

---

# QUIC-SVD: Fast SVD Using Cosine Trees

---

Anonymous Author(s)

Affiliation

Address

email

## Abstract

The Singular Value Decomposition is a key operation in many machine learning methods. Its computational cost, however, makes it unscalable and impractical for the massive-sized datasets becoming common in applications. We present a new method, QUIC-SVD, for fast approximation of the full SVD with automatic sample size minimization and empirical relative error control. Previous Monte Carlo approaches have not addressed the full SVD nor benefited from the efficiency of automatic, empirically-driven sample sizing. Our empirical tests show speedups of several orders of magnitude over exact SVD. Such scalability should enable QUIC-SVD to meet the needs of a wide array of methods and applications.

## 1 Introduction

The Singular Value Decomposition (SVD) is a fundamental linear algebraic operation whose abundant useful properties have placed it at the computational center of many methods in machine learning and related fields. Among such methods, principal component analysis (PCA) and its kernel and nonlinear variants are perhaps the most well known, and countless other examples are found in manifold and metric learning, clustering, natural language processing/search, collaborative filtering, bioinformatics and more.

Notwithstanding the utility of the SVD, it is critically bottlenecked by a computational complexity that renders it intractable on massive datasets. Yet massive datasets are rapidly becoming the norm in applications, many of which require real-time responsiveness or would use the SVD more liberally if it weren't so slow to compute. We present a new method, QUIC-SVD, for fast approximation of the full SVD with automatic sample size minimization and empirical relative error control. This algorithm is based on a new type of data partitioning tree, the cosine tree, that shows excellent ability to home in on the data features needed for good SVD approximation. We demonstrate several-order-of-magnitude speedups on medium-sized datasets, and verify that approximation error is properly controlled. Based on these results, QUIC-SVD seems able to help address the scale of modern problems and datasets, with the potential to benefit a wide array of methods and applications.

## 2 Background

**Notation and Linear Algebra.** For  $A \in \mathbb{R}^{m \times n}$ , we write  $A_{(i)}$  for the  $i$ th row of  $A$  and  $A^{(j)}$  for the  $j$ th column. We will use  $\mathbb{O}^{m \times n}$  to represent the subset of  $\mathbb{R}^{m \times n}$  whose columns are orthonormal.

The singular value decomposition is defined as follows:

**Definition 1.** Let  $A$  be an  $m \times n$  real matrix of rank  $\rho$ . Then there exists a factorization of the form

$$A = U\Sigma V^T, \tag{1}$$

where  $U$  and  $V$  each have orthonormal columns and are of size  $m \times \rho$  and  $n \times \rho$ , respectively, and  $\Sigma$  is diagonal with entries  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_\rho > 0$ .

---

**Algorithm 1** Extraction of optimal SVD from a subspace.

---

EXTRACTSVD

**Input:**  $A \in \mathbb{R}^{m \times n}$ ,  $\widehat{V} \in \mathbb{O}^{n \times k}$ **Output:**  $U\Sigma V^T$ , the  $m \times n$  SVD of the best approximation to  $A$  with rows restricted to lie in  $\text{colspan}(\widehat{V})$ .

1. Compute  $A\widehat{V}$  and its SVD:  $U'\Sigma'V'^T$ .
  2. Let  $V = \widehat{V}V'$ ,  $\Sigma = ((AV)^T AV)^{1/2}$ , and  $U = AV\Sigma^{-1}$ .
  3. Return  $U, \Sigma, V$ .
- 

Equivalently, we can write the SVD as a weighted sum of rank-one outer products:  $A = \sum_{i=1}^p \sigma_i u_i v_i^T$ , where  $u_i$  and  $v_i$  represent the  $i$ th columns of  $U$  and  $V$ . The columns  $u_i$  and  $v_i$  are referred to as the left and right singular vectors, while the weights  $\sigma_i$  are the singular values. If  $A$  is (symmetric) positive definite, it follows that: 1)  $U = V$ , and 2) the SVD is the same as the eigendecomposition. Thus, the SVD can be used to solve many eigenproblems, such as PCA.

