Practical Lossless Federated Singular Vector Decomposition over Billion-Scale Data

Di Chai
dchai@cse.ust.hk
Hong Kong University of Science and Technology
Clustar Co., Ltd

Leye Wang
leyewang@pku.edu.cn
MOE Key Lab of High Confidence Software Technologies, Peking University

Junxue Zhang
jzhangcs@cse.ust.hk
Hong Kong University of Science and Technology
Clustar Co., Ltd

Liu Yang
lyangau@cse.ust.hk
Hong Kong University of Science and Technology
Clustar Co., Ltd

Shuowei Cai
scaiak@cse.ust.hk
Hong Kong University of Science and Technology
Clustar Co., Ltd

Kai Chen
kaichen@cse.ust.hk
Hong Kong University of Science and Technology

Qiang Yang
qyang@cse.ust.hk
Hong Kong University of Science and Technology
AI Group, WeBank Co., Ltd

Abstract

With the enactment of privacy-preserving regulations, e.g., GDPR, federated SVD is proposed to enable SVD-based applications over different data sources without revealing the original data. However, many SVD-based applications cannot be well supported by existing federated SVD solutions. The crux is that these solutions, adopting either differential privacy (DP) or homomorphic encryption (HE), suffer from accuracy loss caused by unremovable noise or degraded efficiency due to inflated data.

In this paper, we propose FedSVD, a practical lossless federated SVD method over billion-scale data, which can simultaneously achieve lossless accuracy and high efficiency. At the heart of FedSVD is a lossless matrix masking scheme delicately designed for SVD: 1) While adopting the masks to protect private data, FedSVD completely removes them from the final results of SVD to achieve lossless accuracy; and 2) As the masks do not inflate the data, FedSVD avoids extra computation and communication overhead during the factorization to maintain high efficiency. Experiments with real-world datasets show that FedSVD is over 10000× faster than the HE-based method and has 10 orders of magnitude smaller error than the DP-based solution ($\epsilon = 0.1, \delta = 0.1$) on SVD tasks. We further build and evaluate FedSVD over three real-world applications: principal components analysis (PCA), linear regression (LR), and latent semantic analysis (LSA), to show its superior performance in practice. On federated LR tasks, compared with two state-of-the-art solutions: FATE [17] and SecureML [19], FedSVD-LR is 100× faster than SecureML and 10× faster than FATE.

CCS Concepts

• **Security and privacy → Privacy-preserving protocols**; • **Computing methodologies → Factorization methods**.

KDD '22, August 14–18, 2022, Washington, DC, USA.

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539402

Keywords

Federated Learning; SVD; Privacy-preserving Matrix Factorization

ACM Reference Format:
Di Chai, Leye Wang, Junxue Zhang, Liu Yang, Shuowei Cai, Kai Chen, and Qiang Yang. 2022. Practical Lossless Federated Singular Vector Decomposition over Billion-Scale Data. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539402

1 Introduction

Singular vector decomposition (SVD) is an essential primitive to build various data analytics and machine learning applications over large-scale data. SVD is widely used in 1) principal component analysis (PCA) to reduce the dimensionality of large-scale features; 2) latent semantic analysis (LSA) [8] on large-scale natural language processing (NLP) tasks to extract compressed embedding features. These large-scale data usually come from various data sources in the real world [26] and it is hard for a single institution to collect sufficient data to produce robust results.

However, since the enacting of privacy-preserving laws, *e.g.*, GDPR [23], the data from different sources are restricted from being collected in one central place for conventional centralized SVD computation. To solve the problem, pioneer researchers have explored the SVD in a *federated*[1] approach (Federated SVD), *i.e.*, the SVD computation can be performed cooperatively by different participants without revealing or gathering their original data. We will give a formal definition of Federated SVD in §2.1.

In this paper, we show that these Federated SVD solutions [2, 9, 11, 16] cannot *efficiently* and *accurately* process large-scale data, making them impractical to support real-world SVD applications. Specifically, prior works either leverage the differential privacy (DP) or the homomorphic encryption (HE) for privacy protection. The DP-based solution [9] suffers from dramatic *accuracy loss* because it adds unremovable noise in the data to hide individual privacy. Our evaluation results show that DP-based federated SVD ($\epsilon = 0.1, \delta = 0.1$) has over 10 orders of magnitude larger

[1]We use the term of *federated* since the Federated SVD works similarly as Federated Learning [26].

accuracy loss compared with centralized SVD (more details are presented in §2.2). The inherent loss of data utility hindered its application in the real world [22], *e.g.*, inaccurate SVD results during medical studies [13] can cause severe issues in the subsequent medical diagnosis tasks. In contrast, the HE-based solution [16] can achieve lossless accuracy. However, the HE involves large computation/communication *overhead* due to the inflated data, causing significant performance degradation. Our evaluation results show that it takes ~ 15 years for HE-based methods [16] to factorize a 1K × 100K matrix, *i.e.*, 100 million elements (more details in in §2.2).

We ask: *Can we build a practical lossless and efficient federated SVD solution over billion-scale data.* Our answer is FedSVD. The core of FedSVD is a matrix masking method delicately designed for the SVD algorithm. The advantage of this matrix masking method is that it can simultaneously achieve lossless accuracy and high efficiency. Specifically, 1) The masking method protects users' private data by multiplying two random orthogonal matrices. These random masks can be removed entirely from the final SVD results to achieve *lossless* accuracy; 2) Unlike HE-based methods that significantly inflate the original data (*e.g.*, from 64-bits to 2048-bits), FedSVD's masking method does not inflate the data size. Therefore, FedSVD can achieve similar performance as centralized SVD theoretically. Furthermore, based on the matrix masking method, we design optimization strategies including block-based mask generation, efficient data masking/recovering through block matrix multiplication, mini-batch secure aggregation, and advanced disk offloading to further improve the efficiency of communication, computation, and memory usage (more details in §3). Eventually, we have provided privacy analysis and attack experiments showing that FedSVD is secure and the raw data cannot be revealed from the masked data when the hyper-parameter is appropriately settled.

We implement and evaluate FedSVD on SVD tasks and three applications: PCA, linear regression (LR), and LSA. Our evaluation results show that: 1) On SVD tasks, FedSVD has 10 orders of magnitude smaller error compared with DP-based methods, and FedSVD is more than 10000× faster than HE-based solution. Approximately, the HE-based method needs more than 15 years to factorize 1K × 100K data (*i.e.*, 100 million elements), while FedSVD only needs 16.3 hours to factorize 1K × 50M data which containers 50 billion elements; 2) On PCA application, FedSVD takes 32.3 hours to compute the top 5 principal components on 100K × 1M synthetic data, which contains 100 billion elements; 3) On LR application, we have compared FedSVD with two well-known federated LR solutions: SecureML [19] and FATE [17]. The evaluation results show that FedSVD is 100× faster than SecureML and 10× faster than FATE; 4) On LSA application, FedSVD takes 3.71 hours to compute the top 256 eigenvectors on a 62K × 162K MovieLens real-world datasets, which contains 10 billion elements; 5) We perform attack experiments showing that, given proper hyper-parameter, FedSVD is secure against state-of-the-art (SOTA) ICA attack [15] which is delicately designed for masked data.

The FedSVD is fully open-sourced [2] and we believe, besides the three mentioned applications, FedSVD can benefit more applications that require SVD as their cores under the increasingly strict data protection laws and regulations.

[2]https://github.com/Di-Chai/FedEval/tree/master/research/FedSVD

2 Background & Motivation

2.1 SVD and Federated SVD

SVD decomposes matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ into a product of three matrices

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}} \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V}^{\mathbf{T}} \in \mathbb{R}^{n \times n}$ are the left and right singular vectors, $\Sigma \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix containing the singular values. $\mathbf{U}$ and $\mathbf{V}^{\mathbf{T}}$ are orthogonal matrices.

SVD is an essential building block in many studies. Here we introduce two of the most well-known SVD-based applications and they all require lossless accuracy and large-scale performance to be simultaneously achieved. 1) Principal components analysis (PCA). PCA is one of the most essential techniques for eliminating redundancy in high-dimensional data, and it is widely used in medical diagnosis [13], biometrics [20], and many other applications [21]. SVD is the standard solution to conduct PCA. The SVD-based PCA deals with large-scale private data containing high-dimensional features, and it also requires lossless accuracy to avoid severe issues like inaccurate disease analysis. 2) Linear regression (LR). LR is a popular machine learning model commonly used for risk management, marketing, *etc.*, for its high efficiency and interpretability. SVD could serve as a basis of the least square solution to LR. Compared with stochastic gradient descent (SGD), solving LR through SVD requires only one iteration and guarantees that the result is the global optimum. In such commercial scenarios, SVD-based LR deals with large-scale sensitive user data and requires lossless precision to avoid financial loss.

