cuTensor-CP: High Performance Third-order CP Tensor Decomposition on GPUs

Xiao-Yang Liu[1,2] , **Han Lu**[3] , **Tao Zhang**[3,4*]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
[2]Department of Electrical Engineering, Columbia University, USA
[3]School of Computer Engineering and Science, Shanghai University, Shanghai, China
[4]Shanghai Institute for Advanced Communication and Data Science, Shanghai, China
xl2427@columbia.edu, 130210201@shu.edu.cn, taozhang@shu.edu.cn

Abstract

Tensor decompositions that factorize multi-dimensional data into latent factors have become a powerful tool for big data analytics and machine learning, e.g., video processing, deep learning, social networks, etc. However, time and space complexities of tensor decomposition algorithms grow rapidly with the size of tensors. Exploiting parallelisms of tensor algorithms and accelerating them on many-core GPUs are promising. In this paper, we develop efficient CP tensor decomposition on GPUs by exploiting tensor algorithm parallelism. We implement and optimize key operations, including tensor matricization and matricized tensor times Khatri-Rao product (MTTKRP). We fully optimize the data transmission and reduce memory footprint, even employ more efficient calculation processes with smaller computational complexity. Compared with the TensorLab library running on a Tesla V100 GPU, our implementation of CP decomposition achieves up to $5.56\times$ speedup.

Keywords GPU, CP tensor decomposition, matricized tensor times Khatri-Rao product

1 Introduction

Tensor decompositions, the higher-order analogue to matrix decomposition (e.g., Singular Value Decomposition (SVD), Principal Component Analysis (PCA), non-negative matrix decompoistion, etc.), have became a powerful tool for mining cross-dimension relationships in large-scale multi-dimensional data. Multi-dimensional data arrays in video processing, social networks are naturally represented as tensors, and tensor (multiway) decompositions are employed to perform factor analysis or compression. Tensor decompositions/factorizations have been widely used in big data analysis [9], computer vision [10] [5], pattern recognition and deep learning [6][1][7][3][12], and genetic analysis [4], etc. With the growing needs to process large amount of multi-way data in a real-time manner, designing high-performance tensor decomposition has become a critical task.

Existing research in accelerating CP tensor decomposition has limitations. TensorLab [8] requires long running time, which was not fully optimized for the GPU architecture. In this paper, we develop high performance CP tensor decomposition on GPUs. We use the bottom-up method to optimize the key operations first, and then accelerate the whole algorithm. Our contributions are summarized as follows.

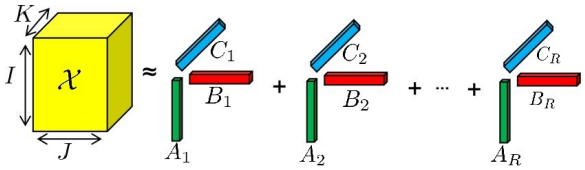


Figure 1: Third-order CP tensor decomposition.

- We implement key tensor operations, including tensor matricization and matricized tensor times Khatri-Rao product (MTTKRP). We propose a novel technique to avoid tensor matricization and thus reduce the memory consumption of MTTKRP. Moreover, for the matrix multiplication in MTTKRP, we divide large matrix into small matrices and map them onto tensor cores.
- We implement and optimize CP tensor decomposition on GPUs. Using the optimized key tensor operations, we map the algorithm onto GPU, and comprehensively reduce the memory occupation and the number of calculations to further improve the performance.
- We perform numerical experiments to evaluate the performance of MTTKRP and CP tensor decomposition. For the MTTKRP operation, we mainly compare with GPU baseline. For CP tensor decomposition, We compare with the TensorLab-GPU [8]. Our CP decomposition achieves up to $5.56\times$ speedups.

The remainder of this paper is organized as follows. Section 2 describes key tensor operations and briefly summarizes the CP tensor decomposition algorithm. Section 3 presents the design, implementation and optimizations of key operations. In Section 4, we evaluate the performance of the key tensor operations and our implementation on GPUs. The conclusions are given in Section 5.

