

# PARALLEL RANDOMIZED TUCKER DECOMPOSITION ALGORITHMS\*

RACHEL MINSTER<sup>†</sup>, ZITONG LI<sup>‡</sup>, AND GREY BALLARD<sup>†</sup>

**Abstract.** The Tucker tensor decomposition is a natural extension of the singular value decomposition (SVD) to multiway data. We propose to accelerate Tucker tensor decomposition algorithms by using randomization and parallelization. We present two algorithms that scale to large data and many processors, significantly reduce both computation and communication cost compared to previous deterministic and randomized approaches, and obtain nearly the same approximation errors. The key idea in our algorithms is to perform randomized sketches with Kronecker-structured random matrices, which reduces computation compared to unstructured matrices and can be implemented using a fundamental tensor computational kernel. We provide probabilistic error analysis of our algorithms and implement a new parallel algorithm for the structured randomized sketch. Our experimental results demonstrate that our combination of randomization and parallelization achieves accurate Tucker decompositions much faster than alternative approaches. We observe up to a 16× speedup over the fastest deterministic parallel implementation on 3D simulation data.

**1. Introduction.** Tucker decompositions are low-rank tensor approximations capable of approximating multidimensional data with large compression rates while maintaining high accuracy. Large scale multidimensional data arises from many applications such as simulations of partial differential equations, data mining, facial recognition, and imaging. Processing these data requires computationally efficient methods. Randomized algorithms have been used to efficiently compute Tucker decompositions in works such as [1, 4, 9, 11, 25, 27, 30, 33], but the growing size of data is outpacing even randomized algorithms. Scaling these methods to handle large data calls for efficient parallelization. Many high-performance implementations of deterministic algorithms have been developed for Tucker decompositions [3, 7, 21, 12, 24]. We develop both sequential and parallel randomized algorithms that efficiently compute Tucker decompositions of large-scale multidimensional data by reducing both computation and communication compared to previous work.

As we review in § 2, there are two dominant computational kernels to computing a Tucker decomposition: computing matrix singular value decompositions (SVD) and computing tensor-times-matrix (TTM) products. For the deterministic algorithms HOSVD [14] (Alg. 2.1) and STHOSVD [29] (Alg. 2.2), computing the SVD is the typical bottleneck, and various methods trade off accuracy for reduced computational complexity. Our goal is to reduce the complexity of the SVD computation via randomization and remove it as the dominant cost without sacrificing too much accuracy. Existing randomized Tucker approaches, discussed in § 3, apply low-rank matrix approximation algorithms in place of matrix SVDs. These matrix algorithms include randomized range finder [17] (RRF, see Alg. 2.3), which computes part of the low-rank approximation and involves a slight overestimate of the target rank, or randomized SVD [17] (RandSVD, see Alg. 2.4), which involves a second pass over the data to obtain the final approximation with the exact target rank.

We propose two randomized algorithms in § 4, one based on HOSVD and one based on STHOSVD, which have comparable accuracy and running time. In our algorithms, we use the RRF approach with Kronecker-structured random matrices, which reduces the computational complexity of the sketch compared to previous ran-

---

\*This work is supported by the National Science Foundation under Grant No. CCF-1942892.

<sup>†</sup>Wake Forest University, Winston-Salem, NC ([minsterr@wfu.edu](mailto:minsterr@wfu.edu), [ballard@wfu.edu](mailto:ballard@wfu.edu))

<sup>‡</sup>University of California Irvine, Irvine, CA ([zitongl5@uci.edu](mailto:zitongl5@uci.edu))

domized approaches. As a significant added practical benefit, the Kronecker structure reduces the amount of random number generation compared to unstructured random matrices such as Gaussian. Furthermore, we propose a deterministic truncation of the resulting core (with overestimated ranks) in order to achieve the exact target ranks, obtaining the same effect as RandSVD-based approaches at much lower cost. We show that our HOSVD-based algorithm can be as computationally efficient as our STHOSVD-based algorithm by employing a dimension tree optimization to avoid recomputation across sketches using memoization.

To accompany our algorithms, we develop probabilistic error guarantees in § 5 for a randomized matrix algorithm using a Kronecker product of random matrices. We use the matrix results to obtain theoretical guarantees for our Tucker algorithms. Our bounds differ from previous results by accounting for the Kronecker structure and rank truncation in our algorithms and by reducing the probability of failure and amplification factors.

In § 6, we describe the parallelization of our proposed algorithms for distributed memory using the TuckerMPI library [3], allowing us to scale the algorithms to large datasets that cannot be processed on a single server. While previous work has combined randomization and parallelization, our implementation is the first to parallelize the randomized sketch, which significantly reduces the computational cost. Moreover, in exploiting the Kronecker structure of our sketch, we implement a new parallel algorithm that communicates less data than the algorithm used by TuckerMPI and in fact minimizes interprocessor communication for the computation [2].

Our experimental results are presented in § 7. We validate the error guarantees of § 5 and show empirically that our structured random matrices are just as accurate as standard Gaussian random matrices. Using synthetic data as well as two large simulation datasets, we demonstrate that our parallel randomized algorithms given in § 6 scale well to thousands of cores and outperform alternative deterministic and randomized algorithms, achieving speedups of up to  $16\times$  over the state-of-the-art implementation of the best deterministic algorithm.

**2. Background.** We first review the relevant background on tensors and randomized algorithms for matrices. For more details on tensors, see [22], and for more details on randomized algorithms, see [17].

**2.1. Tensor notation and operations.** A tensor  $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$  is a  $d$ -way array. We can unfold a  $d$ -mode tensor along each of its modes, or dimensions; the mode- $j$  unfolding, denoted  $\mathbf{X}_{(j)}$ , is a matrix with columns formed as the mode- $j$  fibers of the tensor. Let  $\|\cdot\|$  denote the tensor norm, which generalizes the matrix Frobenius norm. Since the following products will be frequently used to describe the sizes and ranks of a tensor, we define the following notations:  $n^* = \prod_{k=1}^d n_k$ ,  $n_i^{\otimes} = \prod_{k=1}^{i-1} n_k$ ,  $n_i^{\otimes} = \prod_{k=i+1}^d n_k$ ,  $n_i^{\otimes} = \prod_{k \neq i} n_k$ .

One key operation for tensors is the tensor-times-matrix product, or TTM. A tensor  $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$  is multiplied along mode  $j$  by a matrix  $\mathbf{A} \in \mathbb{R}^{m \times n_j}$ , denoted by  $\mathbf{X} \times_j \mathbf{A}$ , to obtain a tensor  $\mathbf{Y} \in \mathbb{R}^{n_1 \times \dots \times n_{j-1} \times m \times n_{j+1} \times \dots \times n_d}$ . This product can also be expressed in terms of its mode- $j$  unfolding as  $\mathbf{Y}_{(j)} = \mathbf{A}\mathbf{X}_{(j)}$ . We can also multiply a tensor  $\mathbf{X} \in \mathbb{R}^{n_1 \times \dots \times n_d}$  by up to  $d$  matrices  $\mathbf{A}_j \in \mathbb{R}^{m_j \times n_j}$ ,  $j = 1, \dots, d$ , across distinct modes to obtain  $\mathbf{Y} = \mathbf{X} \times_1 \mathbf{A}_1 \times_2 \mathbf{A}_2 \times \dots \times_d \mathbf{A}_d \in \mathbb{R}^{m_1 \times \dots \times m_d}$ . We call this product a multi-TTM; it is also known as a multilinear multiplication. If unfolded along mode  $j$ , we have  $\mathbf{Y}_{(j)} = \mathbf{A}_j \mathbf{X}_{(j)} (\mathbf{A}_d \otimes \dots \otimes \mathbf{A}_{j+1} \otimes \mathbf{A}_{j-1} \otimes \dots \otimes \mathbf{A}_1)^\top$ , where  $\otimes$  is the matrix Kronecker product.

**2.2. Tucker Decomposition.** The Tucker decomposition of a given tensor  $\mathcal{X} \in \mathbb{R}^{n_1 \times \dots \times n_d}$  of multirank  $\mathbf{r} = (r_1, \dots, r_d)$ , where  $r_j = \text{rank}(\mathbf{X}_{(j)})$  for each  $j$ , represents  $\mathcal{X}$  as the product of a core tensor  $\mathcal{G} \in \mathbb{R}^{r_1 \times \dots \times r_d}$  and  $d$  factor matrices  $\mathbf{U}_j \in \mathbb{R}^{n_j \times r_j}$  such that  $\mathcal{X} = \mathcal{G} \times_1 \mathbf{U}_1 \times \dots \times_d \mathbf{U}_d$ . We can also obtain a low-rank approximation to  $\mathcal{X}$  in the Tucker form by taking the target rank  $(r_1, \dots, r_d)$ , or size of the core tensor  $\mathcal{G}$ , to be less than the ranks of the unfoldings in each mode.

*Higher-Order SVD (HOSVD) and Sequentially Truncated HOSVD.* Two algorithms that compute low-rank Tucker decompositions of tensors are the higher-order SVD (HOSVD) [14] and sequentially truncated HOSVD (STHOSVD) [29]. The HOSVD algorithm forms each factor matrix  $\mathbf{U}_j$  from the first  $r_j$  left singular vectors of the mode unfolding  $\mathbf{X}_{(j)}$ , and once all the factor matrices are computed, computes the core tensor as  $\mathcal{G} = \mathcal{X} \times_1 \mathbf{U}_1^\top \times \dots \times_d \mathbf{U}_d^\top$  (see Alg. 2.1).

---

**Algorithm 2.1** HOSVD [14]

---

```

1: function  $[\mathcal{G}, \{\mathbf{U}_j\}] = \text{HOSVD}(\mathcal{X}, \mathbf{r})$ 
2:   for  $j = 1 : d$  do
3:      $\mathbf{U}_j = \text{first } r_j \text{ left sing. vecs. of } \mathbf{X}_{(j)}$ 
4:   end for
5:    $\mathcal{G} = \mathcal{X} \times_1 \mathbf{U}_1^\top \times \dots \times_d \mathbf{U}_d^\top$ 
6: end function

```

---

**Algorithm 2.2** STHOSVD [29]

---

```

1: function  $[\mathcal{G}, \{\mathbf{U}_j\}] = \text{STHOSVD}(\mathcal{X}, \mathbf{r})$ 
2:    $\mathcal{G} = \mathcal{X}$ 
3:   for  $j = 1 : d$  do
4:      $\mathbf{U}_j = \text{first } r_j \text{ left singular vecs. of } \mathbf{G}_{(j)}$ 
5:      $\mathcal{G} = \mathcal{G} \times_j \mathbf{U}_j^\top$ 
6:   end for
7: end function

```

---

The STHOSVD algorithm is similar to HOSVD, but instead of handling all modes independently, it processes the modes in a predetermined sequence. After the first factor matrix is computed from the first  $r_j$  left singular vectors of  $\mathbf{X}_{(j)}$ , we truncate in that mode by computing a partially truncated core tensor via a TTM with the factor matrix,  $\mathcal{G} \times_j \mathbf{U}_j^\top$ . We then use the partially truncated core  $\mathcal{G}$  for the next mode instead of the full tensor  $\mathcal{X}$ , as shown in Alg. 2.2.

**2.3. Randomized Matrix Algorithms.** The randomized range finder algorithm, made popular by [17] and shown in Alg. 2.3, efficiently computes a low-rank representation of a matrix  $\mathbf{M} \in \mathbb{R}^{m \times n}$ . Given a target rank  $r$  and oversampling parameter  $p$ , we multiply  $\mathbf{M}$  by a random matrix  $\Omega \in \mathbb{R}^{n \times \ell}$  with  $\ell = r + p$  such that  $\ell < m$ , to form  $\mathbf{Y} \in \mathbb{R}^{m \times \ell}$ , a matrix made up of random linear combinations of the columns of  $\mathbf{M}$ . We then compute a thin QR decomposition of  $\mathbf{Y}$  to obtain a matrix  $\mathbf{Q} \in \mathbb{R}^{m \times \ell}$  whose range is a good estimate of the range of  $\mathbf{M}$ . Projecting  $\mathbf{M}$  onto the range of  $\mathbf{Q}$  gives us the low-rank approximation  $\mathbf{M} \approx \mathbf{Q}\mathbf{Q}^\top \mathbf{M}$ . We can choose any random distribution for random matrix  $\Omega$ . In this paper, we will consider both subsampled random Hadamard transform (SRHT) and standard Gaussian matrices.

Note that the resulting approximation from Alg. 2.3 is actually rank- $\ell$ . If we seek a rank- $r$  approximation, further truncation is necessary. One way of truncating is to take a thin SVD of  $\mathbf{Q}^\top \mathbf{M}$ , which is the process taken in the randomized SVD algorithm in [17], reproduced in Alg. 2.4. We will adapt this truncation method in the algorithms developed in later sections.

---

**Algorithm 2.3** Randomized Range Finder [17]

---

```

1: function  $\mathbf{Q} = \text{RANDRANGEFINDER}(\mathbf{M}, \Omega)$ 
2:    $\mathbf{Y} = \mathbf{M}\Omega$ 
3:   Compute thin QR  $\mathbf{Y} = \mathbf{QR}$ 
4: end function

```

---

**Algorithm 2.4** Randomized SVD [17]

---

```

1: function  $[\mathbf{U}, \Sigma, \mathbf{V}] = \text{RANDSVD}(\mathbf{M}, r, \Omega)$ 
2:    $\mathbf{Q} = \text{RandRangeFinder}(\mathbf{M}, \Omega)$ 
3:    $\mathbf{B} = \mathbf{Q}^\top \mathbf{M}$ 
4:   Compute thin SVD  $\mathbf{B} = \hat{\mathbf{U}}\Sigma\mathbf{V}^\top$ 
5:    $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}(:, 1:r)$ 
6:   Truncate  $\Sigma = (1:r, 1:r)$ ,  $\mathbf{V} = \mathbf{V}(:, 1:r)$ 
7: end function

```

---

**3. Related Work.** Our work builds on three different categories of previous work, namely randomized algorithms for Tucker decompositions, probabilistic analysis of randomized algorithms, and parallel algorithms for tensor computations.

*Randomized Algorithms.* There has been much previous work on randomized algorithms for Tucker decompositions; a good survey of this work can be found in [1]. The basic algorithms for randomized HOSVD and randomized STHOSVD are proposed in [33], while later work improves on the algorithms in various ways. One important distinction among randomized algorithms is the rank of the output approximation. In [30, 33], the approximation has rank  $\ell = r + p$  as the randomized range finder ([Alg. 2.3](#)) is used without additional truncation. Other algorithms, such as those presented in [9, 27] do not oversample at all, limiting the potential accuracy of their methods. In [11], the authors use the randomized range finder, but truncate by only taking the first  $r_j$  columns of each factor matrix. The randomized SVD algorithm ([Alg. 2.4](#)) can be applied instead to both oversample and more accurately obtain the desired rank- $\mathbf{r}$  approximation, which is done in [4, 25]. Our approach is most similar to the randomized SVD approach, but we apply it in a holistic manner, as discussed in [§ 4.1](#).

Another common improvement to the basic randomized algorithms comes from exploiting structure in the random matrices used to reduce storage and/or computational costs, as well as the number of random entries generated. Khatri-Rao products of random matrices are used in [9, 27], compact random matrices are employed in [4], and Kronecker products of random matrices are used in [10, 11]. Our work is most similar to [11] as we also employ Kronecker products, but our algorithms improve upon those in [11] by truncating to the desired target rank in a more accurate manner. Kronecker product structure has also been exploited in other tensor decompositions besides Tucker decompositions: in [5, 19], Kronecker products of random matrices are used to accelerate algorithms for CP decompositions; while in [13], Kronecker product structure was exploited in the context of the tensor-train decomposition. We also discuss how to implement our algorithms on distributed systems and provide improved probabilistic analysis.

*Error Analysis.* To accompany the discussed randomized algorithms, other work has developed probabilistic error analysis. Analysis for the standard version of randomized HOSVD is presented in [16, 25], and for the standard randomized STHOSVD algorithm in [9, 25] for Khatri-Rao products of Gaussian matrices and dense Gaussian random matrices, respectively. Previous error analysis has been done for a randomized STHOSVD algorithm employing Kronecker products of subsampled randomized Fourier transform (SRFT) matrices in [11], but we make several improvements on this work. Our error bound, for our algorithms with Kronecker products of the real-valued equivalent of SRFTs, i.e., subsampled randomized Hadamard transform (SRHT) matrices, has an improved error constant and a smaller probability of failure.

*Parallel Algorithms.* Our parallel algorithms and implementation, described in [§ 6](#), are built upon the foundation of TuckerMPI [3] and its improvements [24]. TuckerMPI is a C++/MPI library that implements the STHOSVD algorithm to compute Tucker decompositions of large dense tensors that are distributed across machines. It implements many other utilities such as file I/O and subroutines such as parallel TTM, that we use in our algorithms and experiments. Other parallel implementations of Tucker algorithms have been developed for both dense [7] and sparse [21] tensors. The most similar work to ours combines parallelism and randomization to compute Tucker decompositions of dense tensors [12]. The approach taken by Choi, Liu, and Chakaravarthy [12] is to employ STHOSVD ([Alg. 2.2](#)) and compute the SVD of  $\mathbf{G}_{(j)}$

by computing its Gram matrix in parallel and then sequentially applying randomized SVD (Alg. 2.4) to the Gram matrix. Our approach differs in that we parallelize the randomized algorithm and avoid the Gram matrix computation; we provide a more detailed comparison in § 6.4.

We propose a novel implementation of a parallel algorithm for the Multi-TTM computation in § 6.1. Communication lower bounds for this computation and theoretical algorithms that achieve those lower bounds are presented in [2]. The Multi-TTM algorithm that we present in this paper can be seen as a specialization of [2, Alg. 8.1], but our implementation is novel. We also highlight previous optimizations of tensor computations using dimension trees, a memoization technique. First introduced in [26] in the context of computing gradients of the CP decomposition optimization problem, dimension trees have also been used for Tucker decompositions. For example, the Higher-Order Orthogonal Iteration (e.g., [15]) benefits from storing and reusing intermediate quantities across tensor modes as demonstrated in [20]. We use the dimension tree approach in a different context in one of our randomized algorithms; this process is described in § 4.4.

**4. Sequential Algorithms.** We present our novel sequential algorithms before discussing how they may be implemented in parallel. There are several different variations of algorithms we will present, and we provide a hierarchy diagram for how they relate in Figure 1. The first optimizations we show, using sequential truncation and randomization, have been developed in previous work. We then progress to using a Kronecker product of random matrices within the randomized algorithms, and then finally reusing Kronecker factors in the HOSVD case. The two most efficient algorithms we present are in the leftmost boxes of the two main subtrees: randomized STHOSVD with Kronecker products (Alg. 4.3) and randomized HOSVD with reused Kronecker factors (Alg. 4.4).

