Entropy Coding Compression of Tree Tensor Networks

Rafael Ballester-Ripoll[1] **and Roxana Bujack**[2]

[1]School of Science and Technology, IE University
[2]Los Alamos National Laboratory
rafael.ballester@ie.edu, bujack@lanl.gov

Abstract

Low-rank decompositions have been successfully used to compactly represent tensors of many kinds, be it neural network layers, activation tensors, or raw datasets arising from tomography scans, sensor data, physical simulations, etc. These methods are often based on low-rank factorization and do not post-process coefficients thereafter (except, sometimes, quantization). That choice allows compressed tensors to be more easily used in learning pipelines, but it is not necessarily optimal for data storage or transmission purposes. Focusing on these tasks, we propose to prioritize data reduction rates by applying entropy coding and successive core orthogonalization to SVD-learned coefficients of a given tensor. Our scheme generalizes earlier Tucker-based compressors to more general acyclic tensor networks, and is thus promising for a wider class of target tensors.

Introduction and Related Work

Lossy tensor compression is nowadays an attractive goal for which multiple algorithms exist, either within a learning pipeline (Novikov et al. 2015) or as a post-processing step: see SZ (Di and Cappello 2016), ZFP (Lindstrom 2014), wavelets, implicit neural representations (Lu et al. 2021), etc. Within the realm of transform compression, SVD-based decompositions have emerged as a family of powerful methods. These decompositions learn data-dependent multilinear projection bases; see e.g. TTHRESH (Ballester-Ripoll, Lindstrom, and Pajarola 2020), TuckerMPI (Ballard, Klinvex, and Kolda 2020), or ATC (Baert and Vannieuwenhoven 2021) which use the so-called Tucker model. The effectiveness of these methods lies in combining tools from signal processing and information theory (such as low-rank truncation, bit plane truncation, run-length encoding, progressive reconstruction, or entropy coding) with the strong decorrelation properties of the higher-order singular value decomposition (HOSVD) (de Lathauwer, de Moor, and Vandewalle 2000).

Despite the success of the HOSVD/Tucker model in the tensor and machine learning literature, many recent works have fruitfully switched to more modern tensor network topologies: the tensor train (TT) (Oseledets 2011), "extended" TT (Schneider and Uschmajew 2014), quantized TT (Khoromskij and Oseledets 2010), hierarchical Tucker (Grasedyck 2010), quantized TT (QTT, Oseledets and Tyrtyshnikov 2011), etc. Often, these alternatives can better break down the curse of dimensionality and offer higher compression rates, especially for higher dimensional tensors. Still, to the best of our knowledge, no compression algorithm that focuses on the floating-point representation (as TTHRESH and ATC do) has been attempted for these, more general tensor networks.

Lossy compression is a multi-objective optimization: there is a trade-off between bit rate b and the lowest error ϵ that can be achieved with it; this trade-off defines the so-called rate-distortion curve. We argue this is a major difficulty of adapting prior methods to larger tensor networks: each core may have a different impact on the overall error. Hence, for best results, one may want to select a different number of bits b_k for each core k. This is much less of an issue in the Tucker case, where the central core tends to take the vast majority of compressed coefficients and each b_k may be safely chosen via an *ad hoc* heuristic. In the more general case, we tackle this via global optimization on a combined rate-distortion curve assembled from recursively orthogonalized cores; see the following section.

Compression

Our algorithm takes three inputs: a target tensor $\mathcal{X}$, a tree topology, and a prescribed *relative error* $\bar{\epsilon}$, meaning that the approximation should satisfy $\|\tilde{\mathcal{X}} - \mathcal{X}\|/\|\mathcal{X}\| \approx \bar{\epsilon}$ (the Frobenius norm is used). Another important concept is that of *isometry*: a core $\mathcal{C}$ is said to be isometric with respect to a subset $\mathcal{I}$ of its indices if, when unfolded as a matrix where $\mathcal{I}$ index the rows, the matrix's columns are orthonormal. This can be depicted with incoming arrows towards $\mathcal{C}$ for all edges in $\mathcal{I}$. We say that a core is *canonical* (or the *center of orthogonality*) if all other cores in the tree have a direct path towards it (see Fig. 1 for an example).

