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[1,3]
[1]*Hong Kong University of Science and Technology*
[2]*MoE Key Lab of High Confidence Software Technologies, Peking University, and School of Computer Science, Peking University* [3]*Webank*

Abstract

Federated singular value decomposition (SVD) is a foundation for many real-world distributed applications. Existing federated SVD studies either require external servers which downgrade privacy protection or leverage homomorphic encryption (HE) to get rid of external servers (*i.e.*, being decentralized) but suffer from significant inefficiencies caused by extensive computational and communication overhead.

This paper presents Excalibur[1], an efficient decentralized federated SVD system. At its core, Excalibur proposes a lightweight matrix protection method to reduce the computational degradation caused by cryptographic operations, improving computation performance. Furthermore, it designs a communication-efficient decentralized SVD workflow based on the quantitative analysis of the design space, optimizing communication performance. To validate the efficiency of Excalibur, we implement a fully functional Excalibur system and evaluate it with real-world applications. Our results show that Excalibur not only removes the external servers but also achieves $3.1\times \sim 6.0\times$ faster performance than state-of-the-art (SOTA) server-aided method on different shapes of billion-scale data. In addition, Excalibur exhibits $> 23000\times$ larger throughput than the SOTA HE-based system.

1 Introduction

Federated singular vector decomposition (SVD) enables privacy-preserving factorization of matrices across distributed systems. SVD is the base of many real-world applications which cover a wide range of scenarios including genetic association studies [7, 17, 25, 43], medical studies [4, 26, 49], risk management in banks [19, 28, 48], language modeling [15, 16, 30], *etc*. These applications usually involve large-scale sensitive data, and one major challenge is that a single party rarely holds enough data to produce accurate and robust results [7, 9, 10, 17, 25, 34, 35]. For instance, the genome-wide association studies (GWAS), which employ SVD for group stratification correction, reveal the correlation between DNA sequences and human diseases. Obtaining data in GWAS involves DNA sequencing, which is costly[2], while robust studies require million-scale samples [10]. This clearly raises challenges for a single party regarding obtaining enough data for robust analysis, hindering the development in these areas. Federated learning is a promising direction to solve this problem by enabling real-world SVD applications to utilize data from distributed datasets while protecting privacy. While several federated SVD systems have been proposed in the literature, many of them rely on external servers (*i.e.*, the server-aided approaches) [7, 9, 10, 25, 34].

Involving external servers significantly decreases the privacy protection of federated SVD systems. Existing server-aided approaches [7, 9, 10, 25, 34] leaks private data to the external servers, either directly or when the servers collude with each other (§2.2). Because the external servers, which are not data contributors, receive a substantial amount of sensitive information during the system's execution. Intuitive ideas for enhancing the server-aided methods are also impractical (§2.2). Removing the external servers to achieve a decentralized federated SVD is essential to solve this problem.

However, existing decentralized federated SVD studies [17, 35] have substantial efficiency issues, making them impractical in real-world scenarios. The major issue is that they use homomorphic encryption (HE) for privacy protection. While HE's strong confidentiality allows the exchange of encrypted data without external servers, the encryption enlarges the data size (*e.g.*, from 64bits to 2048bits) [29] and brings expensive operation (*e.g.*, bootstrapping and ciphertext rotations [1]), which incurs substantial computational overhead. Existing work [7] shows that fully decomposing million-scale matrices requires over 15 years using HE system [35], while real-world applications typically deal with billion-scale data (*e.g.*, genetic studies). Moreover, [17] reports partial decomposition

[1]The legend says that Arthur pulled Excalibur, the sword of power, from the stone and formed the Knights of the Round Table to protect the kingdom. All knights were equal in ranks and levels, showing a decentralized ethos.

[2]Although the cost of DNA sequencing is getting cheaper, sequencing one person's genome still costs ~$1000 according to the National Human Genome Research Institute (NHGRI). https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost

(*i.e.*, top $2 \sim 5$ singular vectors) of million-scale data using severe hours, which is also far from practical for applications with billion-scale data or requiring more decomposition results. The characteristic of SVD (*e.g.*, computationally intensive and sequential workflow) makes it hard to accelerate HE-based SVD through stacking more hardware or designing specific accelerators (§2.3).

In this paper, we ask: *can we design a federated SVD system that enhances privacy protection by removing the external servers while achieving high efficiency?* Our answer is Excalibur, an efficient decentralized federated SVD system. Excalibur confronts two challenges: 1) conventional HE (or other cryptography techniques) causes dramatic computation inefficiency, making the system orders of magnitude slower [7, 60, 61]; 2) decentralized SVD itself causes communication inefficiency due to the multiple rounds of communication and large communication amounts, which will also be enlarged if adding HE protection (*e.g.*, encrypting data from 64bits to 2048bits enlarges by $32\times$).

To address the above two challenges, Excalibur: 1) proposes multiplicative matrix sharing (MMS), a lightweight matrix protection method dedicated to decentralized SVD, to efficiently achieve privacy preservation (§4); 2) proposes communication-efficient decentralized SVD workflow through quantitatively analyzing the design space and overlapping system pipelines (§5). Specifically, MMS achieves protection by randomly projecting data into different shares held by all peers such that a single peer or a subset of peers cannot recover the data unless obtaining consensus from all peers (§4.1). MMS achieves better computational efficiency than HE since MMS does not enlarge the data size, and we optimize the additional multiplicative operations to efficiently support large-scale data (§4.2). Excalibur proposes a decentralized SVD workflow with low communication complexity by quantitatively analyzing the design space (§5.1) and reduces 66% of communication rounds through overlapping system pipelines (§5.2). Regarding privacy preservation, we formally prove that Excalibur satisfies the security definition of secure multi-party computation (Definition 1 and §6), which is also used by many widely known SMPC protocols, *e.g.*, garbled circuits and secret sharing [11, 27].

We implement a fully functional system of Excalibur and perform comprehensive evaluations. The results show that: 1) comparing to SOTA server-aided system [7], Excalibur not only eliminates external servers but also achieves better efficiency (credits to the computational and communication advances of Excalibur which is analyzed in §8.3), and it is $3.1\times \sim 6.0\times$ faster than FedSVD [7] on billion-scale data and reduces more than 68.4% amount of communication; 2) comparing to SOTA HE-based system [17], Excalibur is far more efficient and has $> 23000\times$ larger throughput.

To the best of our knowledge, Excalibur is among the first federated SVD systems that support the efficient decomposition of large-scale data without needing external servers. Furthermore, we note that the privacy protection and optimization strategies used in Excalibur are also broadly applicable to the design of other systems, *e.g.* secure database query [55] and out-sourced matrix computations [14, 18, 42, 62], and thus of independent interest.

2 Backgrounds and Motivation

2.1 Federated SVD

Federated SVD is an essential primitive to support many real-world distributed federated learning applications, *e.g.*, decomposing large-scale gene data distributed in hospitals and genetic sampling agencies to assist federated genetic studies [7, 10, 17, 25], factorizing large-scale user profiles distributed in banks and online shopping companies as dimensional reduction and linear regression tools to assist federated risk management [7, 19, 28, 48, 59], decomposing large-scale distributed word-document data to assist federated language modeling [15, 16, 30], and so on. In this paper, following previous work [7, 9, 10, 17, 25, 34, 35], we focus on the SVD algorithm that decomposes a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ into a product of three matrices $\mathbf{X} = \mathbf{U\Sigma V^T}$, where $\mathbf{U} \in \mathbb{R}^{m\times m}, \mathbf{V^T} \in \mathbb{R}^{n\times n}$ are orthogonal singular vectors and $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is a diagonal matrix containing all the singular values. In federated SVD, the data matrix $\mathbf{X}$ is jointly possessed by different peers, *e.g.*, each peer holds different columns of data if the matrix is vertically split. Different split patterns (*i.e.* horizontally or vertically) do not affect the SVD computation since one pattern could be transferred to another through matrix transpose [7]. In this paper, we assume $\mathbf{X}$ is vertically split. Following previous works [7, 9, 10, 17, 25, 34, 35], we define the problem of federated SVD as computing Equation (1):

$$[\mathbf{X_1}; \mathbf{X_2}; ...; \mathbf{X_k}] = \mathbf{U\Sigma}[\mathbf{V_1^T}; \mathbf{V_2^T}; ...; \mathbf{V_k^T}] \tag{1}$$

where k is the number of peers. After the federated SVD computation, $\mathbf{U}, \mathbf{\Sigma}$ are shared among the peers, and $\mathbf{V_i^T}$ are privately possessed by each peer.

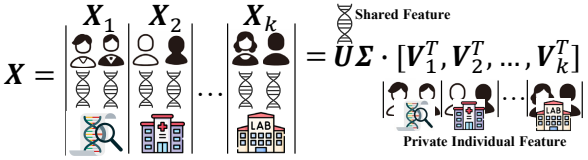


Figure 1: Example of federated SVD on genetic data.

We present real-world examples to demonstrate the practicality and validity of the problem definition. In genetic analysis, as illustrated in Figure 1, the matrix $\mathbf{X}$ represents genetic data, where different columns contain phenotype measurements on one certain DNA segment of different samples (*e.g.*, individuals). After the federated decomposition, the left singular vectors $\mathbf{U}$ characterize publicly shared DNA phenotype patterns summarized from all individuals. The right singular vectors $\{\mathbf{V}_i^T\}_{1\le i\le k}$ are private representations for each

individual, which can help the correction of population stratification in downstream genetic association studies [10, 25]. In latent semantic analysis (LSA), the matrix $\mathbf{X}$ represents word-document data, where different columns are word frequencies across different documents. After federated SVD, the left singular vectors $\mathbf{U}$ represent publicly shared word embeddings, while the right singular vectors $\{\mathbf{V}_i^T\}_{1\leq i\leq k}$ represent private document representations [7].

2.2 External Servers Downgrade the Privacy Protection in Federated SVD

Most pioneered federated SVD works rely heavily on external servers [7, 9, 10, 25, 34]. As shown in Table 1, the data contributors in these systems only perform simple operations and outsource complicated computations (*e.g.*, the complete SVD) to external servers. Using external servers reduces system design complexity by outsourcing complex SVD computations to servers rather than performing joint computations among data contributors.

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

Table 1: Workload summary of data contributors and external servers in existing server-aided federated SVD systems. The servers obtain excessive access to the private data and thus significantly decrease the systems' privacy protection.

However, the involvement of external servers results in a significant degradation of the privacy protections of the system. This is due to the fact that such servers, which are not data contributors and lack a root of trust, receive a substantial amount of sensitive information during system execution. Specifically, FedSVD [7] protects clients' data using random masks and leverages two servers that are responsible for generating the random masks (*i.e.*, masking server) and performing computation on the masked data (*i.e.*, computation server), respectively. During SVD computation, the clients upload masking-protected data to the computation server, while the collusion between these two servers will completely leak the raw data; [9] leverages several fog devices as servers to perform the factorization but directly expose the covariance matrices (*i.e.*, $\mathbf{XX^T}, \mathbf{X^TX}$) to the servers, which can recover the raw data after simple calculations; [10] designed a system of conducting genetic studies through secret sharing the data on three external servers, while the collusion between any two servers will completely leak the raw data; [34] and [25] followed conventional federated learning protocol to design the system, in which the clients jointly upload projections of raw data to one server, which iteratively orthonormalize these projections. However, the projections are sent to the server without protection, which could be utilized to recover the raw data or launch attacks (*e.g.*, membership inference attacks [63]). Table 1 summarizes the external servers' job and the corresponding threats of privacy leakage.

From the perspective of real-world applications, involving external servers limits the potential use cases. Institutions that handle highly sensitive data, such as banks and genetic research organizations, usually prefer federated SVD without involving the external servers [37, 41, 45, 50, 53]. This is due to the fact that they prefer not to place trust in external servers and their unwillingness to compromise privacy preservation by relying on potentially untrustworthy servers, given strict supervision and privacy restrictions.

