$.

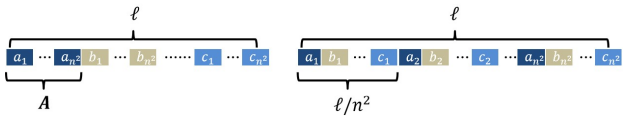

Fig. 2. Row-by-row packing and entry-by-entry packing

Second, for inverting smaller-dimensional matrices, we can fully exploit the SIMD features to pack multiple matrices into a single ciphertext and perform their inversions in parallel, thereby reducing the amortized cost per matrix.

Fig. 2 illustrates two different approaches for packing matrices. In the row-by-row packing, an $n \times n$ matrix is transformed row by row into an n^2-dimensional vector. In the entry-by-entry packing, used in [17]), the ℓ plaintext slots are evenly partitioned into n^2 segments, each of size ℓ/n^2. Each segment stores the corresponding element from each of the ℓ/n^2 matrices. Below, we evaluate how these two packing strategies affect the number of ciphertext operations.

We take $N = 16$ and $n = 2$ as an example. In this setting, each ciphertext is capable of packing 2 matrices. Suppose that we want to exchange the two rows of each matrix. As shown in Fig. 3, for the row-by-row packing, it requires 2 CMuls, 2 Rots, and 1 Add and costs one CMul multiplicative depth, while for the entry-by-entry packing, it requires only a single Rot without any multiplicative depth. For the general case, conclusions remain valid, as shown in Table II. Consequently, we utilize the entry-by-entry packing as our strategy.

IV. SOLVING LINEAR SYSTEMS

In this section we describe how to solve linear systems in batch by exploiting the algorithm presented in Section III.

Since the inversion procedure can be carried out without any interaction, solving single linear systems only requires one additional matrix vector multiplication. By combining

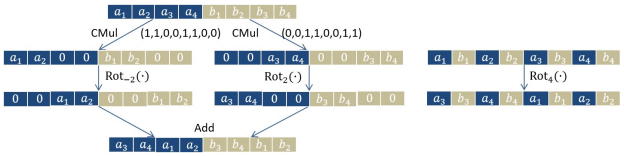

Fig. 3. Rotation under different packing strategies

TABLE II
COST FOR JKLS WITH DIFFERENT PACKING METHODS.

Packing Method	#CMul	#Mul	#Rot	Multiplicative depth
Row-by-row	$7n$	n	$4n + 5\sqrt{n}$	3
Entry-by-entry	$5n$	n	$3n + 5\sqrt{n}$	3

Algorithm 1 with the packing strategy described in Section III-D, we achieve batch solving of linear systems $\boldsymbol{Ax} = \boldsymbol{b}$ summarized in Algorithm 2. Consequently, the entire batch solving procedure consumes $O(n \log d)$ Muls, CMuls and Rots operations, together with multiplicative depth $O(\log d)$.

V. EVALUATION

We employ the CKKS scheme in the SEAL library to implement all algorithms and compare them with the algorithm from [15]. We run all experiments on a PC with an Intel Core i7-12700 processor, 32 GB memory, and Ubuntu 20.04.6 LTS operating system. All codes and matrices used in our test are open-sourced at https://github.com/ws-allen/Inversion.git.

Unless otherwise specified, the CKKS parameters are set as follows: the ring degree N is fixed at 2^{15}; the ciphertext modulus size $\log q$ is set to $60 + L \cdot \log \Delta + 60$, where $L = 3 \log d + 1$ is the multiplicative depth required for computation (including CMul and Mul). The scaling factor used in CKKS is $\Delta = 50$. Note that in all tests, the above parameter setting can achieve a 128-bit security [30].

In addition to the end-to-end running time, we measure the computational accuracy by the relative error

$$\mathsf{RE} = \frac{\|x' - x\|_\infty}{\|x'\|_\infty},$$

where x' denotes the solution obtained from the plaintext linear systems. All errors reported in the tables are the maximum relative errors (MRE) or average maximum relative errors (AMRE) observed across the batch-parallel execution.

Algorithm 2 Solving Linear Systems $\boldsymbol{Ax} = \boldsymbol{b}$

Input: A ciphertext $\mathsf{ct}.\boldsymbol{A}$ encrypting ℓ/n^2 matrices $A \in \mathbb{R}^{n \times n}$, scaling factors α, an error tolerance $\epsilon \in (0, 1)$, and a ciphertext $\mathsf{ct}.\boldsymbol{b}$ encrypting n vectors $\boldsymbol{b} \in \mathbb{R}^n$.

Output: A ciphertext $\mathsf{ct}.\boldsymbol{x}$ encrypting ℓ/n^2 vectors $\boldsymbol{x} \in \mathbb{R}^n$ satisfying $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$.

1: $\mathsf{ct}.\boldsymbol{P} \leftarrow \mathsf{Neumann}(\mathsf{ct}.\boldsymbol{A}, \alpha, \epsilon)$
2: $\mathsf{ct}.\boldsymbol{x} \leftarrow \mathsf{Mul}(\mathsf{ct}.\boldsymbol{P}, \mathsf{ct}.\boldsymbol{b})$
3: **return** $\mathsf{ct}.\boldsymbol{x}$

TABLE III
ERRORS UNDER DIFFERENT δ.