2 CP Tensor Decomposition

In this section, we first briefly introduce notations and key tensor operations. Then, we describe the CP tensor decomposition algorithm.

*Tao Zhang is supported by Science and Technology Committee of Shanghai Municipality under grant No. 19511121002 and No. 19DZ2252600.

Algorithm 1 ALS CP tensor decomposition

1: **Input**: tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, rank R.
2: Randomly initialize $\boldsymbol{A} \in \mathbb{R}^{I\times R}, \boldsymbol{B} \in \mathbb{R}^{J\times R}, \boldsymbol{C} \in \mathbb{R}^{K\times R}$,
3: **while** convergence criterion is not met **do**
4: $\quad \boldsymbol{A} \leftarrow \boldsymbol{X}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B})(\boldsymbol{C}^\top\boldsymbol{C} .* \boldsymbol{B}^\top\boldsymbol{B})^\dagger$,
5: $\quad \boldsymbol{B} \leftarrow \boldsymbol{X}_{(2)}(\boldsymbol{C} \odot \boldsymbol{A})(\boldsymbol{C}^\top\boldsymbol{C} .* \boldsymbol{A}^\top\boldsymbol{A})^\dagger$,
6: $\quad \boldsymbol{C} \leftarrow \boldsymbol{X}_{(3)}(\boldsymbol{B} \odot \boldsymbol{A})(\boldsymbol{B}^\top\boldsymbol{B} .* \boldsymbol{A}^\top\boldsymbol{A})^\dagger$,
7: **end while**
8: **Output**: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$.

2.1 Notions and Key Tensor Operations

We use lowercase boldface letters, uppercase boldface letters and uppercase calligraphic letters to denote vectors, matrices and tensors, respectively, e.g., $\boldsymbol{x} \in \mathbb{R}^I$, $\boldsymbol{X} \in \mathbb{R}^{I\times J}$, and $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$. $\boldsymbol{X}_j$ or $\boldsymbol{X}(:, j)$ denotes the j-th column, and $\boldsymbol{X}(i, j)$ or $\boldsymbol{X}_{ij}$ denotes the (i, j)-th element, the (i, j, k)-th entry is $\mathcal{X}(i, j, k)$ or $\mathcal{X}_{ijk}$. We use $.*$, $\circ$, $\odot$ and $\otimes$ to denote Hadamard (element-wise) product, outer product, Khatri-Rao product and Kronecker (tensor) product. We use $^\top$ and † to the matrix transpose and Moore-Penrose pseudo-inverse, respectively. Indices range from 1 to their capital letters, e.g., $i = 1, ..., I$ or $i \in [I]$.

Rank-one tensor: A third-order tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ has rank one if it is the outer product of three vectors.

Tensor matricization (a.k.a. tensor unfolding or flattening): Tensor matricization converts a tensor into a matrix. The mode-n matricization of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times I_3}$ is denoted by $\mathbf{X}_{(n)}$, which arranges the mode-n tubes to be the columns of the resulting matrix. A tensor element (i_1, i_2, i_3) is mapped to matrix element (i_n, j) for $n = 1, 2, 3$, where

$$j = 1 + \sum_{\substack{k=1\\k\neq n}}^{3} (i_k - 1)J_k \text{ with } J_k = \prod_{\substack{m=1\\m\neq n}}^{k-1} I_m. \tag{1}$$

Matricized tensor times Khatri-Rao product (MTTKRP): For a tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ and given matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$, the mode-1, mode-2 and mode-3 MTTKRP is defined as $\boldsymbol{X}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B})$, $\boldsymbol{X}_{(2)}(\boldsymbol{C} \odot \boldsymbol{A})$ and $\boldsymbol{X}_{(3)}(\boldsymbol{B} \odot \boldsymbol{A})$, respectively.