Intuitive ideas of enhancing privacy protection in server-aided approaches are impractical. Regarding the privacy issue brought by external servers, there are intuitive ideas to enhance privacy protection. Next, we will discuss these ideas and their limitations.

- Pick data contributors to work as the "servers" instead of involving external parties. Existing server-aided federated SVD systems rely on $1 \sim 4$ external servers. One naive idea is directly selecting $1 \sim 4$ data contributors working as the servers. However, this idea will not work since privacy issues still exist among the selected data contributors, *i.e.*, private data is directly leaked to these contributors or leaked when they collude. The system workflow does not change in the view of non-selected data contributors, and their privacy concerns still exist.
- Deploy trusted execution environment (TEE) at the servers to increase privacy protection. TEE can provide encrypted memory and secure code execution at the external servers to improve privacy protection. However, this idea has the following issues, making it not practical in the decentralized SVD system: 1) TEE requires additional hardware in the system, while we want to build a lightweight and plug-and-play system with low adaption cost; 2) the limited memory of TEE making it not suitable for federated SVD system. Typical TEE system offers limited secure memory region (*e.g.*, 92MB for Intel SGX [2, 56]) to store the data while federated SVD deals with matrices much larger. For example, more than 1TB of storage is required to store billion-scale genetic data and perform the decomposition [7]. Although some modern TEE systems can support up to 1TB encrypted memory [32], it is limited to very specific hardware, which clearly increases the adaption cost; 3) the issue of distrust, particularly in the server-aided approach, poses a significant challenge. TEE is susceptible to various attacks [40], including software-based, side-channel, and architectural attacks. Such vulnerabilities raise privacy concerns and instill a lack of trust in the data contributors,

even when TEE is employed [56]. Meanwhile, the security of TEEs relies on trusted assumptions, such as the integrity of the underlying hardware and firmware. However, data contributors have reservations about trusting TEE, as they are aware of the potential for servers to compromise the supply chains, compromise TEE integrity, and ultimately gain access to private data.

- Leverage HE to protect the computation at the servers. Similar to existing studies that explored HE to remove the external servers [17, 35], leveraging HE to protect the computation at external servers suffers from severe computation overheads. We will discuss the efficiency issue of HE-based SVD in more detail in §2.3.

Conclusion. Existing server-aided federated SVD studies involve external servers, which significantly downgrade the systems' privacy protection. Intuitive ideas of enhancing the protection in these systems (*e.g.*, using HE or TEE) are impractical. To solve this problem, it is essential to eliminate the servers and achieve a decentralized system.

2.3 Efficient Decentralization is Challenging

Existing federated SVD studies [17, 35] have explored leveraging the strong protection of HE to remove the servers, and the data contributors could peer-to-peer exchange encrypted sensitive data without the servers. However, they suffer from a significant efficiency downgrade brought by HE. Designing an efficient decentralized federated SVD system raises both computation and communication challenges.

Computation challenge. SVD involves a substantial amount of computations, such as matrix multiplication. According to [20], the complexity of SVD for a $m \times n$ matrix is $O(mn \cdot min(m,n))$. Although existing studies [17, 35] have shown the viability of leveraging HE to achieve a decentralized solution, HE-based SVD [35] is $4 \sim 5$ orders of magnitudes slower than server-aided approach [7] and centralized SVD (*i.e.*, collecting data centrally without considering privacy protection), which is presented in Figure 2. HE's efficiency issue will be particularly severe in large-scale SVD since the amount of computation increases cubically with the data size. Existing work has shown that HE-based solutions require more than 15 years (single CPU with eight cores) to factorize million-scale matrices, while real-world applications (*e.g.*, genetic studies) usually deal with billion-scale data [7].

Meanwhile, accelerating HE operations in SVD using hardware is also challenging. First, we cannot continuously improve the efficiency through stacking more hardware since the computation process of SVD consists of a sequence of orthogonal transformations that need to be computed in sequential order since one transformation relies on the result of the last transformation [20]. Second, designing new hardware accelerators for HE-based SVD is also challenging. SOTA hardware accelerator for HE-based FL [60] can improve the multiplicative operations by $14\times$ compared to CPU, while HE-based SVD is $4 \sim 5$ orders slower.

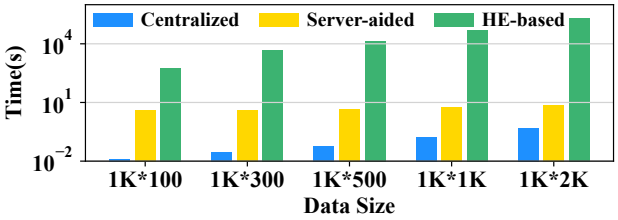


Figure 2: HE-based solution [35] is $4 \sim 5$ orders of magnitudes slower than server-aided [7] and centralized SVD (*i.e.*, collecting data centrally without considering privacy).

Communicaiton challenge. Decentralized SVD itself causes severe communication inefficiency since it involves enormous numerical operations across the distributed matrices, which causes a large peer-to-peer communication size and rounds. Meanwhile, the HE-based approach has more communication overheads since HE encrypts data from 64 to 2048 bits (or more). Next, we analyze the communication cost in detail.

- Overhead of communication size. Existing HE-based approaches [17, 35] achieve decentralized SVD through leveraging the strong protection of HE to collect enough data (*e.g.*, the covariance matrix) at all peers to perform the decomposition locally. Thus, the communication size will be at least the same size as the raw data to successfully perform SVD. In real-world applications that deal with billion-scale data, *e.g.*, $1K \times 50M$ matrix [7], the communication size of each peer will be 372GB before encryption and 11.6TB after encrypted by a 2048-bits key, which causes significant communication overhead.
- Overhead of communication rounds. To reduce the communication size, we can design a decentralized SVD protocol to only peer-to-peer exchange necessary intermediate results during the execution rather than directly gathering all the raw data. However, this approach will cause a large amount of communication rounds. Briefly, we transfer the most popular SVD algorithm, the two-side bidiagonalization with divide-and-conquer bidiagonal SVD, which is widely used in packages like LAPACK, NumPy, and SciPy, into decentralized system (§5). We find that the communication size could be reduced from $O(mn)$ to $O(m \cdot min(m,n))$, which is superior when $n \gg m$. However, it increases the communication rounds to $4m(k-1)$, which is $O(km)$. In particular, when $m = 10,000$, $k = 10$ (*i.e.*, 10 participants), and the RTT is 50ms, the total overhead caused by network latency will be 2.5 hours, while the SVD computation could be done in minutes.

Optimization targets of decentralized federated SVD. We formulate the target of decentralized federated SVD system as reducing the cost C in Equation (2).

$$\min(C) = C_{comp} + C_{comm} \quad s.t., \begin{cases} ||\mathbf{X}_i - \mathbf{U}\mathbf{\Sigma}\mathbf{V}_i^T|| \leq \varepsilon \\ \boldsymbol{I}(\mathbf{v}_i) \leq \boldsymbol{I}(\mathbf{X}, \mathbf{U}, \mathbf{\Sigma}, \mathbf{V}_i^T) \end{cases} \tag{2}$$

where C_{comp} and C_{comm} are the additional computation and communication cost brought by decentralization and privacy protection, ε is a small positive number constraining the error of SVD results. We denote v_i as the viewed message of peer-i during the system execution, and $\boldsymbol{I}(m)$ is a function to measure the information of m. Informally, restricting $\boldsymbol{I}(v_i) \leq \boldsymbol{I}(\mathbf{X}, \mathbf{U}, \boldsymbol{\Sigma}, \mathbf{V}_i^T)$ represents that the viewed message of peer-i contains no more information than its final local outputs. This is the intuitive idea of protecting privacy in secure multi-party computation [11]. We give formal security definition and threat model in §3.1. Overall, Equation (2) illustrates a big picture regarding how to define the utility and security, and our design target is reducing the additional computation and communication cost while satisfying the security and utility requirements.

3 System Overview

3.1 Threat Model and Security Goals

Threat model. Following previous work [7, 9, 10, 17, 25, 34, 35], we assume all peers (*i.e.*, the data contributors) are semi-honest, which is one of the most widely-used assumptions in federated learning [59]. It means that the peers will correctly follow the protocol but try to reveal private data by analyzing the received messages. We consider the threat that the adversaries can compromise up to $k-1$ peers and see all of their internal states (*e.g.*, local private data, memory contents, access patterns, and data sent/received) but without altering its execution, where k is the total number of peers.

Security definition. Since federated SVD is a typical secure multi-party computation (SMPC) system, we follow the definition in SMPC [11, 27] to define the security, which is presented in Definition 1. This definition is also used by many widely known SMPC protocols, *e.g.*, garbled circuits and secret sharing [11, 27]. It ensures that adversaries with certain colluding subsets of participants can learn nothing from executing the system that they could not have learned from the outputs [27].

Definition 1 (Security Definition)**.** *Let $\{x_i, y_i\}_{1 \leq i \leq k}$ be the input (x_i) and output (y_i) of a multi-party computation system, where k is the number of participants. Let $\{v_i\}_{1 \leq i \leq k}$ be the viewed messages during the system execution. Considering the adversary compromises C ($|C| < k$) participants, then the system is secure if there exists an efficient simulator S such that $S(\{x_i, y_i\}_{i \in C})$ has the same distribution with $\{v_i\}_{i \in C}$.*

Attacks out of scope. Similar to other federated SVD studies [7, 9, 10, 17, 25, 34, 35], the following attacks are out of scope for our study: 1) attacks of revealing private data from final results. Protecting final results is out of the scope of SMPC [11, 27]. However, if the peers wish to protect the final results, Excalibur can work with differential privacy (DP) to achieve such guarantees, and more details are discussed in §9; 2) malicious attacks by not following the protocol. By assuming semi-honest, we expect all peers will correctly follow the system's protocol, which is reasonable as the data contributors in real-world SVD applications, *e.g.*, genetic research institutions and banks, usually do not have the motivation to be malicious since it harms their reputations if being caught. Nevertheless, we provide discussions showing it is feasible to defend against malicious attacks in Excalibur (§9).

3.2 Overview of Excalibur

Figure 3 shows an overview of Excalibur, and its design has the following two components to address the computational and communication challenges.

Computation-efficient matrix protection (§4). Instead of using HE, Excalibur proposes a computational-efficient protection method for decentralized federated SVD, called multiplicative matrix sharing (MMS). MMS uses multiplicative operations to randomly rotate and project each peer's raw data into multiple low-dimensional shares held by different peers separately (§4.1). Intuitively, MMS achieves protection as recovering any certain peer's raw data requires shares from all peers. We provide rigorous proof showing that Excalibur, under the protection of MMS, satisfies the security defined in Definition 1. MMS achieves much higher efficiency than HE because HE typically enlarges the data (*e.g.*, from 64 to 2048 bits) and involves time-consuming operations (*e.g.*, bootstrapping [1, 61]), while MMS keeps the data size unchanged. Meanwhile, regarding the additional multiplicative operations when creating the shares, we propose optimizations to improve the efficiency when dealing with large-scale data (§4.2).

Communication-efficient decentralized SVD (§5). Excalibur achieves superior communication efficiency as we have thoughtfully designed the decentralized system to transfer only necessary messages, and we address latency issues by overlapping the pipelines. Compared with directly gathering all required data, Excalibur significantly decreases the communication size. Compared with designing a decentralized system by empirically selecting one SVD solver, we comprehensively analyze the design space of the decentralized SVD system and choose the path with the minimum communication complexity (§5.1). Regarding latency overhead, we carefully overlap the system pipelines, reducing communication rounds by 66% (§5.2).

