Scaling SVD for Matrices with Trillion-Scale Non-Zero Entries on a Single Machine

Xinyu Du*
The Chinese University of Hong Kong
xydu@se.cuhk.edu.hk

Haoming Xian*
The Chinese University of Hong Kong
hmxian@se.cuhk.edu.hk

Xingyi Zhang
The Chinese University of Hong Kong
xyzhang@se.cuhk.edu.hk

Sibo Wang†
The Chinese University of Hong Kong
swang@se.cuhk.edu.hk

ABSTRACT

Singular Value Decomposition (SVD) algorithms find extensive applications across diverse domains. Different SVD algorithms are tailored to specific application scenarios, considering factors such as scalability, efficiency, accuracy, and update cost. However, despite the fundamental importance of SVD algorithms, to the best of our knowledge, no single SVD algorithm excels in all aspects for massive matrices: efficiency, accuracy, scalability, and ease of update with theoretical guarantees. Motivated by this, we propose *Rank Selection Tree SVD (RaSTree-SVD)*, an efficient, accurate, scalable, and update-friendly SVD algorithm based on a novel rank-selection tree structure. Specifically, given a matrix $\boldsymbol{P}$, we first divide its rows into b equal-length batches, forming b sub-matrices, and SVD is then performed on each sub-matrix. Subsequently, a rank-selection process is proposed to retain the most significant singular values and vectors of each sub-matrix. The results from multiple sub-matrices are merged to create the next level sub-matrix. This recursive process continues until the final matrix is factorized. The introduction of the rank-selection tree allows for accelerated matrix decomposition and reduced memory cost compared to previous SVD algorithms. Furthermore, by utilizing cached factorized results within the rank-selection tree, we present an efficient update algorithm with theoretical guarantees. Extensive experiments demonstrate the efficiency, accuracy, and scalability of our RaSTree-SVD in both static and dynamic scenarios. To our best knowledge, RaSTree-SVD is the first SVD algorithm that can scale to matrices with up to a trillion non-zero entries on a single machine.

PVLDB Reference Format:
Xinyu Du, Haoming Xian, Xingyi Zhang, and Sibo Wang. Scaling SVD for Matrices with Trillion-Scale Non-Zero Entries on a Single Machine. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

* Xinyu Du and Haoming Xian are the joint first authors.
† Sibo Wang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Singular Value Decomposition (SVD) finds wide-range applications in areas where data is represented in matrices, e.g., computer vision [3, 54, 55], natural language processing [11, 40, 48], graph representation learning [13, 14, 17, 60], and recommender systems [6, 29, 36, 37]. For a matrix $\boldsymbol{P} \in \mathbb{R}^{m \times n}$ with rank r, it can be expressed using SVD as $\boldsymbol{P} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$, where $\boldsymbol{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_r]$ and $\boldsymbol{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_r]$ denote the orthogonal matrices containing the left and right singular vectors, respectively, and $\Sigma = diag\,(\sigma_1, \sigma_2, \cdots, \sigma_r)$ is the diagonal matrix of singular values. As matrices increase in dimensionality, they tend to become sparser. Consequently, in most scenarios, the focus is primarily on the *number of non-zero entries (NNZs)* in the matrix to expedite the decomposition process. To obtain the best d-rank approximation of $\boldsymbol{P}$, a truncated SVD can be employed, denoted as $\boldsymbol{P}_d = \boldsymbol{U}_d\Sigma_d\boldsymbol{V}_d^\top$. These d largest singular values and corresponding singular vectors capture the most significant information of the original matrix $\boldsymbol{P}$ while reducing its dimensionality. In the remainder of this paper, the term SVD refers specifically to the truncated SVD if the context is clear.

SVD algorithms can be broadly categorized into different branches [30] with varied applications, as summarized in Table 1. Among these categories, randomized SVD algorithms designed for sparse matrices with a time complexity of $O(\text{NNZs})$ are the most efficient. These algorithms typically employ Simultaneous Power Iteration with random start vectors [4, 28, 47, 52, 53] or adopt a randomized variant of the Block Lanczos Algorithm [2, 45] to factorize a matrix. However, scalability issues [26] arise as these algorithms require the full input matrix with its NNZs entries. Consequently, memory consumption becomes a primary concern in randomized SVD algorithms. To tackle this issue, incremental SVD algorithms have been developed, including augmented matrix SVD [14], sliding-window SVD [51], and streaming SVD [31]. These algorithms divide the given matrix into multiple sub-matrices and process each sub-matrix separately, allowing for update operations. Yet, augmented matrix SVD algorithms lack theoretical guarantees for updating SVD results, while sliding-window SVD and streaming SVD algorithms only support updates by adding columns or rows and do not support arbitrary updates to the original matrix.

To overcome these limitations, HSVD [32] introduces a hierarchical structure to divide the matrix into sub-matrices and compute the SVD results on each sub-matrix to derive the final SVD result of the original matrix. However, HSVD cannot be applied to high dimensional sparse matrices. Hence when $m \cdot n$ is large, its scalability is limited, even with only a few million non-zero entries. Later, Tree-SVD [17] overcomes this deficiency by integrating randomized SVD for the sub-matrices and proves that it can still achieve the

Table 1: SVD Categories.

Categories	Representatives	Scale	Efficiency	Accuracy	Update
Direct	Golub-Reinsch Algorithm [25], Jacobi Method [33, 35, 46], Divide-and-Conquer Method [12, 27]	Small	Slow	High	✗
Iterative	Lanczos Algorithm[9, 24, 39] , Golub-Kahan Algorithm [23], Implicitly Restarted Lanczos Method [34]	Medium	Medium	High	✗
Randomized	Randomized Simultaneous Power Iteration SVD [28, 47], Randomized Krylov Subspace Method [2, 45]	Medium	Fast	Medium	✗
Incremental	Sliding-window SVD [1, 51], Augmented SVD[13, 14, 59], Streaming SVD[20, 22, 31]	Large	Medium	Low	✓

same approximation guarantee. This enables Tree-SVD to derive the final SVD result linearly depending on NNZs rather than $m \cdot n$. By incorporating hierarchical structures and caching intermediate results (e.g., the SVD for sub-matrices), Tree-SVD presents efficient algorithms that support arbitrary updates to the input matrices. Nevertheless, the space cost of cached intermediate SVD results in Tree-SVD is $O(b \cdot \max\{m, n\} \cdot d)$, where d is the number of sub-matrices. The memory cost becomes high when the dimensionality of the given matrix is large, limiting the scalability of Tree-SVD.

In this paper, we propose *RaSTree-SVD* (Rank Selection Tree SVD) to address the memory bottleneck challenge. Compared to Tree-SVD, our RaSTree-SVD is significantly more memory-efficient and applicable to matrices with large dimensions m and n simultaneously. It achieves comparable results to randomized SVD and Tree-SVD in terms of Frobenius norm error while consuming one order of magnitude less memory to cache intermediate results. Specifically, in contrast to previous attempts that employ a fixed d-rank truncated SVD for each sub-matrix, RaSTree-SVD introduces a novel rank selection scheme within a tree structure. This scheme dynamically selects important information from different sub-matrices. Note that the weights of denser and larger sub-matrices are bigger than others during the merge operations, enabling RaSTree-SVD to capture more information. Empirical evidence validates the significance of the rank selection scheme. By dividing the given matrix into b blocks, RaSTree-SVD only requires in-memory SVD calculation for a single sub-matrix at a time. Upon completion of the SVD calculation for a block, the occupied memory can be released without affecting the final result. These strategies enable RaSTree-SVD to scale to huge matrices, even with trillion-scale NNZs.

Furthermore, by leveraging the rank selection tree structure, RaSTree-SVD can handle arbitrary updates rather than being limited to adding columns or rows. When matrix updates are concentrated on few sub-matrices, only the affected sub-matrices need updating, which can then be merged along the selection tree, significantly reducing update costs while maintaining SVD results equivalent to static reconstruction. For scenarios where updates are distributed across different sub-matrices, RaSTree-SVD efficiently handles the updates through a threshold-based update strategy with theoretical guarantee. Extensive experiments show the effectiveness and efficiency of our RaSTree-SVD.

In summary, our contributions are listed as follows:

- We introduce RaSTree-SVD, a scalable, efficient, and effective SVD algorithm. By leveraging a rank selection tree structure, it selects crucial information from different sub-matrices, significantly reducing time and space costs. To the best of our knowledge, RaSTree-SVD is the first SVD algorithm capable of scaling to matrices with 1 trillion NNZs on a single machine.
- We propose a dynamic framework for RaSTree-SVD that supports arbitrary updates to the original matrix, significantly reducing update cost when matrix updates are concentrated on few sub-matrices. Even when updates are distributed across various sub-matrices, RaSTree-SVD accurately tracks the most significant changes while maintaining a theoretical guarantee.
- Extensive experiments demonstrate that RaSTree-SVD consumes an order of magnitude less space compared to counterparts while achieving far better scalability. Moreover, RaSTree-SVD achieves comparable accuracy to the randomized SVD algorithm while surpassing it in efficiency, particularly in scenarios involving high-dimensional matrices and matrix updates.

2 PRELIMINARIES

2.1 Problem Definitions

Definition 2.1 (Schatten p-norm). Given an input matrix $\boldsymbol{P}$, the Schatten p-norm is denoted as

$$\|\boldsymbol{P}\|_p = \left(\sum_{i=1}^{\min(m,n)} \sigma_i^p \right)^{\frac{1}{p}},$$

where σ_i is the i-th largest singular value of $\boldsymbol{P}$. When $p = 2$, it is the Frobenius norm, which is equivalent to $\sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{P}_{i,j}^2}$, where $\boldsymbol{P}_{i,j}$ is the element at the i-row ant j-th column of matrix $\boldsymbol{P}$. When $p = \infty$, it is the spectral norm (the maximum singular value σ_1). □

Recap that the SVD of a given matrix $\boldsymbol{P}$ is expressed as $\boldsymbol{U}\Sigma\boldsymbol{V}^\top$, which is closely related to the low-rank approximation problem.

Definition 2.2 (Low-Rank Approximation). Given an input matrix $\boldsymbol{P}$ of size $m \times n$ and a positive integer d, the goal of low-rank approximation is to derive a matrix $\boldsymbol{P'}$ of size $m \times n$ and rank d such that, among all possible rank d matrices, the solution $\boldsymbol{P'}$ solves the following optimization problem:

$$\min_{\substack{\boldsymbol{B} \in \mathbb{R}^{m \times n} \\ \text{s.t. } \operatorname{rank}(\boldsymbol{B}) = d}} \|\boldsymbol{P} - \boldsymbol{B}\|_F. \qquad \square$$

Given the SVD result, by keeping the first d columns of $\boldsymbol{U}$, Σ, and $\boldsymbol{V}$, denoted as $\boldsymbol{U}_d$, Σ_d, and $\boldsymbol{V}_d$, respectively, then $\boldsymbol{P}_d = \boldsymbol{U}_d \Sigma_d \boldsymbol{V}_d^\top$ is the solution for the above low-rank approximation problem.

Deriving the exact (truncated) SVD results can be still expensive. Hence, most existing works focus on the approximate version. Besides, as long as we can identify the matrix $\boldsymbol{V}$ for the right singular vectors, it is easy to derive the matrix $\boldsymbol{U}$ for the right singular vectors and the diagonal matrix Σ. In particular, by taking the normalized columns of $\boldsymbol{PV}$, we can derive $\boldsymbol{U}$ since $\boldsymbol{PV} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{U}\Sigma$. Then, by taking the L_2 norm of columns in $\boldsymbol{PV}$, we derive the singular values and thus the diagonal matrix Σ. Hence, in the literature, most existing work [28, 45, 52, 53] focus on deriving the matrix $\boldsymbol{V}$ (or symmetrically matrix $\boldsymbol{U}$). We follow this setting by approximating the matrix $\boldsymbol{V}$ of the right singular vectors efficiently.

Definition 2.3 (Approximate Truncated SVD [28, 45, 52, 53]). Given a matrix $\boldsymbol{P}$, the approximate truncated SVD returns a rank d matrix $\boldsymbol{V}'$ with orthonormal columns $\boldsymbol{v}'_1, \boldsymbol{v}'_2, ..., \boldsymbol{v}'_d$ satisfying that:

$$\left\|\boldsymbol{P} - \boldsymbol{P}\boldsymbol{V}'\boldsymbol{V}'^T\right\|_F \leq (1+\epsilon)\,\|\boldsymbol{P} - \boldsymbol{P}_d\|_F, \tag{1}$$

where $\boldsymbol{P}_d = \boldsymbol{U}_d\Sigma_d\boldsymbol{V}_d^\top$ is the best d-rank approximation of $\boldsymbol{P}$, ϵ is the approximation error, and $\boldsymbol{V}'$ is an approximation of the matrix $\boldsymbol{V}$ of the right singular vectors. □

Furthermore, We use time sequence model to denote a series of timestamps at which we calculate the SVD results of $\boldsymbol{P}$.

Definition 2.4 (Time Sequence Model). Given a continuous series of timestamps from the start timestamp t_s to the end timestamp t_e, the timestamps within this range are denoted as:

$$\phi_s^e = [t_s, t_{s+1}, \cdots, t_{e-1}, t_e]. \quad \square$$

Meanwhile, efficient update on SVD results should be supported when the entries of the input matrix $\boldsymbol{P}$ get changed.

Definition 2.5 (SVD Maintenance under Dynamic Matrix). Given an input matrix $\boldsymbol{P}^t \in \mathbb{R}^{m\times n}$ at timestamp t, for the ease of later analysis, we horizontally divide matrix $\boldsymbol{P}$ by rows into b different equal-size sub-matrices. :

$$\boldsymbol{P}^t = [\boldsymbol{P}^t_{1,1}; \boldsymbol{P}^t_{1,2}; \cdots; \boldsymbol{P}^t_{1,b}],$$

where $\boldsymbol{P}^t_{1,i} \in \mathbb{R}^{\frac{m}{b}\times n}$ denotes the i-th sub-matrix at timestamp t. At timestamp $t+1$, we assume that there might be a modification $\Delta^t_{1,i}$ to each sub-matrix $\boldsymbol{P}^t_{1,i}$, i.e., $\boldsymbol{P}^{t+1}_{1,i} = \boldsymbol{P}^t_{1,i} + \Delta^t_{1,i}$ and

$$\boldsymbol{P}^{t+1} = [\boldsymbol{P}^{t+1}_{1,1}; \boldsymbol{P}^{t+1}_{1,2}; \cdots; \boldsymbol{P}^{t+1}_{1,b}].$$

The goal of the SVD maintenance under dynamic matrix is to update the SVD result, $\boldsymbol{V}'$ in our case, at each timestamp. □

Notice that we use semi-column to indicate a row-wise split in the remaining paper [31]; here we assume that different sub-matrices are of equal size for ease of analysis. When m cannot be divided by b, we can assume that we append $b\lceil\frac{m}{b}\rceil - m$ rows with all zero values, in which case all of our analysis can be still applied.