2.2 CP Tensor Decomposition Algorithm

We focus on CP tensor decomposition [2], as shown in Fig. 1. The CP tensor decomposition factorizes a tensor into the sum of rank-one tensor components. For a third-order tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ and target rank R, we have $\mathcal{X} \approx \sum_{r=1}^{R} \boldsymbol{A}_r \circ \boldsymbol{B}_r \circ \boldsymbol{C}_r$, where $\boldsymbol{A}_r \in \mathbb{R}^I$, $\boldsymbol{B}_r \in \mathbb{R}^J$, $\boldsymbol{C}_r \in \mathbb{R}^K$ for $r \in [R]$. The target is to compute a CP tensor decomposition with R components that best approximates $\mathcal{X}$, i.e.,

$$\arg\min_{\widehat{\mathcal{X}}} \|\mathcal{X} - \widehat{\mathcal{X}}\|_F \tag{2}$$

where $\widehat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \boldsymbol{A}_r \circ \boldsymbol{B}_r \circ \boldsymbol{C}_r = [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ and $\boldsymbol{A} \in \mathbb{R}^{I\times R}, \boldsymbol{B} \in \mathbb{R}^{J\times R}, \boldsymbol{C} \in \mathbb{R}^{K\times R}$, respectively. The Frobenius norm of a tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ is defined as $\|\mathcal{X}\|_F = \sqrt{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} |\mathcal{X}_{ijk}|^2}$.

The third-order CP tensor decomposition is given in Alg. 1. The alternating least square method (ALS) fixes all but one matrix and reduces the CP tensor decomposition problem to a linear least-squares problem. In line 2, the factor matrices are randomly initialized. Lines 3-7 are the iterative process and the algorithm updates the factor matrices (lines 4-6) alternatively. The algorithm terminates when the approximation error is below a pre-specified threshold or it reaches a preset maximum number of iterations.

3 Optimization of Key Tensor Operations

The horizontal, lateral, and frontal slices of a third-order tensor $\mathcal{X}$ are denoted as $\mathcal{X}(i, :, :)$, $\mathcal{X}(:, j, :)$, and $\mathcal{X}(:, :, k)$, respectively. Alternatively, the k-th frontal slice $\mathcal{X}(:, :, k)$ is denoted as $\mathcal{X}^{(k)}$.

3.1 Tensor Matricization

Tensor matricization is a fundamental operation in CP tensor decomposition. In Alg. 1, lines 4-6 need to compute different mode matricizations $\boldsymbol{X}_{(1)}, \boldsymbol{X}_{(2)}$ and $\boldsymbol{X}_{(3)}$. In conventional implementation, GPU allocates additional memory and performs explicit tensor matricization, which introduces substantial memory and time cost.

We observe that explicit tensor matricizations can be eliminated to save computation and memory space by exploiting the storage format in the GPU memory. A third-order tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ is stored in GPU memory in slice-by-slice, column-major layout using a 1D array $\boldsymbol{x}$, where $\mathcal{X}_{ijk}$ is stored at $\boldsymbol{x}[(k-1)IJ + (j-1)I + i]$. First, the column-major storage of mode-1 matricization $\boldsymbol{X}_{(1)} \in \mathbb{R}^{I\times JK}$ in memory is exactly the same as $\boldsymbol{x}$. Second, by transposing each slice of $\mathcal{X}$, we obtain $\boldsymbol{X}_{(2)}$. Third, the row-major storage of mode-3 matricization $\boldsymbol{X}_{(3)}$ is exactly the same as $\boldsymbol{x}$. Therefore, by exploiting these property, we are able to use $\mathcal{X}$ directly and avoid explicit tensor matricization to save computation and GPU memory.

In CUDA programming, we obtain the three matricizations $\boldsymbol{X}_{(1)}, \boldsymbol{X}_{(2)}, \boldsymbol{X}_{(3)}$ by different ways of fetching the 1D data in the memory. In the cuBLAS library, cublasSgemm() and cublasSgemmBatchedStrided() are interfaces of matrix multiplication and parallel matrix multiplication. For mode-1 matricization $\boldsymbol{X}_{(1)} \in \mathbb{R}^{I\times JK}$, we use cublasSgemm() and set the main dimension as I. For mode-2 matricization $\boldsymbol{X}_{(2)} \in \mathbb{R}^{J\times IK}$, we use cublasSgemmBatchedStrided() and set the leading dimension to J, and the number of matrix multiplications to K. For mode-3 matricization $\boldsymbol{X}_{(3)} \in \mathbb{R}^{K\times IJ}$, we find that the column-major storage of $\mathcal{X}$ in 1D array is same as the transpose of $\boldsymbol{X}_{(3)}$. Therefore, we use cublasSgemm() and set the leading dimension as K. In general, through the access strategy, we cleverly regard physical storage data as the form of logical storage we require.

