SEF-UQR: Scalable and Efficient Privacy-Preserving Federated Updating QR Factorization

Haonan Yuan
yuanhaonan.cigit.ac.cn
Chongqing Key Laboratory of Secure Computing for Biology, Chongqing Institute of Green and Intelligent Technology, CAS
Chongqing, China

Wenyuan Wu*
wuwenyuan@cigit.ac.cn
Chongqing Key Laboratory of Secure Computing for Biology, Chongqing Institute of Green and Intelligent Technology, CAS
Chongqing, China

Jingwei Chen
chenjingwei@cigit.ac.cn
Chongqing Key Laboratory of Secure Computing for Biology, Chongqing Institute of Green and Intelligent Technology, CAS
Chongqing, China

Abstract

Applications in real-time machine learning and data analysis often require incremental updates to matrix decompositions as new data arrive. This capability is particularly crucial for streaming PCA, online learning, and iterative optimization algorithms, where data are continuously generated from distributed sources. However, privacy constraints prevent direct data sharing among participants, making collaborative QR decomposition updates challenging. To address this, we present **SEF-UQR**, a scalable and efficient framework for federated QR updates that focuses on incremental row-addition updates—common in streaming-data scenarios—while leveraging homomorphic encryption and interactive ciphertext protocols to protect both inputs and intermediate computations. SEF-UQR achieves accuracy on par with insecure recomputation, maintaining a mean squared error (MSE) below 1e-12. Empirical results demonstrate that SEF-UQR delivers at least a 10× runtime improvement over existing state-of-the-art methods employing fully homomorphic encryption, confirming its effectiveness for privacy-sensitive, real-time federated data analysis.

CCS Concepts

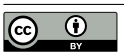
• **Security and privacy → Privacy-preserving protocols**.

Keywords

Privacy-preserving computation, Updating QR, Streaming data, Matrix factorization, Distributed dataset

ACM Reference Format:
Haonan Yuan, Wenyuan Wu, and Jingwei Chen. 2025. SEF-UQR: Scalable and Efficient Privacy-Preserving Federated Updating QR Factorization. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3746252.3761204

*Corresponding author.



CIKM '25, Seoul, Republic of Korea

ACM ISBN 979-8-4007-2040-6/2025/11
https://doi.org/10.1145/3746252.3761204

1 Introduction

QR decomposition is a fundamental operation in numerical linear algebra, factoring a matrix A into an orthonormal matrix Q and an upper-triangular matrix R. Intuitively, Q provides an orthogonal basis for the column space of A, while R encodes the corresponding coefficients in triangular form. Beyond classical use in solving linear systems, least squares regression, and eigenvalue problems [20], QR factorization underpins many modern data analysis and machine learning tasks. In particular, streaming PCA and other online dimensionality reduction techniques exploit QR to orthogonalize updates, enabling real-time processing of continuous data streams [7, 31]. Maintaining an orthonormal basis is thus crucial for incremental algorithms in domains such as IoT sensing, healthcare monitoring, and recommendation systems. In this work, we focus on the prevalent case of incremental updates via row additions, which matches typical streaming-data patterns.

Recomputing a full QR factorization whenever new data are appended is computationally prohibitive [8], especially in continuous streaming scenarios. A key advantage of QR is that it supports efficient updates when new observations arrive, avoiding full refactorization [7]. Many algorithms exploit rank-one QR updates to reduce runtime while preserving accuracy; for example, the incremental LDA method (IDR/QR) achieves accuracy comparable to batch SVD methods but with substantially lower computational cost by updating the QR factors on insertion of new data [31].

However, existing QR update methods assume full access to raw data and thus offer no inherent privacy protection. In many collaborative or distributed environments (e.g., federated recommendation systems or multi-institution analytics), data cannot be centralized due to privacy constraints. While federated learning (FL) enables training across decentralized data silos without sharing raw data [2], it does not directly support QR updates, and even model updates in FL can leak sensitive information [17, 33].

Secure multi-party computation (SMPC) allows joint computation over private inputs but typically incurs high communication and coordination costs, limiting practicality for large-scale or high-frequency updates [5]. Consequently, no existing distributed learning framework efficiently supports privacy-preserving incremental QR factorization, leaving a gap in current research.

In this work, we address this open problem by leveraging Fully Homomorphic Encryption (FHE) for privacy-preserving QR updates. FHE enables computations directly on encrypted data without ever decrypting it during processing [24]. Compared to federated learning or SMPC-based approaches, FHE enables computation

over encrypted data without exposing intermediate results such as gradients, which in federated settings may leak private information unless protected by mechanisms like differential privacy [14]. As a result, FHE requires only minimal ciphertext exchange and reduces communication overhead [18]. By performing the QR update on encrypted matrices, our framework ensures that each participant's sensitive data remains confidential throughout the computation and mitigates computational cost by employing rotation packing optimizations.

In summary, our proposed solution allows multiple parties to collaboratively update a QR decomposition for incremental dimensionality reduction without exposing their private data. The main contributions of this paper are: (i) We propose SEF-UQR, a secure and efficient framework for distributed QR updates that preserves data privacy using homomorphic encryption (HE) combined with lightweight secure interaction protocols; (ii) We design exact encrypted Givens rotations and an interactive protocol for non-linear operations, enabling high-precision updates without error accumulation; (iii) We introduce SIMD-based rotation packing to significantly improve efficiency, supporting both row-wise and block-wise updates for different application constraints.

2 Related Work

2.1 Centralized Privacy-Preserving Matrix Decomposition

Several works have explored matrix decomposition in centralized settings using homomorphic encryption (HE). One line of research proposes directly executing PCA over encrypted data by carefully optimizing arithmetic operations such as matrix multiplication and eigenvalue approximation [9, 23]. Some of these approaches improve runtime by leveraging the CKKS [10] scheme for real-valued operations, or by preconditioning the data to reduce multiplicative depth [25, 32].

These centralized solutions typically assume that all encrypted data are uploaded to a single computing server, which then performs the entire matrix decomposition homomorphically. While they achieve strong data confidentiality, their scalability is limited: uploading large encrypted datasets incurs significant communication overhead, and homomorphic computation over full matrices remains costly. Moreover, most methods adopt iterative optimization for decomposition, which introduces numerical instability due to the inherent approximation error in evaluating non-linear functions (e.g., square roots) under CKKS.

2.2 Distributed Privacy-Preserving Matrix Decomposition

To address scalability, several recent works propose distributed or federated matrix decomposition protocols. Some frameworks use multiparty HE and edge-side processing to allow participants to contribute local data or intermediate results without revealing raw inputs [16, 30]. Others rely on decentralized protocols for secure SVD or PCA computation by jointly aggregating encrypted contributions from multiple users [22].

While these distributed solutions reduce the burden on centralized servers and better align with real-world federated settings, they still face several limitations. First, they often require full recomputation of matrix decompositions when new data arrive, leading to inefficiency in continuous learning scenarios. Second, like their centralized counterparts, these protocols rely on iterative numerical procedures that accumulate error over time, especially when approximating non-linear operations on ciphertexts. Lastly, they generally do not support efficient incremental updates—every recomputation starts from scratch.