Table 2 lists the notations that are frequently used in this paper.

2.2 Tools

Definition 2.6 (Numeric Rank [41]). For any matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the numeric rank $r(\boldsymbol{A}) = \|\boldsymbol{A}\|_F^2/\|\boldsymbol{A}\|_2^2$ is a smooth relaxation of the rank of matrix $\boldsymbol{A}$. □

Definition 2.7 (Covariance Error [31]). The covariance error of $\boldsymbol{B}$ with respect to $\boldsymbol{A}$ is defined as $coverr(\boldsymbol{A}; \boldsymbol{B}) = \left\|\boldsymbol{A}^\top\boldsymbol{A} - \boldsymbol{B}^\top\boldsymbol{B}\right\|_2$. □

Another closely related error measure is the d-projection error.

Table 2: Frequently used notations.

Notations	Descriptions
$\|\boldsymbol{P}\|_2, \|\boldsymbol{P}\|_F$	Spectral and Frobenius norm of $\boldsymbol{P}$.
m, n	The number of rows and columns of $\boldsymbol{P}$.
$\boldsymbol{P}_d, (\boldsymbol{P})_d$	A d-rank approximation and the best d-rank approximation of $\boldsymbol{P}$.
$\boldsymbol{T}_{l,i}$	The i-th sub-matrix factorized in level l.
$\boldsymbol{U}_{l,i}, \boldsymbol{V}_{l,i}, \Sigma_{l,i}$	Left singular vector, singular value, and right singular vector matrices of $\boldsymbol{T}_{l,i}$.
k_l	Number of sub-matrices merged in level l.
$q, b_l = \prod_{c=l}^q k_c$	Total number of level and the number of sub-matrices in level l.
d_l	Number of rank selection in level l.
$(\boldsymbol{V}^{\boldsymbol{B}})_d$	d-rank right singular vector matrix of $\boldsymbol{B}$.
$\pi_B^d(\boldsymbol{A}) = \boldsymbol{A}(\boldsymbol{V}^{\boldsymbol{B}})_d(\boldsymbol{V}^{\boldsymbol{B}})_d^\top$	Projection of $\boldsymbol{A}$ onto the subspace of $(\boldsymbol{V}^{\boldsymbol{B}})_d$.
$\sigma_j(\boldsymbol{P}_{l,i})$	The j-th singular values of $\boldsymbol{P}_{l,i}$.

Definition 2.8 (d-projection Error [31]). Given a matrix $\boldsymbol{B}$, suppose the d-rank right singular vector matrix of $\boldsymbol{B}$ is denoted as $(\boldsymbol{V}^{\boldsymbol{B}})_d$. The d-projection error of $\boldsymbol{B}$ with respect to $\boldsymbol{A}$ is defined as $\left\|\boldsymbol{A} - \pi_B^d(\boldsymbol{A})\right\|_F^2$, where $\pi_B^d(\boldsymbol{A}) = \boldsymbol{A}(\boldsymbol{V}^{\boldsymbol{B}})_d(\boldsymbol{V}^{\boldsymbol{B}})_d^\top$ is the rank-d matrix generated from projecting each row of $\boldsymbol{A}$ onto the subspace spanned by the top-d right singular vectors of $\boldsymbol{B}$. □

Theorem 2.9. *The d-projection will reduce the Frobenius norm of* $\boldsymbol{A}$, $\left\|\pi_B^d(\boldsymbol{A})\right\|_F \leq \|\boldsymbol{A}\|_F$. □

The following lemma establishes the connections between the covariance error and the d-projection error, which is commonly employed in frequent direction problems.

Lemma 2.10 (Projection Bound[21]).

$$\left\|\boldsymbol{A} - \pi_B^d(\boldsymbol{A})\right\|_F^2 \leq \|\boldsymbol{A} - (\boldsymbol{A})_d\|_F^2 + 2d \cdot \left\|\boldsymbol{A}^\top\boldsymbol{A} - \boldsymbol{B}^\top\boldsymbol{B}\right\|_2. \quad \square$$

2.3 Related Work

To address the memory issues faced by randomized SVD algorithms, several incremental SVD algorithms have been proposed. Next, we provide a brief overview of relevant studies, including augmented-matrix SVD, sliding-window SVD, and streaming SVD algorithms.

Augmented SVD. Given the SVD results of $\boldsymbol{P}$, these algorithms calculate the approximate SVD results of the updated matrix when incorporating new rows $[\boldsymbol{P}; \boldsymbol{E}_r]$, new columns $[\boldsymbol{P}, \boldsymbol{E}_c]$, or low-rank modifications $\boldsymbol{P} + \boldsymbol{D}_m\boldsymbol{E}_m$, where $\boldsymbol{E}_r, \boldsymbol{E}_m \in \mathbb{R}^{s\times n}$ and $\boldsymbol{E}_c, \boldsymbol{D}_m \in \mathbb{R}^{m\times s}$. Recently, Deng et al. [13] introduced a novel SVD update algorithm based on the algorithm proposed by Zha et al. [59]. They further enhanced their algorithm by employing a numerical method [14]. However, compared to randomized SVD, these algorithms face significant limitations when computing the SVD of the entire matrix. For example, when factorizing a personalized PageRank matrix with dimensions of $41M \times 41M$ obtained from the Twitter dataset [38], the algorithm proposed by Deng et al. [13] requires over a month to compute the SVD result on a single machine. Furthermore, a critical drawback of these algorithms is the lack of theoretical guarantees for the accuracy of the updated SVD results.

Sliding-window SVD. Initially, algorithms attempted to maintain simple statistics, such as count and sum [10, 44], skyline [49], and quantiles [1] for the sliding-window setting. Next, Wei et al. [51] employed a random sampling algorithm and two deterministic algorithms to improve the practical performance in this setting. Typically, these models operate under either a time-based sliding window assumption, where rows within a fixed time interval are analyzed, or a sequence-based sliding window assumption, where a fixed given number of newest rows are considered.

Streaming SVD. Early attempts [15, 16, 19] presented algorithms that focuses on selecting a subset of columns or rows from the streaming data matrix. Other approaches adopt random projection [4] or hashing [8]. Recently, the *frequent direction (FD)* approach [20, 22, 31] has drawn more attention and emerged as the preferred choice for practical applications.

Remark. Notice that, most existing sliding-window and streaming SVD algorithms do not maintain intermediate SVD results. Instead, they fix the columns and incrementally add new rows to perform SVD computations. Since this strategy only preserves the latest SVD results, any updates to the given matrix must happen in the new rows. Consequently, these algorithms do not support arbitrary updates to the original matrix. In contrast, our proposed RaSTree-SVD introduces a novel rank selection tree structure, enabling efficient arbitrary updates to the original matrix. At the same time, our goals differ as well: RaStree-SVD aims to efficiently factorize the given matrix and efficiently finish arbitrary updates with acceptable memory cost. On the contrary, most sliding-window models and streaming SVD models focus on minimizing the theoretical space cost in the random access memory (RAM), rather than compressing the original matrix into a tight representation of a given d. For example, these sketching models usually require an uncertain theoretical number of representation dimension $O(\frac{d}{\epsilon})$ to maintain a d-rank error guarantee, which is not suitable for a lot of tight compression task. For examples, the mainstream FD [21, 31] model takes a final matrix of $\boldsymbol{P'} \in \mathbb{R}^{O(\frac{d}{\epsilon})\times n}$ to generate a d-rank approximation of $\boldsymbol{P}$ with error bound:

$$\left\|\boldsymbol{P} - \pi^{d}_{\boldsymbol{P'}}(\boldsymbol{P})\right\|_F^2 \leq (1+\epsilon)\left\|\boldsymbol{P} - \boldsymbol{P}_d\right\|_F^2,$$

whereas RaStree-SVD can achieve a d-rank approximation with a theoretical bound from a final matrix $\boldsymbol{P'} \in \mathbb{R}^{d\times n}$.

Theoretical Bound of SVD Update. Recently, Luo et al. [43] present a perturbation projection error bound for Schatten-q low-rank matrices. Vu et al. [50] provide a perturbation expansion error bound for the truncated SVD. Both methodologies enhance the SVD update bound through perturbation analysis. However, these advancements lack corresponding scalable computational algorithms.

2.4 Main Competitors

FD [41]. Frequent Direction (FD) uses matrix sketching for dimension reduction, processing large matrices by maintaining a compact sketch that emphasizes the most significant directions or singular vectors. Through iterative updates with incoming data, it effectively captures the principal directions of the matrix, providing accurate approximations of its top singular vectors with minimal memory usage. However, its complexity depends on the dimension $n \times m$, hindering its scalability on large scale matrices. Alternatives like the state-of-the-art SVS [31] are an order of magnitude slower than FD when run on a single machine in our initial tests, thus are omitted. The reason is that such methods prioritize constraining memory costs while sacrificing the running efficiency.

FRPCA [18]. FRPCA is the state-of-the-art sparse randomized SVD algorithm. This open-sourced algorithm is widely used in literature [57, 60]. Experiments [18] show that FRPCA is approximately an order of magnitude faster than basic randomized SVD, and up to 20 times faster than the truncated SVD function "svds" in Matlab, while achieving comparable accuracy.

Tree-SVD [17]. This algorithm is an enhanced version of HSVD [32] with time complexity depending on the NNZ rather than the dimension of the matrix. Although Tree-SVD is efficient and supports arbitrary updates, it is mainly suitable for rectangular matrices due to its memory consumption. When dealing with a matrix $\boldsymbol{P} \in \mathbb{R}^{m\times n}$, Tree-SVD is not suitable for matrices with both large m and n simultaneously. This limitation highlights the scalability difference between Tree-SVD and RasTree-SVD.

3 OUR SOLUTION: RASTREE-SVD

In this section, we elaborate on our RaSTree-SVD framework. We first horizontally divide the input matrix $\boldsymbol{P}$ by rows into b sub-matrices, which we refer to as the 1st level. Then, we use randomized SVD to derive the SVD result for each sub-matrix. After this process, we merge the SVD results of k_1 consecutive sub-matrices and generate a new matrix as the input sub-matrix in the second level. In the second level, we thus have b/k_1 sub-matrices. From level 2 and above, we use the exact truncated SVD because now the input sub-matrices has low dimensionality and low running cost.

One of our key contribution lies in adaptively selecting the dimension of each sub-matrix when merging these matrices. The high level idea is to select the largest singular vectors and singular values from this group of SVD results to form a corresponding sub-matrix and feed it in the next level. We repeat this process, taking the exact SVD of k_l ($k_1 \geq 2$) sub-matrices in the l-th level ($l \geq 2$) and using it as the input of the $(l+1)$-th level, where we have $b/\Pi_{j=1}^{l}k_j$ sub-matrices in level $(l+1)$, until we reach level q. Note that we will set n and k_i properly so that $b/\Pi_{j=1}^{q-1}k_j = 1$. At this point, we have only one matrix left, and we take the truncated exact SVD of this sub-matrix to output the matrix of the right-singular vectors.

Conceptually, we have a tree structure where each node maps to a sub-matrix. When the SVD results of the sub-matrices merge to a new sub-matrix, we treat these sub-matrices as the children of the new sub-matrix. Since the merging process involves selecting different ranks of the sub-matrices based on their singular values, we refer to this tree structure as the *Rank Selection Tree*. A slight difference here is that the root is at level q while in our typical definition, the root is usually at level 1.

During this process, every SVD result within each level of the RaSTree-SVD is cached in memory. With these cached SVD results, we can avoid recomputing the SVD results from scratch if the sub-matrix is not affected during updates. We further devise a dynamic algorithm for our RaSTree-SVD to efficiently address major updates with theoretical guarantees. In the following sections, we first show how RaSTree-SVD works in static setting in Sec. 3.1. Then, we provide details about dynamic RaSTree-SVD in Sec. 3.2.

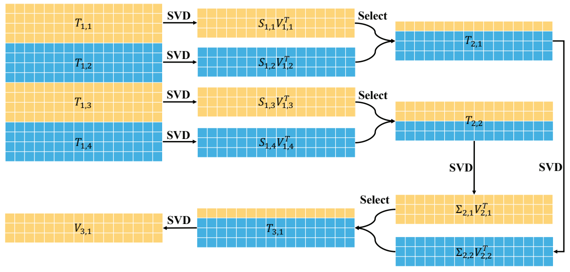


Figure 1: RaSTree-SVD with 3 levels

3.1 Static RaSTree-SVD

As we mentioned earlier, given an input matrix $\boldsymbol{P} \in \mathbb{R}^{m\times n}$, we divide the matrix horizontally by rows into b different sub-matrices:

$$\boldsymbol{P} = [\boldsymbol{P}_{1,1}; \boldsymbol{P}_{1,2}; \cdots ; \boldsymbol{P}_{1,b}],$$

where $\boldsymbol{P}_{1,i} \in \mathbb{R}^{\frac{m}{b}\times n}$ denotes the i-th sub-matrix in the first level. Our goal is to solve the approximate truncated SVD problem (Ref. to Definition 2.3). For a matrix $\boldsymbol{P} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$, we further denote $\overline{\boldsymbol{P}} = \boldsymbol{P}\boldsymbol{V} = \boldsymbol{U}\Sigma$. Note that if $\boldsymbol{P}$ is rank deficient and/or has repeated singular values, $\overline{\boldsymbol{P}}$ is not necessarily uniquely determined by $\boldsymbol{P}$. Let $\boldsymbol{T}_{l,i}$ denote the i-th sub-matrix to be factorized in the l-th level of RaSTree-SVD. According to the definition, $\boldsymbol{T}_{1,i}$ is the i-th sub-matrix to be factorized at level $l = 1$ in RaSTree-SVD, which is also the i-th sub-matrix of the input matrix $\boldsymbol{P}_{1,i}$.

