Privacy-preserving Federated Singular Value Decomposition

Bowen Liu and Qiang Tang

Luxembourg Institute of Science and Technology (LIST),
5, Avenue des Hauts-Fourneaux, L-4362, Esch-sur-Alzette, Luxembourg
{bowen.liu, qiang.tang}@list.lu

Abstract. Modern SVD computation dates back to work in the 1960s that proposed the basis for the eigensystem package and linear algebra package routines [8, 9]. As a result of a long history of research, SVD is now widely applied in various scenarios, such as recommendation system and principal component analysis. Furthermore, federated SVD has emerged as a prevalent privacy-preserving technique. For example, the raw data are not required to be exchanged among different parties; instead, each party trains and processes locally and shares intermediate result. In general, there are two main categories: SVD over horizontally and vertically partitioned data [22]. Imagine a dataset matrix $\mathbb{M}$, where each row stands for a record from a data subject, and the columns stand for the attributes/features of the records. In the horizontally partitioned setting, each party holds a disjoint subset of the rows of $\mathbb{M}$. While in the vertically partitioned setting, each party has a disjoint subset of the columns of $\mathbb{M}$ for all the rows. In real-world applications, the horizontally partitioned setting is much more common than the vertically partitioned setting [12, 13]. In this paper, we have proposed a privacy-preserving federated SVD scheme with secure aggregation. The proposed scheme can aggregate SVD results (eigenspace) from different devices and synchronise the aggregation result with all devices while maintaining privacy protection.
Therefore, we have proposed a privacy-preserving federated SVD scheme with secure aggregation on the former setting.

Keywords: Singular Value Decomposition · Federated Learning · Distributed Computation.

1 Introduction

Advances in networking and hardware technology have made the design and deployment of Internet of Things (IoTs) and decentralised applications a trend. For example, the FoG computing concept and its associated edge computing technologies are pushing computations to the edge so that data aggregation can be avoided to some extent. This naturally brings benefits such as efficiency and privacy, but on the other hand it forces data analysis tasks to be carried out in a distributed manner (due to the fact that data will not be aggregated). To this end, federated learning has become a promising solution direction: raw data are not required to be exchanged among different parties; instead, each party trains and processes locally and shares intermediate results [23].

Among many data analysis methods, Singular Value Decomposition (SVD) is a very interesting one with wide applications. Modern SVD computation can be dated back to the work in the 1960s that proposed the basis for eigensystem package and linear algebra package routines [8, 9]. As a result of a long history of research, SVD is now applied in a wide variety of scenarios, such as recommendation system [16, 25], Principal Component Analysis (PCA) [21], Latent Semantic Analysis (LSA) [6], noise filtering [11, 17], dimension reduction [18], clustering [20], matrix completion [4], etc. As a matter of fact, federated SVD has emerged as a prevalent privacy-preserving technique. Existing solutions fall into two main categories: SVD over horizontally and vertically partitioned dataset [22]. Imagine a dataset $\mathbb{M}$, where each row stands a record from a data subject and the columns stand for the attributes/features of the records. In the horizontally partitioned setting, each party (in our case, an edge device) holds a disjoint subset of the rows of $\mathbb{M}$. While in the vertically partitioned setting, each party holds a disjoint subset of the columns of $\mathbb{M}$ for all the rows.

In real-world applications, the horizontally partitioned setting is much more common than the vertically partitioned setting [12, 13]. Therefore, we focus on the former in this paper.

1.1 Related Work

In the literature, we notice two federated SVD solutions which have explicitly provided privacy analysis. Hartebrodt et al. proposed a federated SVD algorithm for high-dimensional data [13]. It is designed for a star-like architecture where the aggregator does not have access to the complete eigenvector matrix of SVD results, and each edge device has access only to its share part of the eigenvector matrix. However, it does not discuss any additional privacy protection. Guo et al. presented a federated privacy-preserving SVD algorithm based on the distributed power method [12]. It is assumed that each edge device holds its own set of data records, and they want to perform a SVD based on their joint dataset (denoted as a matrix $\mathbb{M}$). With their solution, each edge device performs a (partial) SVD on the matrix multiplication $\mathbb{M}^T\mathbb{M}$ and the eigenvectors are sent to the server with added Gaussian noise. Then, the server aggregates all received eigenvectors and sends the result back to each edge device with additional Gaussian noise. We will provide a detailed description and analysis in the next section.

1.2 Contribution and Organisation

We analyse the fully participation protocol of federated privacy-preserving SVD algorithm from [12]. We find that it has the potential to be improved, thus increasing the accuracy and utility of the final results. We therefore propose a privacy-preserving federated SVD scheme with secure aggregation.

The rest of the paper is organised as follows. In Section 2, we list the fundamental definitions of the relevant techniques. In Section 3, we recap the scheme proposed by Guo et al. [12] and analyse its privacy protection. In Section 4, in addition to a preliminary on secure aggregation, we present our enhanced solution with analysis of its computational complexity, privacy and accuracy. In Section 5, we conclude the paper.

