PuzzleTensor: A Method-Agnostic Data Transformation for Compact Tensor Factorization

Yong-chan Park
Seoul National University
Seoul, Republic of Korea
wjdakf3948@snu.ac.kr

Kisoo Kim
Seoul National University
Seoul, Republic of Korea
kisooofficial@snu.ac.kr

U Kang
Seoul National University
Seoul, Republic of Korea
ukang@snu.ac.kr

Abstract

How can we achieve compact tensor representations without sacrificing reconstruction accuracy? Tensor decomposition is a cornerstone of modern data mining and machine learning, enabling efficient representations of multi-dimensional data through fundamental algorithms such as CP, Tucker, and Tensor-Train decompositions. However, directly applying these methods to raw data often results in high target ranks, poor reconstruction accuracy, and computational inefficiencies, as the data may not naturally conform to the low-rank structures these methods assume.

In this paper, we propose PuzzleTensor, a method-agnostic data transformation technique for compact tensor factorization. Given a data tensor, PuzzleTensor "solves the puzzle" by shifting each hyperslice of the tensor to achieve accurate decompositions with significantly lower target ranks. PuzzleTensor offers three key advantages: (1) it is independent of specific decomposition methods, making it seamlessly compatible with various algorithms, such as CP, Tucker, and Tensor-Train decompositions; (2) it works under weak data assumptions, showing robust performance across both sparse and dense data, regardless of the rank; (3) it is inherently explainable, allowing clear interpretation of its learnable parameters and layer-wise operations. Extensive experiments show that PuzzleTensor consistently outperforms direct tensor decomposition approaches by achieving lower reconstruction errors and reducing the required target rank, making it a versatile and practical tool for compact tensor factorization in real-world applications.

CCS Concepts

• **Computing methodologies → Factorization methods**.

Keywords

Tensor decomposition, Low-rank approximation, Data compression

ACM Reference Format:
Yong-chan Park, Kisoo Kim, and U Kang. 2025. PuzzleTensor: A Method-Agnostic Data Transformation for Compact Tensor Factorization. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3711896.3737095

1 Introduction

Efficient high-dimensional data representations are critical for numerous applications, including data mining [5, 11, 13, 14, 27, 41, 46], machine learning [10, 22, 28, 44], recommender systems [15, 16, 20, 21, 26], signal processing [29, 45], and scientific computing [18, 38, 40, 43]. Tensor decomposition algorithms—such as CANDECOMP/PARAFAC (CP) [19], Tucker [8], and Tensor-Train (TT) [31]—have emerged as essential tools in these fields, offering computational efficiency and insights into underlying data structures. However, directly applying these methods to raw multi-dimensional data often leads to high target ranks, poor reconstruction accuracy, or computational inefficiencies, as real-world tensors rarely conform to the strict low-rank or sparsity assumptions that many decomposition techniques depend upon [4, 35, 37, 39].

As an illustration, consider Figure 1, where X denotes the original data and $\hat{X}$ is its rank-1 approximation, resulting in a relatively large error. Shifting each column of X upward by a certain amount leads to a better-aligned matrix Z, significantly reducing the rank-1 approximation error. A key insight is that the rank of a tensor is closely tied to how its slices and modes are spatially arranged. In practice, high rank often stems from misalignment or heterogeneity in the data, where local patterns do not align naturally across modes. This raises a fundamental question: how can we systematically transform the data so that common structures align naturally, allowing lower-rank factorizations without sacrificing accuracy?

In this paper, we propose PuzzleTensor, a method-agnostic data transformation technique designed for compact tensor factorization. Inspired by the concept of rearranging puzzle pieces to form a cohesive image, PuzzleTensor shifts the hyperslices of a tensor, transforming its structure to enable accurate decompositions with significantly reduced target ranks (see Figure 2). PuzzleTensor offers three distinct advantages that make it a practical tool for a wide range of real-world applications:

(1) **Method-agnostic flexibility.** Unlike methods tied to specific algorithms, PuzzleTensor integrates seamlessly with various decomposition frameworks, from CP and Tucker to TT, broadening its applicability across diverse domains.
(2) **Weak data assumptions.** PuzzleTensor achieves robust performance under minimal constraints, making it suitable for datasets with varying levels of sparsity and rank distributions.
(3) **Explainability.** Each learnable parameter and transformation layer in PuzzleTensor is interpretable, allowing users to understand how the method optimizes tensor representations for improved performance.

Extensive experiments on both real-world and synthetic datasets demonstrate the efficacy of PuzzleTensor. Compared to the direct

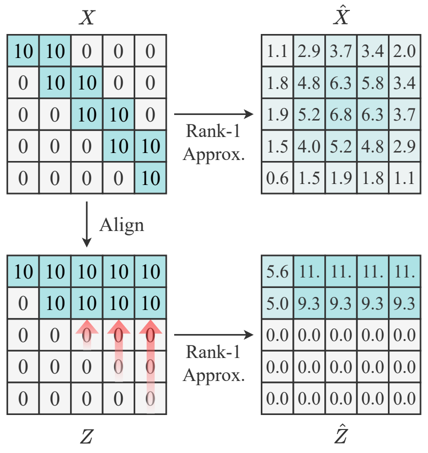


Figure 1: An example of how alignment improves low-rank approximation. Given raw data $\mathcal{X}$, its rank-1 approximation $\hat{\mathcal{X}}$ exhibits a large error. By aligning columns to form $\mathcal{Z}$, the rank-1 approximation $\hat{\mathcal{Z}}$ achieves significantly lower error. This shows that reorganizing the data reveals underlying low-rank structures that remain hidden in the raw form.

decomposition approaches, it consistently achieves lower reconstruction error across various tensor factorization methods, underscoring the versatility and robustness of PuzzleTensor as a practical tool for compact tensor representations.

We summarize our contributions as follows:

- **Method.** We present PuzzleTensor, a method-agnostic data transformation for compact tensor factorization.
- **Analysis.** We provide a theoretical analysis demonstrating that our proposed optimization technique induces a low-rank structure in the resulting tensor. Additionally, we derive the parameter count and time complexity of PuzzleTensor.
- **Performance.** PuzzleTensor consistently enhances the performance of various tensor decomposition methods, including CP, Tucker, and TT in terms of tensor compression.

We provide the source code and datasets used in our paper at **https://github.com/snudatalab/PuzzleTensor**.

2 Related Works

We provide an overview of foundational tensor decomposition methods, and explore how they have been utilized for data compression in various applications. We also highlight their limitations, motivating the need for more flexible and scalable approaches.

2.1 Overview of Tensor Decompositions

Tensor decomposition methods aim to express a high-dimensional tensor in a factorized form that reduces storage and computational complexity while preserving essential data characteristics. Below, we provide a brief explanation of key decomposition techniques with their respective mathematical formulations.