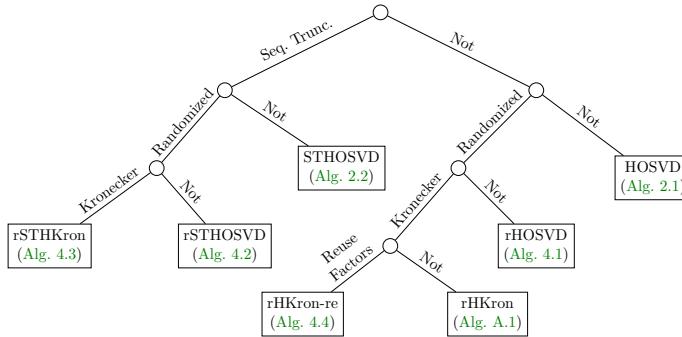


Fig. 1: Hierarchy of algorithms

In all the algorithms presented in this section, we employ a holistic truncation approach. Given a tensor  $\mathbf{X}$ , target rank  $\mathbf{r} = (r_1, \dots, r_d)$ , and oversampling parameter  $p$  and letting  $\ell_j = r_j + p$  for  $j = 1, \dots, d$ , we first apply the randomized range finder algorithm (Alg. 2.3) to each mode unfolding, and obtain an initial core  $\hat{\mathbf{G}} \in \mathbb{R}^{\ell_1 \times \dots \times \ell_d}$ . We then apply the truncation phase of Alg. 2.4 by computing a deterministic STHOSVD of  $\hat{\mathbf{G}}$  such that  $\hat{\mathbf{G}} \approx \mathbf{G} \times_1 \mathbf{V}_1 \times \dots \times_d \mathbf{V}_d$ . The rank- $\mathbf{r}$  representation of  $\mathbf{X}$  is then  $\mathbf{X} \approx \mathbf{G} \times_1 \hat{\mathbf{U}}_1 \mathbf{V}_1 \times \dots \times_d \hat{\mathbf{U}}_d \mathbf{V}_d$ .

**4.1. Randomized HOSVD/STHOSVD.** The first algorithms we present are the basic form of randomized algorithms on which we improve throughout the paper. [Algs. 4.1](#) and [4.2](#) are similar to other randomized projection algorithms for Tucker decompositions found in [\[1, 25\]](#) except for the truncation approach. Instead of directly applying the randomized SVD algorithm ([Alg. 2.4](#)) to each mode unfolding, we take the holistic approach described above. We can use this technique both with HOSVD, shown in [Alg. 4.1](#), and STHOSVD, shown in [Alg. 4.2](#).

---

**Algorithm 4.1** Randomized HOSVD

---

```

1: function  $[\mathcal{G}, \{\mathbf{U}_j\}] = \text{RHOSVD}(\mathbf{X}, \mathbf{r}, p)$ 
2:   for  $j = 1 : d$  do
3:     Draw  $\Omega \in \mathbb{R}^{n_j^{\otimes} \times \ell_j}$ 
4:      $\hat{\mathbf{U}}_j = \text{RandRangeFinder}(\mathbf{X}_{(j)}, \Omega)$ 
5:   end for
6:    $\hat{\mathcal{G}} = \mathbf{X} \times_1 \hat{\mathbf{U}}_1^\top \times \cdots \times_d \hat{\mathbf{U}}_d^\top$ 
7:    $[\mathcal{G}, \{\mathbf{V}_j\}] = \text{STHOSVD}(\hat{\mathcal{G}}, \mathbf{r})$ 
8:    $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$  for  $j = 1, \dots, d$ 
9: end function
```

---

**Algorithm 4.2** Randomized STHOSVD

---

```

1: function  $[\mathcal{G}, \{\mathbf{U}_j\}] = \text{RSTHOSVD}(\mathbf{X}, \mathbf{r}, p)$ 
2:    $\hat{\mathcal{G}} = \mathbf{X}$ 
3:   for  $j = 1 : d$  do
4:     Draw  $\Omega \in \mathbb{R}^{r_j^{\otimes} n_j^{\otimes} \times \ell_j}$ 
5:      $\hat{\mathbf{U}}_j = \text{RandRangeFinder}(\hat{\mathbf{G}}_{(j)}, \Omega)$ 
6:      $\hat{\mathcal{G}} = \hat{\mathcal{G}} \times_j \hat{\mathbf{U}}_j^\top$ 
7:   end for
8:    $[\mathcal{G}, \{\mathbf{V}_j\}] = \text{STHOSVD}(\hat{\mathcal{G}}, \mathbf{r})$ 
9:    $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$  for  $j = 1, \dots, d$ 
10: end function
```

---

**4.2. Randomized HOSVD/STHOSVD with Kronecker product.** Our main algorithm combines the HOSVD/STHOSVD algorithms with a special case of the randomized range finder algorithm used on each mode unfolding. Within the randomized range finder, we will represent the random matrix  $\Omega$  as a Kronecker product of random matrices each with a small number of columns instead of a single large  $\Omega$ . This allows us both to employ a Multi-TTM operation instead of matrix multiplication, reducing the computational complexity, and to exploit properties of tall-and-skinny matrices in our parallel algorithms. Specifically, for a tensor  $\mathbf{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ , rank  $\mathbf{r} = (r_1, \dots, r_d)$ , and oversampling parameter  $p$  with  $\ell_j = r_j + p$ , define  $\Omega_j \in \mathbb{R}^{n_j^{\otimes} \times \ell_j}$  as  $\Omega_j = (\Phi_{j,d} \otimes \cdots \otimes \Phi_{j,j+1} \otimes \Phi_{j,j-1} \otimes \cdots \otimes \Phi_{j,1})^\top$ , with  $\Phi_{j,k} \in \mathbb{R}^{s_{j,k} \times n_k}$  a random matrix from some distribution (e.g. Gaussian, SRHT, etc.), and  $\mathbf{S} \in \mathbb{N}^{d \times d}$  a matrix of subranks. We define  $\mathbf{S}$  to have entries  $s_{j,k}$  the  $k$ -th subrank for mode  $j$  where  $k \neq j$ , and diagonal entries  $s_{j,j} = 1$  for  $j = 1, \dots, d$ , such that the row products  $\prod_{k=1}^d s_{j,k} = \ell_j$ , for  $j = 1, \dots, d$ . We summarize the steps for our algorithm in STHOSVD form in [Alg. 4.3](#) (rSTHKron), and include an HOSVD version in [Alg. A.1](#). Note that Lines 5 to 7 in [Alg. 4.3](#) consist of applying randomized range finder to the current mode unfolding using the Kronecker product as our random matrix.

---

**Algorithm 4.3** Randomized STHOSVD with Kronecker product

---

```

1: function  $[\mathcal{G}, \{\mathbf{U}_j\}] = \text{RSTHKRON}(\mathbf{X}, \mathbf{r}, p)$ 
2:    $\hat{\mathcal{G}} = \mathbf{X}$ 
3:   Compute matrix of subranks  $\mathbf{S}$ 
4:   for  $j = 1 : d$  do
5:     Draw  $d - 1$  random matrices  $\Phi_{j,k} \in \mathbb{R}^{s_{j,k} \times \ell_k}$  for  $k < j$  and  $\Phi_{j,k} \in \mathbb{R}^{s_{j,k} \times n_k}$  for  $k > j$ 
6:      $\mathbf{Y} \leftarrow \hat{\mathcal{G}} \times_1 \Phi_{j,1} \times \cdots \times_{j-1} \Phi_{j,j-1} \times_{j+1} \Phi_{j,j+1} \times \cdots \times_d \Phi_{j,d}$ 
7:     Compute thin QR  $\mathbf{Y}_{(j)} = \hat{\mathbf{U}}_j \mathbf{R}$ 
8:      $\hat{\mathcal{G}} = \hat{\mathcal{G}} \times_j \hat{\mathbf{U}}_j^\top$ 
9:   end for
10:   $[\mathcal{G}, \{\mathbf{V}_j\}] = \text{STHOSVD}(\hat{\mathcal{G}}, \mathbf{r})$ 
11:   $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$  for  $j = 1, \dots, d$ 
12: end function
```

---

*Choosing subranks.* The restriction on the subranks  $\mathbf{S}$  is that  $\prod_{k \neq j}^d s_{j,k} = \ell_j = r_j + p$  for each row  $j = 1, \dots, d$ . In practice, we can actually choose the subranks so that  $\prod_{k \neq j}^d s_{j,k} \geq \ell_j$ . Satisfying this looser condition means we are essentially increasing the oversampling we are already doing through parameter  $p$ . This frees us to use heuristics to choose the subranks. We choose each row of  $\mathbf{S}$  to be composed of  $d - 1$  integer factors of  $\ell_j$ . In the case that  $\ell_j$  does not have exactly  $d - 1$  integer factors, we adjust the oversampling parameter until we can obtain the correct number of factors.

**4.3. Randomized HOSVD with Kronecker Factor Reuse.** An additional adaption we make to reduce the number of random values generated and computation is to reuse the components of the Kronecker product  $\boldsymbol{\Omega}$ . Instead of generating  $d - 1$  random matrices for each mode as in [Alg. 4.3](#), we generate  $d$  random matrices  $\{\Phi_j\}$  once, and use different combinations of  $d - 1$  of those same matrices in each mode. This approach is summarized in [Alg. 4.4](#). One benefit of this approach is that we generate significantly fewer random entries. We can also, as will be addressed in [§ 6](#), implement dimension trees to reduce computational cost. This variation only works for the HOSVD approach as the size of each  $\Phi_j$  remains the same, while it would change after each mode in an STHOSVD approach.

---

**Algorithm 4.4** Randomized HOSVD with Kronecker product re-using factors

---

```

1: function  $[\mathcal{G}, \{\mathbf{U}_j\}] = \text{RHKRON}(\mathfrak{X}, \mathbf{r}, p)$ 
2:   Compute subranks  $\mathbf{s}$ 
3:   Draw  $d$  random matrices  $\Phi_k \in \mathbb{R}^{s_k \times n_k}$  for  $k = 1, \dots, d$ 
4:   for  $j = 1 : d$  do
5:      $\mathcal{Y} \leftarrow \mathfrak{X} \times_1 \Phi_1 \times \cdots \times_{j-1} \Phi_{j-1} \times_{j+1} \Phi_{j+1} \times \cdots \times_d \Phi_d$ 
6:     Compute thin QR  $\mathbf{Y}_{(j)} = \mathbf{U}_j \mathbf{R}$ 
7:   end for
8:    $\hat{\mathcal{G}} = \mathfrak{X} \times_1 \hat{\mathbf{U}}_1^\top \times \cdots \times_d \hat{\mathbf{U}}_d^\top$ 
9:    $[\mathcal{G}, \{\mathbf{V}_j\}] = \text{STHOSVD}(\hat{\mathcal{G}}, \mathbf{r})$ 
10:   $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$  for  $j = 1, \dots, d$ 
11: end function
```

---

*Choosing subranks.* Note that in this case we compute only one vector of subranks  $\mathbf{s} \in \mathbb{N}^d$ , instead of a matrix as in [Alg. 4.3](#). We compute these subranks heuristically as well, and in this case in a straightforward manner, deriving the formula  $s_i = \lceil (\prod_{j=1}^d \ell_j)^{\frac{1}{d-1}} / \ell_i \rceil$ , from the conditions  $\mathbf{s} \in \mathbb{N}^d$  and  $\prod_{k=1}^{d-1} s_k \geq \ell_j$  for  $j = 1, \dots, d$ . This formula, while more straightforward, is more constricting than the method we use to compute subranks for [Alg. 4.3](#) because  $s_j \ell_j$  is approximately fixed for each mode  $j$ . This can create problems when computing with tensors with skewed modes and ranks, so we recommend not reusing Kronecker factors in these cases.

**4.4. Dimension Tree Optimization.** In [Line 5](#) of [Alg. 4.4](#), we perform a multi-TTM to compute the sketch tensor  $\mathcal{Y}$  for each mode, resulting in  $d$  multi-TTM products. Notice, however, that the  $d - 1$  random matrices that we use in each multi-TTM are drawn from the same set of  $d$  random matrices  $\{\Phi_1, \dots, \Phi_d\}$ . Thus a significant number of computations in each multi-TTM are repeated and can be reused. The dimension tree concept, which has been employed to reduce computational complexity in other algorithms [7, 20], also applies in this situation. In our implementation, we use a binary tree with  $d$  leaf nodes for a  $d$ -way tensor. For example, given a 4-way tensor  $\mathfrak{X}$  and four random matrices  $\{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$ , the 4 multi-TTM operations that would be carried out without a dimension tree are shown in the four leaf nodes of [Figure 2](#). The dimension tree provides an efficient way to perform the computations shared between each pair of adjacent leaf nodes and store the results in memory to

reduce computation. For a  $d$ -way tensor  $\mathcal{X} \in \mathbb{R}^{n \times \dots \times n}$  and  $d$  random matrices  $\{\Phi_j\}$  each of size  $r \times n$ , the cost of using dimension trees to sketch every unfolding of  $\mathcal{X}$  is shown in [Table 1](#). When  $r \ll n$ , both costs are approximated by the first terms in the summations. As  $d$  increases, the cost reduction that comes from using a dimension tree increases proportionally to  $d/2$ .

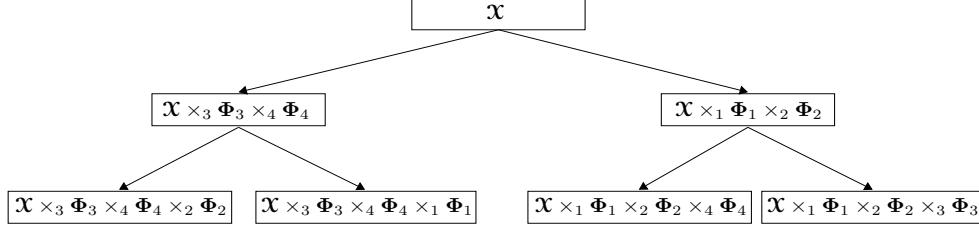


Fig. 2: Dimension tree for computing the sketches of a 4-way tensor  $\mathcal{X}$  in [Alg. 4.4](#).

	with dimTree	without dimTree
$d = 3$	$2(2rn^3 + 3r^2n^2)$	$2(3rn^3 + 3r^2n^2)$
$d = 4$	$2(2rn^4 + 2r^2n^3 + 4r^3n^2)$	$2(4rn^4 + 4r^2n^3 + 4r^3n^2)$
$d = 5$	$2(2rn^5 + 2r^2n^4 + 3r^3n^3 + 5r^4n^2)$	$2(5rn^5 + 5r^2n^4 + 5r^3n^3 + 5r^4n^2)$

Table 1: Complexity reduction examples that can be achieved using dimension trees.

**4.5. Computational Complexity.** To analyze the computational cost of our algorithms, we consider the notationally simpler case with a  $d$ -mode tensor  $\mathcal{X} \in \mathbb{R}^{n \times \dots \times n}$ , target rank  $(r, \dots, r)$ , and oversampling parameter  $p$ , letting  $\ell = r + p$ . We will also let the subranks  $\{s_{j,k}\}$  and  $\{s_k\}$  all be the same value, which we denote as  $s = \ell^{d-1}$ . Assume  $s < r \ll n$ .

There are two dominant costs for each algorithm presented: computing an SVD for each mode, and forming the core tensor via a multi-TTM or a series of TTMs (in the STHOSVD case). We show the leading terms of both dominant steps for the standard algorithms, [Algs. 2.1](#) and [2.2](#), and compare to those for [Algs. 4.1](#) to [4.4](#) in [Table 2](#). More details on the cost analysis for [Algs. 2.1](#) and [2.2](#) can be found in [29], while more details on the analysis for [Alg. 4.1](#) and [Alg. 4.2](#) can be found in [25]. Based on the terms shown in [Table 2](#), we advocate the use of either [Alg. 4.3](#) or [Alg. 4.4](#), and we show more detailed analysis on these two algorithms in [Appendix B](#).

Algorithm	Leading term	
	SVD	TTM
HOSVD ( <a href="#">2.1</a> )	$dn^{d+1}$	$2rn^d$
STHOSVD ( <a href="#">2.2</a> )	$n^{d+1}$	$2rn^d$
rHOSVD ( <a href="#">4.1</a> )	$2d\ell n^d$	$2\ell n^d$
rSTHOSVD ( <a href="#">4.2</a> )	$2\ell n^d$	$2\ell n^d$
rSTHOSVdkron ( <a href="#">4.3</a> )	$2\ell^{\frac{1}{d-1}} n^d$	$2\ell n^d$
rHOSVdkronreuse ( <a href="#">4.4</a> )	$4\ell^{\frac{1}{d-1}} n^d$	$2\ell n^d$

Table 2: Computational Cost

**4.6. Comparison with Previous Work.** We compare our sequential algorithms with previous approaches, in particular those based on the use of randomized SVD [25] and Kronecker-structured random matrices [11]. As summarized in [1],

randomization can be used in multiple ways to compute Tucker approximations of tensors. The most similar approaches to [Algs. 4.1](#) and [4.2](#) are [25, Algs. 3.1 and 3.2]. These algorithms replace the deterministic SVD within [Algs. 2.1](#) and [2.2](#) with randomized SVD ([Alg. 2.4](#)). The randomized SVD requires computing the thin SVD of the projection of the approximate column space; it increases the computational cost compared to randomized range finder by a constant factor greater than 2. Because [Algs. 4.1](#) and [4.2](#) use randomized range finder ([Alg. 2.3](#)), they involve only one operation with the input (the random sketch) at the expense of working with the oversampled rank  $\ell$  rather than the target rank  $r$  until the final core truncation step. The oversampled ranks are only slightly larger than the target ranks, so [Algs. 4.1](#) and [4.2](#) are computationally cheaper than [25, Algs. 3.1 and 3.2].

The Kronecker-structured random sketches of [Algs. 4.3](#) and [4.4](#) are similar to [11, Alg. 3.1]. This algorithm uses a Sampled Random Fourier Transform to sketch each mode's matricization within STHOSVD. Besides using a complex-valued random matrix, the key difference with [Alg. 4.3](#) is the truncation strategy: after computing the thin QR decomposition of the sketched matrix with  $\ell$  columns, all but the first  $r$  columns are truncated. As we demonstrate in § 7.1, our truncation strategy using a deterministic STHOSVD of the core tensor computes a more accurate approximation.

To the best of our knowledge, reusing Kronecker factors and exploiting the possible memoization of temporary quantities as presented in § 4.3 and 4.4 have not been considered before.

