Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.076731

ARTICLE

Privacy-Preserving Parallel Non-Negative Matrix Factorization with Edge Computing

Wenxuan Yu[1], **Wenjing Gao**[1], **Jiuru Wang**[2], **Rong Hao**[1,*] **and Jia Yu**[1,*]

[1]College of Computer Science and Technology, Qingdao University, Qingdao, China
[2]School of Information Science and Engineering, Linyi University, Linyi, China

*Corresponding Authors: Rong Hao. Email: hr@qdu.edu.cn; Jia Yu. Email: qduyujia@gmail.com

Received: 25 November 2025; Accepted: 27 January 2026; Published: 09 April 2026

ABSTRACT: Non-negative Matrix Factorization (NMF) is a computationally intensive matrix operation that resource-constrained clients struggle to complete locally. Privacy-preserving outsourcing allows clients to offload heavy computing tasks to powerful servers, effectively solving the problem of local computing difficulties. However, the existing privacy-preserving NMF outsourcing schemes only allow one server to perform outsourcing computation, resulting in low efficiency on the server side. In order to improve the efficiency of outsourcing computation, we propose a privacy-preserving parallel NMF outsourcing scheme with multiple edge servers. We adopt the matrix blocking technique to divide the computation task into multiple subtasks, and design the NMF parallel computation algorithm based on the multiplication updating rule. The proposed scheme implements the parallel outsourcing of non-negative matrix factorization based on multiple edge servers. We use random permutation matrices to encrypt original matrix, thereby protecting data privacy. In addition, we utilize the iterative nature of the NMF algorithm for result verification. Theoretical analysis and experimental results prove the advantages of the proposed scheme.

KEYWORDS: Secure outsourcing computation; non-negative matrix factorization; parallel outsourcing; edge computing

1 Introduction

1.1 Background

Today's society is increasingly driven by big data. With the widespread adoption of the internet, the volume of generated data is growing exponentially, with vast amounts of information being produced every second. The storage, processing, and analysis of this data have become one of the most significant challenges of modern society. Dimensionality reduction techniques play a crucial role in addressing these challenges by converting high-dimensional data into lower-dimensional representations while retaining key information from the original dataset. This is particularly important when dealing with large-scale data. Since most commonly encountered data, such as images and documents, are non-negative, it is essential to address the processing of non-negative data [1]. Non-negative Matrix Factorization (NMF), a widely used dimensionality reduction method, decomposes a non-negative matrix into the product of two non-negative matrices, enabling the efficient extraction of underlying features from the data. NMF has been successfully applied in data mining [2], image processing [3], and text analysis [4,5], offering advantages such as high stability and reduced storage requirements. Compared to other matrix factorization methods, NMF provides interpretable decomposition results and significantly reduces storage space [6]. However, as the scale of

data continues to grow, the computational cost of NMF also increases. For clients with limited computing resources, performing NMF on large-scale datasets becomes a challenging task.

Edge computing, as an emerging computational paradigm, effectively addresses the challenges of large-scale data processing and has gained popularity in the Internet of Things (IoT). Unlike the traditional cloud computing that relies on remote data centers, edge computing allows client to offload computational tasks to edge servers located near the data source. This reduces the need for large data transfers to remote servers, saving bandwidth and reducing latency. Client offloading computational tasks to multiple edge servers can significantly reducing the client's computational overhead and improving efficiency. By enabling multiple edge servers to execute computing tasks in parallel, productivity can be significantly boosted. Despite its significant benefits, edge computing also faces several challenges. When the client outsources the computational task to the edge servers, both the data and computations performed on the server side fall outside the client's direct control. One of the primary concerns is ensuring data privacy. Data often involves sensitive information, such as personal medical records [7], making privacy protection crucial. Additionally, edge servers may engage in malicious activities, such as falsifying computation results, which requires reliable verification mechanisms to ensure the accuracy of the results [8].

Secure outsourcing computation refers to outsourcing computational tasks to service providers while ensuring data privacy and the correctness of the computation results [9]. In a secure outsourcing computation scheme based on edge servers, both the privacy of the client's input and output must be adequately protected [10], ensuring that the edge servers cannot gain any valuable information about the client from the outsourced tasks. At the same time, the client should be able to verify the correctness of the computation results returned by the edge servers. This guarantees that the results are accurate and protects against data leakage or result tampering. Furthermore, the client's local computational overhead in secure outsourcing should be significantly lower than the computational overhead of the original task.

In recent years, several secure outsourcing computation schemes for NMF have been proposed [11,12]. These schemes allow clients to securely outsource NMF tasks to the cloud server. However, none of these schemes support parallel outsourcing across multiple edge servers. Actually, it is common for multiple edge servers to assist in computations. Meanwhile, most existing schemes focus on client side efficiency, they often neglect server-side efficiency. How to enhance server efficiency based on parallel computing is a significant challenge. To address this challenge, we aim to develop a novel privacy-preserving parallel NMF outsourcing scheme.

1.2 Related Works

In recent years, secure outsourcing computation has been widely concerned by people. Research on secure outsourcing of matrix-related computations is quite common. Lei et al. [13] proposed a secure outsourcing scheme for matrix inversion computation (MIC). They utilized the random permutation matrix to transform the original matrix and employed the Monte Carlo algorithm for verification. Later, Lei et al. [14] proposed a scheme to outsource large-scale matrix multiplication calculation. The encryption and verification were similar with the previous scheme [13]. Fu et al. [15] pointed out a security problem in the scheme published by Lei et al. [14]. To protect the number of zero elements in the two original matrices from being leaked to the cloud server, Fu et al. [15] constructed two random matrices and added them to the original matrices. Lei et al. [16] proposed a secure outsourcing scheme for the calculation of determinants of large matrices. They introduced block matrix and permutation technique to protect privacy. Subsequently, they implemented LU decomposition on the cloud server to facilitate the verification process. Liu et al. [17] pointed out that the scheme proposed by Lei et al. [16] cannot achieve the level of security they claim because the encryption matrix can be simplified. This simplification renders the block-matrix

technique ineffective. They further prove that the block matrix used in the scheme is redundant. In order to solve these problems, they do not employ the block matrix technique, but instead they introduce mix-row and mix-column operations for privacy protection. For the eigen-decomposition (ED) and singular value decomposition (SVD) of the matrix, Zhou and Li [18] designed two privacy-preserving outsourcing protocols. They used random numbers and identity matrix to protect the information of the original matrix. And then they used orthogonal matrix to rearrange the matrix for privacy protection. But the number of zero elements was revealed. Luo et al. [19] proposed a feasible scheme for secure outsourcing of large-scale QR decomposition and LU decomposition. They designed a series of protection measures for the upper triangle, lower triangle, general, and orthogonal matrices during the decomposition process to ensure privacy protection. The scheme also involves outsourcing matrix inversion operations required for result recovery. Additionally, they implemented security measures for continuous data transmission between the client side and the server side. Li et al. [20] proposed a protocol for IoT devices solving SVD. They used Hestenes method to achieve lightweight transformation of matrices while protecting privacy. Liu et al. [21] proposed a subtly designed invertible matrix and designed an outsourcing scheme based on it. They proposed an optimized matrix chain multiplication that can improve efficiency while also being applied to other outsourced tasks. Gao and Su [22] proposed the first distributed verifiable and traceable utility scheme for matrix multiplication. They proposed a traceable method which minimizes the computational load on the client.

For non-negative matrix factorization, Liu et al. [23] proposed a secure outsourcing scheme by using random permutation matrix for encryption and decryption. They also proposed a method using 1-norm of matrix to verify the results. Duan et al. [12] implemented the encryption of the NMF input matrix through the random arrangement and scaling encryption mechanism. They proposed to run one round of iterative operation on the client for verification. This scheme allows the client to verify the accuracy of the returned results. Then Fu et al. [11] designed a new NMF secure outsourcing scheme based on two non-colluding servers to realize interactive iterative computation of NMF. They used Paillier homomorphic encryption to protect data privacy and utilized two cloud servers to serially execute NMF tasks. Saha and Imtiaz [24] proposed a privacy-preserving NMF algorithm based on differential privacy. They injected Gaussian noise into gradient computation and modelled the outliers in the data to ensure differential privacy. In summary, the above privacy-preserving outsourcing scheme of NMF does not support parallel outsourcing of multiple servers, which motivates our research on parallel outsourcing. In order to improve the efficiency of the server, we devote to the study of privacy-preserving parallel non-negative matrix factorization with multiple edge servers in this paper.