In summary, existing HE-based matrix factorization methods generally suffer from two critical limitations: (1) errors arising from inaccurate evaluation of nonlinear functions, compounded progressively by iterative computations; and (2) lack of incremental update capability, leading to inefficiencies due to repeated recomputation under frequent, real-time data updates. Our proposed framework, **SEF-UQR**, is designed to address these limitations. By leveraging structured Givens rotations and a secure interactive protocol for precise inverse square root computation, SEF-UQR enables accurate, efficient, and privacy-preserving QR updates over encrypted data. It further provides two complementary update strategies (row-wise and block-wise), allowing system designers to balance computational cost and communication overhead based on practical deployment needs.

3 Preliminaries

3.1 Notation

We use boldface uppercase and lowercase letters to denote matrices and vectors, respectively, with their dimensions annotated as $\square^{(m\times n)}$; for example, $\boldsymbol{A}^{(m\times n)}$ denotes an $m \times n$ matrix and $\boldsymbol{v}^{(1\times n)}$ denotes a $1\times n$ row vector. The notation $\boldsymbol{A}[i,:]$ refers to the i-th row of matrix $\boldsymbol{A}$, while $\boldsymbol{v}[i]$ denotes the i-th element of vector $\boldsymbol{v}$. We use $c_{\square}$, such as in $c_{\boldsymbol{A}}$ or $c_{\boldsymbol{v}}$, to represent ciphertext matrices and vectors. Finally, the operator $[\cdot]$ denotes element-wise multiplication between vectors; that is, $\boldsymbol{v}\cdot\boldsymbol{u} = [v_1u_1, v_2u_2, \ldots, v_nu_n]$.

3.2 QR Factorizations and Givens Rotation

QR decomposition factors a real matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ (with $m \geq n$) into $\boldsymbol{A} = \boldsymbol{QR}$, where $\boldsymbol{Q} \in \mathbb{R}^{m\times n}$ satisfies $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_n$, and $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is upper-triangular. Classical methods like Householder reflections and Gram-Schmidt orthogonalization have $O(n^3)$ time complexity for dense matrices.

Givens rotation [19] offers an alternative by applying a sequence of plane rotations to zero out subdiagonal elements, making it well-suited for sparse or structured matrices. It also enables efficient incremental updates and is amenable to parallelization. In this work, we adopt Givens rotations as the basis for secure and efficient QR updates over encrypted data.

3.3 Updating Matrix Factorizations

In many real-world applications, once the QR decomposition of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is available, it is often necessary to update the decomposition when new data arrive or rows/columns are removed. A typical case is appending a new row to $\boldsymbol{A}$. Suppose the original decomposition is given by $\boldsymbol{A} = \boldsymbol{QR}$. Then, for the augmented matrix:

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{w}^T \\ \boldsymbol{A} \end{bmatrix} = \tilde{\boldsymbol{Q}}\tilde{\boldsymbol{R}}, \quad \boldsymbol{w} \in \mathbb{R}^n,$$

one can efficiently update the QR decomposition without recomputing it from scratch. As described in [20], by left-multiplying $\tilde{\boldsymbol{A}}$ with $\text{diag}(1, \boldsymbol{Q}^T)$, we obtain:

$$\text{diag}(1, \boldsymbol{Q}^T)\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{w}^T \\ \boldsymbol{R} \end{bmatrix} = \boldsymbol{H},$$

where $\boldsymbol{H}$ is an $(m+1) \times n$ upper Hessenberg matrix. The new QR decomposition can then be obtained by applying a sequence of Givens rotations to $\boldsymbol{H}$, reducing it to upper-triangular form. This update process has a time complexity of $O(n^2)$ and is significantly more efficient than recomputing the QR decomposition from scratch (which requires $O(n^3)$ time). Additionally, inserting a new row at any arbitrary position while preserving the Hessenberg structure can be done by applying an appropriate row permutation before the update.

3.4 Block Matrix Factorizations

Following the QR decomposition strategy in multi-core environments as proposed in [3, 11, 13, 26], we consider a tall-and-skinny matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m \gg n$. Its full QR decomposition is given by

$$\boldsymbol{A} = \boldsymbol{Q}_{\text{global}} \boldsymbol{R}_{\text{global}},$$

where $\boldsymbol{Q}_{\text{global}} \in \mathbb{R}^{m \times n}$ is with orthonormal columns and $\boldsymbol{R}_{\text{global}} \in \mathbb{R}^{n \times n}$ is upper triangular.

To efficiently support high-throughput or batched data updates, we adopt a block-wise update strategy. Specifically, we partition $\boldsymbol{A}$ into k submatrices by rows:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \vdots \\ \boldsymbol{A}_k \end{bmatrix}, \quad \boldsymbol{A}_i \in \mathbb{R}^{m_i \times n}. \tag{1}$$

Each submatrix $\boldsymbol{A}_i$ undergoes an independent local QR decomposition:

$$\boldsymbol{A}_i = \boldsymbol{Q}_i \boldsymbol{R}_i, \quad \boldsymbol{R}_i \in \mathbb{R}^{n \times n}.$$

The resulting upper-triangular matrices $\{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_k\}$ are then vertically stacked and further decomposed:

$$\boldsymbol{R}_{\text{stack}} = \begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{R}_2 \\ \vdots \\ \boldsymbol{R}_k \end{bmatrix} = \boldsymbol{Q}_{\text{block}} \boldsymbol{R}_{\text{global}} \in \mathbb{R}^{kn \times n},$$

where $\boldsymbol{R}_{\text{global}} \in \mathbb{R}^{n \times n}$ is the final global upper-triangular matrix.

This hierarchical structure allows QR updates to proceed in two stages: local QR computation within each block, followed by a global reduction of the upper-triangular results. Structurally, it naturally fits distributed settings, where each user holds an incremental $\boldsymbol{R}_i$ and contributes to updating the global $\boldsymbol{R}_{\text{global}}$. Compared to row-wise updates, the block-wise strategy better supports rotation packing, reducing both the number of update rounds and communication overhead.

3.5 Fully Homomorphic Encryption (FHE)

Fully Homomorphic Encryption allows arbitrary arithmetic computations to be carried out on encrypted data. In an FHE scheme, a public/secret key pair (pk, sk) is generated. Given ciphertexts encrypted under the public key pk, it is possible to perform arithmetic operations (e.g., addition and multiplication) directly on the ciphertexts. Decryption using the secret key sk then reveals a result equivalent to performing those operations on the underlying plaintexts. Modern FHE schemes (e.g., BGV [6], BFV [15], CKKS) can support computations on vectors of numbers and are increasingly efficient. In our framework, since the entire computation process is based on floating-point operations, we use the CKKS homomorphic encryption scheme to ensure the security of data during the update calculation. For ease of description, we will list some of these basic operations.

