Sampling-Based Decomposition Algorithms for Arbitrary Tensor Networks

Osman Asif Malik∗ Vivek Bharadwaj† Riley Murray‡

Abstract
We show how to develop sampling-based alternating least squares (ALS) algorithms for decomposition of tensors into any tensor network (TN) format. Provided the TN format satisfies certain mild assumptions, resulting algorithms will have input sublinear per-iteration cost. Unlike most previous works on sampling-based ALS methods for tensor decomposition, the sampling in our framework is done according to the exact leverage score distribution of the design matrices in the ALS subproblems. We implement and test two tensor decomposition algorithms that use our sampling framework in a feature extraction experiment where we compare them against a number of other decomposition algorithms.

1 Introduction

Tensor decomposition has emerged as an important tool in data mining and machine learning [24, 4, 5, 11]. Applications in data mining include network analysis, web mining, topic modeling and recommendation systems. Tensor decomposition is used widely in machine learning for things like parameter reduction in neural networks, understanding of deep neural network expressiveness, supervised learning and feature extraction.

Due to the multidimensional nature of tensors, they are inherently plagued by the curse of dimensionality. For example, representing a tensor $X \in \mathbb{R}^{I^1 \times \cdots \times I^N}$ with $N$ modes requires $I^N$ numbers. This exponential dependence on $N$ makes its way into algorithms for computing tensor decompositions. Alternating least squares (ALS) is arguably the most popular and successful approach for computing a wide range of tensor decompositions. When decomposing a tensor $X$, each iteration of ALS involves solving a sequence of least squares problems for which the entries of $X$ feature as the dependent variables. The per-iteration cost of ALS therefore naturally inherits the exponential dependence on $N$.

A large number of papers have sought to reduce the cost of tensor decomposition by leveraging techniques from randomized numerical linear algebra. One particularly interesting line of work [3, 14, 9, 19, 18] seeks to construct ALS algorithms with a per-iteration cost which is sublinear in the number of tensor entries, i.e., $o(I^N)$. Since any algorithm considering all entries of $X$ immediately incurs a cost of $\Omega(I^N)$, these works have all resorted to sampling-based techniques.

∗Lawrence Berkeley National Laboratory, oamalik@lbl.gov
†UC Berkeley, vivek_bharadwaj@berkeley.edu
‡UC Berkeley, rjmurray@berkeley.edu
More precisely, they all sample the ALS subproblems according to the leverage score distribution (or an approximation thereof). This is done efficiently by taking advantage of the special structure of the design matrices in the ALS subproblems for tensor decomposition.

The previous works discussed above develop methods for specific tensor decompositions: The CP decomposition in [3, 14, 18], Tucker decomposition in [9], and the tensor ring decomposition in [19, 18]. In this paper, we consider all decompositions that can be expressed in tensor network (TN) format. The TN format allows for a very wide range of decompositions, including the CP, Tucker, tensor train and tensor ring decompositions. The following summarizes our contributions:

- We first show how to efficiently sample rows of any tall-and-skinny matrix which is in TN format according to the exact leverage score distribution.
- We then show how this sampling technique can be used to yield ALS algorithms with a per-iteration cost which is input sublinear for all TN decompositions that satisfy certain mild assumptions.

The decomposition framework we present builds on the work in [18]. That paper is notable since it provided the first sampling-based ALS methods for CP and tensor ring decomposition with a per-iteration cost depending polynomially on \( N \); the dependence in earlier works was exponential [3, 14, 19]. We are able to substantially simplify the scheme in [18] by entirely avoiding the complicated recursive sketching procedure that it relies on. This makes implementation easier and is also what ensures that the sampling is done according to the exact leverage score distribution rather than an approximation of it. The simplification is also what paves the way for generalization to arbitrary TN formats.

2 Related Work

Cheng et al. [3] develop the first ALS method for CP decomposition with an input sublinear per-iteration cost. Their method uses a mixture of leverage score and row-norm sampling applied to the matricized-tensor-times-Khatri–Rao product (MTTKRP) which arises as a key computational kernel in CP decomposition. Larsen and Kolda [14] use leverage score sampling to reduce the size of each ALS subproblem. Their method has improved theoretical guarantees compared to those in [3] as well as improved scalability due to various practical improvements. Malik and Becker [19] propose a sampling-based ALS method for tensor ring decomposition which also uses leverage score sampling to reduce the size of the ALS subproblems.

All three papers [3, 14, 19] require a number of samples which scales exponentially with the number of input tensor modes, \( N \), for performance guarantees to hold. This translates into a per-iteration cost which is \( \Omega(R^{N+1}) \) for the CP decomposition [3, 14] and \( \Omega(R^{2N+2}) \) for the tensor ring decomposition [19], where \( R \) is the relevant notion of rank. For low-rank decompositions (\( R \ll I \)), the cost will be input sublinear despite this exponential cost, but for higher rank it might not be (recall that, unlike in matrix decomposition, we can have \( R > I \) in tensor decomposition). Malik [18] develop methods for both CP and tensor ring decomposition which avoid this exponential dependence on \( N \), instead improving it to a polynomial dependence. This is achieved by sampling from a distribution much closer to the exact leverage score distribution than the previous works [3, 14, 19] do. Since our work builds on and improves the scheme in [18], it also has a polynomial dependence on \( N \) when used for CP and tensor ring decomposition (see Tables 1 and 2).
Fahrbach et al. [9] develop a method for efficient sampling of ridge regression problems involving Kronecker product design matrices. They use these techniques to achieve an efficient sampling-based ALS method for regularized Tucker decomposition.

Efficient sketching of structured matrices is an active research area with applications beyond tensor decomposition. The recent work by Ma and Solomonik [16] is particularly interesting since it proposes structured sketching operators for general TN matrices. These operators take the form of TNs made up of Gaussian tensors and are therefore dense. When used in ALS for tensor decomposition, their sketching operators therefore need to access all entries of the data tensor in each least squares solve which leads to a per-iteration cost which is at least linear in the input tensor size (and therefore exponential in N).

Due to space constraints, we have focused on previous works that develop sampling-based ALS methods here. There is a large number of works that develop randomized tensor decomposition methods using other techniques. For a comprehensive list of such works, see [18, Sec. 2]. For an overview of work on structured sketching, see [20, Sec. 1.2].