At a high-level, the compression steps are:

1. Select a central node i in the desired topology. A central starting point ensures that isometries do not need to be carried too far away, which mitigates accumulation of round-off errors.
2. Decompose the input tensor $\mathcal{X}$ into the topology using

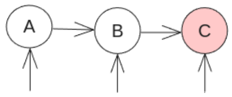


Figure 1: TT network where cores $\mathcal{A}$ and $\mathcal{B}$ are isometric towards $\mathcal{C}$ and therefore $\mathcal{C}$ is canonical (shown in red).

successive SVD factorizations so that, after this process, the core $\mathcal{C}_i$ is canonical and all isometries in the network point towards $\mathcal{C}_i$.

3. Calculate a *local* rate-distortion curve $(b_i(c_i), \epsilon_i(c_i))$ for $\mathcal{C}_i$, where c_i is the core's compression level. The higher the c_i, the higher b_i (bit rate of core $\mathcal{C}_i$) and the lower ϵ_i (relative error in that core).
4. Recursively canonize outwards core by core. At each step, cache the current core k (which is now canonical) and calculate its local curve.
5. Combine all curves to assemble an overall cost function $(b(\boldsymbol{c}), \epsilon(\boldsymbol{c}))$ and solve $\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} b(\boldsymbol{c})$ s.t. $\epsilon(\boldsymbol{c}) \leq \bar{\epsilon}$.
6. Encode each cached core k at level c_k^*.

See Figure 2 for a diagram depicting this tree traversal scheme. For more details on rate selection across the different cores in the network, we refer to the following subsections.

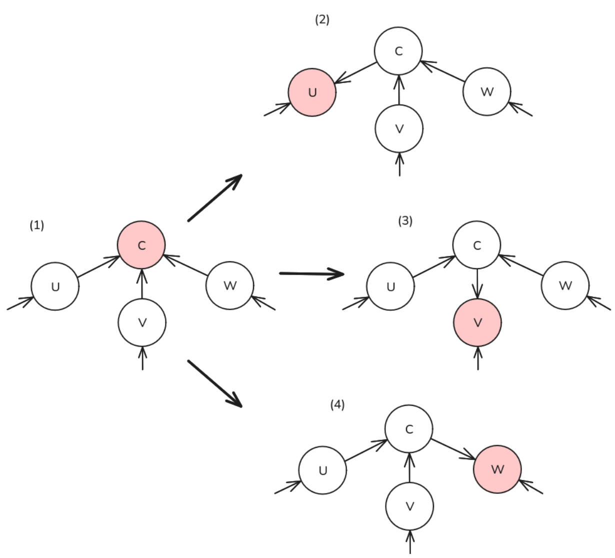


Figure 2: Compression example using 3D Tucker topology: the tree is traversed pre-order and outwards from the central core $\mathcal{C}$. At each traversal step we obtain the local rate-distortion curve for the canonized core (shown in red).

Local Rate-distortion Curves

Let $\mathcal{C}_k$ be a canonical core containing S elements in the tensor network. Canonicity of $\mathcal{C}_k$ means that $\|\mathcal{C}_k\| = \|\mathcal{X}\|$ and, if we perturb it into a distorted core $\tilde{\mathcal{C}}_k$, the L_2 error propagates exactly: $\|\tilde{\mathcal{C}}_k - \mathcal{C}_k\| = \|\tilde{\mathcal{X}} - \mathcal{X}\|$. That principle is very often exploited in transform compression schemes, including low-rank truncation (Ballester-Ripoll, Suter, and Pajarola 2015) or TTHRESH's entropy coding.