δ	MRE	AMRE
$\delta \le 0.3$	5.31×10^{-4}	5.56×10^{-7}
$0.3 < \delta \le 0.4$	6.03×10^{-3}	9.88×10^{-6}
$0.4 < \delta \le 0.5$	4.6×10^{-2}	1.55×10^{-4}
$0.5 < \delta \le 0.6$	2.5×10^{-1}	1.36×10^{-3}

TABLE IV
THE RESULTS OF DIAGONAL DOMINANT MATRICES

Dimension	Total Time (s)	Amortized Time (s)	MRE	AMRE
4	4.37	4.26×10^{-3}	1.88×10^{-3}	2.38×10^{-5}
8	9.01	3.51×10^{-2}	6.71×10^{-4}	2.11×10^{-5}
16	19.59	0.31	5.96×10^{-4}	2.59×10^{-5}
32	40.83	2.55	1.58×10^{-5}	5.00×10^{-6}
64	87.76	21.94	1.04×10^{-5}	5.80×10^{-6}
128	191.06	191.06	4.75×10^{-6}	4.75×10^{-6}

A. Diagonal Dominant Matrices

To generate n dimensional diagonal dominant matrices, we construct $n \times n$ matrices A whose every element is drawn uniformly from $(0, 1)$, and then replace every diagonal element by four times the ∞-norm of its row. We set $\delta = 0.4$, target accuracy $\varepsilon = 0.01$, and choose $\alpha = 1/\max_i |a_{ii}| - 0.001$ to satisfy (4). The multiplicative depth becomes $3 \log d + 1 = 10$ with $d = 8$.

Table III represents the impact of different values of δ on the error when computing 1024 matrices in batch with $n = 4$ and $d = 8$. The results show that the smaller the value of δ, the higher the accuracy of the outcome.

Table IV summarizes the ciphertext evaluation of Algorithm 2 on diagonal dominant matrices. For the ciphertext of length $\ell = N/2$, every evaluation involves ℓ/n^2 matrices in parallel. The results show that the proposed algorithm supports matrices of various dimensions while delivering accuracy sufficient.

B. Real Symmetric Matrices

To generate n dimensional positive definite matrices A, we first randomly choose diagonal entries to form diagonal matrices $\boldsymbol{\Lambda}$. Next we sample $n \times n$ random matrices and perform QR decomposition to obtain orthogonal matrices $\boldsymbol{Q}$. Finally, we set $\boldsymbol{A} = \boldsymbol{Q\Lambda Q}^\top$. For Algorithm 1 we pick $\alpha = 1.1/\lambda_{\max}(\boldsymbol{A})$ to satisfy (5). The multiplicative depth is $3 \log d + 1 = 10$ with $d = 8$.

Table V reports the results of Algorithm 2 on encrypted positive definite matrices. Both running time and accuracy differ marginally from results for diagonal dominant matrices.

C. Comparison

This section presents a comparison between the experiments conducted in this paper and those in [15], using the results on positive definite matrices for the benchmark.

Table VI compares our implementation with [15]. At a comparable level of accuracy and dimension, our scheme incurs significantly lower running time, and the speed-up

TABLE V
THE RESULTS OF POSITIVE DEFINITE MATRICES.

Dimension	Total Time (s)	Amortized Time (s)	MRE	AMRE
4	4.38	4.27×10^{-3}	3.08×10^{-3}	3.88×10^{-6}
8	8.97	3.50×10^{-2}	6.09×10^{-5}	1.17×10^{-6}
16	19.60	0.31	8.28×10^{-5}	2.88×10^{-6}
32	40.73	2.54	1.80×10^{-5}	3.61×10^{-6}
64	86.89	21.72	3.53×10^{-6}	2.05×10^{-6}
128	189.46	189.46	1.30×10^{-6}	1.30×10^{-6}

TABLE VI
THE COMPARISON OF RESULTS

Method	Dimension	Level	Total Time (s)	MRE	Speedup
Ours	4	10	4.38	10^{-3}	$\times 1.23$
[15]	5	13	5.40	10^{-4}	
Ours	8	10	8.97	10^{-5}	$\times 2.34$
[15]	9	21	21.00	10^{-4}	
Ours	16	10	19.60	10^{-5}	$\times 14.59$
[15]	16	35	286.01	10^{-4}	
Ours	32	10	40.73	10^{-5}	$\times 105.80$
[15]	26	55	4309.80	10^{-5}	

grows with the matrix dimension. Moreover, our algorithm supports larger dimensions while keeping the multiplicative depth constant. In contrast, the depth of [15] is $2n + 3$, forcing it to introduce interaction to avoid bootstrapping as the dimension increases. Meanwhile, the maximum dimension only reaches 26.

VI. CONCLUSION

Based on homomorphic encryption, we present a practical solution for securely solving linear systems $\boldsymbol{Ax} = \boldsymbol{b}$ and thoroughly evaluate its overhead and accuracy. We have achieved batch solution of linear systems with acceptable accuracy and minimal overhead without interaction, and the performance basically meets the needs of practical applications.