Given b sub-matrices at level 1, we group k_1 sub-matrices together and have b/k_1 groups. For group i, define $z = (i-1)\cdot k_1$. The matrices at positions $z+1, z+2, \ldots, z+k_1$ belong to this group. We then construct the matrix $\boldsymbol{T}_{2,i}$ as follows (Lines 4-5):

$$\boldsymbol{T}_{2,i} = [\Sigma_{1,z+1}\boldsymbol{V}_{1,z+1}^\top; \Sigma_{1,z+2}\boldsymbol{V}_{1,z+2}^\top; \cdots ; \Sigma_{1,z+k_1}\boldsymbol{V}_{1,z+k_1}^\top]$$

and feed these b/k_1 sub-matrices as the input at level 2.

At level l ($l \geq 2$), we have $b_l = b/\prod_{i=1}^{l} k_{l-1}$ sub-matrices. For example, at level 2, we have $b_2 = b/k_1$ sub-matrices. At level 3, we have $b_2 = \frac{b}{k_1\cdot k_2}$ sub-matrices, etc. Here we set up each k_l so that each $b/\prod_{i=1}^{l} k_l$ is an integer. Also, we assume that m (the number of rows of $\boldsymbol{P}$) can be evenly divided by b for the ease of exposition. Yet, there is no such restriction to RaSTree-SVD, where different levels may aggregate different numbers of sub-matrices. Starting from level $l \geq 2$, $\boldsymbol{T}_{l,i}$ is obtained by using the exact SVD results of k_l matrices $\boldsymbol{T}_{l-1,(i-1)\cdot k_l+1}, \boldsymbol{T}_{l-1,(i-1)\cdot k_l+2}, \cdots, \boldsymbol{T}_{l-1,i\cdot k_l}$ at level $(l-1)$ using a rank selection process, to be explained shortly. Besides, the dimension d_1 serves as a trade-off parameter between the worst-case approximation guarantee and the space cost.

Rank Selection. One of our key contribution is a new rank selection strategy to derive $\boldsymbol{T}_{l,i}$ from a group of k_l sub-matrices at level $l-1$ (for $l \geq 2$). Our rank selection strategy is motivated by two main factors. First, the non-uniform distribution of numerical values across different sub-matrices implies varying scales of significance; some sub-matrices may hold more pertinent information for the truncated SVD in subsequent levels. Yet, earlier hierarchical methods like HSVD [32] and Tree-SVD [17] overlook this variability and uniformly apply the same rank d at all levels. Our approach not only recognizes this discrepancy but also provides a theoretical guarantee for the SVD results, maintaining an empirical error comparable to the state of the art. Second, in anticipation of updates, we cache only the critical intermediate sub-matrices. This means when updates to the input matrix $\boldsymbol{P}$ occur, sub-matrices that remain unchanged do not require re-computation of SVD results. Current methods like Tree-SVD cache entire sub-matrix dimensions, resulting in high memory costs and limiting their applicability to moderately-sized matrices. Our refined rank-selection strategy, by focusing on the sub-matrices with most significant singular values, considerably reduces memory demands, making it feasible for large scale matrices.

Next, we explain how our rank selection works. Let $\boldsymbol{T}_{l-1,i} = \boldsymbol{U}_{l-1,i}\Sigma_{l-1,i}\boldsymbol{V}_{l-1,i}^\top$ represent the exact SVD of matrix $\boldsymbol{T}_{l-1,i}$, where $\boldsymbol{U}_{l-1,i}$ is the matrix of the left singular vectors, $\Sigma_{l-1,i}$ is the diagonal matrix for singular values, and $\boldsymbol{V}_{l-1,i}$ is the matrix of right singular vectors for matrix $\boldsymbol{T}_{l-1,i}$. We further use $(\Sigma_{l-1,i})_{d_{l-1,i}}$ to denote the matrix of the largest $d_{l-1,i}$ singular values of $\boldsymbol{T}_{l-1,i}$ and $(\boldsymbol{V}_{l-1,i})_{d_{l-1,i}}$ to denote the matrix of the first $d_{l-1,i}$ right singular vectors corresponding to $(\Sigma_{l-1,i})_{d_{l-1,i}}$. The value of $d_{l-1,i}$ is determined by the d_{rs}-rank selection defined as follows.

Definition 3.1 (d_{rs}-rank selection). Given a group of k matrices $\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_k$ with dimensionality $m_i \times d'$ for $\boldsymbol{A}_i$ as input, let their corresponding SVD to be $\boldsymbol{A}_i = \boldsymbol{U}_i\Sigma_i\boldsymbol{V}_i^\top$ for each $i \in \{1, 2\cdots, k\}$. The d_{rs}-rank selection first takes the SVD of these k matrices. Then, it ranks all these $k\cdot d'$ singular values in decreasing order of their values and select the top-$k\cdot d_{rs}$ largest singular values. Assume that $\boldsymbol{A}_i$ has d_{a_i} singular values among the top-$k\cdot d_{rs}$ largest singular values (clearly, $\sum_{i=1}^{k} d_{a_i} = k\cdot d_{rs}$). Then, the d_{rs}-rank selection will output a new matrix of dimension d_{rs} as follows:

$$[(\Sigma_1\boldsymbol{V}_1^\top)_{d_{a_1}}; (\Sigma_2\boldsymbol{V}_2^\top)_{d_{a_2}}; \cdots ; (\Sigma_k\boldsymbol{V}_k^\top)_{d_{a_k}}]. \qquad \square$$

Then, we apply the d_l-rank selection process on k_l matrices $\boldsymbol{T}_{l,(i-1)\cdot k_l+1}, \boldsymbol{T}_{l,(i-1)\cdot k_l+2}, \cdots, \boldsymbol{T}_{l,i\cdot k_l}$ at level l for each integer $0 \leq i < b_{l+1}$ (Line 4). Let $z = (i-1)\cdot k_l$. Then $\boldsymbol{T}_{l,i}$ is derived as:

$$\begin{aligned}\boldsymbol{T}_{l+1,i} =& [(\Sigma_{l,z+1}\boldsymbol{V}_{l,z+1})_{d_{l,z+1}}; (\Sigma_{l,z+2}\boldsymbol{V}_{l,z+2})_{d_{l,z+2}};\\ &\cdots ; (\Sigma_{l,z+k_l}\boldsymbol{V}_{l,z+k_l})_{d_{l,z+k_l}}].\end{aligned}$$

These b_{l+1} sub-matrices derived at level $l+1$ is further fed as the input matrices for the next level. We repeat this process until we reach level q and there is only one matrix $\boldsymbol{T}_{q,1}$ at level q. Then we compute the SVD of $\boldsymbol{T}_{q,1}$ and return $(\boldsymbol{V}_q)_d$.

Example 3.2. Figure 1 shows an example of our RaSTree-SVD with three levels. Assume that the final goal is to derive an approximate truncated SVD with a dimension $d = 3$. At level 1, the input matrix includes 4 sub-matrices, $\boldsymbol{T}_{1,1}, \boldsymbol{T}_{1,2}, \boldsymbol{T}_{1,3}$, and $\boldsymbol{T}_{1,4}$, each having 4 rows and 16 columns. In our RaSTree-SVD, an SVD is applied to these four matrices and $d_1' = 3$ dimensions are kept for these four sub-matrices. Next, we apply two 2-rank selection process on $\Sigma_{1,1}\boldsymbol{V}_{1,1}^\top$ with $\Sigma_{1,2}\boldsymbol{V}_{1,2}^\top$ to derive $\boldsymbol{T}_{2,1}$ and $\Sigma_{1,3}\boldsymbol{V}_{1,3}^\top$ with $\Sigma_{1,4}\boldsymbol{V}_{1,4}^\top$ to derive $\boldsymbol{T}_{2,2}$. Assume that among the 6 singular values in $\Sigma_{1,1}$ and $\Sigma_{1,2}$, the top-4 highest singular values consist of one from $\Sigma_{1,1}$ and three from $\Sigma_{1,2}$. Then we construct $\boldsymbol{T}_{2,1}$ as $[(\Sigma_{1,1})_1(\boldsymbol{V}_{1,1}^\top)_1; (\Sigma_{1,2})_3(\boldsymbol{V}_{1,2}^\top)_3]$. Similarly, we construct $\boldsymbol{T}_{2,2}$ as $[(\Sigma_{1,3})_2(\boldsymbol{V}_{1,3}^\top)_2; (\Sigma_{1,4})_2(\boldsymbol{V}_{1,4}^\top)_2]$. Hence, both $\boldsymbol{T}_{2,1}$ and $\boldsymbol{T}_{2,2}$ have a dimension of $k_1\cdot d_1 = 4$. Next, we apply SVD to $\boldsymbol{T}_{2,1}$ and $\boldsymbol{T}_{2,2}$ to obtain $\Sigma_{2,1}\boldsymbol{V}_{2,1}^\top$ and $\Sigma_{2,2}\boldsymbol{V}_{2,2}^\top$, respectively. Then assume that $d_3 = 2$. We apply a 2-rank selection process on $\Sigma_{2,1}\boldsymbol{V}_{2,1}^\top$ and $\Sigma_{2,2}\boldsymbol{V}_{2,2}^\top$. Assume that

Algorithm 1: RaSTree-SVD

```
Input: $P$, SVD dimensions $d'_1, d'_2, \cdots, d'_q$, rank selection dimensions $d_1, d_2, \cdots, d_q$
Output: $(V_{q,1})_d$
1 for $l = 1, ..., q-1$ do
2     for $i = 1, 2, ..., b/\prod_{j=1}^{l} k_j$ do
3         Compute the $d'_l$-rank SVD on each sub-matrix $T_{l,z}$ in group $i$, respectively, where group $i$ includes $k_l$ sub-matrices with $z \in \{(i-1) \cdot k_l + 1, (i-1) \cdot k_l + 2, \cdots, i \cdot k_l\}$
4         Apply a $d_l$-rank selection on the group $i$ of $k_l$ sub-matrices to derive $T_{l+1,i}$
5 SVD of $T_{q,1}$ to get $(V_{q,1})_d$
```

among the 6 singular values in $\Sigma_{2,1}$ and $\Sigma_{2,2}$, the top-4 highest singular values consist of one from $\Sigma_{2,1}$ and three from $\Sigma_{2,2}$. Then we construct $T_{3,1}$ as $[(\Sigma_{2,1})_1(V_{2,1}^\top)_1; (\Sigma_{2,2})_3(V_{2,2}^\top)_3]$. Finally, we apply SVD to $T_{3,1}$ and obtain $V_{3,1}$ as the output. □

Analysis of RaSTree-SVD. Firstly, we establish Theorem 3.3 to show the quality guarantee of our proposed RaSTree-SVD. The proofs are deferred to Sec. 4.

THEOREM 3.3. *Given a high rank input matrix P ($n \gg b$), let $(V_{q,1})_d$ be the d-rank right singular vector matrix of $T_{q,1}$ in the level q of RaSTree-SVD with $q \geq 2$. Algorithm 1 satisfies that:*

$$\left\|P - \pi^d_{T_{q,1}}(P)\right\|_F^2 \leq \|P - (P)_d\|_F^2 + \epsilon \left\|P - (P)_{d_1}\right\|_F^2,$$

where $\pi^d_{T_{q,1}}(P) = P(V_{q,1})_d(V_{q,1})_d^\top$ is the d-rank matrix generated by projecting each row in P onto the subspace spanned by the top-d right singular vectors $(V_{q,1})_d$. □

Note that the approximation guarantee relates to d_1. It is evident that setting d_1 equal to d allows for a $(1+\epsilon)$-approximation of the best d-rank truncated SVD results, albeit at a high space cost and running cost (as our analysis will show later). Conversely, choosing a smaller d_1 intuitively compromises the approximation quality. However, as our experiments will demonstrate, our carefully designed rank selection mechanism enables us to configure d_1 in a manner that reduces space costs without compromising the practical quality of SVD results. This is primarily because we retain the most critical singular values and their corresponding vectors, thereby ensuring high-quality SVD outcomes. Furthermore, when we apply the same dimensionality reduction by preserving only d_1 dimensions to Tree-SVD, the quality of the SVD results significantly deteriorate where the relative error increases by more than 10% compared to our carefully designed rank selection strategy. This further shows the superiority of our rank selection strategy.

Next, we analyze the space cost and the running time of our RaSTree-SVD. As mentioned earlier, the memory consumption is the main bottleneck of scalable SVD computation. In our RaSTree-SVD, the space cost to keep the intermediate sub-matrices is:

$$O(n \cdot b \cdot d_1 + n \cdot \frac{b \cdot d_2}{k_1} + \cdots + n \cdot \frac{b \cdot d_{l+1}}{\prod_{j=1}^{l} k_l} + \cdots n \cdot d_q).$$

In our setting, we set $d_i = O(d_1)$, and with the fact that $k_i \geq 2$, we know the space cost can be bounded by:

$$O(n \cdot b \cdot d_1 + n \cdot \frac{b \cdot d_1}{2} + \cdots + n \cdot \frac{b \cdot d_1}{2^l} + \cdots n \cdot d_1) = O(n \cdot b \cdot d_1).$$

Hence, d_1 determines the space cost of our RaSTree-SVD. The higher d_1 is, the higher the space cost our RaSTree-SVD has. In addition to the cost of the intermediate results, our RaSTree-SVD needs to load the original input matrix block by block. Assume that there are a number $O(NNZ_b)$ of NNZs in each block. Then, the final memory cost is bounded by $O(NNZ_b + n \cdot b \cdot d_1)$.