1.3 Contributions

We propose a privacy-preserving parallel outsourcing scheme for non-negative matrix factorization. Our contributions are as follows:

- To enhance efficiency, we design a parallel outsourcing scheme utilizing multiple nearby edge servers. This scheme divides the computation task into several subtasks using matrix block techniques and employs the multiplicative update rule for NMF parallel computation. This approach not only facilitates NMF parallel computation but also significantly boosts computational efficiency.
- To protect data privacy, we employ random permutation matrix to encrypt the input matrix, ensuring the confidentiality of both the input matrix and output results. Concurrently, to guard against incorrect results from the server, we leverage the iterative nature of NMF to verify results, thereby ensuring their verifiability.

- Through theoretical analysis and experimental comparison, our scheme significantly reduces the client's computation time. Furthermore, parallel computing substantially lightens the computational workload on each edge server.

Organization: The rest of this article is as follows. In Section 2, we present the system model, the threat model and the design goals. In Section 3, we introduce the preliminary content of relevant mathematical knowledge. In Section 4, we construct the scheme of the parallel outsourcing computation of non-negative matrix factorization. Section 5 makes a theoretical analysis of the proposed scheme. Section 6 gives the analysis of the experimental results. Finally we draw the conclusion in Section 7.

2 Models and Definitions

2.1 System Model

As shown in Fig. 1, the system model consists of two types of entities. The first is the client, which wants to complete non-negative matrix factorization tasks but has limited computing power. The second are the edge servers, which have sufficient computing power. In this scenario, the client wants to outsource the NMF task to the edge servers. To protect the privacy of the client, the client first generates a secret key and encrypts the original input matrix. Then the client partitions the encryption matrix and sends the partitioned encrypted matrices to each edge server. The edge servers execute the NMF task in parallel and return the decomposition results W^R and H^R to the client. After receiving the results, the client verifies whether W^R and H^R are valid. The client accepts them if they are valid, and rejects them otherwise. Then it decrypts the results and obtains the matrices W and H.

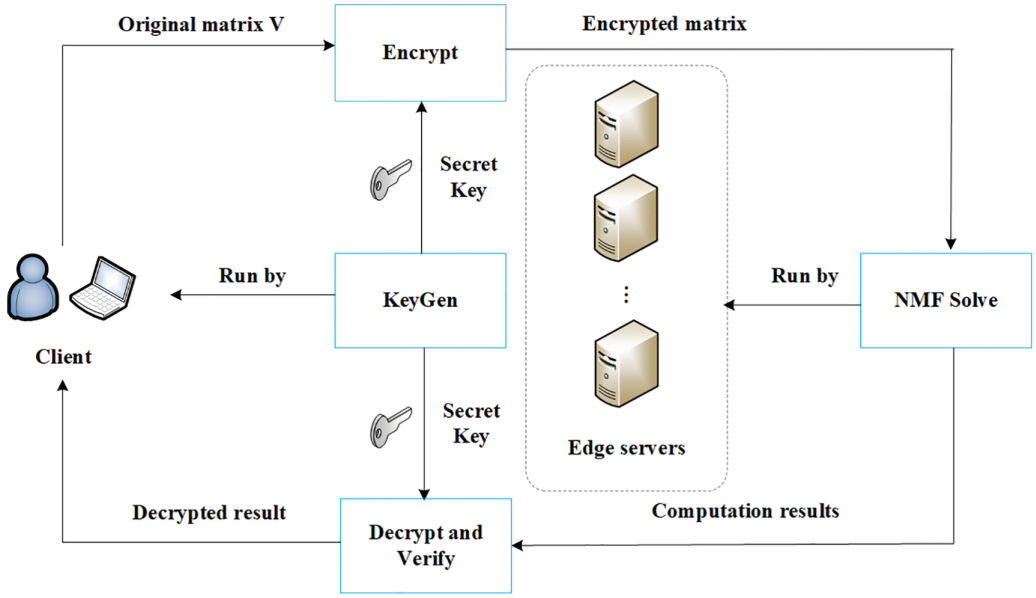

Figure 1: System model.

2.2 Threat Model

Generally speaking, there are three main types of threat models in the field of secure outsourcing computation: "lazy-but-honest", "honest-but-curious" and "fully malicious". We assume that the edge server is fully malicious, which indicates that the server has the strongest attack capability. That is, the edge servers will try to steal the client's data and return the wrong result. So the client needs to process the raw plaintext

data for privacy protection and verify the returned result. In our threat model, the collusion of multiple edge servers is allowed, which requires our scheme to achieve a high level of security.

2.3 Design Goals

The scheme we design should meet the following goals:

- *Correctness*: If the client and the edge servers follow the scheme honestly, the result the client gets should be the correct solution to the original NMF task.
- *Privacy*: The proposed scheme ought to protect the privacy of the client, the edge server can not obtain the knowledge related to the client from the encrypted data.
- *Efficiency*: The client's local computation should be significantly less than what the original NMF would require by itself. The amount of computation in each edge server decreases with the number of servers involving parallel computing increasing.
- *Verifiability*: The client possesses the capability to verify the accuracy of the results received. False results generated by a cheating edge server cannot pass the verification with a non-negligible probability.

2.4 Security Definition

We give the definition of privacy-preserving parallel computation scheme and provide the security definition in terms of privacy. A privacy-preserving parallel computation scheme *PPS* is private, if the PPT adversary $\mathcal{A}$ cannot obtain the input and output data. Here we give the definition of privacy, which are consistent with that in [25]. We define the experiment $Exp_{\mathcal{A}}^{P\,\mathrm{riv}}[PPS, F, \lambda]$ as follows, where $poly()$ is a polynomial.

Experiment $Exp_{\mathcal{A}}^{P\,\mathrm{riv}}[\mathrm{PPS}, \mathrm{F}, \lambda]$

$K \leftarrow KenGen(F, \lambda)$;

For i = 1, . . . , l = $poly\ (\lambda)$

$x_i \leftarrow \mathcal{A}\,(K, x_1, E_{x_1}, \ldots, x_{i-1}, E_{x_{i-1}})$;

$Ex_i \leftarrow Enc_K\,(x_i)$;

$x' \leftarrow Domain(F)$;

$E_{x'} \leftarrow Enc_K\,(x')$;

$\hat{x} \leftarrow \mathcal{A}\,(K, x_1, E_{x_1}, \ldots, x_l, E_{x_l}, E_{x'})$;

If x′ = x̂, *output* ′1′;

Else, output ′0′.

In the above experiment, the adversary $\mathcal{A}$ is granted oracle access to query the public output of Enc on any input, and allowed to query it polynomial times. After the multiple queries, $\mathcal{A}$ tries to acquire the input of a public value $E_{x'}$. If $\mathcal{A}$ can recover x' from $E_{x'}$, then $\mathcal{A}$ succeeds. The advantage of an adversary $\mathcal{A}$ in the above experiment is defined as $Adv_{\mathcal{A}}^{\mathrm{Priv}}\,(PPS, F, \lambda) = Prob\left[Exp_{\mathcal{A}}^{Priv}[PPS, F, \lambda] = 1\right]$.

Definition 1: *A privacy-preserving parallel computation scheme PPS is input-private if for any PPT adversary* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in* $Exp_{\mathcal{A}}^{Priv}[PPS, F, \lambda]$ *is negligible, i.e.,* $Adv_{\mathcal{A}}^{Priv}(PPS, F, \lambda) \leq \varepsilon(\lambda)$.