**5. Error Analysis.** We now present error guarantees for [Algs. 4.3](#) and [4.4](#). Let  $\mathcal{T} = [\mathcal{G}; \mathbf{U}_1, \dots, \mathbf{U}_d]$  be the approximation from either algorithm, and  $\hat{\mathcal{T}} = [\hat{\mathcal{G}}; \hat{\mathbf{U}}_1, \dots, \hat{\mathbf{U}}_d]$  be the intermediate rank- $\ell$  approximation. The overall form of the error is

$$\varepsilon_{\text{total}} = \|\mathcal{X} - \mathcal{T}\| \leq \|\mathcal{X} - \hat{\mathcal{T}}\| + \|\hat{\mathcal{T}} - \mathcal{T}\| = \varepsilon_{\text{rand}} + \varepsilon_{\text{core}},$$

where  $\varepsilon_{\text{rand}}$  represents the error from forming the rank- $\ell$  approximation, and  $\varepsilon_{\text{core}}$  represents the error in truncating the approximation to rank  $r$ . The component  $\varepsilon_{\text{core}}$  is equivalent to the error in computing the STHOSVD of  $\hat{\mathcal{G}}$ , which we can see from

$$\varepsilon_{\text{core}} = \|(\hat{\mathcal{G}} - \mathcal{G} \times_1 \mathbf{V}_1 \times \cdots \times_d \mathbf{V}_d) \times_1 \hat{\mathbf{U}}_1 \times \cdots \times_d \hat{\mathbf{U}}_d\| = \|\hat{\mathcal{G}} - \mathcal{G} \times_1 \mathbf{V}_1 \times \cdots \times_d \mathbf{V}_d\|,$$

where the second equality follows from the orthonormality of  $\{\hat{\mathbf{U}}_j\}$ . We can then apply the error bound for STHOSVD [29, Theorem 6.5] to initial core  $\hat{\mathcal{G}}$ , i.e.,  $\varepsilon_{\text{core}}^2 \leq \sum_{j=1}^d \sum_{i=r_j+1}^{\ell_j} \sigma_i^2(\hat{\mathbf{G}}_{(j)})$ , where  $\sigma_i(\mathbf{A})$  denotes the  $i$ th singular value of  $\mathbf{A}$ .

Then, as  $\hat{\mathcal{G}}$  is a random quantity, we need to relate the singular values of  $\hat{\mathbf{G}}_{(j)}$  to the singular values of  $\mathbf{X}_{(j)}$  to get a deterministic upper bound. Because matrices  $\{\hat{\mathbf{U}}_j\}$  are orthonormal, the mode-wise singular values of  $\hat{\mathcal{G}}$  cannot be larger than those of  $\mathcal{X}$ . A formal proof of this fact can be found in [Appendix C](#). Thus, we have

$$(5.1) \quad \varepsilon_{\text{core}}^2 \leq \sum_{j=1}^d \sum_{i=r_j+1}^{\ell_j} \sigma_i^2(\mathbf{X}_{(j)}).$$

The rest of this section considers the component  $\varepsilon_{\text{rand}}$ . Starting with an error bound for randomized range finder ([Alg. 2.3](#)) using a Kronecker product of SRHT matrices, we extend the results to an error bound for our HOSVD-type algorithm ([Alg. 4.4](#)) and discuss how to adapt the proof for our STHOSVD-type algorithm ([Alg. 4.3](#)).

### 5.1. Matrix Bound.

*Random Matrix.* Let  $n = \prod_{j=1}^q n_j$  and  $\ell = \prod_{j=1}^q s_j$ . We will consider a Kronecker product of SRHT matrices of the form

$$(5.2) \quad \Omega = \mathbf{D}(\mathbf{H}_1 \otimes \mathbf{H}_2 \otimes \cdots \otimes \mathbf{H}_q) \in \mathbb{R}^{n \times \ell},$$

where  $\mathbf{D} \in \mathbb{R}^{n \times n}$  is a diagonal matrix with i.i.d. entries from the Rademacher distribution, i.e. either 1 or  $-1$  with equal probability, and for each  $j = 1, \dots, q$ ,  $\mathbf{H}_j \in \mathbb{R}^{n_j \times s_j}$  is formed from  $s_j$  randomly sampled columns of an  $n_j \times n_j$  Walsh-Hadamard matrix scaled by  $\frac{1}{\sqrt{n_j}}$ . Due to this scaling,  $\Omega$  is orthonormal. Also note that the  $\{\mathbf{H}_j\}$  matrices are generated independently.

*Notation.* We now introduce the setup and notation for our main theorem. Following the notation of [17], let  $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$  be the SVD of matrix  $\mathbf{X} \in \mathbb{R}^{m \times n}$  with  $m \leq n$ . Fix target rank  $r$ , and partition the SVD as

$$(5.3) \quad \mathbf{X} = \mathbf{U} \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^\top \\ \mathbf{V}_2^\top \end{bmatrix},$$

with  $\Sigma_1 \in \mathbb{R}^{r \times r}$ ,  $\Sigma_2 \in \mathbb{R}^{(m-r) \times (m-r)}$ ,  $\mathbf{V}_1 \in \mathbb{R}^{n \times r}$ , and  $\mathbf{V}_2 \in \mathbb{R}^{n \times (m-r)}$  such that  $\mathbf{V}_1$  and  $\mathbf{V}_2$  have orthonormal columns. Now let  $\Omega \in \mathbb{R}^{n \times \ell}$  be the random matrix defined in (5.2), and define

$$(5.4) \quad \Omega_1 = \mathbf{V}_1^\top \Omega, \quad \Omega_2 = \mathbf{V}_2^\top \Omega.$$

We are interested in bounding the largest singular values of  $\Omega_1^\dagger$  and  $\Omega_2$ . We will use the orthonormality of  $\Omega_2$  to bound its largest singular values, but the argument is more complicated for  $\Omega_1$ . Here we will adapt the approach in [31], allowing for the application to a Kronecker product of independent SRHT matrices and making other minor improvements. This bound is stated in Lemma 5.1, which we will then use to prove our approximation error bound. We prove Lemma 5.1 in Appendix D.

LEMMA 5.1. *Let  $\Omega_1 \in \mathbb{R}^{r \times \ell}$  as defined in (5.4), with  $\Omega \in \mathbb{R}^{n \times \ell}$  the Kronecker product of  $q$  SRHT matrices as defined in (5.2), where  $n = \prod_{j=1}^q n_j$  and  $\ell = \prod_{j=1}^q s_j$ . Let  $\alpha, \beta > 1$  be real numbers that satisfy*

$$(5.5) \quad \min_k \{s_k\} \geq \frac{\alpha^2 \beta}{(\alpha - 1)^2} (r^2 + r).$$

*Then  $\frac{1}{\sigma_{\min}^2(\Omega_1)} \leq \frac{\alpha n}{\ell}$ , with probability at least  $1 - \frac{1}{\beta^2}$ .*

*Remark 5.2.* The bound in Lemma 5.1 contains a factor of  $n$ , the number of rows of Kronecker product  $\Omega$ . This factor is not present in singular value bounds for Gaussian random matrices, and is a consequence of using SRHT matrices, as seen in bounds involving a single SRHT matrix in [28]. We choose to use SRHT instead of Gaussian matrices in our context because the Kronecker product of Gaussian matrices is no longer Gaussian, while the Kronecker product of SRHT matrices retains some SRHT properties. In the empirical results shown in § 7.1, we see that using SRHT matrices produces similar accuracy to Gaussian matrices; we anticipate that future work in random matrix theory will be able to improve this factor.

THEOREM 5.3. *Let  $\hat{\mathbf{X}} = \mathbf{Q}\mathbf{Q}^\top \mathbf{X}$  be the approximation given by the randomized range finder of matrix  $\mathbf{X} \in \mathbb{R}^{m \times n}$  with target rank  $r$ , oversampling parameter  $p$  such that  $\ell = r + p \leq \min\{m, n\}$ , and random sampling matrix  $\Omega$  as defined in (5.2). Let  $\alpha, \beta > 1$  satisfy (5.5). Then, with probability at least  $1 - \frac{1}{\beta^2}$ ,*

$$\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top \mathbf{X}\|_F \leq \left( \left(1 + \frac{\alpha n}{\ell}\right) \sum_{i=r+1}^{\min\{m,n\}} \sigma_i^2(\mathbf{X}) \right)^{1/2}.$$

*Proof.* Immediately from [17, Theorem 9.1] and recalling the partitioning from (5.3) and (5.4), we have

$$(5.6) \quad \begin{aligned} \|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top \mathbf{X}\|_F^2 &= \|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top)\mathbf{X}\|_F^2 \leq \|\Sigma_2\|_F^2 + \|\Sigma_2\Omega_2\Omega_1^\dagger\|_F^2 \\ &\leq \left(1 + \|\Omega_2\|_2^2\|\Omega_1^\dagger\|_2^2\right) \|\Sigma_2\|_F^2, \end{aligned}$$

We can apply the singular value bounds from Lemma 5.1 to obtain the bounds for  $\|\Omega_1^\dagger\|_2^2$ . With probability at least  $1 - \frac{1}{\beta^2}$ ,  $\|\Omega_1^\dagger\|_2^2 = \frac{1}{\sigma_{\min}^2(\Omega_1)} \leq \frac{\alpha n}{\ell}$ . For  $\|\Omega_2\|_2^2$ , we use properties of both  $\mathbf{V}_2$  and  $\Omega$ :  $\|\Omega_2\|_2 = \|\mathbf{V}_2^\top \Omega\|_2 \leq \|\Omega\|_2 = 1$ , as  $\mathbf{V}_2^\top$  has orthonormal rows, and  $\Omega$  has orthonormal columns as the Kronecker product of matrices with orthonormal columns. Combining these bounds, we obtain  $\|\Omega_2\|_2^2\|\Omega_1^\dagger\|_2^2 \leq \frac{\alpha n}{\ell}$  with probability at least  $1 - \frac{1}{\beta^2}$ . From (5.6), we now have  $\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top \mathbf{X}\|_F^2 \leq \left(1 + \frac{\alpha n}{\ell}\right) \|\Sigma_2\|_F^2 = \left(1 + \frac{\alpha n}{\ell}\right) \sum_{i=r+1}^{\min\{m,n\}} \sigma_i^2(\mathbf{X})$ , and the result follows.  $\square$

**5.2. Tensor Bound.** To generalize the result from Theorem 5.3 to higher dimensions, we first need a result that expresses the error in a Tucker decomposition in terms of the error in each mode.

LEMMA 5.4 ([29, Theorem 5.1]). *Let  $\mathbf{X} \in \mathbb{R}^{n_1 \times \dots \times n_d}$  and let  $\mathbf{P}_j \in \mathbb{R}^{n_j \times n_j}$  for  $j = 1, \dots, d$  be a sequence of orthogonal projectors. Then*

$$\|\mathbf{X} - \mathbf{X} \times \mathbf{P}_j\|^2 = \sum_{j=1}^d \sum_{i=1}^{j-1} \|\mathbf{X} \times \mathbf{P}_i \times_j (\mathbf{I} - \mathbf{P}_j)\|^2 \leq \sum_{j=1}^d \|\mathbf{X} - \mathbf{X} \times_j \mathbf{P}_j\|^2.$$

Recall the notation  $n_j^\otimes = \prod_{k \neq j}^d n_k$ . We present our main error bound result in Theorem 5.5, which we frame as the error bound for Alg. 4.4. This result can be adapted to also apply to Alg. 4.3 by following similar techniques to [25, Theorem 3.2]. We include the details for completeness in Appendix E.

**THEOREM 5.5.** *Let  $\mathcal{T} = [\mathbf{G}; \mathbf{U}_1, \dots, \mathbf{U}_d]$  be the approximation given by Alg. 4.4 to  $\mathbf{X} \in \mathbb{R}^{n_1 \times \dots \times n_d}$  with target rank  $\mathbf{r} = (r_1, \dots, r_d)$  and oversampling parameter  $p$ . Let  $\ell_j = r_j + p$  for  $j = 1, \dots, d$ . Then, for sequences  $\{\alpha_j\}_{j=1}^d$  and  $\{\beta_j\}_{j=1}^d$  satisfying (5.5), the following bound holds with probability at least  $1 - \sum_{j=1}^d \frac{1}{\beta_j^2}$ ,*

$$\|\mathbf{X} - \mathcal{T}\| \leq \left( \sum_{j=1}^d \left( 1 + \frac{\alpha_j n_j^\otimes}{\ell_j} \right) \sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2} + \left( \sum_{j=1}^d \sum_{i=r_j+1}^{\ell_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2}.$$

*Proof.* Using (5.1) to bound  $\varepsilon_{\text{core}}$ , we need only to bound  $\varepsilon_{\text{rand}}$ . From Lemma 5.4, we have for  $\hat{\mathcal{T}} = [\hat{\mathbf{G}}; \hat{\mathbf{U}}_1, \dots, \hat{\mathbf{U}}_d]$  computed as the intermediate rank- $\ell$  approximation,

$$\|\mathbf{X} - \hat{\mathcal{T}}\|^2 = \|\mathbf{X} - \mathbf{X} \times \sum_{j=1}^d \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top\|^2 \leq \sum_{j=1}^d \|\mathbf{X} - \mathbf{X} \times_j \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top\|^2 = \sum_{j=1}^d \|\mathbf{X}_{(j)} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top \mathbf{X}_{(j)}\|_F^2,$$

where the last equality comes from unfolding the tensor along the  $j$ -th mode for  $j = 1, \dots, d$ . We now apply the matrix bound from Theorem 5.3 on each term in this

sum, which gives  $\|\mathbf{X}_{(j)} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top \mathbf{X}_{(j)}\|_F^2 \leq \left(1 + \frac{\alpha_j n_j^\varnothing}{\ell_j}\right) \sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)})$ , except with failure probability at most  $\frac{1}{\beta_j^2}$ . Then the failure probability for the entire sum is the union of all  $d$  failure probabilities for each mode, which is bounded above by the sum of those probabilities by the union bound. Thus,

$$(5.7) \quad \|\mathbf{X} - \hat{\mathbf{T}}\|^2 \leq \sum_{j=1}^d \|\mathbf{X}_{(j)} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top \mathbf{X}_{(j)}\|_F^2 \leq \sum_{j=1}^d \left(1 + \frac{\alpha_j n_j^\varnothing}{\ell_j}\right) \sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)}),$$

except with probability at most  $\sum_{j=1}^d \frac{1}{\beta_j^2}$ . Then, taking square roots gives  $\varepsilon_{\text{rand}}$ .

Combining (5.7) and (5.1), the total error in approximation is

$$\varepsilon_{\text{total}} \leq \left( \sum_{j=1}^d \left(1 + \frac{\alpha_j n_j^\varnothing}{\ell_j}\right) \sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2} + \left( \sum_{j=1}^d \sum_{i=r_j+1}^{\ell_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2},$$

with probability at least  $1 - \sum_{j=1}^d \frac{1}{\beta_j^2}$ .  $\square$

We consider this result pessimistic due to the factor of  $n_j^\varnothing$  that comes directly from our analysis of SRHT matrices, but does not appear in our accuracy experiments in § 7.1. We anticipate that this factor could be improved in future analysis. Also note that the result of Theorem 5.5 differs from [11, Theorem 4.2] in two main ways: our probability of failure is smaller ( $\frac{1}{\beta^2}$  per mode compared to  $\frac{1}{\bar{\beta}}$ ), and our constant in  $\varepsilon_{\text{rand}}$  is smaller as we divide by  $\ell_j$  for each  $j$ . Additionally, we use a different diagonal matrix that is not a Kronecker product.

**6. Parallel Algorithms.** We design and develop parallel implementations for all the randomized algorithms listed in Table 2. Our implementations are based on TuckerMPI [3], which uses the STHOSVD algorithm. Similarly to TuckerMPI, our implementations leverage distributed-memory clusters to efficiently compute the Tucker decomposition of large multidimensional datasets. We employ the following data distribution scheme, proposed in [3], in our implementations. To distribute a  $d$ -way input tensor, the processors are organized in a  $d$ -way processor grid, the dimensions of which are user-determined. Each processor owns a subtensor of the input tensor. For example, for a  $8 \times 6 \times 2$  tensor and a  $2 \times 3 \times 1$  processor grid, each processor owns a  $4 \times 2 \times 2$  subtensor. All matrices involved in our algorithms are stored redundantly by every processor.

In the following sections we describe two optimizations for computing the sketch tensor via multi-TTMs as well as parallel implementations of the two algorithms from § 4 with the smallest computational cost, Alg. 4.3 and Alg. 4.4. We also compare our implementations with previous work from [12].

**6.1. All-at-Once Multi-TTM.** The multi-TTM operation is one of the most expensive kernels of our algorithms and appears twice; we compute a multi-TTM both to form the sketch tensor  $\mathbf{Y}$  and to form the core tensor  $\hat{\mathbf{G}}$  (e.g. in Line 5 of Alg. 4.4). It is thus crucial to optimize this operation.

A parallel implementation of a single TTM is proposed in [3]. One way to implement the multi-TTM is to simply perform this existing TTM algorithm multiple times, as shown in Alg. 6.1; we call this approach the in-sequence multi-TTM or IS-mTTM. Note that in the algorithms bars over letters denote local data. In this

in-sequence approach, a reduce-scatter is performed at the end of each TTM operation, reducing the amount of data each processor owns so that the computation cost in the next TTM is also reduced. Generally, this approach performs additional communications to obtain lower computational cost. However, depending on the size of the local tensor, the reduction in computational cost may not justify the increased communication cost.

Our algorithm is shown in [Alg. 6.2](#), which we call the all-at-once multi-TTM or

---

**Algorithm 6.1** In-sequence multi-TTM

---

```

1: function  $\bar{\mathbf{y}} = \text{IS-MTTM}(\bar{\mathbf{x}}, j, \{\mathbf{M}_j\}, \mathcal{P})$ 
2:    $(p_1, \dots, p_d) = \text{procID}(\mathcal{P})$ 
3:    $\bar{\mathbf{T}} = \bar{\mathbf{x}}$ 
4:   for  $i = 1 : d$  and  $i \neq j$  do
5:      $\mathcal{F} = \mathcal{P}(p_1, \dots, p_{i-1}, :, p_{i+1}, \dots, p_d)$ 
6:      $\bar{\mathbf{T}}_{(i)} = \bar{\mathbf{M}}_i \bar{\mathbf{T}}_{(i)}$ 
7:      $\bar{\mathbf{Y}}_{(i)} = \text{REDUCE-SCATTER}(\bar{\mathbf{T}}_{(i)}, \mathcal{F})$ 
8:   end for
9: end function
```

---

**Algorithm 6.2** All-at-once multi-TTM

---

```

1: function  $\bar{\mathbf{y}} = \text{AAO-MTTM}(\bar{\mathbf{x}}, j, \{\mathbf{M}_j\}, \mathcal{P})$ 
2:    $(p_1, \dots, p_d) = \text{procID}(\mathcal{P})$ 
3:    $\bar{\mathbf{T}} = \bar{\mathbf{x}}$ 
4:   for  $i = 1 : d$  and  $i \neq j$  do
5:      $\bar{\mathbf{T}}_{(i)} = \bar{\mathbf{M}}_i \bar{\mathbf{T}}_{(i)}$ 
6:   end for
7:    $\mathcal{S} = \mathcal{P}(:, \dots, :, p_j, :, \dots, :)$ 
8:    $\bar{\mathbf{Y}}_{(j)} = \text{REDUCE-SCATTER}(\bar{\mathbf{T}}_{(j)}, \mathcal{S})$ 
9: end function
```

---

AAO-mTTM. In this approach, we avoid communication until all matrices have been multiplied with the local tensor. This strategy reduces communication by increasing the cost of storage and computation.