Throughout this paper we assume  $m \geq n$ , in which case computing the exact SVD has  $O(mn^2)$  runtime. This is highly unscalable, rendering exact SVD impractical for large datasets. However, it is often the case that good approximations can be found using subsets of the rows or columns. Given  $m \geq n$ , we focus on using subsets of rows so that we operate in lower-dimensional space (i.e.,  $\mathbb{R}^n$  as opposed to  $\mathbb{R}^m$ ). This is no loss of generality, since whenever  $m < n$  we can perform SVD on the transpose, then swap  $U$  and  $V$  to get the SVD of the original matrix. Alternatively, all row-based methods have analogous column-based versions that can be used in place of transposition.

Of significant interest are low-rank approximations to a matrix. The optimal  $k$ -rank approximation, in the sense of minimizing  $\|A - \widehat{A}\|_F^2$ , is the  $k$ -rank truncation of the SVD:

$$A_k = \sum_{i=1}^k \sigma_i u_i v_i^T = U_k \Sigma_k V_k^T. \quad (2)$$

$A_k$  is the projection of  $A$ 's rows onto the subspace spanned by the top  $k$  right singular vectors, i.e.,  $A_k = AV_k V_k^T$ . Thus, the optimality of  $A_k$  implies that the columns of  $V_k$  span the subspace of dimension at most  $k$  in which the sum-squared-error of the projections of  $A$ 's rows is minimized. Equivalently, the subspace spanned by  $V_k$  *maximizes* the sum of the squared lengths of  $A$ 's projected rows:  $V_k = \operatorname{argmax}_{\widehat{V} \in \mathbb{O}^{n \times k}} \|A\widehat{V}\widehat{V}^T\|_F^2$ . This leads us to a formulation of SVD approximation in which we seek to find a subspace that captures a sufficient fraction of  $A$ 's magnitude. An important procedure we will require is the extraction of the best approximate SVD restricted to a row subspace with basis  $\widehat{V}$ . Algorithm 1 describes this process; versions of it appeared in [1] and [2], but we give a fuller statement of its properties as a lemma.

**Lemma 1.** *Given a target matrix  $A$  and a row basis contained in the columns of  $\widehat{V}$ :*

1. EXTRACTSVD returns a true SVD; in particular, both  $V$  and  $U$  have orthonormal columns, and  $\Sigma$  is diagonal.
2. The matrix  $U\Sigma V^T$  is the optimal squared-error approximation to  $A$  given the constraint that its rows must lie in  $\text{colspan}(\widehat{V})$ .
3. This optimal approximation to  $A$  is exactly equal to the projection of  $A$ 's rows onto the subspace spanned by  $\widehat{V}$ 's basis.

We omit the straightforward proof. The runtime of the procedure is  $O(kmn)$ . Given an adequate subspace basis  $V$ , Algorithm 1 allows us to efficiently compute a high-quality approximation to the SVD within the lower-dimensional subspace. This motivates the essential idea of our approach, which is to leverage the geometric structure of a matrix to efficiently derive compact (i.e., minimal-rank) subspaces in which to carry out the approximate SVD.

**Previous Work.** A recent vein of work centered in the theory and algorithms community has focused on using subsamples to derive SVD-based low-rank matrix approximations. The user must specify a

desired low rank  $k$ , and the algorithms try to output something close to the optimal  $k$ -rank approximation. Note that this is different from approximating a full SVD; in particular, these algorithms often return only a “description” of the low-rank approximation in the form of a suboptimally-aligned subspace basis  $V$  onto which the matrix must be projected to get the low-rank approximation. While this allows for better  $O()$  runtimes, it is insufficient for most SVD applications. Furthermore, requiring the user to specify the rank of the approximation in advance is problematic, as one generally does not know what rank will be sufficient until *after* performing the SVD.<sup>1</sup>

Error bounds are on the reconstructed matrix, in either in either additive or relative form:

$$\|A - \widehat{A}\|_F \leq \|A - A_k\|_F + \epsilon \|A\|_F \quad (\text{additive}) \quad (3)$$

$$\|A - \widehat{A}\|_F^2 \leq (1 + \epsilon) \|A - A_k\|_F^2 \quad (\text{relative}) . \quad (4)$$