2 Preliminary

Let $\mathbb{M}$ be a $m \times n$ matrix. As shown in Figure 1, the Singular Value Decomposition (SVD) of $\mathbb{M}$ is a factorisation of the form $\mathbb{U}\Sigma\mathbb{V}^T$, where $\mathbb{U}$ is an $m \times m$ left-singular matrix of $\mathbb{M}$, Σ is an $m \times n$ singular matrix of $\mathbb{M}$, $\mathbb{V}$ is a $n \times n$ right-singular matrix of $\mathbb{M}$, and T means conjugate transpose. In addition, there are also two relations:

$$\mathbb{M} \cdot \mathbb{M}^T = \mathbb{U}\Sigma\mathbb{V}^T \cdot \mathbb{V}\Sigma^T\mathbb{U}^T = \mathbb{U}\Sigma\Sigma^T\mathbb{U}^T$$
$$\mathbb{M}^T \cdot \mathbb{M} = \mathbb{V}\Sigma^T\mathbb{U}^T \cdot \mathbb{U}\Sigma\mathbb{V}^T = \mathbb{V}\Sigma^T\Sigma\mathbb{V}^T$$

The columns of $\mathbb{U}$ (left-singular vectors) and $\mathbb{V}$ (right-singular vectors) are, respectively, eigenvectors of $\mathbb{M} \cdot \mathbb{M}^T$ and $\mathbb{M}^T \cdot \mathbb{M}$. The non-zero elements of Σ are the square roots of the non-zero eigenvalues of $\mathbb{M} \cdot \mathbb{M}^T$ and $\mathbb{M}^T \cdot \mathbb{M}$.

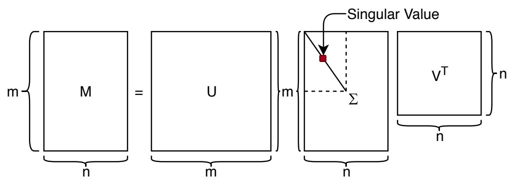

Fig. 1. Singular Value Decomposition

2.1 Singular Value Decomposition with the Power Method

The power method [10] can be used to find the eigenvalues, if one exists, and the corresponding eigenvectors, especially for large size matrix. Let $\mathbb{M}$ be a $m \times n$ matrix, where $m > n$ and its full SVD can be calculated as $\mathbb{M} = \mathbb{U}\Sigma\mathbb{V}^T$, where $\Sigma = (\lambda_1, \cdots, \lambda_k, \cdots, \lambda_n)$ and $\mathbb{V} = (v_1, \cdots, v_k, \cdots, v_n)$. Let $\mathbb{M}' = \mathbb{M}^T\mathbb{M}$ be the power method computes the top k eigenvectors of $\mathbb{M}'$ by iterating

$$\mathbb{Y} = \mathbb{M}'\mathbb{Z} \text{ and } \mathbb{Z} = orth(\mathbb{Y}),$$

where both $\mathbb{Y}, \mathbb{Z}$ are $n \times k$ matrices and $orth(\mathbb{Y})$ stands for orthogonality the columns of $\mathbb{Y}$ with QR factorisation [10].

2.2 Distributed Power Method

Suppose $\mathbb{M}$ is an $m \times n$ matrix, where $m > n$, and is horizontally partitioned into d blocks $\mathbb{M}^T = [\mathbb{M}_1^T, \mathbb{M}_2^T, \dots, \mathbb{M}_d^T]$. Each partitioned matrix $\mathbb{M}_i$ consists of s_i individual vectors of $\mathbb{M}$ and $\sum_{i=1}^d s_i = m$. Let us denote $\mathbb{M}_i' = \frac{1}{s_i}\mathbb{M}_i^T\mathbb{M}_i$, thus,

$$\mathbb{M}' = \frac{1}{m}\mathbb{M}^T\mathbb{M} = \sum_{i=1}^{d} \frac{1}{m}\mathbb{M}_i^T\mathbb{M}_i = \sum_{i=1}^{d} \frac{s_i}{m}\mathbb{M}_i' = \sum_{i=1}^{d} p_i\mathbb{M}_i',$$

where $p_i = \frac{s_i}{m}$. Thereby, $\mathbb{Y}$ in Section 2.1 can be written as

$$\mathbb{Y} = \sum_{i=1}^{d} \frac{s_i}{m}\mathbb{M}_i'\mathbb{Z} = \sum_{i=1}^{d} p_i\mathbb{M}_i'\mathbb{Z}.$$

which indicates that the power method can be distributed to each individual edge device, which holds $\mathbb{M}_i$.

2.3 Differential Privacy

The privacy notion of differential privacy was introduced by Dwork et al. [7], which ensures that the addition, removal, or modification of a single data item does not substantially affect the outcome of the data-based analysis. Typically, differential privacy is enforced by injecting calibrated noise (e.g., Gaussian noise) into the intermediate/final results. It is formally defined as follows.

Definition 1 ((ε, δ)-Differential Privacy). *A randomised mechanism $\mathcal{M} : \mathcal{X} \rightarrow \mathcal{R}$ with domain $\mathcal{X}$ and range $\mathcal{R}$ satisfies (ε, δ)-differential privacy if for any two adjacent inputs $x, x' \in \mathcal{X}$ and for any subset of output $S \subseteq \mathcal{R}$ it holds that*

$$\Pr \mathcal{M}(x) \in S \leq exp(\varepsilon) \Pr \mathcal{M}(x') \in S + \delta.$$

The privacy budget ε is a small constant used to measure privacy loss and maintain the trade-off between privacy and utility/accuracy. The δ is other small constant representing the probability of information being leaked accidently.