The most significant difference between [Alg. 6.1](#) and [Alg. 6.2](#) is that at the end of any iteration  $i$  of the in-sequence approach, we form the intermediate result  $\mathbf{Y} = \mathbf{X} \times_1 \mathbf{M}_1 \times \cdots \times_i \mathbf{M}_i$ ; in the all-at-once approach, each processor stores a contribution to  $\mathbf{Y}$  until all iterations are completed and the all-reduce at the end of the algorithm forms the final result.

**6.1.1. Cost Analysis.** To simplify the notation, we assume that the input tensor is a  $d$ -way cubic tensor  $\mathbf{X} \in \mathbb{R}^{n \times \cdots \times n}$ , that the processor tensor is size  $q$  in each mode ( $q^d = P$  processors in total), and that the input matrices  $\{\mathbf{M}_i\}_{i=1}^d$  are of the same size  $s \times n$  with  $s < n$ . With this notation, we analyze the per-processor computation and communication costs of performing the multi-TTM  $\mathbf{X} \times_1 \mathbf{M}_1 \times \cdots \times_{j-1} \mathbf{M}_{j-1} \times_{j+1} \mathbf{M}_{j+1} \times \cdots \times_d \mathbf{M}_d$  using [Alg. 6.2](#) and compare it with that of using [Alg. 6.1](#).

The computational cost of [Alg. 6.1](#) can be written as

$$(6.1) \quad C_{\text{in-sequence}} = 2 \left( \frac{sn^d}{q^d} + \frac{s^2 n^{d-1}}{q^d} + \dots + \frac{s^d n}{q^d} \right) = 2 \left( \sum_{i=1}^d \frac{s^i n^{d-i+1}}{q^d} \right),$$

and the computational cost of [Alg. 6.2](#) can be written as

$$(6.2) \quad C_{\text{all-at-once}} = 2 \left( \frac{sn^d}{q^d} + \frac{s^2 n^{d-1}}{q^{d-1}} + \dots + \frac{s^d n}{q^2} \right) = 2 \left( \sum_{i=1}^d \frac{s^i n^{d-i+1}}{q^{d-i+1}} \right).$$

The  $i$ th terms of the summations in both (6.2) and (6.1) represent the cost of multiplying the local factor matrix  $\bar{\mathbf{M}}_i$  of size  $s \times \frac{n}{q}$  and the  $i$ th mode unfolding of tensor  $\bar{\mathbf{y}} = \bar{\mathbf{x}} \times_1 \bar{\mathbf{M}}_1 \times \cdots \times_{i-1} \bar{\mathbf{M}}_{i-1}$ . In [Alg. 6.1](#), due to the reduce-scatter at each iteration,  $\bar{\mathbf{Y}}_{(i)}$  is of size  $\frac{n}{q} \times \frac{s^{i-1} n^{d-i}}{q^{d-i}}$ . In [Alg. 6.2](#),  $\bar{\mathbf{M}}_i$  is of the same size. However, since the reduction is delayed until the last mode,  $\bar{\mathbf{Y}}_{(i)}$  is of size  $\frac{n}{q} \times \frac{s^{i-1} n^{d-i}}{q^{d-i}}$ . Comparing (6.1) and (6.2), it is easy to see that the computational cost of an all-at-once multi-TTM is always higher than that of an in-sequence multi-TTM because

each of the summands in (6.2) is at least as large as the corresponding term in (6.1). This increase can be small, however, in certain cases. Note the two series of summands are geometric and have the same leading term. For (6.1), the ratio of the series is  $\frac{s}{n}$  while the ratio of the series in (6.2) is  $\frac{sq}{n}$ . As the subranks become smaller compared to the tensor dimensions (i.e.  $s \ll n$ ), the sums of the two series get closer to their first terms and thus their difference becomes smaller.

Using the same notations and the  $\alpha\beta\gamma$  model [8], we can express the communication cost of Alg. 6.1 and Alg. 6.2. In Alg. 6.2, there is only one communication step at the end where all  $P$  processors communicate their local tensor  $\bar{\mathcal{T}}$ .  $\bar{\mathcal{T}}$  is a  $d$ -way tensor with size  $s$  for all of its modes except for the  $j$ th mode which has size  $\frac{n}{q}$ . Therefore, the communication cost for each processor is:  $\alpha\mathcal{O}(\log P) + \beta\mathcal{O}\left(\frac{s^{d-1}n}{q}\right)$ . The communication cost of Alg. 6.1 is more complicated because there are  $d-1$  communication steps and each one involves  $\bar{\mathcal{T}}_{(i)}$  which is changing in size. At the  $i$ th iteration of the for loop,  $\bar{\mathcal{T}}_{(i)}$  has size  $s \times \frac{s^{i-1}n^{d-i}}{q^{d-1}}$ . Therefore, the communication cost can be written as  $\alpha\mathcal{O}(d\log q) + \beta\mathcal{O}\left(\sum_{i=1}^{d-1} \frac{s^i n^{d-i}}{q^{d-1}}\right) = \alpha\mathcal{O}(\log P) + \beta\mathcal{O}\left(\frac{sn^{d-1}}{q^{d-1}}\right)$ . We obtain the right hand side of the equation using the assumption that  $s \ll n$  so that the summation is approximated by its first summand. When  $sq < n$ , the communication cost of all-at-once multi-TTM is smaller. As the ratio  $\frac{n}{sq}$  increases, the benefits of using all-at-once multi-TTM become more substantial.

**6.2. Dimension Tree Optimization.** In § 4.4, we discuss how dimension trees can be used to make the randomized sketches less expensive for Alg. 4.4. In Alg. 6.3, we present an implementation using AAO-mTTM. Here,  $\mathbf{m}$  and  $\mathbf{n}$  are sets of integers in the range  $[1, d]$ . This algorithm returns  $\mathcal{Y}^{(j)}$ , the sketch of  $\mathbf{X}_{(j)}$  in tensor format, for all integers  $j \in [1, d]$ . Since dimension trees are a tool for reusing only local intermediate results, Alg. 6.3 can also be modified to use IS-mTTM.

---

**Algorithm 6.3** All modes multi-TTM using the dimension tree optimization and all-at-once multi-TTM.

---

```

1: function  $\{\mathcal{Y}^{(j)}\} = \text{ALL-MODES-MULTI-TTM}(\bar{\mathcal{X}}, \{\bar{\Phi}_j\}, \mathbf{m}, \mathbf{n}, \mathcal{P})$ 
2:    $(p_1, \dots, p_d) = \text{procID}(\mathcal{P})$ 
3:    $\bar{\mathcal{Y}} = \bar{\mathcal{X}}$ 
4:   for  $i \in \mathbf{n}$  do
5:      $\bar{\mathcal{Y}}_{(i)} = \bar{\Phi}_i \bar{\mathcal{Y}}_{(i)}$ 
6:   end for
7:   if  $\mathbf{m}$  only contains 1 integer,  $j$  then
8:      $\mathcal{S} = \mathcal{P}(:, \dots, p_j, \dots, :)$ 
9:      $\bar{\mathcal{Y}}^{(j)} = \text{REDUCE-SCATTER}(\bar{\mathcal{Y}}, \mathcal{S})$ 
10:  else
11:    split  $\mathbf{m}$  in equal half  $\mathbf{m}_1$  and  $\mathbf{m}_2$ 
12:    ALL-MODES-MULTI-TTM( $\bar{\mathcal{Y}}, \{\bar{\Phi}_j\}, \mathbf{m}_1, \mathbf{m}_2, \mathcal{P}$ )
13:    ALL-MODES-MULTI-TTM( $\bar{\mathcal{Y}}, \{\bar{\Phi}_j\}, \mathbf{m}_2, \mathbf{m}_1, \mathcal{P}$ )
14:  end if
15: end function

```

---

**6.3. Principal Algorithms.** We now combine the discussed multi-TTM approaches and the dimension tree optimization within parallel implementations of our two best algorithms, Algs. 4.3 and 4.4. The pseudocode is provided in Algs. 6.4 and 6.5, respectively. Note that for both of these algorithms, when the size of core  $\hat{\mathcal{G}}$  is large, it is better to perform the in-sequence multi-TTM as described in Alg. 6.1 to produce a distributed  $\hat{\mathcal{G}}$  so that the parallel STHOSVD can be used to reduce cost.

When  $\hat{\mathbf{G}}$  is very small, performing STHOSVD in parallel can be counterproductive as the communication cost will dominate; it is better in this case to perform an all-gather among all processors after the in-sequence multi-TTM, producing  $\hat{\mathbf{G}}$  redundantly on every processor and then use the sequential STHOSVD.

---

**Algorithm 6.4** Parallel algorithm for [Alg. 4.3](#)


---

```

1: function RSTHKRON( $\mathcal{X}$ ,  $\mathbf{r}$ ,  $p$ ,  $\mathcal{P}$ )       $\triangleright \mathcal{X}$  is distributed, the local part of  $\mathcal{X}$  is denoted as  $\bar{\mathcal{X}}$ 
2:    $\hat{\mathbf{G}} = \mathcal{X}$ 
3:   Redundantly compute matrix of subranks  $\mathbf{S}$ 
4:   for  $i = 1:d$  do
5:     for  $j = 1:d$  and  $j \neq i$  do
6:       Redundantly draw  $d - 1$  random matrices  $\Phi_{i,j} \in \mathbb{R}^{n_j \times s_{i,j}}$ 
7:     end for
8:      $\bar{\mathbf{Y}} \leftarrow \text{AAO-MTTM}(\hat{\mathbf{G}}, i, \{\Phi_{i,1} \dots \Phi_{i,d}\}, \mathcal{P})$        $\triangleright$  Can also use in-sequence multi-TTM
9:      $\mathbf{Y} = \text{ALL-GATHER}(\bar{\mathbf{Y}}, \mathcal{P})$ 
10:     $\hat{\mathbf{U}}_i = \text{QR}(\mathbf{Y}_{(i)})$            $\triangleright$  QR is serial as every processor owns global  $\mathbf{Y}$ 
11:     $\hat{\mathbf{G}} \leftarrow \text{TTM}(\hat{\mathbf{G}}, \hat{\mathbf{U}}_i^\top, i)$      $\triangleright$  For a single TTM we use the implementation proposed in [3]
12:   end for
13:    $[\mathcal{G}, \{\mathbf{V}_i\}] = \text{STHOSVD}(\hat{\mathbf{G}}, \mathbf{r})$ 
14:   for  $i = 1, \dots, d$  do
15:      $\mathbf{U}_i = \hat{\mathbf{U}}_i \mathbf{V}_i$            $\triangleright$  Computed with local matrix multiplication
16:   end for
17: end function

```

---

**Algorithm 6.5** Parallel algorithm for [Alg. 4.4](#) with all-at-once multi-TTM and dimension trees

---

```

1: function RHKRON-RE( $\mathcal{X}$ ,  $\mathbf{r}$ ,  $p$ ,  $\mathcal{P}$ )
2:   Compute subranks  $\mathbf{s}$ 
3:   for  $i = 1:d$  do
4:     Redundantly draw  $d - 1$  random matrices  $\Phi_i \in \mathbb{R}^{s_i \times n_i}$ 
5:   end for
6:    $\{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(d)}\} = \text{ALL-MODES-MULTI-TTM}(\bar{\mathcal{X}}, \{\Phi_1, \dots, \Phi_d\}, \{1, \dots, d\}, \emptyset, \mathcal{P})$ 
7:   for  $i = 1:d$  do
8:      $\hat{\mathbf{U}}_i = \text{QR}(\mathbf{Y}_{(i)}^{(i)})$            $\triangleright$  Serial QR decomposition of the mode  $i$  unfolding of  $\mathbf{y}^{(i)}$ 
9:   end for
10:   $\hat{\mathbf{G}} = \text{IS-MTTM}(\bar{\mathcal{X}}, \emptyset, \{\hat{\mathbf{U}}_1^\top, \dots, \hat{\mathbf{U}}_d^\top\}, \mathcal{P})$ 
11:   $[\mathcal{G}, \{\mathbf{V}_j\}] = \text{STHOSVD}(\hat{\mathbf{G}}, \mathbf{r})$ 
12:  for  $i = 1, \dots, d$  do
13:     $\mathbf{U}_i = \hat{\mathbf{U}}_i \mathbf{V}_i$            $\triangleright$  Computed with local matrix multiplication
14:  end for
15: end function

```

---

**6.4. Comparison to Previous Work.** We compare our parallel algorithms with previous approaches, namely the parallel STHOSVD algorithm in [3] and the approach from [12]. Assume that the input tensor is a  $d$ -way tensor  $\mathcal{X} \in \mathbb{R}^{n \times \dots \times n}$  with rank  $(r, r, \dots, r)$  and that each mode of the  $d$ -way processor tensor has size  $q$ . We also assume that  $s < r < l \ll n$  where  $s$  is the subrank for each mode and  $\ell = r + p$  with  $p$  the oversampling parameter. Here  $s = \ell^{1/(d-1)} \approx r^{1/(d-1)}$ . In [12], Choi et al. proposed a data distribution scheme and a tensor matricization strategy that reduces the communication costs of the Gram and TTM kernels. More specifically, in this new method, before the Gram computation, communication is performed every other mode to redistribute the tensor unfolding from a 2D distribution to a block-column 1D distribution, which can avoid the communication cost of the later TTM operations required to form the core tensor. This method is shown to achieve speedup

over the parallel implementation proposed in [3]. However, these optimizations are not suitable for our randomized algorithms for the following reason. Instead of computing the Gram matrix of each mode unfolding, we compute each mode sketch with a multi-TTM. Since each sketch is much smaller at the end of the multi-TTM, it is beneficial to delay communicating the sketches as much as possible before all multi-TTM's are completed. However, the redistribution proposed by [12] requires every processor to communicate the entire uncompressed local tensor before the computation. While [12] also uses randomization to reduce computation, their focus is on reducing the cost of computing the eigenvectors of the Gram matrix, which is achieved by using a modified randomized SVD to replace the eigendecomposition.

To illustrate the benefit of using Kronecker-structured random matrices, we also compare Alg. 6.4 and Alg. 6.5 with the parallel version of Alg. 4.2, which uses dense Gaussian random matrices. The parallelized Alg. 4.2 is very similar to Alg. 6.4 with the only difference being that we use the parallel multi-TTM to apply the Kronecker-structured random matrices to the input tensor while in parallelized Alg. 4.2 we use a single parallel TTM operation to apply each of the dense Gaussian random matrices.

Algorithm	Form $\{U\}$ comp cost	Form $\{U\}$ comm cost	Form $\mathcal{G}$ comp cost	Form $\mathcal{G}$ comm cost
STHOSVD [3]	$\frac{n^{d+1}}{P}$	$\alpha\mathcal{O}(dP^{1/d}) + \beta\mathcal{O}(\frac{n^d}{P})$	$2\frac{rn^d}{P}$	$\alpha\mathcal{O}(\log P) + \beta\mathcal{O}(\frac{rn^{d-1}}{P^{1-1/d}})$
[12]	$\frac{n^{d+1}}{P}$	$\alpha\mathcal{O}(dP) + \beta\mathcal{O}(\frac{n^d}{P})$	$2\frac{rn^d}{P}$	-
Alg. 6.4	$2\frac{r^{1/(d-1)}n^d}{P}$	$\alpha\mathcal{O}(d\log P) + \beta\mathcal{O}(\frac{drn}{P^{1/d}})$	$2\frac{rn^d}{P}$	$\alpha\mathcal{O}(\log P) + \beta\mathcal{O}(\frac{rn^{d-1}}{P^{1-1/d}})$
Alg. 6.5	$4\frac{r^{1/(d-1)}n^d}{P}$	$\alpha\mathcal{O}(d\log P) + \beta\mathcal{O}(\frac{drn}{P^{1/d}})$	$2\frac{rn^d}{P}$	$\alpha\mathcal{O}(\log P) + \beta\mathcal{O}(\frac{rn^{d-1}}{P^{1-1/d}})$

Table 3: Comparison of computation and communication cost per processor. More details can be found in Appendix G.

**7. Experimental Results.** We now demonstrate the numerical benefits of our algorithms by considering the accuracy of our sequential algorithms in § 7.1, the performance of the multi-TTM and dimension tree optimizations in § 7.2 and 7.3, respectively, and the performance of the parallel implementations of our randomized algorithms on synthetic data in § 7.4 and on two real datasets in § 7.5 and 7.6.

*Computing platform.* The results shown in § 7.1 are generated by running MATLAB implementations of the sequential algorithms on a single node server. The experiments shown from § 7.2 to § 7.6 are run on the Andes cluster at Oak Ridge Leadership Computing Facility. The system consists of 704 compute nodes with 2 AMD EPYC 7302 16-core CPU's and 256 GB of RAM. We directly call OpenBLAS and the Netlib implementation of LAPACK for local linear algebra kernels, which are the only available libraries on Andes.

**7.1. Accuracy Results.** We present an experiment on a synthetic tensor that demonstrates the accuracy of our algorithms compared to existing deterministic and randomized algorithms. In this experiment, we use both SRHT and Gaussian random matrices to compare numerical accuracy to the theoretical results we derived in § 5.

We construct a synthetic 3-way tensor  $\mathfrak{X} \in \mathbb{R}^{500 \times 500 \times 500}$  by forming a (super-)diagonal tensor with geometrically decreasing entries and multiplying that tensor by a random orthogonal matrix along each mode. We set the largest entry of the original tensor to be 1 and choose the rate at which the core entries decrease to be 0.4 so that the 40th entry is approximately machine precision. In our experiment, we compress

this tensor to rank  $(10, 10, 10)$  using an oversampling parameter of  $p = 5$ . We show boxplots of the relative error over 100 trials in [Figure 3](#), comparing results from all our algorithms (Algs. 4.1 to 4.4) using Gaussian random matrices as well as our principal algorithms (Algs. 4.3 and 4.4) using SRHT random matrices. We also compare our truncation strategy to the strategy in [11], and compare the relative error from all randomized algorithms to the relative error obtained from deterministic STHOSVD, Alg. 2.2.