Most of the focus has been on lowering theoretical sample complexities and moving from the additive to the relative error bound. Much of the work has consisted of bound improvements with no algorithmic change. Despite this, three novel ideas have emerged to address the problem of efficiently building a subspace to capture the input matrix magnitude. We will discuss each of these from the perspective of iteratively constructing the subspace by sampling some kind of representative points from the matrix, updating the subspace based on those points, and terminating with an SVD extraction as in Algorithm 1 once the subspace is deemed adequate. This is similar to the framework used by Friedland et al. [1] and Deshpande and Vempala [2], in which new samples are first orthogonalized (e.g., by modified Gram-Schmidt (MGS)), then added to the subspace basis.

*Sampling from the length-squared distribution.* In this scheme, rows are sampled from the input matrix according to a probability distribution assigning probability  $p_i = \|A_{(i)}\|_F^2 / \|A\|_F^2$  to the  $i$ th row  $A_{(i)}$ . Introduced in the seminal work of Frieze, Kannan, and Vempala [3], this technique continued to be employed in much of the follow-on work [4, 5]. The length-squared distribution is shown to minimize expected error among all sampling schemes assigning fixed probabilities to the rows. It is essentially an importance sampling scheme, and is sensible in that it assigns higher probabilities to rows that will have more impact on the error objective. However, it has two important weaknesses. First, a row can have high norm while not representing other rows well enough to be a helpful addition to the subspace. Second, the sampling is non-adaptive, in the sense that a point is equally likely to be drawn whether it is already well represented in the subspace or not. Both of these lead to wasted samples and needless inflation of the subspace rank, which in turn slows the final SVD extraction in the subspace.

*Sampling from the residual length-squared distribution.* This method, introduced by Deshpande and Vempala [2], updates the length-squared distribution after each subspace update by setting  $p_i = \|A_{(i)} - \Pi_V(A_{(i)})\|_F^2 / \|A - \Pi_V(A)\|_F^2$ , where  $\Pi_V$  represents projection onto the subspace  $V$ . By adapting the length-squared distribution to be over residuals, this method avoids drawing samples that are already well represented in the subspace. Unfortunately, there is still nothing to enforce that any sample drawn will be representative of other samples with high residuals, and the updating of the residuals requires an expensive  $s$  passes through the matrix for every  $s$  samples that are added. This significantly limits the practical utility of residual importance sampling.

*Random projections.* Sarlós introduced an alternative relative error method based on random projections [6]. The essential idea is to sample linear combinations of rows, where the combination coefficients are random numbers, e.g., drawn from a standard Gaussian. This method is strong where residual length-squared sampling is weak — because all the rows are informing every sample, there is diminished likelihood that a sample will fail to represent a sizeable number of rows. One disadvantage, though, is that the combinations are not informed by importance, e.g., a row of small length could get a high coefficient and wash out many rows of large length whose coefficients are small. Further, each linear combination requires a full matrix pass, and the sampling distribution is non-adaptive, just as in the basic length-squared method. One could imagine doing random projections on the residuals, but it would get costly, requiring two full matrix passes per sample.

Also deserving mention is the randomized sparsification of Achlioptas et al. [7]. These techniques help accelerate standard  $k$ -rank SVD methods that can exploit sparsity in their inner loops, and while they are less related to the approach we will take, they are interesting in their own right.

<sup>1</sup>For instance, when using PCA for dimensionality reduction, it is common to keep as many components as are required to capture a certain percentage of the data magnitude, and this can't be known without first performing the SVD (or equivalent eigensolver).

The three subspace discovery methods we have described possess various strengths and weaknesses, and in some sense the weaknesses may have been overlooked or tolerated because the methods lend themselves to proving elegant bounds. Nonetheless, there is a clear opportunity to combine aspects of these ideas with a more practical slant, which leads to the method we have developed.