CP Decomposition. The CP decomposition factorizes an n-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$ into a sum of rank-1 tensors:

$$\mathcal{X} \approx \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(n)},$$

where R is the rank of the decomposition, $\boldsymbol{a}_r^{(k)} \in \mathbb{R}^{I_k}$ are the factor vectors for mode k, and $\circ$ denotes the outer product. In matrix form, the decomposition for each mode k can be expressed as:

$$\boldsymbol{X}_{(k)} \approx \boldsymbol{A}^{(k)} \left(\boldsymbol{A}^{(n)} \odot \cdots \odot \boldsymbol{A}^{(k+1)} \odot \boldsymbol{A}^{(k-1)} \odot \cdots \odot \boldsymbol{A}^{(1)} \right)^{\top},$$

where $\boldsymbol{X}_{(k)}$ is the mode-k unfolding of $\mathcal{X}$, $\boldsymbol{A}^{(k)}$ are the factor matrices, and $\odot$ denotes the Khatri-Rao product.

Tucker Decomposition. The Tucker decomposition generalizes CP by introducing a core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_n}$ and factor matrices $\boldsymbol{U}^{(k)} \in \mathbb{R}^{I_k \times J_k}$:

$$\mathcal{X} \approx \mathcal{G} \times_1 \boldsymbol{U}^{(1)} \times_2 \boldsymbol{U}^{(2)} \cdots \times_n \boldsymbol{U}^{(n)},$$

where $\times_k$ denotes the mode-k product between a tensor and a matrix. The core tensor captures the interaction between components across all modes, while the factor matrices reduce the dimensionality of each mode.

Tensor-Train Decomposition. The Tensor-Train (TT) decomposition represents an n-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$ as a chain of 3D tensors (cores) $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$:

$$\mathcal{X}(i_1, i_2, \ldots, i_n) = \mathcal{G}^{(1)}(:, i_1, :) \cdot \mathcal{G}^{(2)}(:, i_2, :) \cdots \mathcal{G}^{(n)}(:, i_n, :),$$

where $R_0 = R_n = 1$. In contrast to Tucker decomposition, where the number of parameters grows exponentially, TT decomposition reduces this complexity to linear scaling with n.

Each of these methods provides a valuable framework for interpreting multi-dimensional data, but they also come with inherent limitations. Issues such as rank selection, computational costs for large dimensions, and potential sensitivity to noise motivate ongoing research in more robust and scalable approaches.

2.2 Tensor Decomposition in Data Compression

This section reviews recent advancements and challenges in tensor-based lossy compression, emphasizing scalability, domain-specific applications, and generalization.

Lossy Compression with Scalability Improvements. Recent advancements in data compression techniques utilize tensor representations to handle the increasing complexity of multi-dimensional data, including multi-spectral images, videos, and scientific datasets. NeuKron [25] compresses sparse reorderable matrices into fixed-size space with Kronecker products and a recurrent neural network, achieving high accuracy and scalability. TensorCodec [24] further improves expressive power by employing Neural Tensor-Train Decomposition (NTTD), leveraging tensor folding to minimize space usage, and applying mode reordering to reveal exploitable patterns.

Both NeuKron and TensorCodec incorporate specialized pipelines (e.g., LSTMs, tensor folding, and axis reordering) tightly coupled to their decomposers. By contrast, PuzzleTensor is a method-agnostic transformation: it shifts hyperslices via a Fourier-based operation before any decomposition step. This shift lowers the effective rank, enabling both classic (CP, Tucker, TT, etc.) and neural methods to achieve comparable accuracy at significantly smaller target ranks.

Compression with Domain Applications. Numerous studies propose advanced techniques for compressing high-dimensional data. Aidini et al. [2] propose a tensor decomposition method that compresses multi-spectral signals by learning shared bases from training data. Ballester et al. [3] propose a lossy compression algorithm using Higher-Order Singular Value Decomposition (HOSVD) and methods like bit-plane, run-length, and arithmetic coding to efficiently compress multi-dimensional data. Adjeroh et al. [1] tackle video compression using 3D Discrete Cosine Transform, enhancing error protection and transmission efficiency over noisy channels.

Despite these advancements, challenges remain in adapting tensor decomposition methods to diverse real-world datasets. Numerous compression schemes depend on strict assumptions about data structure (e.g., low-rank or sparsity), which often fail in complex scenarios. Additionally, existing algorithms balance trade-offs between compression ratio, reconstruction accuracy, and interpretability. This emphasizes the need for more adaptable, generalizable solutions less tied to specific data characteristics.

2.3 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) [6, 30, 32–34] of a real or complex sequence $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1})$ of length N is defined as

$$\hat{x}_m = \mathcal{F}\{\boldsymbol{x}\}(m) = \sum_{n=0}^{N-1} x_n \, e^{-2\pi i m n/N}, \quad m = 0, \ldots, N-1.$$

This transformation maps the time (or spatial) domain sequence to its frequency-domain representation, thereby facilitating signal analysis and processing tasks through well-established spectral methods. The inverse transform that recovers $\boldsymbol{x}$ from its frequency-domain representation $\hat{\boldsymbol{x}} = (\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{N-1})$ is given by

$$x_n = \mathcal{F}^{-1}\{\hat{\boldsymbol{x}}\}(n) = \frac{1}{N} \sum_{m=0}^{N-1} \hat{x}_m \, e^{2\pi i m n/N}, \quad n = 0, \ldots, N-1.$$

For a D-dimensional array (or tensor) $\mathcal{X} \in \mathbb{C}^{N_1 \times \cdots \times N_D}$, the D-dimensional DFT $\hat{\mathcal{X}}(m_1, \ldots, m_D)$ is defined by applying the one-dimensional DFT independently along each dimension:

$$\sum_{n_1=0}^{N_1-1} \cdots \sum_{n_D=0}^{N_D-1} \mathcal{X}(n_1, \ldots, n_D) e^{-2\pi i \left(\frac{n_1 m_1}{N_1} + \cdots + \frac{n_D m_D}{N_D} \right)},$$

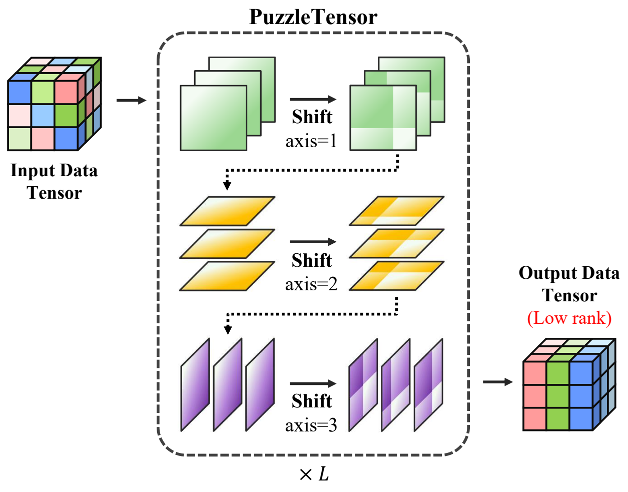
where $m_j = 0, \ldots, N_j - 1$ for each $j = 1, \ldots, D$.