In Figure 3, we see that the relative errors for all our randomized algorithms deviate from the deterministic relative error by at most 10% for the given rank (the medians are within 1%). Also, the relative errors for each trial are very close together with each algorithm, as even the outliers are within the same order of magnitude, with a standard deviation of  $1.9 \times 10^{-5}$ . Regardless of random matrix distribution or whether we reuse  $\Phi_j$  matrices or generate new matrices for each mode, we do not lose significant accuracy compared to either the deterministic or standard randomized approaches. We can also see that the truncation strategy we employ in Algs. 4.1 to 4.4 is much more consistently accurate than the strategy in [11].

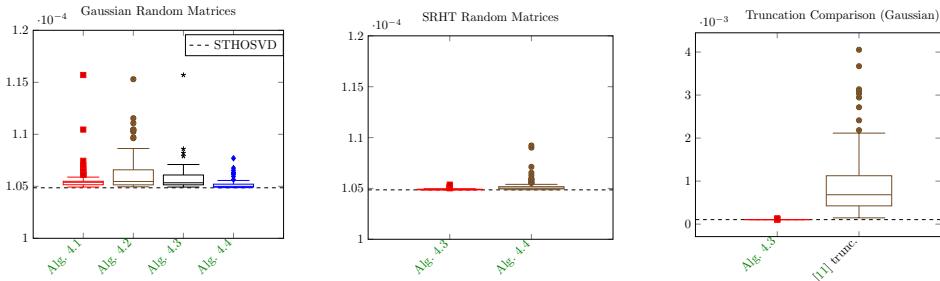


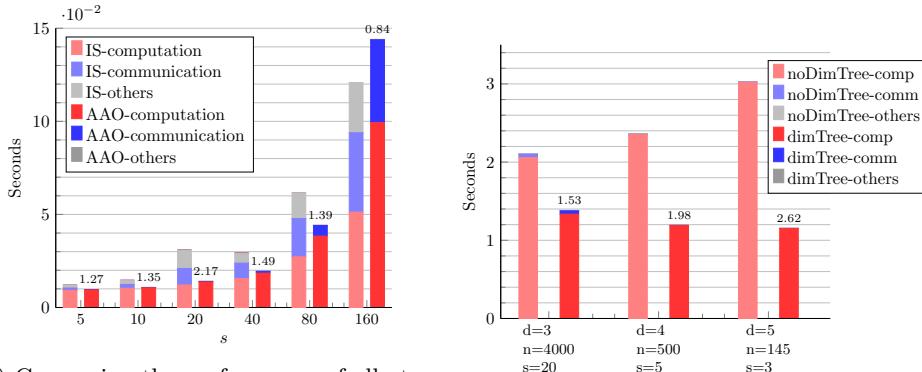
Fig. 3: Boxplots of relative errors for our randomized algorithms using Gaussian random matrices (left) and SRHT random matrices (center) compared to the relative error for STHOSVD for the synthetic tensor with geometrically decreasing values. We also compare the truncation methods we use to those used in [11] (right).

We show an additional accuracy experiment in Appendix F. Overall, we see comparable relative error for Algs. 4.3 and 4.4 to both deterministic STHOSVD and existing randomized approaches. While our theoretical results (Theorem 5.5) apply only to SRHT matrices, these experimental results suggest that Gaussian matrices perform comparably in terms of accuracy.

**7.2. In-Sequence TTM vs All-At-Once TTM.** To compare the performance of the two TTM approaches we discuss in § 6.1, we conduct an experiment performing an in-sequence multi-TTM and an all-at-once multi-TTM of a 3-way tensor  $\mathcal{X} \in \mathbb{R}^{800 \times 800 \times 800}$  with matrices  $\mathbf{U}, \mathbf{V}, \mathbf{W} \in \mathbb{R}^{s \times 800}$ , with varying  $s$ . For this experiment, we use 64 cores (2 nodes) arranged in a  $4 \times 4 \times 4$  processor grid. The results of this experiment are shown in Figure 4a, where we can see that when the matrices have relatively few rows (when  $sq \ll n$ ), the all-at-once multi-TTM is much more communication-efficient. We observe speedups ranging from 27% to over 2 $\times$ . This fits our prediction in § 6.1. However, when  $\frac{n}{sq}$  is very large (i.e. the  $s = 5$  case), both algorithms are cheap and communication is not as dominant. When  $\frac{n}{sq}$  is small, all-at-once multi-TTM has increased computational cost, and also loses its advantage in communication cost. In this case, in-sequence multi-TTM is preferred to all-at-once multi-TTM. For this reason, we think the all-at-once optimization is more suitable for the sketching phase where the random matrices tend to have very few rows.

Note that the gray bars in Figure 4a represent overhead cost. For in-sequence multi-TTM, this overhead mainly comes from reorganizing the data in memory before and after communication (MPI collectives). Since all-at-once multi-TTM avoids those communication steps, it also avoids those reorganizing costs. For higher dimensions, the benefits of all-at-once multi-TTM still depend on  $\frac{n}{sq}$  being large. If this ratio is fixed, all-at-once multi-TTM will continue to outperform in-sequence multi-TTM.

**7.3. Dimension Tree Optimization.** To demonstrate the benefits of using dimension trees, we run Alg. 4.4 with and without dimension trees on synthetic tensors with an increasing number of modes such that the total size of the input tensor and its rank are kept close to constant. We benchmark the time it takes for both methods to apply the random matrices  $\{\Phi\}$  to the input tensor and present the results in Figure 4b. This computation corresponds to Line 5 of Alg. 4.4. Since the communication and overhead costs are low, we see that the practical speedup from using dimension trees aligns closely with the theoretical prediction, with a computational reduction of  $d/2$  as described in § 4.4.



(a) Comparing the performance of all-at-once multi-TTM and in-sequence multi-TTM by multiplying a  $800 \times 800 \times 800$  synthetic tensor with an  $s \times 800$  matrix on all modes ( $s \in [5, 160]$ ). The processor grid used is  $4 \times 4 \times 4$ . The labels on top of bars indicate the overall speedup achieved by all-at-once multi-TTM compared to in-sequence multi-TTM.

(b) Performance gain by using the dimension tree optimization for Alg. 4.4. We compute the sketch of a cubic synthetic  $n$ -way tensor where each mode has the same size  $n$ , resulting in a cubic  $d$ -way core tensor with each mode having the same size  $s$ . The labels on top of the bars show speedup gained by using dimension trees.

Fig. 4: Performance benefits of our multi-TTM and dimension tree optimizations

**7.4. Strong scaling on synthetic data.** In this experiment we benchmark four variations of our randomized algorithms and STHOSVD as a baseline, scaling from 2 nodes (64 cores) to 32 nodes (1024 cores) on a fixed problem size. The input tensor is a  $410 \times 410 \times 410 \times 410$  single-precision synthetic tensor ( $\approx 113$  GB) constructed from multiplying a  $20 \times 20 \times 20 \times 20$  randomly generated core with four  $410 \times 20$  random matrices. No noise is added to this synthetic tensor so it is exactly low rank. This size is close to the largest tensor we can fit in the memory of 2 nodes, which makes the timing results more consistent and less influenced by noise in the system. Any order of modes used to compute the multi-TTM will not affect the performance as the tensor size and rank are the same across modes. All six algorithms are given target ranks (20, 20, 20, 20) and the randomized ones use oversampling parameter  $p = 3$ .

The results are presented in Figure 5, where we see that all six algorithms presented scale well to 1024 cores. All randomized algorithms outperform the determinis-

tic STHOSVD, and the randomized algorithms that use Kronecker-structured random matrices outperform the parallel version of [Alg. 4.2](#). A noticeable speedup of 3–4× is achieved by using [Alg. 6.4](#) compared to STHOSVD. The second-best algorithm is [Alg. 6.5](#) with the dimension tree optimization achieving 2–3× speedup. Note that the use of all-at-once multi-TTM provided little improvement over in-sequence multi-TTM in this case, because the compression ratio is very high so the multi-TTMs are not bottlenecks. We can also see that the dimension tree optimization does provide noticeable and consistent speedup for [Alg. 6.5](#).

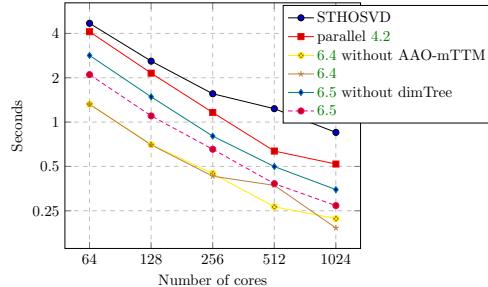


Fig. 5: Strong scaling of different algorithm variants

**7.5. Miranda dataset.** The Miranda dataset [6, 32] contains three-dimensional simulation data of the density ratios of a non-reacting flow of viscous/diffusive fluids. This dataset is  $3072 \times 3072 \times 3072$ . Its values are in single-precision and range between 1 and 3. We show visualizations for this dataset in [Appendix H.1](#).

In this experiment, we compare five variations of our randomized algorithms as well as the deterministic STHOSVD algorithm using the Miranda dataset. We use 4 nodes (128 cores) organized as a  $1 \times 8 \times 16$  tensor. The target rank we choose for this run is (502, 504, 361), which corresponds to a  $10^{-2}$  reconstruction error (estimated using pre-computed singular values of the unfolding of the data tensor for each mode).

The subrank matrix we used for [Alg. 6.4](#) is  $\begin{bmatrix} 1 & 39 & 13 \\ 30 & 1 & 17 \\ 6 & 61 & 1 \end{bmatrix}$ . For [Alg. 6.5](#), the subrank

vector we used is (20, 20, 26). The relative error achieved by STHOSVD is 0.0094, while the relative errors of the randomized algorithms [Algs. 4.2, 6.4](#) and [6.5](#) are 0.0234, 0.0194, and 0.0189, respectively, which are all within 2.5× of the deterministic error.

The performance results are recorded in [Figure 6a](#), and we visualize reconstructed tensors from [Algs. 2.2, 4.2](#) and [6.5](#) in [Appendix H.1](#). First, we note that STHOSVD is particularly slow when compared to the randomized algorithms, in this case partly because the Miranda dataset has fewer modes and each mode is large. As a result, the Gram matrices are large and the eigendecompositions of those Gram matrices are expensive. Moreover, the eigendecompositions are carried out redundantly on every processor due to the TuckerMPI assumption of small individual mode dimensions. (More details on the parallel implementation of STHOSVD can be found in [3]). The randomized algorithms, on the other hand, can avoid this expensive step completely, and we see up to a 16× speedup, comparing [Alg. 6.4](#) to STHOSVD. In the next section, we compare the performance of all the algorithms again with a higher-order tensor where each mode is relatively small. In that case, the Gram matrices are smaller and the eigendecompositions are cheaper, so the deterministic algorithm appears more competitive.

Also note that random number generation (forming the random matrices  $\{\Omega\}$ ) in

[Alg. 4.2](#) takes up a large percentage of the total time. These results demonstrate the benefits of generating fewer random numbers by using Kronecker-structured random matrices. Now, comparing the computation cost (red bar) of [Alg. 4.2](#) with that of the following algorithms to the right, we can see that using Kronecker-structured random matrices further reduces the computation cost of forming the factor matrices as we have predicted. Among the algorithms that use Kronecker-structured random matrices, [Alg. 6.4](#) and [Alg. 6.5](#) achieve the best performance. Comparing multi-TTM methods, we see that using either IS-mTTM or AAO-mTTM in [Alg. 6.4](#) results in very similar performance. Although AAO-mTTM achieved a  $3\times$  speedup in the communication cost of forming the factor matrices (pink bar) over IS-mTTM, the absolute speedup is not significant because the communication cost using IS-mTTM is already very small. This is mostly due to the subranks being very small compared to the size of the input tensor. Finally, we note that with our optimizations, applying the factor matrices to truncate the input tensor (the blue and light blue bars) now becomes the bottleneck of the algorithm.

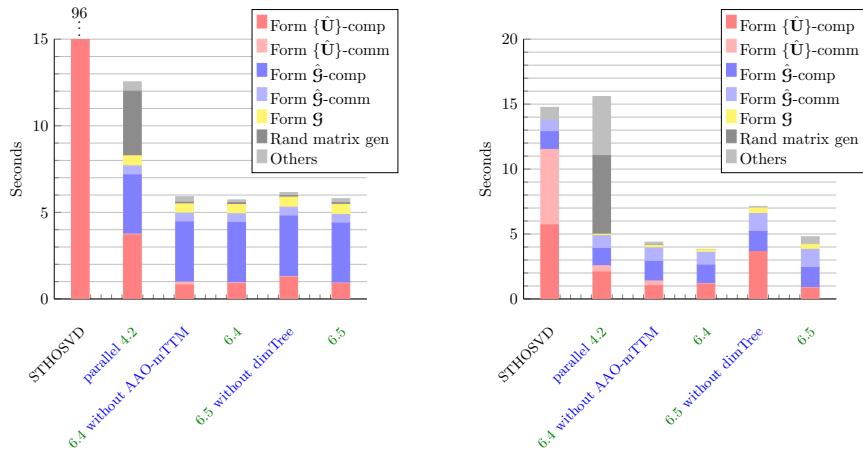
**7.6. Stats-Planar dataset.** The SP dataset is generated from the simulation of a statistically stationary planar (SP) methane-air flame [23]. The data has dimensions  $500 \times 500 \times 500 \times 11 \times 400$ , with the first 3 modes representing a 3D spatial grid, the 4th mode representing 11 variables, and the 5th mode representing time steps. This data has been used in previous studies such as [3] to demonstrate the effectiveness of Tucker decomposition algorithms. In this work, we use the single-precision-max-normalized version of this dataset. We visualize the 250th slice for each of the first three modes of the SP tensor in [Appendix H.1](#).

Similarly to the experiment on the Miranda dataset, we compare five variations of the randomized algorithm and the deterministic STHOSVD algorithm using 1024 cores (32 nodes). The target rank we use is  $(31, 38, 35, 6, 11)$ , which is the rank returned by the STHOSVD algorithm satisfying a  $10^{-2}$  error tolerance. The subrank vector we used for [Alg. 6.5](#) is  $[2 \ 2 \ 2 \ 3 \ 4]$  and the subrank matrix used for [Alg. 6.4](#) is

$$\begin{bmatrix} 0 & 3 & 3 & 2 & 2 \\ 3 & 0 & 4 & 2 & 2 \\ 2 & 5 & 0 & 2 & 2 \\ 2 & 2 & 2 & 0 & 2 \\ 2 & 2 & 2 & 2 & 0 \end{bmatrix}.$$

The relative error achieved by STHOSVD on this dataset is 0.0028, while the relative errors of the randomized algorithms [Algs. 4.2, 6.4](#) and [6.5](#) are 0.0050, 0.0079, and 0.0079, respectively, which are all within  $3\times$  of the deterministic error.

The performance results are shown in [Figure 6b](#). The speedup of the randomized algorithms appears less dramatic compared to the results from the Miranda dataset, which is mainly due to the difference in dimensions of these two datasets. Recall that compared to the Miranda dataset, the SP tensor is of higher order but has a smaller size in each mode. As a result, the sequential eigendecomposition in the STHOSVD algorithm is no longer as expensive. We also see that [Algs. 6.4](#) and [6.5](#) are still the best-performing algorithms. [Alg. 4.2](#) suffers from slow random number generation, similar to the experiments on the Miranda dataset. Finally, forming the factor matrices is no longer a bottleneck for tensors with more cubical dimensions. In this case, accelerating the computations applying these factor matrices to derive the core tensor will become a more important issue.



(a) Comparing the performance of all algorithms on Miranda dataset      (b) Comparing the performance of all algorithms on SP dataset

Fig. 6: Performance breakdown of all algorithms on our two real datasets

**8. Conclusions and Future Work.** We develop new randomized algorithms using a Kronecker product of random matrices that significantly decrease the computational cost in computing a Tucker decomposition. By accelerating the sketching step using Kronecker products, we remove the SVD as the dominant computational bottleneck. Our algorithms also reduce the number of random entries generated, which, as shown in our experiment results, could result in significant savings in the runtime compared to standard randomized algorithms. As the SVD step is no longer the dominant computation, future directions include accelerating the TTM computation, the other dominant portion of computing a Tucker decomposition, perhaps through a one-pass approach similar to [27]. We develop probabilistic error bounds for our algorithms using SRHT matrices as they generalize to Kronecker products better than Gaussian matrices. The empirical results comparing Gaussian and SRHT matrices show that the error incurred from using SRHT matrices is not any worse than using Gaussian matrices. Our theoretical bounds are pessimistic in comparison, so there is room for improvement in the analysis, another potential future direction. We implement our new randomized algorithms in parallel, developing a new algorithm that parallelizes the most expensive SVD component. Previous approaches such as [12] parallelize other components, leaving the most expensive part to be computed locally. Overall, we show in this work that choosing a random matrix that fits the structure of our problem is beneficial. The dense Gaussian matrix typically used in RandSVD in particular is not required, and performance is greatly improved by exploiting appropriate structure.

**Appendix A. Additional Algorithm: Randomized HOSVD with Kronecker product.** We now include the case of Alg. 4.4 where we generate an independent set of Kronecker factors for each mode.

#### Appendix B. Detailed Complexity Analysis for Algs. 4.3 and 4.4.

*Cost of Alg. 4.3.* First consider Alg. 4.3 (rSTHKron). We form the sketch  $\mathbf{Y}$  for mode  $j$  in Line 6, involving an intermediate core tensor as well as random matrices  $\{\Phi_k\}_{k \neq j}^d$ . The intermediate core at iteration  $j$  has dimensions

---

**Algorithm A.1** Randomized HOSVD with Kronecker product

---

```

1: function  $[\mathcal{G}, \{\mathbf{U}_j\}] = \text{RHKRON}(\mathbf{X}, \mathbf{r}, p)$ 
2:   Compute matrix of subranks  $\mathbf{S}$ 
3:   for  $j = 1 : d$  do
4:     Draw  $d - 1$  random matrices  $\Phi_{j,k} \in \mathbb{R}^{s_{j,k} \times n_k}$  for  $k = 1, \dots, j-1, j+1, \dots, d$ 
5:      $\mathbf{Y} \leftarrow \mathbf{X} \times_1 \Phi_{j,1} \times \cdots \times_{j-1} \Phi_{j,j-1} \times_{j+1} \Phi_{j,j+1} \times \cdots \times_d \Phi_{j,d}$ 
6:     Compute thin QR  $\mathbf{Y}_{(j)} = \hat{\mathbf{U}}_j \mathbf{R}$ 
7:   end for
8:    $\hat{\mathbf{G}} = \mathbf{X} \times_1 \hat{\mathbf{U}}_1^\top \times \cdots \times_d \hat{\mathbf{U}}_d^\top$ 
9:    $[\mathcal{G}, \{\mathbf{V}_j\}] = \text{STHOSVD}(\hat{\mathbf{G}}, \mathbf{r})$ 
10:   $\mathbf{U}_j = \hat{\mathbf{U}}_j \mathbf{V}_j$  for  $j = 1, \dots, d$ 
11: end function

```

---

$$(B.1) \quad \underbrace{\ell \times \cdots \times \ell}_{j-1} \times \underbrace{n \times \cdots \times n}_{d-j+1},$$

and the matrices  $\{\Phi_k\}_{k=1}^{j-1}$  have dimensions  $s \times \ell$ , while the matrices  $\{\Phi_k\}_{k=j+1}^d$  have dimensions  $n \times \ell$ . We can compute the multi-TTM with any order of modes, but the most efficient in this particular case is to start with a mode in which the dimension is  $n$ . Without loss of generality, we thus compute the product in reverse order of modes. Then, we have two separate cases corresponding to the different dimensions of the  $\Phi$  matrices. For modes  $d$  to  $j+1$ , the cost is  $\sum_{i=1}^{d-j+1} 2s^i \ell^{j-i} n^{d+2-i-j}$ . The cost for modes  $j-1$  to 1 is  $\sum_{i=1}^{j-2} 2\ell^{j-i} s^{d-j+i+1}$ .