Typically, the federated SVD is defined as following: assume we have k parties, and each party i owns data matrix $\mathbf{X}_i \in \mathbb{R}^{m \times n_i}$. Those k parties would like to carry out SVD jointly on data $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_k]$, where $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $n = \sum_{i=1}^{k} n_i$.

$$[\mathbf{X_1}, ..., \mathbf{X_i}, ..., \mathbf{X_k}] = \mathbf{U}\Sigma[\mathbf{V_1^T}, ..., \mathbf{V_i^T}, ..., \mathbf{V_k^T}] \tag{2}$$

Eq. (2) shows the federated SVD results. Accordingly, in a federated SVD solution, the i-th party ($1 \le i \le k$) gets $\mathbf{X}_i = \mathbf{U}\Sigma\mathbf{V}_i^T$, where $\mathbf{U}, \Sigma$ are shared results among all participants, and $\mathbf{V}_i^T \in \mathbb{R}^{n \times n_i}$ is the secret result possessed by party-i. Figure 1 also illustrates the above problem definition. Party-i's data (*i.e.,* $\mathbf{X}_i$) cannot be leaked to any other parties during the computation.

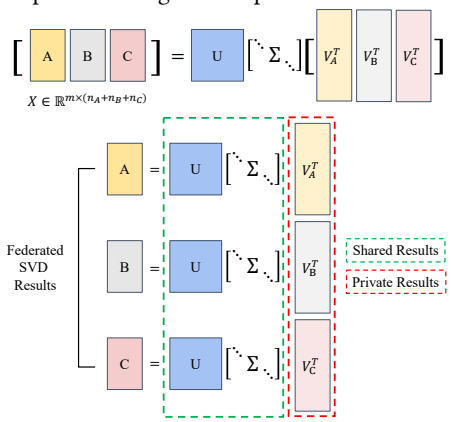
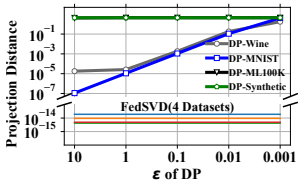


Figure 1: Problem formulation of federated SVD

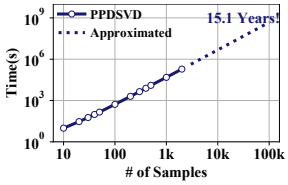
The real-world applications mainly contain two data partition scenarios, *i.e.*, horizontally and vertically partitioned scenarios.

Horizontally partitioned scenario assumes that different parties share the same feature space but different sample space, while the vertically partitioned scenario assumes that participants share different feature space but the same sample space. In this paper, we do not make assumption on the data partition schema and our method is suitable for both two scenarios. Because one type of partition could be easily transferred to another through matrix transpose in SVD. Without loss of generality, as shown in Fig. 1, we assume the data matrix is vertically partitioned among parties.

2.2 Prior Work Suffering from Either Accuracy Loss or Performance Penalty



(a) DP-SVD ($\delta = 0.01$) has 7 ~ 14 magnitudes larger error compared with FedSVD on four datasets.



(b) HE-based SVD needs 15.1 years to factorize 1K × 100K data (100 million elements).

Figure 2: Quantifying accuracy loss and performance penalty.

Pilot federated learning work designed privacy-preserving SVD methods in two brunches: the DP-based and HE-based methods.

Accuracy Loss: On the one hand, Grammenos et al. [9] proposed a federated and (ϵ, δ)-DP principal component analysis method, in which the leaf nodes apply DP locally and upload the local PCA results to one root node, which will asynchronously aggregate the received updates. Although DP-based solutions are easy to implement and does not have efficiency issue, it unavoidably brings loss to the data utility and hindered its application in real-world [22]. For example, accuracy loss of SVD in medical study can cause severe issues in subsequent medical diagnosis. Fig. 2(a) shows that DP-based SVD has 7 ~ 14 orders of magnitude larger error compared with FedSVD under different parameters.

Performance Penalty: On the other hand, Liu and Tang [16] proposed a HE-based SVD solution, in which the parties jointly compute the covariance under additive HE (*i.e.*, HE algorithm that only supports addition operation on cipher-text), then a trusted server decrypts the covariance matrix and conducts the SVD. Although HE is lossless, it brings heavy computation and communication overhead because it swells up the data size from 64-bit to 2048-bit, assuming the key length is set to 2^{20} bits. Thus HE-based SVD has large computation overhead. In particular, as shown in Fig. 2(b), HE-based method needs more than 15 years to factorize a 1K × 100K data (*i.e.*, million-scale data).

Conclusion: None of the exiting federated SVD work can simultaneously achieve lossless accuracy and high efficiency.

3 FedSVD

To solve this problem, we ask: *Can we find a type of removable noise to protect data privacy as well as keep the data size unchanged to simultaneously achieve lossless accuracy and high efficiency?* Our answer is FedSVD. Briefly, 1) we propose a removable random mask delicately designed for SVD to protect privacy, and the masks could be completely removed from SVD results; 2) The masked data has the same size as the raw data, which results in no efficiency overhead during matrix decomposition. Meanwhile, we propose optimizations from algorithm and system aspects, including block-based mask generation, efficient data masking and data recovering, mini-batch secure aggregation, and advanced disk offloading, to further improve efficiency. With our delicate design, FedSVD could achieve lossless accuracy and high practicality over billion-scale data. Furthermore, we provide privacy analysis on FedSVD and show that FedSVD is highly confidential. In this section, we present the technical details of FedSVD.

Roles: According to the different functionalities, we specify three types of roles in our system:

- **Trusted Authority (TA)**: TA is responsible for generating removable secret masks and delivering them to the users. TA can remain offline once the system initialization is done.
- **Computation Service Provider (CSP)**: The CSP is responsible for running a standard SVD algorithm on the masked data and delivering the masked SVD results to the users.
- **Users**: The parties that own raw data (*i.e.*, $\mathbf{X}$) and wish to run an SVD-based algorithm jointly.

Workflow Overview: FedSVD has the following four steps, which is also illustrated in Fig. 3 :

Step ❶ : TA generates two removable random orthogonal masks $\mathbf{P} \in \mathbb{R}^{m\times m}$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$. Mask $\mathbf{P}$ is broadcasted to all users. The matrix $\mathbf{Q}$ is horizontally split into k parts $\mathbf{Q}^T = [\mathbf{Q}_1^T, ., \mathbf{Q}_i^T, ., \mathbf{Q}_k^T]$, and TA sends $\mathbf{Q}_i$ to user-i. The detail of the removable random mask is introduced in §3.1. To support billion-scale applications, we have proposed efficiency mask generation (§3.1) and delivery method (§3.2), reducing the computation and communication complexity from $O(n^3)$ and $O(m^2 + n^2)$ to $O(n)$.

Step ❷ : All users compute $\mathbf{X}'_i = \mathbf{P}\mathbf{X}_i\mathbf{Q}_i$, where $\mathbf{X}'_i$ is the local masked data. The CSP gets $\mathbf{X}'$ through secure aggregation on $\mathbf{X}'_i$. To support billion-scale data, we propose efficient block matrix multiplication to reduce computation complexity from $O(m^2n + mn^2)$ to $O(mn)$ and mini-batch secure aggregation to reduce the memory usage at the server. More details are introduced in §3.2.

Step ❸ : CSP runs a standard SVD algorithm, factorizing $\mathbf{X}'$ into $\mathbf{U}'\mathbf{\Sigma}\mathbf{V}'^T$. We do not specify the algorithm (*e.g.*, householder transformation) of solving the SVD problem, and FedSVD can work with any lossless SVD solver.

Step ❹ : Users downloads $\mathbf{U}', \mathbf{\Sigma}$, and recover $\mathbf{U}$ by $\mathbf{P}^T\mathbf{U}'$. $\mathbf{V}_i^T$ is jointly recovered under the protection of random masks between the CSP and users. We propose efficient mask removing of $\mathbf{V}_i^T$ via block matrix computation which reduces the complexity from $O(n_i^3)$ to $O(n_i)$. The details are introduced in §3.3.

Organization of this section: §3.1 introduces the removable random mask delicately designed for SVD. §3.2 introduces the detail of mask initialization and applying the mask, §3.3 introduces the detail of removing mask. In §3.4, we propose an advanced disk offloading strategy according to the data access patterns. §3.5 gives privacy analysis of FedSVD.