3 Proposed Method

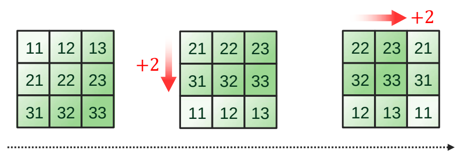
We propose PuzzleTensor, a method-agnostic data transformation technique for low-rank tensor factorization. Before delving into the specifics of our method, we aim to answer the following question: ***Why is shifting hyperslices an effective strategy for inducing low-rank structures in tensors?*** This question lies at the heart of our work and motivates the design of PuzzleTensor. We argue that shifting offers a set of benefits for efficiently reducing tensor rank:

- **Shift-Induced Data Alignment.** The rank of a tensor is closely tied to the interaction between its slices, modes, and the spatial arrangement of its data. In practice, a high rank often arises from misalignment or heterogeneity in the tensor data, where local patterns in different hyperslices are not well-aligned across modes. Shifting hyperslices modifies the spatial arrangement of the tensor, aligning similar patterns along different modes and improving the coherence of the data. This reorganization reduces the effective rank by minimizing redundant, disjoint, or orthogonal components.
- **Perfect Reconstruction.** Shifts are inherently invertible operations. Reversing both the direction (sign) and the sequence (order) of the shifts restores the original data with no loss of information. This perfect reversibility contrasts with many other transformations that introduce residual approximations or require additional constraints to be inverted accurately.
- **Minimal Learnable Parameters.** Unlike transformations whose parameter spaces grow exponentially with the dimensionality of the tensor, our shift-based method needs to learn only the offset for each hyperslice and axis. In practice, this leads to a parameter count that scales *linearly* with the number of slices, rather than exponentially with the tensor's modes or sizes. Consequently, the overall parameter budget remains manageable even for high-dimensional data.
- **Interpretability.** Because shifts re-index the data in a straightforward way, the learned offsets are easy to interpret. One can readily visualize which portions of each hyperslice are being brought together or moved apart, offering a clear rationale for how the data are restructured to reveal potential low-rank patterns. In contrast, many black-box transformations do not provide such intuitive insights into the resulting representations.

These properties make the shift-based transformation a powerful and efficient tool for reducing the effective rank of tensors.



(a) Overview of PuzzleTensor



(b) Example of shifting a hyperslice (frontmost; axis=1)

Figure 2: (a) Overview of PuzzleTensor. Given a D-mode tensor, it shifts each $(D-1)$-dimensional hyperslice along all axes to enable accurate factorization with reduced target ranks. (b) An illustration of shifting the first (frontmost) hyperslice for axis=1. A hyperslice has $(D-1)$ possible shift directions, and any indices exceeding the boundary wrap around to the beginning in a circular manner. Note that each hyperslice may be shifted by different amounts.

In this context, the challenges and ideas behind this work are summarized into three core aspects:

(1) **Learning the Discrete Shift Operation:** Shifting hyperslices of a tensor along specific axes is inherently a discrete operation, which poses a significant challenge for gradient-based optimization. To address this, we leverage the properties of the Fourier transform, which allows us to relax discrete shift operations into continuous ones on the real domain. By reformulating the operation in the frequency domain, we ensure that it becomes differentiable and compatible with standard optimization techniques (Section 3.2).

(2) **Transforming a Tensor into a Low-Rank Structure:** A critical objective in PuzzleTensor is to transform the input tensor into a structure that facilitates low-rank factorization. However, directly computing the rank of a tensor is an NP-hard problem, making this task computationally infeasible. To overcome this, we propose a novel objective function based on a matricized representation of the tensor. This objective function, designed to capture essential low-rank characteristics, is theoretically shown to minimize rank when optimized. Our approach provides a principled and computationally efficient pathway for learning transformations that induce low-rank structures in tensors (Section 3.3).

(3) **Scalability for Large-Scale Tensors:** For extremely large input tensors, directly learning shifts becomes computationally prohibitive due to memory and runtime constraints. To tackle this scalability issue, we present a sub-block decomposition method. In this framework, the input tensor is partitioned into smaller sub-blocks, and the shift operation is independently applied to each block. Sub-block decomposition naturally lends itself to parallel processing, where each block is independently shifted and processed, significantly reducing runtime in large-scale tensor scenarios (Section 3.4).

3.1 PuzzleTensor: Shifting Hyperslices

As illustrated in Figure 2, PuzzleTensor operates on a D-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$, shifting each $(D-1)$-dimensional hyperslice along all $D-1$ directions. Note that each hyperslice is allowed to be shifted by different amounts, and wraps any out-of-range indices back to the start (circular boundary).

Let us define $\mathcal{X}_{i_k} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times I_{k+1} \cdots \times I_D}$ as the $(D-1)$-mode hyperslice obtained by fixing the index i_k for mode k. Formally,

$$\mathcal{X}_{i_k} = \mathcal{X}[\ldots, i_k, \ldots], \quad i_k \in \{1, 2, \cdots, I_k\}.$$

PuzzleTensor then shifts $\mathcal{X}_{i_k}$ in each of the remaining $D-1$ modes (i.e., for every $j \neq k$). Denote by

$$h_{k,j}(i_k) \in \mathbb{Z}$$

the integer shift amount along mode j for the slice indexed by i_k. Since we use circular shifts, indices that exceed the boundary of dimension j wrap around. Using 1-based indexing, define the wrap function

$$\mathrm{wrap}_j(a) = 1 + \big((a-1) \bmod I_j\big),$$

which ensures the result stays in the range $\{1, 2, \cdots, I_j\}$.

After applying these shifts for mode k, we obtain a new tensor $\mathrm{Shift}_k(\mathcal{X}) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$, where each element is taken from a circularly shifted location in $\mathcal{X}$. Concretely,

$$\begin{aligned}&\mathrm{Shift}_k(\mathcal{X})\big(i_1, \cdots, i_D\big)\\ &= \mathcal{X}\Big(\mathrm{wrap}_1\big(i_1 - \delta_{k,1}(i_k)\big), \cdots, \mathrm{wrap}_D\big(i_D - \delta_{k,D}(i_k)\big)\Big),\end{aligned}$$

where

$$\delta_{k,j}(i_k) = \begin{cases} h_{k,j}(i_k), & \text{if } j \neq k, \\ 0, & \text{if } j = k. \end{cases}$$

In other words, for the hyperslice $\mathcal{X}_{i_k}$ indexed by i_k, mode $j \neq k$ is shifted by $h_{k,j}(i_k)$, and we do not mix the elements between slices.

