ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Efficient Decentralized Federated Singular Vector Decomposition

Di Chai[1], Junxue Zhang[1], Liu Yang[1], Yilun Jin[1], Leye Wang[2], Kai Chen[1], and Qiang Yang[1]

[1]Hong Kong University of Science and Technology

[2]Peking University

iSingLab HKUST

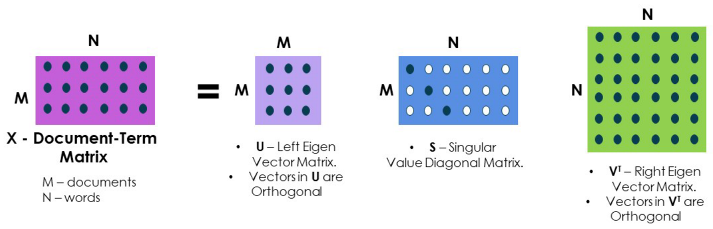
香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

北京大學 PEKING UNIVERSITY

Outline

- Introduction: Background and Motivation
- Excalibur’s Matrix Protection
- Excalibur’s Decentralized SVD Workflow
- Implementation and Evaluation
- Conclusion and Future Work

Federated SVD is an Essential Primitive



Latent sematic analysis / topic modeling

$$\min_{w} \|Xw - y\|_2^2 + \alpha\|w\|_2^2$$

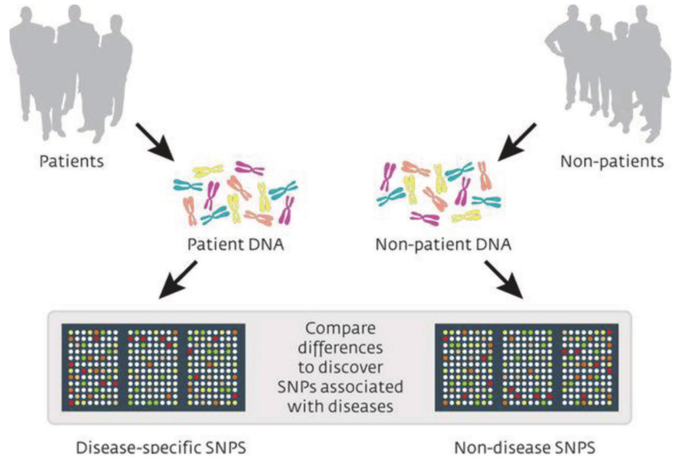
$$\downarrow$$

$$X = U\Sigma V^T$$

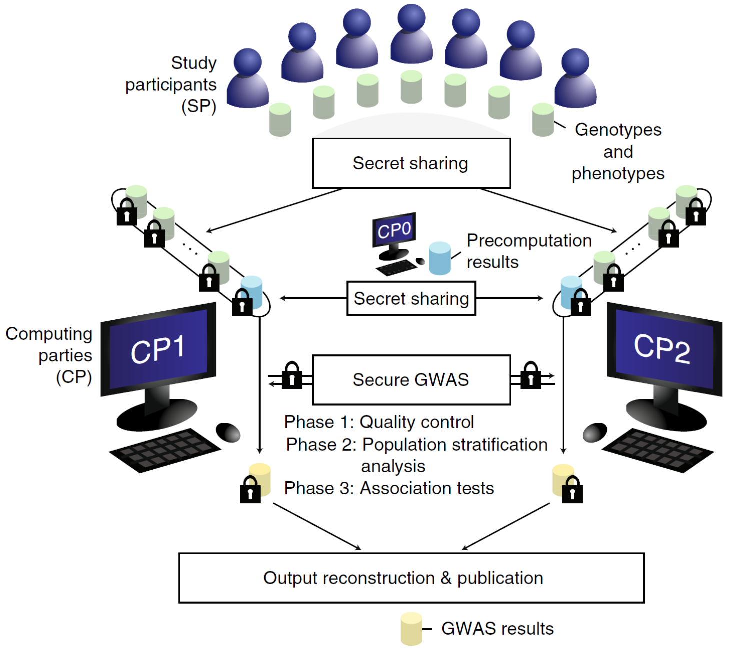
$$\downarrow$$

$$w = V(\Sigma^T\Sigma + \alpha I)^{-1}\Sigma^T U^T y$$

SVD works as the solver for LR.



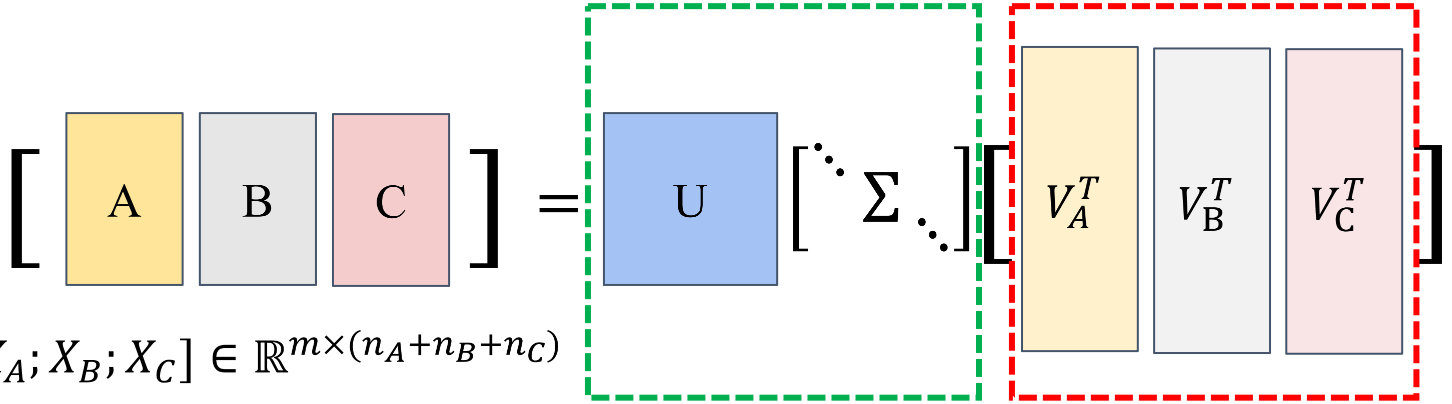
SVD-based Genome-Wide Association Studies (GWAS) require <u>million-scale samples</u>