4 Excalibur's Matrix Protection

In this section, we first introduce multiplicative matrix sharing (MMS) (§4.1), the efficient matrix protection in Excalibur, and then present optimizations to decrease MMS's computational overhead when processing large-scale data (§4.2).

4.1 Multiplicative Matrix Sharing

Excalibur proposes MMS based on the following observations: 1) HE protects matrix data by encrypting each number or vector, which overlooks the data's inherent structure. Given

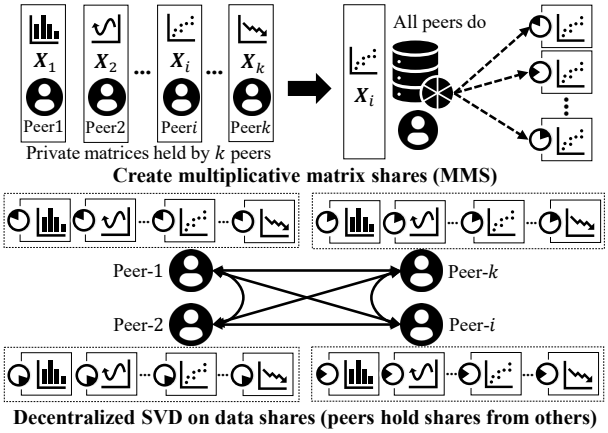


Figure 3: System overview of Excalibur.

that SVD consistently deals with matrix data rather than individual numbers or vectors, we can develop efficient techniques to protect the matrix as a whole, *e.g.*, randomly rotating and stretching the matrix. 2) In SMPC, a common strategy for protecting data while enabling joint computation is creating data shares among the participants [11]. Specifically, each party's local data is divided into additive shares held by different parties. A single party cannot recover the data as the recovery requires shares from multiple parties.

Excalibur employs multiplicative operations to randomly rotate and project the raw data into different low-dimensional shares, *i.e.*, the MMS, which is formally defined in Definition 2. Specifically, the random matrix $\mathbf{B}$ randomly rotates $\mathbf{X}$ and $\mathbf{A}$ projects the rotated matrix into different vectors. These vectors are MMS of $\mathbf{X}$ and recovering $\mathbf{X}$ require all $\{\mathbf{a_i XB}\}_{1\le i\le m}$ as well as the random matrices $\mathbf{A},\mathbf{B}$, *i.e.*, $\mathbf{X}=\mathbf{A}^{-1}\{\mathbf{AXB}\}\mathbf{B}^{-1}$.

Definition 2 (Multiplicative Matrix Sharing)**.** *Let* $\mathbf{X}\in\mathbb{R}^{m\times n}$ *be the private data,* $\mathbf{A}\in\mathbb{R}^{m\times m}$ *and* $\mathbf{B}\in\mathbb{R}^{n\times n}$ *are random inversible matrices (*i.e.*, non-singular). We define the MMS of* $\mathbf{X}$ *as different rows of* $\mathbf{AXB}$*, which are* $\{\mathbf{a_i XB}\}_{1\le i\le m}$ *and* $\mathbf{a_i}$ *is the i-th row of* $\mathbf{A}$.

Now we adopt MMS into decentralized SVD, in which the private data $\mathbf{X}=[\mathbf{X_1},\mathbf{X_2},\ldots,\mathbf{X_i},\ldots,\mathbf{X_k}]$ are jointly held by k peers. Following the notation in Definition 2, we denote $\mathbf{B}=diag([\mathbf{B_1},\ldots,\mathbf{B_i},\ldots,\mathbf{B_k}])$ are the local random rotations independently generated and performed by the peers, and $\mathbf{A}$ is the global random projection matrix. Plugging them into $\mathbf{AXB}$, we get Equation (3), which is formulated into a $k\times k$ block matrix, where $\mathbf{A}=[\mathbf{A_1}^T,\ldots,\mathbf{A_i}^T,\ldots,\mathbf{A_k}^T]^T$. In the $k\times k$ block matrix, each block represents one group of data share (*i.e.*, multiple vectors) of a certain peer. After the data sharing, we let peer-i hold the following blocks $\{\mathbf{A_i X_p B_p}\}_{1\le p\le k}$, which is exactly the i-th row of block matrices in Equation (3). Then all peers jointly perform SVD over the data shares. MMS brings some extra communication while creating the data shares, *i.e.*, peers need to distribute local shares to others. However, the size of extra communication for each peer is proportional to its local data size rather than the overall data size. Furthermore, the communication in MMS could be implemented in parallel.

$$\mathbf{AXB}=\begin{bmatrix}\mathbf{A_1X_1B_1} & \mathbf{A_1X_2B_2} & \ldots & \mathbf{A_1X_iB_i} & \ldots & \mathbf{A_1X_kB_k}\\ \mathbf{A_2X_1B_1} & \mathbf{A_2X_2B_2} & \ldots & \mathbf{A_2X_iB_i} & \ldots & \mathbf{A_2X_kB_k}\\ \vdots & \vdots & & \vdots & & \vdots\\ \mathbf{A_iX_1B_1} & \mathbf{A_iX_2B_2} & \ldots & \mathbf{A_iX_iB_i} & \ldots & \mathbf{A_iX_kB_k}\\ \vdots & \vdots & & \vdots & & \vdots\\ \mathbf{A_kX_1B_1} & \mathbf{A_kX_2B_2} & \ldots & \mathbf{A_kX_iB_i} & \ldots & \mathbf{A_kX_kB_k}\end{bmatrix} \quad (3)$$

Empirical Evidence why MMS achieves the protection. Empirically, MMS achieves protection for two reasons: 1) each peer's data is projected into k shares held by different peers and recovering one certain peer's data require all the shares as well as the local rotation matrix $\mathbf{B_i}$; 2) decentralized SVD peer-to-peer exchanges orthogonal transformations which are also randomized by $\mathbf{A}$ and $\{\mathbf{B_i}\}_{1\le i\le k}$ while creating the data shares.

Formal proof of MMS. Theorem 1 shows that Excalibur protected by MMS can produce SVD results with no accuracy loss and satisfy the security definition of SMPC (*i.e.*, Definition 1). Uniformly generating random matrices (*e.g.*, $\mathbf{B_i}$) from the orthogonal compact group under Haar measure could be achieved by QR decomposition on random matrices with standard Gaussian distributed values [3, 47]. The proof of no accuracy loss is quite straightforward since orthonormal rotations of a matrix could be reverted from the SVD results. The security proof is more complicated since it also depends on the subsequent system workflow. Briefly, we conduct a detailed mathematical analysis showing that we can efficiently compute the distribution of all intermediate results in Excalibur from the federated SVD outputs, which is precisely Definition 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 (*$|C|=k-1$*).*

Proof. We put the proof in §6 since it also involves the subsequent decentralized SVD workflow of Excalibur. □

Difference to other multiplicative matrix protections. Existing studies also have explored protecting data by multiplicating with random matrices, such as random masks [5, 7, 38, 55]. However, these studies are limited to outsourced [38, 55] or server-aided [5, 7] scenarios. While these studies employ random masks to protect computations at external servers, Excalibur leverages multiplicative operations to generate matrix shares among different peers, thereby achieving an efficient decentralized system. Thus, Excalibur has an entirely different system workflow and design target.

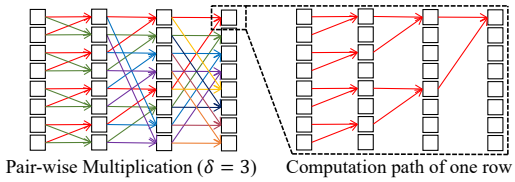
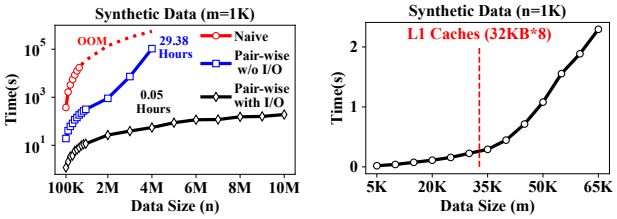


Figure 4: An example of 3 times of pair-wise multiplication and the computation path of one row.



(a) Without I/O optimization, the effectiveness of pair-wise multiplication vanishes as data scale increases.

(b) Time consumption increases dramatically when one column of data exceeds total L1 caches.

Figure 5: Experiments that guide the design of efficient multiplicative operations.

4.2 Accelerating the Multiplicative Operations

According to Theorem 1, the multiplicative operation involves generating orthogonal matrices $\mathbf{A}, \{\mathbf{B_i}\}_{1\le i\le k}$ and multiply them with raw data $\mathbf{X}$. However, generating and multiplying orthogonal matrices have a large computation overhead when dealing with large-scale data. SOTA method [47] costs $O(m^2)$ complexity to generate $\mathbf{A} \in \mathbb{R}^{m\times n}$ and multiplying $\mathbf{A}$ with $\mathbf{X} \in \mathbb{R}^{m\times n}$ has $O(m^2 n)$ complexity. Thus, the overall complexity is $O(m^2 n)$, which is computationally inefficient when m is large. Next, we introduce Excalibur's optimizations to accelerate these multiplicative operations.

Acceleration of protection (*i.e.*, Matrix A). Instead of generating and applying $\mathbf{A}$, we randomly group rows of data matrix (*i.e.*, $\mathbf{X}$) into pairs (*i.e.*, $m/2$ pairs in total) and multiply the pairs using random 2×2 orthogonal matrices. Then, we repeat this pair-wise multiplication δ times. This idea works since the multiplication result of orthogonal matrices remains orthogonal. Figure 4 illustrates an example when $\delta = 3$ and the computation path of one row from the results. We do not explicitly form $\mathbf{A}$, but gradually apply many 2×2 matrices to the data. Theoretically, we only need to perform $\delta = log_2(m)$ times of pair-wise multiplication to make each dimension of $\mathbf{A}$ not sparse, *i.e.*, each row is mixed with all the other rows. Each time of pair-wise multiplication has $O(mn)$ complexity. Thus, the proposed approach reduces the complexity of generating and multiplicating $\mathbf{A}$ from $O(m^2 n)$ to $O(mn\cdot log(m))$, which is significantly faster on large-scale data.

The proposed pair-wise multiplication reduces complexity but causes repeated loading of different rows of data from memory to CPU cache and back. It brings significant I/O overhead and becomes a bottleneck for large-scale data. Figure 5 shows that the efficiency advantage of pair-wise multiplication decreases with larger data scales. To solve this problem, we propose two optimizations:

- We process data by columns instead of by rows. Each column is accessed only once and written back after all of its computations are finished. This transforms pair-wise multiplication from I/O-intensive to computation-intensive.
- When the data is very large-scale (*e.g.*, billion-scale), loading one column of data can be inefficient if it exceeds the size of CPU L1 caches. To address this, we reduce the required data in each computation episode by forming one column of data into a rectangular matrix and then recursively processing each column of the new rectangular matrix. This optimization allows us to load only $\sqrt{m}$ data instead of m, enabling efficient computation when the size of $\mathbf{A}$ is up to $1B\times 1B$.