To fully rearrange the tensor across all modes, we define the Shift function by sequentially applying these mode-wise shifts in any chosen order, typically $k = 1, \cdots, D$. Specifically,

$$\mathrm{Shift}(\mathcal{X}) = \mathrm{Shift}_D \circ \mathrm{Shift}_{D-1} \circ \cdots \circ \mathrm{Shift}_1(\mathcal{X}).$$

Moreover, depending on the complexity of the problem, the Shift function may be applied not just once but instead iterated L times. Such repeated applications allow for progressively refining the tensor's structure to better align with low-rank properties or other optimization objectives. Formally, the iterative shift is expressed as

$$\mathcal{X}^{(t+1)} = \mathrm{Shift}(\mathcal{X}^{(t)}), \quad t = 0, \cdots, L-1,$$

where $\mathcal{X}^{(0)}$ is the input tensor, and the tensor $\mathcal{X}^{(L)}$ is obtained after L iterations. This iterative approach balances flexibility and computational efficiency, enabling our proposed method to adaptively refine the tensor's structure as needed. We define the tensor $\mathcal{X}^{(L)}$ as the final output of PuzzleTensor and denote it by $\mathcal{Z}$:

$$\mathcal{Z} := \mathcal{X}^{(L)} = \mathrm{PuzzleTensor}(\mathcal{X}).$$

3.2 Fourier-Based Shift Operation

The process of shifting hyperslices of a tensor is inherently discrete, creating a substantial challenge for gradient-based optimization techniques. Indeed, an integer-valued shift parameter obstructs gradient flow, making it difficult to learn the optimal shift via standard backpropagation. To address this limitation, we exploit the properties of the Fourier transform, which allows us to treat discrete shifts as continuous transformations in the frequency domain. By this technique, we relax the integer shift parameter into a real-valued one, thereby rendering the operation differentiable and compatible with existing optimization techniques (see Figure 3).

3.2.1 Overview of the Frequency-Domain Shift. Consider a one-dimensional signal $\boldsymbol{x}$ of length N and let $\hat{x}(m) := \mathcal{F}\{\boldsymbol{x}\}(m)$ denote its discrete Fourier transform (DFT). When $\boldsymbol{x}$ is shifted by an integer amount h, namely $y(n) = x(n - h \mod N)$, its frequency-domain counterpart is multiplied by a complex exponential:

$$\mathcal{F}\{\boldsymbol{y}\}(m) = e^{-2\pi i m h / N} \hat{x}(m). \tag{1}$$

This property generalizes naturally to real-valued shifts $h \in \mathbb{R}$. Even if the original shift is inherently discrete, expressing it as a continuous parameter h allows for gradient-based methods to optimize it. Thus, instead of directly shifting $\boldsymbol{x}$ by integer steps in the spatial domain, we work with its frequency-domain representation $\hat{\boldsymbol{x}}$, which allows us to optimize the shift parameters continuously.

However, for the transformed signal to remain real-valued after an inverse Fourier transform, conjugate symmetry in the frequency

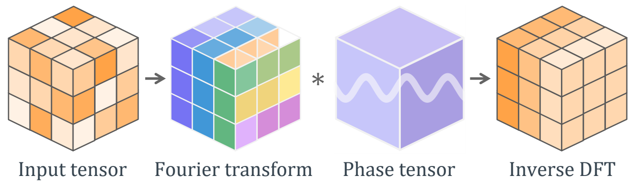


Figure 3: Fourier-based shift operation. The input tensor is processed by DFT of its mode-k hyperslices, element-wise multiplication with the phase tensor, and an inverse DFT.

domain must be preserved [30]. A vector $\boldsymbol{v} = (v_0, \ldots, v_{N-1})$ is conjugate symmetric if and only if $v_{N-m} = \overline{v_m}$ for all $m = 1, \ldots, N-1$. For a non-integer $h \in \mathbb{R} \setminus \mathbb{Z}$, directly multiplying by the term $e^{-2\pi imh/N}$ as in Equation (1) generally breaks the conjugate symmetry since $e^{-2\pi i(N-m)h/N} \neq \overline{e^{-2\pi imh/N}}$. We address this by designing a vector $\boldsymbol{\phi}_{N,h} \in \mathbb{C}^N$ satisfying the following requirements:

(1) Conjugate symmetry: $\phi_{N,h}(N-m) = \overline{\phi_{N,h}(m)}$ $(1 \leq m < N)$.
(2) Generalizability: If $h \in \mathbb{Z}$, then $\phi_{N,h}(m) = e^{-2\pi imh/N}$ $\forall m$.

To this end, let $\omega = e^{-2\pi ih/N}$, and define $\boldsymbol{\phi}_{N,h} \in \mathbb{C}^N$ as follows:

$$\boldsymbol{\phi}_{N,h} = \begin{cases} (\omega^0, \omega^1, \ldots, \omega^{\frac{N-1}{2}}, \omega^{-\frac{N-1}{2}}, \ldots, \omega^{-1}), & N\text{: odd}, \\ (\omega^0, \omega^1, \ldots, \omega^{\frac{N-2}{2}}, \Re(\omega^{\frac{N}{2}}), \omega^{-\frac{N-2}{2}}, \ldots, \omega^{-1}), & N\text{: even}, \end{cases}$$

where $\Re(z)$ denotes the real part of a complex number z. First, it is easy to see that $\boldsymbol{\phi}_{N,h}$ is conjugate symmetric since $\overline{\omega} = \omega^{-1}$. Second, when h is an integer, $\omega^{\frac{N}{2}}$ itself is a real number, so $\Re(\omega^{\frac{N}{2}}) = \omega^{\frac{N}{2}}$. Moreover, $\omega^N = e^{-2\pi ih} = 1$, and thus $\omega^{-m} = \omega^{N-m}$ for $1 \leq m < N/2$, which leads to $\phi_{N,h}(m) = \omega^m = e^{-2\pi imh/N}$ for all m.

The frequency-domain shifted signal is then expressed as

$$\hat{\boldsymbol{x}}_{\text{shifted}} = \boldsymbol{\phi}_{N,h} * \hat{\boldsymbol{x}},$$

where $*$ denotes the element-wise product. This modification ensures that when the inverse Fourier transform is applied, the resulting spatial-domain signal remains real-valued. Finally, the shifted signal in the spatial domain is obtained via the inverse DFT as

$$\boldsymbol{x}_{\text{shifted}} = \mathcal{F}^{-1}\{\hat{\boldsymbol{x}}_{\text{shifted}}\}.$$

This approach generalizes the discrete shift operation to arbitrary $h \in \mathbb{R}$, enabling its use in gradient-based optimization frameworks.

3.2.2 Extension to Hyperslices. We extend the frequency-domain shift operation described for one-dimensional signals to $(D-1)$-dimensional hyperslices in a tensor. This generalization is achieved by applying a $(D-1)$-dimensional DFT to the hyperslice, performing frequency-domain shifts along each axis, and then applying an inverse $(D-1)$-dimensional DFT to obtain the shifted hyperslice in the spatial domain.