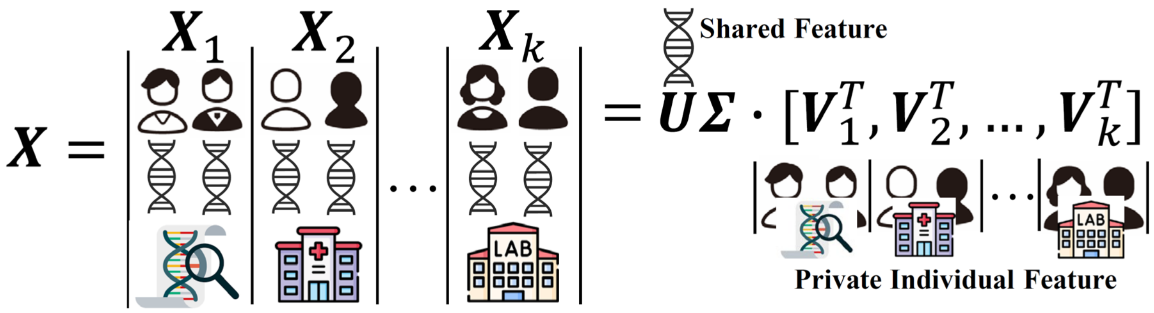
Real-world applications require combining different data sources!

Federated Singular Vector Decomposition (SVD) is an essential primitive to support many real-world distributed application.

Problem Definition of Federated SVD



Federated SVD factorizes matrix from multiple domains into public and secret parts.



An illustration of federated SVD in GWAS.

External Servers Downgrade the Privacy Protection

Most of the existing works rely on **external servers.**

Existing Studies	# of Servers	Job of Data Contributors	Job of Server(s)	Threats of Privacy Leakage
[7]	Two	$\mathbf{X}' = \mathbf{PXQ}$	<u>**SVD**</u> on $\mathbf{X}'$	Raw Data
[9]	Four	$\mathbf{XX}^\mathbf{T}, \mathbf{X}^\mathbf{T}\mathbf{X}$	<u>**SVD**</u> on $\mathbf{XX}^\mathbf{T}, \mathbf{X}^\mathbf{T}\mathbf{X}$	Raw Data
[10]	Three	Secret Sharing $\mathbf{X} = \mathbf{X_a} + \mathbf{X_b}$	<u>**SVD**</u> on $\mathbf{X_a} + \mathbf{X_b}$	Raw Data
[25]	One	Project $\mathbf{X}$ $\mathbf{H} = \mathbf{XG}$	<u>**Orthogonal**</u> $\mathbf{H}$ (*e.g.*, Gram-Schmidt)	Projections of Raw Data
[34]	One	Project $\mathbf{X}^\mathbf{T}\mathbf{X}$ $\mathbf{Y} = \mathbf{X}^\mathbf{T}\mathbf{XZ}$	<u>**Orthogonal**</u> $\mathbf{Y}$ (*e.g.*, Gram-Schmidt)	Projections of Raw Data

The servers obtain excessive access to the private data and thus **significantly decreases the privacy protection.**

Intuitive ideas of enhancing privacy protection at the server side cannot work

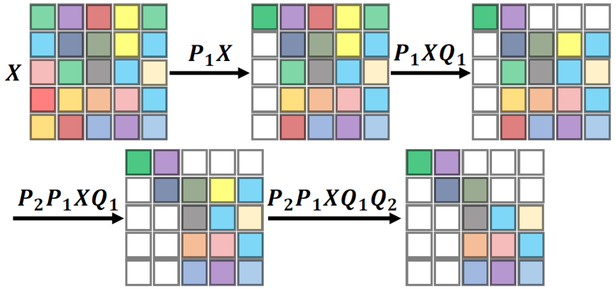
- Pick a subset of the users as "servers". The privacy issues remain in unselected users.
- Deploy TEE at the servers. The issue of distrust, particularly in the server-aided approach, poses a significant challenge.
- Leveraging HE at the servers. HE brings severe computational overhead (will discuss more).

Efficient Decentralization is Challenging

Existing works have explored using **Homomorphic Encryption** to remove the servers but suffer from **significant efficiency issues**.

Computational Challenge

HE-based solution is $4 \sim 5$ orders of magnitudes slower.



Sequential computation makes stacking more hardware less effective. Designing new hardware is also challenging.

Communication Challenge

Data Size	Communication Size	
	Before Encryption	After Encryption
$m = 1K$ $n = 50M$	372GB	11.6TB

The overhead of communication size is significant and becomes more severe after encryption.

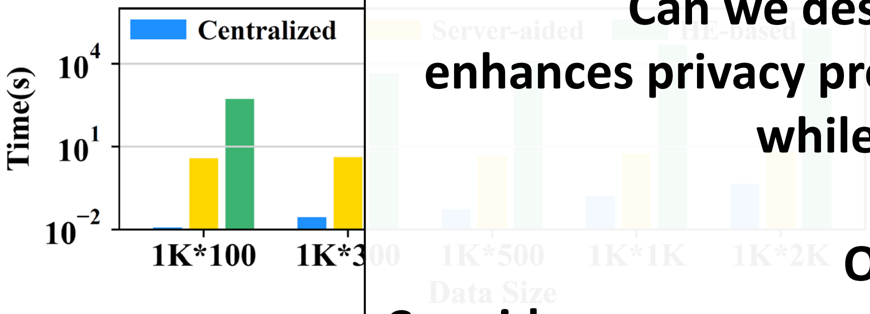
Data Size	# of Peers	Communication Rounds
$m = 10K$	$k = 10$	4m(k-1) 2.5 Hours when RTT=50ms

Popular centralized SVD methods have significant overhead of communication rounds.