- $\text{KeyGen}(1^\lambda)$: Generates a secret key sk, a public key pk, a relinearization key rlk, and a set of rotation keys $\{rk_i\}$ for required rotation indices i. Here, λ denotes the security parameter.
- $\text{Enc}_{pk}(m)$: Encrypts a plaintext vector $m \in \mathbb{C}^n$ under public key pk and returns a ciphertext c_m.
- $\text{Dec}_{sk}(c_m)$: Decrypts ciphertext c_m using the secret key sk, and outputs an approximate plaintext $m' \approx m$ due to the nature of CKKS encoding.
- $\text{EvalAdd}(c_{m_1}, c_{m_2})$: Performs homomorphic addition over two ciphertexts. It returns $c_{m_3} \approx \text{Enc}_{pk}(m_1 + m_2)$. We denote this as $c_{m_1} + c_{m_2}$.
- $\text{EvalMult}(c_{m_1}, c_{m_2})$: Performs homomorphic multiplication over two ciphertexts. It returns $c_{m_3} \approx \text{Enc}_{pk}(m_1 \cdot m_2)$, where $\cdot$ denotes component-wise (Hadamard) multiplication. This operation typically requires relinearization and rescaling to manage ciphertext level and scale.
- $\text{EvalRot}(c_m, i)$: Rotates the encrypted vector c_m by i slots. A positive i indicates a left rotation, and a negative i indicates a right rotation. This operation requires the corresponding rotation key rk_i.
- $\text{EvalRep}(c_m,\ i,\ j)$: A replication operator that copies the value at slot i to the next j consecutive slots within the same block. For example, if $\text{Dec}_{sk}(c_m) = [1,\ 0,\ 0,\ 0,\ 2,\ 0,\ 0,\ 0]$ and $(i,\ j) = (2,\ 2)$, then $\text{EvalRep}(c_m,\ 2,\ 2)$ yields a ciphertext decrypting to $[1,\ 1,\ 1,\ 0,\ 2,\ 2,\ 2,\ 0]$.

4 SEF-UQR System and Security Models

4.1 Framework Overview

We consider a distributed scenario with k participants, each holding a portion of a data matrix. Formally, participant i holds a matrix $\boldsymbol{A}_i \in \mathbb{R}^{m_i \times n}$, where m_i is the number of data samples at participant i and n is the number of features for each sample. The vertically concatenated matrix $\boldsymbol{A}^{(M \times n)}$ as in Eq. (1) is an $M \times n$ matrix (with $M = \sum_{i=1}^{k} m_i$).

Our goal is to continually update the QR factorization of the global matrix $\boldsymbol{A}$ in a privacy-preserving manner. Initially, participants obtain the joint data QR factorization result $\boldsymbol{R}_{\text{old}}^{\text{global}}$ through either federated QR or other distributed computations. As time progresses, each participant could receive new data that needs to be

incorporated into the model. Typically, new data means adding new rows (e.g., new samples) to $\boldsymbol{A}_i$. We assume a progressive scenario where new rows are appended to each $\boldsymbol{A}_i$ over time. We need to update the global QR factorization to reflect the addition of these new rows without any participant sharing their raw data. That is, the global QR is continually updated as new data comes without participants redoing the joint QR factorization.

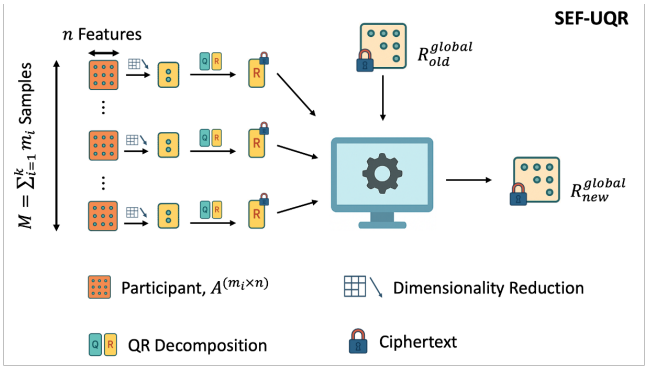


Figure 1: SEF-UQR Framework.

As illustrated in Figure 1, each participant first applies dimensionality reduction to the newly arrived data locally (if necessary), followed by a local QR decomposition to obtain an incremental matrix $\boldsymbol{R}_i$. The encrypted $\boldsymbol{R}_i$(by pk, which is generated by the trusted third party) is then transmitted to the server, which performs the SEF-UQR procedure to update the previously aggregated global matrix $\boldsymbol{R}_{\text{old}}^{\text{global}}$ and obtain a new global upper-triangular matrix $\boldsymbol{R}_{\text{new}}^{\text{global}}$. The output remains encrypted and can be directly used in subsequent computations within the system.

4.2 Security Models

For our system, we adopt the semi-honest model, which means that all participating parties need to follow the specified protocol but may attempt to infer other parties' data information and may also collude with other parties (excluding the key generator). We require that the input data of all participants and the intermediate computation results are secure, meaning that the security of all local data of the participants needs to be ensured, and no participant can infer the input data of other participants. We only require that the participants do not collude with the trusted key generator, so the security of the entire model can be guaranteed.

5 SEF-UQR Protocol Design

Our SEF-UQR protocol is designed to address two main challenges in privacy-preserving incremental QR updates over encrypted data: the high cost of homomorphic operations—especially ciphertext rotations and bootstrapping—and the difficulty of accurately computing non-linear functions such as square roots without incurring cumulative approximation errors. We adopt Givens rotations for their efficiency in modifying only two rows at a time, enabling incremental and parallel updates with minimal ciphertext manipulation compared to Householder reflections. To suit different application needs, SEF-UQR supports both row-wise and block-wise updates, with the latter exploiting SIMD-based rotation packing to process multiple updates within a single ciphertext. Finally, we integrate secure interactive protocols to compute critical non-linear functions exactly, ensuring numerical stability without exposing intermediate results.

5.1 Protocol Detail

The proposed SEF-UQR framework supports two update strategies, as detailed in Algorithm 1. The first is ***row-wise updating***, in which each participant applies a precomputed dimensionality reduction matrix to compress the newly arrived data vector. The reduced vector is then encrypted and transmitted to the server for integration into the global QR decomposition. The second strategy is ***block-wise updating***, where each participant accumulates multiple new data entries into a matrix $\boldsymbol{M}_{\text{new}}$. Similar to the row-wise method, dimensionality reduction is first applied to obtain the compressed matrix $\boldsymbol{M}'_{\text{new}}$. A local QR decomposition is then performed to extract the corresponding upper-triangular matrix $\boldsymbol{R}_{\text{new}}$. This matrix is encrypted using rotation packing and sent to the server, which incorporates it into the global encrypted QR structure.

Algorithm 1: SEF-UQR

Input: Encrypted global upper-triangular matrix $c_{\boldsymbol{R}_{\text{old}}^{\text{global}}}$; Dimensionality reduction matrix $\boldsymbol{Q}^{\text{global}}$; New row data $\boldsymbol{w}_k$ or matrix $\Delta\boldsymbol{M}_{\text{new}}^k$ from the k^{th} participant