3.1 Removable Random Masks for SVD

We propose a masking method that allows running SVD directly on the masked data and the masks could be removed from the results. Denoting the data matrix as $\mathbf{X}$, we use two random orthogonal

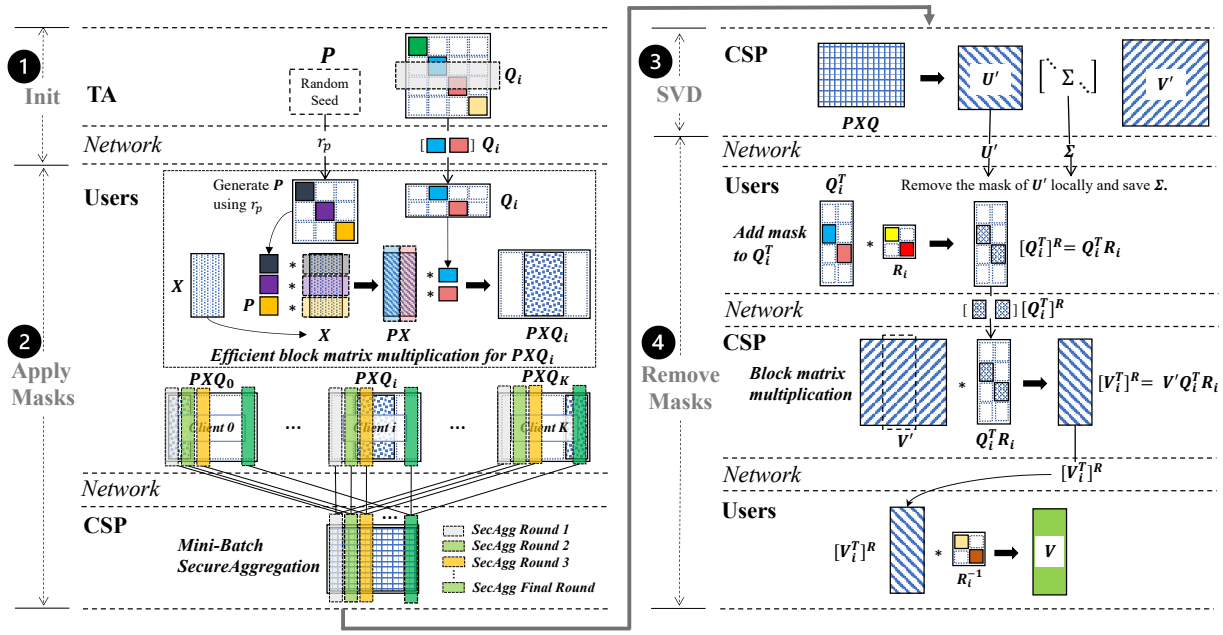


Figure 3: Detailed Workflow of FedSVD, which has four steps: Step ❶: Trusted Authority (TA) initialize and send the masks to users. Step ❷: Users apply masks and deliver the masked data to the computation service provider (CSP) through secure aggregation. Step ❸: CSP conducts standard SVD on masked data. Step ❹: Users remove the masks and get final results.

matrices $\mathbf{P}$ and $\mathbf{Q}$ to mask the data $\mathbf{X}$ as $\mathbf{X}' = \mathbf{PXQ}$. Theorem 1 proves that $\mathbf{X}'$ has the same singular values with $\mathbf{X}$, the singular vectors of $\mathbf{X}$ and $\mathbf{X}'$ can be orthogonal transformed to each other using matrices $\mathbf{P}$ and $\mathbf{Q}$. Thus we can get the singular vectors of $\mathbf{X}$ by removing the masks from the singular vectors of $\mathbf{X}'$ (*i.e.,* using orthogonal transformation).

Theorem 1. *For an arbitrary matrix* $\mathbf{X} \in \mathbb{R}^{m\times n}$ *with SVD result* $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\mathbf{T}$, *we can use two random orthogonal matrices* $\mathbf{P} \in \mathbb{R}^{m\times m}$ *and* $\mathbf{Q} \in \mathbb{R}^{n\times n}$ *to mask* $\mathbf{X}$ *into* $\mathbf{X}' = \mathbf{PXQ}$. *Assuming the SVD result of* $\mathbf{X}'$ *is* $\mathbf{U}'\Sigma'\mathbf{V}'^T$. *Then we can get SVD results of* $\mathbf{X}$ *through:* $\Sigma = \Sigma'$, $\mathbf{U} = \mathbf{P}^T\mathbf{U}'$ *and* $\mathbf{V}^\mathbf{T} = \mathbf{V}'^T\mathbf{Q}^T$.

Proof. By plugging $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\mathbf{T}$ into $\mathbf{X}'$, $\mathbf{X}'$ could be represented as $\mathbf{X}' = \mathbf{PXQ} = (\mathbf{PU})\Sigma(\mathbf{V}^T\mathbf{Q})$. According to Eq. (3), $\mathbf{PU}$ and $\mathbf{V}^T\mathbf{Q}$ are orthogonal matrices:

$$\begin{aligned}(\mathbf{PU})^{-1} &= \mathbf{U}^{-1}\mathbf{P}^{-1} = \mathbf{U}^\mathbf{T}\mathbf{P}^\mathbf{T} = (\mathbf{PU})^\mathbf{T} \\ (\mathbf{V}^\mathbf{T}\mathbf{Q})^{-1} &= \mathbf{Q}^{-1}(\mathbf{V}^\mathbf{T})^{-1} = \mathbf{Q}^\mathbf{T}\mathbf{V} = (\mathbf{V}^\mathbf{T}\mathbf{Q})^\mathbf{T}\end{aligned} \tag{3}$$

Then $(\mathbf{PU})\Sigma(\mathbf{V}^T\mathbf{Q})$ is the SVD result of $\mathbf{X}'$. Accordingly, $\mathbf{PU} = \mathbf{U}'$, $\Sigma = \Sigma'$, and $\mathbf{V}^T\mathbf{Q} = \mathbf{V}'^T$. Then $\mathbf{U} = \mathbf{P}^T\mathbf{U}'$ and $\mathbf{V}^\mathbf{T} = \mathbf{V}'^T\mathbf{Q}^T$. □

We present a random orthogonal matrix generation method in Algorithm 1 using the Gram-Schmidt process [5]. It is proved in prior work [10] that Gram-Schmidt process on Gaussian matrices produces uniformly distributed random orthogonal matrices.

Block-based Efficient Mask Generation: However, the complexity of Gram-Schmidt process on a n dimensional square matrix is $O(n^3)$ [5]. Thus we propose an efficient random orthogonal matrix generation algorithm through building blocks, which is presented in Algorithm 2. Briefly, we decompose the problem of generating a n dimensional orthogonal matrix into generating small orthogonal matrices with size b, placing these small matrices at the diagonal position, and forming a n dimensional matrix. Then the complexity of generating n dimensional orthogonal matrix reduces to $O(b^3\frac{n}{b}) = O(b^2 n) = O(n)$, where $b \ll n$.

Algorithm 1: Generate Random Orthogonal Matrix

Input: Dimension of the matrix n
Output: Orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$
Function `Orthogonal(n)`:
  Randomly sample matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$, where $R_{i,j} \sim \mathcal{N}(0,1)$
  $[\mathbf{Q}, \sim] = GramSchmidt(\mathbf{R})$
  return $\mathbf{Q}$
End Function

Algorithm 2: Efficient Orthogonal Matrix Generation Through Building Blocks

Input: Dimension of the matrix n, size of building blocks b
Output: Orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$
Function `EfficientOrthogonal(n, b)`:
  $\mathbf{Q} \leftarrow []$, $i \leftarrow 0$
  while $i < n$ **do**
    $b' \leftarrow min(b, n-i)$
    $\mathbf{Q}_b \leftarrow Orthogonal(b')$ `// Algorithm 1`
    $\mathbf{Q} \leftarrow \begin{bmatrix}\mathbf{Q} & \mathbf{0}\\ \mathbf{0} & \mathbf{Q}_b\end{bmatrix}$, $i \leftarrow i + b'$
  end
  return $\mathbf{Q}$
End Function

Block size controls the trade-off between efficiency and privacy protection It is worth noting that the block size (*i.e.*, b) simultaneously impacts the system privacy protection and efficiency. Theoretically, large block size increases the freedom of the masks, thus increases the effectiveness of privacy protection. Meanwhile, large block size increases the computation overhead, thus decreases the system efficiency. We have reported attack experiments using

the SOTA attack method in §5.4 showing that the attack fails in recovering valid information as long as b is large enough. We set $b = 1000$ in our experiments since our attacking experiments on many datasets show that 1000 is a good choice of gaining enough privacy protection and benefiting from the efficiency brought by the block-based optimizations. The proper block size may differ on different datasets, and we suggest adjusting block size according to the datasets in the application, which is also discussed in §5.4.