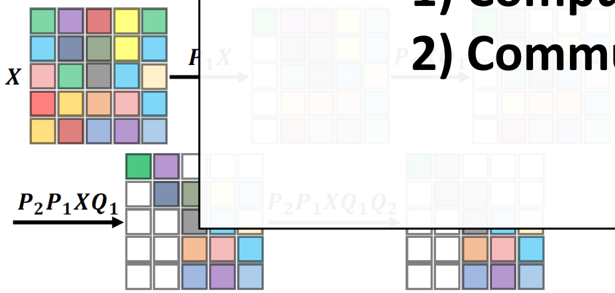
Efficient Decentralization is Challenging

Existing works have explored using **Homomorphic Encryption** to remove the servers but suffer from **significant efficiency issues**.

Computational Challenge



HE-based solution is 4~5 orders of magnitudes slower.



Sequential computation makes stacking more hardware less effective. Designing new hardware is also challenging.

Communication Challenge

Data Size	Communication Size	
	Before Encryption	After Encryption
$m = 1K$ $n = 50M$	372GB	11.6TB

The overhead of communication size is significant and becomes more severe after encryption.

Data Size	# of Peers	Communication Rounds
$m = 10K$	$k = 10$	4m(k-1) 2.5 Hours when RTT=50ms

Popular centralized SVD methods have significant overhead of communication rounds.

Can we design a federated SVD system that enhances privacy protection by removing the external servers while achieving high efficiency?

Our answer is Excalibur.

Core ideas:

1) Computation-efficient matrix protection

2) Communication-efficient decentralized SVD workflow

Outline

- Introduction
- Excalibur’s Matrix Protection (computation-efficient protection)
  - Threat Model and Security Goals
  - Multiplicative Matrix Sharing
  - Accelerating the Multiplicative Operations
- Excalibur’s Decentralized SVD Workflow (communication-efficient workflow)
- Implementation and Evaluation
- Conclusion and Future Work

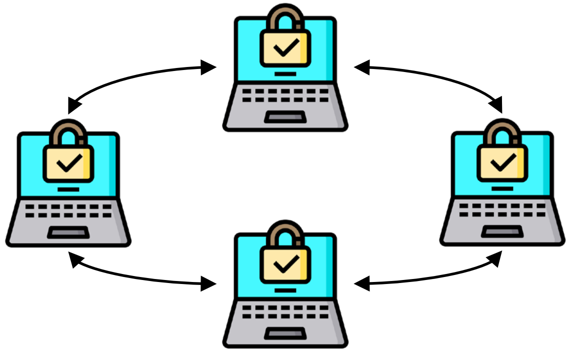
Threat Model and Security Goals

Threat Model: We assume all peers are semi-honest.

✅ Strictly follow pre-defined protocol

❌ But try to discover privacy during execution

Security Definition: The system is secure if all intermediate results could be derived from final results. (Having the same distribution in mathematical language)



Same definition to secure multi-party computation (SMC)

Discussion

(1) What if they do not follow the protocol?

Check whether $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}, \boldsymbol{V}_i\boldsymbol{V}_i^T = \boldsymbol{I}, \boldsymbol{X}_i = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}_i^T$

(2) How to protect the final results?

Leveraging differential privacy.

Multiplicative Matrix Sharing (MMS)

We protect the matrix with random non-singular matrices

$$X' = AXB = A[X_1, \dots . X_k]\begin{bmatrix} B_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & B_k \end{bmatrix}$$

For peer-i

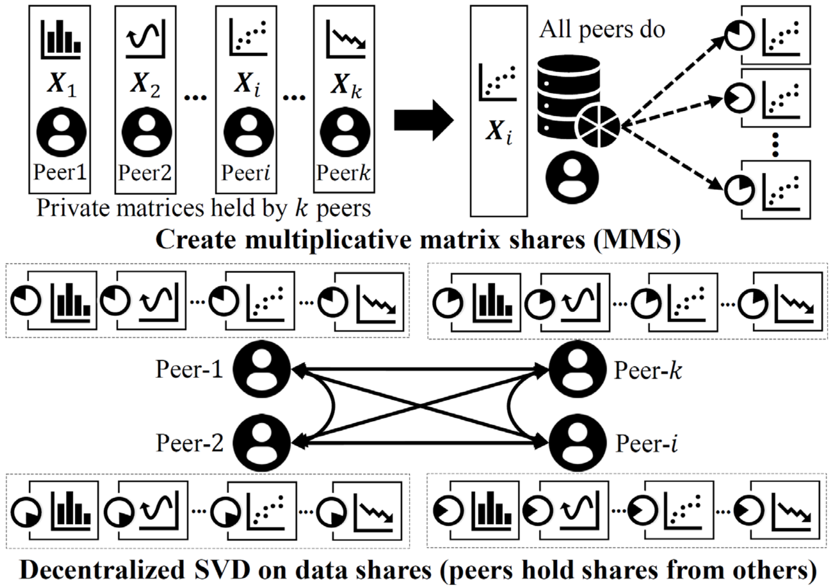
$$X_i' = \begin{bmatrix} A_1 \\ \vdots \\ A_k \end{bmatrix} X_i B_i = \begin{bmatrix} A_1 X_i B_i \\ \vdots \\ A_k X_i B_i \end{bmatrix} = \begin{bmatrix} \boxed{S_i^1} \\ \vdots \\ S_i^k \end{bmatrix}$$

One MMS

Recovering X_i needs all the matrix shares

$$X_i = A^{-1} X_i' B_i^{-1} = [A_1^{-1}, \dots, A_k^{-1}]\begin{bmatrix} S_i^1 \\ \vdots \\ S_i^k \end{bmatrix} B_i^{-1} = \sum_{j=1}^{k} A_j^{-1} S_i^j B_i^{-1}$$

(A_j^{-1} is the columns of A^{-1})



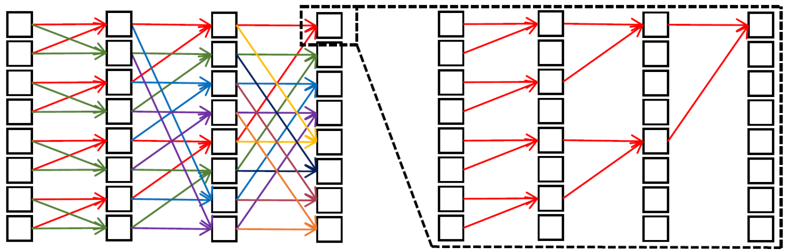
Theorem 1. *Denote $\mathbb{O}_n$ as the compact group of $n \times n$ orthogonal matrices under Haar measure, if we choose dense matrix $\mathbf{A} \in \mathbb{O}_m$ and uniformly generate $\mathbf{B_i} \in \mathbb{O}_{n_i}$, Excalibur produces federated SVD results with no accuracy loss and can satisfy the security defined in Definition 1 while the adversary can compromise up to $k-1$ peers ($|C| = k-1$).*

Accelerating the Multiplicative Operations

Complexity of generating and applying random orthogonal matrices is $O(m^2n)$ or $O(n^2m)$.

