TensorCodec: Compact Lossy Compression of Tensors without Strong Data Assumptions (Supplementary Document)

If the preview is not legible, kindly proceed to download the PDF file.

I. Proofs of Theorems

Lemma 1 (Speed of Order Initialization)**.** *Initializing all reordering functions $\boldsymbol{\pi}$ (i.e., Algorithm 4 in Section II) takes $O((\sum_{i=1}^{d} N_i) \cdot (\prod_{i=1}^{d} N_i))$ time.*

Proof. For each k-th mode, forming the complete graph $G = (V, E)$ takes $O(N_k \prod_{i=1}^{d} N_i)$ time, and obtaining a 2-approximation solution from the complete graph takes $O(|V|^2) = O(N_k^2)$ time (refer to Section II for details). Therefore, initializing π_k takes $O(N_k^2 + N_k \prod_{i=1}^{d} N_i) = O(N_k \prod_{i=1}^{d} N_i)$ time, and initializing all reordering functions $\boldsymbol{\pi}$ takes $O((\sum_{i=1}^{d} N_i) \cdot (\prod_{i=1}^{d} N_i))$ time. □

Lemma 2 (Speed of Order and Model Updates)**.** *Updating θ and π once (i.e., lines 4-5 of Algorithm 1) takes $O(d(d+h^2+hR^2)\prod_{i=1}^{d} N_i \log N_{\max})$ time.*

Proof. Updating θ involves approximation of all entries in the input tensor, which requires $O((d + h^2 + hR^2)(\prod_{i=1}^{d} N_i) \log N_{\max})$ time by Theorem 3. When updating $\boldsymbol{\pi}$ by Algorithm 3, computing the loss on every entry of $\mathcal{X}$ dominates the other steps, and its complexity for all d modes is $O(d(d+h^2+hR^2)(\prod_{i=1}^{d} N_i) \log N_{\max})$, which determines the total complexity. □

Theorem 3 (Compression Speed)**.** *The time complexity of* TensorCodec *(i.e., Algorithm 1) with T update steps is $O((Td(d+h^2+hR^2)\log N_{\max} + \sum_{i=1}^{d} N_i)\prod_{i=1}^{d} N_i)$, where $\prod_{i=1}^{d} N_i$ is the number of tensor entries.*

Proof. The theorem follows from Lemmas 1 and 2. □

Theorem 4 (Memory Requirements for Compression)**.** TensorCodec *(Algorithm 1) requires $O(Bd + h(2^d + B(h + R^2)\log N_{\max}) + \sum_{i=1}^{d} N_i)$ memory space.*

Proof. Storing each mini-batch requires memory of size $O(Bd)$, and storing the current the model θ of NTTD requires $O(h(2^d + h + R^2))$ by Theorem 2. Saving the orders of all modes requires $O(\sum_{i=1}^{d} N_i)$. The operations in LSTM, the embedding layer, and the fully connected layer consume $O(hB(h + R^2)\log N_{\max})$ memory during the inference and backpropagation of the model. The other parts are dominated by the parts above. In conclusion, the memory required for compression is $O(Bd + h(2^d + B(h + R^2)\log N_{\max}) + \sum_{i=1}^{d} N_i)$. □

Algorithm 4: Initialization of reordering functions $\boldsymbol{\pi}$

Input: a tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$
Output: reordering functions $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_d)$

for $k \leftarrow 1$ to d **do**
  // Construct a complete graph
  for $j \leftarrow 0$ to $N_k - 1$, $h \leftarrow j+1$ to $N_k - 1$ **do**
    $w(j,h), w(h,j) \leftarrow \|\mathcal{X}^{(i)}(j) - \mathcal{X}^{(i)}(h)\|_F$
  // Initialize a tree T
  $T.root \leftarrow$ {a randomly sampled integer from $[N_k]$}
  $U \leftarrow [N_k] \setminus T$.nodeset
  // Run Prim's algorithm
  while $|U| > 0$ **do**
    $j, h \leftarrow \text{argmin}_{j \in T, h \in U} w(j,h)$
    $T.j$.child $\leftarrow T.j$.child $\cup \{h\}$
    $U \leftarrow U \setminus \{h\}$
  $P \leftarrow \text{DFS}(T)$ ▷ Depth first search at T
  $P[N_k] \leftarrow P[0]$
  // Remove the edge with the largest weight
  $j \leftarrow \text{argmax}_{j \in [N_k]} w(P[j], P[j+1])$
  $\pi_k \leftarrow \texttt{concat}(P[j+1{:}N_k], P[0{:}j+1])$
return $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_d)$