Similarly to TTHRESH, we encode the core one bit plane at a time using run-length encoding followed by entropy coding. The last bit plane may be encoded partially (see Fig. 3). Since we use double floating-point precision, there are 64 bit planes and therefore $64S + 1$ possible choices for the compression level c_k. Encoding no bits would lead to a zero tensor ($b_k(0) = 0, \epsilon_k(0) = 1$), while encoding all $64S$ bits would lead to perfect reconstruction ($b_k(64S + 1) = 1, \epsilon_k(64S + 1) = 0$).

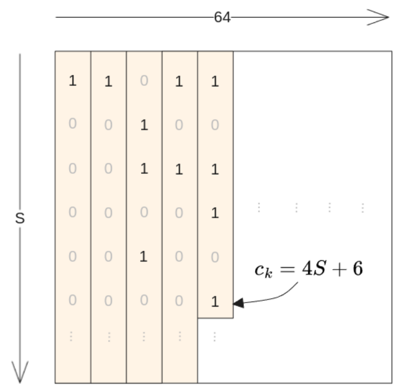


Figure 3: Bit plane encoding for a core $\mathcal{C}_k$, here shown as being flattened into a vector of S elements (rows). The columns denote coefficient bits, ordered from more to less significant. In this example, four planes are encoded completely, whereas only six bits were encoded from the fifth plane until hitting breakpoint c_k.

In practice, we build the curve progressively: we define a set of checkpoints $\{c_{k_1}, \ldots, c_{k_M}\} \subseteq [0, 64S + 1]$, and we compute $(b_k(c_{k_m}), \epsilon_k(c_{k_m}))$ for every c_{k_m}. In this paper we use $M = 64$ checkpoints with each one corresponding to one full bit plane, but a more granular set could be chosen instead.

Global Breakpoint Selection

Once we have a local curve for each core, we proceed to select an optimal vector of breakpoints $\boldsymbol{c}$. To this end, we need to define a global score function. The total amount of bits is the sum of bits spent on each core: $b(\boldsymbol{c}) = \sum_k b_k(c_k)$. In contrast, the global compression error does not propagate additively (see Grasedyck 2010 for bounds when applying successive SVD truncations), but we found $\epsilon(\boldsymbol{c}) := \sum_k \epsilon_k(c_k)$ to be a useful approximation in practice, as it can be tackled by a linear solver.

For each core $\mathcal{C}_k$, we:

1. Calculate points $(b_k(c_{k_m}), \epsilon_k(c_{k_m}))$ as described in the previous subsection.
2. Calculate the Pareto frontier (lower convex hull) of these points, which forms a decreasing piecewise function comprising a number N_k of segments.

3. Turn the frontier's segments into a set of linear constraints

$$\begin{cases} \epsilon_k \geq \alpha_k^{(1)} b_k + \beta_k^{(1)} \\ \dots \\ \epsilon_k \geq \alpha_k^{(N_k)} b_k + \beta_k^{(C_k)}. \end{cases}$$

We then use a linear programming optimizer to find $(\boldsymbol{b}^*, \boldsymbol{\epsilon}^*)$ that minimize $\sum_k b_k$, subject to $\sum_k \epsilon_k \leq \bar{\epsilon}$ and to the constraints above. Last, we use linear interpolation to recover the desired breakpoints, namely c_k such that $b_k(c_k) \approx b_k^*$ for each k.

By working with the convex hull, we are able to approximate the difficult initial optimization problem as a linear program with a few hundred constraints, which converges very quickly in practice (< 0.01s in our experiments).