How to efficiently support large-scale data?

(1) Reduce algorithm complexity

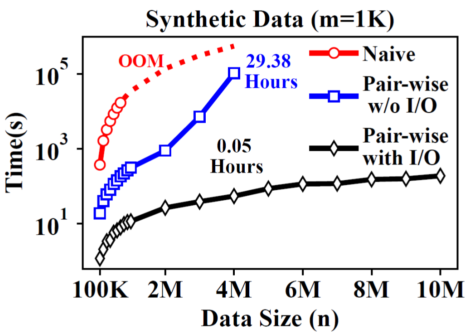


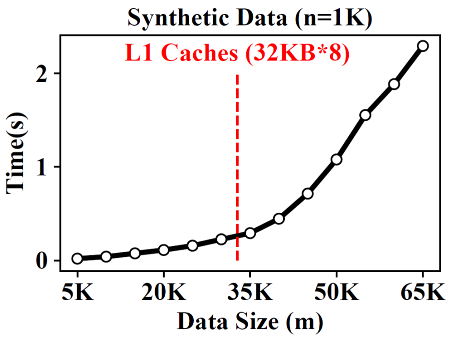
Pair-wise Multiplication ($\delta = 3$) Computation path of one row

$O(m^2n_i)$ -> $O(mn_i logm)$

Transfer A to a group of 2×2 rotations on random selected rows.

(2) Solve the I/O bottleneck





Process by columns instead of by rows. Formulate each column into rectangular matrix if it exceed L1 cache.

(3) Local pre-processing for matrix $\boldsymbol{B}$

$$\boldsymbol{X}_i = \boldsymbol{R}_i^T \boldsymbol{Q}_i^T$$

$$[\boldsymbol{R}_1^T, \dots, \boldsymbol{R}_k^T] = \boldsymbol{U\Sigma}[\boldsymbol{V}_{\boldsymbol{R}_1}^T, \dots, \boldsymbol{V}_{\boldsymbol{R}_k}^T]$$

$O(n_i^2m)$ -> $O(m^2n_i + m^3)$

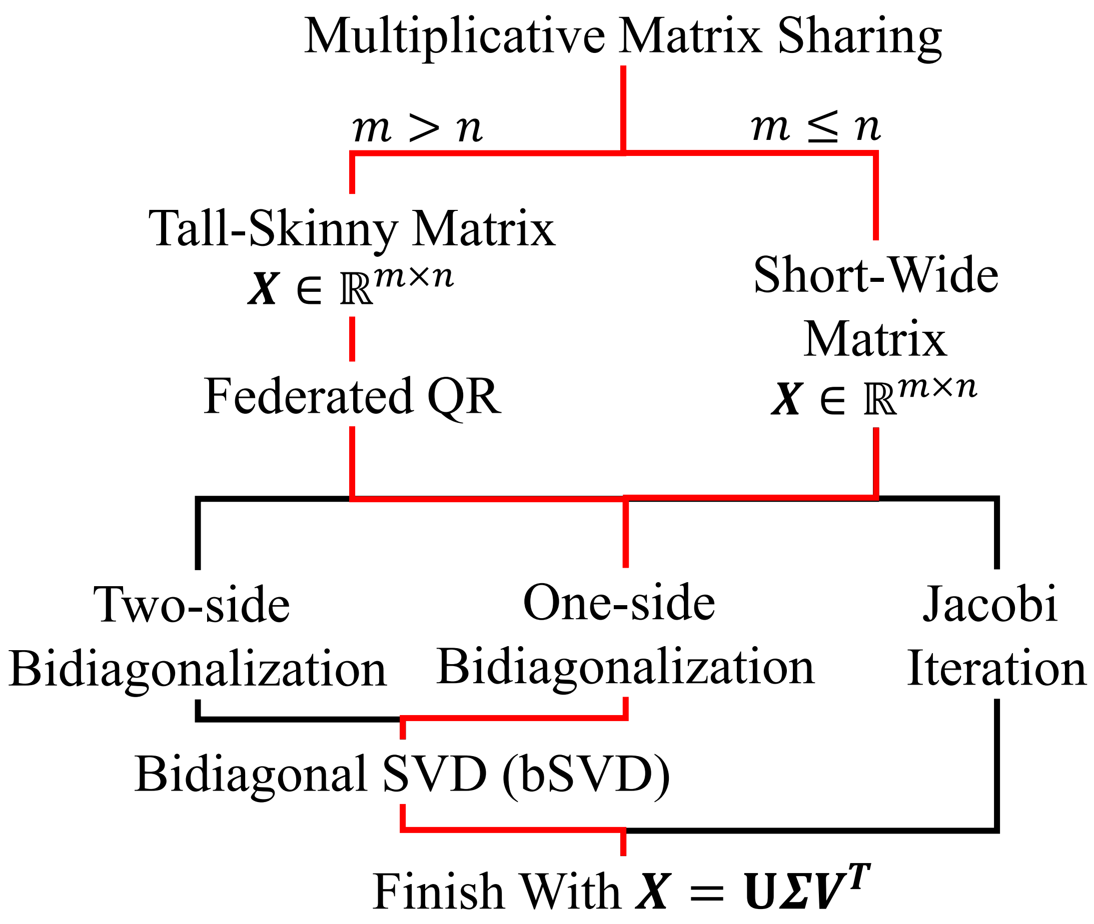
Local pre-process to reduce complexity when $m \ll n$.