Output: Updated encrypted matrix $c_{\boldsymbol{R}_{\text{new}}^{\text{global}}}$

foreach *new data* $\boldsymbol{w}_{new}$ *from the* k^{th} *participant* **do**
  $\tilde{\boldsymbol{w}}_{\text{new}} \leftarrow \boldsymbol{Q}^{\text{global}} \cdot \boldsymbol{w}_{\text{new}}$;
  Encrypt $\tilde{\boldsymbol{w}}_{\text{new}}$ using pk to obtain ciphertext $c_{\tilde{\boldsymbol{w}}_{\text{new}}}$;
  Send $c_{\tilde{\boldsymbol{w}}_{\text{new}}}$ to the server;
  $c_{\boldsymbol{R}_{\text{new}}^{\text{global}}} \leftarrow \text{GivensSingle}(c_{\tilde{\boldsymbol{w}}_{\text{new}}}, c_{\boldsymbol{R}_{\text{old}}^{\text{global}}})$; // Alg. 2
end
foreach *new data* $\Delta\boldsymbol{M}_{new}^k$ *from the* k^{th} *participant* **do**
  $\Delta\tilde{\boldsymbol{M}}_{\text{new}}^k \leftarrow \boldsymbol{Q}^{\text{global}} \cdot \Delta\boldsymbol{M}_{\text{new}}^k$;
  $\Delta\tilde{\boldsymbol{M}}_{\text{RP}}^k \leftarrow \text{RP}(\Delta\tilde{\boldsymbol{M}}_{\text{new}}^k)$; // Alg. 4
  Encrypt $\Delta\tilde{\boldsymbol{M}}_{\text{RP}}^k$ using pk to obtain ciphertext $c_{\Delta\tilde{\boldsymbol{M}}_{\text{RP}}^k}$;
  Send $c_{\Delta\tilde{\boldsymbol{M}}_{\text{RP}}^k}$ to the server;
  $c_{\boldsymbol{R}_{\text{new}}^{\text{global}}} \leftarrow \text{GivensBlock}(c_{\Delta\tilde{\boldsymbol{M}}_{\text{RP}}^k}, c_{\boldsymbol{R}_{\text{old}}^{\text{global}}})$; // Alg. 3
end

5.2 Execution Protocols of SEF-UQR

To enable efficient and privacy-preserving QR updates over encrypted data, our SEF-UQR system introduces several computation protocols. These protocols are organized into three categories: the core algorithmic framework for ciphertext QR updates, optimization strategies for performance and accuracy, and auxiliary operations that support matrix manipulation in encrypted space.

5.2.1 Core Algorithm: Ciphertext-Based Givens Rotation. The core of SEF-UQR lies in homomorphically constructing Givens rotations for encrypted matrix updates. As shown in Algorithm 2, we implement row-wise Givens updates over ciphertexts by inserting an encrypted vector $c_{\tilde{w}}$ into an existing encrypted upper-triangular matrix $c_{R_{\text{old}}^{global}}$ to form c_H. Then, by constructing the Givens rotation matrix and applying it to c_H, we finally obtain $c_{R_{\text{new}}^{global}}$.

Algorithm 2: Givens-Single

Input: Encrypted matrix $c_{R_{\text{old}}}^{global} \in \mathbb{R}^{n\times n}$;
Encrypted new row $c_{\tilde{w}} \in \mathbb{R}^{1\times n}$
Output: Updated ciphertext $c_{R_{\text{new}}^{global}} \in \mathbb{R}^{n\times n}$

$c_H \in \mathbb{R}^{(n+1)\times n} \leftarrow \texttt{InsVec}(c_{\tilde{w}}, c_{R_{\text{old}}^{global}})$; // Alg. 6
for $i \leftarrow 0$ **to** $n-1$ **do**
  $c_p \leftarrow \text{EvalRot}(c_H[i,:],\ i)$;
  $c_q \leftarrow \text{EvalRot}(c_H[i+1,:],\ i)$;
  $c_h \leftarrow (c_p \cdot c_p) + (c_q \cdot c_q)$;
  $(c_h^+,\ c_h^-) \leftarrow \texttt{InvSqrt}(c_h)$; // //Alg. 5
  $u \leftarrow [1, 0, \ldots, 0]$;
  $c_\gamma \leftarrow \text{EvalRep}(c_p \cdot u,\ n,\ 1) \cdot c_h^+$;
  $c_\sigma^+ \leftarrow \text{EvalRep}(c_q \cdot u,\ n,\ 1) \cdot c_h^+$;
  $c_\sigma^- \leftarrow \text{EvalRep}(c_q \cdot u,\ n,\ 1) \cdot c_h^-$;
  $c_H[i,:] \leftarrow \text{EvalRot}(c_\gamma \cdot c_p + c_\sigma^+ \cdot c_q,\ -i)$;
  $c_H[i+1,:] \leftarrow \text{EvalRot}(c_\sigma^- \cdot c_p + c_\gamma \cdot c_q,\ -i)$;
end
$c_{R_{\text{new}}^{global}} \leftarrow c_H[0{:}n,:]$;

For scenarios where parallel computation or communication efficiency is critical, we adopt a block-wise update strategy that processes multiple new data rows simultaneously. Algorithm 3 demonstrates the batch update process using Givens rotations, while Algorithm 4 details how rotation-based packing is used to encode the global matrix and the update block into two ciphertext vectors, thereby reducing communication overhead. A visual illustration of the packing process is provided in Appendix A.1, and the parallelization strategy for block-wise updates is further discussed in Appendix A.2.

5.2.2 Optimization Protocols: Rotation Packing and Secure Inverse Square Root. Block-based Givens updates require structured packing of matrix rows. To support this, Algorithm 4 details the rotation packing encoding procedure, where each row $M[i,:]$ of a plaintext matrix M is left-rotated and concatenated into a single vector $v^{(1\times n^2)}$, ready to be encrypted into one ciphertext.

Another optimization targets the accurate computation of inverse square roots, which are critical for normalizing Givens rotation coefficients. As CKKS does not support non-linear functions such as $\sqrt{\cdot}$ natively, existing works—such as [16]—typically employ polynomial approximations using Chebyshev expansions. However, this approach suffers from limited numerical precision. For instance, the results in [16] show that as the number of features increases, the final approximation error grows significantly, often exceeding 10^{-2} on benchmark datasets — a level that is unsuitable for high-precision applications.

Algorithm 3: Givens-Block

Input: Ciphertexts $c_{v_A}, c_{v_B} \in \mathbb{R}^{1\times d}$; feature count n
Output: Final ciphertext $c_{v_A}^{\text{final}}$

$c_a, c_b \leftarrow c_{v_A}, c_{v_B}$;
for $i \leftarrow 0$ **to** $n-1$ **do**
  $c_h \leftarrow (c_a \cdot c_a) + (c_b \cdot c_b)$;
  $(c_h^+, c_h^-) \leftarrow \texttt{InvSqrt}(c_h)$;
  $u[i] \leftarrow 1$ if $i \bmod n = 0$, else 0;
  $c_\gamma \leftarrow \text{EvalRep}(c_a \cdot u,\ n,\ n) \cdot c_h^+$;
  $c_{\sigma^+} \leftarrow \text{EvalRep}(c_b \cdot u,\ n,\ n) \cdot c_h^+$;
  $c_{\sigma^-} \leftarrow \text{EvalRep}(c_b \cdot u,\ n,\ n) \cdot c_h^-$;
  $c_a \leftarrow c_\gamma \cdot c_a + c_{\sigma^+} \cdot c_b$;
  $c_b \leftarrow c_{\sigma^-} \cdot c_a + c_\gamma \cdot c_b$;
  $c_b \leftarrow \text{EvalRot}(c_b, -(n-1))$;
end
$c_{v_A}^{\text{final}} \leftarrow c_a$

Algorithm 4: Rotation Packing (RP)

Input: Matrix $M^{(n\times n)}$
Output: Vector $v^{(1\times n^2)}$

for $i \leftarrow 1$ **to** n **do**
  $v_i^{(1\times n)} \leftarrow \text{Rot}(M[i,:],\ i-1)$;
end
$v^{(1\times n^2)} \leftarrow [v_1, v_2, \ldots, v_n]$;