Finally, we further analyze the time complexity of our RaSTree-SVD. As the input matrix might be sparse, we apply a randomized SVD [45] to make the cost linear to the NNZs. With $d'_l = O(d_l)$, this takes $O(NNZ \cdot \frac{d_1 \log n}{\sqrt{\epsilon}})$ cost according to [45], where n is the number of columns of P and ϵ is the approximation error parameter. Then, at each level $2 \leq l \leq q$, we take the exact SVD of each sub-matrix at level l, which takes $O(\frac{b \cdot n \cdot d_l^2}{\prod_{c=1}^{l-1} k_c})$. Thus, the total cost for the exact SVDs from level 2 to q is bounded by:

$$O(\sum_{l=2}^{q} \frac{b \cdot n \cdot d_l^2}{\prod_{c=1}^{l-1} k_c}) = O(\sum_{l=2}^{q} \frac{b \cdot n \cdot d_1^2}{2^{l-1}}) = O(b \cdot n \cdot d_1^2).$$

In total, the running cost of our RaSTree-SVD is thus:

$$O(NNZ \cdot \frac{d_1 \log n}{\sqrt{\epsilon}} + b \cdot n \cdot d_1^2).$$

3.2 RaSTree-SVD Update

Next, we introduce how to utilize the rank-selection tree structure to update the SVD results. One key advantage of our RaSTree-SVD is its ability to divide the input matrix into multiple sub-matrices and recursively merge the SVD results of these sub-matrices. Hence, when updates are made to the matrix, it is not necessary to recompute from scratch. Specifically, we can cache the computed intermediate results. For those intermediate results that are either unaffected or only minimally affected, we can skip updates and reuse the existing results, thus saving computational resources.

Suppose we have the RaSTree-SVD result computed at the starting timestamp t_s, and the goal is to update the SVD results of the input matrix from the start timestamp t_s to the (current) end timestamp t_e. For the ordered set ϕ_s^e of timestamps $\phi_s^e = \{t_s, t_{s+1}, \cdots, t_e\}$, the entries of P get updated from P^{t_s} to P^{t_e}. Given the matrix $P^{t_\tau} = [P_{1,1}^{t_\tau}; P_{1,2}^{t_\tau}; \cdots; P_{1,b}^{t_\tau}]$ at a middle timestamp $t_\tau \in \phi_s^e$, we define $\Delta_{1,i}^{t_\tau} = P_{1,i}^{t_\tau} - P_{1,i}^{t_\tau - 1}$ as the difference sub-matrix between the sub-matrices $P_{1,i}^{t_\tau - 1}$ at timestamp $t_\tau - 1$ and the sub-matrices $P_{1,i}^{t_\tau}$ at timestamp t_τ. For each level l, we use $I_{l,c}^{t_\tau}$ to denote the set of cached indexes at timestamp t_τ, and $I_{l,u}^{t_\tau}$ to denote the set of updated indexes at timestamp t_τ. We define T^{t_τ} as the corresponding matrix of P^{t_τ} we have maintained at time t_τ:

$$T^{t_\tau} = [P_{1,1}^{t_{1,1}}; P_{1,2}^{t_{1,2}}, \cdots; P_{1,b}^{t_{1,b}}] = [T_{1,1}^{t_\tau}; T_{1,2}^{t_\tau}, \cdots; T_{1,b}^{t_\tau}],$$

Algorithm 2: RaSTree-SVD-Update

```
Input: Level $q$, SVD dimension $d_l$, dimensions $d_{l,i}$,
       sub-matrices $(\Sigma_{l,i}^{t_{l,i}})_d$ , $(V_{l,i}^{t_{l,i}})_d$ , $\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t$
Output: $(V_{q,1})_d$
1  Init $I_{l,u}^{t_\tau} \leftarrow \emptyset$ for $1 \le l \le q$, sort $\left\|\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t\right\|_F$ for $1 \le i \le b$.
2  while $\sum_{i=1}^{b} \left\|\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t\right\|_F > \beta \left\|P^{t_\tau}\right\|_F$ do
3      Select $i_u = \arg\max_i \left\|\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t\right\|_F$; add $i_u$ to set $I_{1,u}^{t_\tau}$;
4      Run the SVD on $P_{1,i_u}^{t_\tau}$, update $T_{1,i_u}^{t_\tau} = P_{1,i_u}^{t_\tau}$;
5      Add $\lfloor \frac{i_u-1}{k_1} \rfloor + 1$ to the set $I_{2,u}^{t_\tau}$;
6  Update $T_{2,i}^{t_\tau}$ for each $i \in I_{2,u}^{t_\tau}$;
7  for $l = 2$ to $q-1$ do
8      for $i_u \in I_{l,u}^{t_\tau}$ do
9          Compute the $d_l'$-rank SVD of $T_{l,i_u}^{t_\tau}$;
10         Add $\lfloor \frac{i_u-1}{k_l} \rfloor + 1$ to set $I_{l+1,u}^{t_\tau}$;
11     for $i_u \in I_{l+1,u}^{t_\tau}$ do
12         Apply the $d_l$-rank selection on sub-matrices at
             position $k_l(i_u-1)+1, k_l(i_u-1)+2, \cdots, k_l i_u$ at
             level $l$ and derive $T_{l+1,i_u}^{t_\tau}$;
13 Return the SVD of $T_{q,1}^{t_\tau}$ to get $(V_{q,1}^{t_\tau})_d$;
```

where $t_{1,i} \in \phi_s^\tau$ is the latest timestamp when we compute SVD for the i-th sub-matrix at the first level, i.e., $P_{1,i}^{t_{1,i}} = T_{1,i}^{t_\tau}$. Then we have

$$P^{t_\tau} - T^{t_\tau} = [\sum_{t=t_{1,1}}^{t_\tau} \Delta_{1,1}^t; \sum_{t=t_{1,2}}^{t_\tau} \Delta_{1,2}^t; \cdots ; \sum_{t=t_{1,b}}^{t_\tau} \Delta_{1,b}^t],$$

where $\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t = \mathbf{0}$ if there is no update of sub-matrix $P_{1,i}^{t_{1,i}}$ from the timestamp $t_{1,i}$ to the timestamp t_τ or the sub-matrix $P_{1,i}$ is updated at the timestamp t_τ with a randomized SVD.

Clearly, given the set of b sub-matrices, if there is only a small number of sub-matrices get affected, we can selectively update those affected sub-matrices and follow the rank selection tree to update the results along the tree until we reach the root. For example, consider Figure 1. If only matrix $T_{1,1}$ is updated from $T_{1,1} = T_{1,1}^{t_{1,1}}$ to $T_{1,1} = T_{1,1}^{t_\tau}$, then we can re-use the cached SVD results for $T_{1,2}, T_{1,3}, T_{1,4}$ as $T_{1,2}^{t_{1,2}} = T_{1,2}^{t_\tau}, T_{1,3}^{t_{1,3}} = T_{1,3}^{t_\tau}, T_{1,4}^{t_{1,4}} = T_{1,4}^{t_\tau}$. At the second level, we may only update $T_{2,1}$ from $T_{2,1}^{t_{2,1}}$ to $T_{2,1}^{t_\tau}$ and re-use the cached SVD results for $T_{2,2}$ as it is not affected with $T_{2,2}^{t_{2,2}} = T_{2,2}^{t_\tau}$. Finally, we update the SVD result for $T_{3,1}$ as it has also changed. By employing this strategy, we can have the same result as a re-computation from scratch, saving computational resources.

However, in real-world scenarios, matrix updates may occur in an arbitrary manner, and the pace at which data changes nowadays is faster than ever. Therefore, it is highly likely that most of the sub-matrices in the input matrix P will undergo changes. In such cases, the above solution is the same as recomputing the entire SVD results from scratch, resulting in unnecessarily high computational costs. To address these limitations, we propose a threshold-based update strategy to adaptively update the SVD results. We note that Tree-SVD also incorporates a lazy-update mechanism. However, their solution needs to keep track of $\|P_i - (P_i)_d\|_F$, which becomes computationally expensive for matrices with large dimensions m and n. Consequently, the practical utility of Tree-SVD is limited to rectangular matrices. In contrast, our RaSTree-SVD simplifies this process by only requiring the tracking of $\|P\|_F$ and $\|\Delta\|_F$, leading to a significant reduction in computational costs. The dynamic RaSTree-SVD algorithm is detailed as follows.

Algorithm 2 shows the pseudo-code of our threshold-based update strategy of the RaSTree-SVD algorithm. At time t_τ, we initialize $I_{l,u}^{t_\tau} \leftarrow \emptyset$ for all $1 \le l \le q$, and sort $\left\|\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t\right\|_F$ for $\forall i \in \{1, 2, \cdots, b\}$. Clearly, the larger $\left\|\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t\right\|_F$ is, the more significant the sub-matrices $P_{1,i}$ get changed since the latest SVD computation on matrix $P_{1,i}$ at timestamp $t_{1,i}$. Then our goal is to identify all the most significantly changed sub-matrices and update them until the remaining changes are below a certain threshold.

More accurately, we check if the current cached sub-matrix and the latest sub-matrices at timestamp τ, i.e. $\sum_{i=1}^{b} \left\|\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t\right\|_F$ is above $\beta \left\|P^{t_\tau}\right\|_F$. If the sum exceeds the threshold, we select the sub-matrix with largest $\left\|\sum_{t=t_{1,i}}^{t_\tau} \Delta_{1,i}^t\right\|_F$ each time and add its index i_u to the set $I_{1,u}^{t_\tau}$ (Line 3). After adding the index of matrix $T_{1,i_u}^{t_\tau}$ to the set $I_{1,u}^{t_\tau}$, we further run the randomized SVD on $P_{1,i_u}^{t_\tau}$ and set $T_{1,i_u}^{t_\tau} = P_{1,i_u}^{t_\tau}$ (Line 4). Finally, we add $x = \lfloor \frac{i_u-1}{k_1} + 1 \rfloor$ to the set $I_{2,u}^{t_\tau}$ (Line 5). To explain, the matrix at the second level that depends on i_u has an index of x. We repeat this process (Line 2) until the remaining sub-matrices exhibit changes below the threshold.

Then, we update the input sub-matrices affected at level 2. For each affected sub-matrix at level 2, we have recorded their indices in $I_{2,u}^{t_\tau}$. Let $z = (i-1) \cdot k_1$. Hence, for a level 2 sub-matrix affected with id i, we use the group of corresponding SVD results from level 1 with indices $z+1, \cdots, z+k_1$. Each sub-matrix in level 1 is either newly updated at timestamp t_τ or unaffected. We use these results to derive the new sub-matrix at level 2: $T_{2,i} = [\Sigma_{1,z+1}^{t_{1,z+1}} V_{1,z+1}^{t_{1,z+1}\top}; \Sigma_{1,z+2}^{t_{1,z+2}} V_{1,z+2}^{t_{1,z+2}\top}; \cdots ; \Sigma_{1,z+k_1}^{t_{1,z+k_1}} V_{1,z+k_1}^{t_{1,z+k_1}\top}]$ (Line 6).

Next, we track the indices of updated sub-matrices from level l to level $l+1$. For each $i_u \in I_{l,u}^{t_\tau}$, we re-compute the SVD of its corresponding matrix T_{l,i_u} and select the top-d_{l,i_u} singular values and their corresponding right singular vectors. We further include the indices of the affected sub-matrices at level $l+1$ that depends on i_u, denoted as $\lfloor \frac{i_u-1}{k_l} \rfloor + 1$, into the set $I_{l+1,u}^{t_\tau}$ (Lines 8-10). After that, for any level $l+1$ sub-matrix whose index i_u falls into $I_{l+1,u}^{t_\tau}$, we apply a d_{l+1} rank selection on the sub-matrices from group i_u at level l to re-compute $T_{l+1,i_u}^{t_\tau}$. According to how our sub-matrices are grouped, the sub-matrices of group i_u of level l have indices: $k_l(i_u-1)+1, k_l(i_u-1)+2, \cdots, k_l \cdot i_u$. This iterative loop continues until we reach the root. Finally, we compute the SVD of $T_{q,1}$ and return the updated SVD results (Line 13).

Remark. A slightly tricky aspect of dynamically updating the matrix is that the weight of the sub-matrices might change significantly. Recall that in the static RaSTree-SVD algorithm, we apply a d_l-rank selection to obtain a matrix at level l. However, if we only cache the exact dimensions used for the rank selection of each sub-matrix, we might fail to track the right singular vectors corresponding to the most significant singular values of the sub-matrices when the weights get changed. For instance, consider the matrices $\boldsymbol{T}_1$ (resp. $\boldsymbol{T}_2$) with singular values 1, 2, 3, 6, 8 (resp. 1.5, 2.5, 4, 5, 7) before the update. Suppose we keep 4 dimensions in the rank selection. Then, we cache two dimensions for matrix $\boldsymbol{T}_1$ and two dimensions for matrix $\boldsymbol{T}_2$. Now, assume $\boldsymbol{T}_1$ gets updated and the singular values become 1, 2, 3, 3.3, 3.4, 6. Consequently, we should select one dimension from $\boldsymbol{T}_1$ and three dimensions from $\boldsymbol{T}_2$. However, since we only keep two dimensions for $\boldsymbol{T}_2$, the results of the cached sub-matrix for $\boldsymbol{T}_2$ cannot be directly used and need to be re-computed, even if we have cached the intermediate results. To address this issue, instead of caching exactly the same dimensions based on the rank selection process, we maintain a larger number of dimensions to allow some tolerance for updates. In our design, if the rank selection selects d_i dimensions for a matrix $\boldsymbol{T}_i$, then we keep $1.5 \cdot d_i$ dimensions to avoid the aforementioned issue. Besides, to facilitate the rank-selection process, we keep all the singular values of the sub-matrices, which only takes constant space costs. For the singular vectors that are not selected, we sort them based on their corresponding singular values and flush them into the disk so that we can later re-use these pre-computed results if they must be added into the next level by the rank-selection process.

Analysis of Dynamic RaSTree-SVD. Next, we analyze RaSTree-SVD update algorithm. When there is no sub-matrix gets changed, or RaSTree-SVD has updated all the sub-matrices, i.e., $\boldsymbol{T}^{t_\tau} = \boldsymbol{P}^{t_\tau}$. In such cases, following the steps of Algorithm 2 is equivalent to running the operations of Algorithm 1 for $\boldsymbol{P}^{t_\tau}$. In this case, it is trivial to prove that Theorem 3.3 holds for $\boldsymbol{P} = \boldsymbol{P}^{t_\tau}$ by Algorithm 2. Hence, the worst-case time complexity and space cost is the same as our analysis in the static setting. The approximation guarantee of the updated SVD results also perfectly aligns with the re-computation of the static RaSTree-SVD. In case only a subset of sub-matrices are updated, the following theorem shows the theoretical guarantee of our dynamic RaSTree-SVD for the final SVD result.