3.2 Matricized Tensor Times Khatri-Tao Product (MTTKRP)

MTTKRP is a basic operation in tensor computing. The convention approach to compute MTTKRP include the following three steps:

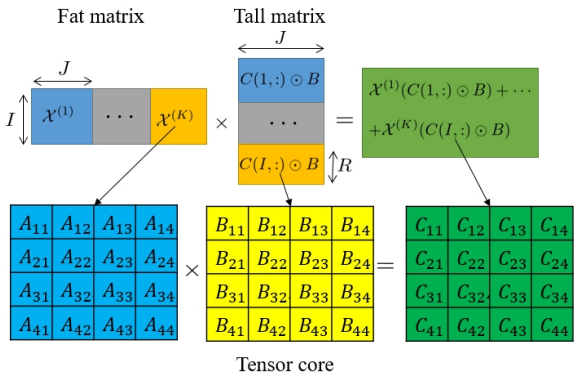


Figure 2: Block computation for the MTTKRP operation.

- Matricizing a tensor into a matrix: $\mathcal{X} \rightarrow \boldsymbol{X}_{(1)}$, $\boldsymbol{X}_{(2)}$, $\boldsymbol{X}_{(3)}$;
- Calculating Khatri-Rao product and obtain $(\boldsymbol{C} \odot \boldsymbol{B})$, $(\boldsymbol{C} \odot \boldsymbol{A})$, $(\boldsymbol{B} \odot \boldsymbol{A})$;
- Executing matrix multiplication: $(\boldsymbol{X}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B}))$, $(\boldsymbol{X}_{(2)}(\boldsymbol{C} \odot \boldsymbol{A}))$, $(\boldsymbol{X}_{(3)}(\boldsymbol{B} \odot \boldsymbol{A}))$.

We eliminate the tensor matricization in the first step through the technique in Section 3.1. Since in the third step, tensor matricization results in a fat matrix and the Khatri-Rao product results in a tall matrix, we use tensor core to accelerate this matrix multiplication.

Batching Block Computations onto Tensor Cores

In Alg. 1, the main while-loop in lines 4-6 performs three MTTKRP operations. This process is time-consuming and becomes a bottleneck. As matrix multiplication can be calculated in a block manner, we divide the large matrices into smaller matrices and batch the block matrix multiplications onto tensor cores.

Tensor cores are novel computing units introduced in latest NVIDIA GPU architectures including Volta and Turing. Compared with conventional CUDA cores, tensor cores are especially efficient for accelerating the computation of block matrix multiplications. Tesla V100 GPU has 640 tensor cores in total. The matrix multiplication of two 4×4 matrices can be calculated on a tensor core at one time. Fig. 2 illustrates the computing of $\boldsymbol{X}_{(1)}(\boldsymbol{C} \odot \boldsymbol{B})$, where $\boldsymbol{X}_{(1)} \in \mathbb{R}^{I \times JK}$, $\boldsymbol{C} \in \mathbb{R}^{K \times R}$ and $\boldsymbol{B} \in \mathbb{R}^{J \times R}$. We divide the fat matrix $\boldsymbol{X}_{(1)}$ and the tall matrix $\boldsymbol{C} \odot \boldsymbol{B}$ into $(IJK)/16$ and $(JKR)/16$ small blocks, respectively. Then, we carry out a total of $I/4 \times (JK)/4 \times (R/4)$ block multiplications and batch them onto 640 tensor cores. When one is calculated, the blocks in the queue will immediately occupy the computing resources.

4 Performance Evaluations

We run all experiments on a server with an NVIDIA Tesla V100 GPU and dual Intel Xeon E5-2640 V4 CPUs. The Tesla V100 GPU has 32 GB device memory, $5,120$ CUDA cores and 640 tensor cores. Each CPU has 10 cores running at 2.4GHz. The operating system of the server is 64-bit Ubuntu 18.04.