In Line 8, we perform a single TTM to compute the intermediate core. In mode  $j$ , we compute a TTM with core  $\hat{\mathbf{G}}$  with the same dimensions as in (B.1), and factor matrix  $\hat{\mathbf{U}}_j^\top$  with dimensions  $n \times \ell$ . The cost of this product is  $2\ell^j n^{d-j+1}$ . Putting all modes together, the dominant costs of Alg. 4.3 are

$$2 \sum_{j=1}^d \left( \ell^j n^{d-j+1} + \sum_{i=1}^{d-j+1} s^i \ell^{j-i} n^{d+2-i-j} + \sum_{i=1}^{j-2} \ell^{j-i} s^{d-j+i+1} \right).$$

*Cost of Alg. 4.4.* Now considering Alg. 4.4, the sketch is computed in Line 5 of the full tensor  $\mathbf{X} \in \mathbb{R}^{n \times n \times \cdots \times n}$  and random matrices  $\Phi_j \in \mathbb{R}^{s \times n}$  for  $j = 1, \dots, d$ . For any mode, the cost is  $\sum_{i=1}^{d-1} 2s^i n^{d-i+1}$ . Then without the dimension tree optimization, the cost for all modes is  $d \sum_{i=1}^{d-1} 2s^i n^{d-i+1}$ . Incorporating dimension trees as discussed in § 4.4 reduces the constant  $2d$  to 4, so the cost of Line 5 in Alg. 4.4 for all modes is  $4 \sum_{i=1}^{d-1} s^i n^{d+1-i}$ .

After all the modes are processed, we compute the core via a multi-TTM in Line 8 in Alg. 4.4 of tensor  $\mathbf{X} \in \mathbb{R}^{n \times n \times \cdots \times n}$  and factor matrices  $\hat{\mathbf{U}}_j$  for  $j = 1, \dots, d$ . The cost of this multi-TTM, computing in any order of modes, is  $\sum_{j=1}^d 2\ell^j n^{d+1-j}$ . Combining all the costs, including  $d$  sketch computations and core computations, the complexity of Alg. 4.4 is

$$4 \sum_{i=1}^{d-1} s^i n^{d+1-i} + 2 \sum_{j=1}^d \ell^j n^{d+1-j}.$$

Note that we do not include the core truncation step in Line 10 of Alg. 4.3 or Line 9 of Alg. 4.4 in the dominant costs as the leading order term in this cost is only  $\ell^{d+1}$  for all algorithms. We also ignore the cost of the thin QR factorizations in Line 6 in Alg. 4.4 and Line 7 in Alg. 4.3 as that cost is  $2d\ell^2 n$  in all three algorithms.

## Appendix C. Bounds on Core Singular Values.

LEMMA C.1. Let tensors  $\mathbf{X} \in \mathbb{R}^{n_1 \times \dots \times n_d}$ ,  $\mathbf{Y} \in \mathbb{R}^{\ell_1 \times \dots \times \ell_d}$  and matrices  $\mathbf{U}_j \in \mathbb{R}^{n_j \times \ell_j}$  with orthonormal columns such that  $\mathbf{Y} = \mathbf{X} \times_1 \mathbf{U}_1^\top \times \dots \times_d \mathbf{U}_d^\top$ . Then

$$\sigma_i(\mathbf{Y}_{(j)}) \leq \sigma_i(\mathbf{X}_{(j)})$$

for each mode  $j$  and for each  $i = 1, \dots, \min\{\ell_j, \ell_j^\otimes\}$ .

*Proof.* First consider an  $m \times n$  matrix  $\mathbf{B}$  and matrix  $\mathbf{A} = \mathbf{BV}$  for  $\mathbf{V}$  with  $k$  orthonormal columns. We can show that  $\sigma_i(\mathbf{A}) \leq \sigma_i(\mathbf{B})$  for any index  $i = 1, \dots, \min\{m, k\}$ . Consider first that  $\mathbf{A}$  is a submatrix of  $\mathbf{B}[\mathbf{V} \ \mathbf{V}_\perp]$ , implying that  $\mathbf{A}^\top \mathbf{A}$  is a principal submatrix of  $\mathbf{B}^\top \mathbf{B}$ . Then we can directly apply the Cauchy interlacing theorem [18, Theorem 4.3.28] to see that  $\sigma_i(\mathbf{A}) = \lambda_i(\mathbf{A}^\top \mathbf{A}) \leq \lambda_i(\mathbf{B}^\top \mathbf{B}) = \sigma_i(\mathbf{B})$ .

The same argument applies to the transpose of  $\mathbf{A} = \mathbf{U}^\top \mathbf{B}$  for  $\mathbf{U}$  with orthonormal columns. Thus, for  $\mathbf{U}$  and  $\mathbf{V}$  with orthonormal columns and  $\mathbf{A} = \mathbf{U}^\top \mathbf{BV}$ ,  $\sigma_i(\mathbf{A}) \leq \sigma_i(\mathbf{B})$  for  $i$  ranging from 1 to the minimum of the numbers of columns of  $\mathbf{U}$  and  $\mathbf{V}$ .

Now, let  $\mathbf{A} = \mathbf{Y}_{(j)}$ , recalling that  $\mathbf{Y}_{(j)} = \mathbf{U}_j \mathbf{X}_{(j)} (\mathbf{U}_d \otimes \mathbf{U}_{d-1} \otimes \dots \otimes \mathbf{U}_{j+1} \otimes \mathbf{U}_{j-1} \otimes \dots \otimes \mathbf{U}_1)$ . Recall that the Kronecker product of matrices with orthonormal columns also has orthonormal columns. Thus,  $\sigma_i(\mathbf{Y}_{(j)}) \leq \sigma_i(\mathbf{X}_{(j)})$  for  $i = 1, \dots, \min\{\ell_j, \ell_j^\otimes\}$  and  $j = 1, \dots, d$ .  $\square$

#### Appendix D. Proof of Lemma 5.1.

*Proof.* Let  $\mathbf{W} = \mathbf{V}_1^\top \in \mathbb{R}^{r \times n}$  for notational simplicity, and recall that  $\boldsymbol{\Omega} = \mathbf{D}(\mathbf{H}_1 \otimes \dots \otimes \mathbf{H}_q)$  is the Kronecker product of independent SRHT matrices where  $\mathbf{D} \in \mathbb{R}^{n \times n}$  and  $\mathbf{H}_j \in \mathbb{R}^{n_j \times s_j}$  for every  $j$ , given  $n = \prod_{j=1}^q n_j$  and  $\ell = \prod_{j=1}^q s_j$ . Define  $\mathbf{G} = (\mathbf{W}\boldsymbol{\Omega})(\mathbf{W}\boldsymbol{\Omega})^\top \in \mathbb{R}^{r \times r}$ . Note that  $\mathbf{W}\boldsymbol{\Omega}$  is equivalent to  $\boldsymbol{\Omega}_1 \in \mathbb{R}^{r \times \ell}$ .

Our approach will focus on the elementwise representation of  $\mathbf{G}$  and will be composed of 3 main steps: first, we will express the elements of  $\mathbf{G}$  in terms of two summands  $\mathbf{M}$  and  $\mathbf{N}$  that can be bounded more easily; second, we will obtain a deterministic bound for  $\|\mathbf{M}\|_2$  and then bound  $\mathbb{E}[N_{ij}^2]$ , which is the bulk of the proof; and third, we use our result from the previous step in conjunction with Markov's inequality to obtain a concentration inequality for  $\|\mathbf{N}\|_2$ . Combining all these pieces will then give us the desired bound. Note that our three main steps follow the approach of [31], which analyzes the case where  $\boldsymbol{\Omega}$  is a single SRHT matrix.

Define  $\mathbf{H} = \mathbf{H}_1 \otimes \mathbf{H}_2 \otimes \dots \otimes \mathbf{H}_q$ , letting the Kronecker product of subsampled Hadamard matrices be  $\mathbf{H}$  for ease of notation. Then we have another way to express  $\mathbf{G}$  as

$$(D.1) \quad \mathbf{G} = (\mathbf{W}\boldsymbol{\Omega})(\mathbf{W}\boldsymbol{\Omega})^\top = \mathbf{WDHH}^\top \mathbf{DW}^\top = \mathbf{WDFD}\mathbf{W}^\top,$$

letting  $\mathbf{F} = \mathbf{HH}^\top$ . There are some important properties of  $\mathbf{D}$  and  $\mathbf{F}$  we will need, which we now explore. The diagonal matrix  $\mathbf{D}$  has i.i.d. entries drawn from the Rademacher distribution so that  $\mathbb{E}D_a = 0$  and  $D_a^2 = 1$  for  $a = 1, \dots, n$ .

Each element of  $\mathbf{F}$  can be written as a product of the entries of individual Gram matrices  $\mathbf{H}_j \mathbf{H}_j^\top$ . Specifically,  $F_{ab} = (\mathbf{H}_1 \mathbf{H}_1^\top)_{i_1 j_1} (\mathbf{H}_2 \mathbf{H}_2^\top)_{i_2 j_2} \dots (\mathbf{H}_q \mathbf{H}_q^\top)_{i_q j_q}$ , with  $a$  the linear index with respect to  $i_1, \dots, i_q$  and  $b$  the linear index with respect to  $j_1, \dots, j_q$ . This representation allows us to break dependent expressions down into their independent parts, as each  $\mathbf{H}_i$  is independent from  $\mathbf{H}_j$  when  $i \neq j$ . We can then write the expectation of  $F_{ab}$  as

$$(D.2) \quad \mathbb{E}F_{ab} = \mathbb{E}(\mathbf{H}_1 \mathbf{H}_1^\top)_{i_1 j_1} \mathbb{E}(\mathbf{H}_2 \mathbf{H}_2^\top)_{i_2 j_2} \dots \mathbb{E}(\mathbf{H}_q \mathbf{H}_q^\top)_{i_q j_q}.$$

From [31, Eqn. 38], we have that  $\mathbb{E}(\mathbf{H}_k \mathbf{H}_k^\top)_{i_k j_k} = 0$  for  $i_k \neq j_k$ . If  $a \neq b$ , then  $i_k \neq j_k$  for at least one  $k$ . Combining this and (D.2), we can say that  $\mathbb{E}F_{ab} = 0$  for  $a \neq b$ . Now

consider the case where  $a = b$ . From [31, Eqn. 37], we have that  $(\mathbf{H}_k \mathbf{H}_k^\top)_{i_k i_k} = s_k/n_k$  deterministically. Then,

$$(D.3) \quad F_{aa} = \prod_{k=1}^q \frac{s_k}{n_k} = \frac{\ell}{n}.$$

The last piece we will need is  $\mathbb{E}[F_{ab}^2]$ . From [31, Eqn. 39],  $\mathbb{E}[(\mathbf{H}_k \mathbf{H}_k^\top)_{i_k j_k}^2] = s_k/n_k^2$  for  $i_k \neq j_k$ , and  $\mathbb{E}[(\mathbf{H}_k \mathbf{H}_k^\top)_{i_k i_k}^2] = s_k^2/n_k^2$  from above. If  $a \neq b$ , then  $i_{k'} \neq j_{k'}$  for at least one  $k'$ . Combining this and (D.2),

$$(D.4) \quad \mathbb{E}[F_{ab}^2] \leq \frac{s_{k'}}{n_{k'}^2} \prod_{\substack{k=1 \\ k \neq k'}}^q \frac{s_k^2}{n_k^2} = \frac{\ell^2}{s_{k'} n^2} \leq \frac{\ell^2}{\min_k \{s_k\} n^2} = \frac{\ell^2}{s_k^* n^2},$$

letting  $s_k^* = \min_k \{s_k\}$ .

With all these pieces in mind, we begin the main steps of the proof. We start with an elementwise representation of  $\mathbf{G}$ , using the form in (D.1), with

$$\begin{aligned} G_{ij} &= \sum_{a,b=1}^n W_{ia} D_a F_{ab} D_b W_{jb} \\ &= \sum_{a=1}^n W_{ia} W_{ja} D_a^2 F_{aa} + \sum_{a=1}^n W_{ia} D_a \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab}, \end{aligned}$$

for  $1 \leq i, j \leq r$ . Consider the first term, where we have isolated the case  $a = b$ . As the rows of  $\mathbf{W}$  are orthonormal,  $D_a^2 = 1$ , and  $F_{aa} = \ell/n$ ,  $G_{ij}$  can be written as  $G_{ij} = \frac{\ell}{n} \delta_{ij} + \sum_{a=1}^n W_{ia} D_a \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab}$ , where  $\delta_{ij}$  is the Kronecker delta which is 1 when  $i = j$  and 0 otherwise. Defining  $\mathbf{M} \in \mathbb{R}^{r \times r}$  to be the diagonal matrix with  $\ell/n$  as each diagonal entry, and letting  $\mathbf{N} \in \mathbb{R}^{r \times r}$  be the matrix with entries  $N_{ij} = \sum_{a=1}^n W_{ia} D_a \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab}$ , we have  $\mathbf{G} = \mathbf{M} + \mathbf{N}$ .

As  $\mathbf{M}$  is a diagonal matrix, we can easily see  $\|\mathbf{M}\|_2 = \ell/n$ . Bounding  $\|\mathbf{N}\|$  is trickier; our approach will be to use the fact  $\mathbb{E}\|\mathbf{N}\|_2^2 \leq \mathbb{E}\|\mathbf{N}\|_F^2 = \sum_{i,j=1}^r \mathbb{E}[N_{ij}^2]$  and first bound  $\mathbb{E}[N_{ij}^2]$ . We start by expanding the product

$$\begin{aligned} (D.5) \quad \mathbb{E}[N_{ij}^2] &= \mathbb{E} \left( \sum_{a=1}^n W_{ia} D_a \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \right) \left( \sum_{c=1}^n W_{ic} D_c \sum_{\substack{f=1 \\ f \neq c}}^n W_{jf} D_f F_{cf} \right) \\ &= \mathbb{E} \sum_{a=1}^n W_{ia}^2 \left( \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \right)^2 + \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \sum_{\substack{f=1 \\ f \neq c}}^n W_{jf} D_f F_{cf}, \end{aligned}$$

which we have separated into the terms where  $a = c$  and  $a \neq c$ . Consider the first term of (D.5), where  $a = c$ . As  $\mathbf{W}$  is a deterministic matrix, the expectation only affects the terms with  $D_b$  and  $F_{ab}$ , so we have

$$(D.6) \quad \mathbb{E} \sum_{a=1}^n W_{ia}^2 \left( \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \right)^2 = \sum_{a=1}^n W_{ia}^2 \mathbb{E} \left( \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \right)^2.$$

Now consider the expectation portion of (D.6) for a fixed  $1 \leq a \leq n$ . We can expand this product and distribute the expectation as

$$\begin{aligned} \mathbb{E} \left( \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \right)^2 &= \mathbb{E} \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \sum_{\substack{f=1 \\ f \neq a}}^n W_{jf} D_f F_{af} \\ &= \sum_{\substack{b,f=1 \\ b \neq a}}^n W_{jb} W_{jf} \mathbb{E}[D_b D_f F_{ab} F_{af}] = \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb}^2 \mathbb{E}[F_{ab}^2] + \sum_{\substack{b,f=1 \\ b \neq f \\ b \neq a}}^n W_{jb} W_{jf} \mathbb{E}[D_b D_f F_{ab} F_{af}] \end{aligned}$$

where the last equality separates terms into where  $b = f$  and where  $b \neq f$ , respectively. From (D.4) and as the rows of  $\mathbf{W}$  are normalized, we can write the term where  $b = f$  as  $\sum_{\substack{b=1 \\ b \neq a}}^n W_{jb}^2 \mathbb{E}[F_{ab}^2] \leq \sum_{b=1}^n W_{jb}^2 \frac{\ell^2}{s_k^* n^2} = \frac{\ell^2}{s_k^* n^2}$ . The term where  $b \neq f$  can be written as  $\sum_{\substack{b,f=1 \\ b \neq f \\ b \neq a}}^n W_{jb} W_{jf} \mathbb{E}[D_b D_f F_{ab} F_{af}] = \sum_{\substack{b,f=1 \\ b \neq f \\ b \neq a}}^n W_{jb} W_{jf} \mathbb{E} D_b \mathbb{E} D_f \mathbb{E}[F_{ab} F_{af}] = 0$  from  $\mathbb{E} D_b = 0$ .

These two results can be combined into the expectation portion of (D.6) to obtain  $\sum_{a=1}^n W_{ia}^2 \mathbb{E} \left( \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \right)^2 \leq \sum_{a=1}^n W_{ia}^2 \frac{\ell^2}{s_k^* n^2} = \frac{\ell^2}{s_k^* n^2}$ .

