Secure Dimensionality Reduction: Applying Linear Discriminant Analysis over the TFHE Homomorphic Encryption Scheme*

Geonwoo Jeon
School of Cybersecurity
Korea University
Seoul, Republic of Korea
ajwwjsrjsdn@korea.ac.kr

Joon Soo Yoo
School of Cybersecurity
Korea University
Seoul, Republic of Korea
sandiegojs@korea.ac.kr

Baekkyung Song
School of Cybersecurity
Korea University
Seoul, Republic of Korea
baekkyung777@korea.ac.kr

Mi Yeon Hong
School of Cybersecurity
Korea University
Seoul, Republic of Korea
hachikohmy@korea.ac.kr

Ji Won Yoon
School of Cybersecurity
Korea University
Seoul, Republic of Korea
jiwon_yoon@korea.ac.kr

***Abstract*—In the context of the proliferating AI landscape, driven by entities like ChatGPT, the demand for extensive data utilization for training has surged, raising concerns about unauthorized data aggregation and privacy breaches. Paradoxically, this surge in data consumption has raised a critical concern—the breach of privacy stemming from unauthorized data aggregation. Sensitive data categories, including credit card details, medical records, and geographical locations, are particularly vulnerable to misuse. Homomorphic Encryption (HE), a post-quantum attack-resistant cryptographic technique, addresses this concern by enabling secure computations on encrypted data. However, HE's potential is hindered by limitations in evaluation speed, particularly evident in high-dimensional data analysis. This paper introduces contributions, including efficient inverse matrix computation, tailored eigenvector extraction via the power method for the TFHE scheme, and eigenvalue calculation using the Rayleigh quotient within TFHE. The feasibility of applying LDA in the encrypted domain is demonstrated using Fast Fully Homomorphic Encryption over the Torus (TFHE) scheme.**

***Index Terms*—Homomorphic Encryption, Linear Discriminant Analysis, TFHE, Newton's method, Power method**

I. Introduction

In the current landscape, the widespread emergence of AI technologies, as demonstrated by entities like ChatGPT and AlphaGo, has led to a heightened need for substantial data utilization, primarily for training purposes. Paradoxically, this surge in data consumption has given rise to a significant concern – the violation of privacy resulting from the unauthorized accumulation of data. A notable portion of this data is collected without obtaining the necessary consent from data providers or users. Importantly, sensitive data categories, including elements such as credit card details, patients' medical records, and geographical location information, are particularly susceptible to potential misuse by large corporations.

Homomorphic Encryption (HE) is a cryptographic technique enabling computation on encrypted data, resilient against post-quantum attacks. HE effectively counters the unauthorized aggregation of sensitive cloud-based information, as all data is processed in an encrypted state. This approach ensures user data confidentiality while facilitating cloud services. Consequently, advancements in Homomorphic Encryption technology hold the promise of averting personal information leaks, alleviating concerns regarding data privacy.

While HE technology holds significant promise, it faces a notable limitation in terms of evaluation speed. To illustrate, homomorphic gates such as AND and XOR demand approximately 13 microseconds on a single-core personal computer. This issue is exacerbated when dealing with data analysis techniques that involve high-dimensional data processing. Employing HE for such high-dimensional data analysis proves impractical due to excessive time requirements. Consequently, addressing the challenges of high-dimensional data analysis entails the utilization of statistical techniques such as Principal Component Analysis (PCA) [1] and Linear Discriminant Analysis (LDA) [2]. These techniques effectively reduce the dimensionality of the data while preserving a significant portion of the underlying information.

This paper demonstrates the application of LDA in the encrypted domain, aimed at reducing high-dimensional encrypted data into a more manageable format conducive to efficient post-LDA data analysis. Specifically, the study employs Fast Fully Homomorphic Encryption over the Torus (TFHE) [10], a promising HE scheme, to showcase the fea-

This work was supported by an Institute of Information & Communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00558-003, Development of National Statistical Analysis System using Homomorphic Encryption Technology).

sibility of implementing dimensionality reduction techniques within the encrypted realm. The establishment of core operations, comprising four fundamental functions and nonlinear counterparts, finds its basis in existing literature [3], [22]. Employing the homomorphic gates accessible within the TFHE library, we successfully instantiated these operations.