3 Preliminaries

3.1 General Notation

By tensor, we mean a multidimensional array containing real numbers. We refer to a tensor with N indices as an N-way tensor and we say that it is of order N or that it has N modes. We use bold uppercase Euler script letters (e.g., X) to denote tensors of order 3 or greater, bold uppercase letters (e.g., X) to denote matrices, bold lowercase letters (e.g., x) to denote vectors, and regular lowercase letters to denote scalars (e.g., x). We denote entries of an object either in parentheses or with subscripts. For example, X(i, j, k) = X_{ijk} is the entry on position (i, j, k) in the 3-way tensor X, and x(i) = x_i is the i_th entry in the vector x. We use a colon to denote all entries along a particular mode. For example, X(i,:) denotes the i_row of the matrix X. Superscripts in parentheses will be used to denote a sequence of objects. For example, A^{(1)}, ..., A^{(M)} is a sequence of tensors and e^{(i)} is the i_th canonical basis vector. The values that a tensor’s indices can take are referred to as dimensions. For example, if X = (X_{ijk})_{i,j,k=1} then the dimensions of modes 1, 2 and 3 are I, J and K respectively. Note that dimension does not refer to the number of modes, which is 3 for X. For a positive integer I, we use the notation [I] def = {1, ..., I}. For indices i_1 ∈ [I_1], ..., i_N ∈ [I_N], the notation i_1⋯i_N def = 1 + ∑_{n=1}^{N} (i_n - 1) ∏_{j=1}^{n-1} I_j will be used to denote the linear index corresponding to the multi-index (i_1, ..., i_N). The pseudoinverse of a matrix A is denoted by A^+. We use ⊗, ⊙ and ⊛ to denote the Kronecker, Khatri–Rao and Hadamard matrix products which are defined in Section A.

3.2 Tensor Networks

A tensor network (TN) consists of tensors and connections between them that indicate how they should be contracted with each other. Since the mathematical notation easily gets unwieldy when working with TNs, it is common practice to use graphical representations when discussing such networks. Figure 1 shows how scalars, vectors, matrices and tensors can be represented graphically. A circle or node is used to represent a tensor, and dangling edges are used to indicate modes of the tensor.
Figure 1: Graphical TN representation of a (a) scalar, (b) vector, (c) matrix and (d) 4-way tensor.

Contraction between two tensors or a tensor with itself can be represented by connecting the appropriate dangling edges. This is illustrated in Figure 2. In mathematical notation, these contractions can be written elementwise as

(a) $\sum_i A_{ii} = \text{trace}(A) = c$,
(b) $\sum_j A_{ij} x_j = y_i$,
(c) $\sum_j A_{ij} B_{jk} = C_{ik}$,
(d) $\sum_{\ell mn} x_{\ell mn} A_{i\ell} B_{jm} C_{kn} = y_{ijk}$.

Figure 2: TN representation of (a) matrix trace, (b) matrix-vector multiplication, (c) matrix-matrix multiplication and (d) a 3-way Tucker decomposition.

To reduce computational cost when contracting a TN, it is optimal to contract two tensors at a time [26]. Determining the optimal contraction order is NP-hard, and developing heuristics for this problem is an active research area [26, 21]. There are several software packages that compute exact or approximate optimal contraction orders, e.g., NCON [25] for Matlab and the opt_einsum package for Python\(^1\).

\(^1\)Available at https://github.com/dgasmith/opt_einsum.
A TN matrix is a matrix which is represented by a TN. The dangling edges in such a network represent either rows or columns of the matrix. Figure 3 shows an example of a TN matrix with dangling edges pointing to the left representing rows and dangling edges pointing right representing columns. Suppose there are \( N_r \) and \( N_c \) dangling edges representing rows and columns, respectively, and let \( N = N_r + N_c \). Let \( \mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) denote the \( N \)-way TN representing the matrix, and let \( j \) be a length-\( N \) vector with the first \( N_r \) entries enumerating the modes of \( \mathcal{A} \) corresponding to rows and the remaining entries enumerating those modes that correspond to columns. The matrix \( \mathcal{A} \) represented by \( \mathcal{A} \) is then given elementwise by

\[
\mathcal{A}(i_1 \cdots i_{j(1)} \cdots i_{j(N_r)} \cdots i_{j(N_r+1)} \cdots i_{j(N)}) = \mathcal{A}_{i_1 \cdots i_N}
\]

and is of size \( \prod_{n=1}^{N_r} I_j(n) \times \prod_{m=N_r+1}^{N} I_j(m) \). For example, if the mode dimensions corresponding to the dangling edges in Figure 3 is 100, then the matrix represented by the TN is of size \( 10^6 \times 100 \).

Graphical TN notation does not specify the order in which the modes corresponding to dangling edges should appear when the tensor is represented as a multidimensional array. A similar ambiguity exists for the TN matrices where it is not clear how the row and column modes should be permuted. In both cases, any ordering can be used as long the choice is consistent in all computations.

### 3.3 Leverage Score Sampling

In the context of least squares, leverage scores indicate how important the rows of the design matrix are. By sampling according to this importance metric, it is possible to compute a good approximation (with high probability) to the least squares solution using a random subset of the equations of the full least squares problem. We provide the necessary preliminaries on leverage score sampling in this section and refer the reader to [17, 29] for further details.

**Definition 3.1.** The \( i \)th leverage score of a matrix \( \mathcal{A} \in \mathbb{R}^{I \times R} \) is defined as 

\[
\ell_i(\mathcal{A}) \triangleq e^{(i)^\top} \mathcal{A}^+ e^{(i)}
\]

for \( i \in [I] \), where \( e^{(i)} \) is the \( i \)th canonical basis vector of length \( I \).

**Definition 3.2.** Let \( \mathcal{A} \in \mathbb{R}^{I \times R} \) and let \( p \in \mathbb{R}^I \) be a vector with entries \( p(i) = \ell_i(\mathcal{A})/\text{rank}(\mathcal{A}) \). Then \( p \) is the leverage score distribution on row indices of \( \mathcal{A} \). Let \( f : [J] \rightarrow [I] \) be a random map such that each \( f(j) \) is independent and distributed according to \( p \). The random matrix \( S \in \mathbb{R}^{J \times I} \) defined elementwise via

\[
S(j, i) \triangleq \text{Ind}\{f(j) = i\}/\sqrt{Jp(f(j))}
\]

is an \( I \)-by-\( J \) matrix with leverage scores at the rows. The expected leverage score at row \( i \) is

\[
\mathbb{E}[\ell_i(\mathcal{A})] = \frac{1}{Jp(i)}
\]

for each row \( i \).
is the leverage score sampling matrix for $A$. In (2), $\text{Ind}\{A\}$ is the indicator function which is 1 if the random event $A$ occurs and 0 otherwise.