We use running time as performance metric. The speedup of our GPU implementation over a reference GPU implementation is calculated as: (running time of a reference GPU implementation) / (running time of our GPU implementation). The compared GPU implementations are listed out as follows

- **GPU Baseline**: We provide baseline implementation on GPU using the BLAS and cuSOLVER libraries. This implementation does not utilize the optimization techniques in Section 3.
- **TensorLab-GPU** [8]: TensorLab is a well-maintained MATLAB toolbox for tensor computations. We run TensorLab on GPUs.
- **Our GPU implementation (Ours)**: For the key tensor operations and CP tensor decomposition algorithm, our implementations employ the optimization techniques in Section 3.

4.1 Key Tensor Operations

In this subsection, our experiments are running on Tesla V100 GPU. Running time is measured in log scale. For the key tensor operation (MTTKRP), we report the kernel running time that do not include the data transfer time between CPU and GPU. Because the tensor operation is normally called during the computation process of the tensor decomposition algorithms and the data are already in the GPU device memory.

Fig. 3 shows the running time and speedups of the mode-1 MTTKRP, mode-2 MTTKRP and mode-3 MTTKRP with varying tensor and matrix sizes, respectively. Our input is a third-order tensor $I \times I \times I$ and two matrices of the same size $I \times R$, which R is set to be $0.1 \times I$ and the corresponding outputs are three tensors of the same size $I \times R$.

In Fig. 3(a), compared with GPU Basline, our mode-1 MTTKRP using CUDA cores and tensor cores achieves up to $1.84\times$ and $5.34\times$ speedups, respectively.

In Fig. 3(b), our mode-2 MTTKRP using CUDA cores and tensor cores achieves up to $1.04\times$ and $1.43\times$ speedups, respectively. The reason for the low performance improvement than mode-1 MTTKRP is that mode-2 MTTKPR cannot fully avoid the tensor matricization operation.

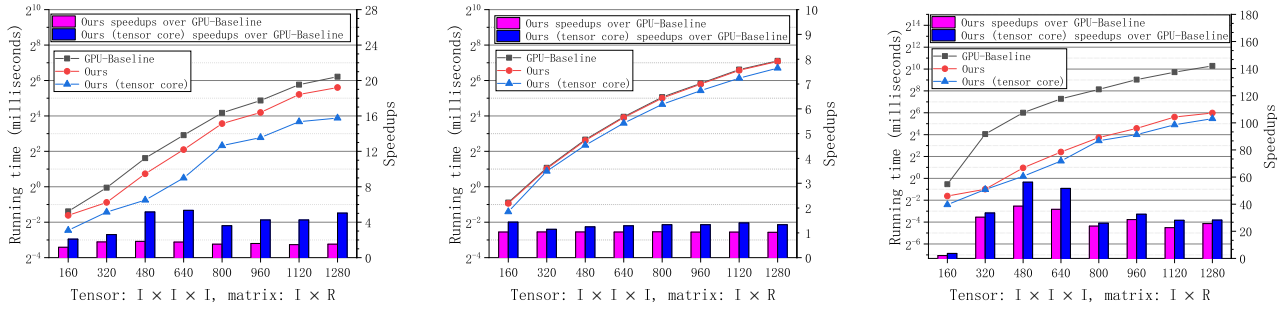
In Fig. 3(c), our mode-3 MTTKRP using CUDA cores and tensor cores achieves up to $38.55\times$ and $56.21\times$ speedups, respectively. The reason is that the original tensor matricization in the operation of mode-3 MTTKPR breaks the continuity of data access seriously. Eliminating such a tensor matricization operation leads to this high performance improvement.

4.2 Third-order CP Tensor Decomposition

Fig. 4 shows the running time and speedups of the CP tensor decomposition. We test the third-order tensor of size $I \times I \times I$ varying from $100 \times 100 \times 100$ to $1,200 \times 1,200 \times 1,200$. Two implementations are compared: our GPU implementation and TensorLab-GPU [8]. Our GPU implementation achieves up to $5.56\times$ speedup versus the TensorLab-GPU [8].