Acceleration of local rotation (*i.e.*, Matrix $\mathbf{B_i}$). According to Theorem 1, the random matrix $\mathbf{B}$ should follow uniform distribution over the compact orthogonal group. We follow existing work [47] to generate $\mathbf{B}$ that satisfies the distribution requirement. Overall, generating and multiplicating $\mathbf{B_i}$ with $\mathbf{X_i}$ has $O(mn_i^2)$ complexity, which becomes the efficiency bottleneck when the local matrix is short and wide, *i.e.*, $m \ll n_i$. We leverage data pre-processing to improve efficiency. Specifically, we locally reduce the dimension of the data through QR decomposition such that $\mathbf{X_i} = \mathbf{R_i^T Q_i^T}$ where $\mathbf{R_i} \in \mathbb{R}^{m\times m}$ and $\mathbf{Q_i} \in \mathbb{R}^{m\times n_i}$. Then, peers can use $\mathbf{R_i}$ instead of $\mathbf{X_i}$ in the SVD computation. After the SVD computation, each peer can apply $\mathbf{Q_i}$ back to private local singular vectors $\mathbf{V_i^T}$. Thus, the pre-processing does not harm the SVD's accuracy. Notably, the peer only needs to generate and multiply $\mathbf{B_i}$ to $\mathbf{R_i}$, which is far more efficient than working with $\mathbf{X_i}$. The complexity of QR decomposition is $O(m^2 n_i)$. Thus, the pre-processing strategy reduces the complexity from $O(mn_i^2)$ to $O(m^2 n_i + m^3)$, significantly reducing the computation overhead when $m \ll n_i$.

5 Excalibur's Decentralized SVD Workflow

To solve the communication challenge, Excalibur first comprehensively analyzes the design space of decentralized SVD and selects the path with the least communication cost (§5.1). Furthermore, Excalibur overlaps the pipelines to reduce 66% of communication rounds (§5.2). This section may involve some linear algebra concepts, and we have prepared preliminaries of necessary computations in Appendix A.

5.1 Analyzing the Design Space

In this section, we first analyze the categorization of SVD solvers that constitute the design space of decentralized SVD, then analyze the complexity of each path in that space, and finally, we present Excalibur's decentralized SVD workflow.

Existing SVD solvers can be broadly classified into two categories: 1) methods without bidiagonalization, *e.g.*, power iteration [34, 46] and Jacobi iteration [12]. These methods directly use orthogonal transformations to iteratively eliminate the off-diagonal elements; 2) methods with bidiagonalization, *e.g.*, Golub-Kahan SVD [20], divide-and-conquer SVD [22], and bisection-twisted SVD [54]. These methods have two

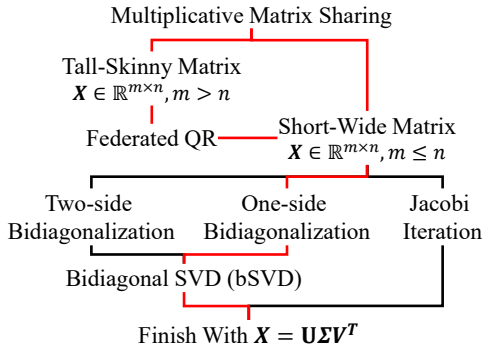


Figure 6: Design space of decentralized SVD. The path used in Excalibur is indicated in red color.

steps: first, the matrix is bidiagonalized such that only the diagonal and sub-diagonal positions contain non-zero elements, and second, used orthogonal transformations to eliminate the sub-diagonal elements, which is also known as bidiagonal SVD (bSVD). In this paper, we analyze the communication complexity of four methods from both categories, which are Jacobi iteration [12], Golub-Kahan SVD [20], divide-and-conquer SVD [22], and bisection-twisted SVD [54].

Design space of decentralized SVD. The design space of adopting those conventional SVD solvers into the decentralized system is illustrated in Figure 6, and it is based on the following observations:

- Different shapes of input matrix may have very different efficiency. For clarity, we name two types of data shapes: tall-skinny matrix (*i.e.*, $m > n$) and short-wide matrix (*i.e.*, $m \leq n$). We observe that tall-skinny matrices tend to have more communication overhead than short-wide matrices due to more complex data interactions. Thus we use decentralized QR decomposition to transfer tall-skinny matrix to short-wide matrices, *i.e.*, all peers jointly exchange householder reflectors to reduce the matrix. Due to the space limitation, we put the detailed workflow of decentralized QR in Appendix B.
- Golub-Kahan SVD [20], divide-and-conquer SVD [22], and bisection-twisted SVD [54] can share the same bidiagonalization computation and only differ at the bidiagonal SVD (*i.e.*, bSVD). Meanwhile, the bidiagonalization accounts for most of the computation and communication [20], thus we mainly focus on analyzing two different bidiagonalization methods: the two-side [20] and one-side [44] approaches.

In summary, as illustrated in Figure 6, the design space has the following three paths: 1) Jacobi iteration; 2) two-side bidiagonalization with bSVD; 3) one-side bidiagonalization with bSVD. The bSVD could be one of these three methods: Golub-Kahan SVD, divide-and-conquer SVD, and bisection-twisted SVD. We only analyze the complexity under short-wide matrices for simplicity since the tall-skinny matrices share the same computation path after federated QR.

Results of analyzing the design space. Table 2 shows the analyzing results and we put the detailed computation in Appendix C due to the space limitation. It is worth noting that our analyzing results are calculated after necessary optimizations, including packing all intermediate results together if they could be sent in one round of communication and avoiding transferring redundant data (*e.g.*, only transfer half of the matrix if it is symmetric). The analysis shows that:

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	$\frac{k-1}{k}(m^2-m)$ $\Rightarrow O(m^2)$	$(6m-6)(k-1)$ $\Rightarrow O(km)$

Table 2: Quantitative communication analysis of three decentralized SVD paths.

- Jacobi iteration has significantly higher communication rounds (*i.e.*, $O(km^2)$) compared to the other two methods (*i.e.*, $O(km)$), leading to more network latency overhead. Meanwhile, Jacobi iteration usually requires more than one iteration to converge, affected by the data distribution, and the increased iterations can further cause large overhead regarding communication size.
- Compared to the one-side approach, the two-side bidiagonalization with bSVD is the most popular method for centralized SVD (*e.g.*, used by NumPy, LAPACK, *etc.*), but its communication size is large and increases with the number of peers (*i.e.*, $O(km^2)$), making it unsuitable for decentralized SVD.
- The one-side bidiagonalization approach has the minimum complexity of communication size (*i.e.*, $O(m^2)$) and communication rounds (*i.e.*, $O(km)$), making it the best choice for Excalibur.

Excalibur's decentralized SVD workflow. Based on the analysis, Excalibur selects the one-side bidiagonalization with bSVD as the most efficient path, which is also indicated in Figure 6. We present the detailed workflow in Algorithm 1. Since Excalibur's decentralized workflow is designed on MMS, the final SVD results are also in the form of data shares between the peers. The recovery of these data shares is a straightforward process, and we put detailed procedures in Appendix E.

5.2 Overlapping the Pipelines

To further reduce the communication rounds, we perform an in-depth study on Algorithm 1. We find that Algorithm 1 involves three looped ring all-reduce communications, which are highlighted and could be further optimized through overlapping the system 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 $\mathbf{X} = [\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_k}]$ held by k peers, where $\mathbf{X} \in \mathbb{R}^{m \times n}$, $m \leq n$, $\mathbf{X_i} \in \mathbb{R}^{m \times n_i}$, and $\sum_{i=1}^{k} n_i = n$.
Output: $\mathbf{U}, \mathbf{\Sigma}, [\mathbf{V_1^T}, \mathbf{V_2^T}, ..., \mathbf{V_k^T}]$ (*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.
  $\alpha \leftarrow \{0\}^m, \beta \leftarrow \{0\}^{m-1}$
  $\alpha[1] \leftarrow \sqrt{RingAllReduce(||X_c[1]||_2^2)}$
  $\mathbf{V_c^T}[1] = \mathbf{X_c}[1]/\alpha[1]$
  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 $\mathbf{U}, \mathbf{\Sigma}, [\mathbf{V_1^T}, \mathbf{V_2^T}, ..., \mathbf{V_k^T}]$
End Function

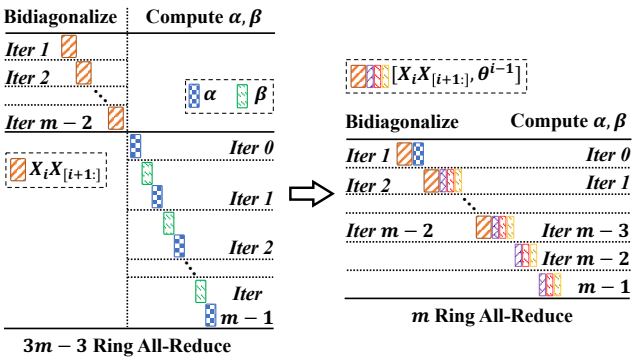


Figure 7: Overlapping the pipeline reduces the number of ring all-reduce communication from $3m-3$ to m, *i.e.*, approximately reduced by 66%.

We first analyze the third loop ring all-reduce in Algorithm 1, which could be represented by Equation (4):

$$\alpha_i^2 = \sum_{j=1}^{k} ||\mathbf{X_j^i}||_2^2 = \sum_{j=1}^{k} ||\mathbf{X_j^i} - \beta_{i-1}(\mathbf{V_j^{i-1}})^{\mathbf{T}}||_2^2 \tag{4}$$

Plugging in $\beta_{i-1} = \sum_{l=1}^{k} \mathbf{X_l^i V_l^{i-1}}$, which is exactly the second looped ring all-reduce communication, and we get:

$$\begin{aligned} \alpha_i^2 &= \sum_{j=1}^{k} ||\mathbf{X_j^i} - (\sum_{l=1}^{k} \mathbf{X_l^i V_l^{i-1}})(\mathbf{V_j^{i-1}})^{\mathbf{T}}||_2^2 \\ &= \sum_{j=1}^{k} \mathbf{X_j^i}(\mathbf{X_j^i})^{\mathbf{T}} - 2(\sum_{l=1}^{k} \mathbf{X_l^i V_l^{i-1}})(\sum_{j=1}^{k} \mathbf{X_j^i V_j^{i-1}}) + \\ &\quad (\sum_{l=1}^{k} \mathbf{X_l^i V_l^{i-1}})^2 \sum_{j=1}^{k} [(\mathbf{V_j^{i-1}})^{\mathbf{T}} \mathbf{V_j^{i-1}}] \end{aligned} \tag{5}$$

If we denote $\theta_1^i = \sum_{j=1}^{k} \mathbf{X_j^i}(\mathbf{X_j^i})^{\mathbf{T}}$, $\theta_2^i = \sum_{j=1}^{k} \mathbf{X_j^i V_j^{i-1}}$, $\theta_3^i = \sum_{j=1}^{k} [(\mathbf{V_j^{i-1}})^{\mathbf{T}} \mathbf{V_j^{i-1}}]$, then α_i, β_{i-1} could be represented as:

$$\alpha_i = \sqrt{\theta_1^i - 2(\theta_2^i)^2 + (\theta_2^i)^2 \theta_3^i}, \quad \beta_{i-1} = \theta_2^i \tag{6}$$

Thus we can compute α_i and β_{i-1} through only one time of ring all-reduce communication on $\theta^i = [\theta_1^i, \theta_2^i, \theta_3^i]$.

Additionally, the computation of α_i and β_{i-1} require bidiagonalized X_i which is computed by the first looped ring all-reduce at iteration i, thus we can parallel these two pipelines by computing α_i and β_{i-1} in a delayed manner, *i.e.*, computing α_i, β_{i-1} in iteration $i+1$. Figure 7 illustrates the pipelines before and after the optimization, and our optimization reduces 66% ring all-reduce communication.

6 Security Analysis

Now we formally prove Theorem 1.

Proof. (**Accuracy**) The proof of no accuracy loss is quite straightforward. The random rotation and projection are orthogonal transformations that could be inverted from SVD results. Thus Excalibur does not impact model accuracy. (**Security**) When the adversary compromises $C = \{peer_i\}_{1 \leq i \leq k-1}$ and try to attack peer-k, the adversary holds the input and output $\{\mathbf{A}, \mathbf{X}_i, \mathbf{B}_i, \mathbf{U}, \mathbf{\Sigma}, \mathbf{V_i^T}\}_{i \in C}$. The adversary views the following messages from peer-k: 1) the data share from peer-k: $\{\mathbf{A_i X_k B_k}\}_{1 \leq i \leq k-1}$; 2) partial implicit matrix of one-side bidiagonalization: $\{\mathbf{B_k^T X_k^T A_k^T A_k X_i B_i}\}_{1 \leq i \leq k}$. These two groups of messages correspond to the two stages of Excalibur's computation: creating MMS and decentralized SVD.