The following result is well-known and variants have appeared throughout the literature [6, 7, 8, 14].

**Theorem 3.3.** Let $A \in \mathbb{R}^{I \times R}$ be a matrix and suppose $S \in \mathbb{R}^{J \times I}$ is the leverage score sampling matrix for $A$. Moreover, let $\varepsilon, \delta \in (0, 1)$, and define $\text{OPT} \overset{\text{def}}{=} \min_X \|AX - Y\|_F$ and

$$
\tilde{X} \overset{\text{def}}{=} \arg \min_X \|SAX - SY\|_F.
$$

(3)

If $J \gtrsim R \max(\log(R/\delta), 1/(\varepsilon \delta))$, then with probability at least $1 - \delta$ it holds that $\|AX - Y\|_F \leq (1 + \varepsilon)\text{OPT}$.

## 4 Proposed Sampling Method

In this section, we propose an efficient method for sampling rows of a tall-and-skinny TN matrix $A \in \mathbb{R}^{I \times R}$ according to its leverage scores. The method is useful when $I$ is so large that costs and storage on the order $\Omega(I)$ are too expensive, and $R$ is small enough that costs and storage on the order $O(R^3)$ are affordable. In this setting it is infeasible to sample the rows of $A$ directly via the probability distribution in Definition 3.2 since doing so would require computing the pseudoinverse of $A$ which costs $O(IR^2)$. Moreover, once that pseudoinverse is computed, we would be required to compute, store and sample according to a length-$I$ vector of probabilities.

In view of Definition 3.1, note that

$$
\ell_i(A) = e^{(i)\top} AA^+ e^{(i)} = e^{(i)\top} A \Phi A^\top e^{(i)},
$$

(4)

where $\Phi \overset{\text{def}}{=} (A^\top A)^+$. This formulation is the basis for our sampling method. As we show in Section 4.1, the matrix $\Phi$ can be computed efficiently when $A$ is a TN matrix. Once $\Phi$ is computed, we can again leverage the TN structure of $A$ to draw samples from its leverage score distribution via the formula in (4) without forming a length-$I$ probability vector. We describe how this is done in Section 4.2.

### 4.1 Computing $\Phi$

When $A$ is a TN matrix, it is straightforward to represent the Gram matrix $A^\top A$ in such a format as well. This can be done by linking up the dangling column edges of $A^\top$ with the corresponding dangling row edges of $A$. For example, suppose $A$ is the TN matrix in Figure 3. The graphical representation of $A^\top$ is then a horizontal mirror image of Figure 3. The resulting Gram matrix is illustrated in Figure 4.

A dense representation of the Gram matrix can be computed by contracting its TN representation. For a wide range of TN matrices, this will be much more efficient that computing $A^\top A$ na"ively by first forming $A$ as a dense matrix and then carrying out a dense matrix-matrix multiplication between $A^\top$ and $A$. Once $A^\top A$ is computed, its pseudoinverse $\Phi = (A^\top A)^+$ is affordable to compute via standard methods.
4.2 Drawing Samples Efficiently

We now describe how to sample rows of $A$ according to its leverage score distribution without needing to form $A$. In order to keep the notation simple, we will assume without loss of generality that the modes of the TN $A$ describing the matrix $A$ are arranged so that

$$A(i_1 \cdots i_{N_r}, i_{N_r+1} \cdots i_N) = A(i_1 \cdots i_N).$$

Sampling a row of $A$ is therefore equivalent to sampling a multi-index $i_1 \cdots i_{N_r} \in \prod_{n=1}^{N_r} I_n$ with each $i_n \in [I_n]$.

In a nutshell, our sampling scheme sequentially samples each index $i_1, \ldots, i_{N_r}$ one after another by conditioning on the previously drawn indices. This allows us to sample from the leverage score distribution while at the same time avoid forming the full probability vector of length $\prod_{n=1}^{N_r} I_n$.

In the following we use $P(i_1)$ to denote the probability that the first index is $i_1$. We use $P(i_n | i_1, \ldots, i_{n-1})$ to denote the conditional probability that the $n$th index is $i_n$ given that the previous $n-1$ indices are $i_1, \ldots, i_{n-1}$. Let $\rho \triangleq \text{rank}(A)$. From Definition 3.2 and the formula in (4) we would like to sample row $i_1 \cdots i_{N_r}$ with probability

$$P(i_1) = \frac{1}{\rho} \sum_{i_2, i_3, \ldots, i_{N_r}} A(i_1 \cdots i_{N_r}, :) \Phi A(i_1 \cdots i_{N_r}, :)^\top.\ (5)$$

Therefore, we want the first index to be $i_1$ with probability

$$P(i_1) = \frac{1}{\rho} \sum_{i_2, i_3, \ldots, i_{N_r}} A(i_1 \cdots i_{N_r}, :) \Phi A(i_1 \cdots i_{N_r}, :)^\top.\ (5)$$

The matrix $A \Phi A^\top$ can easily be formulated as a TN matrix by linking up the TNs for $A$ and $A^\top$ with $\Phi$. Moreover, the summation in (5) in amounts to adding an additional $N_r - 1$ contractions to the TN representation of $A \Phi A^\top$. All in all, this results in a TN matrix $M^{(1)} \in \mathbb{R}^{I_1 \times I_1}$ with only two dangling edges. The desired probabilities then lie along the diagonal of this matrix: $P(i_1) = M^{(1)}_{i_1 i_1}$. The dense representation of $M^{(1)}$ can be computed efficiently by contracting the underlying network (it is sufficient to only compute diagonal entries which further reduces the cost).

An example of what this looks like when $A$ is the TN matrix in Figure 3 is illustrated in Figure 5. The first index $i_1$ is then drawn according to the probability distribution $(P(i_1))_{i_1=1}^{I_1}$.