Algorithm 5: Secure Inverse Square Root (InvSqrt)

Input: Ciphertext c_v
Output: Ciphertexts c_v^+, c_v^-

Sample random vector $r \leftarrow \text{Rand}([2^{-\lambda}, 2^\lambda])$;
// λ is the security parameter
$c_{\text{mask}} \leftarrow \text{EvalMult}(c_v,\ r \cdot r)$;
$v_{\text{mask}} \leftarrow \text{Dec}_{sk}(c_{\text{mask}})$;
$v^+ \leftarrow 1/\sqrt{v_{\text{mask}}}$; $v^- \leftarrow -1/\sqrt{v_{\text{mask}}}$;
$c_v^+ \leftarrow \text{Enc}_{pk}(v^+) \cdot r$; $c_v^- \leftarrow \text{Enc}_{pk}(v^-) \cdot r$;

To address this issue, inspired by the protocol in [4, 32], we design a secure interactive computation strategy as described in Algorithm 5. The input ciphertext is masked with a random vector, decrypted by the data owner, and processed in plaintext to compute the exact inverse square root. The result is then re-encrypted while preserving the original mask. This enables precise evaluation of $\pm 1/\sqrt{x}$ under encryption, significantly improving numerical accuracy and avoiding approximation-induced errors.

5.2.3 Auxiliary Operations: Encrypted Matrix Manipulation. Certain utility procedures are required to facilitate ciphertext QR updates. Algorithm 6 defines a row insertion operation InsVec, which

Algorithm 6: InsertVector (InsVec)

Input: Encrypted matrix $c_{\boldsymbol{M}}, \boldsymbol{M} \in \mathbb{R}^{m \times n}$;
Encrypted vector $c_{\boldsymbol{v}}, \boldsymbol{v} \in \mathbb{R}^{1 \times n}$;
Insertion index p
Output: New matrix $c_{\boldsymbol{M}'} \in \mathbb{R}^{(r+1) \times n}$

Initialize $c_{\boldsymbol{M}'} \leftarrow [\,]$;
for $i \leftarrow 0$ **to** $r-1$ **do**
  if $i = p$ **then**
    $c_{\boldsymbol{M}'}[i,:] \leftarrow c_{\boldsymbol{v}}$;
  end
  $c_{\boldsymbol{M}'}[i,:] \leftarrow c_{\boldsymbol{M}}[i,:]$;
end

injects a new ciphertext row $c_{\boldsymbol{v}}$ into an encrypted matrix $c_{\boldsymbol{M}}$ at index p. This step is necessary to build the extended matrix $c_{\boldsymbol{H}}$ used in Givens rotations.

6 Security Analysis

We analyze the security of our SEF-UQR protocol from both the server and participant perspectives, considering the use of fully homomorphic encryption (FHE) and secure interaction protocols throughout the computation.

6.1 Server-Side Privacy

The server receives only encrypted data from participants. All sensitive values, including each participant's update $c_{\boldsymbol{R}_i^{\text{new}}}$ and the global matrix $c_{\boldsymbol{R}_{\text{old}}^{\text{global}}}$, are encrypted under a semantically secure FHE scheme. This implies that, under standard cryptographic assumptions (e.g., the hardness of the Ring-LWE problem [6]), each ciphertext is computationally indistinguishable from random. In particular, given $c_{\boldsymbol{R}_i^{\text{new}}}$, the server cannot infer any information about the underlying matrix values or statistical patterns beyond their dimensionality.

During the protocol, the server only performs homomorphic manipulations on ciphertexts. Although it knows the structure of the operations (e.g., which rows are being merged or rotated), it does not learn the plaintext operands or intermediate results. All intermediate ciphertexts remain encrypted and indistinguishable from random noise. Even the final output $c_{\boldsymbol{R}_{\text{new}}^{\text{global}}}$ is never decrypted by the server. Hence, at no point in the protocol does the server observe or infer any plaintext data from participants.

This provides a significantly stronger privacy guarantee compared to traditional federated learning systems, where gradient updates have been shown to leak sensitive information under model inversion and reconstruction attacks [17, 33].

6.2 Participant Privacy

From the perspective of each participant, their only interaction is with encrypted messages and optional decryption tasks during secure interaction protocols (e.g., inverse square root evaluation). Although a participant may decrypt intermediate masked ciphertexts, these are perturbed by the server using large random masks. Based on the "Smudging Lemma" [4], a sufficiently large perturbation can fully obscure the original value, preventing meaningful leakage.

Moreover, even if the final encrypted matrix $\boldsymbol{R}_{\text{new}}^{\text{global}}$ is decrypted and shared among participants in certain deployments, it still reveals negligible information about any individual party. This is because $\boldsymbol{R}_{\text{new}}^{\text{global}}$ results from a sequence of orthogonal transformations (e.g., Givens rotations) applied to the collective updates from all parties, effectively mixing their contributions. Without auxiliary knowledge, no participant can algebraically isolate another's update from the aggregated result.

Overall, our protocol guarantees that no honest-but-curious party (whether the server or a participant) can learn the private data of others. The semantic security of the FHE scheme ensures confidentiality against passive adversaries, assuming the encryption scheme remains unbroken. Additionally, by carefully integrating secure interaction protocols (e.g., for division and square root operations), we maintain this security even for non-linear operations that require partial decryption and re-encryption. Compared to other privacy-preserving techniques such as differential privacy or secure aggregation, our combination of FHE and interactive masking provides significantly stronger end-to-end privacy assurances.

7 Complexity Analysis

In this section, we theoretically analyze the computational efficiency and communication overhead of the proposed SEF-UQR protocol. To the best of our knowledge, there currently exists no privacy-preserving matrix factorization method that supports incremental updates. Therefore, we use conventional QR decomposition as a baseline and compare the performance of two update strategies within SEF-UQR: the row-wise update and the block-wise update.

Computation Complexity. As discussed in Section 3.3, the time complexity of a full QR decomposition is $O(n^3)$, while using Givens rotations for incremental updates reduces this to approximately $O(n^2)$ per row. In the SEF-UQR framework, the dominant computational costs arise from three sources: the number of homomorphic multiplications, ciphertext rotations, and bootstrapping.

Table 1: Operation Counts for SEF-UQR

Operation	Row-wise (m rows)	Block-wise (m rows)
EvalAdd	$mn(2n+1)$	$3n$
EvalMult	$16mn$	$14n$
EvalRot	$4mn(n-1)$	$(n^2+n)(n-1)$
Bootstrapping	mn	n

Table 1 summarizes the computation cost of two methods for updating m entries of history global R with the same addition parameters (e.g., ring dimension, multiplication depth). For the row-wise method, each operation will be affected by the number of updated entries, and the number of ciphertext rotations and bootstrapping will increase significantly as the number of updated entries m increases, which leads to a decrease in computational efficiency. In contrast, the block method benefits from rotation packing. Each participant performs plaintext QR decomposition on the matrix blocks

locally and only encrypts the resulting upper triangular matrix. This makes the total number of ciphertext operations only related to n, which has a natural advantage for datasets after dimension reduction.