Theorem 3.4. *Given a high rank input matrix $\boldsymbol{P}^t$ and level $q \geq 2$, Algorithm 2 updates the SVD results with an error bound:*

$$\left\|\boldsymbol{P}^{t_\tau} - \pi^d_{\boldsymbol{T}^{t_\tau}_{q,1}}(\boldsymbol{P}^{t_\tau})\right\|_F \leq \left(\sqrt{1+\epsilon} + \beta(2+\sqrt{1+\epsilon})\right)\|\boldsymbol{P}^{t_\tau}\|_F .$$

where $\pi^d_{\boldsymbol{T}^{t_\tau}_{q,1}}(\boldsymbol{P}^{t_\tau}) = \boldsymbol{P}^{t_\tau}(\boldsymbol{V}^{t_\tau}_{q,1})_d(\boldsymbol{V}^{t_\tau}_{q,1})^\top_d$ is the d-rank matrix generated by projecting each row in $\boldsymbol{P}^{t_\tau}$ onto the subspace spanned by the top-d right singular vectors $(\boldsymbol{V}^{t_\tau}_{q,1})_d$. □

As for the update time, assume that there are b' sub-matrices are selected for SVD update (Algorithm 2 Line 2) and there are NNZ' number of non-zero entries in these b' sub-matrices. Then, the running cost to re-compute the SVD for these b' sub-matrices is $O(NNZ' \cdot \frac{d_1 \log n}{\sqrt{\epsilon}})$. In addition, the number of sub-matrices that needs to re-compute the SVD for level 2 to q is upper bounded by $O(b')$ assuming that q and k_l are constants. Hence, the running cost is $O(b'n \cdot d_1^2)$. The total update cost is hence:

$$O\left(NNZ' \cdot \frac{d_1 \log n}{\sqrt{\epsilon}} + b'n \cdot d_1^2\right).$$

4 THEORETICAL ANALYSIS

Proof of Theorem 2.9. We first have the following theorem to facilate our proof.

Theorem 4.1 (Matrix Pythagorean Theorem [45]). *If two matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ have the same dimension and $\boldsymbol{M}\boldsymbol{N}^T = \boldsymbol{0}$, then $\|\boldsymbol{M}+\boldsymbol{N}\|_F^2 = \|\boldsymbol{M}\|_F^2 + \|\boldsymbol{N}\|_F^2$.* □

Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, for any d-rank orthogonal projection matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, we have:

$$\boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^\top\left(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{X}^\top\right)\boldsymbol{A}^\top = \boldsymbol{0}.$$

By Theorem 4.1, we know:

$$\left\|\boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^T\right\|_F^2 + \left\|\boldsymbol{A}\left(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{X}^\top\right)\right\|_F^2 = \|\boldsymbol{A}\|_F^2.$$

and thus

$$\left\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^\top\right\|_F^2 = \|\boldsymbol{A}\|_F^2 - \left\|\boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^T\right\|_F^2 \geq 0.$$

This implies that the d-rank orthogonal projection reduces the Frobenius norm of the original matrix,: $\left\|\boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^\top\right\|_F \leq \|\boldsymbol{A}\|_F$. Since $\pi^d_{\boldsymbol{B}}(\boldsymbol{A})$ is the rank-d matrix resulting from projecting each row of $\boldsymbol{A}$ onto the subspace spanned by the top-d right singular vectors of $\boldsymbol{B}$, we have $\left\|\pi^d_{\boldsymbol{B}}(\boldsymbol{A})\right\|_F \leq \|\boldsymbol{A}\|_F$. □

Proof of Theorem 3.3. Let $\Omega_h^p = \prod_{c=h}^p k_c$. For the convenience of notation, $\tilde{d}$ is used to denote the rank selection number $d_{l,j}$ for any given matrix with index l, j. Let $\boldsymbol{P}$ denote the input matrix horizontally divided by rows into b sub-matrices, i.e., $\boldsymbol{P} = [\boldsymbol{P}_{1,1}; \boldsymbol{P}_{1,2}; \cdots; \boldsymbol{P}_{1,b}]$, where $b = \Omega_1^q$, and $\boldsymbol{P}_{l,i}$ denote the error-free sub-matrix of the input matrix $\boldsymbol{P}$ corresponding to the sub-matrix $\boldsymbol{T}_{l,i}$ in RasTree-SVD. Define $z = (i-1)\cdot k_l$ for the i-th group in level l. Let $b_l = \Omega_l^q$. Therefore, $\boldsymbol{P} = \left[\boldsymbol{P}_{l,1}; \boldsymbol{P}_{l,2}; \cdots; \boldsymbol{P}_{l,b_l}\right]$ holds for each integer $1 \leq l \leq q$, where $\boldsymbol{P}_{l+1,i} = \left[\boldsymbol{P}_{l,z+1}; \cdots; \boldsymbol{P}_{l,z+k_l}\right]$ for $l \in \{1, 2, \cdots, q-1\}$. We use $\sigma_j(\boldsymbol{P}_{l,i})$ to denote the j-th exact singular values of $\boldsymbol{P}_{l,i}$. Our goal is to bound the approximation error of the final matrix $\boldsymbol{T}_{q,1}$ with respect to the original matrix $\boldsymbol{P}$. We first prove the following statement by induction on level l.

Statement: For any level $l \geq 2, \forall i \in [1, b_l]$, the following holds:

$$\left\|\left(\boldsymbol{T}_{l,i}\right)^\top_{d_{l,i}}\left(\boldsymbol{T}_{l,i}\right)_{d_{l,i}} - \boldsymbol{P}^\top_{l,i}\boldsymbol{P}_{l,i}\right\|_2 \leq \left\|\boldsymbol{T}^\top_{l,i}\boldsymbol{T}_{l,i} - \left(\boldsymbol{T}_{l,i}\right)^\top_{\tilde{d}}\left(\boldsymbol{T}_{l,i}\right)_{\tilde{d}}\right\|_2$$
$$+ \sum_{h=1}^{l-1}\sum_{j=0}^{\Omega_h^{l-1}-1}\left\|\boldsymbol{T}^\top_{h,\Omega_h^{l-1}i-j}\boldsymbol{T}_{h,\Omega_h^{l-1}i-j} - \left(\boldsymbol{T}_{h,\Omega_h^{l-1}i-j}\right)^\top_{\tilde{d}}\left(\boldsymbol{T}_{h,\Omega_h^{l-1}i-j}\right)_{\tilde{d}}\right\|_2 .$$

Base Case: At level $l = 1$, we initialize $\boldsymbol{T}_{1,i} = \boldsymbol{P}_{1,i} = \boldsymbol{U}_{1,i}\Sigma_{1,i}\boldsymbol{V}^\top_{1,i}$. Next, at level $l = 2$, let $z = (i-1)\cdot k_1$ and we set $\boldsymbol{T}_{2,i}$ as follows:

$$\boldsymbol{T}_{2,i} = [(\Sigma_{1,z+1})_{d_{1,z+1}}(\boldsymbol{V}^\top_{1,z+1})_{d_{1,z+1}}; \cdots; (\Sigma_{1,z+k_1})_{d_{1,z+k_1}}(\boldsymbol{V}^\top_{1,z+k_1})_{d_{1,z+k_1}}].$$

Let $\boldsymbol{P}_{2,i} = [\boldsymbol{P}_{1,z+1}; \cdots; \boldsymbol{P}_{1,z+k_1}]$ be the original sub-matrix corresponding to $\boldsymbol{T}_{2,i}$ in the input matrix $\boldsymbol{P}$, we have:

$$\boldsymbol{P}^\top_{2,i}\boldsymbol{P}_{2,i} = \sum_{j=0}^{k_1-1}\boldsymbol{T}^\top_{1,ik_1-j}\boldsymbol{T}_{1,ik_1-j}.$$

Then we can derive the following bound of the result:

$$\begin{aligned}
&\left\|\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}^{\top}\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}-\boldsymbol{P}_{2,i}^{\top}\boldsymbol{P}_{2,i}\right\|_2\\
=&\left\|\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}^{\top}\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}-\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}+\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}-\boldsymbol{P}_{2,i}^{\top}\boldsymbol{P}_{2,i}\right\|_2\\
\leq&\left\|\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}-\boldsymbol{P}_{2,i}^{\top}\boldsymbol{P}_{2,i}\right\|_2+\left\|\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}^{\top}\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}-\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}\right\|_2\\
=&\left\|\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}-\sum_{j=0}^{k_1-1}\boldsymbol{T}_{1,ik_1-j}^{\top}\boldsymbol{T}_{1,ik_1-j}\right\|_2+\left\|\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}^{\top}\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}-\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}\right\|_2\\
=&\left\|\sum_{j=0}^{k_1-1}\left(\boldsymbol{T}_{1,k_1i-j}^{\top}\boldsymbol{T}_{1,k_1i-j}-\left(\boldsymbol{T}_{1,k_1i-j}\right)_{d_{1,k_1i-j}}^{\top}\left(\boldsymbol{T}_{1,k_1i-j}\right)_{d_{1,k_1i-j}}\right)\right\|_2\\
&+\left\|\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}-\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}^{\top}\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}\right\|_2\\
\leq&\sum_{j=0}^{k_1-1}\left\|\boldsymbol{T}_{1,k_1i-j}^{\top}\boldsymbol{T}_{1,k_1i-j}-\left(\boldsymbol{T}_{1,k_1i-j}\right)_{d_{1,k_1i-j}}^{\top}\left(\boldsymbol{T}_{1,k_1i-j}\right)_{d_{1,k_1i-j}}\right\|_2\\
&+\left\|\boldsymbol{T}_{2,i}^{\top}\boldsymbol{T}_{2,i}-\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}^{\top}\left(\boldsymbol{T}_{2,i}\right)_{d_{2,i}}\right\|_2. \qquad (2)
\end{aligned}$$

Hence, the statement holds for base case $l = 2$ for any $i \in [1, b_2]$.
Inductive Hypothesis: For $l = p - 1$, we define $\boldsymbol{T}_{p-1,i}$ as follows:

$$\begin{aligned}
\boldsymbol{T}_{p-1,i}=&[(\Sigma_{p-2,k_{p-2}i-(k_{p-2}-1)})_{\tilde{d}}(\boldsymbol{V}_{p-2,k_{p-2}i-(k_{p-2}-1)}^{\top})_{\tilde{d}};\\
&\cdots;(\Sigma_{p-2,k_{p-2}i})_{\tilde{d}}(\boldsymbol{V}_{p-2,k_{p-2}i}^{\top})_{\tilde{d}}].
\end{aligned}$$

Let $\boldsymbol{P}_{p-1,i} = [\boldsymbol{P}_{1,\Omega_1^{p-2}-(\Omega_1^{p-2}-1)};\ldots;\boldsymbol{P}_{1,\Omega_1^{p-2}i}]$ be the original submatrix corresponding to $\boldsymbol{T}_{p-1,i}$ in the $\boldsymbol{P}$ matrix.

Assume the statement is true for $l = p - 1, \forall i \in [1, b_{p-1}]$, i.e.,

$$\begin{aligned}
&\left\|\left(\boldsymbol{T}_{p-1,i}\right)_{d_{p-1,i}}^{\top}\left(\boldsymbol{T}_{p-1,i}\right)_{d_{p-1,i}}-\boldsymbol{P}_{p-1,i}^{\top}\boldsymbol{P}_{p-1,i}\right\|_2\\
\leq&\sum_{h=1}^{p-2}\sum_{j=0}^{\Omega_h^{p-2}-1}\left\|\boldsymbol{T}_{h,\Omega_h^{p-2}i-j}^{\top}\boldsymbol{T}_{h,\Omega_h^{p-2}i-j}-\left(\boldsymbol{T}_{h,\Omega_h^{p-2}i-j}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{h,\Omega_h^{p-2}i-j}\right)_{\tilde{d}}\right\|_2\\
&+\left\|\boldsymbol{T}_{p-1,i}^{\top}\boldsymbol{T}_{p-1,i}-\left(\boldsymbol{T}_{p-1,i}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{p-1,i}\right)_{\tilde{d}}\right\|_2. \qquad (3)
\end{aligned}$$

Inductive Step: we show that it holds for $l = p - 1 + 1 = p$.

For $l = p$, we define $z = (i - 1) \cdot k_{p-1}$ and set $\boldsymbol{T}_{p,i}$ as follows:

$$\begin{aligned}
\boldsymbol{T}_{p,i}=&[(\Sigma_{p-1,k_{p-1}i-(k_{p-1}-1)})_{\tilde{d}}(\boldsymbol{V}_{p-1,k_{p-1}i-(k_{p-1}-1)}^{\top})_{\tilde{d}};\\
&\cdots;(\Sigma_{p-1,k_{p-1}i})_{\tilde{d}}(\boldsymbol{V}_{p-1,k_{p-1}i}^{\top})_{\tilde{d}}].
\end{aligned}$$

Let $\boldsymbol{P}_{p,i} = [\boldsymbol{P}_{1,\Omega_1^{p-1}i-(\Omega_1^{p-1}-1)};\ldots;\boldsymbol{P}_{1,\Omega_1^{p-1}i}]$ be the original submatrix corresponding to $\boldsymbol{T}_{p,i}$ in the $\boldsymbol{P}$ matrix. The frobenius norm of the factorization result with the original matrix is as follows:

$$\begin{aligned}
&\left\|\left(\boldsymbol{T}_{p,i}\right)_{d_{p,i}}^{\top}\left(\boldsymbol{T}_{p,i}\right)_{d_{p,i}}-\boldsymbol{P}_{p,i}^{\top}\boldsymbol{P}_{p,i}\right\|_2\\
\leq&\left\|\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}-\boldsymbol{T}_{p,i}^{\top}\boldsymbol{T}_{p,i}\right\|_2+\left\|\boldsymbol{T}_{p,i}^{\top}\boldsymbol{T}_{p,i}-\boldsymbol{P}_{p,i}^{\top}\boldsymbol{P}_{p,i}\right\|_2\\
=&\left\|\sum_{j=0}^{k_{p-1}-1}\left(\boldsymbol{P}_{p-1,k_{p-1}i-j}^{\top}\boldsymbol{P}_{p-1,k_{p-1}i-j}-\left(\boldsymbol{T}_{p-1,k_{p-1}i-j}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{p-1,k_{p-1}i-j}\right)_{\tilde{d}}\right)\right\|_2\\
&+\left\|\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}-\boldsymbol{T}_{p,i}^{\top}\boldsymbol{T}_{p,i}\right\|_2\\
\leq&\sum_{j=0}^{k_{p-1}-1}\left\|\boldsymbol{P}_{p-1,k_{p-1}i-j}^{\top}\boldsymbol{P}_{p-1,k_{p-1}i-j}-\left(\boldsymbol{T}_{p-1,k_{p-1}i-j}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{p-1,k_{p-1}i-j}\right)_{\tilde{d}}\right\|_2\\
&+\left\|\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}-\boldsymbol{T}_{p,i}^{\top}\boldsymbol{T}_{p,i}\right\|_2\\
\leq&\sum_{h=1}^{p-1}\sum_{j=0}^{\Omega_h^{p-1}-1}\left\|\boldsymbol{T}_{h,\Omega_h^{p-1}i-j}^{\top}\boldsymbol{T}_{h,\Omega_h^{p-1}i-j}-\left(\boldsymbol{T}_{h,\Omega_h^{p-1}i-j}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{h,\Omega_h^{p-1}i-j}\right)_{\tilde{d}}\right\|_2\\
&+\left\|\boldsymbol{T}_{p,i}^{\top}\boldsymbol{T}_{p,i}-\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}\right\|_2 \text{ (By Inductive Hypothesis).} \qquad (4)
\end{aligned}$$

Then it holds for $l = p - 1 + 1 = p$.
Conclusion: By induction, the statement is true for all $l \geq 2$.