In summary, our paper introduces the following key contributions:

- We demonstrate an efficient approach to compute the inverse matrix of a square matrix within the encrypted domain. Our method employs Newton's method, deviating from the commonplace Gaussian elimination used in unencrypted scenarios.
- We present an effective methodology to extract eigenvectors through the power method, meticulously tailored for the TFHE scheme.
- We propose an adept technique for calculating eigenvalues within the TFHE scheme, employing the Rayleigh quotient.
- We showcase the practical implementation outcomes of applying LDA across varying dimensions and data quantities. Our results substantiate the feasibility of LDA's dimensional reduction within the encrypted domain.

Outline. The paper's structure is organized as follows: In Section II, we offer background insights encompassing homomorphic encryption, the building blocks realized through TFHE, and the essence of LDA. We proceed by discussing related works centered around dimensional reduction within the encrypted domain. Subsequently, our model is elaborated upon, outlining the interaction protocol governing interactions between the client and server. The subsequent section introduces algorithms designed for specific tasks, such as the computation of inverses, eigenvalues, and eigenvectors, all tailored to the TFHE scheme. Moving forward to Sections VI and VII, we present an in-depth exposition of our experimental methodology and the resultant outcomes. Ultimately, we conclude with a comprehensive discussion and concluding remarks.

II. Background

A. Homomorphic Encryption (HE)

1) HE basic: In 1978, Rivest et al. [9] introduced the concept of HE, which capitalizes on the group homomorphism property [5]. To elaborate, HE exploits the characteristic that the outcome of a computation between plaintexts, once encrypted, remains equivalent to the outcome of the computation between ciphertexts.

In the realm of HE, security relies on the noise parameter within the ciphertext. However, the accumulation of noise post each evaluation causes the ciphertext to lose validity at a certain noise threshold. Consequently, the HE scheme gained substantial prominence, especially following Gentry's pioneering work [6] in 2009. Gentry introduced Fully Homomorphic Encryption (FHE) capable of managing ciphertext noise growth. This breakthrough enabled the evaluation of arbitrary boolean circuits, in contrast to Leveled Homomorphic Encryption (LHE) [7], [8], which is constrained to a fixed circuit depth.

2) Boolean evaluation and fixed-point arithmetic: We performed homomorphic encryption of plaintext bits based on the TFHE schemes [10] and used a fixed-point number system in our research. Specifically, we assigned $\frac{r}{2}$, $\frac{r}{2}-1$, and 1 bit for integers, decimals, and signed bits, respectively. The encryption of each bit is designated in the same position as in the plaintext. In consequence, the values between plaintext and its corresponding ciphertext are precisely equal.

Next, we constructed various FHE operations like four fundamental arithmetic operations from the combination of FHE bootstrapping gates (AND, OR, XOR, and so on) provided by the TFHE library. In order to grasp a fully-understanding of our approach, interested readers should refer to [22]. The LDA algorithm is implemented utilizing the basic operations designed as in [22].

B. Linear Discriminanat Analysis (LDA)

LDA is a generalization of Fisher's linear discriminator [11], a method for dimensionality reduction and classification by projection from high-dimensional data onto a low-dimensional data. The LDA's goal [12] is to find a linear transformation that maximizes how the classes are separated in the reduced dimensional space by the projection. Algorithm 1 elaborates details of LDA procedure.