The next step is to draw subsequent indices conditionally on the previously drawn indices. Suppose we have drawn $i_1, \ldots, i_{n-1}$. When sampling the $n$th index we compute the probabilities

$$P(i_n | i_1, \ldots, i_{n-1}) = \frac{P(i_1, \ldots, i_n)}{P(i_1, \ldots, i_{n-1})}.\ (6)$$
for each $i_n \in [I_n]$. Since $\mathbb{P}(i_1, \ldots, i_{n-1}) = \sum_{i_n} \mathbb{P}(i_1, \ldots, i_n)$ it is sufficient to compute the numerator in (6) for each $i_n \in [I_n]$ and then normalize them so that they add up to 1. Note that

$$\mathbb{P}(i_1, \ldots, i_n) = \frac{1}{\rho} \sum_{i_{n+1} \cdots i_{N_r}} A(i_1 \cdots i_N,:) \Phi A(i_1 \cdots i_N,:)^T.$$  

(7)

As earlier, this can be represented as a TN by adding $N_r - n$ additional contractions over the indices $i_{n+1}, \ldots, i_{N_r}$ to the TN representation of $A \Phi A^\top$. Moreover, since the indices $i_1, \ldots, i_{n-1}$ are fixed, this effectively eliminates the corresponding modes of the underlying tensor representation by reducing their dimensionality to 1. This results in a TN matrix $M^{(n)} \in \mathbb{R}^{I_n \times I_n}$ with only two dangling edges, with the desired joint probabilities laying along the diagonal: $\mathbb{P}(i_1, \ldots, i_n) = M^{(n)}$. Figure 6 illustrates what this looks like when we are computing the probability $\mathbb{P}(i_1, i_2)$ for the example from Figure 5 with $i_1$ fixed.

This sampling procedure is carried out until all indices $i_1, \ldots, i_{N_r}$ have been drawn, which corresponds to drawing a single row index $i_1 \cdots i_{N_r}$. The procedure is then repeated until the desired number of samples has been drawn. Note that the distribution $\langle \mathbb{P}(i_1) \rangle_{i_1=1}^N$ remains the same for all samples, so in order to speed up the overall sampling procedure it is a good idea to draw $i_1$ for all samples immediately.

The efficiency of the proposed sampling scheme relies on the possibility to efficiently contract the TNs corresponding to the Gram matrix computation $A^\top A$ and the probability distributions in (5) and (6). For a large class of tensor networks, this will be possible. We discuss the computational complexity of these contractions for the TN matrices that arise for CP and tensor ring decompositions in Section B.
5 Application to Tensor Decomposition

TNs can be used to express a very wide range of tensor decompositions, including the CP [10], Tucker [27], tensor train [23] and tensor ring decompositions [31].

Suppose $X \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is a data tensor that we want to decompose into some TN format consisting of tensors $A^{(1)}, \ldots, A^{(M)}$. We use $\text{TN}(A^{(1)}, \ldots, A^{(M)}) \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ to denote this TN. This decomposition problem can be formulated as the optimization problem

$$\arg \min_{A^{(1)}, \ldots, A^{(M)}} \|X - \text{TN}(A^{(1)}, \ldots, A^{(M)})\|_F,$$

where $\|\cdot\|_F$ is a straightforward generalization of the matrix Frobenius norm to tensors (see Section A). In general, this is a difficult non-convex optimization problem which is hard to solve. A popular approach to finding an approximate solution to (8) is to iteratively update one tensor $A^{(m)}$ at a time via alternating least squares (ALS). Minimizing the objective in (8) with respect to a single tensor at a time turns it into a linear least squares problem:

$$\arg \min_{A^{(m)}} \|X^{(m)} - \hat{A}^{\neq m} A^{(m)}\|_F,$$

where $X^{(m)}$ and $A^{(m)}$ are appropriate matricizations of $X$ and $A^{(m)}$, respectively, and $\hat{A}^{\neq m}$ is a TN matrix which depends on the tensors $A^{(1)}, \ldots, A^{(m-1)}, A^{(m+1)}, \ldots, A^{(M)}$. For example, for the CP and Tucker decompositions, $\hat{A}^{\neq m}$ takes the form of a Khatri–Rao and Kronecker product matrix, respectively [12]. Figure 7 shows examples of three tensor decompositions (top plots) and examples of what a TN matrix $\hat{A}^{\neq m}$ corresponding to each decomposition looks like (bottom plots). Algorithm 1 provides a high-level ALS algorithm for computing any TN decomposition.

Algorithm 1: ALS for TN decomposition

| Algorithm 1: ALS for TN decomposition |
|--------------------------------------|
| Input: $X \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ |
| Output: Factorization tensors $A^{(1)}, \ldots, A^{(M)}$ |
| 1 Initialize tensors $A^{(2)}, \ldots, A^{(M)}$ |
| 2 while termination criteria not met do |
| 3 for $m = 1, \ldots, M$ do |
| 4 $A^{(m)} = \arg \min_{A^{(m)}} \|X^{(m)} - \hat{A}^{\neq m} A^{(m)}\|_F$ |
| 5 return $A^{(1)}, \ldots, A^{(M)}$ |

ALS is the “workhorse approach” to tensor decomposition [12]. ALS-based methods for various decompositions are implemented in many leading tensor software packages, including Tensor Toolbox [2] and Tensorlab [28] for Matlab and TensorLy [13] for Python. ALS typically works well, especially when the data tensor $X$ is not too large. However, as previously discussed, each iteration of ALS depends on all entries of $X$. If $X$ has $N$ modes each of dimension $I$, then $X$ contains $I^N$ entries. Algorithms for tensor decomposition therefore inherit this exponential dependence on $N$. In the case of ALS, each iteration computed via (9) requires access to all entries of $X$. Moreover, for every iteration, it requires forming the matrix $\hat{A}^{\neq m}$ and then solving a linear system involving this matrix.

As discussed in Section 2, several previous works have sought to address the curse of dimensionality in ALS algorithms by sampling the least squares problem in (9). These works develop methods...
Figure 7: The top plots show examples of tensor decompositions: (a) CP decomposition, (b) tensor ring decomposition, and (c) a TN decomposition discovered in [15] via an evolutionary search procedure. The bottom plots (d)–(f) show examples of TN matrices that arise when updating the tensor marked in gray in the corresponding top plot. A tensor $A$ is said to be diagonal if only the elements $A_{ii...i}$ are nonzero.

for specific tensor decompositions. By contrast, our sampling framework described in Section 4 can be used to develop a sampling-based decomposition scheme for any TN decomposition. This is done by independently sampling each least squares problem on line 4 in Algorithm 1 according to the leverage scores of the TN matrix $A_{\#m}$. The sampling is done by first determining the indices to sample by applying our techniques in Section 4. Letting $S$ represent the sampling operation, the next step is to compute $S X^{(m)}$ and $S A_{\#m}$. The final step is to solve the sampled least squares problem

$$A^{(m)} = \arg \min_A \|SX^{(m)} - SA_{\#m} A\|_F.$$  

In order to make the discussion in Sections 4 and 5 a bit more concrete, we now apply these ideas to the CP decomposition.