II. Traveling Salesman Problem and Order Initialization

We introduce the background of the traveling salesman problem (TSP) [1] which is related to formulating the initialization of orders of an input tensor in our model (see details in Section IV-D). Given a complete loopless undirected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}^+$ that maps each edge to a non-negative real value, the TSP problem aims to find a Hamiltonian cycle H that visits each node exactly once such that $w(H) = \sum_{e \in H} w(e)$ is minimized where V and E are the sets of nodes and edges, respectively.

If the weight function w satisfies the triangle inequality ($w(u,v) \leq w(u,x) + w(x,v), \forall u, v, x \in V$), the problem is called Metric TSP [2].

As Metric TSP belongs to NP-complete, we need approximation algorithms to solve within a certain factor of the optimal answer in polynomial time. In this work, we consider a 2-approximation algorithm for Metric TSP as follows. It first computes a minimum spanning tree (MST) T of G and obtains an Eulerian multigraph T' by duplicating each edge of T. After then, it computes an Eulerian tour of T', which is the

TABLE III
HYPERPARAMTERS OF TENSORCODEC AND COMPETITORS USED IN SECTION 5.2 WHERE $[\![1, N]\!]$ IS THE INTEGER INTERVAL FROM 1 TO N.

Method	Hyperparameter	Stock	Airquality	PEMS-SF	Activity	Action	Uber	Absorb	NYC
TENSORCODEC	(TT rank, hidden size of LSTM)	(7, 3), (5, 7), (8, 8), (8, 10)	(7, 11), (7, 20), (8, 25), (17, 21)	(6, 5), (5, 9), (7, 11), (14, 8)	(8, 6), (12, 8), (9, 13), (11, 15)	(6, 8), (11, 9) (13, 11), (13 ,14)	(7, 8), (7, 8), (7, 13), (12, 13), (12, 16)	(6, 4), (4, 8), (6, 10), (8, 11)	(2, 5), (5, 7), (6, 9), (10, 9)
	Learning Rate	1	0.1	1	1	1	0.1	1	0.1
	Tolerance	100	100	100	10	10	100	10	100
	Maximum Epochs	500	1400	1000	5000	5000	5000	5000	1500
NeuKron	hidden size of LSTM	5, 8, 11, 14	14, 23, 29, 35	7, 11, 14, 17	10, 15, 19, 23	10, 15, 19, 23	11, 16, 21, 24	5, 8, 11, 14	5, 8, 11, 14
	Learning Rate	10	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Tolerance	100	100	100	10	10	100	10	100
	Maximum Epochs	500	1400	1000	5000	5000	5000	5000	1500
CPD	Rank	$[\![1, 5]\!]$	$[\![1, 5]\!]$	$[\![1, 4]\!]$	$[\![1, 7]\!]$	$[\![1, 9]\!]$	$[\![1, 6]\!]$	$[\![1, 10]\!]$	$[\![1, 8]\!]$
TKD		$[\![1, 5]\!]$	$[\![1, 5]\!]$	$[\![1, 4]\!]$	$[\![1, 7]\!]$	$[\![1, 8]\!]$	$[\![1, 5]\!]$	$[\![1, 7]\!]$	$[\![1, 5]\!]$
TTD		$[\![1, 5]\!]$	$[\![1, 4]\!]$	$[\![1, 4]\!]$	$[\![1, 5]\!]$	$[\![1, 5]\!]$	$[\![1, 5]\!]$	$[\![1, 4]\!]$	$[\![1, 4]\!]$
TRD	Target Accuracy	1.4, 1.3, 1.2, 1.1, 1.05	0.6, 0.55, 0.5, 0.46	0.7, 0.65 0.6, 0.57	0.7, 0.6 0.5, 0.4	0.6, 0.47, 0.43, 0.395	0.6, 0.45, 0.41	0.9, 0.8 0.72, 0.6	0.8, 0.6, 0.5 0.42, 0.41
TTHRESH	Target Relative Error	0.94, 0.92, 0.9 0.85, 0.83, 0.8	0.4, 0.37, 0.35 0.32, 0.315, 0.3	0.48, 0.47, 0.46 0.45, 0.44, 0.43	0.57, 0.55, 0.54 0.53, 0.52	0.45, 0.43, 0.41 0.4, 0.39, 0.38	0.45, 0.43 0.4, 0.38	0.7, 0.65, 0.6 0.6, 0.55, 0.53	0.6, 0.5, 0.45 0.43, 0.42, 0.4
SZ3	Absolute Error Bound	$2 \times 10^{16}, 1.4 \times 10^{16}$, $1.15 \times 10^{16}, 10^{16}$	30, 28, 26, 24, 22	0.18	0.8, 0.7, 0.6	0.9, 0.8, 0.7, 0.6, 0.5	11, 10, 9, 8, 7	100, 50, 30	45, 40