3 Preliminaries

3.1 Non-Negative Matrix Factorization

For a given non-negative matrix $V \in \mathbb{R}^{m\times n}$, NMF looks for two non-negative matrices $W \in \mathbb{R}^{m\times r}$ and $H \in \mathbb{R}^{r\times n}$ such that

$$V \approx WH, \tag{1}$$

where r is a dimension parameter. The problem of solving NMF can actually be transformed into the following minimization problem

$$\min_{W,H} D(V \mid WH) \text{ s.t. } W \geq 0, H \geq 0. \tag{2}$$

The cost function $D(V \mid WH)$ has many calculation methods, such as squared Euclidean distance, Kullback-Leibler (KL) divergence, and so on. Our scheme uses the squared Euclidean distance as the cost function, which can be expressed as

$$D(V \mid WH) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(V_{ij} - (WH)_{ij}\right)^2, \tag{3}$$

which subjects to $W_{ia} \geq 0$ and $H_{bj} \geq 0, \forall i, a, b, j$. Lee and Seung [26] take squared Euclidean distance as the cost function and put forward the multiplicative update rule of NMF algorithm. In the multiplicative update rule, as long as the initial matrix is non-negative, then the iterative matrices of each round will be non-negative. Specifically, after initializing the non-negative matrices W and H, the following iterative update rule are applied to W and H:

$$W_{ik} = W_{ik}\frac{(VH^{\top})_{ik}}{(WHH^{\top})_{ik}}, \tag{4}$$

$$H_{kj} = H_{kj}\frac{(W^{\top}V)_{kj}}{(W^{\top}WH)_{kj}}. \tag{5}$$

3.2 Kronecker Delta Function

The Kronecker delta function is widely used in engineering and is often employed in matrix construction. The Kronecker delta function is defined as

$$\delta_{x,y} = \begin{cases} 1, & x = y \\ 0, & x \neq y \end{cases}. \tag{6}$$

3.3 Random Permutation Function

Permutation functions have been extensively studied. The permutation function can be expressed as follows:

$$\pi : \begin{pmatrix} 1 & \cdots & n \\ p_1 & \cdots & p_n \end{pmatrix}, \tag{7}$$

where $\pi(i) = p_i (i = 1, ..., n)$. Algorithm 1 can be utilized to generate a random permutation.

Algorithm 1: Random permutation generation.

Input: Integers $1, \ldots, m$.

Output: A permutation π of the integers $1, \ldots, m$.

1: Set $\pi = I_m$.
2: **for** $i = m$ **to** 2 **do**
3: Set j to be a random integer with $1 \leq j \leq i$.
4: Swap $\pi[j]$ and $\pi[i]$.
5: **end for**

4 Scheme Construction

For a non-negative matrix $V \in \mathbb{R}^{m \times n}$, the client aims to outsource the NMF computation task of $V \approx WH$ to multiple edge servers for execution in parallel. We design a privacy-preserving parallel outsourcing scheme for non-negative matrix factorization, which consists of five algorithms as shown below:

- **Key generation:** The client generates the secret key for encryption and decryption.
- **Encryption:** The client encrypts the original input matrix using the secret key. After that, the client partitions the encrypted matrix by row and column. Then client sends them to edge servers.
- **NMF parallel computation:** Edge servers that received the outsourced task performs the iterative calculation of the non-negative matrix factorization in parallel.
- **Result verification:** The client verifies the validity of the result. Accept the result if it passes the verification, reject it otherwise.
- **Decryption:** The client decrypts the verified matrices and obtains the decomposition result of the original non-negative matrix.

The outline of the proposed scheme is presented in Fig. 2. The detailed descriptions of the algorithms are as follows:

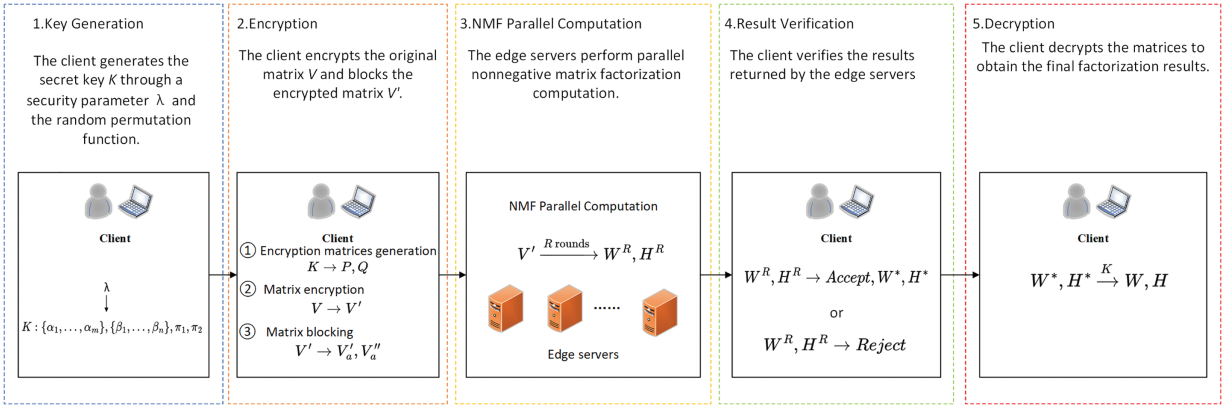

Figure 2: The outline of the proposed scheme.

4.1 Key Generation

The client generates secret key K through Algorithm 2. The secret key K consists of two sets of random numbers $\{\alpha_1, \ldots, \alpha_m\}$, $\{\beta_1, \ldots, \beta_n\}$ and random permutations π_1, π_2. Given a security parameter λ, the client generates two key spaces $\mathcal{K}_\alpha$ and $\mathcal{K}_\beta$. Then, the client generates random permutations π_1 and π_2 through Algorithm 1.

Algorithm 2: Key generation.

Input: A security parameter λ.

Output: The secret key $K : \{\alpha_1, \ldots, \alpha_m\}, \{\beta_1, \ldots, \beta_n\}, \pi_1, \pi_2$.

1: On input a security parameter λ, which specifies two key spaces $\mathcal{K}_\alpha$ and $\mathcal{K}_\beta$, the client picks two sets of random numbers: $\{\alpha_1, \ldots, \alpha_m\} \leftarrow \mathcal{K}_\alpha$ and $\{\beta_1, \ldots, \beta_n\} \leftarrow \mathcal{K}_\beta$, where $0 \notin \mathcal{K}_\alpha \cup \mathcal{K}_\beta$.

2: The client invokes Algorithm 1 to generate two random permutations: $\pi_1 \leftarrow \mathrm{RandP}(1, \ldots, m)$, $\pi_2 \leftarrow \mathrm{RandP}(1, \ldots, n)$.

4.2 Encryption

The client implements the encryption of the original matrix through Algorithm 3. The client first generates random permutation matrices P and Q using the secret key K. The matrices P and Q are both invertible matrices, which can be supported by Theorem 1. Furthermore, their inverse can be expressed as follows:

$$P^{-1}(i,j) = \left(\alpha_j\right)^{-1} \delta_{\pi_1^{-1}(i),j},\ i = 1, 2, \cdots, m.$$

$$Q^{-1}(i,j) = \left(\beta_j\right)^{-1} \delta_{\pi_2^{-1}(i),j},\ j = 1, 2, \cdots\cdots, n.$$

Algorithm 3: Encryption.

Input: The original matrix V and the secret key K: $\{\alpha_1, \ldots, \alpha_m\}, \{\beta_1, \ldots, \beta_n\}, \pi_1, \pi_2$.

Output: The encrypted matrix V'.

1: The client generates matrices P and Q, satisfying $P(i,j) = \alpha_i \delta_{\pi_1(i),j}$ and $Q(i,j) = \beta_i \delta_{\pi_2(i),j}$.

2: The client computes $V' = PVQ^{-1}$.

3: The client divides encrypted matrix V' into p parts by row, i.e., $V' = \begin{bmatrix} V_1' \\ \vdots \\ V_p' \end{bmatrix}$, $V_\alpha' \in \mathbb{R}^{\frac{m}{p} \times n}, 1 \le \alpha \le p$. Then the client sends V_α' to edge server ES_α.

4: The client divides encrypted matrix V' into p parts by column, i.e., $V' = \begin{bmatrix} V_1'' & \cdots & V_p'' \end{bmatrix}$, $V_\alpha'' \in \mathbb{R}^{m \times \frac{n}{p}}$, $1 \le \alpha \le p$. Then the client sends V_α'' to ES_α.