Pick $\mathbf{V}_{null} \in \mathbb{R}^{n \times n_k}$ as an orthogonal base in the null space of the subspace spanned by $\{\mathbf{V_i^T}\}_{i \in C}$. Then $\mathbf{V}_{null}\mathbf{R_1}$, where $\mathbf{R_1}$ uniformly distributed in $\mathbb{O}_{n_k}$, uniformly distributed on all the possible orthogonal base on the corresponding null space. According to the definition of SVD, $\mathbf{V}_k^T$ is also an orthogonal base in the null space of the subspace spanned by $\{\mathbf{V_i^T}\}_{i \in C}$. Meanwhile, $\mathbf{B}_k$ is uniformly distributed in $\mathbb{O}_{n_k}$. Thus we have $\mathbf{V}_{null}\mathbf{R}_1 \overset{d}{=} \mathbf{V}_k^T \mathbf{B}_k$, *i.e.*, they have the same distribution which is uniformly distributed over all possible orthogonal bases. Since $\mathbf{A}_i \mathbf{X_k B}_k = \mathbf{A}_i \mathbf{U \Sigma V}_k^T \mathbf{B}_k$, we can deduce:

$$\mathbf{A}_i \mathbf{X}_k \mathbf{B}_k \overset{d}{=} \mathbf{A}_i \mathbf{U \Sigma V}_{null} \mathbf{R}_1 \quad 1 \leq i \leq k-1 \tag{7}$$

Similarly, we can deduce that:

$$\mathbf{B_k^T X_k^T A_k^T A_k X_i B_i} \overset{d}{=} \begin{cases} \mathbf{R}_1^T \mathbf{V}_{null}^T \mathbf{\Sigma}^T \mathbf{U}^T \mathbf{A}_k^T \mathbf{A}_k \mathbf{X}_i \mathbf{B}_i & 1 \le i \le k-1 \\ \mathbf{R}_1^T \mathbf{V}_{null}^T \mathbf{\Sigma}^T \mathbf{U}^T \mathbf{A}_k^T \mathbf{A}_k \mathbf{U} \mathbf{\Sigma} \mathbf{V}_{null} \mathbf{R}_1 & i = k \end{cases} \quad (8)$$

We can build a simulator S based on Equation (7) and Equation (8) that produces messages with the same distribution to the viewed messages from peer-k during the system execution. Specifically, we can generate $\mathbf{V}_{null}$ through Gram–Schmidt process and uniformly generate $\mathbf{R_1} \in \mathbb{O}_{n_k}$ following [47]. Thus, Excalibur satisfies security defined in Definition 1 while the adversary compromises $k-1$ peers. □

7 Implementation

We implement a fully functional prototype of Excalibur using C/C++. Peers communicate with others through sockets. We use BLAS and LAPACKE from Intel MKL [52] as the major library for matrix computation and use double precision (*i.e.*, 64bit) in the whole system.

Parallelism. For computations not included in BLAS and LAPACKE, *e.g.*, computation for creating MMS, we implement from scratch and use OMP [8] and AVX2 for parallelism. For computations included in BLAS and LAPACKE, we directly use multi-threading and SIMD from MKL library. Implementations of baseline methods also leverage similar parallelism, *e.g.*, [7] used NumPy with MKL, which already includes the above parallel methods.

Handling large-scale data. For very large-scale matrices that cannot fit into memory, we create memory-mapped files such that the large matrices are physically stored on disk. Then, we schedule I/O such that the frequently used data are loaded to RAM before computation and written back only when all computations in certain steps are finished.

Choice of δ. δ is the number of pair-wise multiplication times when applying the matrix $\mathbf{A}$ in MMS (§4.2). We find setting $\delta = 32$ can support data up to $(2^{32})^2 = 2^{64}$ rows or columns, which is enough to support any real-world datasets with satisfactory computation efficiency. Thus, we directly set $\delta = 32$ in all evaluations.

Applications based on SVD. We implement three applications based on SVD results in Excalibur, including principal component analysis (PCA), latent semantic analysis (LSA), and linear regression (LR). We put the detailed workflow in Appendix F.

8 Evaluation

In this section, we comprehensively evaluate Excalibur, and the key results are:

- Compared to the SOTA server-aided system, Excalibur not only removes the external servers but also achieves better efficiency. Briefly, Excalibur is $3.1\times \sim 6.0\times$ faster than FedSVD [7] on different shapes of billion-scale data and reduces more than 68.4% amount of communication.
- 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 [36] and SecureML [39], Excalibur is 100x and 1000x faster, respectively.

8.1 Experimental Setup

Testbed setup. To simulate different network settings, we follow previous work [7] and put all peers in separated docker containers, which are connected using the docker bridge network, and use traffic control tools to change the bandwidth and latency between the containers. Each container (*i.e.*, one peer) uses 4 CPU cores and 64GB of memory. Unless stated otherwise, following previous work [7], the default bandwidth and round trip time (RTT) between the peers used in our evaluation is 1Gbps and 50ms.

Datasets, baselines, and tasks. We have used four datasets in our evaluation: MNIST [33], Wine [13], ML100K [24], and synthetic data [21]. We evaluate Excalibur on SVD tasks and three applications (*i.e.*, PCA, LSA, and LR), and compare it with: FedSVD [7] which is the SOTA server-aided federated SVD system that leverages random masks as protection, and SF-PCA [17] which is the SOTA multi-key HE based SVD system. Regarding the LR application, we also compare Excalibur with two most popular federated LR systems: FATE [36] which leverages partial HE as protection and SecureML [39] which is an SS-based system and relies on two non-colluding servers. Due to the space limitation, we put detailed parameters of datasets and baselines in Appendix D.

8.2 High Accuracy

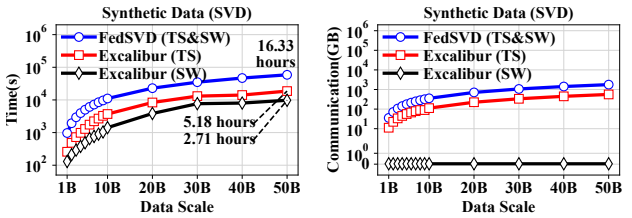
Before diving into efficiency evaluations, we would like to first demonstrate Excalibur's high accuracy after the overall computations. Table 3 shows the reconstruction error (*i.e.*, $||\mathbf{X} - \mathbf{U\Sigma V}^T||_2$) on SVD of Excalibur and FedSVD [7]. Excalibur achieves very high accuracy with only $10^{-13} \sim 10^{-17}$ reconstruction error on different datasets. We only report the accuracy of SVD since SVD-based applications (*i.e.*, PCA, LSA, and LR) will also have high accuracy if SVD is accurate.

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.

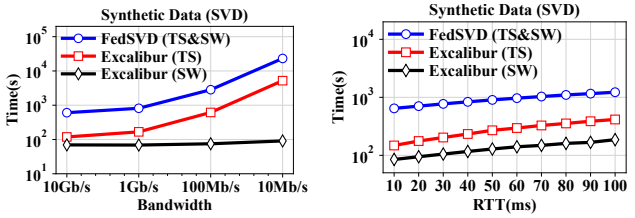
8.3 Efficiency on SVD Task

Compaing to SOTA server-aided system on SVD. To demonstrate the high efficiency of Excalibur, we compare it with FedSVD which is the SOTA server-aided federated SVD



(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.

Figure 8: Comparing Excalibur and FedSVD on SVD task using billion-scale tall-skinny (TS) and short-wide (SW) data.

system. The evaluation results are illustrated in Figure 8. We use billion-scale synthetic data which is uniformly partitioned on two peers, and specify two types of data shapes: the tall-skinny (TS) and short-wide (SW) matrices. For the TS matrix, we set $n = 1K$ and vary m from 1M to 50M to control the data scale from 1B to 50B. Similarly, we set $m = 1K$ for the SW matrix and change n to control the data scale. Excalibur has different optimizations for these two data shapes thus has different time consumption which are denoted as Excalibur (TS) and Excalibur (SW) in Figure 8, while FedSVD does not specify matrix shapes and has identical performance under TS and SW matrices. The evaluation results show that Excalibur not only eliminates the privacy concerns brought by involving the external servers but also has better efficiency than FedSVD on billion-scale data. Specifically, 1) on billion-scale tall-skinny matrices, Excalibur is 3.1x faster than FedSVD and reduces 68.4% communication amount; 2) on billion-scale short-wide matrices, Excalibur is 6.0x faster than FedSVD and reduces 99.9% communication amount; 3) Excalibur consistently has better performance than FedSVD under different network bandwidth and latency.

Superior performance on short-wide matrices. It is worth noting that Excalibur has superior performance and very low communication amount on short-wide matrix, as illustrated in Figures 8(a) and 8(b), because we choose the most communication-efficient decentralized SVD path. According to our analysis (*i.e.*, Table 2), the communication amount under the short-wide matrix is $\frac{k-1}{k}(m^2-m)$ which is only correlated with m and does not change if the peers horizontally increase more data (*i.e.*, increasing n).

Analyzing why Excalibur performs better than FedSVD. We perform a fine-grained performance comparison between Excalibur and FedSVD, and Figure 9 shows the result. If we

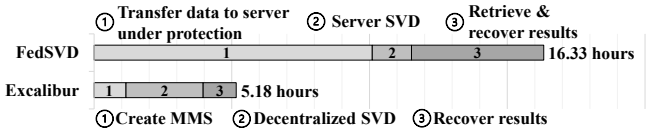


Figure 9: Fine-grained efficiency comparison between Excalibur and FedSVD on 50B tall-skinny data (Figure 8(a)). For easy comparison, we divide both systems into three phases: 1) applying protection, 2) SVD, and 3) revealing final results.

compare the SVD computation process solely (*i.e.*, ignoring the overhead of protecting the data), the server-aided approach does run faster than decentralized SVD, which is reasonable since decentralized SVD involves peer-to-peer communication. However, the server-aided approach requires collecting all data on the server before the computation, which brings large communication overhead and raises privacy concerns. Technically, Excalibur outperforms the SOTA server-aided system (*i.e.*, FedSVD) for two reasons: 1) Excalibur has better communication efficiency. It thoroughly analyzes the design space of decentralized SVD, ensuring that only necessary intermediate results are exchanged from peer to peer. In contrast, clients in FedSVD roughly upload all the masked data to the servers. As a result, Excalibur has 68.4%∼99.9% less communication size than FedSVD. 2) Excalibur has better computational efficiency. FedSVD involves many additional computations for privacy preservation, including random masking and secure aggregation, while Excalibur only leverages multiplicative operation to create MMS. Moreover, the complexity of random masking in FedSVD is $O(mnk)$,where k is a hyper-parameter to balance the privacy protection and computation complexity and they set $k = 1000$. The complexity of creating MMS in Excalibur is $O(mnlog(m))$ on the tall-skinny matrix. Excalibur has lower computation complexity (*i.e.*, $log(m) < k$). In summary, Excalibur outperforms FedSVD in both computation and communication efficiency. We also notice that the computational optimizations for creating MMS in Excalibur could be adopted in FedSVD to improve efficiency. However, the communication optimizations cannot be used in FedSVD since they are specifically proposed for decentralized systems.