Generally, for $\forall l \in \{2, 3, \cdots, q\}, i \in \{1, 2, \cdots, b_l\}$,

$$\begin{aligned}
&\left\|\left(\boldsymbol{T}_{l,i}\right)_{d_{l,i}}^{\top}\left(\boldsymbol{T}_{l,i}\right)_{d_{l,i}}-\boldsymbol{T}_{l,i}^{\top}\boldsymbol{T}_{l,i}\right\|_l\\
=&\left\|\sum_{j=0}^{k_{l-1}-1}\boldsymbol{V}_{l-1,k_{l-1}i-j}\left((\Sigma_{l-1,k_{l-1}i-j}^2)_{d_{l-1,k_1i-j}}-\Sigma_{l-1,k_{l-1}i-j}^2\right)\boldsymbol{V}_{l-1,k_{l-1}i-j}^{\top}\right\|_2\\
\leq&\sum_{j=0}^{k_{l-1}-1}\left\|\boldsymbol{V}_{l-1,k_{l-1}i-j}\left((\Sigma_{l-1,k_{l-1}i-j}^2)_{d_{l-1,k_1i-j}}-\Sigma_{l-1,k_{l-1}i-j}^2\right)\boldsymbol{V}_{l-1,k_{l-1}i-j}^{\top}\right\|_2\\
\leq&\sum_{j=0}^{k_{l-1}-1}\left\|\left(\Sigma_{l-1,k_{l-1}i-j}\right)_{d_{l-1,k_{l-1}i-j}}^2-\left(\Sigma_{l-1,k_{l-1}i-j}\right)^2\right\|_2\\
=&\sum_{j=0}^{k_{l-1}-1}\sigma_{d_{l-1,k_{l-1}i-j}+1}^2(\boldsymbol{T}_{l-1,k_{l-1}i-j}). \qquad (5)
\end{aligned}$$

By Eq. 5, Eq. 4 can be rewritten as:

$$\begin{aligned}
&\left\|\left(\boldsymbol{T}_{l,i}\right)_{d_{l,i}}^{\top}\left(\boldsymbol{T}_{l,i}\right)_{d_{l,i}}-\boldsymbol{P}_{l,i}^{\top}\boldsymbol{P}_{l,i}\right\|_2\leq\left\|\boldsymbol{T}_{l,i}^{\top}\boldsymbol{T}_{l,i}-\left(\boldsymbol{T}_{l,i}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{l,i}\right)_{\tilde{d}}\right\|_2\\
&+\sum_{h=1}^{l-1}\sum_{j=0}^{\Omega_h^{l-1}-1}\left\|\boldsymbol{T}_{h,\Omega_h^{l-1}i-j}^{\top}\boldsymbol{T}_{h,\Omega_h^{l-1}i-j}-\left(\boldsymbol{T}_{h,\Omega_h^{l-1}i-j}\right)_{\tilde{d}}^{\top}\left(\boldsymbol{T}_{h,\Omega_h^{l-1}i-j}\right)_{\tilde{d}}\right\|_2\\
\leq&\sum_{h=1}^{l-1}\sum_{j=0}^{\Omega_h^{l-1}-1}\sigma_{d_{h,\Omega_h^{l-1}i-j}+1}^2(\boldsymbol{T}_{h,\Omega_h^{l-1}i-j})+\sigma_{d_{l,i}+1}^2(\boldsymbol{T}_{l,i}).
\end{aligned}$$

In level 2, $\boldsymbol{T}_{1,i} = \boldsymbol{P}_{1,i}$, by the rank-selection, we have:

$$\sum_{j=0}^{k_1-1}\sigma_{d_{1,k_1i-j}+1}^2(\boldsymbol{P}_{1,k_1i-j})\leq\sum_{j=0}^{k_1-1}\sigma_{d_1+1}^2(\boldsymbol{P}_{1,k_1i-j}). \qquad (6)$$

For each level $h \geq 2$, by Eq. 5 and the rank-selection with $\alpha_{l,i} = \frac{\sigma^2_{d_l+1}(\boldsymbol{T}_{l,i})}{\sum_{x=1}^{b}\sigma^2_{d_1+1}(\boldsymbol{P}_{1,x})}$, we have:

$$\sum_{h=2}^{p-1}\sum_{j=0}^{\Omega_h^{p-1}-1}\left\|\boldsymbol{T}^\top_{h,\Omega_h^{p-1}i-j}\boldsymbol{T}_{h,\Omega_h^{p-1}i-j} - \left(\boldsymbol{T}_{h,\Omega_h^{p-1}i-j}\right)^\top_{\tilde{d}}\left(\boldsymbol{T}_{h,\Omega_h^{p-1}i-j}\right)_{\tilde{d}}\right\|_2$$
$$+\left\|\boldsymbol{T}_{p,i}\boldsymbol{T}^\top_{p,i} - \left(\boldsymbol{T}_{p,i}\right)^\top_{\tilde{d}}\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}}\right\|_2$$
$$=\sum_{h=2}^{p-1}\sum_{j=0}^{\Omega_1^{p-1}-1}\alpha_{h,\Omega_1^{p-1}i-j}(\sum_{x=1}^{b}\sigma^2_{d_1+1}(\boldsymbol{P}_{1,x})) + \alpha_{p,i}(\sum_{x=1}^{b}\sigma^2_{d_1+1}(\boldsymbol{P}_{1,x})). \tag{7}$$

Then in level p, combine Eq. 6 and Eq. 7, we can rewrite Eq. 4 as

$$\left\|\left(\boldsymbol{T}_{p,i}\right)^\top_{d_{p,i}}\left(\boldsymbol{T}_{p,i}\right)_{d_{p,i}} - \boldsymbol{P}^\top_{p,i}\boldsymbol{P}_{p,i}\right\|_2$$
$$\leq(1+\sum_{j=0}^{\Omega_1^{p-1}-1}\sum_{h=2}^{p}\alpha_{h,\Omega_1^{p-1}i-j} + \alpha_{p,i})(\sum_{x=1}^{b}\sigma^2_{d_1+1}(\boldsymbol{P}_{1,x})). \tag{8}$$

By Definition 2.6, the numeric rank of $\boldsymbol{P}_{1,x} - (\boldsymbol{P}_{1,x})_{d_1}$ is

$$r_{1,x} = \frac{\|\boldsymbol{P}_{1,x} - (\boldsymbol{P}_{1,x})_{d_1}\|_F^2}{\|\boldsymbol{P}_{1,x} - (\boldsymbol{P}_{1,x})_{d_1}\|_2^2},$$

which is a smooth relaxation of $\textbf{rank}(\boldsymbol{P}_{1,x} - (\boldsymbol{P}_{1,x})_{d_1})$.

$$\left\|\left(\boldsymbol{T}_{p,i}\right)^\top_{d_{p,i}}\left(\boldsymbol{T}_{p,i}\right)_{d_{p,i}} - \boldsymbol{P}^\top_{p,i}\boldsymbol{P}_{p,i}\right\|_2$$
$$\leq(1+\sum_{j=0}^{\Omega_1^{p-1}-1}\sum_{h=2}^{p}\alpha_{h,\Omega_1^{p-1}i-j} + \alpha_{p,i})(\sum_{x=1}^{b}\sigma^2_{d_1+1}(\boldsymbol{P}_{1,x}))$$
$$\leq(1+\sum_{j=0}^{\Omega_1^{p-1}-1}\sum_{h=2}^{p}\alpha_{h,\Omega_1^{p-1}i-j} + \alpha_{p,i})(\sum_{x=1}^{b}\frac{1}{r_{1,x}}\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2).$$

Then we know

$$2d\cdot\left\|\left(\boldsymbol{T}_{p,i}\right)^\top_{\tilde{d}}\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}} - \boldsymbol{P}^\top_{p,i}\boldsymbol{P}_{p,i}\right\|_2$$
$$\leq 2d(1+\sum_{j=0}^{\Omega_1^{p-1}-1}\sum_{h=2}^{p}\alpha_{h,\Omega_1^{p-1}i-j} + \alpha_{p,i})(\sum_{x=1}^{b}\frac{1}{r_{1,x}}\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2)$$
$$=\frac{2d}{r'}(1+\sum_{j=0}^{\Omega_1^{p-1}-1}\sum_{h=2}^{p}\alpha_{h,\Omega_1^{p-1}i-j} + \alpha_{p,i})(\sum_{x=1}^{b}\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2),$$

where

$$r' = \frac{\sum_{x=1}^{b}\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2}{(\sum_{x=1}^{b}\frac{1}{r_{1,x}}\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2)}$$

with $\min(r_{1,x}) < r' < \max(r_{1,x}), \forall x \in [1, 2, ..., b]$. Because any $\sigma^2_{d_l+1}(\boldsymbol{T}_{l,i})$ is far smaller than $\sum_{x=1}^{b}\sigma^2_{d_1+1}(\boldsymbol{P}_{1,x})$. We know $\alpha_{l,i} = \frac{\sigma^2_{d_l+1}(\boldsymbol{T}_{l,i})}{\sum_{x=1}^{b}\sigma^2_{d_1+1}(\boldsymbol{P}_{1,x})}$ is a small number. At the same time, for a high rank matrix, r' is far larger than $2\cdot d$. Then we have:

$$\frac{2d}{r'}(1+\sum_{j=0}^{\Omega_1^{p-1}-1}\sum_{h=2}^{p}\alpha_{h,\Omega_1^{p-1}i-j} + \alpha_{p,i})$$

should be a small number. Then by adjusting the constant, we have

$$2d\cdot\left\|\left(\boldsymbol{T}_{p,i}\right)^\top_{\tilde{d}}\left(\boldsymbol{T}_{p,i}\right)_{\tilde{d}} - \boldsymbol{P}^\top_{p,i}\boldsymbol{P}_{p,i}\right\|_2$$
$$\leq\frac{2d}{r'}(1+\sum_{j=0}^{\Omega_1^{p-1}-1}\sum_{h=2}^{p}\alpha_{h,\Omega_1^{p-1}i-j} + \alpha_{p,i})(\sum_{x=1}^{b}\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2)$$
$$\leq\epsilon(\sum_{x=1}^{b}\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2).$$

Let $[(\boldsymbol{P})_d]_{1,\Omega_1^{p-1}i-j}$ denote the sub-matrix of the best low-rank approximation of $\boldsymbol{P}$, i.e. $(\boldsymbol{P})_d$, in the $\boldsymbol{P}_{1,\Omega_1^{p-1}i-j}$ region.

$$2d\cdot\left\|\left(\boldsymbol{T}_{p,i}\right)^\top_{d_{p,i}}\left(\boldsymbol{T}_{p,i}\right)_{d_{p,i}} - \boldsymbol{P}^\top_{p,i}\boldsymbol{P}_{p,i}\right\|_2 \leq \sum_{x=1}^{b}\epsilon\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2$$
$$\leq\epsilon\sum_{x=1}^{b}\left\|\boldsymbol{P}_{1,x} - [(\boldsymbol{P})_{d_1}]_{1,x}\right\|_{\mathrm{F}}^2 = \epsilon\left\|\boldsymbol{P} - (\boldsymbol{P})_{d_1}\right\|_{\mathrm{F}}.$$

Finally in level q, we form $\boldsymbol{T_{q,1}}$ as follows:

$$\boldsymbol{T}_{q,1} = [(\Sigma_{q-1,k_{q-1}i-(k_{q-1}-1)})_{\tilde{d}}(\boldsymbol{V}^\top_{q-1,k_{p-1}i-(k_{q-1}-1)})_{\tilde{d}};$$
$$\cdots;(\Sigma_{q-1,k_{q-1}i})_{\tilde{d}}(\boldsymbol{V}^\top_{q-1,k_{q-1}i})_{\tilde{d}}].$$

Let $\boldsymbol{P}_{q,1} = [\boldsymbol{P}_{1,\Omega_1^{q-1}-(\Omega_1^{q-1}-1)};\cdots;\boldsymbol{P}_{1,\Omega_1^{q-1}}] = \boldsymbol{P}$ be the original sub-matrix corresponding to $\boldsymbol{T}_{q,1}$ in the $\boldsymbol{P}$ matrix. Thus we know:

$$2d\cdot\left\|\boldsymbol{T}^\top_{q,1}\boldsymbol{T}_{q,1} - \boldsymbol{P}^\top\boldsymbol{P}\right\|_2 \leq \sum_{x=1}^{b}\epsilon\left\|\boldsymbol{P}_{1,x} - \left(\boldsymbol{P}_{1,x}\right)_{d_1}\right\|_{\mathrm{F}}^2$$
$$\leq\epsilon\sum_{x=1}^{b}\left\|\boldsymbol{P}_{1,x} - [(\boldsymbol{P})_{d_1}]_{1,x}\right\|_{\mathrm{F}}^2 = \epsilon\left\|\boldsymbol{P} - (\boldsymbol{P})_{d_1}\right\|_{\mathrm{F}}.$$