Algorithm 1: General Linear Discriminant Analysis :

```
Input: data $\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{n}$
Output: dominant eigenvector $\mathbf{w}_{a,b}$, eigenvalue $\lambda_{a,b}$
1 $\mathbf{D}_i \leftarrow \{(\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n)\}, i = 1, \ldots, k$ // class-specific subsets
2 $\mu_i \leftarrow \texttt{mean}(\mathbf{D}_i), i = 1, \ldots, k$ // class means
3 $\mathbf{B}_{a,b} \leftarrow (\mu_a - \mu_b)(\mu_a - \mu_b)^T$ for $1 \le b < a \le k$ // between-class scatter matrix
4 $\mathbf{B} \leftarrow \Sigma\, \mathbf{B}_{a,b}$ // between-class scatter matrix
5 $\mathbf{Z}_i \leftarrow \mathbf{D}_i - 1_{n_i}\mu_i^T, i = 1, \ldots, k$ // center class matrices
6 $\mathbf{S}_i \leftarrow \mathbf{Z}_i^T \mathbf{Z}_i, i = 1, \ldots, k$ // class scatter matrices
7 $\mathbf{S} \leftarrow \Sigma_{i=1}^{k} \mathbf{S}_i$ // within-class scatter matrix
8 $\lambda, \mathbf{w} \leftarrow \texttt{eigen}(\mathbf{S}^{-1}\mathbf{B})$ // compute dominant eigenvector
```

The central challenge of LDA arises in step 7 of Algorithm 1. Achieving reduced-dimensionality data hinges on executing two primary operations: matrix inversion and matrix eigenpair extraction. While several approaches exist for these tasks, the necessity to operate within the encrypted domain requires careful consideration when selecting suitable algorithms.

III. Related Work

A. LDA within Paillier Cryptosystem

A work by Khodaparast et al. [13] performed LDA in the Paillier cryptosystem that supports the evaluation of ciphertexts using its homomorphic property. In terms of speed,

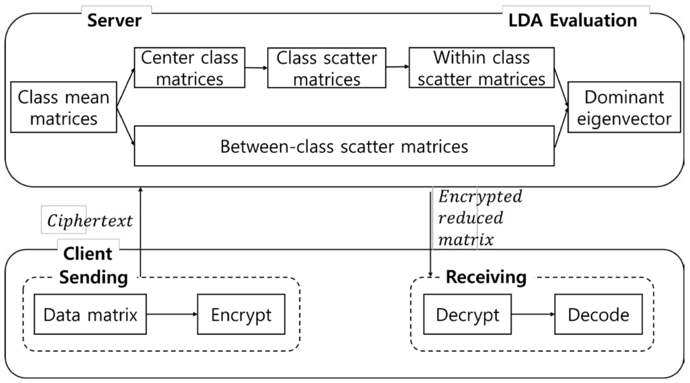

Fig. 1. **An overview of our model.** The model assumes two-party computation over the TFHE scheme. The server evaluates the LDA algorithm over the encrypted data matrix.

[16] the Paillier cryptosystem has the advantage of being fast overall because it encrypts quickly. However, this needs to be compared with FHE. The Paillier cryptosystem could support homomorphic addition and multiplication within limited number. However, FHE enables both homomorphic addition and multiplication without the limited number of operation. Thus, in terms of homomorphic evaluation, it can be seen that FHE is more appropriate for evaluating LDA than the Paillier cryptosystem as LDA requires numerous matrix multiplications.

B. Theoretical construction of LDA

The work presented in [14] primarily demonstrates the theoretical framework of LDA within the FHE domain. In contrast, our paper places greater emphasis on practical application, showcasing the efficient implementation of LDA utilizing the TFHE scheme.

C. Dimensionality Reduction Technique Using FHE

Principal Component Analysis (PCA) is one of the popular dimensionality reduction techniques along with LDA. It aims to reduce the dimensionality of the dataset through maximize the variance of the data along the principal components. A work by Panda et al. [20] adopted a CKKS scheme that supports the approximate computation on complex numbers by power method in PCA. This approach allows vector normalization to be performed using an iterative algorithm of the inverse square root function.

IV. Our model

Our proposed model assumes non-interactive two-party computation scenario (see Fig. 1). In this scenario, the client uses TFHE to evaluate encrypted data. The security is based on the hardness of Learning with Errors (LWE) assumption [21].

A. Model Protocol.

Client. A client encodes the data and encrypts the encoded data using the secret key sk to generate the LWE ciphertext. Thereafter, the generated ciphertext is delivered to the server.
Server. The server performs the LDA algorithm on the provided LWE ciphertext using the following steps:

1) Calculation of the mean for each class matrix.
2) Computation of the between-class scatter matrices between the different class mean matrices.
3) Calculation of the center class matrices from the class means.
4) Computation of the class scatter matrices center class matrices.
5) Calculation of the within class scatter matrices from the class scatter matrices.
6) Evaluation of eigenvalues and eigenvectors using the Newton's method and the power method.
7) Extraction of the dominant eigenvector.
8) Finally, the server sends the resulting processed data back to the client.