3.2 Initialization & Applying the Masks

We propose a federated computation process based on the removable random masks to apply masks on the raw data. At the beginning of the computation, TA holds masks $\mathbf{P}, \mathbf{Q}$ and users jointly hold $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_k]$. At the end of the computation, CSP receives $\mathbf{X'} = \mathbf{PXQ}$ and does not learn any other information.

Equation 4 shows our idea of federally computing $\mathbf{X'}$. According to the rule of block matrix multiplication, we can decompose $\mathbf{PXQ}$ into $\sum_{i=1}^{k} \mathbf{PX}_i\mathbf{Q}_i$.

$$\mathbf{X'} = \mathbf{PXQ} = \mathbf{P}[\mathbf{X_1}, ., \mathbf{X_i}, ., \mathbf{X_K}][\mathbf{Q_1^T}, ., \mathbf{Q_i^T}, ., \mathbf{Q_k^T}]^\mathbf{T} = \sum_{i=1}^{k} \mathbf{PX_iQ_i} \quad (4)$$

Thus the federated computation of $\mathbf{X'}$ can be divided into two steps. **Step ❶**: TA broadcasts $\mathbf{P}$ to all users, then horizontally splits the mask $\mathbf{Q}$ into $\{\mathbf{Q}_i \in \mathbb{R}^{n_i \times n} | 1 \le i \le k\}$, and sends $\mathbf{Q}_i$ to user-i. **Step ❷**: Users compute $\mathbf{PX_iQ_i}$, the CSP runs a secure aggregation to get $\sum_i \mathbf{PX}_i\mathbf{Q}_i$. The secure aggregation conceals the intermediate results (*i.e.,* $\mathbf{PX}_i\mathbf{Q}_i$), and guarantees that CSP only learns $\mathbf{X'}$.

Communication Efficient Mask Delivery: We observe that directly transferring $\mathbf{P}, \mathbf{Q}$ has $O(m^2{+}n^2)$ communication complexity. Based on Algorithm 2, we propose to reduce the communication complexity through transferring only one random number or small blocks of the mask. More specifically, the TA only broadcast a random seed r_p for mask $\mathbf{P}$ since Gram-Schmidt is a deterministic algorithm that yields the same orthogonal matrix as long as the input matrices are the same, thus the users can generate $\mathbf{P}$ locally using the same random seed. TA only sends the sliced matrix blocks for mask $\mathbf{Q}$ and the zeros are omitted during the transmission. In summary, communication complexity of transferring $\mathbf{P}, \mathbf{Q}$ are reduced to $O(1)$ and $O(b^2\frac{n}{b}) = O(n)$.

Efficient Data Masking via Block Matrix Multiplication: We observe that our data masking process (*i.e.*, computing $\mathbf{PXQ}$) has cubic complexity (*i.e.*, $O(m^2n{+}mn^2)$) which brings large computation overhead in large-scale applications. To reduce the complexity, we adopt block matrix multiplications since $\mathbf{P}, \mathbf{Q}$ are sparse matrices and consist of blocks. A concrete example is presented in Eq. (5), where the zeros are omitted in the computation. After adopting the block matrix multiplication, the data masking complexity is reduced from cubic complexity to $O(\frac{m}{b} * b^2 * n + \frac{n}{b} * b^2 * m) = O(mn)$.

$$\begin{bmatrix} \mathbf{P_1} & 0 & 0 \\ 0 & \mathbf{P_2} & 0 \\ 0 & 0 & \mathbf{P_3} \end{bmatrix} \begin{bmatrix} \mathbf{X_1} \\ \mathbf{X_2} \\ \mathbf{X_3} \end{bmatrix} = \begin{bmatrix} \mathbf{P_1X_1} \\ \mathbf{P_2X_2} \\ \mathbf{P_3X_3} \end{bmatrix} \quad (5)$$

Memory Efficient Mini-batch Secure Aggregation: We observe that secure aggregation (SecAgg) directly processes the whole data matrix (*i.e.*, $\mathbf{X}'_i = \mathbf{PX}_i\mathbf{Q}_i$), and it will bring significant memory burden to the server and users in FedSVD since $\mathbf{X}'_i$ is a large matrix. We propose to split $\mathbf{X}'_i$ into batches and only process one batch of data in each round of SecAgg. Mini-batch SecAgg works because the aggregations of different rows or columns of $\mathbf{X}'_i$ are independent.

3.3 Removing the Masks

Intuitively, the masks in the final results could be removed by each user locally if the CSP broadcast $\mathbf{U'}$ and $\mathbf{V'}^T$, *i.e.* $\mathbf{U} = \mathbf{P}^T\mathbf{U'}$ and $\mathbf{V'}_i^T = \mathbf{Q}_i^T\mathbf{V'}^T$. However, $\mathbf{V'}^T$ contains masked eigenvectors of all users, sending $\mathbf{V'}^T$ from CSP to users may bring privacy issues because users hold more information than CSP, *e.g.*, $\mathbf{Q}_i$. Thus we propose a federated computation process to recover $\mathbf{V'}^T$. For $\mathbf{U'}$, users can remove the mask locally because $\mathbf{U}$ is the defined as the shared result in federated SVD (*i.e.*, §2.1).

During the recovery of $\mathbf{V'}^T$, we want to guarantee the confidentiality of both $\mathbf{Q}_i^T$ and $\mathbf{V'}$, *i.e.,* the users cannot get the whole $\mathbf{V'}$ matrix and the CSP cannot learn $\mathbf{Q}_i^T$.

Our solution is first masking $\mathbf{Q}_i^T$ using another random matrix $\mathbf{R}_i \in \mathbb{R}^{n_i \times n_i}$ according to Eq. (6). Then user i sends the $[\mathbf{Q_i^T}]^R$ (*i.e.,* the masked $\mathbf{Q}_i^T$) to the CSP, which will subsequently compute $[\mathbf{V}_i^T]^R$ and send $[\mathbf{V}_i^T]^R$ back to user i. Then user i can remove the random mask according to Eq. (6) and get the final result (*i.e.,* $\mathbf{V}_i^T$).

$$[\mathbf{Q_i^T}]^R = \mathbf{Q_i^T}\mathbf{R_i}, [\mathbf{V_i^T}]^R = \mathbf{V'^T}[\mathbf{Q_i^T}]^R, \mathbf{V_i^T} = [\mathbf{V_i^T}]^R\mathbf{R_i^{-1}} \quad (6)$$

It is worth noting that $\mathbf{V}_i^T$ also could be recovered through $\mathbf{V}_i^T = \Sigma_n^{-1}\mathbf{U}_n^T\mathbf{X}_i$, where Σ_n^{-1} and $\mathbf{U}_n^T$ mean the first n rows of Σ^{-1} and $\mathbf{U}^T$. However, this method only works when $m >= n$. When $m < n$, we can only recover the first m rows of $\mathbf{V}_i^T$ but not the full matrix. Thus this method is not a general solution.

Efficient Recovery of $\mathbf{V}^T$ via Block Matrix Computation: We observe that although $\mathbf{Q}_i^T$ is a sparse matrix consisting of blocks according to Algorithm 2. However, the computing and transferring $\mathbf{Q}_i^T\mathbf{R}_i$ is costly since $\mathbf{R}_i$ is a dense random matrix. The computation and communication of $\mathbf{Q}_i^T\mathbf{R}_i$ has $O(\frac{n_i}{b}b^2n_i) = O(n_i^2)$ complexity. To improve efficiency, our solution generates $\mathbf{R}_i$ through putting a bunch of square random matrix diagonally and the size of each small random matrix is decided by $\mathbf{Q}_i^T$, such that $\mathbf{Q}_i^T\mathbf{R}_i$ is still a sparse matrix consists of blocks, Eq. (7) shows an example. The complexity is reduced from $O(n_i^2)$ to $O(\frac{n_i}{b}b^3) = O(n_i)$. Moreover, since $\mathbf{R}_i$ is consist of block matrices, the complexity of computing its inverse (*i.e.*, $\mathbf{R_i^{-1}}$) also reduces from $O(n_i^3)$ to $O(n_i)$.

$$\mathbf{Q_i^T R_i} = \begin{bmatrix} 0 & 0 \\ \mathbf{Q_{i,1}^T} & 0 \\ 0 & \mathbf{Q_{i,2}^T} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R_i^1} & 0 \\ 0 & \mathbf{R_i^2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \mathbf{Q_{i,1}^T R_i^1} & 0 \\ 0 & \mathbf{Q_{i,2}^T R_i^2} \\ 0 & 0 \end{bmatrix} \quad (7)$$

3.4 Disk Offloading via Data Access Patterns

Observation: Dealing with large-scale matrices usually requires large hardware memory. For example, a 100K × 1M 64-bit matrix requires approximately 745GB RAM, making the memory space for computation very limited. A standard solution is offloading part of the memory storage to disks using swap memory and reloading the data when needed. The operating system (OS) will automatically schedule the disk offloading. However, naively following the OS scheduling with no specific design for our algorithm is inefficient.