Decompression

For reconstruction we undo the traversal outlined in the previous section. Since each core C_k has been encoded in canonical form, we ensure that the network is canonized at the position where the decoded C_k is going to be inserted:

1. $\mathcal{C}_k :=$ dummy for $k = 1, \dots, K$.
2. Traverse the tree recursively in the reverse order that was used in the compression step.
3. At each step of the recursion, decode core k and store the result in place of $\mathcal{C}_k$.
4. Isometrize $\mathcal{C}_k$ towards the central core $\mathcal{C}_i$. Ignore any changes that would affect dummy cores.
5. After the recursion is completed, reconstruct the tensor by contracting all virtual indices, so that only the physical indices (free edges in the tensor network) survive.

See Figure 4 for a diagram.

Experiments

We implemented our method and the following experiments in Python. We use the libraries *quimb* (Gray 2018) for the SVD orthogonalization steps during compression and *cotengra* (Gray and Kourtis 2021) for efficient tensor contraction during decompression, as well as for selecting the most central node i. For entropy coding we call the *constriction* package (Bamler 2022) and its `RangeEncoder`. We chose `scipy.spatial` for convex hull computation and `scipy.optimize.linprog` with its linear solver HIGHS to select the optimal breakpoint c_k of each core. We have publicly released our code[1], which can be called either as a Python API or via a command-line interface.

We tested 12 datasets which are divided in three types: (a) three 5D analytical functions, with and without noise; (b) seven datasets (4D and 5D) obtained from physical simulations and sensing measurements; and (c) two of the largest convolutional layers of ResNet-18 and VGG16, both 4D.

We have run our compressor multiple times for each dataset, each time at a different target relative error $\bar{\epsilon}$. We tested four tensor formats: Tucker (in which case the algorithm is almost identical to TTHRESH's), TT, extended

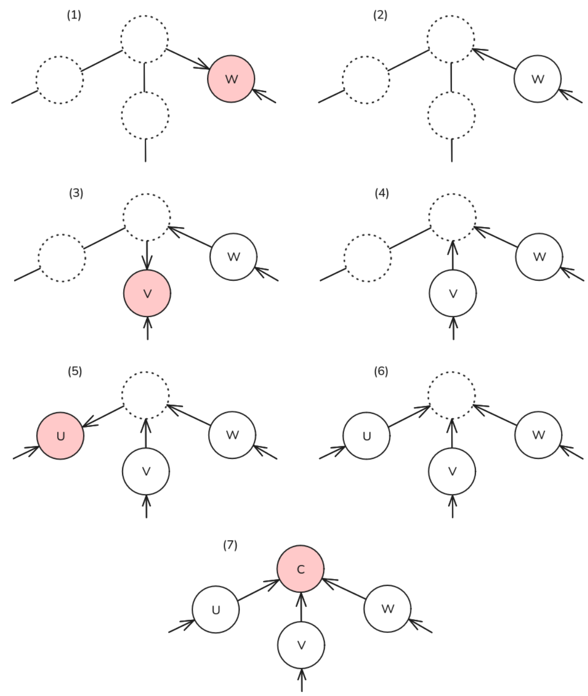


Figure 4: Decompression for a 3D Tucker example: the tree is traversed in reverse order to Fig. 2. At each step, a canonical core is decoded (in red) and then isometrized towards $\mathcal{C}$'s position. Dashed circles indicate dummy placeholders.

TT, and QTT. Resulting compression ratios and RMSEs are shown in Fig. 5. TT often outperforms the other methods by a significant margin, while QTT consistently lags behind.