The original input matrix is encrypted by calculating $V' = PVQ^{-1}$. Additionally, the matrix V' can be easily formulated based on Theorem 2. After the original matrix V is encrypted, the client performs matrix partition on the encrypted matrix V'. After the matrix is partitioned by rows and columns, the client sends the partitioned results $V_\alpha' \in \mathbb{R}^{\frac{m}{p} \times n}$ and $V_\alpha'' \in \mathbb{R}^{m \times \frac{n}{p}}$ to the corresponding $ES_\alpha (1 \le \alpha \le p)$. Note that p is the number of edge servers executing the parallel NMF algorithm.

Theorem 1: *The matrices P and Q generated in Algorithm 3 are invertible matrices.*

Proof of Theorem 1: Since $0 \notin \mathcal{K}_\alpha \cup \mathcal{K}_\beta$, the determinants of P and Q satisfy $\det(P) > 0$ and $\det(Q) > 0$. Therefore, P and Q are invertible. □

Theorem 2: *During the encryption process, the entry in i^{th} row and j^{th} column of the encrypted matrix V' can be denoted as:*

$$V'(i,j) = \left(\alpha_i / \beta_j\right) \cdot V\left(\pi_1(i), \pi_2(j)\right).$$

Proof of Theorem 2: Let

$$V = \begin{pmatrix} v_{1,1} & \cdots & v_{1,n} \\ \vdots & \ddots & \vdots \\ v_{m,1} & \cdots & v_{m,n} \end{pmatrix}.$$

Since $P(i,j) = \alpha_i \delta_{\pi_1(i),j}$, we can obtain:

$$PV = \begin{pmatrix} \alpha_1 v_{\pi_1(1),1} & \cdots & \alpha_1 v_{\pi_1(1),n} \\ \vdots & \ddots & \vdots \\ \alpha_i v_{\pi_1(i),1} & \cdots & \alpha_i v_{\pi_1(i),n} \\ \vdots & \ddots & \vdots \\ \alpha_m v_{\pi_1(m),1} & \cdots & \alpha_m v_{\pi_1(m),n} \end{pmatrix}.$$

Due to $Q^{-1}(i,j) = \left(\beta_j\right)^{-1} \delta_{\pi_2^{-1}(i),j}$, we can obtain:

$$PVQ^{-1} = \begin{pmatrix} \alpha_1/\beta_1 v_{\pi_1(1),\pi_2(1)} & \cdots & \alpha_1/\beta_n v_{\pi_1(1),\pi_2(n)} \\ \vdots & \ddots & \vdots \\ \alpha_i/\beta_1 v_{\pi_1(i),\pi_2(1)} & \cdots & \alpha_i/\beta_n v_{\pi_1(i),\pi_2(n)} \\ \vdots & \ddots & \vdots \\ \alpha_m/\beta_1 v_{\pi_1(m),\pi_2(1)} & \cdots & \alpha_m/\beta_n v_{\pi_1(m),\pi_2(n)} \end{pmatrix}.$$

Therefore, $V' = PVQ^{-1}$ can be written as:

$$V'(i,j) = \left(\alpha_i/\beta_j\right) V\left(\pi_1(i), \pi_2(j)\right).$$

By the above calculation, the client can efficiently compute $V\prime$ with computational complexity of $O(mn)$. □

4.3 NMF Parallel Computation

For the encrypted matrix $V\prime$, edge servers perform NMF parallel computing algorithm as shown in Algorithm 4. The edge server ES_α initializes the matrices $W_\alpha^0 \in \mathbb{R}^{\frac{m}{p} \times r}$ and $H_\alpha^0 \in \mathbb{R}^{r \times \frac{n}{p}}$ with random elements greater than or equal to zero, and then computes $L_\alpha = H_\alpha^0 (H_\alpha^0)^\top$. Then ES_α sends the calculated result L_α and the initialized matrix H_α^0 to other edge servers. Meanwhile, ES_α accepts matrices from other edge servers. After receiving matrices from all other servers, matrix L is obtained by aggregating the matrix L_α of each server. At the same time, the matrix H^0 is obtained by splicing H_α^0 by matrix columns. ES_α can then compute

$$w_{ik}^1 = w_{ik}^0 \times \frac{\sum_{j=1}^{n} v'_{ij} \times h_{kj}^0}{\sum_{p=1}^{r} w_{ip}^0 \times l_{pk}}$$

to update the local matrix. In our scheme, the edge servers update the matrix W through the matrix H and then update the matrix H through the matrix W. Therefore, ES_α sends the local matrix W_α^t or H_α^t to other servers every time to update the local matrix as H_α^{t+1} or W_α^{t+1}.

Algorithm 4: NMF parallel computation.

Input: The encrypted matrix $V' \in \mathbb{R}^{m \times n}$, the matrix dimension r, the iteration round R.
Output: The matrices W^R and H^R.

for each edge server $ES_\alpha (1 \le \alpha \le p)$ **do**
  ES_α initializes $W_\alpha^0 \in \mathbb{R}^{\frac{m}{p} \times r}$ with elements $w_{ij}^0 \ge 0$.
  ES_α initializes $H_\alpha^0 \in \mathbb{R}^{r \times \frac{n}{p}}$ with elements $h_{ij}^0 \ge 0$.
  ES_α computes $L_\alpha = H_\alpha^0 (H_\alpha^0)^\top$.
  ES_α sends L_α and H_α^0 to other edge servers.
  ES_α gathers L_α to obtain L by computing $L = L_1 + L_2 + \ldots + L_p$.
  ES_α gathers H_α^0 to obtain H^0 by splicing the matrices by column, i.e., $H^0 = \begin{bmatrix} H_1^0 & \cdots & H_p^0 \end{bmatrix}$.
  ES_α updates local block matrix W_α^0 by computing w_{ik}^1 and sends W_α^1 to other servers.
  ES_α obtains $W^{\{1\}}$ by splicing the matrices by row, i.e., $W^1 = \begin{bmatrix} W_1^1 & \cdots & W_p^1 \end{bmatrix}^\top$.
end for
for $ES_\alpha (1 \le \alpha \le p)$ **do**
  for $t = 1$ **to** R **do**
    If obtained W^t **then**
    // **Update matrix** $\{H\}$
    ES_α computes $S_\alpha = (W_\alpha^t)^\top W_\alpha^t$ and sends it to other servers.
    ES_α obtains S by computing $S = S_1 + S_2 + \ldots + S_p$.
    ES_α updates local matrix H_α^{t-1} by computing h_{kj}^t and sends H_α^t to other servers.
    ES_α obtains H^t by splicing the matrices by column, i.e., $H^t = \begin{bmatrix} H_1^t & \cdots & H_p^t \end{bmatrix}$.
    Else if obtained H^t **and** $t \ne R$ **then**
    // **Update matrix** $\{W\}$
    ES_α computes $L_\alpha = H_\alpha^t (H_\alpha^t)^\top$ and sends it to other servers.
    ES_α obtains L by computing $L = L_1 + L_2 + \ldots + L_p$.
    ES_α updates local matrix W_α^t by computing w_{ik}^{t+1} and sends W_α^{t+1} to other servers.
    ES_α obtains W^{t+1} by splicing the matrices by row, i.e., $W^{t+1} = \begin{bmatrix} W_1^t & \cdots & W_p^t \end{bmatrix}^\top$.
  end for
end for
Edge servers return the results W^R and H^R to client.

After the initialization part over, the iterative update process begin. The turn R of the iteration is entered as a parameter by the client which is usually determined by experience. In the initialization process, each edge server has obtained W^1, so the matrix H can be iterated. At that time, ES_α can calculate $S = (W^t)^\top W^t$. In order to reduce the complexity of calculation, it chooses to calculate $S_\alpha = (W_\alpha^t)^\top (W_\alpha^t)$ and sends it to other servers for aggregation to obtain matrix S. Then ES_α can compute

$$h_{kj}^t = h_{kj}^{t-1} \times \frac{\sum_{i=1}^{m} w_{ik}^t \times v_{ij}''}{\sum_{p=1}^{r} s_{kp} \times h_{pj}^{t-1}}$$

to updates the local matrix and sends it to other servers. After the matrix H^t is obtained through aggregation, the iterative update of W^t to W^{t+1} can be performed. The steps here are similar to those during initialization. Finally, Algorithm 4 outputs the NMF calculation results W^R and H^R.