With all the above optimizations, the MMS can efficiently support billion-scale data.

Outline

- Introduction
- Excalibur’s Matrix Protection (computation-efficient protection)
- Excalibur’s Decentralized SVD Workflow (communication-efficient workflow)
  - Analyzing the Design Space
  - Overlapping the Pipelines
- Implementation and Evaluation
- Conclusion and Future Work

Design Space of Decentralized SVD



Design space of decentralized SVD

Decentralized SVD Path	Short-Wide Matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ $m \leq n$; k peers	
	CommAmount (Bandwidth)	CommTimes (Latency)
Jacobi iteration (1 iteration)	$3\frac{k-1}{k}(m^2 - m)$ $\Rightarrow O(m^2)$	$m(m-1)(k-1)$ $\Rightarrow O(km^2)$
Two-side Bidiagonal + bSVD	$(k-1)(\frac{3}{2}m^2 + \frac{7}{2}m)$ $\Rightarrow O(km^2)$	$4m(k-1)$ $\Rightarrow O(km)$
One-side Bidiagonal + bSVD	$\frac{k-1}{k}(m^2 - m)$ $\Rightarrow O(m^2)$	$(6m-6)(k-1)$ $\Rightarrow O(km)$

- Jacobi iteration has significantly higher communication rounds.
- Two-side bidiagonalization is the popular method used in NumPy and LAPACK, but its communication size is large.
- The one-side bidiagonalization has the minimum communication complexity.

Overlapping the Pipelines

Algorithm 1: Excalibur's decentralized SVD workflow. (The three looped ring all-reduce are highlighted and we will reduce them to only one through overlapping the pipelines (§5.2).)

Input: Matrix $\mathbf{X} = [\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_k}]$ held by k peers, where $\mathbf{X} \in \mathbb{R}^{m\times n}$, $m \le n$, $\mathbf{X_i} \in \mathbb{R}^{m\times n_i}$, and $\sum_{i=1}^{k} n_i = n$.
Output: $\mathbf{U}, \mathbf{\Sigma}, [\mathbf{V_1^T}, \mathbf{V_2^T}, \ldots, \mathbf{V_k^T}]$ (*i.e.*, SVD of $\mathbf{X}$)

Function DecSVD ($\mathbf{X}$) **:**
- // All peers run this function in parallel
- $\mathbf{U} \leftarrow I, c \leftarrow MyPeerID$ ▷ *e.g.*, $c = 1$ for peer-1
- **for** i = 1, 2, ..., m-2 **do**
  - $\mathbf{h} \leftarrow RingAllReduce(\mathbf{X_c}[i] * \mathbf{X_c}[i+1:]^T)$
  - // Apply reflector to $\mathbf{X}$ and $\mathbf{U}$
  - $\mathbf{X_c}[\mathbf{i+1}:] \leftarrow house(\mathbf{h}) \otimes \mathbf{X_c}[\mathbf{i+1}:]$
  - $\mathbf{U}[\mathbf{i+1}:] \leftarrow house(\mathbf{h}) \otimes \mathbf{U}[\mathbf{i+1}:]$
- **end**
- // α contains the diagonal elements.
- // β contains the subdiagonal elements.
- $\alpha \leftarrow \{0\}^m, \beta \leftarrow \{0\}^{m-1}$
- $\alpha[1] \leftarrow \sqrt{RingAllReduce(\|X_c[1]\|_2^2)}$
- $\mathbf{V_c^T}[\mathbf{1}] = \mathbf{X_c}[1]/\alpha[1]$
- **for** i = 2, 3, ..., m **do**
  - $\beta[i-1] = RingAllReduce(\mathbf{X_c}[\mathbf{i}] * \mathbf{V_c}[\mathbf{i-1}])$
  - $\mathbf{X_c}[\mathbf{i}] \leftarrow \mathbf{X_c}[\mathbf{i}] - \beta[i-1] * \mathbf{V_c^T}[\mathbf{i-1}]$
  - $\alpha[i] \leftarrow \sqrt{RingAllReduce(\|X_c[i]\|_2^2)}$
  - $\mathbf{V_c^T}[\mathbf{i}] = \mathbf{X_c}[i]/\alpha[i]$
- **end**
- $\mathbf{U_b}, \mathbf{\Sigma}, \mathbf{V_b^T} \leftarrow bSVD(\alpha, \beta)$
- // Combine results together
- $\mathbf{U} \leftarrow \mathbf{U} * \mathbf{U_b}$
- $\mathbf{V_c^T} \leftarrow \mathbf{V_b^T} * \mathbf{V_c^T}$
- **return** $\mathbf{U}, \mathbf{\Sigma}, [\mathbf{V_1^T}, \mathbf{V_2^T}, \ldots, \mathbf{V_k^T}]$

End Function

The three **all-reduce communications** in the for loops **become the bottleneck**.

$$\boxed{\alpha_i^2} = \sum_{j=1}^{k} \|\mathbf{X_j^i}\|_2^2 = \sum_{j=1}^{k} \|\mathbf{X_j^i} - \boxed{\beta_{i-1}}(\mathbf{V_j^{i-1}})^\mathbf{T}\|_2^2$$

Overlapping the communications is challenging due to data dependency.

$$\alpha_i = \sqrt{\theta_1^i - 2(\theta_2^i)^2 + (\theta_2^i)^2\theta_3^i}\ , \quad \beta_{i-1} = \theta_2^i$$

We find the underlying shared components of these communications.

3	**for** i = 1, 2, ..., m-2 **do**
11	**for** i = 2, 3, ..., m **do**

Merge for loops via pipeline parallel.