### 3 Our Approach

Our first differentiation from prior work is a change of objective: rather than a fixed low-rank approximation, we seek to approximate the full SVD with as high a rank as required to obtain the following relative error bound:  $\|A - \hat{A}\|_F \leq \epsilon \|A\|_F$ , where  $\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^T$  is the matrix reconstructed by our SVD approximation. In contrast to the error bounds of previous methods, which are stated in terms of the unknown optimal  $A_k$ , our error bound is in terms of the known  $A$ . This enables us to use an empirical Monte Carlo technique to determine with high confidence when we have achieved the error target, and therefore to terminate with as few samples and as compact a subspace as possible. Minimizing subspace rank is crucial for speed, as the final SVD extraction is greatly slowed by excess rank when the input matrix is large.

We seek to accomplish this using an iterative subspace buildup as described in the previous section, but with a significantly more efficient subspace discovery method that combines the strengths of the length-squared and random projection ideas while also going beyond them. In particular, we define a new spatial partitioning structure called a *cosine tree*, which leverages the geometrical structure of the problem to quickly home in on good representative samples from the regions least well represented. Key to the efficiency of the iterative buildup is an efficient error checking scheme, which we accomplish by Monte Carlo error estimation at judiciously chosen stages. Such a combination of spatial partitioning trees and Monte Carlo estimation has been used before to good effect [8], and we find it to be a successful pairing here as well.

**Cosine Trees for Efficient Subspace Discovery.** The ideal subspace discovery algorithm would oracularly choose as samples the singular vectors  $v_i$ . Each  $v_i$  is precisely the direction that, added to the subspace spanned by the previous singular vectors, will capture the most residual magnitude over all rows of the matrix. This intuition is the launch point for the cosine tree heuristic.

A full cosine tree is constructed as follows. Starting at the root node, which contains all points (rows), we randomly sample a point to serve as the node center. When used for SVD approximation, we choose this sample from the basic length-squared distribution, that being the cheapest source of information as to sample importance. The remaining points are sorted by the absolute value of their cosine with the center point, then clustered according to whether they are closer to the high or low end of the cosines. The node is then split, with each cluster assigned to a child node that recursively follows the same procedure, stopping when all points in a node are parallel. This splitting procedure is defined in Algorithm 2.

Why is this helpful for subspace discovery? By grouping points that are similar and dissimilar from a center point, we increase the average amount of parallelism (i.e., high cosines) within a node, while moving toward orthogonality (i.e., lower cosines) between nodes. If this worked perfectly, we would end up with all points parallel within each node, and all points orthogonal between nodes. Sampling one point from each node would then give us a perfect set for constructing a lossless subspace. In reality, this ideal is not achieved, but this explanation motivates the benefit gained from

---

**Algorithm 2** Construction of child nodes in a cosine tree.

---

CTNODESPLIT

**Input:** Cosine tree node  $N$  with points contained in the rows of  $X$ ,  $X_{(i)} \in \mathbb{R}^n$

**Output:** Two child nodes obtained by cosine-splitting of the points in  $N$ .

1.  $center = rowSampleL2(X)$ .
  2. Sort  $X_{(i)}$  by descending value of  $|cos(X_{(i)}, center)|$ . If all values are 1, **return**.
  3. Split the sorted  $X_{(i)}$  as close as possible to where  $|cos(X_{(i)}, center)|$  becomes closer to the lowest than the highest value, while also ensuring a minimum fraction of points on each side.
  4. **return** two cosine tree nodes, one containing each side of the split  $X$ .
-

---

**Algorithm 3** Monte Carlo estimation of the squared error (in Frobenius norm) of a matrix projected on a subspace.

---

MCSQERROR

**Input:**  $A \in \mathbb{R}^{m \times n}$ ,  $\widehat{V} \in \mathbb{O}^{n \times k}$ ,  $s \in [1 \dots m]$ ,  $\delta \in [0, 1]$

**Output:**  $q \in \mathbb{R}$ , an upper bound on  $\|A - A\widehat{V}\widehat{V}'\|_F^2$  with probability  $1 - \delta$ .