Solution: We propose an advanced disk offloading strategy according to the data access pattern for FedSVD. 1) Offloading strategy

for $\mathbf{P}, \mathbf{Q}$. According to our observation, $\mathbf{P}, \mathbf{Q}$ are used twice in the computation when applying and removing the masks. Hence, on the client-side, we immediately save the blocks of $\mathbf{P}, \mathbf{Q}$ to disk when they are generated or received from TA. When applying or removing the masks, we load and use $\mathbf{P}, \mathbf{Q}$ block by block (*i.e.*, sequentially). And each block will be removed from the memory when its computation finishes; 2) Offloading strategy for large dense data matrices (*e.g.*, $\mathbf{X}, \mathbf{PXQ}, \mathbf{U}, \mathbf{V}^T$). We store the large data matrices in disk and leave a file map in memory. The file map will automatically read the needed matrix components. However, direct adoption of file maps may bring severe efficiency issues. The file map uses consistent storage on the disk and the matrix is stored by rows by default. If the manner we access the matrix conflicts with the storage manner (*e.g.*, access by column), the efficiency will be very low. Thus we have optimized the implementation such that all the file-map matrices are stored adaptively according to the access pattern. The evaluation results show that our advanced disk offloading strategy reduces the time consumption by 44.7% compared with using swap memory scheduled by OS. Detail could be found in §5.5.

3.5 Privacy Analysis

In this section, we analyze the confidentiality of FedSVD. We consider the TA to be a fully trusted entity, while the CSP and users are semi-honest parties. This means, both the CSP and users will honestly follow the pre-designed protocols but also attempt to infer private data. We also assume there is no collusion between the CSP and the users.

CSP cannot reveal the original matrix: According to Fig. 3, the total messages received by the CSP are $\mathbf{X}' = \sum_i [\mathbf{X}'_i]^R$ and $[\mathbf{Q}_i^T]^R$. 1) According to the prior work [3], the CSP only learns the aggregated results $\mathbf{X}'$, no information is leaked during the secure aggregation; 2) In Theorem 2, we show that there is an infinite number of raw data that could be masked into the same matrix. If the CSP has no prior knowledge about the data distribution, it can never recover the true data because the true data is not identifiable. Alternatively, the CSP can empirically choose data distribution as prior knowledge and perform attacks [15] on the masked data. However, the attack experiments in §5.4 show that the attack fails in getting valid information if we set the hyper-parameter properly; 3) According to the prior work [27], the masked data $[\mathbf{Q}_i^T]^R$ cannot be computationally distinguished from a random matrix, thus leaks no information.

In conclusion, FedSVD is secure against CSP which cannot reveal the raw data.

Theorem 2. *Given a masked data* $\mathbf{X}' = \mathbf{P}_1\mathbf{X}_1\mathbf{Q}_1$, *there are infinite number of raw data* $\mathbf{X}_2$ *that can be masked into* $\mathbf{X}'$, i.e., $\mathbf{P}_2\mathbf{X}_2\mathbf{Q}_2 = \mathbf{P}_1\mathbf{X}_1\mathbf{Q}_1 = \mathbf{X}'$.

Proof. Given two random orthogonal matrix $\mathbf{R}_1 \in \mathbb{R}^{m\times m}$ and $\mathbf{R}_2 \in \mathbb{R}^{n\times n}$, we can rewrite $\mathbf{X}'$ into

$$\begin{aligned}\mathbf{X}' &= \mathbf{P}_1\mathbf{X}_1\mathbf{Q}_1 = \mathbf{P}_1\mathbf{U}\Sigma\mathbf{V}^\mathbf{T}\mathbf{Q}_1 = \mathbf{P}_1\mathbf{U}(\mathbf{R}_1^\mathbf{T}\mathbf{R}_1)\Sigma(\mathbf{R}_2\mathbf{R}_2^\mathbf{T})\mathbf{V}^\mathbf{T}\mathbf{Q}_1 \\ &= (\mathbf{P}_1\mathbf{U}\mathbf{R}_1^\mathbf{T})(\mathbf{R}_1\Sigma\mathbf{R}_2)(\mathbf{R}_2^\mathbf{T}\mathbf{V}^\mathbf{T}\mathbf{Q}_1)\end{aligned} \tag{8}$$

Let $\mathbf{X}_2 = \mathbf{R}_1\Sigma\mathbf{R}_2$, $\mathbf{P}_2 = \mathbf{P}_1\mathbf{U}\mathbf{R}_1^T$, $\mathbf{Q}_2 = \mathbf{R}_2^T\mathbf{V}^T\mathbf{Q}_2$, then we get $\mathbf{P}_2\mathbf{X}_2\mathbf{Q}_2 = \mathbf{P}_1\mathbf{X}_1\mathbf{Q}_1 = \mathbf{X}'$. $\mathbf{R}_1, \mathbf{R}_2$ are random orthogonal matrices and the number of orthogonal matrices with certain size is infinite in real number field, thus we have infinite number of $\mathbf{X}_2$ that also can be masked into $\mathbf{X}'$ and the CSP cannot identify the real data. □

The users can only learn the final results: According to Fig. 3, the user i receives: $\mathbf{P}, \mathbf{Q}_i, \mathbf{U}', \Sigma, [\mathbf{V}_i^T]^R$, and the valid information are $\mathbf{U}, \Sigma, \mathbf{V}_i$, which are exactly the final results of the federated SVD problem defined in §2.1. Thus each user only learn its final results and receives nothing about other users' private data. Additionally, FedSVD is secure against collusion between the users because a group of cooperated users could be treated as a single user who owns more local data, and they cannot obtain the privacy of other users outside the group.

The TA learns nothing: Since TA receives nothing in Fig. 3 and remains offline after initialization, it learns nothing in the algorithm.

In summary, FedSVD is secure against CSP, TA receives nothing during the computation, and the users only get their final results. FedSVD is highly confidential.

4 Applications Based on FedSVD

Based on FedSVD, we propose three applications: principal component analysis (PCA), linear regression (LR), and latent semantic analysis (LSA). All these applications have the same first three steps with FedSVD and only differ at the last step. Tailored optimizations are also made for each application to further improve efficiency.

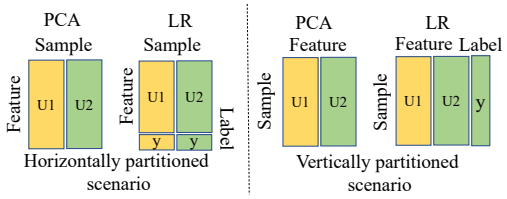


Figure 4: Federated PCA and LR under different data settings.

PCA in horizontally partitioned scenario: PCA in federated learning setting typically has two data partition schemas, *i.e.*, horizontally and vertically, which are illustrated in Fig. 4. In this paper, we consider the horizontal federated PCA since it is the most common data setting in medical and biometric studies in which multiply institutions have the same feature on different samples. Given a normalized matrix $\mathbf{X}$, PCA decomposes it into $\mathbf{X} = \mathbf{U}_r\Sigma_r\mathbf{V}_r^T$, where r is the number of principal components in PCA, $\mathbf{U}_r \in \mathbb{R}^{m\times r}$ and $\mathbf{V}_r^T \in \mathbb{R}^{r\times n}$ are the top-r singular vectors with largest singular values. Such decomposition is also called truncated SVD. Considering PCA in horizontally partitioned scenario, the PCA result for user i is $\mathbf{U}_r^T\mathbf{X}_i \in \mathbb{R}^{r\times n_i}$. Accordingly, in FedSVD-based PCA, CSP only calculates and broadcasts the masked $\mathbf{U}'_r$ to all users and ignores the computation and transmission of $\Sigma, \mathbf{V}'^T$ to improve efficiency.