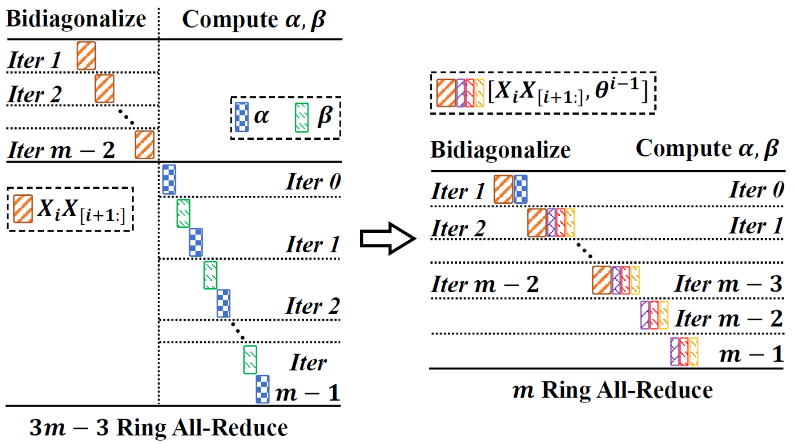


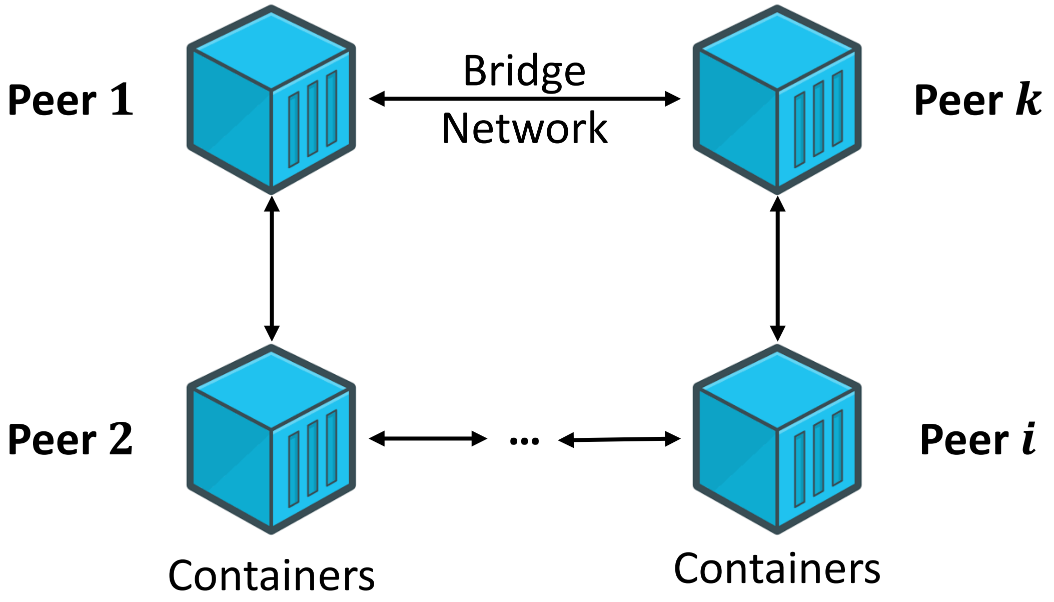
Figure 7: Overlapping the pipeline reduces the number of ring all-reduce communication from $3m-3$ to m, *i.e.*, approximately reduced by 66%.

Reducing 66% communication cost.

Outline

- Introduction
- Excalibur's Matrix Protection
- Excalibur's Decentralized SVD Workflow
- Implementation and Evaluation
- Conclusion and Future Work

Implementation and Testbed



Each container is assigned with 4 Cores and 64GB RAM
Default network setting: 1Gbps bandwidth and 50ms RTT

We implement a fully functional prototype using C/C++. The system runs in double precision, i.e., 64 bits.

- We use BLAS and LAPACKE from Intel MKL as the major library.
- For operations not included in existing library, we implement from scratch and use OMP and AVX2 for parallelism.
- To support large-scale data that cannot fit into memory, we create memory-mapped files and offload the data to NVMe SSD.

Evaluation

Datasets

We have used four datasets in the evaluation: MNIST, Wine, ML100K, and synthetic data.

Baselines

1) FedSVD: state-of-the-art (SOTA) server-aided federated SVD.

2) SF-PCA: SOTA multi-key HE-based solution.

3) FATE and SecureML: widely used federated linear regression (LR) systems.

Tasks

SVD task and its three applications:

Principal components analysis (PCA), latent semantic analysis (LSA), and LR.

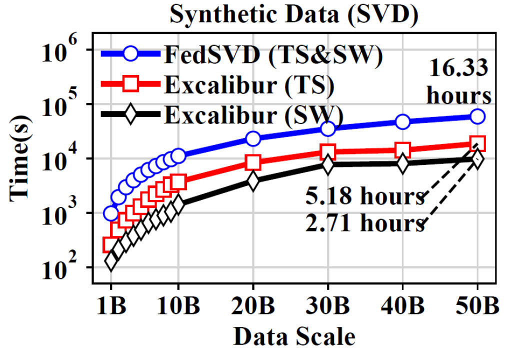
Accuracy Evaluation

Method	Wine	MNIST	ML100K	Synthetic
FedSVD	1.44×10^{-13}	2.66×10^{-10}	2.56×10^{-12}	5.67×10^{-15}
Excalibur	3.56×10^{-14}	2.15×10^{-13}	3.76×10^{-15}	2.96×10^{-17}

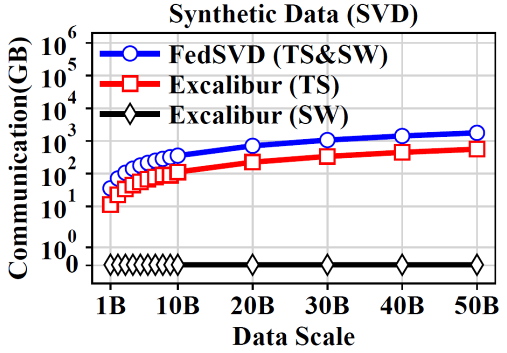
Table 3: Reconstruction error of SVD on three real-world datasets and synthetic data.