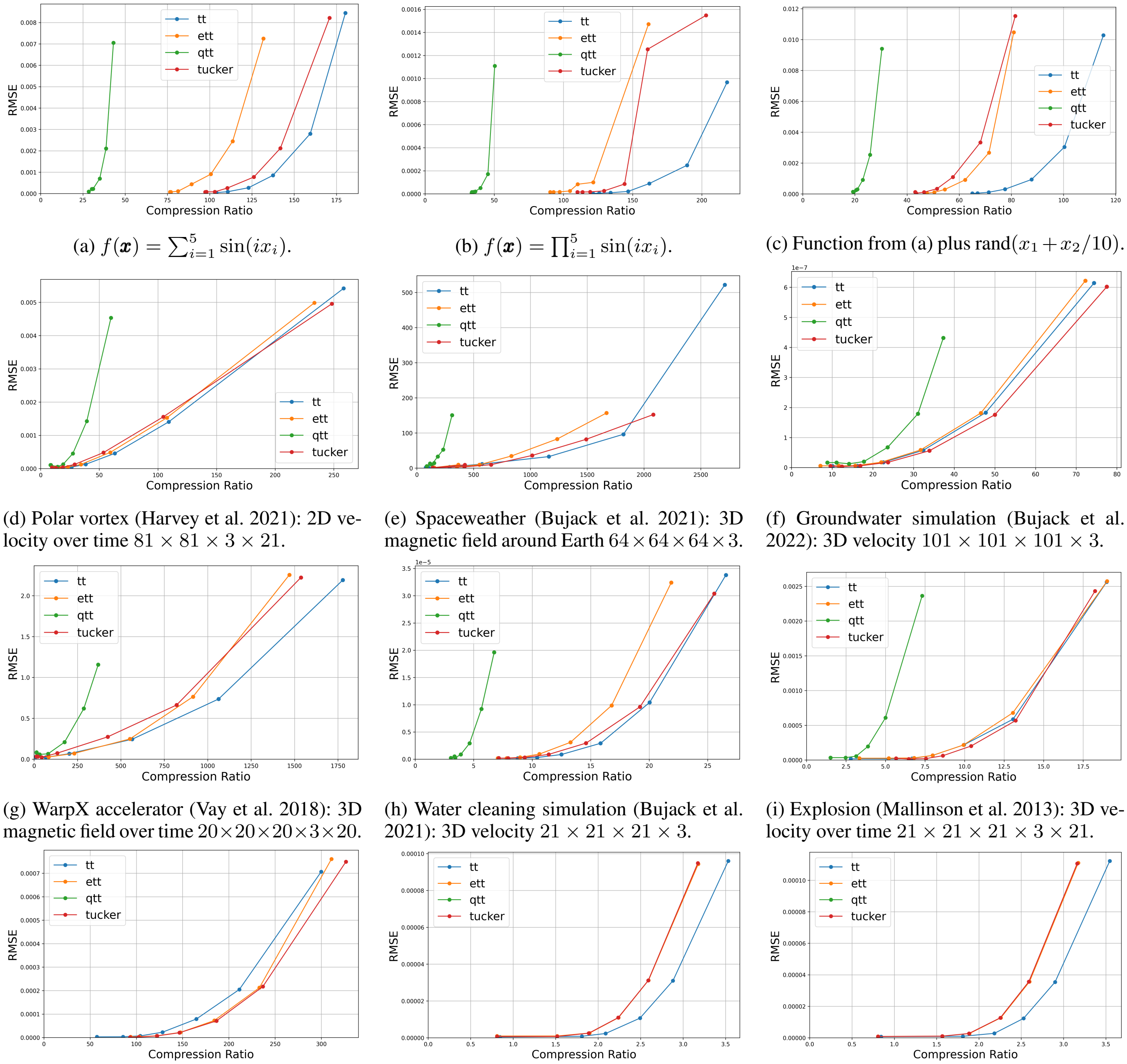
Discussion and Conclusion

This paper generalizes the entropy coding pipeline used in methods such as TTHRESH and ATC, which could only leverage the Tucker decomposition, to more general tensor networks. We showed that alternative tensor network topologies, particularly the tensor train, are often a significantly better ansatz than the Tucker decomposition, yielding up to 50% higher compression factors for comparable RMSE.

We believe our algorithm's main strength is its flexibility, as it can be applied on arbitrary tree networks and thus aim for very competitive rate-distortion curves across a wide range of tensors. Of course, no single network topology is a single silver bullet, and each tensor may benefit differently from different topologies. In the future, we will explore heuristics for automatically selecting a topology and dimension ordering that are most favorable for the target tensor. A key component to do this efficiently could be the estimation of the approximation error via random subsets of the tensor, see e.g. (Hayashi and Yoshida 2017).