Communication Complexity. Similarly, we evaluate the communication cost of the two update strategies. Assuming fixed cryptographic parameters, the size of each ciphertext remains unchanged. As shown in Table 2, both methods require a total of 4 ciphertexts to be transmitted due to the need to perform one InvSqrt. However, updating m rows of data by the row-wise method requires mn executions of InvSqrt, while the block-wise method only needs n executions using rotation packing. Therefore, similar to computational overhead, the communication overhead of the Row-wise method will be affected by the number of ciphertext rows m, and when m is larger, the communication overhead is larger compared to the Block-wise method.

Table 2: Communication Cost for SEF-UQR

Metric	Row-wise (m rows)	Block-wise (m rows)
Rounds	mn	n
Ciphertexts	3	3
Total	$3mn + 1$	$3n + 1$

While the row-wise update offers better responsiveness and finer-grained control, it incurs significantly higher computational and communication costs. In practice, the block-wise method is more efficient for batch updates and high-throughput applications. Additionally, in both strategies, ciphertext refreshing can optionally be done via secure interactive re-encryption to avoid expensive bootstrapping operations.

8 Experimental Evaluation

8.1 Experimental Setup

SEF-UQR is implemented in Go [27] and leverages the open-source Lattigo library [1] for fully homomorphic encryption. We conducted experiments on a personal computer equipped with an Intel i9-12900KF CPU @ 3.4GHz and 64GB of RAM. We adopt the CKKS scheme, which supports approximate arithmetic over real numbers, using the parameters summarized in Table 3.

Table 3: SEF-UQR System Parameter Summary

Definition	**Parameter**	**Value(default)**
Ring Dimension(Encryption)	N_{Enc}	$\geq 2^{14}$
Ring Dimension(Bootstrapping)	N_{BTS}	$\geq 2^{15}$
Scale	Δ	2^{45}
Multiplicative Depth	d	7
Number of Updating Samples	m	-
Number of Features	n	-

We divide the experiments into two parts: (1) Synthetic datasets, primarily used to analyze the computational overhead of each component within the SEF-UQR system as well as its communication cost. These datasets are generated as random dense matrices with elements uniformly sampled from [-1,1], normalized per column to unit ℓ_2 norm. and (2) Real datasets, mainly employed to benchmark SEF-UQR against other distributed or centralized QR decomposition methods in the updating scenario, thereby demonstrating its practical applicability. The real datasets include Wine [12], Pima [29], Parkinsons [21], and UR3 CobotOps [28]. Note that participants operate independently, thus the system's computational overhead depends solely on update dimensions—the length of each data entry and total updates—not on the number of participants. Consequently, the explicit number of participants is not specified in our experimental setup.

8.2 Performance of SEF-UQR

We use the synthetic datasets to test the run time of each module in SEF-UQR – homomorphic operations (addition and multiplication), ciphertext rotations, and bootstrapping. It is worth noting that we do not scale up the number of samples or features excessively in the generation of synthetic datasets. This is because in SEF-UQR, each participant needs to perform a dimension reduction step before applying any updates, so we directly simulate the reduced datasets.

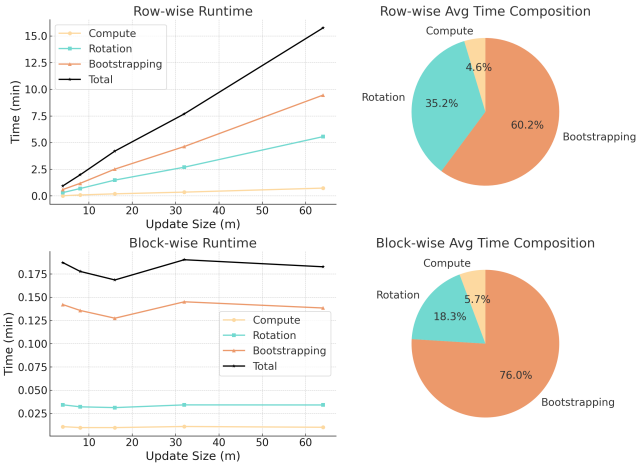


Figure 2: Runtime comparison of different modules in Row-wise and Block-wise methods.(n=4)

In the case of $n = 4$, we evaluate the runtime performance of the two update strategies under increasing numbers of updates, as illustrated in Figure 2. As expected, the row-wise update method exhibits a linear growth in computation time with respect to the number of updates. This observation aligns well with the theoretical analysis, where each additional row introduces a proportional increase in the overall computation workload. In contrast, the block-wise update method demonstrates stable computation time across all update quantities. This is because, as theoretically analyzed, its runtime primarily depends on the fixed parameter n and is largely unaffected by the number of updates. Furthermore, the figure provides insights into the internal time composition of each method. In both strategies, the ciphertext computation phase (i.e., homomorphic arithmetic) contributes around 5% to the total runtime. The ciphertext rotation phase contributes slightly more, but the

majority of the time is consistently consumed by the bootstrapping operation, which accounts for over 50% of the total runtime in both update approaches.

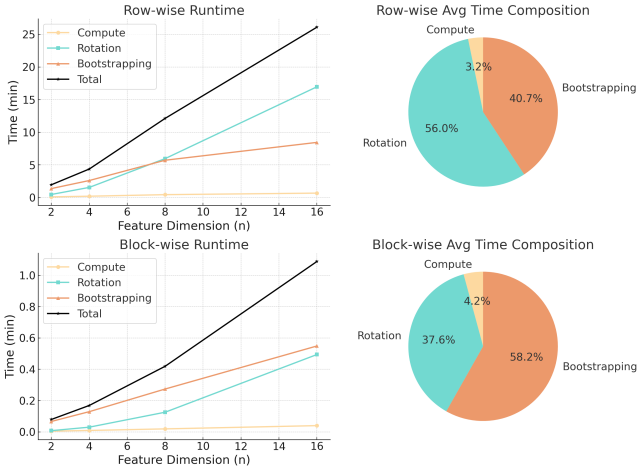


Figure 3: Runtime comparison of Row-wise and Block-wise methods under varying feature dimensions.

To examine the impact of feature dimensionality on computational performance, we fix the number of updates to 20 and vary the number of features per update. The results are presented in Figure 3. As shown in the top-left subplot, the total runtime of the *Row-wise* update method increases significantly with the feature dimension. This trend aligns with the theoretical complexity analysis, as higher dimensionality leads to larger matrices involved in rotation and bootstrapping operations. Among the three computational components, ciphertext rotation and bootstrapping dominate the total cost, while ciphertext computation consistently accounts for less than 5% of the overall time. The accompanying pie chart further confirms that **rotation and bootstrapping together contribute to over 95% of the total runtime** in the Row-wise case. Compared to the results in Figure 2, we also observe a substantial increase in the cost of rotation as feature dimension grows. In contrast, the *Block-wise* method (bottom-left subplot) demonstrates significantly lower total runtime. The runtime increases slowly with the number of features, highlighting the superior scalability of this approach. As illustrated in the corresponding pie chart, rotation and bootstrapping remain the dominant contributors; however, the overall time consumption is substantially reduced compared to the Row-wise method. Overall, the Block-wise update strategy offers better efficiency and scalability for high-dimensional encrypted updates under homomorphic encryption constraints.