Client. The client decrypts with the secret key sk and decodes the received result for further analysis or utilization.

V. Proposed Method

A. Newton's method

Newton's method is an algorithm used to find the inverse of a given matrix. We used this algorithm to complete the expression to find the inverse of the within-class scatter matrix. In fact, the inverse matrix is calculated using Gaussian elimination. Therefore, in order to accurately obtain the inverse matrix, it is necessary to use the Gaussian elimination instead of Newton's method. However, it is hard to express finding the inverse matrix by Gaussian elimination due to use TFHE. Thus, it is required.

```
Algorithm 2: Newton's Method:
  Input: the given square matrix $\mathbf{A}$
  Output: the inversed square matrix $\mathbf{A}^{-1}$
1 $\mathbf{A}^T \leftarrow$ TFHE.Transpose($\mathbf{A}$) // Find $\mathbf{A}^T$.
2 $\mathbf{AA}^T \leftarrow$ TFHE.MatMult($\mathbf{A}, \mathbf{A}^T$) // Calculate $\mathbf{AA}^T$.
3 trace $\leftarrow$ TFHE.Trace($\mathbf{AA}^T$) // Find the trace of $\mathbf{AA}^T$.
4 e1 $\leftarrow$ TFHE.P2C(1) // Make 1 ciphertext.
5 $\alpha \leftarrow$ TFHE.Div(e1, trace) // Calculate 1/trace $= \alpha$.
6 $\mathbf{B} \leftarrow$ TFHE.ScalarMult($\alpha, \mathbf{A}^T$) // Calculate
    $\mathbf{B} = \alpha \times \mathbf{A}^T$.
7 Let $\mathbf{I}_2$ be an matrix such that the matrix has the
    doubled elements of identity matrix.
8 for $i = 1, \ldots, t'$ do
9     $\mathbf{AB} \leftarrow$ TFHE.MatMult($\mathbf{A}, \mathbf{B}$) // Multiply matrices
        $\mathbf{A}$ and $\mathbf{B}$.
10    $\mathbf{T} \leftarrow$ TFHE.MatSub($\mathbf{I}_2, \mathbf{AB}$) // Subtract $\mathbf{I}_2$ and
        $\mathbf{AB}$.
11    $\mathbf{B} \leftarrow$ TFHE.MatMult($\mathbf{B}, \mathbf{T}$) // Multiply matrices $\mathbf{B}$
        and $\mathbf{T}$.
```

In Algorithm 2, there is something unusual. In fact, the Newton's method for inverse matrix [17] needs eigenvalue as α. The details are as follows. By using Gerschgorin's Theorem, we know that, $\lambda_1(\mathbf{AA}^T)$ is bounded by the maximum absolute value row sum of $\mathbf{AA}^T$, that is, $\lambda_1(\mathbf{AA}^T) \leq \mathbf{max}_{i\in\{1,\ldots,n\}} \Sigma_{j=1}^{n} \mid b_{ij} \mid$. Thus, letting $R = \Sigma_{j=1}^{n} \mid b_{ij} \mid$, we can select α according to, $\alpha \in (0, \frac{2}{R})$. Therefore, we selected the trace value as α.

B. Power method

The power method [18] is an algorithm used to find the eigenvector for a given matrix $\mathbf{A}$. Let the initial vector has all elements of 1. For the efficient implementation, we replaced them with sums of each row's elements instead of performing matrix multiplication in the first attempt. Subsequent attempts used the matrix multiplication. In fact, in the normalizing step, dividing to the magnitude of the vector is recommended. However, the dividing the vector to the magnitude has a lot of time, so we replace it by dividing the vector to its element that has the absolute maximum.