1.  $S = \text{rowSamplesL2}(A, s)$ .
  2.  $\text{wgtMagSq}[i] = \frac{1}{p_{S(i)}} \|S_{(i)}V\|_F^2$
  3.  $\hat{\mu} = \text{mean}(\text{wgtMagSq})$ ,  $\hat{\sigma}^2 = \text{var}(\text{wgtMagSq})$ ,  $\text{magSqLB} = \text{bound}(\hat{\mu}, \hat{\sigma}^2, s, \delta)$
  4. **return**  $\|A\|_F^2 - \text{magSqLB}$ .
- 

the partitioning. Note that, because Algorithm 2 guarantees some fraction of the points go to each side of the split, the tree depth is bounded by  $\log m$ , and therefore the full tree construction has runtime bounded by  $O(mn \log m)$ .

### 3.1 QUIC-SVD

**Strong error control.** Algorithm 4, QUIC-SVD-STRONG (QUantized Iterative Cosine tree)<sup>2</sup>, specifies a way to leverage cosine trees in the construction of an SVD subspace while providing a strong probabilistic error guarantee. The essence of the procedure is to expand a cosine tree one node at a time, rather than all at once, with new representative samples coming from newly expanded nodes to add to the subspace. Expansion priority is a critical issue. We want the expansion that will lead to an adequate subspace in the smallest number of total expansions. A greedy approximation to this is the node whose points have greatest error when projected onto the current subspace, which we approximate by a Monte Carlo procedure in step 3.d.i. This expansion priority allows us to focus expansion and sampling in the regions of row space that most need it. It gives us an adaptivity similar to residual length-squared sampling, but with the added benefit of focusing samples where an entire groups of points are not being represented, rather than just a single point. Further, tree expansion is much faster than updating the residuals matrix. By choosing the mean as the representative sample from a node, we can get a direction that has higher likelihood of representing many points in the node, since it combines them all. This is similar to the combinational benefit of random projections, but has the added advantage that row norm will naturally give an importance weighting to the combination. All of these factors combine to give the cosine trees the edge in sample efficiency.

A key potential bottleneck in the algorithm is estimating the error of  $A$ 's projection onto the current subspace  $V$ . While we have to do this projection at the end for EXTRACTSVD, meaning the algorithm can't escape an overall  $O(kmn)$  runtime, repeatedly performing the exact projection for intermediate error checking is far too costly. We therefore employ a fast, importance-sampled Monte Carlo estimate as specified in Algorithm 3 and used in step 3.e of QUIC-SVD-STRONG. With the geometric-series scaling of the confidence parameter of the error estimate, we arrive at the following error guarantee for the algorithm.

**Theorem 1.** *Given arguments  $A$ ,  $\epsilon$ , and  $\delta$ , QUIC-SVD-STRONG returns an SVD  $\widehat{U}\widehat{\Sigma}\widehat{V}^T$  such that  $\widehat{A} = \widehat{U}\widehat{\Sigma}\widehat{V}^T$  satisfies  $\|A - \widehat{A}\|_F \leq \epsilon\|A\|_F$  with probability  $1 - \delta$ .*

*Proof sketch.* Two things need to be established: 1) the estimate from MCSQERROR in step 3.e is a valid  $1 - \delta$  lower bound on the true squared error, and 2) the scheduling of the confidence parameter passed to MCSQERROR in 3.e is such that the overall failure probability is no more than  $\delta$ . Assuming the former, the latter is easily shown by a union bound on the geometric series of the confidence parameter (step 3.e). The correctness of the lower bound from MCSQERROR follows from the unbiasedness of the mean and and variance computed in step 3. As long as we use a bound that holds for the unbiased empirical mean and variance (e.g., see [9]), the bound will be valid.  $\square$

**Relaxed error control.** We now present a relaxation of QUIC-SVD-STRONG that uses additional principled approximations to more aggressively speed the computation while still keeping error well under control (this will be demonstrated empirically). The relaxed procedure, QUIC-SVD, is listed

---

<sup>2</sup>Quantized alludes to each node being represented by a single point that is added to the subspace basis.