TABLE IV
THE MODES OF REAL-WORLD TENSORS THAT ARE REORDERED BY THE ORDER INITIALIZATION METHOD OF TENSORCODEC.

Uber	Airquality	Action	PEMS-SF	Activity	Stock	NYC	Absorb
1	2, 3	2, 3	1, 3	2, 3	1, 2, 3	1, 2, 4	4

2-approximation solution. The running time of this algorithm is dominated by the computing process of MST which can be earned by Prim's algorithm [3] whose time complexity is $O(|V|^2)$ when the given graph is complete.

Algorithm 4 describes the detailed process of initializing the reordering functions $\boldsymbol{\pi}$ using the Metric TSP problem formulation and the 2-approximation solution, which is outlined in Section IV-D.

III. SEMANTICS OF DATA

`Uber`[1] [4] stores the counts of Uber pickups in New York City, and it is in the form of (date, hour, latitude; count). `Air Quality` [5] is air pollutants information consisting of (hour, location, pollutant; measurement). `Action` and `Activity`[2] [6], [7] are composed of features from motion videos, and they have the form of (frame, feature, video; value). `PEMS-SF`[3] [8] is composed of the rates of use of car lanes in San Francisco bay area with the form of (station, timestamp, day; measurement). `Stock` [9] is the collection of stocks in the form of (time, feature, stock; value). `NYC`[4] is the trip record of yellow and green taxis in New York City from 2020 to 2022 of which the form is (pick up zone, drop off zone, day, month; count). `Absorb`[5] contains aerosol absorption, and its format is (longitude, latitude, altitude, time; measurement).

[1]http://frostt.io/
[2]https://github.com/titu1994/MLSTM-FCN
[3]http://www.timeseriesclassification.com/
[4]https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page
[5]https://www.earthsystemgrid.org/

IV. DETAILED EXPERIMENTAL SETTING

We used the ALS implementations in Tensor Toolbox [10] for CPD and TKD.[6] We used the TT-SVD implementation in TT-toolbox [12] after slight modification [13] for unifying the TT ranks of all TT cores. We used the TR-SVD [14] implementation[7] for TRD.

TENSORCODEC and NeuKron are trained by the Adam optimizer. For CPD and TKD, which employ Alternating Least Squares, we terminate the optimization when the increase in fitness is less than 10^{-4}. To ensure fairness in comparison, we do not apply the final lossless compression step in SZ3, as other methods could also potentially benefit from the same lossless compression technique. For the methods optimized using gradient descent, (i.e., TENSORCODEC and NeuKron), we stopped optimization when the increase in fitness was less than 10^{-4} for a specific number of epochs (i.e., tolerance) or when the number of epochs exceeded a certain threshold. We adjusted learning rates by a factor of 10 and selected the maximum values that did not result in divergence during optimization.