Algorithm 3: Power method:

```
Input: the given square matrix $\mathbf{A}$
Output: approximated eigenvector $\mathbf{x}_t$
1 $\mathbf{x}_0 = [1 \quad 1 \quad \dots \quad 1]^T$ // Choose the nonzero initial
    vector $\mathbf{x}_0$.
2 for $i = 1, \dots, t$ do
3     $\mathbf{x}_i \leftarrow$ TFHE.MatMult($\mathbf{A}, \mathbf{x}_{i-1}$) // Multiply matrices
        $\mathbf{A}$ and $\mathbf{x}_{i-1}$.
4     Let $m_i$ be the element that has absolute maximum
        value of $\mathbf{x}_i$.
5     $\mathbf{x}_i \leftarrow$ TFHE.ScalarMult($\frac{1}{m_i}, \mathbf{x}_i$) // Divide $\mathbf{x}_i$ by
        the maximum element of $\mathbf{x}_i$.
6 $\mathbf{x}_t$ // We have approximate $\mathbf{x}_t$ as a dominant
    eigenvector of the matrix $A$.
```

C. Rayleigh quotient

Rayleigh quotient [19] is an algorithm used to find the eigenvalue of a given matrix. In fact, performing inner product in the encrypted implementation takes a lot of time, so we obtained eigenvalue in the following modified form such that do not perform inner product instead of the original form.

Algorithm 4: Rayleigh quotient:

```
Input: the given square matrix $\mathbf{A}$ and the eigenvector
        $\mathbf{x}$
Output: corresponding eigenvalue $\lambda$
1 $\mathbf{Ax}$ // Multiply matrices $\mathbf{A}$ and $\mathbf{x}$.
2 $\mathbf{Ax} \cdot \mathbf{x}$ // Inner product $\mathbf{Ax}$ and $\mathbf{x}$.
3 $\mathbf{x} \cdot \mathbf{x}$ // Inner product $\mathbf{x}$ and $\mathbf{x}$.
4 $\lambda = \frac{\mathbf{Ax} \cdot \mathbf{x}}{\mathbf{x} \cdot \mathbf{x}}$ // Divide $\mathbf{Ax} \cdot \mathbf{x}$ by $\mathbf{x} \cdot \mathbf{x}$.
```

Let t be the iteration number of the power method finding eigenvector $\mathbf{x}$. And let $\mathbf{x}'$ be the eigenvector obtained by the power method for the iteration number $t-1$. We obtained the eigenvalue by dividing the first element of $\mathbf{Ax}'$ (the multiple of the matrix $\mathbf{A}$ and $\mathbf{x}'$) to the first element of $\mathbf{x}'$.

Algorithm 5: Efficient Rayleigh quotient:

```
Input: the given square matrix $\mathbf{A}$ and the iteration
        time $t$
Output: corresponding eigenvalue $\lambda$
1 $\mathbf{x}' \leftarrow$ TFHE.Eigenvector($\mathbf{A}, t-1$) // Find the
    eigenvector of given $\mathbf{A}$ and the iteration time $t-1$.
2 $\mathbf{Ax}' \leftarrow$ TFHE.MatMult($\mathbf{A}, \mathbf{x}'$) // Multiply matrices $\mathbf{A}$
    and $\mathbf{x}'$.
3 $\lambda \leftarrow$ TFHE.Div($\mathbf{Ax}'[1], \mathbf{x}'[1]$) // Divide the first
    element of $\mathbf{Ax}'$ by the first element of $\mathbf{x}'$.
```

VI. Experiment

A. Experiment Setting

1) Environment Setting: Our research was conducted on a system with Intel i5-12400 CPU working on 2.5 GHz with 12 cores, 16GB RAM, and running Ubuntu 20.04 LTS. Additionally, we employed version 1.1 of the TFHE library for the implementation of the PCA algorithm.

2) Parameter: We randomly set all values in the range $(0, 1)$ for data matrix. Subsequently, we tested the time based on the varying dimension ranging from 2 to 10.

B. Power Method

We achieved an efficient implementation of the power method as follows. In the initial iteration of the power method, matrix-vector multiplication is a requisite operation. Given the initial vector's unitary composition, the matrix multiplication outcome simplifies to the summation of matrix columns. Consequently, to alleviate computational overhead, we opted for ciphertext addition as a substitute for resource-intensive multiplication operations amidst ciphertexts, thereby enhancing computational efficiency. We compared the timing analysis based on our optimization technique and the result is presented in Table I. Note that we varied the matrix size from 2 to 5 for comparison.