Since $\sum_{j=0}^{k} d_{q-1,j} = d$, by Lemma 2.10, we could have the Frequent Direction to SVD error bound as

$$\left\|\boldsymbol{P} - \pi^d_{\boldsymbol{T}_{q,1}}(\boldsymbol{P})\right\|_F^2 = \left\|\boldsymbol{P} - \pi^d_{\boldsymbol{T}_{q,1}}\boldsymbol{P}\right\|_F^2$$
$$\leq\|\boldsymbol{P} - (\boldsymbol{P})_d\|_F^2 + 2d\cdot\left\|\boldsymbol{T}^\top_{q,1}\boldsymbol{T}_{q,1} - \boldsymbol{P}^\top\boldsymbol{P}\right\|_2$$
$$\leq\|\boldsymbol{P} - (\boldsymbol{P})_d\|_F^2 + \epsilon\left\|\boldsymbol{P} - (\boldsymbol{P})_{d_1}\right\|_F^2.$$

This finishes the proof. □

Proof of Theorem 3.4. When we partially update sub-matrices in the first level following the steps of RaStree-SVD update in Algorithm 2, the operations of Algorithm 2 for $\boldsymbol{P}^{t_\tau}$ is equivalent to the operations of Algorithm 1 for $\boldsymbol{T}^{t_\tau}$. It is trivial to prove that Theorem 3.3 holds for $\boldsymbol{P} = \boldsymbol{T}^{t_\tau}$. Then we only need to bound the difference between $\boldsymbol{T}^{t_\tau}$ and $\boldsymbol{P}^{t_\tau}$:

$$
\begin{aligned}
&\left\| P^{t_\tau} - \pi^d_{T^{t_\tau}_{q,1}}(P^{t_\tau}) \right\|_F \\
\leq &\left\| P^{t_\tau} - T^{t_\tau} + T^{t_\tau} - \pi^d_{T^{t_\tau}_{q,1}}(T^{t_\tau}) + \pi^d_{T^{t_\tau}_{q,1}}(T^{t_\tau}) - \pi^d_{T^{t_\tau}_{q,1}}(P^{t_\tau}) \right\|_F \\
\leq &\left\| T^{t_\tau} - \pi^d_{T^{t_\tau}_{q,1}}(T^{t_\tau}) \right\|_F + \left\| \pi^d_{T^{t_\tau}_{q,1}}(T^{t_\tau}) - \pi^d_{T^{t_\tau}_{q,1}}(P^{t_\tau}) \right\|_F + \left\| T^{t_\tau} - P^{t_\tau} \right\|_F \\
\leq &\left\| T^{t_\tau} - \pi^d_{T^{t_\tau}_{q,1}}(T^{t_\tau}) \right\|_F + \left\| \pi^d_{T^{t_\tau}_{q,1}}(T^{t_\tau}) - \pi^d_{T^{t_\tau}_{q,1}}(P^{t_\tau}) \right\|_F + \left\| T^{t_\tau} - P^{t_\tau} \right\|_F \\
\leq &\sqrt{1+\epsilon} \left\| T^{t_\tau} - [T^{t_\tau}]_{d_1} \right\|_F \\
&+ \left\| \sum_{i=1}^{b} \sum_{t=t_{1,i}}^{t_\tau} \Delta^t_{1,i} \right\|_F + \left\| \pi^d_{T^{t_\tau}_{q,1}}\left(\sum_{i=1}^{b} \sum_{t=t_{1,i}}^{t_\tau} \Delta^t_{1,i} \right) \right\|_F \\
\leq &\sqrt{1+\epsilon} \left\| T^{t_\tau} \right\|_F + 2 \left\| \sum_{i=1}^{b} \sum_{t=t_{1,i}}^{t_\tau} \Delta^t_{1,i} \right\|_F \quad (By \text{ Theorem 2.9}) \\
= &\sqrt{1+\epsilon} \left\| P^{t_\tau} - \sum_{i=1}^{b} \sum_{t=t_{1,i}}^{t_\tau} \Delta^t_{1,i} \right\|_F + 2 \left\| \sum_{i=1}^{b} \sum_{t=t_{1,i}}^{t_\tau} \Delta^t_{1,i} \right\|_F \\
\leq &\sqrt{1+\epsilon} \left\| P^{t_\tau} \right\|_F + (2 + \sqrt{1+\epsilon}) \left\| \sum_{i=1}^{b} \sum_{t=t_{1,i}}^{t_\tau} \Delta^t_{1,i} \right\|_F \\
\leq &\left(\sqrt{1+\epsilon} + \beta(2 + \sqrt{1+\epsilon}) \right) \left\| P^{t_\tau} \right\|_F .
\end{aligned}
$$

This finishes the proof. □

5 EXPERIMENT

In this section, We evaluate our RaSTree-SVD performance against competitors. All experiments are conducted on a Linux machine with 2 CPUs (2.30GHz), 32 cores (64 threads), and 832 GB memory.

5.1 Experimental Settings

Datasets. In our experiments, we focus on large-scale matrices where existing solutions face scalability issues. Hence, we adopt datasets used in the literature that includes billions of entries. The first dataset we use is an image dataset, which can be used for image compression with SVD as shown in the literature [5, 7]. Specifically, we use a large image dataset CelebA [42], which contains 202,599 pictures from 10,177 famous identities. Each picture consists of 178×218 RGB pixels, which we transform into vectors with a column dimension of $178 \times 218 \times 3 = 116,412$. The CelebA dataset contains over 20 billion non-zero entries. Besides, proximity matrices of nodes in graphs are widely utilized in the literature for node embedding, which takes SVD operation on the proximity matrix to derive embeddings for each node. In existing works [17, 56, 58, 60], they first derive the proximity matrix using personalized PageRank as the proximity measure followed by filtering out values below a specific threshold value θ and setting them to zero. Therefore, by adjusting θ, we can control the NNZs in these proximity matrices. We follow [17, 56, 58, 60] and use three popular datasets: YouTube, Orkut, and Twitter. In the default setting (for comparision with other baseline methods), we set θ to ensure that the proximity matrices of YouTube and Orkut contain 10 billion NNZs each, while the proximity matrix of Twitter contains 20 billion NNZs. Later, in scalability test, we vary the number of non-zero entries from 10 billion to 1 trillion to evaluate the scalability of our RaSTree-SVD.The statistics of datasets are shown in Table 3.

Table 3: Statistics of datasets.

Dataset	Matrix Type	m	n	NNZs (default)
CelebA (CA)	Image	202, 599	116, 412	23 billion
YouTube (YT)	Proximity	1, 138, 499	1, 138, 499	10 billion
Orkut (OR)	Proximity	3, 072, 441	3, 072, 441	10 billion
Twitter (TW)	Proximity	41, 652, 230	41, 652, 230	20 billion

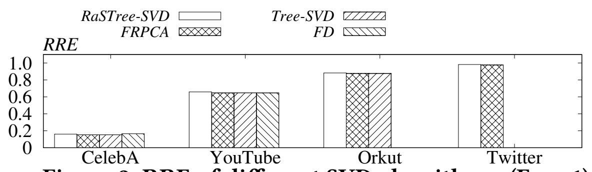


Figure 2: RRE of different SVD algorithms. (Exp. 1)

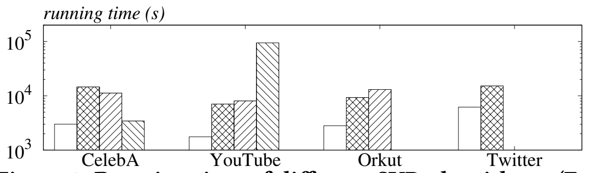


Figure 3: Running time of different SVD algorithms. (Exp. 1)

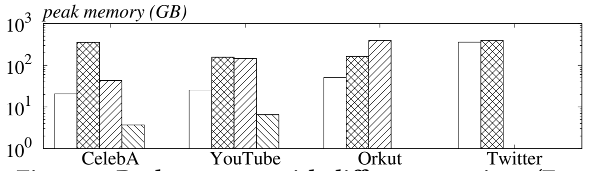
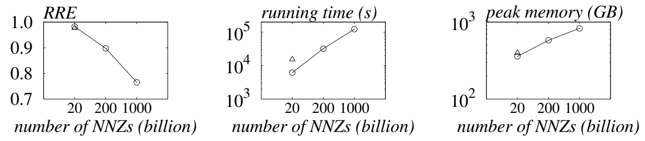
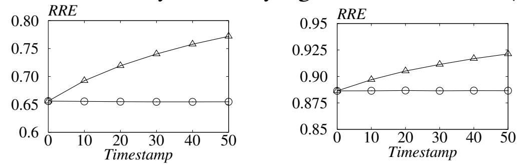


Figure 4: Peak memory with different matrices. (Exp. 1)



(a) RRE (b) Running Time (c) Peak Memory

Figure 5: Scalability test: varying NNZs on Twitter. (Exp. 2)



(a) YouTube (b) Orkut

Figure 6: Impact of updates. (Exp. 3)

Competitors. We compare our RaSTree-SVD against state-of-the-art SVD algorithms, including the state-of-the-art FD method SVS [31], the randomized SVD method FRPCA [18], and an incremental SVD method Tree-SVD [17]. All competitors are run in parallel using 64 threads. We obtain their source codes from GitHub and use the default settings recommended by their authors. For our methods, RaSTree-SVD refers to the static version of RaSTree-SVD, while RaSTree-SVD-U refers to its dynamic algorithm.

Evaluation Metrics. We evaluate RaSTree-SVD and other competitors using the following four metrics: *(i)* Relative Reconstruction

Error (RRE): $\frac{\|P-PV'V'^T\|_F}{\|P\|_F}$, where V' is the output matrix; *(ii)* the runing time; *(iii)* the peak memory consumption; *(iv)* the scalability as the NNZs of the input matrix $P \in \mathbb{R}^{m\times n}$ increases from billion scale to trillion scale.

Parameter Settings. We set the total number of levels $q = 3$, the merge numbers $k_1 = 8, k_2 = 4, k_3 = 2$, the dimension number in each level l as $d_l = \frac{d}{k_l}$ as default values. For competitors, we use their default settings. To ensure that baseline methods can finish in reasonable time, we set d larger on smaller-scale matrices ($n \times m$). In particular, we set $d = 256$ on the CelebA dataset, 128 on the YouTube and Orkut datasets, and 64 on the Twitter dataset. We will further explore the impact of d and d_1 on our RaSTree-SVD in another set of experiments.

5.2 Static RaSTree-SVD

Exp1: Static SVD Comparison. In the first set of experiments, we compare our RaSTree-SVD against other SVD alternatives: FD, Tree-SVD, and FRPCA. As shown in Figure 2, our RaSTree-SVD shows similar relative reconstruction error (RRE) as Tree-SVD and FRPCA. In terms of running time, as we can observe in Figure 3, our RaSTree-SVD is up to 5x faster than both FRPCA and Tree-SVD while preserving the same accuracy of the final results. Tree-SVD cannot finish on the Twitter dataset as it runs out of memory. FD only consumes a small memory cost as it uses the matrix sketching to summarize the input matrix. However, it can only finish on datasets with a moderate dimension $n \times m$ like CelebA. However, when $n \times m$ further increases, the running time of FD increases significantly as shown on YouTube (being over 40x slower than RaSTree-SVD) and cannot finish processing on Orkut and Twitter within a week as its time complexity depends on $n \times m$. Thus, FD is not as efficient as other methods when computing SVD results on large scale matrices. Finally, we examine the memory cost of different SVD algorithms. As shown in Figure 4, FRPCA consumes up to 17x memory cost as that of RaSTree-SVD; Tree-SVD consumes up to 8x memory cost as that of RaSTree-SVD and fails to complete on the Twitter dataset as it runs out of memory. These results demonstrate that our RaSTree-SVD gains the best trade-off among accuracy, running time, and memory cost, making it the preferred choice among these SVD algorithms.

Exp. 2: Scalability Test. In the second set of experiments, we examine the scalability of our RaSTree-SVD (circle) against FRPCA (triangle), the only competitor capable of completing SVD computations on the Twitter dataset while providing identical accuracy. In this set of experiments, we increase the NNZs from 20 billion to 1 trillion. Unfortunately, FRPCA cannot scale to matrices when we increase the matrix size to 200 billion non-zero entries, not to mention matrices with 1 trillion non-zero entries. Clearly, the running time increases with the increase of NNZs, which is expected as our RaSTree-SVD linearly depends on NNZs. In addition, the memory cost also increases accordingly. Note that even with NNZs increasing from 20 billion to 1 trillion, the memory cost of our RaSTree-SVD only increases by around 2.3x, going from around 360GB to around 830GB. This clearly demonstrates the scalability of our method, due to our carefully designed methodology: dividing the input matrix into sub-matrices and incorporating a rank-selection mechanism to reduce the memory cost of the cached intermediate results. In contrast, FRPCA needs to load the entire input matrix, which limits its scalability when the matrix size grows.

5.3 RaSTree-SVD Update

Exp. 3: Importance of Dynamic Updates. In this set of experiments, we aim to demonstrate the significance of updating the SVD results. To simulate arbitrary updates, we conduct a process where rows and columns are randomly deleted simultaneously from the input matrices and subsequently reintegrated back into the original matrix. We divide the updates into 50 timestamps where at the initial timestamp, the matrix has a Frobenius norm equivalent to 50% of the original matrix, with each timestamp increasing the Frobenius norm by 1% of the original matrix. Hence, after 50 timestamps, all removed rows and columns are added back. Figure 6 shows the results when we only compute the SVD result at the beginning (triangle) against the result when we perform SVD updates (circle). Due to limited space, we only show the result on the YouTube and Orkut datasets. The results on other datasets show a similar trend and thus are omitted due to limited space. Clearly, employing SVD updates lead to significantly lower RRE compared to using the obsolete SVD results computed at the beginning. This emphasizes the importance of updating SVD results.