Remark: *The number of iterations R is a parameter specified by the client based on matrix size and accuracy requirements. In practice, larger matrices and higher accuracy demands require more iterations. We know a common criterion set* $R \geq 20r$. *For small matrices with* $m, n < 1000$, *setting the number of iterations R to a value between 200 and 300 is typically sufficient.*

4.4 Result Verification

The client verifies the results W^R and H^R returned by the edge servers through Algorithm 5. We take advantage of the iterative nature of NMF computation to verify the results with less computational cost. The client takes W^R and H^R received from the edge servers as initial values and executes one iteration of the NMF iterative algorithm to obtain new matrices W^* and H^*. The client can verify the result by using the following inequality

$$\frac{D(V'|W^R H^R)}{D(V'|W^* H^*)} - 1 \leq \epsilon\,, \tag{8}$$

where ε is a threshold. If the above inequality is satisfied, the client accepts the results; otherwise refuses them.

Algorithm 5: Result-verification.

Input: The encrypted matrix V', the returned results W^R and H^R, a threshold ϵ.
Output: Accept or reject.

1: The client obtains two new matrices W^* and H^* by computing $W^* = (W^R)^\top \frac{\left(V'(H^R)^\top\right)}{(W^R H^R (H^R)^\top)}$ and $H^* = (H^R)^\top \frac{\left((W^R)^\top V'\right)}{((W^R)^\top W^R H^R)}$.

2: The client verifies the results by computing whether $\frac{D(V'|W^R H^R)}{D(V'|W^* H^*)} - 1 \leq \epsilon$. If it is true, the client outputs accept; otherwise, it outputs reject.

4.5 Decryption

If the returned result passes verification, the client runs Algorithm 6 with W^* and H^* (instead of W^R and H^R) as the matrices to be decrypted. The output is the final matrix decomposition result. Similar to encryption operation, decryption operation can be calculated as

$$W(i,j) = 1/\alpha_{\pi_1^{-1}(i)} \left(w'_{\pi_1^{-1}(i),j}\right),$$
$$H(i,j) = h'_{i,\pi_2^{-1}(j)} \beta_{\pi_2^{-1}(j)}.$$

By the above calculation, the client can efficiently compute W and H with the complexity computation $O(mr)$ and $O(nr)$.

Algorithm 6: Decryption.

Input: Secret key K, encrypted matrices W^* and H^*.
Output: Matrix factorization results W and H.

1: The client uses the secret key K to compute $W(i,j) = 1/\alpha_{\pi_1^{-1}(i)} \left(w'_{\pi_1^{-1}(i),j}\right)$ and $H(i,j) = h'_{i,\pi_2^{-1}(j)} \beta_{\pi_2^{-1}(j)}$.

5 Theoretical Analysis

In this section, we provide the analysis of our scheme from the aspects of correctness, privacy, verifiability and efficiency.

5.1 Correctness

Theorem 3: *If the client and the edge servers implement the proposed scheme, the client can eventually get the correct non-negative matrix factorization result.*

Proof of Theorem 3: Since the solution of NMF is not necessarily unique, there may exist multiple solutions. Our scheme is based on the multiplicative update rules. The results obtained are not exact solutions. Therefore, we consider the results acceptable when the algorithm performs R rounds of iterations. Next we prove that encryption and decryption do not affect the results of NMF for V. According to Algorithm 3, we have

$$V' = PVQ^{-1} \approx W'H' = PWHQ^{-1}.$$

According to Algorithm 6, we can get $W = P^{-1}W'$ and $H = H'Q$. Obviously, matrices W and H are the factorization results of the original matrix V. This shows that encryption and decryption have no effect on the factorization results of V.

Finally, we need to prove that the solution is acceptable. If the results returned by the server satisfy inequality (8), we consider that the servers correctly perform the required number of iteration rounds R. The results are acceptable to the client. □

5.2 Data Privacy

Theorem 4: *The proposed scheme can protect the privacy of input and output data according to Definition 1. That is, the original matrix V and the factorization results W and H will not be leaked to the edge servers.*

Proof of Theorem 4: First we discuss the privacy of the input matrix V. The elements in the original matrix V are randomly rearranged under two random permutations, denoted as:

$$N(i,j) = V\left(\pi_1(i), \pi_2(j)\right).$$

Here, π_1 is a random permutation with respect to 1, . . ., m, π_2 is a random permutation with respect to 1, . . ., n. Generating such permutations has $m!n!$ results. Each result has a probability of $\frac{1}{m!n!}$. This means that if there is an attack on the encrypted matrix N, the probability that the attacker gets V from N by guessing π_1 and π_2 is $\frac{1}{m!n!}$. In addition, the elements of the matrix V are blinded by a factor, denoted as:

$$\begin{aligned} V'(i,j) &= \left(\alpha_i/\beta_j\right) \cdot V\left(\pi_1(i), \pi_2(j)\right) \\ &= \left(\alpha_i/\beta_j\right) \cdot N(i,j). \end{aligned} \tag{9}$$

This means that even if the malicious edge servers get the correct matrix $V'(i,j)$ and the random arrangement π_1, π_2, the expected probability for a brute force attack on the key space to get $\{\alpha_1, \ldots, \alpha_m\}$ and $\{\beta_1, \ldots, \beta_n\}$ is $\frac{1}{|K_\alpha|^m |K_\beta|^n}$.

Combining both factors, the total probability that an adversary successfully recovers V from V' is:

$$\Pr[\text{success}] \leq \frac{1}{m! \cdot n! \cdot |\mathcal{K}_\alpha|^m \cdot |\mathcal{K}_\beta|^n} = \text{negl}(\lambda) \tag{10}$$

In addition, when an outsourced non-negative matrix factorization is performed on a new input matrix, the secret key is regenerated. Therefore, without the key, the edge servers cannot get the original input from the outsourced task.

Next, we discuss the situation of the collusion among edge servers. Throughout the scheme, each edge server ES_α possesses only a part of the matrix V'. And even if they conspired to obtain the entire encrypted matrix $V\prime$, it would be difficult to recover the original matrix V without knowing the secret key. In the algorithm, the matrices W and H are randomly generated by edge servers and iteratively updated in the subsequent process, so the information of the original matrix V will not be leaked. Finally, we prove the output privacy of W^R and H^R. That is, the attacker cannot get the matrices W and H based on W^R and H^R. According to Algorithm 6, $W(i,j) = 1/\alpha_{\pi_1^{-1}(i)}\left(w'_{\pi_1^{-1}(i),j}\right)$ and $H(i,j) = h'_{i,\pi_2^{-1}(j)}\beta_{\pi_2^{-1}(j)}$. Therefore, without the key, the edge servers cannot get the output matrices from the outsourcing task.

Besides, the transformation $V' = PVQ^{-1}$ is a linear operation and preserves certain statistical structures. The main leakage pathway is that zero elements are preserved as $V(i,j) = 0 \Leftrightarrow V'(\pi_1^{-1}(i), \pi_2^{-1}(j)) = 0$. Due to most of the original matrices used for NMF are dense matrices, the number of zero elements in the matrix is not considered to leak statistical information. □

5.3 Verifiability

Theorem 5: *A malicious edge server may return incorrect results. In our scheme, the correct results must pass the verification and the incorrect results must not pass the verification.*

Proof of Theorem 5: According to the convergence property of the multiplicative update rule, $D(V'|W^{t+1}H^{t+1}) \le D(V'|W^tH^t)$ for all t. Therefore, after one additional iteration on (W^R, H^R), we obtain (W^*, H^*) with $D(V'|W^*H^*) \le D(V'|W^RH^R)$. This guarantees:

$$0 \le \frac{D(V'|W^RH^R)}{D(V'|W^*H^*)} - 1 \le \epsilon \tag{11}$$

For properly chosen ϵ, honest results satisfy the inequality. □

Theorem 6: *Unless (W_R, H_R) is a near-stationary point of the NMF that is consistent with the multiplicative update dynamics on V', the probability of passing verification is negligible.*

Proof of Theorem 6: In our scheme, a dishonest edge server that skips the R iterations does not have access to the intermediate states generated during honest execution. As a result, unless the returned (W_R, H_R) is deliberately constructed to approximate a stationary point of the NMF objective on $V\prime$, the acceptance condition is unlikely to be satisfied. We further observe that constructing a near-stationary solution without following the update trajectory essentially requires solving the NMF problem on the encrypted matrix $V\prime$. This effort is comparable to that of honestly performing the outsourcing computation. Hence, any server that attempts to avoid this computation will fail the verification except with negligible probability. □

5.4 Computational Complexity

The computational complexity analysis is divided into two parts: Client-side computational complexity and server-side computational complexity. We first analyze the client-side computational complexity. The client-side computational complexity mainly arises from the key generation, encryption, result-verification and decryption algorithms. In the key generation algorithm, the complexity mainly comes from generating random permutations π_1 and π_2. This computational complexity is $O(m+n)$. In the encryption algorithm, the client performs encryption based on random permutation matrix. This computational complexity is

$O(mn)$. Similarly, in the decryption algorithm, the client performs decryption based on random permutation matrix. This computational complexity is $O(mr + nr)$. Verification requires an additional round of iterations by the client with a complexity of $O(mnr)$. In summary, the total computational complexity of the client is $O(mnr)$.