C. Linear Discriminant Analysis

We supposed the number of classes $k = 2$ and the number of iteration number for Newton's method $t' = 5$. First, we let each of class have $n_i = 2$ for $i = 1, 2$ and $d = 2$, and performed the experiment. And we performed the experiment with $n_i = 2, 3, 4$ and $d = 2, 3, 4$. For the simple comparison, we let the iteration number for power method $t = 5$. Because the experiment needs $t-1$th eigenvector to find the eigenvalue, t must be larger than 1. And to check if the change of n_i and d are reflected in the experiment, we supposed the situation with the change of n_i, the situation with the change of d and the situation with both of the change of n_i and d as 8 cases . Especially without loss of generality, we compare the situation with the change of n_1 and the situation with the change of both n_i. For the comparison with the real LDA value, we let the iteration number $t = 5$ in the experiment.

Also, we set n_1 and n_2 differently under the condition of the dimension $d = 4$ and the iteration number $t = 5$. Since in the LDA process, the data matrix of each class does not need to have the same n_i, we could set the n_i differently. Thus in order to compare the time when different n_i were set and the time when same n_i were set, we operated LDA with different n_1 and n_2 - $(2,3)$, $(2,4)$, and $(3,4)$. Then, we compared the consumed time of the operation with the consumed time of the former operation.

VII. Result

A. *Efficient Design of Power Method for TFHE*

In Table I, we set the iteration number t' to be 3 and compared their execution time.

TABLE I
Execution Time of power method in each of matrix within TFHE scheme. The execution time is measured in minutes.

	2×2	3×3	4×4	5×5
Original method (m)	13.004	21.271	33.045	47.063
Method we suppose (m)	11.770	17.699	25.926	36.051
Difference (m)	1.233	3.572	7.119	11.012
Ratio (%)	9.4847	16.7936	21.5441	23.3981

The processing time for 2×2 matrix in Table I amounted to 13.004 minutes in the case of using the original method, while the processing time for the matrix amounted to 11.770 minutes in the case of using the supposed method. The difference between the two methods for the matrix is 1.233 minutes, which accounted for 9.4847 percent of the total processing time of using the original method.

The time it took to process a 3×3 matrix, as indicated in Table I, was 21.271 minutes when using the conventional approach. However, when employing the alternative method, the processing time was reduced to 17.699 minutes. The disparity between the two methods for the matrix was 3.572 minutes, constituting 16.7936 percent of the total processing time with the conventional approach.

According to the data in Table I, the processing time for a 4×4 matrix was 33.045 minutes using the original method, whereas, with the supposed method, it reduced to 25.926 minutes. The variation between the two methods for the matrix was 7.119 minutes, representing 21.5441 percent of the total processing time with the original method.

The Table I shows that processing a 5×5 matrix took 47.063 minutes using the conventional method, but when the alternative method was applied, the processing time decreased to 36.051 minutes. The difference between the two methods for the matrix amounted to 11.012 minutes, which accounted for 23.3981 percent of the total processing time with the conventional method.

Timing Analysis. It can be seen that the time taken to obtain the eigenvector by applying the power method we proposed was much reduced compared to when the eigenvector was obtained by applying the existing power method. In particular, it can be seen that as the distance increases, the decreasing ratio increases. The ratio will be smaller as the iteration number increases, but the ratio comparison between square matrices of different sizes is still valid. Therefore, this means that the time required to operate the LDA for a sufficiently large distance number can be significantly reduced by the power method we supposed.

B. *LDA Evaluation in Encrypted Domain*

The values for each matrix were chosen randomly using the approach suggested by the experiment, in the same way as former subsection. As suggested by experiment, LDA was operated in nine cases, and the time taken was measured. We also operated LDA for matrices with one of the different matrices, $d = 4$, and $n_1 = 3$ and $n_2 = 5$, to measure the time taken. The iteration number t' is 5.

TABLE II
Execution Time of LDA in each of matrix within TFHE scheme. The execution time is measured in minutes.