Evaluation

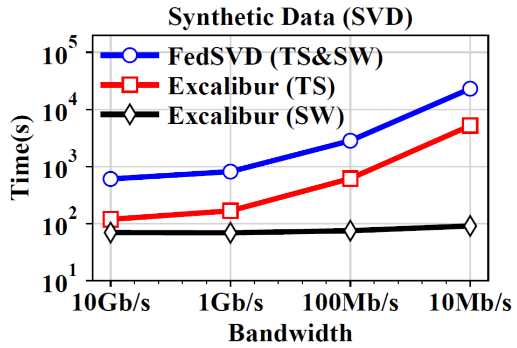
Efficiency on SVD Task



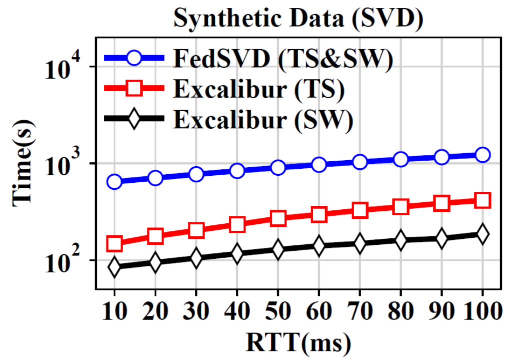
(a) End-to-end time comparison on SVD task under 1~50 billion data.



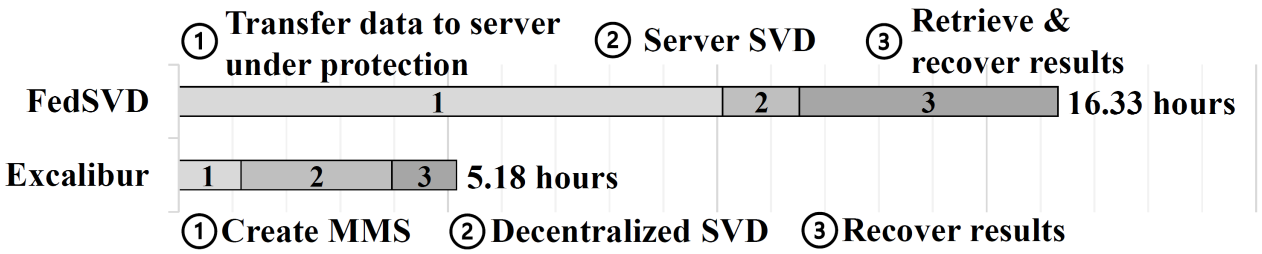
(b) Communication cost on SVD task under 1~50 billion data.



(c) Impact of network bandwidth on SVD efficiency using 1 billion data.



(d) Impact of network latency on SVD efficiency using 1 billion data.



Compared to the SOTA server-aided system, Excalibur not only removes the external servers but also achieves better efficiency.

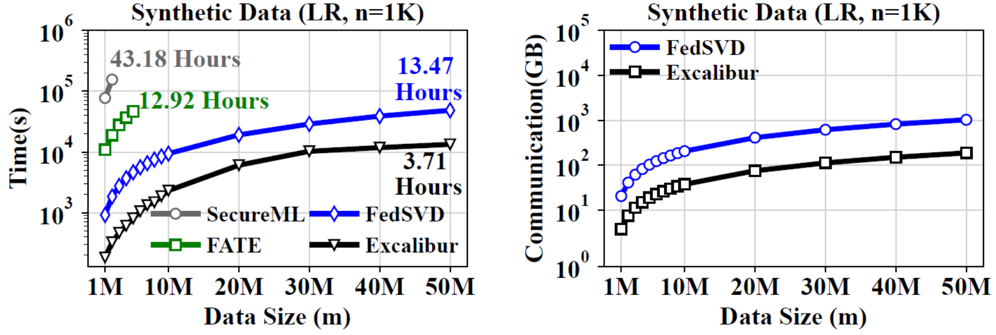
- Excalibur is 3.1 × ~6.0 × faster than FedSVD.
- Excalibur reduces more than 68.4% amount of communication.

Evaluation

Efficiency on SVD Applications

Solution	Data	CPU Cores	Time	Single Core Throughput
SF-PCA[17]	45.6M (760×60K)	72	2.22 Hours	0.28 M/H
Excalibur	10000M (1K×10M)	24	0.063 Hours	6595.55 M/H

Comparing Excalibur with SF-PCA on PCA application, while computing the top-5 principal components (bandwidth=1Gb/s, RTT=20ms, six peers).



(a) Efficiency of Excalibur, FedSVD, FATE, and SecureML on LR.

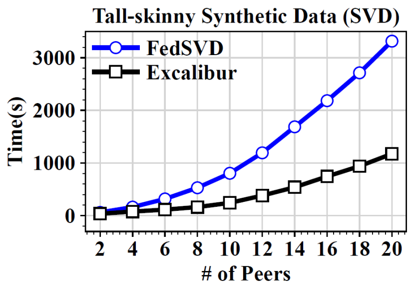
(b) Communication size of Excalibur and FedSVD on LR.

Comparing Excalibur with FedSVD, FATE, and SecureML on LR application.

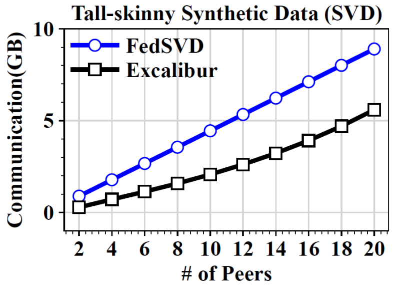
- Compared to the SOTA HE-based system that attempted to remove the servers, Excalibur is far more efficient and has $> 23000 \times$ larger throughput.
- Comparing to two widely used federated LR systems: FATE and SecureML, Excalibur is 100x and 1000x faster, respectively.