---

**Algorithm 4** Cosine-tree SVD with strong relative error guarantee.

---

QUIC-SVD-STRONG

**Input:**  $A \in \mathbb{R}^{m \times n}$ ,  $\epsilon \in [0, 1]$ , and  $\delta \in [0, 1]$ **Output:**  $\hat{U}\hat{\Sigma}\hat{V}^T$ , an  $m \times n$  SVD such that  $\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^T$  satisfies  $\|A - \hat{A}\|_F \leq \epsilon\|A\|_F$  with probability  $1 - \delta$ .

1.  $V = []$ ,  $mcSqErr = \|A\|_F^2$ ,  $j = 0$ .
  2.  $pQueue = \text{initPriorityQueue}(CTNode(A, \text{cosine}))$ .
  3. **do until**  $\sqrt{mcSqErr} \leq \epsilon\|A\|_F$ :
    - (a)  $N = pQueue.pop()$ .
    - (b) Remove  $N$ 's basis vectors from  $V$ .
    - (c) **for each** child  $C$  of  $N$ :
      - i.  $V = [V \text{ MGS}(V, C.mean)]$ .
    - (d) **for each** child  $C$  of  $N$ :
      - i.  $errC = \text{MCSQERROR}(C.rows, V, O(\log C.numRows), (1 - \frac{1}{8})^{j++})$ .
      - ii.  $pQueue.add(C, errC)$ .
    - (e)  $mcSqErr = \text{MCSQERROR}(A, V, O(\log m))$ .
  4. **return**  $\text{EXTRACTSVD}(A, V)$ .
- 

---

**Algorithm 5** Cosine-tree SVD with aggressive termination conditions.

---

QUIC-SVD

**Input:**  $A \in \mathbb{R}^{m \times n}$ ,  $\epsilon \in [0, 1]$ **Output:**  $\hat{U}\hat{\Sigma}\hat{V}^T$ , an  $m \times n$  SVD with relative error target  $\epsilon$ .

1.  $V = []$ ,  $mcSqErr = \|A\|_F^2$ .
  2.  $pQueue = \text{initPriorityQueue}(CTNode(A, \text{cosine}))$ .
  3. **while** true
    - (a) **if** three calls to  $\text{MCSQERROR}(A, V, O(\log m))$  yield values  $\leq \epsilon\|A\|_F$ , **break**.
    - (b)  $N = pQueue.pop()$ .
    - (c) Remove  $N$ 's basis vectors from  $V$ .
    - (d) **for**  $i = 1$  **to**  $\text{estimateSplitsNeeded}()$ 
      - i. **for each** child  $C$  of  $N$ :
        - A.  $V = [V \text{ MGS}(V, C.mean)]$ .
      - ii. **for each** child  $C$  of  $N$ :
        - A.  $errC = \text{MCSQERROR}(C.rows, V, O(\log C.numRows), 1)$ .
        - B.  $pQueue.add(C, errC)$ .
  4. **return**  $\text{EXTRACTSVD}(A, V)$ .
- 

in Algorithm 5. The principal changes to the algorithm are: 1) use of the mean estimated error rather than a lower bound, 2) several independent verifications of the mean error estimate being below the threshold before termination, and 3) performing multiple splits between error checks to avoid the cost of error checking after each split. The second of these acts as a counter to the variance induced by the first. We estimate the number of samples needed by fitting a line to the set of previous (number of splits, error) pairs, which gives an idea of how fast error is decreasing with respect to split count. While these changes lead to the loss of our error guarantee, we find that in practice the error performance holds up quite well, typically exceeding the target by no more than 10%.

## 4 Performance

We report the results of two sets of experiments, one comparing the sample efficiency of cosine trees to past methods such as length-squared sampling, and the other evaluating the composite speed and error performance of QUIC-SVD. Due to space considerations we present only a few datasets, and

due to the need to compute the exact SVD as a baseline we limit ourselves to medium-sized matrices. Nonetheless, these results are illustrative of the more general performance of the algorithm.