**Example 5.1.** Suppose $X \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an $N$-way tensor and that we want to compute a rank-$R$ CP decomposition of it. Figure 7 (a) shows what the CP decomposition looks like when $N = 4$. 

10
Mathematically, for an arbitrary $N$ it takes the form
\[
X_{i_1 \cdots i_N} \approx \sum_{r=1}^{R} \lambda_r A_{i_1 \cdots i_N}^{(1)} \cdots A_{i_1 \cdots i_N}^{(N)},
\] (11)
where $\lambda_1, \ldots, \lambda_R$ are the elements in the diagonal tensor in the middle of Figure 7 (a), and the other tensors now have two modes and are therefore denoted by $A^{(m)} \in \mathbb{R}^{I_m \times R}$ and referred to as factor matrices. The values of the diagonal elements can be incorporated into the factor matrices, and we therefore assume that they are all equal to 1 without loss of generality. The design matrices now take the form
\[
A^{\neq m} = A^{(N)} \odot \cdots \odot A^{(m+1)} \odot A^{(m-1)} \odot \cdots \odot A^{(1)}.
\]
The corresponding unfoldings $X^{(m)}$ are defined elementwise via
\[
X^{(m)}(i_1 \cdots i_{m-1} \cdot i_{m+1} \cdots i_N, i_m) = X_{i_1 \cdots i_N}.
\]
The TN corresponding to the Gram matrix $A^{\neq m \top} A^{\neq m}$ can be efficiently contracted via the well-known formula
\[
A^{\neq m \top} A^{\neq m} = \bigotimes_{j \neq m} A^{(j) \top} A^{(j)},
\] (12)
where $\odot$ denotes elementwise product. Drawing a row index $i \in [\prod_{j \neq m} I_j]$ for $A^{\neq m}$ corresponds to drawing a multi-index $i_1 \cdots i_{m-1} i_{m+1} \cdots i_N$ with each $i_j \in [I_j]$. It is straightforward to show that the appropriate sampling probabilities in (5) and (7) are now given by
\[
P((i_j)_{j \leq n, j \neq m} = \frac{1}{R} \sum_{r_k} \Phi_{r_k} \cdot \left( \prod_{j \leq n, j \neq m} A^{(j)}_{i_j r} A^{(j)}_{i_j k} \right) \left( \prod_{j > n, j \neq m} (A^{(j) \top} A^{(j)})_{r_k} \right).
\]
This formula appears in Lemma 10 of [18], but in that formula $\Phi$ is an approximation of $(A^{\neq m \top} A^{\neq m})^+$. This also explains why a different normalization constant is used in [18]. The proof, however, remains the same.

### 5.1 Computational Complexity Examples

In this section we give the computational complexity of our proposed approach when used for CP and tensor ring decomposition. This allows us to compare our work with [18] which also considers these two decompositions. Tables 1 and 2 present the computational complexity of various methods for CP and tensor ring decomposition, respectively, when doing a rank-$R$ decomposition of an $N$-way cubic tensor of size $X \in \mathbb{R}^{I_1 \times \cdots \times I}$. For a precise definition of the rank $R$ for each decomposition, see (11) and (14). Our approaches for CP and tensor ring decomposition are called TNS-CP and TNS-TR, respectively, where TNS stands for Tensor Network Sampling. The complexities for our methods are computed in Section B. The complexities for the competing methods are taken from Tables 1 and 2 in [18]. For the randomized methods, the complexities reflect the parameter choices required for theoretical performance guarantees; see the respective cited works for further details.

As mentioned in Section 2, the primary contribution of [18] was to reduce the dependence on $N$ in the per-iteration cost from exponential to polynomial in ALS for CP and tensor ring decomposition. Our methods further improve on that cost by reducing the dependence on $R$. The improvement is due to the avoidance of the complicated recursive sketching step used in [18]. Note that all methods other than ours and those by [18] have an exponential dependence on $N$. 

11
Table 1: Leading order computational cost (ignoring log factors) for various CP decomposition methods. \#it is the number of ALS iterations. $R$ denotes the CP rank. Note that $\tilde{\varepsilon}$ is used as the accuracy parameter instead of $\varepsilon$ for SPALS. The reason is that SPALS has additive-error guarantees which are weaker than the relative-error guarantees of the other randomized methods; see Section B.3 for a discussion.

| Method                  | Complexity                                      |
|-------------------------|-------------------------------------------------|
| CP-ALS [12]             | \#it · $N(N+I)N^{-1}R$                          |
| SPALS [3]               | $I^N + \#it · N(N+I)R^{N+1}/\tilde{\varepsilon}^2$ |
| CP-ARLS-LEV [14]        | \#it · $N(R+I)R^N/(\varepsilon\delta)$        |
| CP-ALS-ES [18]          | \#it · $N^2R^3(R + NI/\varepsilon)/\delta$    |
| TNS-CP (our)            | \#it · $N^3IR^3/(\varepsilon\delta)$          |

Table 2: Leading order computational cost (ignoring log factors) for various tensor ring decomposition methods. \#it is the number of ALS iterations. The tensor ring rank is $(R, \ldots, R)$.

| Method                  | Complexity                                      |
|-------------------------|-------------------------------------------------|
| TR-ALS [31]             | \#it · $NI^N R^2$                               |
| rTR-ALS [30]            | $NI^N K + \#it · NK^N R^2$                      |
| TR-SVD [31]             | $I^{N+1} + I^N R^3$                             |
| TR-SVD-Rand [1]         | $I^N R^2$                                      |
| TR-ALS-Sampled [19]     | \#it · $NI^R R^2/(\varepsilon\delta)$         |
| TR-ALS-ES [18]          | \#it · $N^3IR^8(R + I/\varepsilon)/\delta$     |
| TNS-TR (our)            | \#it · $N^3IR^8/(\varepsilon\delta)$          |

6 Experiments

We run the same feature extraction and classification experiment as in [18]. The experiment considers 7200 images from the COIL-100 dataset [22]. These images depict 100 different objects (e.g., a toy car, an onion, a tomato), each photographed from 72 different angles. Each image is 128 by 128 pixels and has three color channels. We arrange these images into a tensor $\mathbf{X}$ of size $7200 \times 128 \times 128 \times 3$ and then decompose it using either a rank-25 CP decomposition or a rank-$(5, 5, 5, 5)$ tensor ring decomposition. For the CP decomposition, we treat the rows of the factor matrix $\mathbf{A}^{(1)} \in \mathbb{R}^{7200 \times 25}$ as feature vectors for each image. We treat 90% of the images as labeled and use them to classify the images in the remaining 10% using a $k$-nearest neighbor algorithm with $k = 1$ and 10-fold cross validation. The tensor ring decomposition is used in the same way, but instead of the CP factor matrix we now use the core tensor $\mathbf{A}^{(1)} \in \mathbb{R}^{5 \times 7200 \times 5}$ corresponding to the first mode and reshape the slices ($\mathbf{A}^{(1)}_i$), into length-25 feature vectors. Further details on the experiment setup are given in Section C.