3 Analysis of an Existing Federated SVD Solution

3.1 Recap of the Fully Participation Protocol by Guo et al.

Assume there are d edge devices, and each device i holds an independent dataset, a $s_i \times n$ matrix $\mathbb{M}_i$ and stores a multiplication $\mathbb{M}_i' = \frac{1}{s_i}\mathbb{M}_i^T\mathbb{M}_i$. Besides, let $\mathbb{M}$ denote the ingratiation matrix of joining all $\mathbb{M}_i$, where $\mathbb{M}^T = [\mathbb{M}_1^T, \mathbb{M}_2^T, \dots, \mathbb{M}_d^T]$. The solution by Guo et al. [12] is summarised in Figure 2 and is detailed as follows.

In the initialisation phase, the server needs to generate the following parameters:

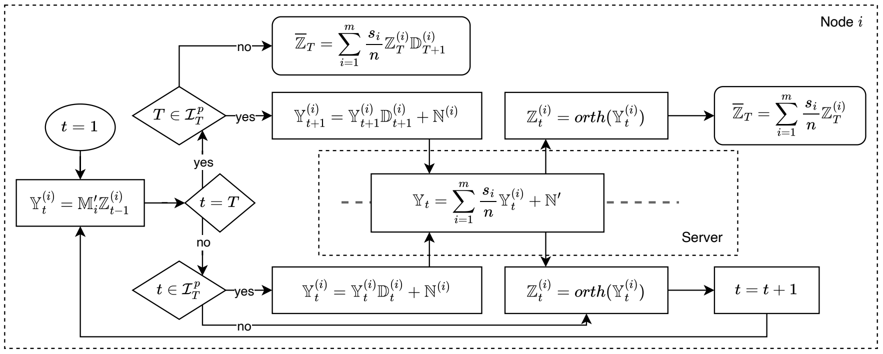
Node i
$t = 1$
$\mathbb{Y}_t^{(i)} = \mathbb{M}_i'\mathbb{Z}_{t-1}^{(i)}$
$t = T$
yes
no
$T \in \mathcal{I}_T^p$
$t \in \mathcal{I}_T^p$
$\overline{\mathbb{Z}}_T = \sum_{i=1}^{m} \frac{s_i}{n}\mathbb{Z}_T^{(i)}\mathbb{D}_{T+1}^{(i)}$
$\mathbb{Y}_{t+1}^{(i)} = \mathbb{Y}_{t+1}^{(i)}\mathbb{D}_{t+1}^{(i)} + \mathbb{N}^{(i)}$
$\mathbb{Y}_t^{(i)} = \mathbb{Y}_t^{(i)}\mathbb{D}_t^{(i)} + \mathbb{N}^{(i)}$
$\mathbb{Y}_t = \sum_{i=1}^{m} \frac{s_i}{n}\mathbb{Y}_t^{(i)} + \mathbb{N}'$
Server
$\mathbb{Z}_t^{(i)} = orth(\mathbb{Y}_t^{(i)})$
$\overline{\mathbb{Z}}_T = \sum_{i=1}^{m} \frac{s_i}{n}\mathbb{Z}_T^{(i)}$
$t = t + 1$

Fig. 2. Illustration of the Fully Participation Protocol by Guo et al.

- T: the number of local computations performed by each edge device.
- $\mathcal{I}_T^p$: the communications between the edge devices and server only perform every p interactions, where $\mathcal{I}_T^p$ is an index subset that $\mathcal{I}_T^p = \{t \in [T] : t \bmod p = 0\} = \{0, p, 2p, \ldots, p\lfloor T/p \rfloor\}$.
- (σ, σ'): the variance of noises.
- $\mathbb{Z}_0$: the initial approximation of one of the dominant eigenvectors of $\mathbb{M}$.

At the end of the initialisation, the parameters $(T, \mathcal{I}_T^p, \sigma, \sigma', \mathbb{Z}_0)$ are shared with all edge devices. And during the privacy-preserving SVD protocol, each edge device and the server perform as follows:

```
1: for $t = 1$ to $T$ do
2:     each edge device $i$ computes $\mathbb{Y}_t^{(i)} = \mathbb{M}_i'\mathbb{Z}_{t-1}^{(i)}$
3:     if $t \in \mathcal{I}_T^p$ then
4:         each edge device $i$ adds the Gaussian noise $\mathbb{N}^{(i)}$, thus $\mathbb{Y}_t^{(i)} = \mathbb{Y}_t^{(i)}\mathbb{D}_t^{(i)} + \mathbb{N}^{(i)}$, where $\mathbb{Y}_t^{(i)}\mathbb{D}_t^{(i)}$ is the orthogonal transformation of $\mathbb{Y}_t^{(i)}$, and sends it to the server
5:         the server performs perturbed aggregations, and adds another Gaussian noise $\mathbb{N}'$ into the result, thus, $\mathbb{Y}_t = \sum_{i=1}^{d} \frac{s_i}{m}\mathbb{Y}_t^{(i)} + \mathbb{N}'$
6:         the server broadcasts $\mathbb{Y}_t$ to each edge device $i$, and let $\mathbb{Y}_t^{(i)} = \mathbb{Y}_t$ on each edge device $i$
7:     end if
8:     each edge device $i$ performs orthogonalisation $\mathbb{Z}_t^{(i)} = orth(\mathbb{Y}_t^{(i)})$
9: end for
10: the final result is the approximated eigenspace:
```