**Sample efficiency.** Figure 1 shows results for the various subspace discovery methods on two matrices, one a  $2000 \times 1783$  image of a Martian surface (mars), and the other a  $2000 \times 2000$  Gaussian kernel matrix produced by the Madelon dataset from the NIPS 2003 Workshop on Feature Extraction (madelon). Plotted are the relative errors of the input matrix’s projection onto the subspace at each rank. Also shown is the optimal error produced by the exact truncated SVD at each rank.

The clear trend in both graphs is that length-squared sampling, length-squared residual sampling, and random projections all perform very similarly to one another, while the cosine tree subspace dominates them all, most of the time by a significant margin. It is particularly interesting how closely the cosine tree error tracks that of the exact SVD. This would seem to give some justification behind the principle of grouping points according to their degree of parallelism.

**Speedup and error.** In the second set of experiments we evaluate the performance of the overall algorithm. Figure 2 shows results for two image matrices, one being the mars image from the first set of experiments, and the other a  $4656 \times 3923$  scan of the US Declaration of Independence (declaration). On the left side we see how speedup over exact SVD varies with the target error  $\epsilon$ . Speedups range from 200 at  $\epsilon = 0.1$  to over 1600 at  $\epsilon = 0.25$  for mars, and from 118 at  $\epsilon = 0.1$  to nearly 20,000 at  $\epsilon = 0.175$  for declaration. On the right side we show the actual error of the algorithm in comparison to the target. While the actual is in all but one case above the target, it is nevertheless quite close, never exceeding the target by more than 10% in these instances. Overall, the several-order-of-magnitude speedups combined with good, controllable error performance would seem to make QUIC-SVD an attractive option for any algorithm requiring large SVDs.

## 5 Conclusion

We have presented a fast approximate SVD algorithm, QUIC-SVD, and demonstrated several-order-of-magnitude speedups with controlled approximation error on some medium-sized dataset matrices. This algorithm differs from previous related work in that it addresses the full SVD, not a  $k$ -rank truncation, and it uses empirical Monte Carlo error estimates to adaptively determine needed sample sizes, rather than fixing a loose sample size a priori. The algorithm comes in two versions, one with strong theoretical error guarantees and the other with good empirical error control. As part of the algorithm, we have presented a new form of tree, the cosine tree, and demonstrated that it finds good SVD subspaces with significantly better sample efficiency than alternative approaches.

In addition to theoretical justifications, the empirical performance of QUIC-SVD argues for its effectiveness and utility. More work is needed to explore the SVD-using methods to which QUIC-SVD can be applied, particularly with an eye to how controlled error in the SVD will affect the

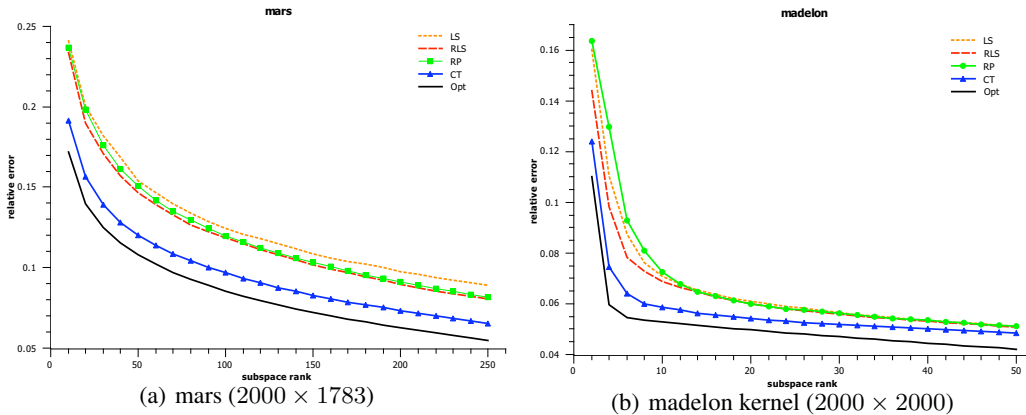


Figure 1: Relative error vs. subspace rank for various subspace discovery methods. LS is length-squared, LSR is length-squared residual, RP is random projection, and CT is cosine tree.