Let $\mathcal{X}_{i_k} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times I_{k+1} \cdots \times I_D}$ be the $(D-1)$-dimensional hyperslice obtained by fixing the index i_k along mode k. We first compute the $(D-1)$-dimensional DFT of $\mathcal{X}_{i_k}$, denoted by $\hat{\mathcal{X}}_{i_k} = \mathcal{F}_{D-1}\{\mathcal{X}_{i_k}\} \in \mathbb{C}^{I_1 \times \cdots \times I_{k-1} \times I_{k+1} \cdots \times I_D}$. The frequency-domain shifted hyperslice is computed by multiplying $\hat{\mathcal{X}}_{i_k}$ with the exponential factor corresponding to the shift $h_{k,j}(i_k)$ along each axis:

$$\hat{\mathcal{X}}_{\text{shifted},i_k} = \bigotimes_{j \neq k}^{\text{outer}} \boldsymbol{\phi}_{I_j, h_{k,j}(i_k)} * \hat{\mathcal{X}}_{i_k},$$

where $h_{k,j}(i_k) \in \mathbb{R}$ is the real-valued shift amount along axis j, and $\bigotimes_{j \neq k}^{\text{outer}} \boldsymbol{\phi}_{I_j, h_{k,j}(i_k)} \in \mathbb{C}^{I_1 \times \cdots \times I_{k-1} \times I_{k+1} \cdots \times I_D}$ denotes the outer product of the exponential factors. Finally, the shifted hyperslice in the spatial domain is obtained by applying the inverse $(D-1)$-dimensional DFT to $\hat{\mathcal{X}}_{\text{shifted},i_k}$:

$$\mathcal{X}_{\text{shifted},i_k} = \mathcal{F}_{D-1}^{-1}\{\hat{\mathcal{X}}_{\text{shifted},i_k}\}.$$

After performing the above operations for all hyperslices $\mathcal{X}_{i_k}$ (i.e., for each $i_k \in \{1, \ldots, I_k\}$), the final mode-$k$ shifted tensor $\text{Shift}_k(\mathcal{X})$ is constructed by concatenating the hyperslices along mode k:

$$\text{Shift}_k(\mathcal{X}) = \bigoplus_{i_k=1}^{I_k} \mathcal{X}_{\text{shifted},i_k}$$

where $\bigoplus$ denotes tensor concatenation along mode k. By applying these shifts iteratively to each hyperslice along all modes, PuzzleTensor effectively reorganizes tensor data for low-rank representations.

3.3 Optimization for Low-Rank Structures

A core objective of PuzzleTensor is to reshape the input tensor into a form that naturally admits a low-rank approximation. However, determining the exact rank of an n-mode tensor is known to be NP-hard for n larger than two [9], rendering direct rank-minimization approaches computationally intractable. To tackle this, we propose a novel objective function grounded in a matricized representation of the tensor, designed to capture essential low-rank properties through a theoretically justified criterion. In particular, Theorem 1 shows that minimizing the nuclear norm of each matricized view of $\mathcal{Z} = \text{PuzzleTensor}(\mathcal{X})$ induces sparsity in the core tensor of the corresponding higher-order singular value decomposition (HOSVD). This sparsity in turn facilitates approximating the original tensor $\mathcal{X}$ at a reduced target rank without sacrificing accuracy. Formally, we aggregate these norms across all modes as the following loss:

$$\mathcal{L} = \sum_{1 \leq k \leq D} \frac{1}{\sqrt{I_k}} \|Z_{(k)}\|_*, \tag{2}$$

where I_k is the size of the k-mode, $Z_{(k)} \in \mathbb{R}^{I_k \times \prod_{j \neq k} I_j}$ represents the k-mode matricization of the transformed tensor $\mathcal{Z}$, and $\|\cdot\|_*$ denotes the nuclear norm. This formulation provides a principled and computationally efficient mechanism for inducing low-rank structures, as discussed in the following theorem.

Theorem 1. *Let $\mathcal{S} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ be the core tensor of the HOSVD of $\mathcal{Z} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ and $1 \leq k \leq D$. Then, for sufficiently large $I_1 \cdots I_{k-1} I_{k+1} \cdots I_D$, we have the asymptotic equality*

$$\mathrm{E}\left[\|vec(\mathcal{S})\|_1\right] \sim \|Z_{(k)}\|_* \sqrt{\frac{2}{\pi} I_1 \cdots I_{k-1} I_{k+1} \cdots I_D}\,, \tag{3}$$

where $vec(\cdot)$ denotes the vectorization of a tensor, $Z_{(k)}$ is the k-mode matricization of $\mathcal{Z}$, and $\|\cdot\|_*$ is the nuclear norm. □

Proof. Let $U_d \in \mathbb{R}^{I_d \times I_d}$ be the left singular matrix in the SVD of $Z_{(d)}$ for each $1 \leq d \leq D$. Then, the core tensor $\mathcal{S}$ is defined as

$$\mathcal{S} = \mathcal{Z} \times_1 U_1^\top \times_2 U_2^\top \times_3 \cdots \times_D U_D^\top,$$

where $\times_d$ denotes the d-mode product. This may be written in matricized form as follows:

$$\begin{aligned} S_{(k)} &= U_k^\top Z_{(k)} \left(U_D^\top \otimes \cdots \otimes U_{k+1}^\top \otimes U_{k-1}^\top \otimes \cdots \otimes U_1^\top\right)^\top \\ &= U_k^\top Z_{(k)} \left(U_D \otimes \cdots \otimes U_{k+1} \otimes U_{k-1} \otimes \cdots \otimes U_1\right), \end{aligned} \tag{4}$$

where $S_{(k)}$ is the mode-k matricization of $\mathcal{S}$, and $\otimes$ is the Kronecker product. Let the SVD of $Z_{(k)}$ be $U_k \Sigma_k V_k^\top$. Then, (4) becomes

$$S_{(k)} = \Sigma_k V_k^\top \left(U_D \otimes \cdots \otimes U_{k+1} \otimes U_{k-1} \otimes \cdots \otimes U_1\right)$$

since U_k is a unitary matrix. Denote $I_{\neq k} := I_1 \cdots I_{k-1} I_{k+1} \cdots I_D$ and $U_{\neq k} := U_D \otimes \cdots \otimes U_{k+1} \otimes U_{k-1} \otimes \cdots \otimes U_1$. We have

$$\begin{aligned}\mathrm{E}\left[\|vec(\mathcal{S})\|_1\right] = \mathrm{E}\left[\|vec(S_{(k)})\|_1\right] = \mathrm{E}\left[\|vec(\Sigma_k V_k^\top U_{\neq k})\|_1\right] \\ = \sum_{i=1}^{I_k} \sigma_{k,i} \mathrm{E}\left[\|(V_k^\top U_{\neq k})[i,:]\|_1\right],\end{aligned}$$

where $\sigma_{k,i}$ is the i-th singular value of $Z_{(k)}$, and $M[i,:]$ represents the i-th row of a matrix M.