In future work, we will further optimize the scheme and consider how to reduce the restrictions on matrix norms to make the scheme more universal. At the same time, we will seek a better packing strategy to reduce the consumption during matrix multiplication and further improve efficiency.

ACKNOWLEDGMENT

This work is partly supported by Chongqing Natural Science Foundation (CSTB2023NSCQ-MSX0441), Chongqing Municipal Education Commission (KJZD-K202500706), and National Natural Science Foundation of China (12571553).

REFERENCES

[1] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms," Found. Secur. Comput., vol. 4, pp. 169–180, 1978.
[2] O. Goldreich, "Secure multi-party computation," Manuscript. Preliminary version, 1998.
[3] C. Zhao, S. Zhao, M. Zhao, et al., "Secure multi-party computation: Theory, practice and applications," Information Sciences, vol. 476, pp. 357–372, 2019.
[4] C. Dwork, "Differential privacy," in Proc. ICALP, Heidelberg: Springer, 2006, pp. 1–12.
[5] C. Zhang, Y. Xie, H. Bai, et al., "A survey on federated learning," Knowledge-Based Systems, vol. 216, p. 106775, 2021.
[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," ACM Transactions on Computation Theory, vol. 6, no. 3, pp. 1–36, 2014.
[7] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, 2012.
[8] J. H. Cheon, A. Kim, M. Kim, and S. Kim, "Homomorphic encryption for arithmetic of approximate numbers," in Proc. ASIACRYPT 2017, LNCS, vol. 10624, pp. 409–437, Springer, 2017.
[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," Journal of Cryptology, vol. 33, no. 1, pp. 34–91, 2020.
[10] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in Proc. 2013 IEEE Security and Privacy, 2013, pp. 334–348.
[11] S. Hu, Q. Wang, J. Wang, T. Wang, and S. Chen, "Securing fast learning! Ridge regression over encrypted big data," in Proc. 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 2016, pp. 19–26.
[12] Y. Zhang, P. Zheng, W. Luo, and S. G. Social, "Privacy-preserving outsourcing computation of QR decomposition in the encrypted domain," in Proc. 2019 IEEE TrustCom/BigDataSE, 2019, pp. 389–396.
[13] N. Mital, C. Ling, and D. Gunduz, "Secure distributed matrix computation with discrete Fourier transform," IEEE Transactions on Information Theory, vol. 68, no. 7, pp. 4666–4680, Jul. 2022.
[14] G. W. Stewart, Matrix Algorithms, Volume II: Eigensystems. Philadelphia, PA, USA: SIAM, 2001.
[15] Y. Lv and W. Wu, "Linear system solving scheme based on homomorphic encryption," Comput. Sci., vol. 49, pp. 338–345, 2022. (in Chinese)
[16] Microsoft, "Microsoft SEAL (release 4.1.2)," accessed Apr. 2025. [Online]. Available: https://github.com/microsoft/SEAL.
[17] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in Proc. 2018 ACM SIGSAC CCS, 2018, pp. 1209–1222.
[18] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," J. ACM, vol. 60, no. 6, pp. 1–35, 2013.
[19] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in Proc. EUROCRYPT 2021, Zagreb, Croatia, LNCS, vol. 12696, pp. 648–677, Springer, 2021.
[20] J. H. Cheon, S. Hong, and D. Kim, "Remark on the security of CKKS scheme in practice," Cryptology ePrint Archive 2020/1531, 2020.
[21] J. H. Park, "Ciphertext-ciphertext matrix multiplication: Fast for large matrices," in Proc. EUROCRYPT 2025, LNCS, vol. 14803, pp. 153–180, Springer, Cham, 2025.
[22] X. Zheng, H. Li, and D. Wang, "A new framework for fast homomorphic matrix multiplication," Des. Codes Cryptogr. 93: 2671–2693, 2025
[23] J. Chen, L. Yang, W. Wu, and Q. Zhang, "Homomorphic matrix operations under bicyclic encoding," IEEE Transactions on Information Forensics and Security, 20: 1390-1404, 2025
[24] S. Halevi and V. Shoup, "Algorithms in HElib," in Proc. CRYPTO 2014, LNCS, vol. 8616, pp. 554–571, Springer, 2014.
[25] S. Halevi and V. Shoup, "Bootstrapping for HElib," Journal of Cryptology, vol. 34, no. 1, p. 7, 2021.
[26] C. Neumann, Untersuchungen über das logarithmische und Newton'sche Potential. Leipzig: BG Teubner, 1877.
[27] D. Werner, Funktionalanalysis. Berlin, Heidelberg: Springer, 2007.
[28] S. A. Gershgorin, "Uber die Abgrenzung der Eigenwerte einer Matrix," Bull. Russ. Acad. Sci., vol. 6, pp. 749–754, 1931.
[29] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials," SIAM Journal on Computing, vol. 2, no. 1, pp. 60–66, 1973.
[30] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," Journal of Mathematical Cryptology, vol. 9, no. 3, pp. 169–203, 2015.