Under our default system configuration, each ciphertext is approximately 2MB in size. During the execution of SEF-UQR, regardless of whether the row-wise or block-wise strategy is adopted, each communication round involves one invocation of the `InvSqrt` protocol. In this protocol, the server returns one ciphertext, and the client uploads two ciphertexts, resulting in a total communication cost of roughly 6MB per round. For a task consisting of 30 updates with 20 features each, the total communication overhead differs significantly between the two update strategies. The row-wise approach requires 20 rounds (per row), resulting in a total communication volume of approximately 3.5GB. In contrast, the block-wise strategy only requires overhead of approximately 0.36GB.

8.3 Comparison with Existing Work

To further highlight the advantages of SEF-UQR in matrix updates, we compare it with two representative homomorphic encryption (HE)-based matrix decomposition frameworks proposed in recent years: SF-PCA [16] and H-PCA [25].

SF-PCA represents one of the most recent and advanced homomorphic encryption-based solutions for privacy-preserving distributed matrix factorization. As a practical system designed for federated PCA, it leverages multi-party homomorphic encryption, interactive protocols, and edge computing to securely aggregate encrypted data across multiple parties. Notably, SF-PCA employs QR decomposition as a core subroutine to perform dimensionality reduction over encrypted inputs, making it highly relevant for comparison with SEF-UQR.

In contrast, H-PCA adopts a centralized design. Each user encrypts and packages their local data before transmitting it to the server. The server then performs computation directly on encrypted data without requiring expensive operations such as matrix inversion or full matrix multiplication. However, both SF-PCA and H-PCA share a common limitation: when new data arrive, they must re-run the full matrix decomposition to reflect the changes. This makes them inherently inefficient for update-intensive scenarios.

Table 4 summarizes the performance of all three methods on four real-world datasets under a fixed workload of 20 updates. Both SF-PCA and H-PCA incur high runtime overhead, as each update requires full recomputation—making them inefficient in high-frequency or large-scale settings. SEF-UQR, by contrast, shows clear computational advantages, particularly in block-wise updates. However, its performance on high-dimensional data is still constrained by bootstrapping, which accounts for over 50% of the total runtime. To address this, we discuss optimization strategies in Section 8.4 for reducing bootstrapping overhead, which we estimate could cut computation time by half or more.

The reduced accuracy in baseline methods stems from two main sources: (1) approximation errors from polynomial-based evaluation of non-linear functions (e.g., square roots), and (2) error accumulation across PCA's iterative structure. These issues intensify as the feature dimension increases.

In contrast, SEF-UQR employs a secure inverse square root protocol (`InvSqrt`) that computes square roots accurately in a masked, partially decrypted domain. Combined with its deterministic update scheme based on exact Givens rotations, SEF-UQR achieves stable and dimension-independent accuracy, making it well-suited for long-term, privacy-preserving learning.

8.4 Discussion and Extension

While previous results demonstrate that SEF-UQR significantly improves update efficiency over traditional matrix decompositions, there remains room for further optimization. As shown by the pie charts in Figure 2 and Figure 3, ciphertext bootstrapping often dominates runtime, exceeding 50% in many cases. To alleviate

Table 4: Comparison of SEF-UQR with SF-PCA and H-PCA on Real-World Datasets (20 Updates)

Dataset	Dim.	SF-PCA		H-PCA		SEF-UQR		
		MSE	Time (h)	MSE	Time (h)	MSE	Row-wise Time (h)	Block-wise Time (h)
wine [12]	4898×11	10^{-6}	16.0	/	3.72	10^{-13}	0.27	0.01
Pima [29]	767×8	10^{-11}	0.8	/	2.7	10^{-14}	0.20	0.007
Parkinsons [21]	5875×16	10^{-4}	18.0	/	14.4	10^{-13}	0.42	0.018
UR3 CobotOps [28]	7358×20	/	24.0	/	11.2	10^{-12}	0.54	0.024

Note: '/' indicates the approximation error exceeds 10^{-2} on the corresponding dataset.

this bottleneck, we explore an alternative inspired by the `InvSqrt` protocol: refreshing ciphertexts via secure interactive computation instead of bootstrapping.

Under our parameter setting, bootstrapping a single ciphertext takes roughly 2 seconds, whereas secure refreshing requires only 0.03 seconds (excluding communication). This yields substantial computational savings, but significantly increases communication: for instance, Row-wise updates require exchanging $7mn$ ciphertexts instead of $3mn + 1$. As such, secure refreshing may be unsuitable for low-bandwidth or high-latency settings.

Finally, while SEF-UQR excels in small-batch updates, its efficiency diminishes when a large number of new rows are introduced at once. In such cases, the cumulative overhead from encrypted Givens rotations and bootstrapping can surpass that of direct recomputation. Empirically, when the update size becomes much larger than the original global matrix size, recomputing from scratch can be more efficient. This suggests hybrid strategies that alternate between incremental updates and full recomputation may offer better overall performance. Formalizing this switching threshold remains a promising direction for future work.

9 Conclusion

We proposed SEF-UQR, a framework for efficient and privacy-preserving distributed QR updates. By combining homomorphic encryption, secure interaction protocols, and structured linear algebra, SEF-UQR enables collaborative model maintenance without exposing private data. Rotation packing and incremental updates ensure high efficiency and scalability, while our secure inverse square root protocol (`InvSqrt`) achieves accurate non-linear evaluation. Experiments on real-world datasets demonstrate stable, precise results with significant runtime savings.

SEF-UQR provides a practical solution for secure incremental model updates across distributed data sources, making it suitable for privacy-sensitive applications in domains such as healthcare, finance, and personalized services.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (Grant No. 2020YFA0712303), the Natural Science Foundation of Chongqing (Grant No. CSTB2023YSZX-JCX0008), and the National Science Foundation for Young Scientists of China (Grant No. 12301650).

A Optimization Techniques Details

In this appendix, we briefly explain how *parallel computation* and *rotation packing* are executed in Givens rotations. Using two 3×3 upper-triangular matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as an example:

$$\boldsymbol{A}^{(3\times3)} = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}, \quad \boldsymbol{B}^{(3\times3)} = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}.$$

A.1 Rotation Packing

With *rotation packing*, ciphertexts are organized to minimize rotation offsets across steps. Packed vectors take the form:

$$\begin{bmatrix} * & * & * \mid * & * & 0 \mid * & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} * & * & * \mid * & * & 0 \mid * & 0 & 0 \end{bmatrix}$$

After one Givens step:

$$\begin{bmatrix} * & * & * \mid * & * & 0 \mid * & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & * & * \mid 0 & * & 0 \mid 0 & 0 & 0 \end{bmatrix}$$

To realign for the next rotation, only 2 shifts are required:

$$\begin{bmatrix} * & * & * \mid * & * & 0 \mid * & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \mid * & * & 0 \mid * & 0 & 0 \end{bmatrix}$$

This optimization reduces the total number of required ciphertext rotations to $n(n-1)$.