LR in vertically partitioned scenario: LR in federated learning setting also has two data partition schemas, *i.e.*, horizontally and vertically, which are illustrated in Fig. 4. In this paper, we consider the vertical federated LR since it is the most common scenario of federated risk management and marking in the real-world applications [26], in which different institutions hold different features on the same samples. Given a data matrix $\mathbf{X} = [\mathbf{X}_0; b] \in \mathbb{R}^{m\times n}$ and label $\mathbf{y}$, where $\mathbf{b}$ is the bias term, LR try to find a vector $\mathbf{w} \in \mathbb{R}^n$ such that $\mathbf{y} = \mathbf{X}\mathbf{w}$. $\mathbf{w}$ could be solved through SVD on $\mathbf{X}$ and $\mathbf{w} = \mathbf{V}\Sigma^{-1}\mathbf{U}^T\mathbf{y}$. In FedSVD-based LR, the user add mask to label $\mathbf{y}$ through $\mathbf{y}' = \mathbf{P}\mathbf{y}$,

then upload the masked label to CSP, which will subsequently compute $\mathbf{w}' = \mathbf{V}'\Sigma^{-1}(\mathbf{U}')^T\mathbf{y}' = \mathbf{Q}^T\mathbf{V}\Sigma^{-1}\mathbf{U}^T\mathbf{y} = \mathbf{Q}^T\mathbf{w}$. Then CSP broadcast the masked parameter matrix $\mathbf{w}'$ to all users, and each user can get the local parameters through $\mathbf{w}_i = \mathbf{Q}_i\mathbf{w}'$, where $\mathbf{w}_i \in \mathbb{R}^{n_i}$. In our LR design, the CSP will only broadcast the masked parameters and the $\mathbf{U}', \Sigma$ and $\mathbf{V}'^T$ are not transmitted to improve the communication efficiency.

LSA: Federated LSA is not sensitive to the data partition schemas since there is no clear definition of sample and feature in LSA. Briefly, LSA decomposes a data matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ (*e.g.*, word-document matrix) into $\mathbf{X} = \mathbf{U}_r\Sigma_r\mathbf{V}_r^T$, where r is the number of embedding feature in LSA and Σ_r are the top-r singular values. After the decomposition, $\mathbf{U}_r$ and $\mathbf{V}_r^T$ are treated as embedding features and used in the subsequent tasks, *e.g.*, computing the similarity of different documents in NLP. Accordingly, in FedSVD-based LSA, the CSP and users run the same protocol of recovering $\mathbf{U}'$ and $\mathbf{V}'^T$ to recover their first r vectors with the largest singular values, and the vectors outside r are ignored to improve the efficiency.

5 Experiments

In this section, we provide a comprehensive evaluation of FedSVD regarding the lossless and efficiency on SVD task (§5.2) and three applications (§5.3). Then we present the attack experiments in §5.4. Lastly, we show the effectiveness of the proposed system optimizations in §5.5.

5.1 Experiment Settings

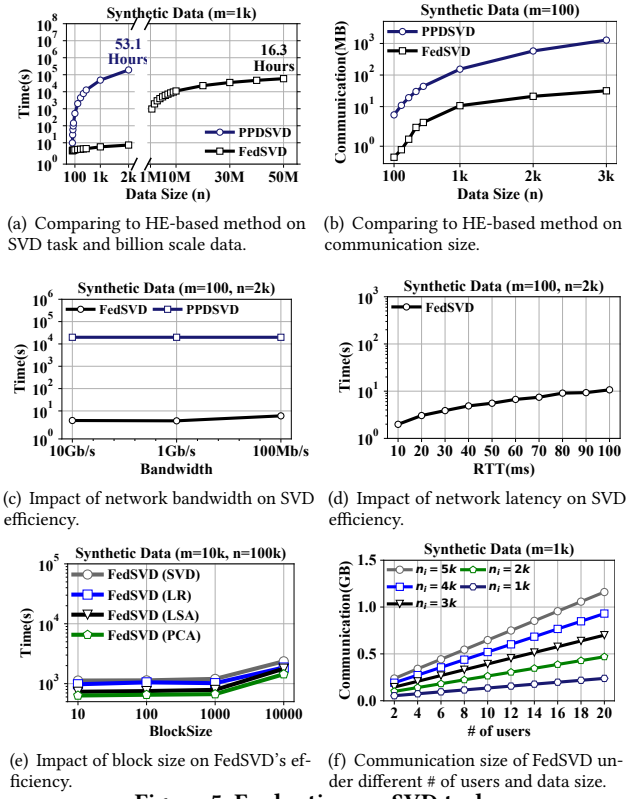
We have used five datasets in our experiments: MNIST [14], Wine [6], MovieLens-100K [12], MovieLens-25M [12], and synthetic data [9]. We compare FedSVD with three state-of-the-art models: WDA-PCA [2] which is a distributed rank-k PCA method, FedPCA [9] which is a federated (ϵ, δ)-differentially private PCA method, and PPD-SVD [16] which is a HE-based distributed SVD method. In particular, on LR application, we compare FedSVD with two well-known federated LR solution: FATE [17] and SecureML [19]. We set $b = 1000$ in FedSVD and $\epsilon = 0.1, \delta = 0.1$ for DP-based method. By default, following the prior work [11, 19, 26], we uniformly partition the data on two users, and partitioning data to more users will not impact our evaluations. Due to the space limitation, we put the detailed experiment setting in the Appendix A.

5.2 Evaluation on SVD

Lossless: We have proved in Theorem 1 that the masking-based protection in FedSVD is lossless. Here we would like to use more experimental results to show that the precision of FedSVD is lossless in the implementation.

We compare the precision of FedSVD with FedPCA on SVD tasks. The precision of SVD is measured by calculating the distance of singular vectors [9] between the proposed methods and the standalone SVD. We use root-mean-square-error (RMSE) as the distance metric. Tab. 1 shows the results. FedSVD has about 10 orders of magnitude smaller error compared with DP-based solution.

To give a more straightforward understanding, we also evaluate the reconstruction error of FedSVD, *i.e.,* distance to the raw data: $||\mathbf{X} - \mathbf{U}\Sigma\mathbf{V}^T||$. Using mean absolute percentage error as the metric, FedSVD's reconstruction error is only 0.000001% of the raw data. It is worth noting that FedSVD's tiny deviation in the experiment is brought by the floating number representation in computers. Theoretically, as proved in Theorem 1, FedSVD is lossless.



(a) Comparing to HE-based method on SVD task and billion scale data.

(b) Comparing to HE-based method on communication size.

(c) Impact of network bandwidth on SVD efficiency.

(d) Impact of network latency on SVD efficiency.

(e) Impact of block size on FedSVD's efficiency.

(f) Communication size of FedSVD under different # of users and data size.

Figure 5: Evaluation on SVD task.

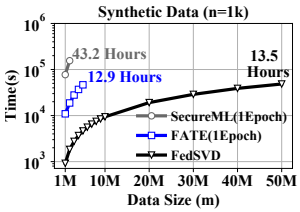
Time Consumption: Fig. 5(a) shows the time consumption of HE-based SVD (*i.e.*, PPDSVD) and FedSVD on large-scale data. Specifically, we use synthetic data matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, fix m = 1K, and vary n from 10 to 50 million. The experiment of PPDSVD stops at n = 2K because it takes too much time to further increase n. PPDSVD takes 53.1 hours to factorize a 1K × 2K matrix which is 10000× slower than FedSVD. Meanwhile, we also observe that the time consumption of PPDSVD increases quadratically with n when fixing m, while FedSVD increase linearly. Approximately, PPDSVD needs more than 15 years to factorize a 1K × 100K matrix, *i.e.*, million-scale elements. FedSVD only needs 16.3 hours to factorize a 1K × 50M matrix, which contains 50 billion elements.

Communication: FedSVD also has more than 10 times smaller communication size compared with PPDSVD, which is presented in Fig. 5(b). Fig. 5(c) and Fig. 5(d) show the efficiency when we change networking bandwidth and latency, and FedSVD works well given different networking conditions. Figure 5(f) shows the amount of communication data per user when we change the data size of each user (*i.e.*, n_i) and the number of users. Each user's communication size linearly increases with the size of local data.