Evaluation

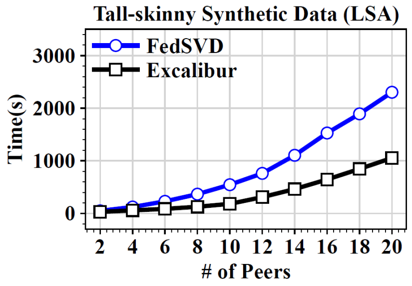
Scalability on SVD and LSA



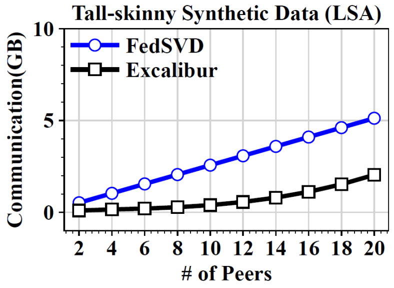
(a) End-to-end time consumption when increasing the # of peers.



(b) Comm cost (each peer) when increasing the # of peers.



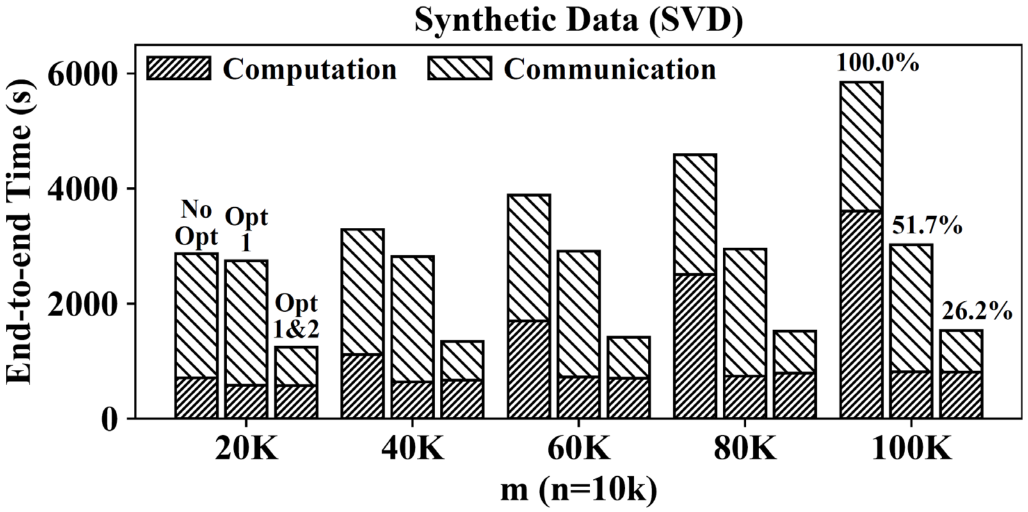
(c) End-to-end time consumption when increasing the # of peers.



(d) Comm cost (each peer) when increasing the # of peers.

We evaluate Excalibur's scalability when increasing the number of peers, assuming all peers hold the same amount of data and test the efficiency when more peers join the federation.

Effectiveness of the Optimizations

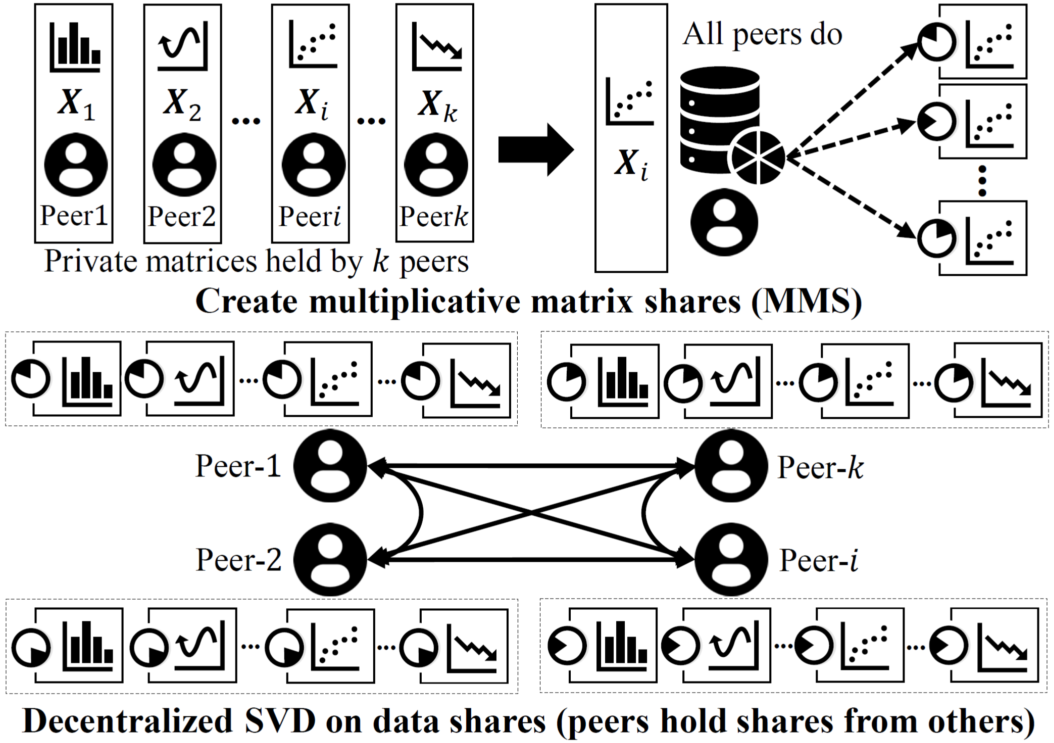


Measuring the effectiveness of system optimizations in Excalibur, while NoOpt means no optimization, Opt1 is optimizing the multiplicative operations in MMS, and Opt2 is overlapping pipelines to reduce communication rounds.

Outline

- Introduction
- Excalibur’s Matrix Protection
- Excalibur’s Decentralized SVD Workflow
- Implementation and Evaluation
- Conclusion and Future Work

Conclusion and Future Work



Conclusion

In this paper, we propose Excalibur, an efficient decentralized federated SVD that **not only eliminates the privacy concerns** caused by external servers **but also can efficiently decompose large-scale matrices**.

Future Work

- How to update the results when more peers are joining in the computation?
- How to utilize the matrix protection in this paper in other scenarios? For example, the secure model inference.

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Thanks

Artifact available at: https://github.com/Di-Chai/Excalibur