We now focus on the second term of (D.5), where  $a \neq c$ . We split up both of the last two sums to extract the terms where  $b = c$  and where  $f = a$ , giving

$$\begin{aligned} \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c \sum_{\substack{b=1 \\ b \neq a}}^n W_{jb} D_b F_{ab} \sum_{\substack{f=1 \\ f \neq c}}^n W_{jf} D_f F_{cf} &= \\ \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c \left( W_{jc} D_c F_{ac} + \sum_{\substack{b=1 \\ a \neq b \neq c}}^n W_{jb} D_b F_{ab} \right) \left( W_{ja} D_a F_{ca} + \sum_{\substack{f=1 \\ a \neq f \neq c}}^n W_{jf} D_f F_{cf} \right). \end{aligned}$$

We expand this product into four terms we can bound separately, as

$$\begin{aligned} (D.7a) \quad &\mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c \left( W_{jc} D_c F_{ac} + \sum_{\substack{b=1 \\ a \neq b \neq c}}^n W_{jb} D_b F_{ab} \right) \left( W_{ja} D_a F_{ca} + \sum_{\substack{f=1 \\ a \neq f \neq c}}^n W_{jf} D_f F_{cf} \right) \\ &= \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c W_{jc} W_{ja} D_c D_a F_{ac} F_{ca} \end{aligned}$$

$$(D.7b) \quad + \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c \sum_{\substack{b=1 \\ a \neq b \neq c}}^n W_{jb} D_b F_{ab} \sum_{\substack{f=1 \\ a \neq f \neq c}}^n W_{jf} D_f F_{cf}$$

$$(D.7c) \quad + \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c W_{jc} D_c F_{ac} \sum_{\substack{f=1 \\ a \neq f \neq c}}^n W_{jf} D_f F_{cf}$$

$$(D.7d) \quad + \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c W_{ja} D_a F_{ca} \sum_{\substack{b=1 \\ a \neq b \neq c}}^n W_{jb} D_b F_{ab}.$$

Consider (D.7a). As  $D_k^2 = 1$  and  $\mathbf{F}$  is symmetric, the expectation is just affected by

$F_{ac}^2$ . We then have

$$\begin{aligned} \mathbb{E} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} D_a D_c W_{jc} W_{ja} D_c D_a F_{ac} F_{ca} &= \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} W_{jc} W_{ja} \mathbb{E}[F_{ac}^2] \\ &\leq \frac{\ell^2}{s_k^* n^2} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} W_{jc} W_{ja} \leq \frac{\ell^2}{s_k^* n^2} \sum_{\substack{a,c=1 \\ a \neq c}}^n W_{ia} W_{ic} W_{jc} W_{ja} \\ &= \frac{\ell^2}{s_k^* n^2} \left( \sum_{a=1}^n W_{ia} W_{ja} \right)^2 = \frac{\ell^2}{s_k^* n^2} [\mathbf{W} \mathbf{W}^\top]_{ij}^2 = \delta_{ij} \frac{\ell^2}{s_k^* n^2}. \end{aligned}$$

In all three of the remaining parts, (D.7b), (D.7c), and (D.7d), distributing the expectation to the independent random components gives us the expectation of the product of independent Rademacher entries. This means all three of these parts are equal to 0.

With all these pieces, we now have  $\mathbb{E}[N_{ij}^2] \leq \frac{\ell^2}{s_k^* n^2} + \delta_{ij} \frac{\ell^2}{s_k^* n^2}$ . With the bound on  $\mathbb{E}[N_{ij}^2]$ , we can now bound the expectation of the norm of  $\mathbf{N}$ :  $\mathbb{E}\|\mathbf{N}\|_2^2 \leq \mathbb{E}\|\mathbf{N}\|_F^2 = \mathbb{E} \sum_{i,j=1}^r N_{ij}^2 \leq \frac{\ell^2}{s_k^* n^2} \sum_{i,j=1}^r (1 + \delta_{ij}) = (r^2 + r) \frac{\ell^2}{s_k^* n^2}$ . Then, using Markov's inequality,  $\|\mathbf{N}\|_2^2 \leq \frac{\beta(r^2+r)\ell^2}{s_k^* n^2}$  with probability at least  $1 - \frac{1}{\beta^2}$ .

Now consider the term  $\|\mathbf{G}^\dagger\|_2$ . Recalling that  $\mathbf{G} = \mathbf{M} + \mathbf{N}$ , we can express this instead as  $\mathbf{G} = (\mathbf{I} + \mathbf{NM}^{-1})\mathbf{M}$ . Then we can write  $\mathbf{G}^\dagger = \mathbf{M}^{-1}(\mathbf{I} + \mathbf{NM}^{-1})^\dagger$ . Taking norms, we have  $\|\mathbf{G}^\dagger\|_2 \leq \|\mathbf{M}^{-1}\|_2 \|\mathbf{I} + \mathbf{NM}^{-1}\|_2 \leq \frac{n}{\ell} \sum_{k=0}^{\infty} \|\mathbf{NM}^{-1}\|_2^k$ , where we use the Taylor expansion  $(\mathbf{I} + \mathbf{NM}^{-1})^\dagger = \sum_{k=0}^{\infty} (-\mathbf{NM}^{-1})^k$  (see [18, Corollary 5.6.16] for more details). We can then write  $\|\mathbf{G}^\dagger\|_2 \leq \frac{n}{\ell} \sum_{k=0}^{\infty} (\|\mathbf{N}\|_2 \|\mathbf{M}^{-1}\|_2)^k$ . We now consider  $\|\mathbf{N}\|_2 \|\mathbf{M}^{-1}\|_2$  before the entire expression. As  $\|\mathbf{M}^{-1}\|_2 = n/\ell$ ,  $\|\mathbf{N}\|_2 \|\mathbf{M}^{-1}\|_2 \leq \sqrt{\frac{\beta(r^2+r)}{s_k^*}} \leq 1 - \frac{1}{\alpha}$ , with probability at least  $1 - \frac{1}{\beta^2}$ , where the last inequality comes from (5.5). Then,  $\|\mathbf{G}^\dagger\|_2 \leq \frac{n}{\ell} \sum_{k=0}^{\infty} (1 - \frac{1}{\alpha})^k = \frac{n\alpha}{\ell}$ . Our smallest singular value is then  $\frac{1}{\sigma_{\min}^2(\mathbf{W}\Omega)} = \|\mathbf{G}^\dagger\|_2 \leq \frac{\alpha n}{\ell}$ , with probability at least  $1 - \frac{1}{\beta^2}$ . Taking the square root, we obtain the desired result.  $\square$

## Appendix E. STHOSVD Error Analysis.

**THEOREM E.1.** Let  $\mathcal{T} = [\mathcal{G}; \mathbf{U}_1, \dots, \mathbf{U}_d]$  be the approximation given by Alg. 4.3 to  $\mathfrak{X} \in \mathbb{R}^{n_1 \times \dots \times n_d}$  with target rank  $\mathbf{r} = (r_1, \dots, r_d)$  and oversampling parameter  $p$ . Let  $\ell_j = r_j + p$  for  $j = 1, \dots, d$ . Then for sequences  $\{\alpha_j\}_{j=1}^d$  and  $\{\beta_j\}_{j=1}^d$  satisfying (5.5), the following bound holds with probability at least  $1 - \sum_{j=1}^d \frac{1}{\beta_j^2}$ ,

$$\|\mathfrak{X} - \mathcal{T}\| \leq \left( \sum_{j=1}^d \left( 1 + \frac{\alpha_j n_j^\emptyset}{\ell_j} \right) \sum_{i=r_j+1}^{\ell_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2} + \left( \sum_{j=1}^d \sum_{i=r_j+1}^{\ell_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2}.$$

*Proof.* The quantity  $\varepsilon_{\text{core}}$  is bounded from (5.1), so we only consider  $\varepsilon_{\text{rand}}$ . Let  $\hat{\mathcal{G}}^{(j)} = \mathfrak{X} \times_1 \hat{\mathbf{U}}_1^\top \times_2 \dots \times_j \hat{\mathbf{U}}_j^\top$  be the partially truncated core tensor after processing mode  $j$ , and let  $\hat{\mathcal{T}}^{(j)} = \hat{\mathcal{G}}^{(j)} \times_1 \hat{\mathbf{U}}_1 \times_2 \dots \times_j \hat{\mathbf{U}}_j$  the resulting partial approximation

to  $\mathfrak{X}$ . From the first equality of [Lemma 5.4](#), we have

$$\begin{aligned}\|\mathfrak{X} - \hat{\mathfrak{T}}\|^2 &= \|\mathfrak{X} - \mathfrak{X} \times_1 \hat{\mathbf{U}}_1 \hat{\mathbf{U}}_1^\top \times_2 \cdots \times_d \hat{\mathbf{U}}_d \hat{\mathbf{U}}_d^\top\|^2 \\ &= \sum_{j=1}^d \|\hat{\mathfrak{T}}^{(j-1)} - \hat{\mathfrak{T}}^{(j)}\|^2 \\ &= \sum_{j=1}^d \|\hat{\mathfrak{G}}^{(j-1)} \times_1 \hat{\mathbf{U}}_1 \times_2 \cdots \times_{j-1} \hat{\mathbf{U}}_{j-1} \times_j (\mathbf{I} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top)\|^2.\end{aligned}$$

Consider the  $j$ -th term in the sum. We unfold to obtain

$$\begin{aligned}\|\hat{\mathfrak{T}}^{(j-1)} - \hat{\mathfrak{T}}^{(j)}\|^2 &= \|(\mathbf{I} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top) \hat{\mathbf{G}}_{(j)}^{(j-1)} (\underbrace{\mathbf{I} \otimes \cdots \otimes \mathbf{I}}_{d-j} \otimes \hat{\mathbf{U}}_{j-1} \otimes \cdots \otimes \hat{\mathbf{U}}_1)^\top\|_F^2 \\ &\leq \|(\mathbf{I} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top) \hat{\mathbf{G}}_{(j)}^{(j-1)}\|^2,\end{aligned}$$

as the columns of factor matrices  $\hat{\mathbf{U}}_j$  are orthonormal for all  $j = 1, \dots, d$ . We can then apply [Theorem 5.3](#) to  $\|(\mathbf{I} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top) \hat{\mathbf{G}}_{(j)}^{(j-1)}\|$ , giving

$$\|(\mathbf{I} - \hat{\mathbf{U}}_j \hat{\mathbf{U}}_j^\top) \hat{\mathbf{G}}_{(j)}^{(j-1)}\|^2 \leq \left(1 + \frac{\alpha_j n_j^\emptyset}{\ell_j}\right) \sum_{i=r_j+1}^{n_j} \sigma_i^2(\hat{\mathbf{G}}_{(j)}^{(j-1)}),$$

except with probability at most  $\frac{1}{\beta_j^2}$ . As  $\hat{\mathfrak{G}}^{(j-1)}$  is a random quantity, we must bound this by the singular values of  $\mathfrak{X}$ . We can directly apply [Lemma C.1](#) to  $\hat{\mathfrak{G}}^{(j-1)} = \mathfrak{X} \times_1 \hat{\mathbf{U}}_1^\top \times_2 \cdots \times_j \hat{\mathbf{U}}_j^\top$  relating the singular values of  $\hat{\mathfrak{G}}^{(j-1)}$  to those of  $\mathfrak{X}$ . Thus,

$$\|\hat{\mathfrak{T}}^{(j-1)} - \hat{\mathfrak{T}}^{(j)}\|^2 \leq \left(1 + \frac{\alpha_j n_j^\emptyset}{r_j}\right) \sum_{i=\ell_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)}),$$

except with probability at most  $\frac{1}{\beta_j^2}$ . The failure probability for the entire sum is the union of all  $d$  failure probabilities for each mode, bounded above by the sum of those probabilities by the union bound. Then,

$$\begin{aligned}\|\mathfrak{X} - \hat{\mathfrak{T}}\|^2 &= \sum_{j=1}^d \|\hat{\mathfrak{T}}^{(j-1)} - \hat{\mathfrak{T}}^{(j)}\|^2 \\ &\leq \sum_{j=1}^d \left(1 + \frac{\alpha_j n_j^\emptyset}{\ell_j}\right) \sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)}),\end{aligned}$$

except with probability at most  $\sum_{j=1}^d \frac{1}{\beta_j^2}$ . Taking square roots gives  $\varepsilon_{\text{rand}}$ . Combining  $\varepsilon_{\text{rand}}$  and  $\varepsilon_{\text{core}}$ , the total error is then  $\square$

$$\|\mathfrak{X} - \mathfrak{T}\| \leq \left( \sum_{j=1}^d \left(1 + \frac{\alpha_j n_j^\emptyset}{\ell_j}\right) \sum_{i=r_j+1}^{n_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2} + \left( \sum_{j=1}^d \sum_{i=r_j+1}^{\ell_j} \sigma_i^2(\mathbf{X}_{(j)}) \right)^{1/2}.$$

**Appendix F. Accuracy.** We construct a different synthetic tensor for our second accuracy experiment. For this case, we generate  $\mathbf{X} \in \mathbb{R}^{500 \times 500 \times 500}$  to be a random 3-way tensor with true rank  $(50, 50, 50)$  and added  $10^{-4}$  relative Gaussian noise. We use  $p = 5$  for our oversampling parameter. In Figure 7, we plot the relative error resulting from our algorithms with increasing target rank. We compare our algorithms to the deterministic STHOSVD as well as the randomized algorithm using one large random matrix. In the left plot, we show the relative error from all algorithms using Gaussian random matrices, and in the right plot, we show the relative error from our suggested algorithms using SRHT random matrices. In both plots, the relative error for all algorithms is large until we reach the true rank. At that point, the error drops close to the noise level, but the randomized algorithms have a higher relative error than the deterministic. All the randomized algorithms are comparable to each other, with errors exceeding the deterministic algorithm by factors of 2 to  $7\times$ .

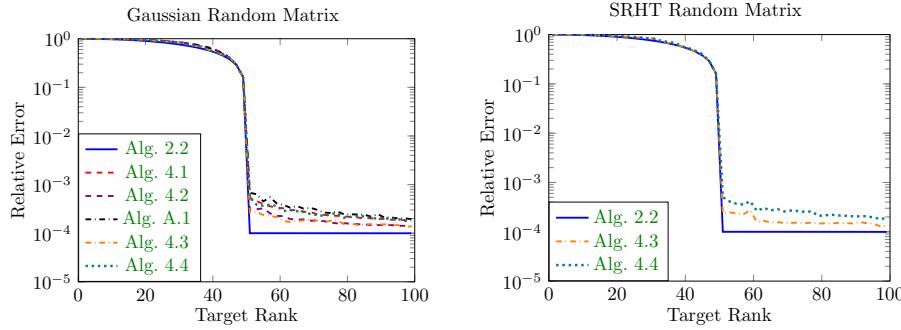


Fig. 7: Relative error of our randomized algorithms with Gaussian (left) and SRHT (right) random matrices as the target rank  $(r, r, r)$  increases on the synthetic tensor with true rank  $(50, 50, 50)$  and  $10^{-4}$  relative Gaussian noise.

**Appendix G. Computation and Communication Cost Comparison with Previous Work.** In this section, we compare the leading terms of the per-processor computation and communication cost of performing the Tucker decomposition using the four algorithms listed in Table 3. We will go over the expressions in each cell row by row. To reiterate notations, the input tensor is a  $d$ -way tensor  $\mathbf{X} \in \mathbb{R}^{n \times \dots \times n}$  with rank  $(r, r, \dots, r)$  and the size of each mode of the  $d$ -way processor tensor is  $q$ . We also assume that  $s < r < l \ll n$  where  $s$  is the subrank for each mode and  $\ell = r + p$  with  $p$  the oversampling parameter.

For the STHOSVD algorithm analyzed in [3, Section 7], the computation cost of forming the factor matrices includes computing the Gram matrices of the tensor unfoldings and the eigendecomposition of those Gram matrices. Computing the Gram matrices dominates the cost in this scenario. Furthermore, although there are  $d$  Gram matrices, one for each mode, the latter  $d - 1$  Gram matrices are much cheaper to compute than the first; this reduction is due to the truncation occurring after processing each mode of the tensor. Therefore, when comparing the computation and bandwidth cost, we count only the first Gram matrix computation. For computation cost, the leading term is  $\frac{n^{d+1}}{P}$ , resulting from multiplying the local tensor unfolding of size  $\frac{n}{q} \times (\frac{n}{q})^{d-1}$  with its own transpose. Leveraging the symmetry, the constant prefactor is 1. Considering bandwidth cost, computing the first Gram matrix is dominated by an all-to-all on the local input tensor which has  $\frac{n^d}{P}$  elements. Since

latency cost is not affected by truncation of the tensor, we count all  $d$  Gram matrix computations. The point-to-point all-to-all collective is performed over the processor fiber, so the cost is  $\mathcal{O}(q) = \mathcal{O}(P^{1/d})$  per mode.

Forming the core tensor requires  $d$  TTMs. Again, due to truncation, we count only the first TTM as the leading computational cost. In the first TTM, we multiply the local factor matrix of size  $r \times \frac{n}{q}$  with the local tensor unfolding of size  $\frac{n}{q} \times (\frac{n}{q})^{d-1}$ , which leads us to the term  $2\frac{rn^d}{P}$ . The communication involves one reduce-scatter on the product of the local factor matrix and the local input tensor, leading us to the term  $\beta\mathcal{O}(r(\frac{n}{q})^{d-1}) = \beta\mathcal{O}(\frac{rn^{d-1}}{P^{1-1/d}})$ . For latency cost, there are  $d$  reduce-scatters, one for each mode, over the processor fiber, yielding  $\mathcal{O}(d \log q) = \mathcal{O}(\log P)$  messages.

For the algorithm proposed by Choi et al. in [12], the computational cost of forming the factor matrices is the same as stated above since they also compute the SVD through local eigendecomposition of the Gram matrices. The same applies to the computation cost of forming the core tensor.

The communication differs slightly. In this algorithm, for every other mode, an all-to-all among all the processors is performed to redistribute the global tensor unfolding into a 1D block-column fashion, which contributes  $\alpha\mathcal{O}(dP)$  to the latency cost and  $\beta\mathcal{O}(\frac{n^d}{P})$  to the bandwidth cost. Again we only count the first all-to-all for bandwidth due to truncation of the tensor in subsequent modes. Computing the Gram matrices requires an additional  $d$  all-reduces among all the processors, which contributes  $\alpha\mathcal{O}(d \log P)$  and  $\beta\mathcal{O}(dn^2)$  to the latency and bandwidth cost respectively, but these are lower order terms. Forming the core tensor does not require additional communication given the redistributions described above.

In Alg. 6.4, the computation cost of forming the factor matrices is dominated by the cost of forming the 1st factor matrix due to truncation, which is further dominated by the cost of multiplying the first random matrix of size  $s \times \frac{n}{P^{1/d}}$  with the local tensor unfolding. Thus the leading term arrives at  $2s\frac{n^d}{P}$ , which has leading order term  $2\frac{r^{1/(d-1)}n^d}{P}$ . Using the all-at-once multi-TTM algorithm, the communication is that of a reduction of only the final result, which has dimensions  $(n/P^{1/d}) \times s \times \dots \times s$ , for a cost of  $\alpha\mathcal{O}(\log P) + \beta\mathcal{O}(s^{d-1}n/P^{1/d})$ . This cost is consistent across modes. Forming the core in Alg. 6.4 has similar computation and communication cost to that of the STHOSVD algorithm except for the sizes of the factor matrices are  $l \times \frac{n}{P^{1/d}}$ , but the leading order term in the costs stays the same. This assumes the in-sequence Multi-TTM algorithm is used, as in our implementation.