For real-world tensors used in Section 5, we initially assign 2 to $n_{k,l}$ for all $k \in \{1, \cdots, d\}$ and $l \in \{1, \cdots, d'\}$ and modify some of them using integers at most 5 to ensure that both the input and folded tensors possess similar numbers of entries. The assigned values in the form of a d by d' matrix for each tensor are as follows:

- `Uber`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 3 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 5 \end{bmatrix}$$

[6]Faster one between two implementations for dense and sparse tensors [11].
[7]https://github.com/oscarmickelin/tensor-ring-decomposition

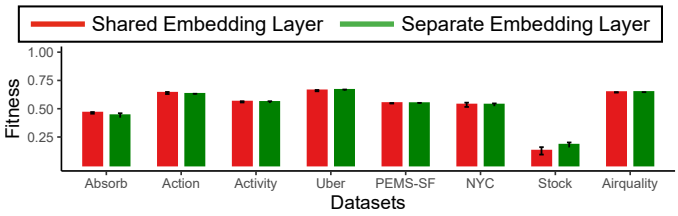

Fig. 10. The Fitness comparison between our models with different embedding layers. The fitness of our model does not significantly change when the design of the embedding layers is changed.

- `Airquality`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 3 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- `Action`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 5 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 5 \end{bmatrix}$$

- `PEMS-SF`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 5 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \end{bmatrix}$$

- `Activity`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 5 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 3 \end{bmatrix}$$

- `Stock`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

- `NYC`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 1 & 1 \\ 2 & 2 & 3 & 3 & 1 & 1 & 1 \end{bmatrix}$$

- `Absorb`:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 3 \\ 2 & 2 & 2 & 2 & 2 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

The hyperparameter settings of the algorithms used in Section 5.2 are provided in Table III. The sizes of tensors used in Section 5.4 are given in Table V. The modes of the tensor where the order of indices is initialized by our method are provided in Table IV.

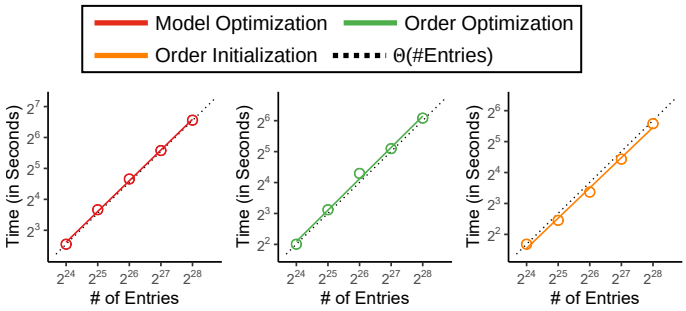

Fig. 11. The compression time of TENSORCODEC scales near linearly with the number of entries in the input tensor. The figure represents the result achieved using a 4-order tensor.

TABLE V
SIZES OF TENSORS USED IN FIGURE 5 OF SECTION 5.4 OF THE MAIN PAPER.

Order	Size of Tensor	Order	Size of Tensor
3	$256 \times 256 \times 128$	4	$64 \times 64 \times 64 \times 64$
	$256 \times 256 \times 256$		$128 \times 64 \times 64 \times 64$
	$512 \times 256 \times 256$		$128 \times 128 \times 64 \times 64$
	$512 \times 512 \times 256$		$128 \times 128 \times 128 \times 64$
	$512 \times 512 \times 512$		$128 \times 128 \times 128 \times 128$

V. EXPERIMENTS ON THE DESIGN OF EMBEDDING LAYERS

The comparison of the Fitness of the model when using shared embedding layers and separate embedding layers is in Figure 10. We set the TT rank R and hidden size h to the settings of Section 5.3 for `Action`, `Airquality`, `PEMS-SF`, and `Uber` datasets. For the remaining datasets, R and h is set to the settings of the smallest compressed size in Section 5.2 (i.e. the first row of Table III).

The fitness when sharing embedding layers is higher in 5 datasets. There was no significant dependency of accuracy on the choice of the design of embedding layers. Thus, we choose to share embedding layers across different modes since the theoretical space complexity of this design is independent to the size of the input tensor.

VI. Q3. SCALABILITY: COMPRESSION SPEED

The size of tensors used for experiments are in Table V. The compression time of TENSORCODEC also increases near linearly with the number of entries in the input tensor when the order of tensor is 4, as illustrated in Figure 11.

VII. Q6. EFFECT OF HYPERPARAMETERS

We investigated the sensitivity of TENSORCODEC to its hyperparameters, particularly the hidden dimension h and the TT-rank R. They affect the compressed size of TENSORCODEC as described in Theorem 1, and they also affect the reconstruction accuracy. For this experiment, we adjusted h and R so that each configuration produces outputs with nearly the same size, as shown in Table VI.