While this paper only considered bit encoding schemes, these can be combined with low-rank truncation. We expect such a hybrid method can be significantly faster, at the expense of a less favorable rate-distortion trade-off. Last, note that our method is tailored to acyclic tensor networks with-

[1] https://github.com/rballester/pytthresh



(a) $f(\boldsymbol{x}) = \sum_{i=1}^{5} \sin(ix_i)$.

(b) $f(\boldsymbol{x}) = \prod_{i=1}^{5} \sin(ix_i)$.

(c) Function from (a) plus rand$(x_1 + x_2/10)$.

(d) Polar vortex (Harvey et al. 2021): 2D velocity over time $81 \times 81 \times 3 \times 21$.

(e) Spaceweather (Bujack et al. 2021): 3D magnetic field around Earth $64 \times 64 \times 64 \times 3$.

(f) Groundwater simulation (Bujack et al. 2022): 3D velocity $101 \times 101 \times 101 \times 3$.

(g) WarpX accelerator (Vay et al. 2018): 3D magnetic field over time $20 \times 20 \times 20 \times 3 \times 20$.

(h) Water cleaning simulation (Bujack et al. 2021): 3D velocity $21 \times 21 \times 21 \times 3$.

(i) Explosion (Mallinson et al. 2013): 3D velocity over time $21 \times 21 \times 21 \times 3 \times 21$.

(j) Riemannization of CIEDE2000 (Bujack et al. 2023): 3D metric over 2D plane matrix field $256 \times 256 \times 3 \times 3$.

(k) Convolutional layer of VGG16 (Qassim, Verma, and Feinzimer 2018): weights of shape $512 \times 512 \times 3 \times 3$.

(l) Convolutional layer of ResNet-18 (Sarwinda et al. 2021): weights of shape $512 \times 512 \times 3 \times 3$.

Figure 5: Error vs compression ratio for 12 datasets over four topologies. Synthetic datasets have shape 10^5 where each x_i takes 10 evenly spaced values in $[-1, 1]$.

out hyperedges. Allowing cycles or hyperedges could be numerically more delicate and may outstretch the assumptions we make in this paper about error propagation when calculating and distorting the cores. Still, that could also deserve future study.

Acknowledgments

This work was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory project number 20250145ER and published under LA-UR-24-33383.

References

Baert, W.; and Vannieuwenhoven, N. 2021. ATC: An Advanced Tucker Compression Library for Multidimensional Data. *ACM Transactions on Mathematical Software*, 49(21): 1–25.

Ballard, G.; Klinvex, A.; and Kolda, T. G. 2020. TuckerMPI: A Parallel C++/MPI Software Package for Large-scale Data Compression via the Tucker Tensor Decomposition. *ACM Transactions on Mathematical Software*, 46(2): 1–31.

Ballester-Ripoll, R.; Lindstrom, P.; and Pajarola, R. 2020. TTHRESH: Tensor Compression for Multidimensional Visual Data. *IEEE Transactions on Visualization and Computer Graphics*, 26(9): 2891–2903.

Ballester-Ripoll, R.; Suter, S. K.; and Pajarola, R. 2015. Analysis of Tensor Approximation for Compression-Domain Volume Visualization. *Computers & Graphics*, 47: 34–47.

Bamler, R. 2022. Understanding Entropy Coding With Asymmetric Numeral Systems (ANS): a Statistician's Perspective. *arXiv preprint arXiv:2201.01741*.

Bujack, R.; Bresciani, E.; Waters, J.; and Schroeder, W. 2022. Topological Segmentation of 2D Vector Fields. In *Leipzig Symposium on Visualization In Applications (LEVIA)*.