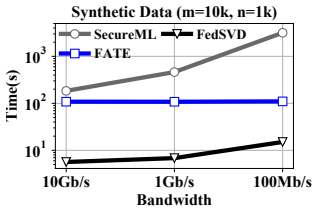
Hyper-parameter (Block Size): Block size is the only hyper-parameter in our solution and we present the system efficiency using different block size in Fig. 5(e). FedSVD's time consumption slowly increases with b. We suggest using a proper block size to gain enough privacy protection, which is discussed in §5.4, and benefit from the efficiency brought by the block-based optimizations.

Table 1: Lossless evaluation on SVD task and three applications.

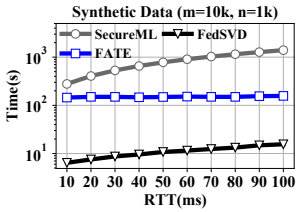
Datasets	SVD		PCA / LSA Applications			LR Application			
	FedPCA	FedSVD	FedPCA	WDA	FedSVD	SGD (10 Epoch) (FATE & SML)	SGD (100 Epoch) (FATE & SML)	SGD (1000 Epoch) (FATE & SML)	FedSVD
Wine	$3.25 * 10^{-1}$	$5.51 * 10^{-10}$	1.68	1.69	$1.37 * 10^{-10}$	1.04	0.767	0.666	0.539
MNIST	$9.37 * 10^{-2}$	$1.99 * 10^{-10}$	$5.34 * 10^{-2}$	$5.97 * 10^{-3}$	$2.79 * 10^{-14}$	48.7	5.53	3.78	3.19
ML100K	$7.95 * 10^{-2}$	$1.45 * 10^{-13}$	4.45	$6.02 * 10^{-1}$	$1.11 * 10^{-14}$	127	53.8	45.1	43.9
Synthetic	$1.79 * 10^{-1}$	$9.03 * 10^{-12}$	4.45	$9.13 * 10^{-4}$	$9.09 * 10^{-15}$	1.71	0.974	0.849	0.813



(a) Comparing FedSVD with FATE and SecureML on billion-scale data.



(b) Impact of network bandwidth on LR efficiency.



(c) Impact of network latency on LR efficiency.

Figure 6: Evaluation on LR Application.

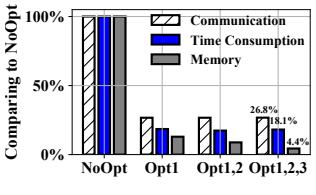


Figure 7: Effectiveness of the Proposed Optimizations.

5.3 Evaluation on the Applications

In this section, we evaluate FedSVD on three applications: PCA, LR, and LSA regarding accuracy and efficiency.

Lossless: The lossless evaluations of three applications are presented in Tab. 1. For PCA and LSA, we measure the precision by calculating the the projection distance [9] (*i.e.*, $||\mathbf{U}\mathbf{U}^T - \hat{\mathbf{U}}\hat{\mathbf{U}}^T||_2$) to standalone SVD. For LR, we report the mean square error (MSE) on the training data. PCA and LSA share the same evaluation results because their nature are both truncated SVD. In Tab. 1, we set $r = 10$ for PCA and LSA. Compared with FedPCA and WDA, FedSVD consistently has more than 10 orders of magnitude lower projection distance on PCA and LSA applications. On LR application, FedSVD has the lowest MSE compared to FATE and SecureML which solves LR using SGD. Moreover, FedSVD only needs to factorize the data once to find the optimal solution, while SGD-based method usually needs multiple epochs of training to converge.

Efficiency: Fig. 6(a) shows the LR time consumption of FedSVD, FATE and SecureML when we fix $n = 1$K and vary m from 1M to 50M, and the results show that FedSVD is 100x faster than SecureML and 10x faster than FATE. Fig. 6(b) and Fig. 6(c) show the time consumption of LR under different network bandwidth and latency, the results show that FedSVD is less sensitive to network compared with SecureML, and FedSVD achieves consistently best performance under different network conditions. We have performed billion-scale data evaluation on all the applications and the results are reported in Tab. 2. The results show that FedSVD is practical and successfully supports billion-scale applications.

Table 2: Evaluate Applications on Billion-Scale Data. The data is uniformly partitioned on 2 users and network bandwidth=1Gb/s, RTT=50ms.

Application	Datasets	Data Size	Time
PCA (top-r = 5)	Synthetic Data	100K × 1M (100 Billion Elements)	32.3 Hours
LSA (top-r = 256)	MovieLens-25M (RealWorld)	62K × 162k (10 Billion Elements)	3.71 Hours
LR	Synthetic Data	1K × 50M (50 Billion Elements)	13.5 Hours

5.4 Attacks

We have provided privacy analysis of FedSVD in §3.5 showing that CSP cannot recover the raw data from the masked data when having no prior knowledge. In this section, we assume the CSP empirically choose data distributions as prior knowledge and perform independent component analysis (ICA) attacks [15] on the masked data. Meanwhile, we set block size to different values and observe its impact on the effectiveness of privacy protection.

The ICA attack is the SOTA attack method on masked data proposed by Li et al. [15] for revealing raw data from masked databases. The main idea is to treat the masked data as a linear combination of different data sources, which are assumed to be independent and non-gaussian distributed. The attackers empirically choose distributions of the data sources (*e.g.*, using sigmoid as the cumulative probability distribution function), and try to find the inverse of the linear combination that maximizes the likelihood function.

Table 3: ICA attacks on the masked data. Pearson correlation between the attack results and raw data are reported.

Attacks	b	MNIST	ML-100K	Wine
Random Values	NA	0.12590	0.17957	0.49313
ICA	10	0.20329	0.18623	0.44268
ICA(b)	10	0.32029	0.28434	0.45971
ICA	100	0.12590	0.18387	0.45183
ICA(b)	100	0.13051	0.20910	0.45826
ICA	1000	0.11104	0.18020	0.44712
ICA(b)	1000	0.12531	0.18057	0.44862

In our experiments, we run ICA attack on both side of the masked data since FedSVD has two masks, and Tab. 3 shows the results. Meanwhile, we also perform attacks assuming the CSP knows the block size b, denoted as ICA(b) in Tab. 3, which reduces the number of parameters to solve in the attack. We use Pearson correlation to assess the attack results. Since ICA has disordered outputs (*i.e.*, recovered data might be shuffled by row or by column), we compute n-to-n matching Pearson correlation between the attack results and real data, and report the maximum value. We use random value as the baseline, and if the Pearson correlation between the attack results and the raw data is close to the Pearson correlation between random value and raw data, then we can conclude that the attack fails in recovering valid information. We can observe from Tab. 3

that 1) ICA(b) is more effective than ICA, which means that knowing b is helpful to the attacks; 2) When increasing b from 10 to 1000, attacking effectiveness of both ICA and ICA(b) decrease; 3) When setting $b = 1000$, all the attacks fail in recovering valid information.

In conclusion, 1) The Pearson correlation between the attack results and raw data decreases with the increase of block size, when the block size is large enough (*e.g.*, 1000 in our experiments), the ICA attack fails in recovering valid information; 2) Leaking the block size reduces the complexity of ICA attack, however, the attack still could be defensed as long as the block size is large enough.

In the application, since different datasets have various distributions, we suggest the users run local ICA attacks and choose a proper block size that can resist the attack.

5.5 Effectiveness of Proposed Optimizations

In this section, we compare the efficiency with and without the proposed optimizations to show the effectiveness of our design.

We categorize three types of optimizations from our system: 1) Opt1: the block-based optimizations including efficient mask initialization, data masking, and recovery of $\mathbf{V'^T}$; 2) Opt2: mini-batch secure aggregation; 3) Opt3: advanced disk offloading. Fig. 7 shows the evaluation results using 10K × 50K synthetic data. Compared with using no optimizations, our solution reduces the communication, time consumption, and memory usage by 73.2%, 81.9%, and 95.6%, respectively. To further demonstrate the effectiveness of Opt3, we compare the efficiency of RAM+AdvancedOffLoading and RAM+SwapOffLoading on larger data (10K × 100K), the results show that our solution reduces the time consumption by 44.7% compared with swap disk offloading scheduled by OS.

6 Related Work

Apart from the federated SVD methods introduced in §1, there are also other research topics that closely related to our work:

Privacy-preserving Funk-SVD: The Funk-SVD is utilized in the federated recommender system [25]. The major difference between Funk-SVD and SVD is that Funk-SVD runs on the sparse rating matrix. Chai et al. [4] solved the federated Funk-SVD problem using HE. Berlioz et al. [1] proposed a DP-based Funk-SVD method.