$$\overline{\mathbb{Z}}_T \begin{cases} \sum_{i=1}^{d} \frac{s_i}{m}\mathbb{Z}_T^{(i)}\mathbb{D}_{T+1}^{(i)} & \text{if } T \notin \mathcal{I}_T^p \\ \sum_{i=1}^{d} \frac{s_i}{m}\mathbb{Z}_T^{(i)} & \text{otherwise.} \end{cases}$$

Briefly, in the proposed solution, each individual edge device holds its own raw data and processes the SVD locally, its eigenvectors are aggregated on the server by Orthogonal Procrustes Transformation (OPT) mechanism, and the aggregation result is sent back for further iterations. More details (e.g. the computation of $\mathbb{D}_t^{(i)}$ and Gaussian noise) are given in [12].

3.2 Analysis of the Fully Participation Protocol by Guo et al.

The privacy protection of [12] is achieved in terms of local differential privacy by adding Gaussian noise on each edge device in Step 4 when synchronisation occurred, and central differential privacy by adding Gaussian noise on the server in Step 5.

From the perspective of privacy protection, adding Gaussian noise provides the indistinguishability between the true eigenspace and the computation result in each iteration round. It is not a surprise that the added Gaussian noise affects the accuracy of the final result, and it is usually accepted as a tradeoff between privacy and utility/accuracy. However, we consider that the noise from the server to be unnecessary because an edge device's input has already been masked when local differential privacy is applied. On the other hand, it is well known that local differential privacy may seriously downgrade the utility/accuracy of the final result. To this end, central differential privacy does have an advantage. combining these two observation, it is preferably to apply the central privacy concept in such a manner so that it protects every edge device's privacy against both curious individuals and the server. This is what we will do in the enhanced solution in the next section.

During each iteration, every edge device needs to perform orthogonalisation $\mathbb{Z}_t^{(i)} = orth(\mathbb{Y}_t^{(i)})$. This is very inefficient, particularly we would like to reduce the computational complexity of edge devices. Trivially, the server can do this once and share the result with all edge devices. Clearly, this does not impact privacy at all.

4 Enhanced Privacy-preserving Federated SVD Solution

4.1 Preliminary on Secure Aggregation

In federated machine learning literature, the secure aggregation protocol of Bonawitz et al. [3] has been widely used by many solutions. Next, we recap this protocol and then use it in benchmarking our enhanced SVD solution.

In simple terms, with the secure aggregation protocol, the original data of each edge device are locally masked in a particular way and shared to the server, when the masked data are aggregated on the server, in the meanwhile, the masks are cancelled and offset.

- The following algorithms are defined, and parameters are generated during the setup phase and sent to relevant edge devices.
  - Pseudorandom Generator (PRG) [2, 24]: PRG which takes a fixed length seed as input and outputs in space $[0, R)$, where R is a prefixed value.

- Secret Sharing [19]: $\mathsf{SS.share}(s, t, \mathcal{U}) \to \{(u, s_u)\}_{u\in\mathcal{U}}$, it takes a secret s, a set of user IDs (e.g. integers), a threshold $s \leq |\mathcal{U}|$ as input, and outputs a set of shares s_u associated with the user $u \in \mathcal{U}$; and a reconstruction algorithm $\mathsf{SS.recon}(\{(u, s_u)\}_{v\in\mathcal{V}}, t) \to s$ takes the following values as input: threshold t and shares corresponding to a user subset $\mathcal{V} \subseteq \mathcal{U}$ such that $|\mathcal{V}| \geq t$, and outputs a field element s.
- Key Agreement [5]: $\mathsf{KA.param}(k) \to pp$ takes a security parameter k and returns some public parameters; $\mathsf{KA.gen}(pp) \to (s^{SK}, s^{PK})$ generates a secret/public key pair; $\mathsf{KA.agree}(s_u^{SK}, s_v^{PK}) \to s_{u,v}$ allows a user u to combine its private key with the public key of another user v into a private shared key between them.
- Authenticated Encryption [14]: $\mathsf{AE.enc}$ and $\mathsf{AE.dec}$ are algorithms for encrypting a plaintext with a public key and for decrypting a ciphertext with a secret key.
- Signature Scheme [1]: $\mathsf{SIG.gen}$ takes a security parameter k and outputs a secret/public key pair; $\mathsf{SIG.sign}$ signs a message with a secret key and returns the relevant signature; $\mathsf{SIG.ver}$ verifies the signature of the relevant message and returns a boolean bit indicating whether the signature is valid.

- Number of edge devices m.
- Security parameter k.
- Public parameter of key agreement $pp \leftarrow \mathsf{KA.param}(k)$.
- Threshold value t, where $t < n$ and n is the number of edge devices.
- Input space $\mathbb{Z}_R$.
- Secrets sharing field $\mathbb{F}$.
- Signature key pairs (d_u^{SK}, d_u^{PK}) of each edge device, where $u \in [1, m]$.