In Alg. 6.5, the computation cost of forming the factor matrices is double that of Alg. 4.3 because the cost is dominated by the first two internal nodes of the dimension tree, each of which involves multiplying one random matrix with the local input tensor. The communication cost of forming the factor matrices is also similar to that of Alg. 6.4. For Alg. 6.5, the leading terms of the computation and communication cost of forming the core tensor are the same as that of Alg. 6.4.

## Appendix H. Additional Experiments.

**H.1. Miranda and SP visualizations.** We show a three-dimensional visualization of a portion of the original Miranda tensor in Figure 8.

To validate the effectiveness of our algorithms on the Miranda tensor, we compare visualizations of a slice of the reconstructed tensor computed via Algs. 2.2, 4.2 and 6.5. The slices are shown in Figure 9. All three algorithms achieve a reconstruction error of 0.01 and very little difference can be seen between the results of the algorithms and the

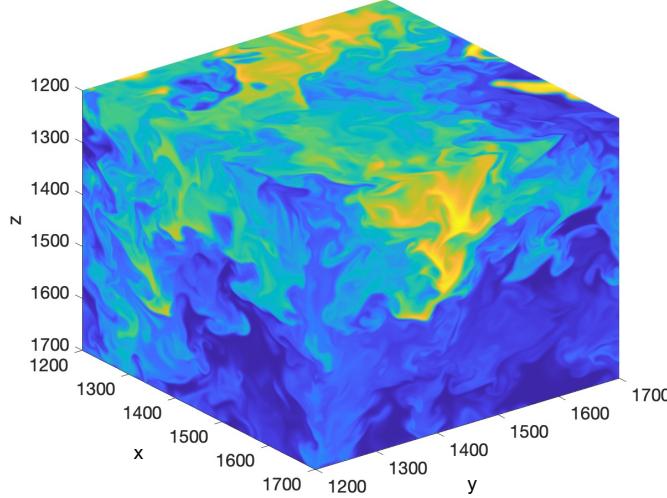


Fig. 8: Three-dimensional visualization of a subtensor of the Miranda tensor, with indices from 1200 to 1700 in each mode

original data, shown in the first row. Note that we picked [Alg. 6.5](#) as a representative algorithm while [Alg. 6.4](#) and [Alg. A.1](#) with AAO-mTTM generate similar results.

We also visualize the slices of the the original SP tensor as well as of the tensors reconstructed from the decompositions computed by our randomized algorithms in [Figure 10](#). We can see that all of the algorithms preserve the significant features of the three slices although some compression artifacts can be seen. One reason why these artifacts are more noticeable, compared to the results we get from the Miranda dataset, is that we are zooming into a much smaller portion ( $500 \times 500$  instead of  $3072 \times 3072$ ) of the total tensor.

## REFERENCES

- [1] S. AHMADI-ASL, S. ABUKHOVICH, M. G. ASANTE-MENSAH, A. CICHOCKI, A. H. PHAN, T. TANAKA, AND I. OSELEDETS, *Randomized algorithms for computation of Tucker decomposition and higher order SVD (HOSVD)*, IEEE Access, 9 (2021), pp. 28684–28706, <https://doi.org/10.1109/ACCESS.2021.3058103>.
- [2] H. AL DAAS, G. BALLARD, L. GRIGORI, S. KUMAR, AND K. ROUSE, *Communication lower bounds and optimal algorithms for multiple tensor-times-matrix computation*, Tech. Report 2207.10437, arXiv, 2022, <https://doi.org/10.48550/ARXIV.2207.10437>.
- [3] G. BALLARD, A. KLINVEX, AND T. G. KOLDA, *TuckerMPI: A parallel C++/MPI software package for large-scale data compression via the Tucker tensor decomposition*, ACM Transactions on Mathematical Software, 46 (2020), <https://doi.org/10.1145/3378445>.
- [4] K. BATSELIER, W. YU, L. DANIEL, AND N. WONG, *Computing low-rank approximations of large-scale matrices with the tensor network randomized SVD*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 1221–1244, <https://doi.org/10.1137/17M1140480>.
- [5] C. BATTAGLINO, G. BALLARD, AND T. G. KOLDA, *A practical randomized CP tensor decomposition*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 876–901, <https://doi.org/10.1137/17M1112303>.
- [6] W. H. CABOT AND A. W. COOK, *Reynolds number effects on Rayleigh–Taylor instability with possible implications for type ia supernovae*, Nature Physics, 2 (2006), pp. 562–568, <https://doi.org/10.1038/nphys361>.
- [7] V. T. CHAKARAVARTHY, J. W. CHOI, D. J. JOSEPH, X. LIU, P. MURALI, Y. SABHARWAL, AND D. SREEDHAR, *On optimizing distributed Tucker decomposition for dense tensors*, in 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2017,

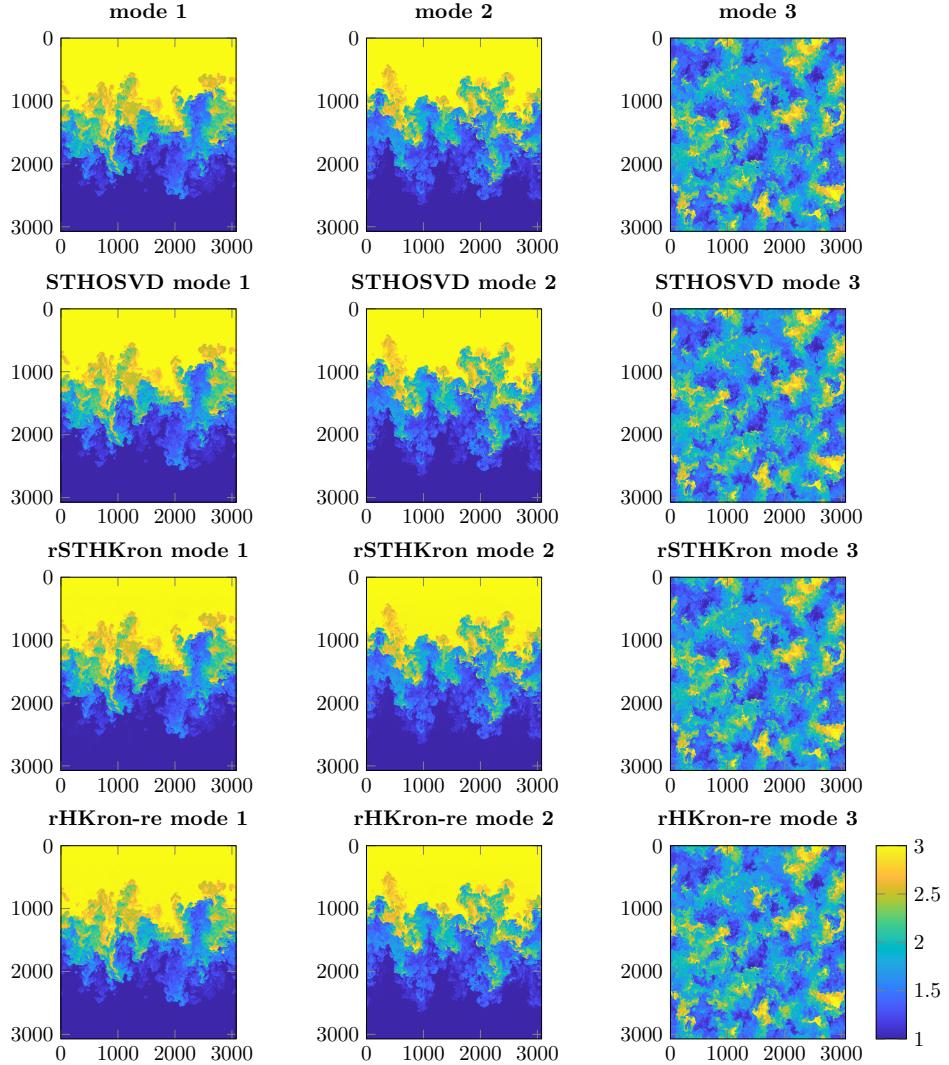


Fig. 9: 1500-th slices of the original Miranda tensor for each mode (first row) compared to reconstructions of the 1500-th slices generated by STHOSVD (second row), [Alg. 6.4](#) (third row), and [Alg. 6.5](#) (last row)

- pp. 1038–1047, <https://doi.org/10.1109/IPDPS.2017.86>.
- [8] E. CHAN, M. HEIMLICH, A. PURKAYASTHA, AND R. VAN DE GEIJN, *Collective Communication: Theory, Practice, and Experience*, Concurrency and Computation: Practice and Experience, 19 (2007), pp. 1749–1783, <https://doi.org/10.1002/cpe.1206>.
  - [9] M. CHE AND Y. WEI, *Randomized algorithms for the approximations of Tucker and the tensor train decompositions*, Advances in Computational Mathematics, 45 (2019), pp. 395–428, <https://doi.org/10.1007/s10444-018-9622-8>.
  - [10] M. CHE, Y. WEI, AND H. YAN, *The computation of low multilinear rank approximations of tensors via power scheme and random projection*, SIAM Journal on Matrix Analysis and Applications, 41 (2020), pp. 605–636, <https://doi.org/10.1137/19M1237016>.
  - [11] M. CHE, Y. WEI, AND H. YAN, *An efficient randomized algorithm for computing the approximate Tucker decomposition*, Journal of Scientific Computing, 88 (2021), pp. 1–29, <https://doi.org/10.1007/s10915-021-01545-5>.
  - [12] J. CHOI, X. LIU, AND V. CHAKARAVARTHY, *High-performance dense Tucker decomposition*

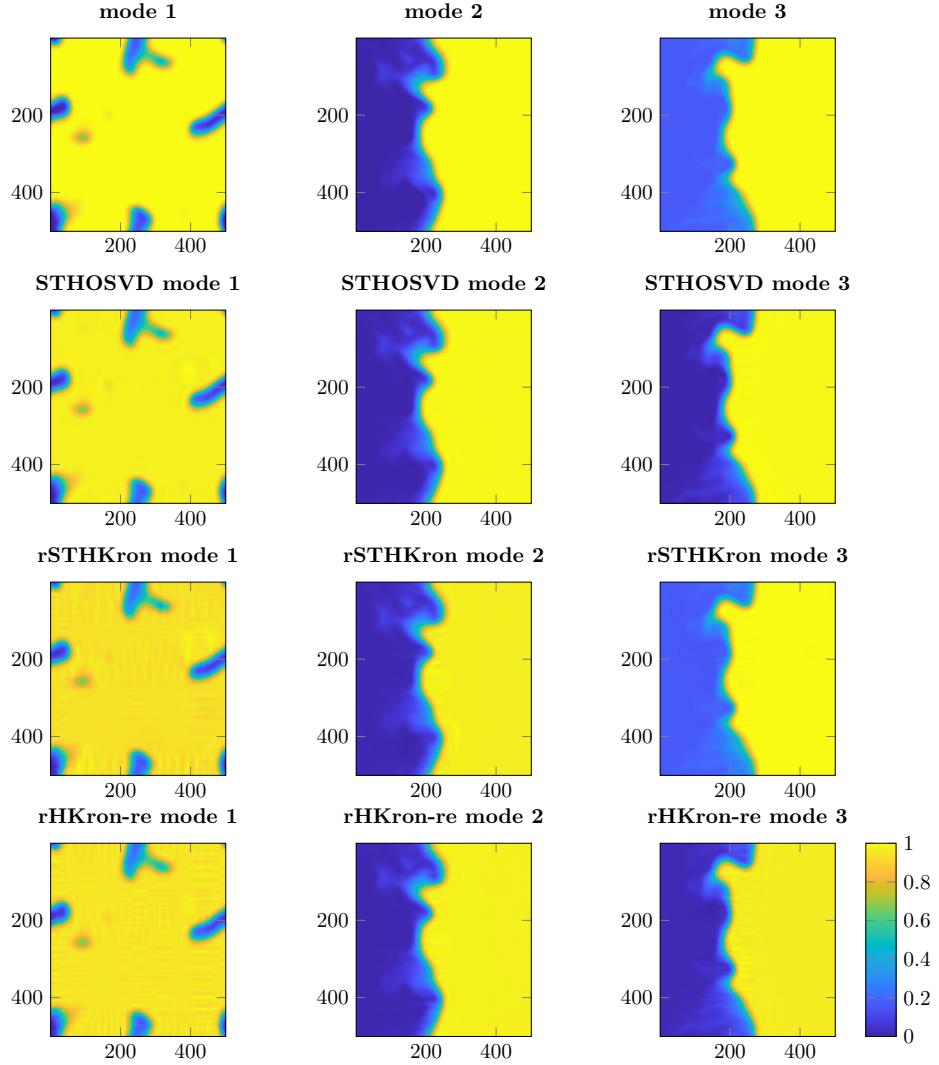


Fig. 10: 250th slices of the original SP tensor for each mode (first row) compared to reconstruction of the 250th slices generated by STHOSVD (second row), [Alg. 6.4](#) (third row), and [Alg. 6.5](#) (last row)

on GPU clusters, in SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 543–553, <https://doi.org/10.1109/SC.2018.00045>.

- [13] H. A. DAAS, G. BALLARD, P. CAZEAUX, E. HALLMAN, A. MIEDLAR, M. PASHA, T. W. REID, AND A. K. SAIBABA, *Randomized algorithms for rounding in the tensor-train format*, arXiv preprint arXiv:2110.04393, (2021).
- [14] L. DE LATHAUWER, B. DE MOOR, AND J. VANDEWALLE, *A multilinear singular value decomposition*, SIAM journal on Matrix Analysis and Applications, 21 (2000), pp. 1253–1278, <https://doi.org/10.1137/S0895479896305696>.
- [15] L. DE LATHAUWER, B. DE MOOR, AND J. VANDEWALLE, *On the best rank-1 and rank- $(r_1, r_2, \dots, r_n)$  approximation of higher-order tensors*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1324–1342, <https://doi.org/10.1137/S0895479898346995>.
- [16] N. B. ERICHSON, K. MANOHAR, S. L. BRUNTON, AND J. N. KUTZ, *Randomized CP tensor decomposition*, Machine Learning: Science and Technology, 1 (2020), p. 025012, <https://doi.org/10.1109/MLST50032.2020.9117001>.

[//doi.org/10.1088/2632-2153/ab8240](https://doi.org/10.1088/2632-2153/ab8240).

- [17] N. HALKO, P.-G. MARTINSSON, AND J. A. TROPP, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM review, 53 (2011), pp. 217–288, <https://doi.org/10.1137/090771806>.
- [18] R. A. HORN AND C. R. JOHNSON, *Matrix analysis*, Cambridge University Press, 2012.
- [19] R. JIN, T. G. KOLDA, AND R. WARD, *Faster Johnson–Lindenstrauss transforms via Kronecker products*, Information and Inference: A Journal of the IMA, 10 (2021), pp. 1533–1562, <https://doi.org/10.1093/imaiia/iaaa028>.
- [20] O. KAYA AND Y. ROBERT, *Computing dense tensor decompositions with optimal dimension trees*, Algorithmica, 81 (2019), pp. 2092–2121, <https://doi.org/10.1007/s00453-018-0525-3>.
- [21] O. KAYA AND B. UÇAR, *High performance parallel algorithms for the Tucker decomposition of sparse tensors*, in 45th International Conference on Parallel Processing (ICPP ’16), 2016, pp. 103–112, <https://doi.org/http://dx.doi.org/10.1109/ICPP.2016.19>.
- [22] T. G. KOLDA AND B. W. BADER, *Tensor decompositions and applications*, SIAM review, 51 (2009), pp. 455–500, <https://doi.org/10.1137/07070111X>.
- [23] H. KOLLA, X.-Y. ZHAO, J. H. CHEN, AND N. SWAMINATHAN, *Velocity and reactive scalar dissipation spectra in turbulent premixed flames*, Combustion Science and Technology, 188 (2016), pp. 1424–1439, <https://doi.org/10.1080/00102202.2016.1197211>.
- [24] Z. LI, Q. FANG, AND G. BALLARD, *Parallel Tucker decomposition with numerically accurate SVD*, in 50th International Conference on Parallel Processing, ICPP ’21, New York, NY, USA, August 2021, ACM, p. 11, <https://doi.org/10.1145/3472456.3472472>.
- [25] R. MINSTER, A. K. SAIBABA, AND M. E. KILMER, *Randomized algorithms for low-rank tensor decompositions in the Tucker format*, SIAM Journal on Mathematics of Data Science, 2 (2020), pp. 189–215, <https://doi.org/10.1137/19M1261043>.
- [26] A.-H. PHAN, P. TICHAVSKY, AND A. CICHOCKI, *Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations*, IEEE Transactions on Signal Processing, 61 (2013), pp. 4834–4846, <https://doi.org/10.1109/TSP.2013.2269903>.
- [27] Y. SUN, Y. GUO, C. LUO, J. TROPP, AND M. UDELL, *Low-rank Tucker approximation of a tensor from streaming data*, SIAM Journal on Mathematics of Data Science, 2 (2020), pp. 1123–1150, <https://doi.org/10.1137/19M1257718>.
- [28] J. A. TROPP, *Improved analysis of the subsampled randomized Hadamard transform*, Advances in Adaptive Data Analysis, 3 (2011), pp. 115–126, <https://doi.org/10.1142/S1793536911000787>.
- [29] N. VANNIEUWENHOVEN, R. VANDEBRIL, AND K. MEERBERGEN, *A new truncation strategy for the higher-order singular value decomposition*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1027–A1052, <https://doi.org/10.1137/110836067>.
- [30] A. S. J. W. WOLF, *Low rank tensor decompositions for high dimensional data approximation, recovery and prediction*, PhD thesis, Technical University of Berlin, 2019, <https://doi.org/10.14279/depositonce-8109>.
- [31] F. WOOLFE, E. LIBERTY, V. ROKHLIN, AND M. TYGERT, *A fast randomized algorithm for the approximation of matrices*, tech. report, Tech. rep. 1380, Yale University, Department of Computer Science, 2007.
- [32] K. ZHAO, S. DI, X. LIAN, S. LI, D. TAO, J. BESSAC, Z. CHEN, AND F. CAPPELLO, *SDRBench: Scientific data reduction benchmark for lossy compressors*, in 2020 IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 2716–2724, <https://doi.org/10.1109/BigData50022.2020.9378449>.
- [33] G. ZHOU, A. CICHOCKI, AND S. XIE, *Decomposition of big tensors with low multilinear rank*, arXiv preprint arXiv:1412.1885, (2014), <https://doi.org/10.48550/arXiv.1412.1885>.