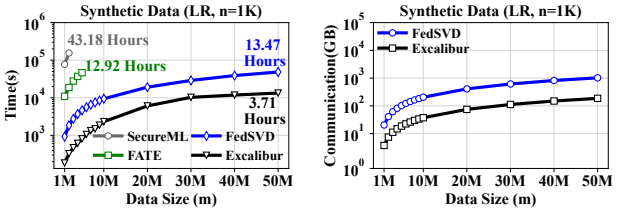
8.4 Efficiency on SVD Applications

Comparing to SOTA HE-based system on PCA. Table 4 shows the results of Excalibur and SF-PCA on PCA task. Although SF-PCA also does not rely on external servers, it has significant efficiency overhead caused by HE. Excalibur has >23000x larger throughput compared with SF-PCA.

Comparing to SOTA server-aided system on LR. Figure 10 illustrates the LR evaluation results. The time consumptions reported for FATE and SecureML are the running time of one epoch and it is worth noting that they usually require many epochs to coverage, while Excalibur only needs one factorization to get final results. The results show that Excalibur is 1000× faster than SecureML (one epoch), 100× faster than

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

Table 4: 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.

Figure 10: Comparing Excalibur with FedSVD, FATE, and SecureML on LR application.

FATE (one epoch), and 3.6× faster than FedSVD on billion-scale data. Excalibur consistently has smaller communication size on different data scales and reduces the communication by 81.7% on 50B data compared with FedSVD.

8.5 Scalability

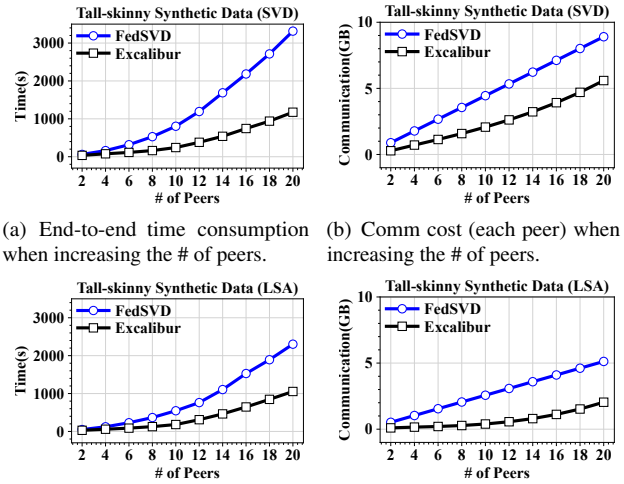
We evaluate Excalibur's scalability when increasing the number of peers. We assume all peers hold the same amount of data and test the efficiency when more peers join the federation. Figure 11 presents results on the SVD task and LSA application when each peer holds a 100K×250 matrix and the number of peers increases from 2 to 20. This setting simulates real-world genetic studies in which peers hold large-scale gene features (*i.e.*, 100K) of a few samples (*i.e.*, 250), and they cooperate to increase the sample size. The results show that Excalibur has better scalability than the FedSVD and is continuously more efficient when increasing the # of peers.

8.6 Effectiveness of the Optimizations

To measure the effectiveness of the proposed optimizations, we report Excalibur's end-to-end time consumption with and without these optimizations, and the results are presented in Figure 12. We divided the total system runtime into computation and communication parts. Regarding the communication, overlapping pipeline (*i.e.*, Opt2) stably reduces about 60% of the communication time under different data scales. The optimization of accelerating multiplicative operations in MMS (*i.e.*, Opt1) becomes more significant when the scale of data increases. Overall, when the data scale is 1B (100K×10K), using both optimizations reduces the total runtime by 73.8%.

9 Discussion

Excalibur and DP. Federated SVD systems protected by DP [21] have different security goals from our system as they



(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.

Figure 11: Scalability in SVD task and LSA application.

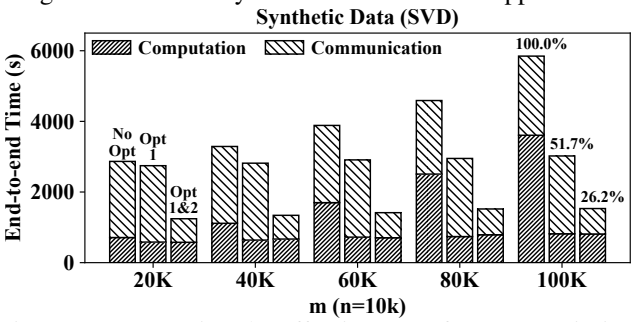


Figure 12: Measuring the effectiveness of system optimizations in Excalibur, while NoOpt means no optimization, Opt1 is optimizing the multiplicative operations in MMS, and Opt 2 is overlapping pipelines to reduce communication rounds.

introduce unremovable noises to protect both intermediate and final results but make the final results less accurate. Excalibur proposes MMS as protection and thus does not need DP during the computation. However, if the peers wish to protect the final results, Excalibur can follow [21] to work with DP and achieve such guarantees.

Excalibur and HE. The semantic security of HE, as defined in cryptography, ensures that no information is leaked from the ciphertext. Consequently, HE can be utilized in various scenarios beyond SMPC, such as outsourced computation. Excalibur is primarily designed as an SMPC system and is focused on the security definition in SMPC (*i.e.*, Definition 1). In the SMPC scenario, Excalibur achieves the same level of security as HE-based solutions because they both satisfy Definition 1. It is important to note that if we were to extend part of Excalibur's idea, such as MMS, to support scenarios beyond SMPC, such as outsourced computation [51], it would be necessary to enhance the security to match the strength of HE's definition.

Defending malicious attacks in Excalibur. We assume the data contributors are semi-honest, meaning they will follow

the system protocol correctly. Now we show it is feasible to defend the malicious attacks (*i.e.*, not following the protocol) in Excalibur. The main idea is that Excalibur produces lossless SVD results. Thus, the correctness of the workflow could be locally verified by checking whether $\mathbf{X_i}$ equals to $\mathbf{U\Sigma V}_i^T$. If some peers tamper with data shares or messages during decentralized SVD, any peer could locally find the federated SVD results do not match local data, *i.e.*, $\mathbf{X_i} \neq \mathbf{U\Sigma V}_i^T$.

10 Related Work

Apart from the studies introduced in §2, there are also other research topics that are closely related to our work:

Randomized SVD. Randomized SVD was first proposed by [23] in 2011 to improve the SVD efficiency by randomly projecting the data from high dimensions to low dimensions but sacrificing the accuracy and numerical stability. Excalibur focuses on SVD that produces accurate and stable results. However, Excalibur could be adapted to randomized SVD by adding one random projection procedure.

Large-scale sparse SVD. SVD on large-scale sparse matrices usually has different computation workflow and application scenarios (*e.g.*, recommender systems [31]) compared to SVD on large-scale dense matrices. Many studies have explored the federated factorization of sparse matrices in RecSys [6, 58]. In this paper, we focus on the SVD of large-scale dense matrices, and large-scale sparse SVD will be covered in our future work.

11 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. Extensive evaluations show that Excalibur effectively achieves its design targets.

Acknowledgments

We thank the anonymous reviewers for their constructive suggestions. This work is supported by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001), 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) [57]. Junxue Zhang and Kai Chen are the corresponding authors.

References

[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

[2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13. ACM New York, NY, USA, 2013.

[3] Theodore W Anderson, Ingram Olkin, and Les G Underhill. Generation of random orthogonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(4):625–629, 1987.

[4] K Aswani and D Menaka. A dual autoencoder and singular value decomposition based feature optimization for the segmentation of brain tumor from mri images. *BMC medical imaging*, 21(1):1–11, 2021.

[5] Jianping Cai, Ximeng Liu, Zhiyong Yu, Kun Guo, and Jiayin Li. Efficient vertical federated learning method for ridge regression of large-scale samples. *IEEE Transactions on Emerging Topics in Computing*, 2022.

[6] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. Secure federated matrix factorization. *IEEE Intell. Syst.*, 36(5):11–20, 2021.

[7] Di Chai, Leye Wang, Junxue Zhang, Liu Yang, Shuowei Cai, Kai Chen, and Qiang Yang. Practical lossless federated singular vector decomposition over billion-scale data. In *KDD*, pages 46–55. ACM, 2022.

[8] Rohit Chandra, Leo Dagum, Ramesh Menon, David Kohr, Dror Maydan, and Jeff McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[9] Shuo Chen, Rongxing Lu, and Jie Zhang. A flexible privacy-preserving framework for singular value decomposition under internet of things environment. In *IFIPTM*, volume 505 of *IFIP Advances in Information and Communication Technology*, pages 21–37. Springer, 2017.

[10] Hyunghoon Cho, David J Wu, and Bonnie Berger. Secure genome-wide association analysis using multiparty computation. *Nature biotechnology*, 36(6):547–551, 2018.

[11] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[12] Zlatko Drmač and Krešimir Veselić. New fast and accurate jacobi svd algorithm. i. *SIAM Journal on matrix analysis and applications*, 29(4):1322–1342, 2008.

[13] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[14] Jia Duan, Jiantao Zhou, and Yuanman Li. Secure and verifiable outsourcing of large-scale nonnegative matrix factorization (NMF). *IEEE Trans. Serv. Comput.*, 14(6):1940–1953, 2021.

[15] Susan T. Dumais. Latent semantic analysis. *Annu. Rev. Inf. Sci. Technol.*, 38(1):188–230, 2004.

[16] Susan T Dumais, George W Furnas, Thomas K Landauer, Scott Deerwester, and Richard Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–285, 1988.

[17] D. Froelicher, H. Cho, M. Edupalli, J. Sa Sousa, J. Bossuat, A. Pyrgelis, J. R. Troncoso-Pastoriza, B. Berger, and J. Hubaux. Scalable and privacy-preserving federated principal component analysis. In *IEEE Symposium on Security and Privacy (SP)*, 2023.

[18] Anmin Fu, Zhenzhu Chen, Yi Mu, Willy Susilo, Yinxia Sun, and Jie Wu. Cloud-based outsourcing for enabling privacy-preserving large-scale non-negative matrix factorization. *IEEE Trans. Serv. Comput.*, 15(1):266–278, 2022.

[19] Dimitris Gavalas and Theodore Syriopoulos. Bank credit risk management and rating migration analysis on the business cycle. *International Journal of Financial Studies*, 2(1):122–143, 2014.

[20] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[21] Andreas Grammenos, Rodrigo Mendoza-Smith, Jon Crowcroft, and Cecilia Mascolo. Federated principal component analysis. In *NeurIPS*, 2020.

[22] Ming Gu and Stanley C Eisenstat. A divide-and-conquer algorithm for the bidiagonal svd. *SIAM Journal on Matrix Analysis and Applications*, 16(1):79–92, 1995.

[23] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[24] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2016.

[25] Anne Hartebrodt, Reza Nasirigerdeh, David B. Blumenthal, and Richard Röttger. Federated principal component analysis for genome-wide association studies. In *ICDM*, pages 1090–1095. IEEE, 2021.

[26] Kanji Hasegawa, Satoru Goto, Chihiro Tsunoda, Chihiro Kuroda, Yuta Okumura, Ryosuke Hiroshige, Ayako Wada-Hirai, Shota Shimizu, Hideshi Yokoyama, and Tomohiro Tsuchida. Using singular value decomposition to analyze drug/β-cyclodextrin mixtures: insights from x-ray powder diffraction patterns. *Physical Chemistry Chemical Physics*, 25(42):29266–29282, 2023.

[27] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *IEEE Symposium on Security and Privacy*, pages 1220–1237. IEEE, 2019.

[28] Chien-Ta Bruce Ho and Desheng Dash Wu. Online banking performance evaluation using data envelopment analysis and principal component analysis. *Computers & Operations Research*, 36(6):1835–1842, 2009.

[29] Xinyang Huang, Junxue Zhang, Xiaodian Cheng, Hong Zhang, Yilun Jin, Shuihai Hu, Han Tian, and Kai Chen. Accelerating privacy-preserving machine learning with genibatch. In *EuroSys*, pages 489–504. ACM, 2024.