Table 3 shows the run time, relative decomposition error (Err.) and classification accuracy (Acc.) for a number of different CP and tensor ring decomposition methods. The algorithms for our TNS-CP and TNS-TR were implemented\(^2\) by modifying the codes for CP-ALS-ES and TR-ALS-ES by [18] appropriately.

\(^2\)Our code is available at https://github.com/OsmanMalik/TNS.
Table 3: Run time, decomposition error and classification accuracy in the feature extraction experiment.

| Method               | Time (s) | Err. | Acc. (%) |
|----------------------|----------|------|----------|
| CP-ALS (Ten. Toolbox)| 48.0     | 0.31 | 99.2     |
| CPD-ALS (Tensorlab)  | 71.9     | 0.31 | 99.0     |
| CPD-MINF (Tensorlab) | 108.6    | 0.33 | 99.7     |
| CPD-NLS (Tensorlab)  | 112.1    | 0.31 | 92.6     |
| CP-ARLS-LEV          | 28.6     | 0.32 | 97.7     |
| CP-ALS-ES            | 29.0     | 0.32 | 98.3     |
| TNS-CP (our)         | 23.5     | 0.32 | 98.3     |
| TR-ALS               | 10278.4  | 0.31 | 99.5     |
| TR-ALS-Sampled       | 5.3      | 0.33 | 98.0     |
| TR-ALS-ES            | 29.7     | 0.33 | 97.2     |
| TNS-TR (our)         | 14.5     | 0.33 | 97.3     |

All methods achieve roughly the same decomposition error, with the randomized methods typically having a slightly higher error. All methods except CPD-NLS yield similar classification accuracy. It is clear that our methods are faster than CP-ALS-ES and TR-ALS-ES from [18], corroborating the improved complexity reported in Tables 1 and 2.

7 Conclusion

We have presented a sampling-based ALS approach for decomposing a tensor into any TN format. The generality of the framework is notable—all previous works on randomized tensor decomposition we are aware of develop methods for specific decompositions. Additionally, unlike most previous sampling-based ALS algorithms in the literature, our framework makes it possible to do the sampling according to the exact leverage scores. Our approach simplifies and generalizes the scheme developed in [18]. Both complexity analyses and numerical experiments confirm the improved run time we are able to achieve compared to [18] when we use our framework for CP and tensor ring decomposition.

There are many exciting avenues for future research. One direction is to adapt and implement our method for use in a high-performance distributed memory setting. Another interesting direction is designing an exact leverage score sampling scheme for general TN matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$ that avoid the $\Omega(R^3)$ cost that we incur when computing the pseudoinverse $\Phi = (\mathbf{A}^\top \mathbf{A})^+$. Such methods are known for certain matrices with Kronecker product structure [20], but for general TN matrices such results are not known.

Acknowledgments

OAM was supported by the LDRD Program of Lawrence Berkeley National Laboratory under U.S. DOE Contract No. DE-AC02-05CH11231. VB was supported in part by the U.S. DOE, Office of Science, Office of Advanced Scientific Computing Research, DOE Computational Science Graduate Fellowship under Award No. DE-SC0022158. RM was supported by NSF Grant No. 2004235.
References

[1] S. Ahmadi-Asl, A. Cichocki, A. H. Phan, M. G. Asante-Mensah, F. Mousavi, I. Oseledets, and T. Tanaka. Randomized algorithms for fast computation of low-rank tensor ring model. *Machine Learning: Science and Technology*, 2020.

[2] B. W. Bader, T. G. Kolda, and others. MATLAB Tensor Toolbox, Version 3.2.1. Available online at https://www.tensortoolbox.org, 2021.

[3] D. Cheng, R. Peng, Y. Liu, and I. Perros. SPALS: Fast alternating least squares via implicit leverage scores sampling. In *Advances In Neural Information Processing Systems*, pages 721–729, 2016.

[4] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends in Machine Learning*, 9(4-5):249–429, 2016.

[5] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, and D. P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. *Foundations and Trends in Machine Learning*, 9(6):431–673, 2017.

[6] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Sampling algorithms for $\ell_2$ regression and applications. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 1127–1136, 2006.

[7] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881, 2008.

[8] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós. Faster least squares approximation. *Numerische Mathematik*, 117(2):219–249, 2011.

[9] M. Fahrbach, T. Fu, and M. Ghadiri. Subquadratic Kronecker regression with applications to tensor decomposition. *Preprint arXiv:2209.04876*, 2022.

[10] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[11] Y. Ji, Q. Wang, X. Li, and J. Liu. A survey on tensor techniques and applications in machine learning. *IEEE Access*, 7:162950–162990, 2019.

[12] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, Aug. 2009.

[13] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic. TensorLy: Tensor learning in Python. *Journal of Machine Learning Research*, 20(26):1–6, 2019.

[14] B. W. Larsen and T. G. Kolda. Practical leverage-based sampling for low-rank tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 43(3):1488–1517, 2022.

[15] C. Li and Z. Sun. Evolutionary topology search for tensor network decomposition. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 5947–5957. PMLR, 2020.
[16] L. Ma and E. Solomonik. Cost-efficient Gaussian tensor network embeddings for tensor-structured inputs. Preprint arXiv:2205.13163, 2022.

[17] M. W. Mahoney. Randomized algorithms for matrices and data. Foundations and Trends in Machine Learning, 3(2):123–224, 2011.

[18] O. A. Malik. More efficient sampling for tensor decomposition with worst-case guarantees. In Proceedings of the 39th International Conference on Machine Learning, volume 162, pages 14887–14917. PMLR, 2022.