5 Conclusions

Tensor operations have attracted lots of attention in recent years. Tensor decompositions have been widely used in big



(a) mode-1 MTTKRP. (b) mode-2 MTTKRP. (c) mode-3 MTTKRP.

Figure 3: Running time and speedups of MTTKRP in three modes, respectively.

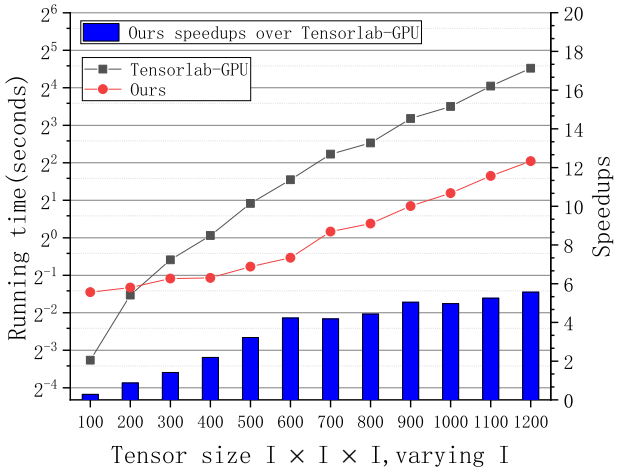


Figure 4: Running time and speedups of CP decomposition.

data analysis, computer vision, pattern recognition, and deep learning, etc. However, due to the high computational complexity, existing implementations are not satisfactory in terms of running time and memory consumption. In this paper, we optimized the computations of CP tensor decomposition on many-core GPUs. We proposed optimization strategies for reduced memory consumption to accelerate tensor operations. Compared with TensorLab-GPU running on a Tesla V100 GPU, our implementation of CP tensor decomposition achieves up to a $5.56\times$ speedup. Our future work will incorporate this implementation into the cuTensor library [11].

References

[1] Yunpeng Chen, Xiaojie Jin, Bingyi Kang, Jiashi Feng, and Shuicheng Yan. Sharing residual units through collective tensor factorization to improve deep neural networks. In *IJCAI*, pages 635–641, 2018.

[2] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[3] Xiaochen Han, Bo Wu, Zheng Shou, Xiao-Yang Liu, Yimeng Zhang, and Linghe Kong. Tensor FISTA-Net for real-time snapshot compressive imaging. In *AAAI*, pages 10933–10940, 2020.

[4] Victoria Hore, Ana Viñuela, Alfonso Buil, Julian Knight, Mark I McCarthy, Kerrin Small, and Jonathan Marchini. Tensor decomposition for multiple-tissue gene expression experiments. *Nature Genetics*, 48(9):1094, 2016.

[5] Fei Jiang, Xiao-Yang Liu, Hongtao Lu, and Ruimin Shen. Efficient multi-dimensional tensor sparse coding using t-linear combination. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[6] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *ICLR*, 2015.

[7] Jiawei Ma, Xiao-Yang Liu, Zheng Shou, and Xin Yuan. Deep tensor ADMM-Net for snapshot compressive imaging. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10223–10232, 2019.

[8] Nico Vervliet, Otto Debals, Laurent Sorber, Marc Van Barel, and Lieven De Lathauwer. Tensorlab 3.0. *available online, URL: www. tensorlab. net*, 2016.

[9] Yuto Yamaguchi and Kohei Hayashi. Tensor decomposition with missing indices. In *IJCAI*, pages 3217–3223, 2017.

[10] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3891–3900. JMLR. org, 2017.

[11] Tao Zhang, Xiao-Yang Liu, Xiaodong Wang, and Anwar Walid. cuTensor-Tubal: Efficient primitives for tubal-rank tensor learning operations on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 31(3):595–610, 2019.

[12] Yimeng Zhang, Xiao-Yang Liu, Bo Wu, and Anwar Walid. Video synthesis via transform-based tensor neural networks. In *ACM Multimedia*, 2020.