– Round 0 (AdvertiseKeys):

0.1. each edge device u generates secret/public key pairs of encryption and sharing algorithm (c_u^{SK}, c_u^{PK}) and (s_u^{SK}, s_u^{PK})
0.2. each edge device u signs c_u^{PK} and s_u^{PK} into $\sigma_u \leftarrow \mathsf{SIG.sign}(d_u^{SK}, c_u^{PK}||s_u^{PK})$
0.3. the two public keys and all n signatures $(c_u^{PK}||s_u^{PK}||\sigma_u)$ are sent to the server
0.4. if the server receives at least t messages from individual edge devices (denote by $\mathcal{U}_1$ this set of edge devices), then broadcasts $\{(v, c_v^{PK}, s_v^{PK}, \sigma_v)\}_{v\in\mathcal{U}_1}$ to all edge devices in $\mathcal{U}_1$

– Round 1 (ShareKeys):

1.1. once an edge device u in $\mathcal{U}_1$ receives the messages from the server, it verifies if all signatures are valid with $\mathsf{SIG.ver}(d_{u'}^{PK}, c_{u'}^{PK}||s_{u'}^{PK}, \sigma_{u'})$, where $u' \in \mathcal{U}_1$
1.2. the edge device u sample a random element $b_u \leftarrow \mathbb{F}$ as a seed for a PRG
1.3. the edge device u generates two t-out-of-$|\mathcal{U}_1|$ shares of s_u^{SK} : $\{(v, s_{u,v}^{SK})\}_{v\in\mathcal{U}_1} \leftarrow \mathsf{SS.share}(s_u^{SK}, t, \mathcal{U}_1)$ and b_u : $\{(v, b_{u,v})\}_{v\in\mathcal{U}_1} \leftarrow \mathsf{SS.share}(b_u, t, \mathcal{U}_1)$

1.4. for each edge device $v \in \mathcal{U}_1 \setminus \{u\}$, u computes $e_{u,v} \leftarrow \mathsf{AE.enc}(\mathsf{KA.agree}(c_u^{SK}, c_v^{PK}), u||v||s_{u,v}^{SK}||b_{u,v})$ and sends them to the server
1.5. if the server receives at least t messages from individual edge devices (denoted by $\mathcal{U}_2 \subseteq \mathcal{U}_1$ this set of edge devices), then it shares to each edge device $v' \in \mathcal{U}_2$ all ciphertexts for it $\{e_{u',v'}\}_{u' \in \mathcal{U}_2}$

– Round 2 (MaskedInputCollection):

2.1. for the edge device $u \in \mathcal{U}_2$, once the ciphertexts are received, it computes $s_{u,v} \leftarrow \mathsf{KA.agree}(s_u^{SK}, s_v^{PK}$, where $v \in \mathcal{U}_2 \subseteq \{u\}$
2.2. $s_{u,v}$ is expanded using PRG into a random vector $p_{u,v} = \Delta_{u,v} \cdot \mathsf{PRG}(s_{u,v})$, where $\Delta_{u,v} = 1$ when $u > v$ and $\Delta_{u,v} = -1$ when $u < v$, besides, define $p_{u,u} = 0$
2.3. the edge device u computes its own private mask vector $p_u = \mathsf{PRG}(b_u)$ and the masked input vector x_u into $y_u \leftarrow x_u + p_u + \sum_{v \in \mathcal{U}_2} p_{u,v} \pmod{R}$, then y_u is sent to the server
2.4. if the server receives at least t messages (denote with $\mathcal{U}_3 \subseteq \mathcal{U}_2$ this set of edge devices), and share the edge device set $\mathcal{U}_3$ with all edge devices in $\mathcal{U}_3$

– Round 3 (ConsistencyCheck):

3.1. once the edge device $u \in \mathcal{U}_3$ receives the message, it returns the signature $\sigma'_u \leftarrow \mathsf{SIG.sign}(d_u^{SK}, \mathcal{U}_3)$
3.2. if the server receives at least t messages (denoted by $\mathcal{U}_4 \subseteq \mathcal{U}_3$ this set of edge devices) and shares the set $\{u', \sigma'_{u'}\}_{u' \in \mathcal{U}_4}$

– Round 4 (Unmasking):