[19] O. A. Malik and S. Becker. A sampling-based method for tensor ring decomposition. In Proceedings of the 38th International Conference on Machine Learning, volume 139, pages 7400–7411. PMLR, 2021.

[20] O. A. Malik, Y. Xu, N. Cheng, S. Becker, A. Doostan, and A. Narayan. Fast algorithms for monotone lower subsets of Kronecker least squares problems. Preprint arXiv:2209.05662, 2022.

[21] E. Meirom, H. Maron, S. Mannor, and G. Chechik. Optimizing tensor network contraction using reinforcement learning. In Proceedings of the 39th International Conference on Machine Learning, volume 162, pages 15278–15292. PMLR, 2022.

[22] S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (COIL-100). Technical Report CUCS-006-96, Columbia University, 1996.

[23] I. V. Oseledets. Tensor-train decomposition. SIAM Journal on Scientific Computing, 33(5):2295–2317, 2011.

[24] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. ACM Transactions on Intelligent Systems and Technology (TIST), 8(2):1–44, 2016.

[25] R. N. Pfeifer, G. Evenbly, S. Singh, and G. Vidal. NCON: A tensor network contractor for MATLAB. Preprint arXiv:1402.0939, 2014.

[26] R. N. C. Pfeifer, J. Haegeman, and F. Verstraete. Faster identification of optimal contraction sequences for tensor networks. Phys. Rev. E, 90:033315, 2014.

[27] L. R. Tucker. Implications of factor analysis of three-way matrices for measurement of change. Problems in measuring change, 15(122-137):3, 1963.

[28] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer. Tensorlab 3.0. Available online at https://www.tensorlab.net, Mar. 2016.

[29] D. P. Woodruff. Sketching as a tool for numerical linear algebra. Foundations and Trends in Theoretical Computer Science, 10(1-2):1–157, 2014.

[30] L. Yuan, C. Li, J. Cao, and Q. Zhao. Randomized tensor ring decomposition and its application to large-scale data reconstruction. In IEEE International Conference on Acoustics, Speech and Signal Processing, pages 2127–2131. IEEE, 2019.

[31] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki. Tensor ring decomposition. Preprint arXiv:1606.05535, 2016.
A Additional Notation

Let $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{K \times L}$ be two matrices. Their Kronecker product is denoted by $A \otimes B \in \mathbb{R}^{IK \times JL}$ and defined as

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1J}B \\ A_{21}B & A_{22}B & \cdots & A_{2J}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{I1}B & A_{I2}B & \cdots & A_{IJ}B \end{bmatrix}.$$ 

Suppose now that $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{K \times J}$, i.e., $A$ and $B$ have the same number of columns. The Khatri–Rao product of $A$ and $B$ is denoted by $A \odot B \in \mathbb{R}^{IK \times J}$ and defined as

$$A \odot B = [A_1 \otimes B_1 \quad A_2 \otimes B_2 \quad \cdots \quad A_J \otimes B_J].$$

The Khatri–Rao product is sometimes called the columnwise Kronecker product for obvious reasons.

Now, suppose that $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{I \times J}$ are of the same size. The Hadamard product, or elementwise product, of $A$ and $B$ is denoted by $A \diamond B \in \mathbb{R}^{I \times J}$ and defined elementwise in the obvious way:

$$(A \diamond B)_{ij} = A_{ij}B_{ij}.$$ 

The Frobenius norm is a standard matrix norm that can be easily extended to tensors. For a tensor $X \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ it is defined as

$$\|X\|_F = \sqrt{\sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} X_{i_1,\ldots,i_N}^2}.$$ 

B Detailed Complexity Analysis

Here we provide further details on the complexity analysis in Section 5.1.

B.1 CP Decomposition

Suppose we are decomposing an $N$-way cubic tensor of size $X \in \mathbb{R}^{I_1 \times \cdots \times I}$ using a rank-$R$ CP decomposition. As mentioned in Example 5.1, the entries of the diagonal tensor (see Figure 7 (a)) can be treated as ones without loss of generality. The update in (10) therefore involves updating one of the CP factor matrices $A^{(m)}$, $m \in [N]$.

**Computing $\Phi$** Computing the Gram matrix $A^\dagger A^\dagger$ via (12) costs $O(NIR^2)$. When doing this computation, we store all the Gram matrices $A^{(j)^\top} A^{(j)}$ for no additional computational cost for later use. Computing the pseudoinverse $\Phi = (A^\dagger A^\dagger)^+$ costs an additional $O(R^3)$. The total cost of computing $\Phi$ is therefore $O(NIR^2 + R^3)$.

**Drawing $J$ Samples** The cost is the same as the sampling cost for CP-ALS-ES in [18], but without the one-time costs since $\Phi$ and the Gram matrices $A^{(j)^\top} A^{(j)}$, $j \in [N] \setminus \{m\}$, have already been computed. The cost is therefore $O(JR^2N^2I)$; see [18, Sec. C.1] for details.
Solving Sampled Least Squares Problem  The cost of forming the sampled matrices $SA \neq m$ and $SX \neq m$, and solving the associated linear system (10) via direct methods, is the same as for CP-ALS-ES in [18], namely $O(JR(N + R + I))$; see [18, Sec. C.1] for details.

Adding It All Up  The costs above are for solving one least squares problem of the form (10). Each ALS iteration requires solving $N$ such systems. Consequently, the per-iteration cost for our proposed method is

$$O(JR^2N^3I),$$

(13)

where we have used the fact that the sample size $J$ must satisfy $J \geq R$ (see Theorem 3.3) to simplify the complexity expression. Since $A \neq m$ has $R$ columns, the bound on $J$ from Theorem 3.3 is $J \gg R\max(\log(R/\delta), 1/(\varepsilon\delta))$. Inserting this into (13) gives the per-iteration cost

$$O(R^3N^3I\log(R/\delta)/(\varepsilon\delta)) = \tilde{O}(R^3N^3I/(\varepsilon\delta))$$

which is what we report in Table 1.

B.2 Tensor Ring Decomposition

Suppose that we are decomposing an $N$-way cubic tensor of size $X \in \mathbb{R}^{I \times \cdots \times I}$ using a tensor ring decomposition with all ranks set to $R$. The decomposition takes the form

$$X_{i_1\cdots i_N} \approx \sum_{r_1, \ldots, r_N} A_{r_N, i_1, r_1}^{(1)} A_{r_1, i_2, r_2}^{(2)} \cdots A_{r_N-1, i_N, r_N}^{(N)},$$

(14)

where each $A^{(m)}$ is now a 3-way tensor of size $R \times I \times R$, and where each summation index $r_n$ goes from 1 to $R$. See [31] for further details on the tensor ring decomposition.