Bujack, R.; Caffrey, E.; Teti, E.; Turton, T. L.; Rogers, D. H.; and Miller, J. 2023. Efficient Computation of Geodesics in Color Space. *IEEE Transactions on Visualization and Computer Graphics*.

Bujack, R.; Tsai, K.; Morley, S. K.; and Bresciani, E. 2021. Open source vector field topology. *SoftwareX*, 15: 100787.

de Lathauwer, L.; de Moor, B.; and Vandewalle, J. 2000. A Multilinear Singular Value Decomposition. *SIAM Journal of Matrix Analysis and Applications*, 21(4): 1253–1278.

Di, S.; and Cappello, F. 2016. Fast Error-Bounded Lossy HPC Data Compression with SZ. In *International Parallel and Distributed Processing Symposium*, 730–739.

Grasedyck, L. 2010. Hierarchical Singular Value Decomposition of Tensors. *SIAM Journal of Matrix Analysis and Applications*, 31(4): 2029–2054.

Gray, J. 2018. quimb: a Python library for quantum information and many-body calculations. *Journal of Open Source Software*, 3(29): 819.

Gray, J.; and Kourtis, S. 2021. Hyper-optimized tensor network contraction. *Quantum*, 5: 410.

Harvey, V. L.; Datta-Barua, S.; Pedatella, N. M.; Wang, N.; Randall, C. E.; Siskind, D. E.; and van Caspel, W. E. 2021. Transport of nitric oxide via Lagrangian coherent structures into the top of the polar vortex. *Journal of Geophysical Research: Atmospheres*, 126(11): e2020JD034523.

Hayashi, K.; and Yoshida, Y. 2017. Fitting Low-Rank Tensors in Constant Time. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Khoromskij, B.; and Oseledets, I. 2010. Quantics-TT Collocation Approximation of Parameter-Dependent and Stochastic Elliptic PDEs. *Computational Methods in Applied Mathematics*, 10(4): 376–394.

Lindstrom, P. 2014. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization & Computer Graphics*, 20(12): 2674–2683.

Lu, Y.; Jiang, K.; Levine, J. A.; and Berger, M. 2021. Compressive Neural Representations of Volumetric Scalar Fields. *Computer Graphics Forum*, 40(3): 135–146.

Mallinson, A.; Beckingsale, D. A.; Gaudin, W.; Herdman, J.; Levesque, J.; and Jarvis, S. A. 2013. Cloverleaf: Preparing hydrodynamics codes for exascale. *The Cray User Group*, 2013.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. 2015. Tensorizing Neural Networks. In *Advances in Neural Information Processing Systems*.

Oseledets, I.; and Tyrtyshnikov, E. 2011. Algebraic Wavelet Transform via Quantics Tensor Train Decomposition. *SIAM Journal on Scientific Computing*, 33(3): 1315–1328.

Oseledets, I. V. 2011. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317.

Qassim, H.; Verma, A.; and Feinzimer, D. 2018. Compressed residual-VGG16 CNN model for big data places image recognition. In *2018 IEEE 8th annual computing and communication workshop and conference (CCWC)*, 169–175. IEEE.

Sarwinda, D.; Paradisa, R. H.; Bustamam, A.; and Anggia, P. 2021. Deep learning in image classification using residual network (ResNet) variants for detection of colorectal cancer. *Procedia Computer Science*, 179: 423–431.

Schneider, R.; and Uschmajew, A. 2014. Approximation rates for the hierarchical tensor format in periodic Sobolev spaces. *Journal of Complexity*, 30(2): 56–71. Dagstuhl 2012.

Vay, J.-L.; Almgren, A.; Bell, J.; Ge, L.; Grote, D.; Hogan, M.; Kononenko, O.; Lehe, R.; Myers, A.; Ng, C.; et al. 2018. Warp-X: A new exascale computing platform for beam–plasma simulations. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 909: 476–479.