Next, we analyze the server-side computational complexity. The server-side computational complexity mainly arises from the NMF parallel computation. In each iteration of NMF parallel computation algorithm, matrices W and H are updated by calculating matrix multiplication. The computational complexity of calculating matrix W is $O(mnr)$. The computational complexity of calculating matrix H is $O(mnr)$. Since we adopt NMF parallel computation, each edge server only needs to undertake part of the computing tasks. By distributing the NMF computation tasks to p edge servers, the parallel computation is realized and the complexity is reduced. Specifically, the computational complexity of each edge server updating its own portion of the matrix is $O(R * mnr/p)$.

Finally, we compare the time complexity of our proposed scheme with that of representative privacy-preserving NMF outsourcing algorithms in recent years and present the results in Table 1. Here, m and n denote the dimensions of the original matrix V, r represents the dimension parameter, and R signifies the number of iterations. It is evident that our scheme effectively improves the efficiency on the edge server side. This means that edge servers can respond to client's needs more quickly.

Table 1: Computational complexity comparsion.

Computational Complexity	Client				Server
	KeyGen	***Encrypt***	***Decrypt***	***Verify***	***Computation***
Liu et al. [23]	$O(m+n)$	$O(mn)$	$O(mr+nr)$	$O(mn)$	$O(R*mnr)$
Duan et al. [12]	$O(m+n)$	$O(mn)$	$O(mr+nr)$	$O(mnr)$	$O(R*mnr)$
Fu et al. [11]	$O(m+n)$	$O(mnr)$	$O(mr^2+nr^2)$	$O(mr^2+nr^2)$	$O(R*mnr)$
Our scheme	$O(m+n)$	$O(mn)$	$O(mr+nr)$	$O(mnr)$	$O(R*mnr/p)$

5.5 Communication Complexity

The computational complexity analysis is divided into two parts: communication between the client and the edge server, communication among edge servers. We first analyze the communication complexity between the client and the edge server. In the encryption phase, the client encrypts the original matrix and partitions the encrypted matrix V' by rows and columns. The client sends matrices $V'_\alpha \in \mathbb{R}^{\frac{m}{p}\times n}$ and $V''_\alpha \in \mathbb{R}^{m\times\frac{n}{p}}$ to ES_α. Therefore, the amount of data transmitted from the client to all edge servers is $O(mn)$. After completing the parallel NMF computation, edge servers return the final factorization results $W_R \in \mathbb{R}^{m\times r}$ and $H_R \in \mathbb{R}^{r\times n}$ to the client. The communication complexity is $O(mr + nr)$, which is significantly smaller than $O(mn)$ when $r \ll \min(m, n)$.

Next, we analyze the communication complexity among edge servers in Algorithm 4. In the initialization phase of Algorithm 4, each edge server initializes $H^0_\alpha \in \mathbb{R}^{r\times\frac{n}{p}}$ and computes $L_\alpha = H^0_\alpha (H^0_\alpha)^\top \in \mathbb{R}^{r\times r}$. The matrices L_α and H^0_α are exchanged among the edge servers. Thus, the communication cost of the initialization phase for edge servers is $O\left(pr^2 + rn\right)$. In the iterative computation phase, the edge servers alternately update matrices W and H. When updating matrix W, each edge server computes $L_\alpha = H^t_\alpha (H^t_\alpha)^\top \in \mathbb{R}^{r\times r}$ and sends it to other servers. They also sends its local block $W^{t+1}_\alpha \in \mathbb{R}^{\frac{m}{p}\times r}$. The communication cost of this step is $O\left(pr^2 + mr\right)$. When updating matrix H, each edge server computes $S_\alpha = (W^t_\alpha)^\top W^t_\alpha \in \mathbb{R}^{r\times r}$ and sends it to other servers. They also sends its local block $H^t_\alpha \in \mathbb{R}^{r\times\frac{n}{p}}$. The communication cost of this step is $O\left(pr^2 + rn\right)$.

Therefore, the communication complexity of each edge server in one iteration is $O\left(2pr^2 + r(m+n)\right)$ and the total communication complexity among edge servers over R iterations is $O\left(R\left(2pr^2 + r(m+n)\right)\right)$.

It should be noted that our computation is "embarrassingly parallel". After the initial matrix partitioning, each edge server can independently compute its local matrix blocks W_α and H_α with only periodic synchronization required for aggregating matrices $L_\alpha \in \mathbb{R}^{r\times r}$ and $S_\alpha \in \mathbb{R}^{r\times r}$. Besides, since our scheme targets edge computing environments where servers connected via high-speed networks (e.g., 5G), the inter-server communication latency is very low [27,28]. In this discussion, we disregard communication delays, as computation time predominates.

6 Performance Evaluation

In this section, we conduct the experiment to evaluate the performance of the proposed scheme. We use the Numpy library and Pool library in Python to implement our scheme and test it on a laptop with an Intel Core i5 CPU and 16 GB RAM. The reason for adopting the Pool library is that our computational tasks belong to the "embarrassingly parallel" category, with low communication requirements between tasks. Therefore, we chose the Pool library over MPI-based multiprocessing.

First, we measure the time to resolve the original NMF task locally on the client, expressed as T_{original}. Next, we measure the time to execute the NMF parallel computation on the edge servers, expressed as T_{server}. We define T_{server} as the execution time of a single edge server. Finally, we measure the time to perform key generation, encryption, result verification, decryption on the client, expressed as T_{client}.

At the same time, we define $T_{\text{original}}/T_{\text{client}}$ as the client computing advantage. This value should be as large as possible, representing more computing resources saved by the client. It should be noted that in this experiment, we set the number of iteration rounds R to 1000 and the threshold ϵ to 10^{-5}. By conducting experiments with the original matrix V across varying dimensions m and n, we measure the resulting changes in the execution time described above. The results are presented in Table 2. The results presented in Table 2 indicates that the execution time of the client is significantly shorter than the original task's duration. The dimension parameter r is set to 40 and the number of servers participating in parallel computation is set to 4.

Table 2: Comparison of execution time (in Seconds).

m	n	$T_{original}$	T_{client}	T_{edge}	$T_{original}/T_{client}$
3000	2400	128.9449	2.7155	62.0353	47.4848
6000	4800	445.5048	10.2086	121.8091	43.6337
10000	8000	1544.7122	44.4903	278.7890	34.7204
20,000	16,000	4345.4866	134.6020	1802.0270	32.2839

The time overhead is significantly influenced by the size of the original input matrix. We keep the row value m of the original matrix fixed at 3000 and vary the range of column values n from 500 to 3000. As depicted in Fig. 3, it can be observed that both the edge servers and the client experience an increase in computing time, with a greater impact on the edge servers. Subsequently, we calculate how the client computing advantage changes with variations in column size, and the results are shown in Fig. 4. The results show that with the increase of column value, the client computing advantage has an increasing trend.