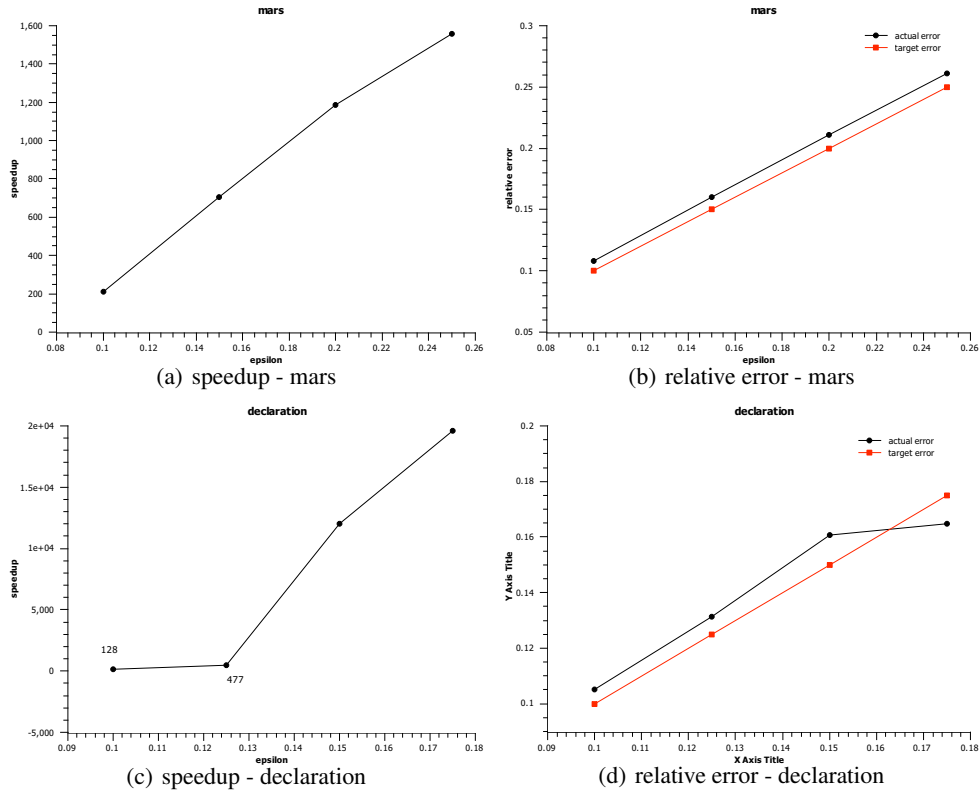


Figure 2: Speedup and actual relative error vs. epsilon for QUIC-SVD on mars and declaration.

quality of the methods using it. We expect there will be many opportunities to enable new applications through the scalability of this approximation.

## References

- [1] Shmuel Friedland, Amir Niknejad, Mostafa Kaveh, and Hossein Zare. Fast Monte-Carlo low rank approximations for matrices. In *Proceedings of Int. Conf. on System of Systems Engineering*, 2006.
- [2] A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. In *10th International Workshop on Randomization and Computation (RANDOM06)*, 2006.
- [3] Alan M. Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. In *IEEE Symposium on Foundations of Computer Science*, pages 370–378, 1998.
- [4] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.
- [5] P. Drineas, E. Drinea, and P. S. Huggins. An experimental evaluation of a monte-carlo algorithm for singular value decomposition. *Lectures Notes in Computer Science*, 2563:279–296, 2003.
- [6] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 143–152, 2006.
- [7] D. Achlioptas, F. McSherry, and B. Scholkopf. Sampling techniques for kernel methods. In *Advances in Neural Information Processing Systems (NIPS) 17*, 2002.
- [8] Michael P. Holmes, Alexander G. Gray, and Charles Lee Isbell, Jr. Ultrafast Monte Carlo for kernel estimators and generalized statistical summations. In *Advances in Neural Information Processing Systems (NIPS) 21*, 2008.
- [9] Jean-Yves Audibert, Remi Munos, and Csaba Szepesvari. Variance estimates and exploration function in multi-armed bandits. Technical report, CERTIS, 2007.