We show that for all i, $\mathrm{E}[\|(V_k^\top U_{\neq k})[i,:]\|_1] \to \sqrt{2I_{\neq k}/\pi}$ as $I_{\neq k}$ approaches infinity, from which the proof follows. Let us denote the i-th column vector of V_k as $X(i) \in \mathbb{R}^{I_{\neq k}}$, and the j-th column vector of $U_{\neq k}$ as $Y(j) \in \mathbb{R}^{I_{\neq k}}$. We may assume that for sufficiently large $I_{\neq k}$, $\{X(i)_1, \cdots, X(i)_{I_{\neq k}}\}$ are independent and normally distributed random variables: $X(i)_n \sim \mathcal{N}(0, \sigma^2)$ for $n = 1, \cdots, I_{\neq k}$, and σ^2 is a variance. Because $X(i)$ is a unit vector, the expected value of $\sum_{n=1}^{I_{\neq k}} X(i)_n^2 \sim \sigma^2 \chi^2(I_{\neq k})$ must be equal to 1, where $\chi^2(I_{\neq k})$ is the chi-squared distribution with $I_{\neq k}$ degrees of freedom. It follows from $\mathrm{E}[\sigma^2 \chi^2(I_{\neq k})] = \sigma^2 I_{\neq k} = 1$ that $\sigma^2 = 1/I_{\neq k}$, which yields

$$X(i)_n \sim \mathcal{N}(0, 1/I_{\neq k}), \quad n = 1, \cdots, I_{\neq k}.$$

A similar argument for $Y(j)$ leads to the following:

$$Y(j)_n \sim \mathcal{N}(0, 1/I_{\neq k}), \quad n = 1, \cdots, I_{\neq k}.$$

Assuming $X(i)_n$ and $Y(j)_n$ are independent, we have

$$\begin{aligned}\mathrm{E}[X(i)_n Y(j)_n] &= \mathrm{E}[X(i)_n] \cdot \mathrm{E}[Y(j)_n] = 0, \\ \mathrm{Var}[X(i)_n Y(j)_n] &= \mathrm{Var}[X(i)_n] \cdot \mathrm{Var}[Y(j)_n] \\ &\quad + \mathrm{E}[X(i)_n]^2 \cdot \mathrm{Var}[Y(j)_n] \\ &\quad + \mathrm{E}[Y(j)_n]^2 \cdot \mathrm{Var}[X(i)_n] = 1/I_{\neq k}^2\end{aligned}$$

for all $n = 1, \cdots, I_{\neq k}$. Now, for sufficiently large $I_{\neq k}$, the distribution of $I_{\neq k}^{-1/2} \sum_{n=1}^{I_{\neq k}} X(i)_n Y(j)_n$ is approximately $\mathcal{N}(0, 1/I_{\neq k}^2)$ by the central limit theorem, and thus, $\sum_{n=1}^{I_{\neq k}} X(i)_n Y(j)_n \sim \mathcal{N}(0, 1/I_{\neq k})$. Then, it can be readily seen that its absolute deviation is given by

$$\mathrm{E}\left[\left|\sum_n X(i)_n Y(j)_n\right|\right] = \sqrt{2/(\pi I_{\neq k})}.$$

This implies that

$$\begin{aligned}\mathrm{E}[\|(V_k^\top U_{\neq k})[i,:]\|_1] &= \mathrm{E}\left[\sum_{j=1}^{I_{\neq k}} \left|\sum_n X(i)_n Y(j)_n\right|\right] \\ &= \sum_{j=1}^{I_{\neq k}} \mathrm{E}\left[\left|\sum_n X(i)_n Y(j)_n\right|\right] = \sqrt{2I_{\neq k}/\pi},\end{aligned}$$

hence the proof. □

Note that it follows from (3) that

$$\mathrm{E}\left[\|vec(\mathcal{S})\|_1\right] \propto \frac{1}{\sqrt{I_k}} \|Z_{(k)}\|_*$$

for each $1 \leq k \leq D$ since $I_1 \cdots I_D$ is constant. We aggregate these terms for all k to account for the influence across all axes, which leads to our proposed objective function (Equation (2)).

The loss $\mathcal{L}$ is then minimized through a gradient-based optimization procedure, allowing PuzzleTensor to learn the shift parameters end-to-end. By incorporating this optimization into standard back-propagation frameworks, PuzzleTensor iteratively adjusts the shift parameters to reorganize the tensor, effectively reducing the rank while maintaining accurate reconstruction of the original data.

3.4 Sub-Block Shifting

For extremely large tensors, applying the shift operation across the entire dataset becomes impractical due to high memory demands and significant runtime overhead. To address this issue, we partition the input tensor into smaller, more manageable sub-blocks, allowing shifts to be applied independently within each block. This strategy not only reduces computational complexity but also enables parallel processing, as each sub-block is processed independently.

Let $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ be the original D-mode input tensor. We divide each dimension $k \in \{1, \ldots, D\}$ into B_k blocks. Define a set of partition boundaries:

$$0 = p_{k,0} < p_{k,1} < \cdots < p_{k,B_k-1} < p_{k,B_k} = I_k,$$

where B_k needs not evenly divide I_k. In cases where I_k is not a multiple of B_k, the last sub-block will have a smaller (or larger) residual size. Consequently, each mode k is split into B_k segments:

$$\text{Segment } b_k \text{ for mode } k: \quad p_{k,b_k-1} + 1 \leq n_k \leq p_{k,b_k},$$

where $b_k \in \{1, \ldots, B_k\}$, and n_k indexes the coordinate in mode k. A sub-block of $\mathcal{X}$ is then indexed by $\boldsymbol{b} = (b_1, b_2, \ldots, b_D)$, and we denote the $\boldsymbol{b}$-th sub-block of $\mathcal{X}$ as follows:

$$\mathcal{X}^{(\boldsymbol{b})} = \{\mathcal{X}(n_1, \ldots, n_D) : p_{k,b_k-1} + 1 \leq n_k \leq p_{k,b_k},\ k = 1, \ldots, D\}.$$

Hence, $\mathcal{X}$ is fully partitioned into the collection $\{\mathcal{X}^{(\boldsymbol{b})} : 1 \leq b_k \leq B_k, 1 \leq k \leq D\}$. Typically, the partition boundaries $\{p_{k,b_k}\}$ are chosen such that the sub-blocks are of approximately equal size. Formally, we set $p_{k,b_k} = b_k \cdot \lfloor I_k / B_k \rfloor$ for $b_k \in \{1, \ldots, B_k - 1\}$.

Within each sub-block $\mathcal{X}^{(\boldsymbol{b})}$, we apply the shift operation presented in Section 3.2. Specifically, for each mode k, we learn a mode-k shift function $\mathrm{Shift}_k(\cdot)$ that shifts the $(D-1)$-dimensional hyperslices in $\mathcal{X}^{(\boldsymbol{b})}$. Crucially, the shift amounts for one sub-block are not shared with other sub-blocks, allowing each block's shifts to be learned independently. Thus, the shifts can be computed in parallel, drastically reducing the overall runtime for large-scale datasets. Once the shifts have been applied to each sub-block, the transformed blocks are concatenated back into a single tensor:

$$\mathrm{Shift}(\mathcal{X}) = \left[\mathrm{Shift}_D \circ \mathrm{Shift}_{D-1} \circ \cdots \circ \mathrm{Shift}_1\left(\mathcal{X}^{(\boldsymbol{b})}\right)\right]_{\boldsymbol{b}}.$$

In practice, the partition boundaries $\{p_{k,b_k}\}$ and the block indices $\{B_k\}$ are selected to balance computational load and memory usage, accommodating a wide range of tensor sizes and shapes.