	$n = 2$	$n = 3$	$n = 4$
$d = 2$	88.0827	88.2700	95.4732
$d = 3$	255.9794	251.6964	259.5779
$d = 4$	528.7798	542.8419	560.4587

The processing time for 2×2, 2×3, 2×4 matrices in Table II are 88.0827 minutes, 88.2700 minutes, 95.4732 minutes for each. The processing time for matrices of size 3×2, 3×3, and 3×4 in Table is 255.9794 minutes, 251.6964 minutes, and 259.5779 minutes, respectively.

Also, it took 528.7798 minutes to process 4×2 matrices, 542.8419 minutes to process 4×3 matrices, and 560.4587 minutes to process 4×4 matrices. Without loss of generality, under the $d = 4$ and the iteration number $t = 5$, from $n_1 = 3$ matrix and $n_2 = 5$ matrix, the processing time is 563.4918 minutes.

Timing Analysis. Except that the values in each matrix are set at random, the only difference between the data matrices put in to operate the LDA is that n_i and d are different. Therefore, the measured time is mainly affected by the values of n_i and d of the matrix. According to Table II, both d and n_i tend to take longer to operate LDA. It is similar to what we want to talk about through the values obtained by setting another n_i. According to the result of the experiment for the matrices with different n_i - 4×3 matrix and 4×5 matrix, the consumed time taken is 563.4918 minutes, which is similar to the time taken in other experiments of $d = 4$. This supports that d is the main factor influencing the time required. However, there is a limitation that more accurate results actually require the result of substituting more data into the implementation.

It was observed that as the d value increased, the time required also increased absolutely. On the other hand, it was observed that an increase in n_i value did not absolutely increase the time required. We would like to examine the change in the time required as the d and n_i values change and the reason. Considering the method of LDA, it can be inferred without difficulty. The process of obtaining LDA proceeds in the manner shown in Algorithm 1. LDA uses mean, addition between matrices, subtraction, and multiplication. In fact, addition and subtraction do not have a significant impact on the time the implementation is progressing. Since the difference in the values to be divided does not have a significant effect, the process of obtaining the average also cannot be said to have a significant effect just because n_i increases. Considering these points, n_i does not strictly affect the time required.

Conversely, as the d value changes, the structure of the between-class scatter matrix and class scatter matrices that are handled changes. As previously stated, between-class scatter matrix and class scatter matrix have form $d \times d$. However, the time to obtain the inverse matrix, eigenvector, and eigenvalue of the $d \times d$ matrix itself is greatly affected because it depends on the d value. Thus it implies that the time is strongly affected by finding inverse matrix, eigenvector or eigenvalue and d value. Therefore, the overall time taken is generally proportional to d^2, since the error exists but is calculated using the $d \times d$ matrix itself in the part calculated with the $d \times d$ matrix, which is the latter part of the LDA implementation.

VIII. Discussion

Limitation. Our experimental results are limited to small datasets, primarily due to the inherent nature of the TFHE scheme. The evaluation of a single gate alone necessitates 13 microseconds on a single-core computer. As a result, tasks involving matrix arithmetic, which forms the foundational building block of data analysis techniques, incur notable time overheads. It is crucial to note, however, that the LDA procedure serves as a pre-processing step. This implies that once LDA is executed, the reduced data can subsequently undergo further analysis employing a variety of data analytic tools.

Future work. The number of classes we have covered is 2, and the size of the data is small. In addition, the set iteration number is set as a small value. Therefore, it is necessary to deal with a larger number of classes and data, and the iteration number should also be large enough. However, our experiment serves as a guide for a larger scale experiments.

Moreover, the application of diverse optimization techniques holds potential. While this work does not encompass such optimization strategies, their incorporation promises heightened speed improvements. Such techniques can significantly enhance the execution of LDA on high-dimensional data, paving the way for more efficient analyses.