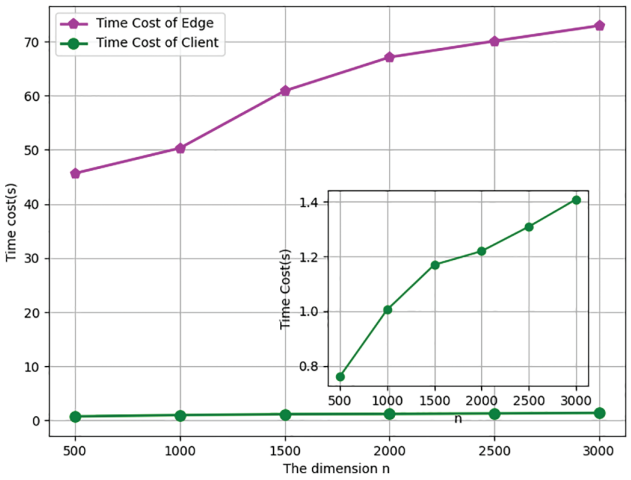

Figure 3: The time cost varying with column size n when m = 3000.

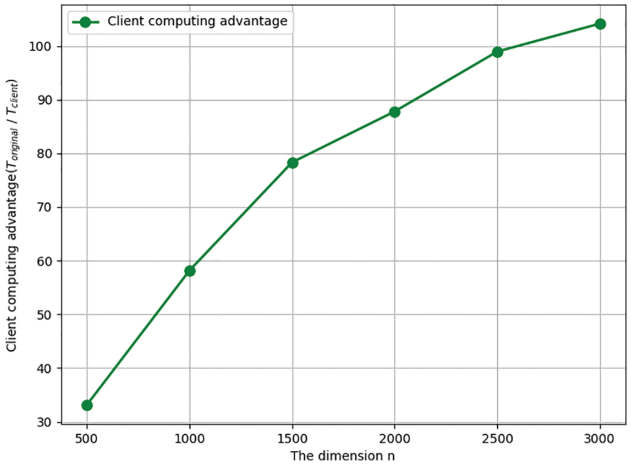

Figure 4: Client computing advantage varying with n when m = 3000.

Meanwhile, we keep the column value n of the original matrix fixed at 3000 and vary the range of row values m. As depicted in Fig. 5, it is evident that both the edge servers and the client experience an increase in computing time, with a greater impact on the edge servers. Subsequently, we calculate how the client computing advantage changes with variations in row size, and the results are shown in Fig. 6. The results show that with the increase of row value, the client computing advantage has an increasing trend.

In NMF tasks, the dimension parameter r plays a crucial role. To evaluate its impact on performance, we vary the value of r while keeping the size of V constant. Specifically, we set m = 3000 and n = 2400, and define the range of r as 20 to 200. We record the computing time for both client and the edge servers, as depicted in Fig. 7. It is evident that an increase in r leads to longer computing time for both client and edge servers. Building upon this observation, we calculate the client computing advantage. The results are shown in Fig. 8, which demonstrate a positive correlation between the client computing advantage and the values of r.

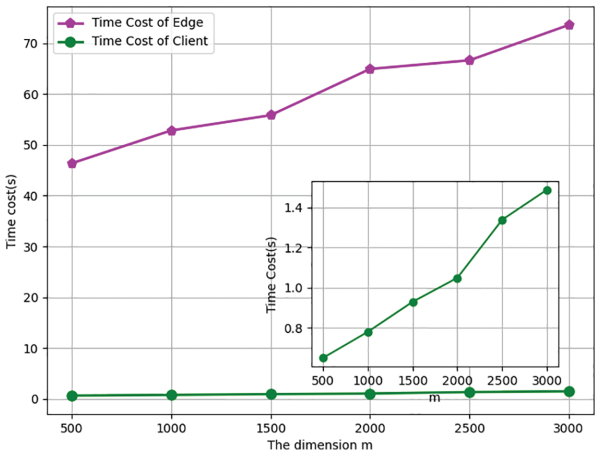

Figure 5: The time cost varying with row size m when n = 3000.

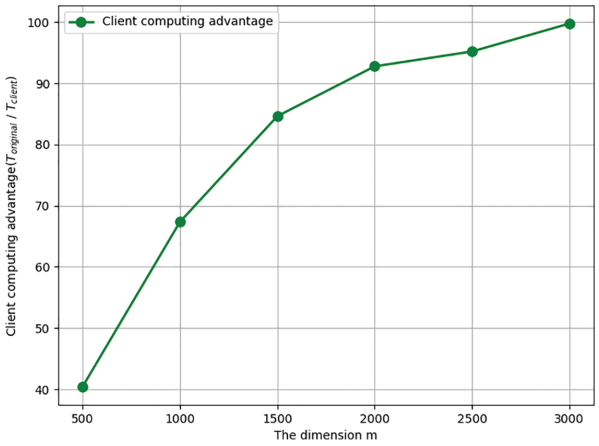

Figure 6: Client computing advantage varying with m when n = 3000.

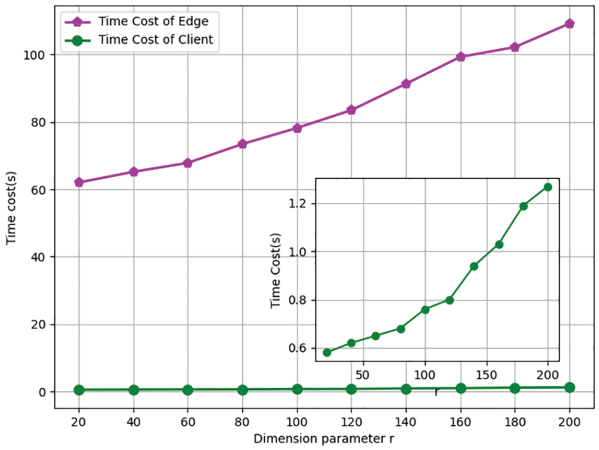

Figure 7: The time cost varying with dimension parameter.

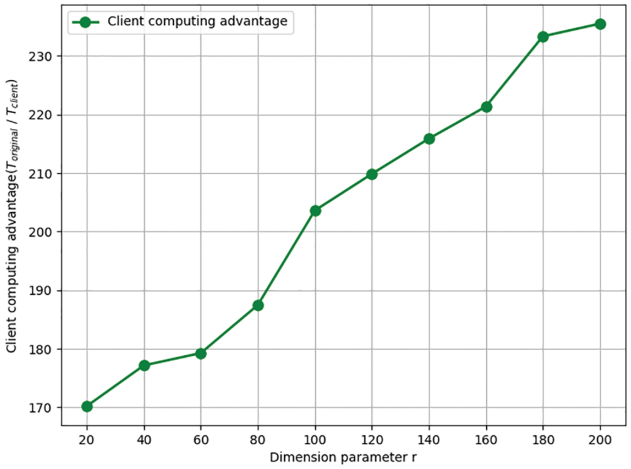

Figure 8: Client computing advantage varying with dimension parameter.

At the same time, we test the trend of average computation time per server as the number of edge servers involved in the computation increased. The results are shown in Fig. 9. The row size m of matrix V is set as 20,000, the column size n is set as 16,000 and the dimension parameter r is set as 40. As can be seen, the computation time per server decreases significantly as the number of invested servers increases.

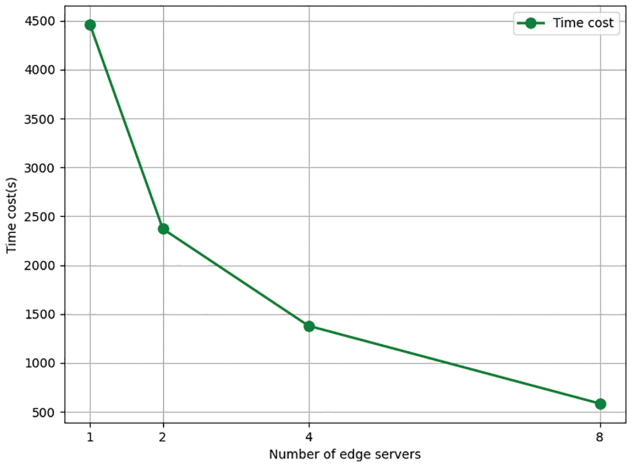

Figure 9: Relationship between the number of servers and the time cost of each server.

Additionally, we report the convergence trajectory of $\text{Norm}(V - WH)$, where V is the original matrix and WH is the reconstructed result at each iteration round. The results are shown in Fig. 10. The reconstruction norm consistently decreases as the iteration round increases and becomes nearly stable after sufficient iterations, indicating a steady convergence behavior of our iterative optimization.