Given a similar budget for compressed size, TENSORCODEC exhibits insensitivity to hyperparameter settings, as shown in Table VI. Specifically, for each dataset, TENSORCODEC displays consistent fitness levels regardless of the R and h values.

TABLE VI

TENSORCODEC EXHIBITS INSENSITIVITY TO HYPERPARAMETER SETTINGS. TENSORCODEC DEMONSTRATES SIMILAR FITNESS LEVELS, IRRESPECTIVE OF THE TT-RANK R AND THE HIDDEN DIMENSION h OF LSTM, PROVIDED THAT THE COMPRESSED SIZE REMAINS NEARLY THE SAME.

Uber				Air Quality				Action				PEMS-SF			
R	h	Compressed Size (Bytes)	Fitness	R	h	Compressed Size (Bytes)	Fitness	R	h	Compressed Size (Bytes)	Fitness	R	h	Compressed Size (Bytes)	Fitness
1	11	12,702	0.6637±0.0025	1	14	25,886	0.6421±0.0009	1	10	9,944	0.6163±0.0059	1	7	7,523	0.5462±0.0045
4	10	12,918	0.6638±0.0035	3	13	25,230	0.6429±0.0019	4	9	10,056	0.6389±0.0055	4	6	7,507	0.5467±0.0023
5	9	12,126	0.6636±0.0036	7	11	26,030	**0.6442±0.0021**	6	8	10,176	**0.6432±0.0063**	6	5	7,427	**0.548±0.0018**
7	8	12,510	**0.6647±0.006**	9	9	24,846	0.6435±0.0017	7	7	9,464	0.6378±0.0071	8	4	7,411	0.5416±0.0011
9	7	13,086	0.6602±0.0058	12	7	25,134	0.6438±0.0026	9	6	9,816	0.6361±0.0025	10	3	7,267	0.5428±0.0038
10	6	12,374	0.6569±0.0043	16	5	26,174	0.6427±0.0009	11	5	10,104	0.6312±0.0072	13	2	7,451	0.5416±0.0034

Although the gap is small, the fitness of TENSORCODEC tends to marginally decrease as the difference between R and h increases. This is because if R is significantly larger than h, the expressiveness of LSTM may be limited, whereas the expressiveness of TT cores can be restricted if R is considerably smaller than h. Consequently, TENSORCODEC typically demonstrates better fitness when the values of R and h are balanced, as evidenced in Table VI.

REFERENCES

[1] K. L. Hoffman, M. Padberg, G. Rinaldi *et al.*, "Traveling salesman problem," *Encyclopedia of operations research and management science*, vol. 1, pp. 1573–1578, 2013.

[2] M.-Y. Kao, *Encyclopedia of algorithms*. Springer Science & Business Media, 2008.

[3] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[4] (2017) FROSTT: The formidable repository of open sparse tensors and tools. [Online]. Available: http://frostt.io/

[5] J.-G. Jang and U. Kang, "D-tucker: Fast and memory-efficient tucker decomposition for dense tensors," in *ICDE*, 2020.

[6] J. Wang, Z. Liu, Y. Wu, and J. Yuan, "Mining actionlet ensemble for action recognition with depth cameras," in *CVPR*, 2012.

[7] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate lstm-fcns for time series classification," *Neural Networks*, vol. 116, pp. 237–245, 2019.

[8] M. Cuturi, "Fast global alignment kernels," in *ICML*, 2011.

[9] J.-G. Jang and U. Kang, "Fast and memory-efficient tucker decomposition for answering diverse time range queries," in *KDD*, 2021.

[10] (2023) Tensor toolbox for matlab v. 3.5. [Online]. Available: https://tensortoolbox.org/

[11] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SISC*, vol. 30, no. 1, pp. 205–231, 2008.

[12] (2022) Tt-toolbox v. 2.2.2. [Online]. Available: https://github.com/oseledets/TT-Toolbox

[13] C. Yin, D. Zheng, I. Nisa, C. Faloutsos, G. Karypis, and R. Vuduc, "Nimble gnn embedding with tensor-train decomposition," in *KDD*, 2022.

[14] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, "Tensor ring decomposition," *arXiv*, 2016.