IX. Conclusion

As the collection and processing of data increasingly occur without the consent of data providers, homomorphic encryption emerges as a potent solution for such concerns. However, the slow evaluation speed coupled with the demands of high-dimensional data analysis necessitate faster approaches. This paper addresses this by leveraging LDA to effectively diminish data dimensionality within the encrypted domain, while preserving crucial information. Notably, our approach employs the TFHE library, a prominent Boolean-based homomorphic encryption framework. Additionally, we present efficient algorithms for computing matrix inverses and eigenpairs—a core facet of the LDA technique. Finally, the paper offers empirical results, showcasing the application of LDA across varying input matrix sizes.

References

[1] Wold, S. & Esbensen, K. & Geladi, P. Principal component analysis. *Chemometrics and intelligent laboratory systems*. **2**, pp. 37-52 (1987)

[2] Izenman, A. Linear discriminant analysis. *Modern multivariate statistical techniques: regression, classification, and manifold learning*. pp. 237-280 (2013)

[3] Yoo, J. & Hwang, J. & Song, B. & Yoon, J. A bitwise logistic regression using binary approximation and real number division in homomorphic encryption scheme. *Information Security Practice and Experience: 15th International Conference, ISPEC 2019, Kuala Lumpur, Malaysia, November 26–28, 2019, Proceedings 15*. pp. 20-40 (2019)

[4] Song, B. & Yoo, J. & Hong, M. & Yoon, J. A bitwise design and implementation for privacy-preserving data mining: from atomic operations to advanced algorithms. *Security and Communication Networks*. **2019**, pp. 1-14 (2019)

[5] Yi, X., Paulet, R. & Bertino, E. Homomorphic encryption. *Springer International Publishing*. pp. 27-46 (2014)

[6] , Gentry, C. Fully homomorphic encryption using ideal lattices. *Proceedings of the forty-first annual ACM symposium on Theory of computing*. pp. 169-178 (2009)

[7] Fan, J. & Vercauteren, F. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*. (2012).

[8] Cheon, J. & Kim, A. & Kim, M. & Song, Y. Homomorphic encryption for arithmetic of approximate numbers. *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. pp. 409-437 (2017)

[9] Rivest, R., Shamir, A. & Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications Of The ACM*. **21**, 120-126 (1978)

[10] Chillotti, I. & Gama, N. & Georgieva, M. & Izabachène, M. 2020, TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1), pp.34-91.

[11] Ye, J. Least squares linear discriminant analysis. *Proceedings of the 24th international conference on Machine learning*. pp. 1087-1093 (2007)

[12] Park, C. & Park, H., 2008, A comparison of generalized linear discriminant analysis algorithms. *Pattern Recognition*, 41(3), pp.1083-1097.

[13] Khodaparast, F., Sheikhalishahi, M., Haghighi, H. & Martinelli, F. Privacy-preserving LDA classification over horizontally distributed data. *Intelligent Distributed Computing XIII*. pp.65-74 (2020)

[14] Arita, S. & Nakasato, S., 2017, Fully homomorphic encryption for classification in machine learning. *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*. pp.1-4

[15] Wu, M. & Zhang, Z., 2010, Handwritten digit classification using the mnist data set. *Course project CSE802: Pattern Classification & Analysis.*, 336.

[16] Jost, C., Lam, H., Maximov, A., & Smeets, B. Encryption performance improvements of the paillier cryptosystem. *Cryptology ePrint Archive*. (2015).

[17] Ben-Israel, A., 1965, An iterative method for computing the generalized inverse of an arbitrary matrix. *Mathematics of Computation*, 19(91), pp.452-455

[18] Chu, M. & Watterson, J., 1993, On a multivariate eigenvalue problem, Part I: Algebraic theory and a power method. *SIAM Journal on scientific computing*, 14(5), pp.1089-1106

[19] Victoria, A., Ahmad, M., Ahamad, N. & Lyashenko, V. Algorithmic research and application using the rayleigh method. (2015)

[20] Panda, S. Principal component analysis using ckks homomorphic encryption scheme. *Cryptology EPrint Archive*. (2021)

[21] Regev, O., 2009, On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6), pp. 1-40

[22] Song, B., Yoo, J., Hong, M. & Yoon, J. A bitwise design and implementation for privacy-preserving data mining: from atomic operations to advanced algorithms. *Security And Communication Networks*. **2019** pp. 1-14 (2019)