4.1. each edge device u verifies $\mathsf{SIG.ver}(d_v^{PK}, \mathcal{U}_3, \sigma'_v)$ for all $v \in \mathcal{U}_4$
4.2. for each edge device $v \in \mathcal{U}_2 \setminus \{u\}$, u decrypts the ciphertext (received in the MaskedInputCollection round) $v'||u'||s_{v',u'}||b_{v',u'} \leftarrow \mathsf{AE.dec}(\mathsf{KA.agree}(c_{u^\times}^{SK}, c_v^{PK}), e_{v,u})$ and asserts that $u' = u \wedge v' = v$
4.3. each edge device u sends the shares $s_{v,u}^{SK}$ for edge devices $v \in \mathcal{U}_2 \setminus \mathcal{U}_3$ and $b_{v,u}$ for edge devices in $v \in \mathcal{U}_3$ to the server
4.4. if the server receives at least t messages (denote with $\mathcal{U}_5$ this set of edge devices), it re-constructs, for each edge device $u \in \mathcal{U}_2 \setminus \mathcal{U}_3$, $s_u^{SK} \leftarrow \mathsf{SS.recon}(\{s_{u,v}^{SK}\}_{v \in \mathcal{U}_5}, t)$ and re-computes $p_{v,u}$ using PRG for all $v \in \mathcal{U}_3$
4.5. the server also re-constructs, for all edge devices $u \in \mathcal{U}_3$, $b_u \leftarrow \mathsf{SS.recon}(\{b_{u,v}\}_{v \in \mathcal{U}_5}, t)$ and re-computes $p_{v,u}$ using the PRG
4.6. finally, the server outputs $z = \sum_{u \in \mathcal{U}_3} x_u = \sum_{u \in \mathcal{U}_3} y_u - \sum_{u \in \mathcal{U}_3} p_u + \sum_{u \in \mathcal{U}_3, v \in \mathcal{U}_2 \setminus \mathcal{U}_3} p_{v,u}$

We summarise the asymptotic computational complexity of each edge device and the server in Table 1. For simplicity of description, we assume that all devices participate in the protocol, that is, $t = m$. Since some operations can be considered as offline pre-configuration, we focus on online operations starting from masking messages in Step 2.3.

	Vector Add	SIG.sign	SIG.vef	KA.agree	AE.dec	SS.recon	PRG
Edge	$m+1$	1	$m-1$	$m-1$	$m-1$		1
Server	$2m-1$					m	m

Table 1. Asymptotic Computational Complexity of Online Operations

4.2 Our Enhanced SVD Solution

As mentioned in Section 3.2, in the context of achieving the privacy-preserving objective, we would like to reduce the added Gaussian noises so as to improve the accuracy of the results. Hence, we adapt the solution of [12] (see Section 3.1) from two aspects: (1) we apply secure aggregation to aggregate the intermediate results from edge devices; (2) we use a secure multi-party computation (SMC) protocol to enforce the central differential privacy concept in an oblivious manner to the server. The enhanced steps are underlined in the following description.

During the initialisation phase, the server needs to generate additionally a key pair (sk_{hm}, pk_{hm}) for a homomorphic encryption scheme. The enhanced privacy-preserving SVD protocol works as follows.

```
1:  for $t = 1$ to $T$ do
2:      each edge device $i$ computes $\mathbb{Y}_t^{(i)} = \mathbb{M}_i' \mathbb{Z}_{t-1}^{(i)}$
3:      if $t \in \mathcal{I}_T^p$ then
4:          each edge device $i$ adds the Gaussian noise $\mathbb{N}^{(i)}$, thus $\mathbb{Y}_t^{(i)} = \mathbb{Y}_t^{(i)} \mathbb{D}_t^{(i)} + \mathbb{N}^{(i)}$, where $\mathbb{Y}_t^{(i)} \mathbb{D}_t^{(i)}$ is the orthogonal transformation of $\mathbb{Y}_t^{(i)}$
5:          all edge devices and the server run a secure aggregation protocol with $\mathbb{Y}_t^{(i)}$ from all edge devices $i \in [1, d]$ as the inputs, and the output aggregation result for the server is denoted as $\mathbb{Y}_t$.
6:          the server chooses one random index $j$ from $[1, d]$ and generates one $k \times k$ zero matrix $\mathbb{C}$ and another all-ones matrix $\mathbb{C}'$ of the same size. Besides, the server sends the encryption value $\mathbb{E}^{(j)} = \mathsf{Enc}_{pk_h}(\mathbb{C})$ to edge device $j$ and independently generates $\mathbb{E}^{(j')} = \mathsf{Enc}_{pk_h}(\mathbb{C}')$ for each edge devices $j' \in [1, d] \setminus \{j\}$, respectively
7:          each edge device $i$ computes $\mathbb{N}^{(i)} \cdot \mathsf{Enc}_{pk_h}(\mathbb{C}_{(i)})$ and sends it back to the server ($\cdot$ stands for elementary matrix multiplication in the homomorphic sense)
8:          the server first decrypts the receiving messages to obtain $\left(\mathbb{N}^{(i)} \cdot \mathbb{C}_{(i)}\right) = \mathsf{Dec}_{sk_h}\left(\mathbb{N}^{(i)} \cdot \mathsf{Enc}_{pk_h}\left(\mathbb{C}_{(i)}\right)\right) = \mathsf{Dec}_{sk_h}\left(\mathsf{Enc}_{pk_h}\left(\mathbb{N}^{(i)} \cdot \mathbb{C}_{(i)}\right)\right)$ for all $i \neq j$, then the aggregation result is $\mathbb{Y}_t' = \mathbb{Y}_t - \sum_{i \in [1,m] \setminus \{j\}} \left(\mathbb{N}^{(i)} \cdot \mathbb{C}_{(i)}\right) = \sum_{i \in [1,d]} \left(\frac{s_i}{m} \mathbb{Y}_t^{(i)}\right) + \mathbb{N}^{(j)}$
9:          the server performs orthogonalisation $\mathbb{Z}_t = orth(\mathbb{Y}_t')$ and broadcasts it to all edge devices ($\mathbb{Z}_t^{(i)} = \mathbb{Z}_t$ on each edge device $i$)
10:     end if
11: end for
```