Exp. 4: Dynamic RaSTree-SVD. In this set of experiments, we examine the effectiveness and efficiency of our dynamic RaSTree-SVD algorithm, dubbed as RaSTree-SVD-U. We compare our dynamic RaSTree-SVD-U against the static version of RaSTree-SVD. As mentioned in Section 3.2, even though Tree-SVD also includes a dynamic method, it needs to keep track of $\|P_i - (P_i)_d\|_F$ for each sub-matrix, which only works for rectangular matrices with a small number of rows. In our setting, the number of rows are generally huge and thus Tree-SVD is impractical for our problem.

In real world scenarios, the matrix data to be analyzed usually involves batch updates. Therefore, we evaluate the update performance under the previous batch setting. Specifically, we evaluate performance from the 40th timestamp to the 50th timestamp in Exp. 3, where the matrix at timestamp 40 exhibits a Frobenius norm 90% of the original matrix. For the static RaSTree-SVD, we re-compute SVD results from scratch at each timestamp while RaSTree-SVD-U applies a threshold-based update strategy at each timestamp. We do not test for all 50 timestamps as re-running time the SVD algorithm at each timestamp for the static RaSTree-SVD is time-intensive. Besides, to showcase the scalability of our dynamic RaSTree-SVD algorithm, we adopted the Twitter dataset with 200 billion NNZ entries rather than with 20 billion entries. We did not show the result with the 1 trillion version since it takes too much time to re-compute the SVD results for the static RaSTree-SVD. Figure 7 shows the RRE of both methods. As we can see, both methods have identical RRE values on all tested datasets. Note that our dynamic algorithm is up to an order of magnitude faster than the static version, showcasing a superb performance. This demonstrates the effectiveness and efficiency of our dynamic RaSTree-SVD algorithm.

5.4 Parameter Analysis

In the last set of experiment, we evaluate the influence of dimension d_1 at level 1, which controls the trade-off between accuracy

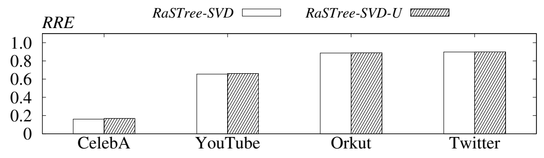


Figure 7: RRE of RaSTree-SVD and RaSTree-SVD-U. (Exp. 4)

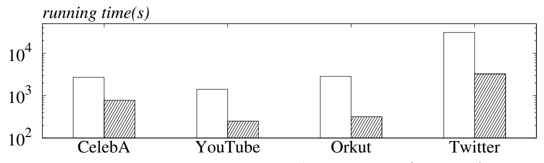


Figure 8: Average update time. (Exp. 4)

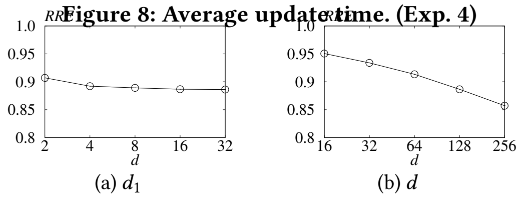


Figure 9: RRE of RaSTree-SVD with varying d_1 and d. (Exp. 5)

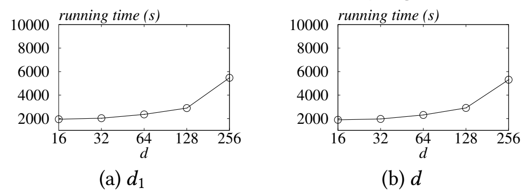


Figure 10: Running time of RaSTree-SVD with varying d_1 and d. (Exp. 5)

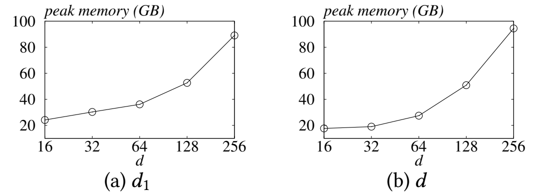


Figure 11: Peak memory of RaSTree-SVD with varying d_1 and varying d. (Exp. 5)

and time/space complexity as shown in Sec. 3.1, and the output dimension d in our RaSTree-SVD algorithm. We report the results on the Orkut dataset only as the observation on other datasets are similar. As we can observe from Figures 9-11, with the increase of d_1, the RRE slightly decreases. However, when d_1 reaches 16, the trends become stable. To explain, by using the rank-selection mechanism, we keep the most important singular vectors and thus preserves a significant portion of the matrix information, showing the efficacy of our rank-selection strategy. Besides, with the increase of d_1, the running time and space cost also increase. This also aligns with our theoretical analysis where our space cost and time complexity increases with d_1. As for the output dimension d, the RRE decreases significantly as we increase the dimension d. This is in contrast to the observations when we increase d_1 as we have carefully designed rank-selection mechanism. This is expected as a larger d preserves more information from the input matrix. In addition, as we increase d, the growth in both running time and peak memory costs is not as fast as the pace of d, indicating a good trade-off of our RaSTree-SVD in terms of running time, space cost and the accuracy.

6 CONCLUSION

In this paper, we present RaStree-SVD, a scalable, efficient and effective SVD algorithm. RaStree-SVD has a dynamic rank selection tree structure to dynamically select the most important information from different sub-matrices. It helps RaStree-SVD reduce time and space cost to avoid the memory bottleneck. Furthermore, utilizing the dynamic rank selection tree, a dynamic update framework is proposed for RaStree-SVD to further accelerate the SVD update. Experiments show that RaStree-SVD is the first SVD algorithm that scales SVD to trillions of non-zeros entries in a single server and supports arbitrary updates with theoretical guarantees.

REFERENCES

[1] Arvind Arasu and Gurmeet Singh Manku. 2004. Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 286–296.

[2] Ainesh Bakshi, Kenneth L. Clarkson, and David P. Woodruff. 2022. Low-rank approximation with $1/\epsilon^{1/3}$ matrix-vector products. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing* (Rome, Italy) *(STOC 2022)*. Association for Computing Machinery, New York, NY, USA, 1130–1143. https://doi.org/10.1145/3519935.3519988

[3] N. Benjamin Erichson, Steven L. Brunton, and J. Nathan Kutz. 2017. Compressed Singular Value Decomposition for Image and Video Processing. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*.

[4] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. 2014. Near-optimal column-based matrix reconstruction. *SIAM J. Comput.* 43, 2 (2014), 687–717.

[5] Ori Bryt and Michael Elad. 2008. Compression of facial images using the K-SVD algorithm. *Journal of Visual Communication and Image Representation* 19, 4 (2008), 270–282.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. 891–900.

[7] Di Chai, Leye Wang, Junxue Zhang, Liu Yang, Shuowei Cai, Kai Chen, and Qiang Yang. 2022. Practical lossless federated singular vector decomposition over billion-scale data. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*. 46–55.

[8] Kenneth L Clarkson and David P Woodruff. 2017. Low-rank approximation and regression in input sparsity time. *JACM* 63, 6 (2017), 1–45.

[9] Jane Cullum, Ralph A Willoughby, and Mark Lake. 1983. A Lanczos algorithm for computing singular values and vectors of large matrices. *SIAM J. Sci. Statist. Comput.* 4, 2 (1983), 197–215.

[10] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SIAM journal on computing* 31, 6 (2002), 1794–1813.

[11] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

[12] James Demmel and William Kahan. 1990. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Statist. Comput.* 11, 5 (1990), 873–912.

[13] Haoran Deng, Yang Yang, Jiahe Li, Haoyang Cai, Shiliang Pu, and Weihao Jiang. 2023. Accelerating Dynamic Network Embedding with Billions of Parameter Updates to Milliseconds. In *SIGKDD*. 414–425.

[14] Haoran Deng, Yang Yang, Jiahe Li, Cheng Chen, Weihao Jiang, and Shiliang Pu. 2024. Fast Updating of Truncated SVD for Representation Learning in Sparse Matrix. In *ICLR*.

[15] Amit Deshpande and Luis Rademacher. 2010. Efficient volume sampling for row/column subset selection. In *2010 ieee 51st annual symposium on foundations of computer science*. IEEE, 329–338.

[16] Petros Drineas, Ravi Kannan, and Michael W Mahoney. 2006. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on computing* 36, 1 (2006), 158–183.

[17] Xinyu Du, Xingyi Zhang, Sibo Wang, and Zengfeng Huang. 2023. Efficient Tree-SVD for Subset Node Embedding over Large Dynamic Graphs. *Proc. ACM Manag. Data* 1, 1, Article 96 (may 2023), 26 pages. https://doi.org/10.1145/3588950

[18] Xu Feng, Yuyang Xie, Mingye Song, Wenjian Yu, and Jie Tang. 2018. Fast Randomized PCA for Sparse Data. In *ACML*. 710–725.

[19] Alan Frieze, Ravi Kannan, and Santosh Vempala. 2004. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)* 51, 6 (2004), 1025–1041.

[20] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. 2016. Frequent directions: Simple and deterministic matrix sketching. *SIAM J. Comput.* 45, 5 (2016), 1762–1792.

[21] Mina Ghashami and Jeff M. Phillips. 2014. Relative Errors for Deterministic Low-Rank Matrix Approximations. In *SODA*. 707–717.

[22] Mina Ghashami, Jeff M. Phillips, and Feifei Li. 2014. Continuous matrix approximation on distributed data. *Proc. VLDB Endow.* 7, 10 (jun 2014), 809–820. https://doi.org/10.14778/2732951.2732954

[23] Gene Golub and William Kahan. 1965. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2, 2 (1965), 205–224.

[24] Gene H Golub, Franklin T Luk, and Michael L Overton. 1981. A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Transactions on Mathematical Software (TOMS)* 7, 2 (1981), 149–169.

[25] Gene H Golub and Christian Reinsch. 1971. Singular value decomposition and least squares solutions. *Numer. Math.* 14 (1971), 403–420.

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

[27] Ming Gu and Stanley C Eisenstat. 1995. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.* 16, 1 (1995), 172–191.

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

[29] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*. 549–558.

[30] Roger A Horn and Charles R Johnson. 2012. *Matrix analysis*. Cambridge university press.

[31] Zengfeng Huang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2021. Communication-Efficient Distributed Covariance Sketch, with Application to Distributed PCA. *JMLR* 22, 80 (2021), 1–38.

[32] Mark A Iwen and BW Ong. 2016. A distributed and incremental SVD algorithm for agglomerative data analysis on large networks. *SIMAX* 37, 4 (2016), 1699–1718.

[33] C.G.J. Jacobi. 1846. Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen. *Journal für die reine und angewandte Mathematik* 30 (1846), 51–94.

[34] Zhongxiao Jia and Datian Niu. 2003. An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition. *SIAM journal on matrix analysis and applications* 25, 1 (2003), 246–265.

[35] EG Kogbetliantz. 1955. Solution of linear equations by diagonalization of coefficients matrix. *Quart. Appl. Math.* 13, 2 (1955), 123–132.

[36] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*. 426–434.

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

[38] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a Social Network or a News Media?. In *WWW*. 591–600.

[39] Cornelius Lanczos. 1950. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. (1950).

[40] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *NeurIPS* 27 (2014).

[41] Edo Liberty. 2013. Simple and Deterministic Matrix Sketching. In *SIGKDD*. 581–588.

[42] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

[43] Yuetian Luo, Rungang Han, and Anru R Zhang. 2021. A Schatten-q low-rank matrix perturbation analysis via perturbation projection error bound. *Linear Algebra Appl.* 630 (2021), 225–240.

[44] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 346–357.

[45] Cameron Musco and Christopher Musco. 2015. Randomized Block Krylov Methods for Stronger and Faster Approximate Singular Value Decomposition. In *NeurIPS*. 1396-–1404.

[46] John C. Nash. 1975. A one-sided transformation method for the singular value decomposition and algebraic eigenproblem. *Comput. J.* 18, 1 (1975), 74–76.

[47] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. 2010. A randomized algorithm for principal component analysis. *SIAM J. Matrix Anal. Appl.* 31, 3 (2010), 1100–1124.

[48] Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. 2017. SVD-Softmax: Fast Softmax Approximation on Large Vocabulary Neural Networks. In *NeurIPS*, Vol. 30.

[49] Yufei Tao and Dimitris Papadias. 2006. Maintaining sliding window skylines on data streams. *IEEE Transactions on Knowledge and Data Engineering* 18, 3 (2006), 377–391.

[50] Trung Vu, Evgenia Chunikhina, and Raviv Raich. 2021. Perturbation expansions and error bounds for the truncated singular value decomposition. *Linear Algebra Appl.* 627 (2021), 94–139.

[51] Zhewei Wei, Xuancheng Liu, Feifei Li, Shuo Shang, Xiaoyong Du, and Ji-Rong Wen. 2016. Matrix sketching over sliding windows. In *Proceedings of the 2016 International Conference on Management of Data*. 1465–1480.

[52] Rafi Witten and Emmanuel Candes. 2015. Randomized algorithms for low-rank matrix factorizations: sharp performance bounds. *Algorithmica* 72 (2015), 264–281.

[53] David P Woodruff et al. 2014. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science* 10, 1–2 (2014), 1–157.

[54] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. 2008. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence* 31, 2 (2008), 210–227.

[55] Aming WU, Suqi Zhao, Cheng Deng, and Wei Liu. 2021. Generalized and Discriminative Few-Shot Object Detection via SVD-Dictionary Enhancement. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 6353–6364. https://proceedings.neurips.cc/paper_files/paper/2021/file/325995af77a0e8b06d1204a171010b3a-Paper.pdf

[56] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. 2020. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. *PVLDB* 13, 5 (2020), 670–683.

[57] Yuan Yin and Zhewei Wei. 2019. Scalable Graph Embeddings via Sparse Transpose Proximities. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 1429–1437. https://doi.org/10.1145/3292500.3330860

[58] Yuan Yin and Zhewei Wei. 2019. Scalable graph embeddings via sparse transpose proximities. In *SIGKDD*. 1429–1437.

[59] Hongyuan Zha and Horst D Simon. 1999. On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing* 21, 2 (1999), 782–791.

[60] Xingyi Zhang, Kun Xie, Sibo Wang, and Zengfeng Huang. 2021. Learning Based Proximity Matrix Factorization for Node Embedding. In *SIGKDD*. 2243–2253.