[30] Suhyeon Kim, Haecheong Park, and Junghye Lee. Word2vec-based latent semantic analysis (w2v-lsa) for topic modeling: A study on blockchain technology trend analysis. *Expert Systems with Applications*, 152:113401, 2020.

[31] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[32] Marc Lacoste and Vincent Lefebvre. Trusted execution environments for telecoms: Strengths, weaknesses, opportunities, and threats. *IEEE Security & Privacy*, 2023.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[34] Xiang Li, Shusen Wang, Kun Chen, and Zhihua Zhang. Communication-efficient distributed SVD via local power iterations. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 6504–6514. PMLR, 2021.

[35] Bowen Liu and Qiang Tang. Privacy-preserving decentralised singular value decomposition. In *ICICS*, volume 11999 of *Lecture Notes in Computer Science*, pages 703–721. Springer, 2019.

[36] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. FATE: an industrial grade platform for collaborative learning with data protection. *J. Mach. Learn. Res.*, 22:226:1–226:6, 2021.

[37] Songtao Lu, Yawen Zhang, and Yunlong Wang. Decentralized federated learning for electronic health records. In *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5. IEEE, 2020.

[38] Changqing Luo, Kaijin Zhang, Sergio Salinas, and Pan Li. Secfact: Secure large-scale qr and lu factorizations. *IEEE Transactions on Big Data*, 7(4):796–807, 2017.

[39] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society, 2017.

[40] Antonio Muñoz, Ruben Ríos, Rodrigo Román, and Javier López. A survey on the (in) security of trusted execution environments. *Computers & Security*, 129:103180, 2023.

[41] Yang Pei, Renxin Mao, Yang Liu, Chaoran Chen, Shifeng Xu, Feng Qiang, and Blue Elephant Tech. Decentralized federated graph neural networks. In *International Workshop on Federated and Transfer Learning for Data Sparsity and Confidentiality in Conjunction with IJCAI*, 2021.

[42] Chakan Pramkaew and Sudsanguan Ngamsuriyaroj. Lightweight scheme of secure outsourcing SVD of a large matrix on cloud. *J. Inf. Secur. Appl.*, 41:92–102, 2018.

[43] Alkes L Price, Nick J Patterson, Robert M Plenge, Michael E Weinblatt, Nancy A Shadick, and David Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nature genetics*, 38(8):904–909, 2006.

[44] Rui Ralha. One-sided reduction to bidiagonal form. *Linear Algebra and its Applications*, 358(1-3):219–238, 2003.

[45] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):119, 2020.

[46] Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

[47] Gilbert W Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, 17(3):403–409, 1980.

[48] Ovidiu Stoica, Seyed Mehdian, and Alina Sargu. The impact of internet banking on the performance of romanian banks: Dea and pca approach. *Procedia Economics and Finance*, 20:610–622, 2015.

[49] Huilin Tan, Zhen Zhang, Xin Liu, Yiming Chen, Zinuo Yang, and Lei Wang. Mdsvdnv: Predicting microbe-drug associations by singular value decomposition and node2vec. *Frontiers in Microbiology*, 14:1303585.

[50] Bernardo Camajori Tedeschini, Stefano Savazzi, Roman Stoklasa, Luca Barbieri, Ioannis Stathopoulos, Monica Nicoli, and Luigi Serio. Decentralized federated learning for healthcare networks: A case study on tumor segmentation. *IEEE Access*, 10:8693–8708, 2022.

[51] Han Tian, Chaoliang Zeng, Zhenghang Ren, Di Chai, Junxue Zhang, Kai Chen, and Qiang Yang. Sphinx: Enabling privacy-preserving online learning over the cloud. In *SP*, pages 2487–2501. IEEE, 2022.

[52] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, Yajuan Wang, Endong Wang, Qing Zhang, Bo Shen, et al. Intel math kernel library. *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*, pages 167–188, 2014.

[53] Stefanie Warnat-Herresthal, Hartmut Schultze, Krishnaprasad Lingadahalli Shastry, Sathyanarayanan Manamohan, Saikat Mukherjee, Vishesh Garg, Ravi Sarveswara, Kristian Händler, Peter Pickkers, N Ahmad Aziz, et al. Swarm learning for decentralized and confidential clinical machine learning. *Nature*, 594(7862):265–270, 2021.

[54] Paul R Willems, Bruno Lang, and Christof Vömel. Computing the bidiagonal svd using multiple relatively robust representations. *SIAM journal on matrix analysis and applications*, 28(4):907–926, 2006.

[55] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 139–152, 2009.

[56] Pengfei Wu, Jianting Ning, Jiamin Shen, Hongbing Wang, and Ee-Chien Chang. Hybrid trust multi-party computation with trusted execution environment. In *NDSS*. The Internet Society, 2022.

[57] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. TACC: A full-stack cloud computing infrastructure for machine learning tasks. *CoRR*, abs/2110.01556, 2021.

[58] Liu Yang, Ben Tan, Vincent W. Zheng, Kai Chen, and Qiang Yang. Federated recommendation systems. In *Federated Learning*, volume 12500 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2020.

[59] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2):12:1–12:19, 2019.

[60] Junxue Zhang, Xiaodian Cheng, Wei Wang, Liu Yang, Jinbin Hu, and Kai Chen. {FLASH}: Towards a high-performance hardware acceleration architecture for cross-silo federated learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1057–1079, 2023.

[61] Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Ximeng Liu, and Kai Chen. Sok: Fully homomorphic encryption accelerators. *CoRR*, abs/2212.01713, 2022.

[62] Yushu Zhang, Xiangli Xiao, Lu-Xing Yang, Yong Xiang, and Sheng Zhong. Secure and efficient outsourcing of pca-based face recognition. *IEEE Trans. Inf. Forensics Secur.*, 15:1683–1695, 2020.

[63] Jie Zhu, Leye Wang, and Xiao Han. Safety and performance, why not both? bi-objective optimized model compression toward ai software deployment. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

Appendix

A Preliminaries

In this section, we review the necessary linear algebra computations used in our design.

Givens Rotation and Householder Reflector Givens rotation and householder reflectors are techniques used to zero selected components of a vector through orthogonal transformations [20]. They only rotate the vector and keep the vector's norm unchanged. Equation (9) shows an definition of givens rotation (*i.e.*, $[[c,s],[-s,c]]$) that rotates the vector $[a,b]^T$ such that the second element (*i.e.*, one specified entry) becomes zero. Given a,b, there are algorithms [20] that allow us to compute c,s that satisfies Equation (9).

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sqrt{a^2+b^2} \\ 0 \end{bmatrix} \tag{9}$$

While Givens rotation can zero one selected entry in one rotation, the householder reflector can zero at most $n-1$ entries of size n vector in one transformation. The householder reflector is defined as $\mathbf{P} = \mathbf{I} - \beta \mathbf{v}\mathbf{v}^\mathbf{T}$, where I is identity matrix, $\mathbf{v} \in \mathbb{R}^n$ is the householder vector, $\beta = 2/\mathbf{v}^\mathbf{T}\mathbf{v}$ is a scaler, and $\mathbf{P}$ is a $n \times n$ orthogonal matrix. Applying householder reflector to a vector $\mathbf{x}$ is multiplying $\mathbf{x}$ with $\mathbf{P}$, *i.e.*, $\mathbf{Px}$. By setting $\mathbf{v} = house(x) = \mathbf{x} - ||\mathbf{x}||_2\mathbf{e_1}$, where $\mathbf{e_1} = [1,0,...,0]^T$, $\mathbf{P}$ can zero all the element in $\mathbf{x}$ except for the first one, which is presented in Equation (10). It is worth noting that there is more than one way to compute the householder vector given $\mathbf{x}$ (*i.e.*, $house(\mathbf{x})$), and a more practical solution is provided in [20].

Equation (10) shows the computation of applying householder reflector $\mathbf{P} = \mathbf{I} - \beta \mathbf{v}\mathbf{v}^\mathbf{T}$ to vector $\mathbf{x}$, where $\mathbf{v}$ is the householder vector and $\beta = 2/(\mathbf{v}^\mathbf{T}\mathbf{v})$. Equation (10) holds when setting $\mathbf{v} = \mathbf{house}(\mathbf{x}) = \mathbf{x} - ||\mathbf{x}||_2\mathbf{e_1}$, *i.e.*, the householder vector that zeros all the element in $\mathbf{x}$ except for the first one. The detailed computation inside $\mathbf{house}()$ function could be found in [20].

$$\mathbf{Px} = (\mathbf{I} - \beta \mathbf{v}\mathbf{v}^\mathbf{T})\mathbf{x} = \begin{bmatrix} ||\mathbf{x}||_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad s.t. \;\; \mathbf{v} = \mathbf{house}(\mathbf{x}) \tag{10}$$

Givens rotation is more flexible and preferred when it is necessary to zero elements more selectively [20], and householder reflector tends to have lower computation complexity and is preferred when plenty of continuous entries in one vector need to be zeroed (*e.g.*, QR decomposition).

QR Decomposition QR decomposition transfer matrix $\mathbf{X}$ into $\mathbf{X} = \mathbf{QR}$ where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper diagonal matrix. Figure 13 illustrates the process of performing QR decomposition using householder reflectors, which gradually zero elements below the diagonal.

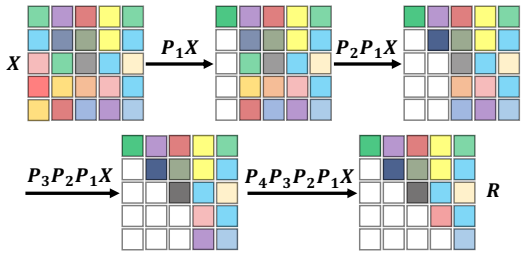


Figure 13: An illustration of using householder reflectors to perform QR decomposition and $\mathbf{Q} = \mathbf{P_1^T P_2^T P_3^T P_4^T}$.

Bidiagonalization Bidiagonalization reduces matrix $\mathbf{X}$ to bidiagonal form, which only contains non-zero values at diagonal and subdiagonal positions. Figure 14 illustrates the process of two-side bidiagonalization [20], which uses householder reflectors to gradually eliminate the elements outside the diagonal and subdiagoanl positions. Apart from the two-side bidiagonalization, another type of approach is one-side bidiagonalization [44]. The main idea of one-side bidiagonalization is firstly implicitly tridiagonalizing the matrix $\mathbf{XX^T}$, then explicitly forming the bidiagonal matrix using a modified Gram-Schmidt algorithm, and the detail could be found in [44].

B Decentralized Federated QR Decomposition

Algorithm 2 presents the detailed workflow of decentralized federated QR decomposition.

C Detailed Analysis of the Design Space

Base operations Assuming we have k peers. During the quantitative analysis, we will use the communication complexity

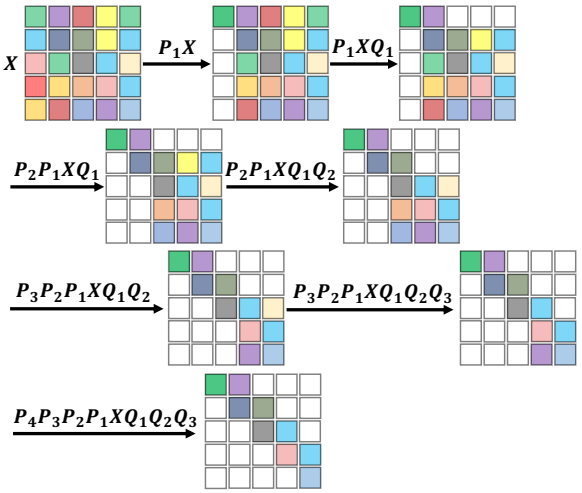


Figure 14: An illustration of two-side bidiagonalization using householder reflectors.

of the following four basic operations as the primitives:

- CommTimes of one ring all-reduce: $rl = 2(k-1)$.
- CommSize of one ring all-reduce on size t message: $rb(t) = 2t(k-1)/k$.
- CommTimes of left householder reflector: $hl = (k-1)$.
- CommuSize of left householder reflector with size t: $hb(t) = t(k-1)$.