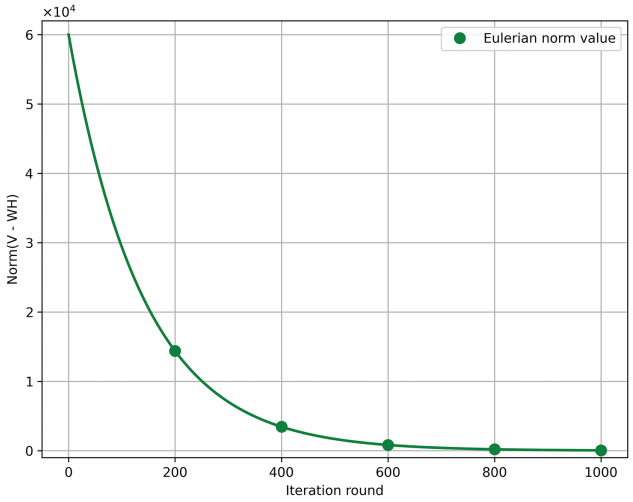

Figure 10: Eulerian norm value of (V − WH) comparison.

7 Conclusion

In this paper, we propose a privacy-preserving parallel non-negative matrix factorization outsourcing scheme with edge computing. The scheme divides the computational task into multiple sub-tasks using matrix block techniques and designs a parallel NMF computation scheme based on the multiplicative update rule, thereby improving the computational efficiency of edge servers. This means that client's needs can be quickly satisfied. Additionally, we employ random permutation matrices to protect data privacy and leverage the iterative nature of NMF for result verification. The experimental results and theoretical analysis demonstrate the superiority of the proposed scheme in terms of both computational efficiency and privacy protection.

Acknowledgement: Not applicable.

Funding Statement: This research was supported in part by Shandong Provincial Natural Science Foundation under Grant (ZR2024MF038), and Qingdao Natural Science Foundation (25-1-1-103-zyyd-jchZ).

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Wenxuan Yu and Wenjing Gao; methodology, Jiuru Wang, Jia Yu; validation, Wenxuan Yu, Jia Yu and Rong Hao; formal analysis, Wenjing Gao; data curation, Wenxuan Yu; review and editing, Wenjing Gao and Jia Yu; supervision, Rong Hao and Jia Yu. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Langville AN, Meyer CD, Albright R, Cox J, Duling D. Algorithms, initializations, and convergence for the nonnegative matrix factorization. arXiv:1407.7299. 2014.
2. Xu W, Liu X, Gong Y. Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY, USA: ACM; 2003. p. 267–73.

3. Liu H, Wu Z, Li X, Cai D, Huang TS. Constrained nonnegative matrix factorization for image representation. IEEE Trans Pattern Anal Mach Intell. 2012;34(7):1299–311. doi:10.1109/tpami.2011.217.
4. Yang CF, Ye M, Zhao J. Document clustering based on nonnegative sparse matrix factorization. In: Advances in Natural Computation (ICNC 2005). Vol. 3611. Berlin/Heidelberg, Germany: Springer; 2005. p. 557–63.
5. Shahnaz F, Berry MW, Pauca VP, Plemmons RJ. Document clustering using nonnegative matrix factorization. Inf Process Manag. 2006;42(2):373–86 doi:10.1016/j.ipm.2004.11.005.
6. Guo Y, Li Q, Liang C. The rise of nonnegative matrix factorization: algorithms and applications. Inf Syst. 2024;123:102379.
7. Wang H, He D, Fu A, Li Q, Wang Q. Provable data possession with outsourced data transfer. IEEE Trans Serv Comput. 2021;14(6):1929–39 doi:10.1109/tsc.2019.2892095.
8. Shan Z, Ren K, Blanton M, Wang C. Practical secure computation outsourcing: a survey. ACM Comput Surv. 2018;51(2):31:1–31:40. doi:10.1145/3158363.
9. Zhang H, Gao P, Yu J, Lin J, Xiong NN. Machine learning on cloud with blockchain: a secure, verifiable and fair approach to outsource the linear regression. IEEE Trans Netw Sci Eng. 2022;9(6):3956–67.
10. Gao W, Yu J, Yang M, Wang H. Enabling privacy-preserving parallel outsourcing matrix inversion in IoT. IEEE Internet Things J. 2022;9(17):15915–27 doi:10.1109/jiot.2022.3150956.
11. Fu A, Chen Z, Mu Y, Susilo W, Sun Y, Wu J. Cloud-based outsourcing for enabling privacy-preserving large-scale non-negative matrix factorization. IEEE Trans Serv Comput. 2022;15(1):266–78 doi:10.1109/tsc.2019.2937484.
12. Duan J, Zhou J, Li Y. Secure and verifiable outsourcing of large-scale nonnegative matrix factorization (NMF). IEEE Trans Serv Comput. 2021;14(6):1940–53 doi:10.1109/tsc.2019.2911282.
13. Lei X, Liao X, Huang T, Li H, Hu C. Outsourcing large matrix inversion computation to a public cloud. IEEE Trans Cloud Comput. 2013;1(1):78–87. doi:10.1109/tcc.2013.7.
14. Lei X, Liao X, Huang T, Heriniaina F. Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud. Inf Sci. 2014;280:205–17 doi:10.1016/j.ins.2014.05.014.
15. Fu S, Yu Y, Xu M. A secure algorithm for outsourcing matrix multiplication computation in the cloud. In: Proceedings of the Fifth ACM International Workshop on Security in Cloud Computing. New York, NY, USA: ACM; 2017. p. 27–33.
16. Lei X, Liao X, Huang T, Li H. Cloud computing service: the case of large matrix determinant computation. IEEE Trans Serv Comput. 2015;8(5):688–700.
17. Liu J, Bi J, Li M. Secure outsourcing of large matrix determinant computation. Frontiers Comput Sci. 2020;14(1):146807 doi:10.1007/s11704-019-9189-7.
18. Zhou L, Li C. Outsourcing eigen-decomposition and singular value decomposition of large matrix to a public cloud. IEEE Access. 2016;4:869–79 doi:10.1109/access.2016.2535103.
19. Luo C, Zhang K, Salinas S, Li P. SecFact: secure large-scale QR and LU factorizations. IEEE Trans Big Data. 2021;7(4):796–807.
20. Li Y, Zhang H, Lin J, Liang F, Xu H, Liu X, et al. Secure edge-aided singular value decomposition in internet of things. IEEE Internet Things J. 2024;11(13):23207–21 doi:10.1109/jiot.2024.3375394.
21. Liu C, Hu X, Chen X, Wei J, Liu W. SDIM: a subtly designed invertible matrix for enhanced privacy-preserving outsourcing matrix multiplication and related tasks. IEEE Trans Dependable Secur Comput. 2024;21(4):3469–86.
22. Gao Z, Su Q. Enabling verifiable and traceable distributed outsourced matrix multiplication in neural networks. Cluster Comput. 2025;28(11):722 doi:10.1007/s10586-025-05472-0.
23. Liu Z, Li B, Han Q. Secure and verifiable outsourcing protocol for non-negative matrix factorisation. Int J High Perform Comput Netw. 2018;11(1):14–23 doi:10.1504/ijhpcn.2018.088875.
24. Saha S, Imtiaz H. Privacy-preserving non-negative matrix factorization with outliers. ACM Trans Knowl Discov Data. 2024;18(3):1–26. doi:10.1145/3632961.
25. Tian C, Yu J, Zhang H, Xue H, Wang C, Ren K. Novel secure outsourcing of modular inversion for arbitrary and variable modulus. IEEE Trans Serv Comput. 2022;15(1):241–53 doi:10.1109/tsc.2019.2937486.

26. Lee DD, Seung HS. Algorithms for non-negative matrix factorization. In: Advances in Neural Information Processing Systems. Vol. 13. Cambridge, MA, USA: MIT Press; 2000. p. 556–62.
27. Guo L, Yu J, Yang M, Kong F. Privacy-preserving convolution neural network inference with edge-assistance. Comput Secur. 2022;123:102910 doi:10.1016/j.cose.2022.102910.
28. Shen W, Yin B, Cao X, Cheng Y, Shen XS. A distributed secure outsourcing scheme for solving linear algebraic equations in Ad Hoc clouds. IEEE Trans Cloud Comput. 2019;7(2):16 doi:10.1109/tcc.2016.2647718.