12: the final result is the approximated eigenspace:

$$\overline{\mathbb{Z}}_T \begin{cases} \sum_{i=1}^{d} \frac{s_i}{m} \mathbb{Z}_T^{(i)} \mathbb{D}_{T+1}^{(i)} & \text{if } T \notin \mathcal{I}_T^p \\ \sum_{i=1}^{d} \frac{s_i}{m} \mathbb{Z}_T^{(i)} & \text{otherwise.} \end{cases}$$

With the secure aggregation in Step 4, the server obtains the aggregated result with Gaussian noises from all edge devices. With the simple SMC procedure (Step 5-8), the server obtains all Gaussian noises apart from the one (i.e. j) that it has selected randomly. Then, it removes all noises from the output of secure aggregation protocol, except for that from device j. In comparison to the SVD protocol from Section 3.1, the intermediary aggregation result only contains the Gaussian noise from the edge device j. While this index j is hidden from the edge devices.

Regarding Steps 6 and 8, we have two remarks. In Step 6, depending on the homomorphic encryption scheme, it may be more efficient to use ciphertext re-randomisation to generate $\mathbb{E}^{(j')}$ based on an existing ciphertext, rather than generating it from scratch. In Step 8, it may be more efficient to aggregate all the encrypted results and then performs a single decryption. We will discuss this in the performance evaluation.

4.3 Computational Complexity Analysis

Regarding computational complexity, we compare the proposed scheme with the original solution in Table 2. The major difference is that we have integrated secure aggregation to facilitate our new privacy protection strategy. Let SA_e and SA_s be the asymptotic computational complexities of secure aggregation on each edge device and server side, respectively.

		Add	Mul	Noise Gen.	Enc	Dec	Secure Agg.
[12]	Edge	$T \times (k^2 - k) + \lfloor T/p \rfloor \times k^2$	$T \times k^2$	$\lfloor T/p \rfloor \times k^2$			
	Server	$(\lfloor T/p \rfloor + 1) \times k^2 \times (d-1) + \lfloor T/p \rfloor$	$(\lfloor T/p \rfloor + 1) \times d \times k^2 + \lfloor T/p \rfloor \times d + 1$	$\lfloor T/p \rfloor \times k^2$			
Ours	Edge	$T \times (k^2 - k) + \lfloor T/p \rfloor \times k^2$	$T \times k^2 + \lfloor T/p \rfloor \times k^2$	$\lfloor T/p \rfloor \times k^2$			$\lfloor T/p \rfloor \times SA_e$
	Server	$\lfloor T/p \rfloor \times k^2 \times d + k^2 \times (d-1)$	$d \times (k^2 + 1)$		$\lfloor T/p \rfloor \times k^2 \times m$	$\lfloor T/p \rfloor \times k^2 \times m$	$\lfloor T/p \rfloor \times SA_s$

Table 2. Comparison tetween [12] and Ours

Compared to [12], although we have added more operations, we have distributed some computations to individual edge devices and, most importantly, we no longer add secondary server-side Gaussian noise to the final aggregation result and only retain the Gaussian noise from one single edge device.

We implement the scheme in C with GNU Multiple Precision Arithmetic Library (GMP)[1]. In terms of homomorphic encryption, we use Paillier cryptosystem [15], besides, we use [3] as the secure aggregation component. The experiment is performed on a MacBook Pro, which has a 10-core Apple M1 Pro CPU with 16 GB RAM, we use it for both edge devices and the server. Regarding performance, we refer to [12] for parameter selections:

- number of edge device: $d = 100$
- number of local computations: $T = 40$
- number of local computations before communication: $p = 4$
- number of item features: $f = 100$
- number of top eigenvectors: $k = 5$
- security parameter of Paillier cryptosystem and secure aggregation: 128
- threshold value in secure aggregation: 100

In the implementation, on the basis of the observation at the end of Section 4.2, we only encrypt 0 and 1 once at Step 6, and use the property of Paillier cryptosystem to re-randomise their ciphertexts to obtain new ciphertexts, so as to optimise the performance. In addition, we also take advantage of the homomorphic property in Step 8. Instead of decrypting each value ($d \times k^2$ times), we first calculate the product of all the ciphertexts (elementary matrix multiplication) and then perform the decryption on a signal matrix to obtain the sum of all Gaussian noises, except for the noise generated by edge device j chosen in Step 6.

In addition to the optimisation above, we consider the encryption and re-randomisation processes are preconfigured offline. We record the time of all online operations of 10 tests, and give the average running times. It takes 18.09 ms on a signal device and 3.23 ms on the server side. Note that they include 11.79 and 1.48 ms of secure aggregation on each edge device and the server, respectively.

4.4 Privacy and Accuracy Analysis

We conduct privacy analysis under the trust assumption that the server and all edge devices are semi-honest. In this case, since the server is not possible to collude with any edge device, the server cannot eliminate the remaining noise from the final result. In terms of edge device, since no one except the server is aware of the random selection in Step 6, apart from its own data, an edge device only knows the aggregation result with added noise, even if the retained noise comes from itself. Compared to the original solution by Guo et al., we have improved the utility/accuracy of the aggregation result by keeping the added noise from only one edge device. As a side effect, the complexity has grown due to the secure aggregation protocol. This can be regarded as a trade-off between result accuracy and solution efficiency.