Remark. By partitioning the tensor into sub-blocks and shifting each block independently, our method substantially alleviates the computational challenges posed by very large tensors. Although the sub-blocks do not share shift parameters, we find that localized transformations often suffice to reduce rank and preserve key structure, making this strategy both scalable and effective for large-scale tensor factorization.

3.5 Data Compression with PuzzleTensor

Applying PuzzleTensor to an input tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ reveals a structure that can be more effectively approximated by standard decomposition techniques, such as CP, Tucker, or Tensor-Train. By shifting hyperslices to reduce the tensor's effective rank, PuzzleTensor enables each decomposition algorithm to operate with lower target ranks, thereby creating factor representations that significantly reduce storage requirements.

Once $\mathcal{X}$ has been shifted and factorized, we compress the data by storing only the resulting factors (e.g., factor matrices and core tensors) along with the learned shift parameters. In practice, any of the well-established decompositions may be used, chosen according to the desired compression ratio or domain-specific considerations.

To reconstruct $\mathcal{X}$, we first regenerate the tensor from the stored factors. Specifically, we apply the inverse decomposition to obtain the shifted tensor $\mathcal{Z}$. We then recover the original tensor by reversing the PuzzleTensor shifts. Using the stored shift parameters, we invert the shifts by applying them in the opposite direction and sequence. This strictly reverses the transformation, ensuring that no information is lost in the backward shifting process.

In this context, PuzzleTensor requires storing the learned shift parameters. Assume, for simplicity, that each dimension I_k is evenly divided into B_k blocks and that the shift procedure is iterated L times, then the following theorem holds:

Theorem 2. *The number of shift parameters in PuzzleTensor is*

$$L(D-1) \cdot \sum_{k=1}^{D} I_k / B_k \cdot \prod_{k=1}^{D} B_k,$$

provided that I_k is evenly divided into B_k blocks. □

Proof. See Appendix A.1. □

When the block sizes B_k are fixed, the parameter count grows linearly with the tensor sizes I_k, making the method highly scalable for large tensors. Increasing the number of blocks enhances parallelization and reduces computation time. Although the larger parameter set introduces overhead and might cause a decline in reconstruction accuracy, our experimental evaluation confirms that the reduction in accuracy is minor (see Section 4.4).

Finally, we provide the time complexity of PuzzleTensor. As stated in Theorem 3, PuzzleTensor exhibits a quasi-linear time complexity with respect to the number of entries in the input tensor, implying that it remains efficient even for large-scale tensors.

Theorem 3. *The time complexity of PuzzleTensor is given by*

$$O\left(L(D-1) \cdot \prod_{k=1}^{D} I_k \cdot \log\left(\prod_{k=1}^{D} I_k / B_k\right)\right).$$ □

Proof. See Appendix A.2. □

4 Experiments

Through experiments, we answer the following questions:

Q1 Performance (Section 4.2). How accurately does PuzzleTensor reconstruct tensor data compared to baselines?

Q2 Scalability (Section 4.3). How does PuzzleTensor scale with increasing input size?

Q3 Ablation study (Section 4.4). How do different design choices (number of shift layers and block size) affect performance?

Table 1: Dataset summarization.

Dataset	Type	Size	Density
$\{\mathrm{D}_n\}_{n=4,\cdots,8}$[1]	Synthetic	$2^n \times 2^n \times 2^n$	1.000
$\{\mathrm{S}_n\}_{n=4,\cdots,8}$[1]	Synthetic	$2^n \times 2^n \times 2^n$	0.010
Uber[2]	Real-world	$183 \times 24 \times 1140$	0.138
Action[3]	Real-world	$100 \times 570 \times 567$	0.393
PEMS-SF[4]	Real-world	$963 \times 144 \times 440$	0.999
Activity[5]	Real-world	$337 \times 570 \times 320$	0.569
Stock[6]	Real-world	$1317 \times 88 \times 916$	0.816
NYC[7]	Real-world	$265 \times 265 \times 28 \times 35$	0.118

4.1 Experimental Setup

Machine. Our system utilizes an Intel Core i7-10700KF @ 3.80GHz processor paired with 32GB of RAM and a single GPU machine with RTX 3070 Ti.

Datasets. We evaluate PuzzleTensor on both synthetic and real-world data, as summarized in Table 1. The synthetic datasets[1] $\{\mathrm{D}_n\}$ and $\{\mathrm{S}_n\}$ are 3-mode tensors of size $2^n \times 2^n \times 2^n$. D_n has a density of 1.00, generated from a standard normal distribution, whereas S_n is a sparse binary tensor with a density of 0.01, where nonzero entries are randomly assigned. Following [24], we use six real-world datasets. The Uber [36] dataset records the number of Uber pickups in New York City, represented by (date, hour, latitude; count). Action and Activity [17, 42] contain features derived from motion videos, each represented as (frame, feature, video; value). PEMS-SF [7] consists of car-lane usage rates in the San Francisco Bay Area, expressed as (station, timestamp, day; measurement). Stock [12] captures various stocks in the format (time, feature, stock; value). Finally, NYC encompasses trip records from yellow and green taxis in New York City between 2020 and 2022, structured as (pick up zone, drop off zone, day, month; count). We also add Gaussian noise to the real-world datasets to further evaluate the robustness of each tensor decomposition method. All data are represented in double-precision floating-point format.

Baselines. We compare against three widely used tensor decomposition methods—CP, Tucker, and TT—as well as each of these methods enhanced by our PuzzleTensor (i.e., CP+PuzzleTensor, Tucker+PuzzleTensor, TT+PuzzleTensor).

- **CP, Tucker, and TT.** These standard techniques are often the first choice for high-dimensional data factorization. However, in many cases they struggle when local structures are misaligned or when the data do not fit strict low-rank assumptions.
- **PuzzleTensor-augmented CP, Tucker, and TT.** We apply PuzzleTensor as a preprocessing step (i.e., shifting hyperslices) prior to the decomposition methods. After obtaining the transformed tensor $\mathcal{Z}$, we run the corresponding factorization with lower target ranks to ensure a fair comparison. For fairness, both the baseline and baseline+PuzzleTensor approaches operate on each subblock independently.

[1]https://github.com/snudatalab/PuzzleTensor/
[2]http://frostt.io/tensors/
[3]https://github.com/titu1994/MLSTM-FCN
[4]https://www.timeseriesclassification.com/
[5]https://github.com/titu1994/MLSTM-FCN
[6]https://github.com/jungijang/KoreaStockData
[7]https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Table 2: Reconstruction errors at varying compression sizes across different datasets, where the lower errors are highlighted. Each decomposition method benefits from PuzzleTensor, demonstrating its effectiveness and broad applicability.