Computing $\Phi$  For the tensor ring decomposition, the TN representation of $A \neq m$ is shown in Figure 7 for the case $N = 4$. Figure 8 illustrates how we contract the TN corresponding to the Gram matrix $A \neq m \top A \neq m$ for an arbitrary $N$. Each edge we eliminate in subplot (a) costs $O(IR^4)$. Since there are $N - 1$ pairs to contract, this brings the total cost of the step in (a) to $O(NIR^4)$. Each step in the sequence of contractions in (b) costs $O(R^6)$. Since $N - 2$ such contractions are required, the total cost of step (b) is $O(NR^6)$. Once the matrix $A \neq m \top A \neq m$ in (c) is computed, it costs $O(R^6)$ to compute its pseudoinverse, i.e., less than the cost of the step in (b). Adding up the costs for the steps in (a) and (b) and the cost of the pseudoinverse brings the total cost for computing $\Phi$ to $O(N(R^6 + IR^4)).$

Drawing $J$ Samples  The cost is the same as the sampling cost for TR-ALS-ES in [18], but without the one-time costs in that paper\textsuperscript{3}. The cost is therefore $O(JN^2IR^6)$; see [18, Sec. C.2] for details.

Solving Sampled Least Squares Problem  The cost of forming the sampled matrices $SA \neq m$ and $SX^{(m)}$, and solving the associated linear system (10) via direct methods, is the same as for TR-ALS-ES in [18], namely $O(J(NR^3 + R^4 + IR^2))$.

\textsuperscript{3}The Gram matrices $G^{(j)} \top G^{(j)}$ in [18] are precisely what we compute when we do the contractions in Figure 8 (a). These can be saved during that step, which is why the one-time costs of computing those Gram matrices is already accounted for from the computation of $\Phi$.  

17
Adding It All Up The costs above are for solving one least squares problem of the form (10). Each ALS iteration requires solving \( N \) such systems. Consequently, the per-iteration cost for our proposed method is

\[
O(JN^3IR^6). \tag{15}
\]

Since \( A^{\otimes m} \) has \( R^2 \) columns, the bound on \( J \) in Theorem 3.3 is \( J \geq R^2 \max(\log(R^2/\delta), 1/(\varepsilon\delta)) \). Inserting this into (15) gives the per-iteration cost

\[
O(N^3IR^8 \log(R^2/\delta)/(\varepsilon\delta)) = \tilde{O}(N^3IR^8/(\varepsilon\delta)),
\]

which is what we report in Table 2.

B.3 Some Remarks on Tables 1 and 2

All methods in Table 1 except CP-ALS are randomized. The theoretical guarantees for CP-ARLS-LEV, CP-ALS-ES and TNS-CP are all of the same type: They provide relative-error guarantees of the form in Theorem 3.3 for each least squares solve on line 4 in Algorithm 1 when it is adapted to CP decomposition. For these three methods, the meaning of the variables \( \varepsilon \) and \( \delta \) are therefore identical. SPALS, by contrast, has weaker additive-error guarantees. Expressed using the same notation as in Theorem 3.3, their guarantees take the form

\[
\|A\hat{X} - Y\|^2_F \leq \text{OPT}^2 + \tilde{\varepsilon}\|Y\|^2_F, \tag{16}
\]

where the statement in (16) holds with probability at least \( 1 - \delta \) when the number of samples is large enough. The constant \( \delta \) has the same meaning as for the other three randomized methods.
but $\tilde{\varepsilon}$ has a different meaning than $\varepsilon$: The latter is a relative-error accuracy parameter while the former is an additive-error accuracy parameter.

For the three methods TR-ALS-Sampled, TR-ALS-ES and TNS-TR in Table 2, the parameters $\delta$ and $\varepsilon$ are all used in the sense of Theorem 3.3.

The number of iterations, denoted by $\#it$, required to reach a certain accuracy or fulfill certain termination criteria may differ between the various methods. For example, the deterministic ALS methods—by virtue of being exact and non-random—may require fewer iterations to reach a certain accuracy. Similarly differences may exist between the various randomized ALS methods, and between the different decomposition types.

C Further Details on Experiment

C.1 Competing Methods

We provide some further details on the methods we compare against below.

- **CP-ALS.** This is an implementation of the standard ALS method for CP decomposition without any randomization. It comes with the Tensor Toolbox for Matlab [2] which can be downloaded at https://www.tensortoolbox.org.

- **CPD-ALS, -MINF, -NLS.** These methods use three different minimization approaches to compute the CP decomposition. They are all available in the Tensorlab package for Matlab [28] which can be downloaded at https://www.tensorlab.net/.

- **CP-ARLS-LEV.** This the method proposed in [14]. We use the implementation by Malik [18] which is available at https://github.com/OsmanMalik/TD-ALS-ES.

- **CP-ALS-ES.** This is the method for CP decomposition proposed in [18]. We use the code available at https://github.com/OsmanMalik/TD-ALS-ES.

- **TR-ALS.** This is a standard ALS methods for tensor ring decomposition, which was proposed in [31]. We use the implementation by Malik and Becker [19] which is available at https://github.com/OsmanMalik/tr-als-sampled.

- **TR-ALS-Sampled.** This is the sampling-based approach for tensor ring decomposition proposed in [19]. We use the code available at https://github.com/OsmanMalik/tr-als-sampled.

- **TR-ALS-ES.** This is the method for tensor ring decomposition proposed in [18]. We use the code available at https://github.com/OsmanMalik/TD-ALS-ES.

C.2 Parameter Choices

We sample $J = 2000$ rows in all ALS subproblems for the sampling-based CP decomposition methods (CP-ARLS-LEV, CP-ALS-ES and TNS-CP). For the sampling-based tensor ring methods (TR-ALS-Sampled, TR-ALS-ES and TNS-TR) we use $J = 1000$ samples in the ALS subproblems. Both CP-ALS-ES and TR-ALS-ES require an intermediate embedding via a recursive sketching procedure. For both methods, we use an embedding dimension of 10000 for these intermediate steps, as suggested in [18].
C.3 Hardware/Software and Dataset

The experiments are run in Matlab R2021b on a laptop computer with an Intel Core i7-1185G7 CPU and 32 GB of RAM.

The COIL-100 dataset [22] is available for download at https://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php.