[1] https://gmplib.org/

We use Euclidean distance to represent the similarity of two $m \times n$ matrix $\mathbb{A} = (a_{ij})$ and $\mathbb{B} = (b_{ij})$,

$$dist(\mathbb{A}, \mathbb{B}) = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (a_{ij} - b_{ij})^2}.$$

Let $\mathbb{Z}$ denote the true eigenspace computed without any noise, let $\mathbb{Z}_L$ denote the outcoming eigenspace of our scheme (refer to Section 4.2), and let $\mathbb{Z}_G$ denote the eigenspace generated with the Gaussian noise from both edge devices and server (refer to Section 3.1). We have $dist(\mathbb{Z}, \mathbb{Z}_G) = 1.04145$ and $dist(\mathbb{Z}, \mathbb{Z}_L) = 0.09747$. From this, it is clear that the final result from our solution is very close to the true results which are computed from the original data and do not contain any noise. In other words, our scheme better preserves the accuracy and usability of the final results.

5 Conclusion

Motivated by Guo et al.'s distributed privacy-preserving SVD algorithm based on federated power method [12], we have proposed a privacy-preserving federated SVD scheme with secure aggregation. The proposed solution reverts to the initial design intent and interest, although the added operations trade off some performance, it significantly improves the accuracy and utility of the results.

With regard to performance, although we use rerandomisation mechanism instead of re-encrypting 0 and 1 in the experiment, we believe that there are possibilities for further optimisation.

Acknowledgement

Bowen Liu is supported in the context of the project CATALYST funded by Fonds National de la Recherche Luxembourg (FNR, reference 12186579). Qiang Tang is supported by the 5G-INSIGHT bi-lateral project (ANR-20-CE25-0015-16) funded by the FNR and by the French National Research Agency (ANR).

References

1. Bellare, M., Neven, G.: Transitive signatures: new schemes and proofs. IEEE Transactions on Information Theory **51**(6), 2133–2151 (2005)
2. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudorandom bits. SIAM journal on Computing **13**(4), 850–864 (1984)
3. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1175–1191 (2017)
4. Candès, E.J., Recht, B.: Exact matrix completion via convex optimization. Foundations of Computational mathematics **9**(6), 717–772 (2009)

5. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
6. Dumais, S.T., et al.: Latent semantic analysis. Annu. Rev. Inf. Sci. Technol. **38**(1), 188–230 (2004)
7. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Theory of cryptography conference. pp. 265–284. Springer (2006)
8. Golub, G., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis **2**(2), 205–224 (1965)
9. Golub, G.H., Reinsch, C.: Singular value decomposition and least squares solutions. In: Linear algebra, pp. 134–151. Springer (1971)
10. Golub, G.H., Van Loan, C.F.: Matrix computations. JHU press (2013)
11. Guo, Q., Zhang, C., Zhang, Y., Liu, H.: An efficient svd-based method for image denoising. IEEE transactions on Circuits and Systems for Video Technology **26**(5), 868–880 (2015)
12. Guo, X., Li, X., Chang, X., Wang, S., Zhang, Z.: Privacy-preserving distributed svd via federated power. arXiv preprint arXiv:2103.00704 (2021)
13. Hartebrodt, A., Röttger, R., Blumenthal, D.B.: Federated singular value decomposition for high dimensional data. arXiv preprint arXiv:2205.12109 (2022)
14. McGrew, D., Viega, J.: The galois/counter mode of operation (gcm). submission to NIST Modes of Operation Process **20**, 0278–0070 (2004)
15. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238. Springer (1999)
16. Polat, H., Du, W.: Svd-based collaborative filtering with privacy. In: Proceedings of the 2005 ACM symposium on Applied computing. pp. 791–795 (2005)
17. Rajwade, A., Rangarajan, A., Banerjee, A.: Image denoising using the higher order singular value decomposition. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(4), 849–862 (2012)
18. Ravi Kanth, K., Agrawal, D., Singh, A.: Dimensionality reduction for similarity searching in dynamic databases. ACM SIGMOD Record **27**(2), 166–176 (1998)
19. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
20. Von Luxburg, U.: A tutorial on spectral clustering. Statistics and computing **17**(4), 395–416 (2007)
21. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemometrics and intelligent laboratory systems **2**(1-3), 37–52 (1987)
22. Yakut, I., Polat, H.: Privacy-preserving svd-based collaborative filtering on partitioned data. International Journal of Information Technology & Decision Making **9**(03), 473–502 (2010)
23. Yang, Q.: Advances and open problems in federated learning. Foundations and Trends in Machine Learning (2021)
24. Yao, A.C.: Theory and application of trapdoor functions. In: 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982). pp. 80–91. IEEE (1982)
25. Zhang, S., Wang, W., Ford, J., Makedon, F., Pearlman, J.: Using singular value decomposition approximation for collaborative filtering. In: Seventh IEEE International Conference on E-Commerce Technology (CEC'05). pp. 257–264. IEEE (2005)