Jacobi iteration Jacobi iteration gradually eliminates the off-diagonal values of $\mathbf{X}\mathbf{X}^\mathbf{T}$ in a implicit manner, *i.e.*, $\mathbf{X}\mathbf{X}^\mathbf{T}$ is not explicitly computed. One single Jacobi rotation requires $\mathbf{X_i}\mathbf{X_i^T}, \mathbf{X_i}\mathbf{X_j^T}, \mathbf{X_j}\mathbf{X_j^T}$ which needs one round of ring all-reduce communication on three numbers, and each Jacobi iteration needs $\frac{m(m-1)}{2}$ Jacobi rotations.

Thus, the communication size of one Jacobi iteration is:

$$\frac{m(m-1)}{2} * rb(3) = 3\frac{k-1}{k}(m^2-m) \tag{11}$$

The communication time of one Jacobi iteration is:

$$\frac{m(m-1)}{2} * rl = m(m-1)(k-1) \tag{12}$$

Two-side bidiagonalization + bSVD The two-side bidiagonalization involves both left and right householder reflectors. The analysis of left householder reflectors is quite simple: each peer locally computes the vector and broadcasts it to the others through AllGather. The analysis of right householder reflectors is complicated since it involves column-wise data exchange between adjacent peers. As illustrated in fig. 15, we use a normal householder reflector inside each peer and leverage Givens rotation when crossing the peers. Each time crossing the peer on size t element costs three times $2t+2$ communication. During the whole two-side bidiagonalization, it requires m times of right householder reflectors and $m(k-1)$ times of transmission between the peers.

Algorithm 2: Decentralized QR via Householder Reflectors for Tall-Skinny Matrices.

Input: Matrix $\mathbf{X} = [\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_k}]$ held by k peers, where $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{X_i} \in \mathbb{R}^{m \times n_i}$, and $\sum_{i=1}^{k} n_i = n$.
Output: Decompose $[\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_k}]$ to $\mathbf{Q}[\mathbf{R_1}, \mathbf{R_2}, ..., \mathbf{R_k}]$, where $[\mathbf{R_1}, \mathbf{R_2}, ..., \mathbf{R_k}]$ overall is an upper triangular matrix.

Function `DecQR(X)`**:**
  // All peers run this function in parallel
  $counter \leftarrow 0$, $\mathbf{Q} \leftarrow \mathbf{I}$
  for i = *1, 2, ..., k* **do**
    if i == *MyPeerID* **then**
      // Local QR using householder reflectors H
      $[\mathbf{H}, \mathbf{R_i}] \leftarrow \mathbf{X_i}[\mathbf{counter}:]$ ▷ Rows after counter
      Broadcast($\mathbf{H}$)
    else
      Receive($\mathbf{H}$) ▷ Receive from peer-i
      // Apply $\mathbf{H}$ to $\mathbf{X_i}$
      $\mathbf{X_i}[\mathbf{counter}:] \leftarrow \mathbf{H} \otimes \mathbf{X_i}[\mathbf{counter}:]$
    end
    // Apply $\mathbf{H}$ to $\mathbf{Q}$
    $\mathbf{Q}[\mathbf{counter}:] \leftarrow \mathbf{H} \otimes \mathbf{Q}[\mathbf{counter}:]$
    $counter \leftarrow counter + n_i$
  end
  return $\mathbf{Q}, [\mathbf{R_1}, \mathbf{R_2}, ..., \mathbf{R_k}]$
End Function

Overall, the two-side bidiagonalization + bSVD has the following amount of communication:

$$\sum_{t=1}^{m} hb(t) + (k-1)\sum_{j=0}^{m-1}(2(m-j)+2) = (k-1)(\frac{3}{2}m^2 + \frac{7}{2}m) \tag{13}$$

And the following times of communication:

$$m * hl + 3 * (k-1)m = 4m(k-1) \tag{14}$$

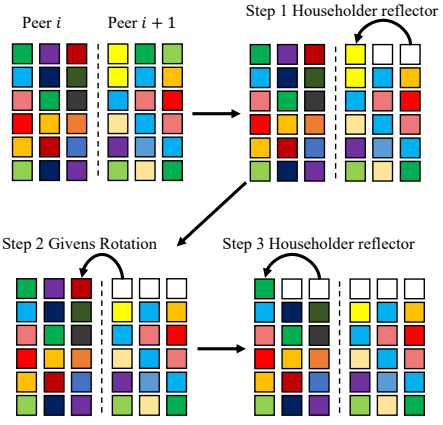


Figure 15: An illustration of right-side householder reflectors when crossing the adjacent peers.

One-side bidiagonalization and bSVD The one-side bidi-

agonalization only involves left householder reflectors and additional communication for computing the bidiagonal matrix (*i.e.*, α and β).

The total communication amount of one-side bidiagonalization and bSVD is:

$$\sum_{t=1}^{m-1} rb(t) = \frac{k-1}{k}(m^2 - m) \tag{15}$$

The total communication times are:

$$(m-1) * rl + (2m-2) * rm = (6m-6)(k-1) \tag{16}$$

D Parameters of Datasets and Baselines

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

- MNIST [33]: MNIST is a standard hand-written digits image dataset containing 70K samples, and each image contains 784 (i.e., 28×28) features.
- Wine [13]: The physicochemical data for 6498 variants of red and white wine, and each sample has 12 features.
- MovieLens (ML100K) [24]: MovieLens dataset consists of people's ratings on different movies. ML100k contains 943 users' rating on 1682 movies.
- Synthetic data [21]: Apart from the real-world datasets, we also use one synthetic data in the experiment. 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: Following are the parameter settings for baseline work:

- FedSVD [7]: FedSVD is a lossless and efficient federated SVD solution which leverages random masks as protection. Following the setting in their paper, we set the block size of random masks to 1000. For all the experiments on FedSVD, we directly use the open-sourced code published by the author of [7].
- SF-PCA [17]: SF-PCA is the SOTA HE-based federated PCA. Since their code is not open-sourced, we directly use the results from their paper and compare the throughput of a single CPU core between SF-PCA and Excalibur under the same number of peers and network settings. SF-PCA runs the experiments on 6 machines, each with a 12-Cores CPU and 256GB RAM.
- FATE [36] and SecureML [39]: For FATE and SecureML, we use the default parameters provided by their systems, and use the same logic for controlling the network conditions, *i.e.*, deploying their peers in separated containers and use traffic control (tc) tool to change the bandwidth and latency.

E Recovering the SVD Results

Since Excalibur's decentralized workflow is designed on MMS, the final SVD results are also in the form of data shares between the peers. Now, we show how to recover the final results from the data shares. Specifically, each peer only possesses low-dimensional shares of the left singular vectors (*i.e.*, $\mathbf{U}$). Thus, getting $\mathbf{U}$ requires data shares from all the peers. The $\mathbf{V_i^T}$ could be recovered by peer i locally through $\mathbf{V_i^T B_i^T}$ since $\mathbf{V_i}$ is locally possessed. To recover $\mathbf{U}$, the peers all gather $\mathbf{A_i U}$, and vertically concatenate to get $\mathbf{AU}$. Then recover $\mathbf{U}$ through $\mathbf{U} = \mathbf{A^T} \cdot \mathbf{AU}$.

F Workflow of PCA, LSA, and LR

These three applications have the same workflow as SVD and only differ in processing the final results (*e.g.*, LR needs to compute the parameters based on the results of SVD).

Workflow of PCA and LSA. After running the decentralized SVD protocol, the peers all gather top-r columns of $\mathbf{A_i U_r}$ and vertically concatenate to get $\mathbf{AU_r}$ which is $m \times r$. Then obtain the results through $\mathbf{U_r} = \mathbf{A^T} \cdot \mathbf{AU_r}$. All peers can recover $\mathbf{V_i^T}$ locally without communication.

Workflow of LR. LR requires the model parameters based on the results of SVD. In this paper, we focus on vertical LR since it is the most popular application in banks [7]. We assume m is the sample dimension, n is the feature dimension and peer k holds all the labels. Given matrix $\mathbf{X}$, LR tries to find parameter matrix $\mathbf{w}$ such that $\mathbf{y} = \mathbf{Xw}$. If we factorize $\mathbf{X}$ into $\mathbf{U\Sigma V^T}$, then $\mathbf{w} = \mathbf{V\Sigma^{-1}U^T y}$. Since $\mathbf{X}$ is jointly hold by k parties, then $[\mathbf{w_1^T}, \mathbf{w_2^T}, ..., \mathbf{w_k^T}]^\mathbf{T} = [\mathbf{V_1^T}, \mathbf{V_2^T}, ..., \mathbf{V_k^T},]^\mathbf{T} \mathbf{\Sigma^{-1} U^T y}$, where $\mathbf{w_i^T}$ is the parameter held by peer-i. Our protocol for privately compute $\mathbf{w_i^T}$ has three steps: 1) peer k protect the label vector $\mathbf{y}$ using $\mathbf{A}$ into $\mathbf{y'} = \mathbf{Ay}$, split vector $\mathbf{y'}$ into k shares, and send $\mathbf{y'_i}$ to peer-i; 2) all peers compute the randomized parameter $\mathbf{w'} = \mathbf{B^T V\Sigma^{-1} U^T y} = \mathbf{B^T V\Sigma^{-1}} \sum_{j=1}^{k} \mathbf{U^T A_j^T A_j y} = \sum_{j=1}^{k} (\mathbf{B^T V\Sigma^{-1} U^T A_j^T A_j y})$. Denote $\mathbf{\Gamma_j} = \mathbf{B^T V\Sigma^{-1} U^T A_j^T A_j y}$, all peers locally compute $\mathbf{\Gamma_j}$, split it into k parts, and send the i-th part $\mathbf{\Gamma_j^i}$ to peer-i; 3) all peers compute $\mathbf{w'_i} = \sum_{j=1}^{k} \mathbf{\Gamma_j^i}$ and recover $\mathbf{w_i} = \mathbf{B_i^T w'_i}$.

G Artifact Appendix

Abstract

Excalibur is an efficient decentralized federated SVD system. The artifact provides the source code of a fully functional Excalibur system and the guidelines to reproduce the experimental results in our paper. We provide scripts and Dockerfile to quickly build the experimental settings.

Scope

The system's implementation primarily comprises two components: 1) matrix computations, and 2) the communication workflow. Matrix computations encompass computational optimizations (*e.g.*, accelerating multiplicative operations in MMS) and engineering efforts to implement SVD methods

that are not supported by existing libraries (*e.g.*, the bisection-twisted bSVD), which is beneficial to the community. The system workflow pertains to the communication-efficient decentralized SVD and includes efforts to overlap pipelines, thereby reducing communication costs. In terms of reproducibility, we have provided scripts to reproduce all evaluation results presented in the paper.

Hosting

The system is open-sourced and hosted on GitHub. The link is https://github.com/Di-Chai/Excalibur. We introduce the project structure in README and provide instructions to build and reproduce the experimental results.

Requirements

The hardware requirements depend on the scale of the experiments. We provide the detailed hardware requirement and approximate runtime for each evaluation on GitHub. Generally, we assign four CPU cores for each peer, which could be configured in the scripts, meaning that one machine with an 80-core CPU can support 20 peers. To support large-scale data, a fast SSD is recommended. We use a 2TB NVMe SSD in our experiments. The system is tested on Ubuntu 20.04. Comprehensive instructions for setting up the environment can be found in the README file of our GitHub repository.