A.2 Parallel Computation

To enable parallel execution, we stack multiple encrypted matrices vertically, interleaving rows to form a 6×3 structure. This layout allows row pairs (e.g., rows 0–1, 2–3, 4–5) to be processed independently using Givens rotations in parallel. The required number of rotations is only $n(n-1)/2$ in total:

$$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & 0 \\ * & * & 0 \\ * & 0 & 0 \\ * & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} * & * & * \\ 0 & * & * \\ * & * & 0 \\ 0 & * & 0 \\ * & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{Rot(2,4)} \begin{bmatrix} * & * & * \\ * & * & 0 \\ * & * & 0 \\ * & 0 & 0 \\ * & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{\cdots}$$

Generative AI Tools Disclosure

During the preparation of this work, ChatGPT was used for language polishing and grammar correction of the manuscript. GitHub Copilot was employed to assist in optimizing portions of the code, such as improving syntax and suggesting minor refactorings. All core research activities—including theoretical formulation, algorithm design, experimental implementation, and result analysis—were solely conducted by the authors without the use of generative AI tools.

References

[1] S. Agahi, M. Coulon, J. Dorier, C. Mouchet, D. Rotondo, N. Gama, M. Orfila, and M. Vanel. 2023. Lattigo: A Go Library for Lattice-Based Homomorphic Encryption. https://github.com/tuneinsight/lattigo Version 3.0, Accessed: March 2024.

[2] Charu C. Aggarwal and Philip S. Yu. 2008. A General Survey of Privacy-Preserving Data Mining Models and Algorithms. In *Privacy-Preserving Data Mining: Models and Algorithms*, Charu C. Aggarwal and Philip S. Yu (Eds.). Springer US, Boston, MA, 11–52. doi:10.1007/978-0-387-70992-5_2

[3] Omar Ahsan. 2011. QR Decomposition in a Multicore Environment. (2011).

[4] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *Advances in Cryptology – EUROCRYPT 2012*, David Pointcheval and Thomas Johansson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 483–501.

[5] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1175–1191. doi:10.1145/3133956.3133982

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 309–325. doi:10.1145/2090236.2090262

[7] Matthew Brand. 2006. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra Appl.* 415, 1 (May 2006), 20–30. doi:10.1016/j.laa.2005.07.021

[8] James R. Bunch and Christopher P. Nielsen. 1978. Updating the Singular Value Decomposition. *Numer. Math.* 31, 2 (1978), 111–129.

[9] Jung Hee Cheon, Hyeongmin Choe, Saebyul Jung, Duhyeong Kim, Dah Hoon Lee, and Jai Hyun Park. 2023. Arithmetic PCA for Encrypted Data. Cryptology ePrint Archive, Paper 2023/1544. https://eprint.iacr.org/2023/1544

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer International Publishing, Cham, 409–437.

[11] Paul G. Constantine and David F. Gleich. 2011. Tall and skinny QR factorizations in MapReduce architectures. In *Proceedings of the Second International Workshop on MapReduce and Its Applications* (San Jose, California, USA) *(MapReduce '11)*. Association for Computing Machinery, New York, NY, USA, 43–50. doi:10.1145/1996092.1996103

[12] Paulo Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. 2009. Wine Quality. Online dataset, UCI Machine Learning Repository. doi:10.24432/C56S3T Accessed: March 2022.

[13] Jack Dongarra, Mathieu Faverge, Thomas Hérault, Mathias Jacquelin, Julien Langou, and Yves Robert. 2013. Hierarchical QR factorization algorithms for multi-core clusters. *Parallel Comput.* 39, 4 (2013), 212–232. doi:10.1016/j.parco.2013.01.003

[14] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 265–284.

[15] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144

[16] David Froelicher, Hyunghoon Cho, Manaswitha Edupalli, Joao Sa Sousa, Jean-Philippe Bossuat, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, Bonnie Berger, and Jean-Pierre Hubaux. 2023. Scalable and Privacy-Preserving Federated Principal Component Analysis. In *2023 IEEE Symposium on Security and Privacy (SP)*. 1908–1925. doi:10.1109/SP46215.2023.10179350 ISSN: 2375-1207.

[17] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting Gradients – How easy is it to break privacy in federated learning? arXiv:2003.14053 [cs.CV] https://arxiv.org/abs/2003.14053

[18] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (Bethesda, MD, USA) *(STOC '09)*. Association for Computing Machinery, New York, NY, USA, 169–178. doi:10.1145/1536414.1536440

[19] Wallace Givens. 1958. Computation of plane unitary rotations transforming a general matrix to triangular form. *J. Soc. Indust. Appl. Math.* 6, 1 (1958), 26–50.

[20] Gene H. Golub and Charles F. Van Loan. 2013. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA. doi:10.1137/1.9781421407944 _eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781421407944.

[21] Max Little. 2007. Parkinsons. Online dataset, UCI Machine Learning Repository. doi:10.24432/C59C74 Accessed: March 2022.

[22] Bowen Liu and Qiang Tang. 2020. Privacy-Preserving Decentralised Singular Value Decomposition. In *Information and Communications Security*, Jianying Zhou, Xiapu Luo, Qingni Shen, and Zhen Xu (Eds.). Springer International Publishing, Cham, 703–721.

[23] Xirong Ma, Chuan Ma, Yali Jiang, and Chunpeng Ge. 2024. Improved privacy-preserving PCA using optimized homomorphic matrix multiplication. *Comput. Secur.* 138, C (March 2024), 19 pages. doi:10.1016/j.cose.2023.103658

[24] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. 2013. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 801–812. doi:10.1145/2508859.2516751

[25] Samanvaya Panda. 2021. Principal Component Analysis Using CKKS Homomorphic Scheme. In *Cyber Security Cryptography and Machine Learning*, Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann (Eds.). Springer International Publishing, Cham, 52–70.

[26] Mostafa I. Soliman. 2011. Efficient implementation of QR decomposition on intel multi-core processors. In *2011 Seventh International Computer Engineering Conference (ICENCO'2011)*. 25–30. doi:10.1109/ICENCO.2011.6153928

[27] The Go Team. 2024. The Go Programming Language. https://golang.org Accessed: March 2024.

[28] Marios Tyrovolas, Khurshid Aliev, Dario Antonelli, and Chrysostomos Stylios. 2024. UR3 CobotOps. Online dataset, UCI Machine Learning Repository. doi:10.24432/C5J891 Accessed: March 2024.

[29] UCI Machine Learning Repository. 2022. Pima Indians Diabetes Dataset. https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database Accessed: March 2022.

[30] Dong Wang, Ming cheng Ma, Lingli Liu, Xiongxiong Du, Xiaoruo Li, and Bingnan Zhu. 2024. Principal Component Analysis Scheme Based on Homomorphic Encryption in a Distributed Environment, Cheqing Jin, Shiyu Yang, Xuequn Shang, Haofen Wang, and Yong Zhang (Eds.). Springer Nature Singapore, Singapore, 415–427.

[31] Jieping Ye, Qi Li, Hui Xiong, H. Park, R. Janardan, and V. Kumar. 2005. IDR/QR: an incremental dimension reduction algorithm via QR decomposition. *IEEE Transactions on Knowledge and Data Engineering* 17, 9 (Sept. 2005), 1208–1222. doi:10.1109/TKDE.2005.148

[32] Haonan Yuan, Wenyuan Wu, and Jingwei Chen. 2024. Fast, Large Scale Dimensionality Reduction Schemes Based on CKKS. Cryptology ePrint Archive, Paper 2024/849. https://eprint.iacr.org/2024/849

[33] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. *Deep leakage from gradients*. Curran Associates Inc., Red Hook, NY, USA.