Outsourcing matrix factorization techniques: The secure outsourcing computation is a traditional research topic. Zhang et al. [27] proposed a secure outsourcing computation framework for PCA-based face recognition. Duan et al. [7] proposed outsourcing computation frameworks for non-negative matrix factorization. Luo et al. [18] proposed a masking based outsourcing computation method for QR and LU factorization.

7 Conclusion

In this paper, we propose a practical lossless federated SVD method over billion-scale data. Compared with the existing federated SVD methods, FedSVD is lossless and efficient. The experiments show that FedSVD is over 10000× faster than HE-based method and has 10 orders of magnitude smaller error compared DP-based method.

Acknowledgments

The work is supported by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001), the NSFC Grant no. 61972008, the Hong Kong RGC TRS T41-603/20R, the National Key Research and Development Program of China under Grant No.2018AAA0101100, and the Turing AI Computing Cloud (TACC) [24].

References

[1] Arnaud Berlioz, Arik Friedman, Mohamed Ali Kâafar, Roksana Boreli, and Shlomo Berkovsky. 2015. Applying Differential Privacy to Matrix Factorization. In *RecSys*. ACM, 107–114.
[2] Aditya Bhaskara and Maheshakya Wijewardena. 2019. On Distributed Averaging for Stochastic k-PCA. In *NeurIPS*. 11024–11033.
[3] Kallista A. 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 *CCS*. ACM, 1175–1191.
[4] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. 2021. Secure Federated Matrix Factorization. *IEEE Intell. Syst.* 36, 5 (2021), 11–20.
[5] James W Daniel, Walter Bill Gragg, Linda Kaufman, and Gilbert W Stewart. 1976. Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Math. Comp.* 30, 136 (1976), 772–795.
[6] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository.
[7] Jia Duan, Jiantao Zhou, and Yuanman Li. 2021. Secure and Verifiable Outsourcing of Large-Scale Nonnegative Matrix Factorization (NMF). *IEEE Trans. Serv. Comput.* 14, 6 (2021), 1940–1953.
[8] Susan T. Dumais. 2004. Latent semantic analysis. *Annu. Rev. Inf. Sci. Technol.* 38, 1 (2004), 188–230.
[9] Andreas Grammenos, Rodrigo Mendoza-Smith, Jon Crowcroft, and Cecilia Mascolo. 2020. Federated Principal Component Analysis. In *NeurIPS*.
[10] Arjun K Gupta and Daya K Nagar. 2018. *Matrix variate distributions*. Vol. 104. CRC Press.
[11] Shuguo Han, Wee Keong Ng, and Philip S. Yu. 2009. Privacy-Preserving Singular Value Decomposition. In *ICDE*. IEEE Computer Society, 1267–1270.
[12] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (2016), 19:1–19:19.
[13] Fatma Latifoglu, Kemal Polat, Sadik Kara, and Salih Günes. 2008. Medical diagnosis of atherosclerosis from Carotid Artery Doppler Signals using principal component analysis (PCA), k-NN based weighting pre-processing and Artificial Immune Recognition System (AIRS). *J. Biomed. Informatics* 41, 1 (2008), 15–23.
[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[15] Rui Li, Alex X. Liu, Ying Liu, Huanle Xu, and Huaqiang Yuan. 2019. Insecurity and Hardness of Nearest Neighbor Queries Over Encrypted Data. In *ICDE*. IEEE, 1614–1617.
[16] Bowen Liu and Qiang Tang. 2019. Privacy-Preserving Decentralised Singular Value Decomposition. In *ICICS (Lecture Notes in Computer Science, Vol. 11999)*. Springer, 703–721.
[17] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. 2021. FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection. *J. Mach. Learn. Res.* 22 (2021), 226:1–226:6.
[18] Changqing Luo, Kaijin Zhang, Sergio Salinas, and Pan Li. 2021. SecFact: Secure Large-scale QR and LU Factorizations. *IEEE Trans. Big Data* 7, 4 (2021), 796–807.
[19] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 19–38.
[20] Alkes L Price, Nick J Patterson, Robert M Plenge, Michael E Weinblatt, Nancy A Shadick, and David Reich. 2006. Principal components analysis corrects for stratification in genome-wide association studies. *Nature genetics* 38, 8 (2006), 904–909.
[21] Parinya Sanguansat. 2012. *Principal Component Analysis: Engineering Applications*. BoD–Books on Demand.
[22] H. Tian, C. Zeng, Z. Ren, D. Chai, J. Zhang, K. Chen, and Q. Yang. 2022. Sphinx: Enabling Privacy-Preserving Online Learning over the Cloud. In *2022 2022 IEEE Symposium on Security and Privacy (SP)*. 1135–1149.
[23] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–5555.
[24] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. 2021. TACC: A Full-stack Cloud Computing Infrastructure for Machine Learning Tasks. *CoRR* abs/2110.01556 (2021).
[25] Liu Yang, Ben Tan, Vincent W. Zheng, Kai Chen, and Qiang Yang. 2020. Federated Recommendation Systems. In *Federated Learning*. Lecture Notes in Computer Science, Vol. 12500. Springer, 225–239.
[26] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* 10, 2 (2019), 12:1–12:19.
[27] Yushu Zhang, Xiangli Xiao, Lu-Xing Yang, Yong Xiang, and Sheng Zhong. 2020. Secure and Efficient Outsourcing of PCA-Based Face Recognition. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 1683–1695.

A Datasets and Baseline Models

Datasets: We have used five datasets in the experiments. Following is the detailed description and the parameter settings:

- MNIST [14]: A standard hand-written digits image testset, and each image contains 784 (i.e., 28×28) features. We take 10K labeled images in the experiment, thus $X_{mnist} \in \mathbb{R}^{784 \times 10K}$.
- Wine [6]: The physicochemical data for 6498 variants of red and white wine, and each sample has 12 features. Thus $X_{wine} \in \mathbb{R}^{12 \times 6489}$.
- movielens [12]: Movielens dataset describes people's expressed preferences for movies. It contains contains millions of users' rating records over different movies. We select two groups of movielens data: movielens-100K and movielens-25M for our experiment. Movielens-100k contians 943 users' rating on 1682 movies, thus $X \in \mathbb{R}^{1682 \times 943}$. Movielens-25M contians 162542 users' rating on 59047 movies, thus $X \in \mathbb{R}^{59047 \times 162542}$.
- Synthetic data [9]: Apart from the real-world datasets, we also use synthetic data in the evaluation. The synthetic data is generated from a power-law spectrum $Y_\alpha \sim Synth(\alpha)^{m \times n}$ using $\alpha = 0.01$. More specifically, $Y = U\Sigma V^T$, where $[U, \sim] = QR(N^{m \times m})$, $[V, \sim] = QR(N^{m \times n})$, $\Sigma_{i,i} = i^{-\alpha}$, and $N^{m \times n}$ is an matrix with i.i.d entries drawn from $\mathcal{N}(0, 1)$.

Baseline Models: We compare FedSVD with three existing works, and following is the detailed introduction and parameter setting.

- WDA-PCA [2]: In the weighted distributed averaging PCA (WDA-PCA), the participants upload local rank-k approximation of the covariance matrix to the server, which will aggregate all the approximations through weighted average and do a rank-k PCA on the aggregated matrix to get the final results. WDA-PCA reduces the private data leakage since each users only uploads a rank-k approximation of the covariance matrix. In our experiments, we only compared FedSVD and WDA-PCA in PCA applications, since WDA-PCA is specially designed for rank-k PCA and not suitable for SVD tasks.
- FedPCA [9]: Federated principal component (FedPCA) analysis is a federated, asynchronous, and (ϵ, δ)-differentially private algorithm. Follow the setting in [9], we set $\epsilon = 0.1, \delta = 0.1$. We compare FedSVD and FedPCA in both PCA and SVD tasks.
- PPD-SVD [16]: Privacy-preserving decentralized SVD (PPD-SVD) used homomorphic encryption to protect user's private data during the joint computation of covariance matrix, then decrypt the covariance matrix and do regular SVD tasks. According to the original paper's setting, we set the key size of HE to 1024.

Hardware: All the experiments are performed on a Ubuntu 20.04 Server with a 3.6GHz 8-core CPU, 128GB RAM, and 2TB SSD. The programming language is Python. For all the experiments, we put participants into different Docker containers, which are connected using the docker-bridge network, and we simulate the network bandwidth and latency between containers.