Dataset	D_8			S_8			Uber			Action		
Compressed Size (MB)	10	25	50	10	25	50	3.0	7.5	15	20	45	90
CP	0.9428	0.8677	0.7139	0.9095	0.7857	0.6211	0.7314	0.6294	0.5363	0.7010	0.6358	0.5300
CP+PuzzleTensor	0.9385	0.8233	0.6757	0.8928	0.7145	0.5064	0.7206	0.6219	0.5151	0.7015	0.6298	0.5185
Tucker	0.9333	0.8490	0.7334	0.8970	0.7905	0.6592	0.7456	0.6437	0.5535	0.6741	0.6227	0.5469
Tucker+PuzzleTensor	0.9228	0.8073	0.6809	0.8674	0.6546	0.4832	0.6903	0.6019	0.5159	0.6617	0.6046	0.5230
TT	0.9401	0.8664	0.7360	0.9133	0.8216	0.6754	0.7786	0.6599	0.5430	0.6851	0.6261	0.5236
TT+PuzzleTensor	0.9284	0.8274	0.6681	0.8692	0.7342	0.5137	0.7167	0.6050	0.5017	0.6722	0.6065	0.4981

Dataset	PEMS-SF			Activity			Stock			NYC		
Compressed Size (MB)	45	100	200	58	115	230	100	200	400	65	130	260
CP	0.6785	0.6200	0.5002	0.6688	0.6068	0.4698	0.6516	0.5880	0.4633	0.6604	0.6137	0.4954
CP+PuzzleTensor	0.6799	0.6148	0.4897	0.6660	0.6006	0.4583	0.6521	0.5730	0.4446	0.6597	0.5952	0.4686
Tucker	0.6672	0.6163	0.5193	0.6482	0.5846	0.4888	0.6343	0.5728	0.4798	0.6486	0.6111	0.4786
Tucker+PuzzleTensor	0.6523	0.5967	0.4936	0.6315	0.5625	0.4636	0.6120	0.5416	0.4423	0.6318	0.5890	0.4479
TT	0.6702	0.6188	0.5149	0.6587	0.5937	0.4841	0.6449	0.5830	0.4768	0.6402	0.5926	0.5460
TT+PuzzleTensor	0.6543	0.5980	0.4881	0.6415	0.5711	0.4582	0.6229	0.5520	0.4389	0.6305	0.5832	0.5346

Hyperparameters. The configuration of the hyperparameters in PuzzleTensor is described in Appendix C.

Measure. We focus on *reconstruction error* of the tensor decomposition methods. Given a tensor $\mathcal{X}$, denote its reconstruction from factors as $\hat{\mathcal{X}}$. The reconstruction error is defined by $\|\mathcal{X}-\hat{\mathcal{X}}\|_F/\|\mathcal{X}\|_F$, where $\|\cdot\|_F$ denotes the Frobenius norm.

4.2 Performance (Q1)

We begin by assessing the reconstruction performance of PuzzleTensor under different compression sizes. For each dataset, we specify several target compressed sizes (in MB) and measure the corresponding reconstruction error using various decomposition methods. We calculate the compressed size by combining both the PuzzleTensor parameter count and that of the decomposition itself. Note that increasing the compressed size typically allows for higher ranks in the decomposition, thus improving accuracy.

Table 2 presents the results for eight representative datasets, each evaluated at three different compressed sizes. Incorporating PuzzleTensor reduces the reconstruction error in most scenarios, suggesting that shifting hyperslices allows the factorization to capture more compact and precise representations. Furthermore, we observe consistent gains across different tensor decomposition families. This broad applicability indicates the versatility of PuzzleTensor in enhancing various factorization pipelines. Additional results for even larger compression sizes are provided in Appendix B.

4.3 Scalability (Q2)

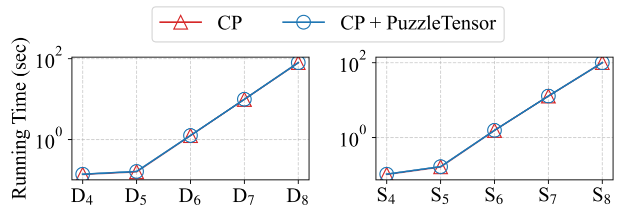
We evaluate the scalability of PuzzleTensor by measuring its running time as the input tensor size increases, using both dense data $\{D_n\}$ and sparse data $\{S_n\}$. Specifically, we consider tensors whose spatial dimensions range from $2^4 \times 2^4 \times 2^4$ up to $2^8 \times 2^8 \times 2^8$. Figure 4 presents the running time of CP, Tucker, and TT decompositions with and without PuzzleTensor across various input tensor sizes. Even though PuzzleTensor introduces additional shift operations, its sub-block strategy (Section 3.4) ensures that the overall computational cost remains nearly identical to that of the original decomposition method without PuzzleTensor. Across all tested decompositions (CP, Tucker, and TT) and a wide range of dense and sparse tensors, PuzzleTensor exhibits minimal overhead, offering nearly linear scaling with respect to the number of elements.

4.4 Ablation Study (Q3)

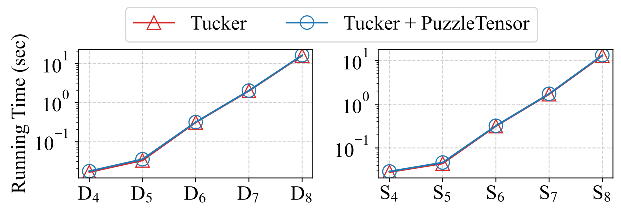
We conduct an ablation study to evaluate the impact of two core components in PuzzleTensor: (1) the number L of shift layers, and (2) the block size B_k in sub-block shifting.

Effect of the number of shift layers. Recall that PuzzleTensor can be applied L times in succession (Section 3.1). Table 3 reports the reconstruction errors on the S_8 dataset (compressed size = 50MB) when increasing L from 0 to 4, where $L = 0$ indicates the naïve baseline without PuzzleTensor. We observe that even a *single* layer of PuzzleTensor yields a notable improvement over the baselines. Additional layers further enhance alignment, but diminishing returns appear after $L \approx 3$. For most practical scenarios, $L = 2$ or 3 achieves a good balance of accuracy vs. computational cost.

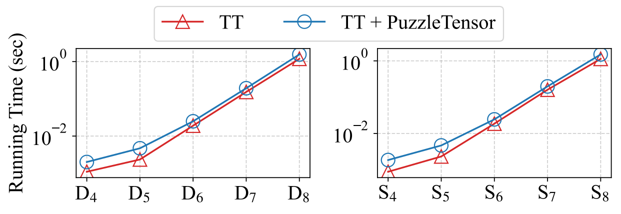
Effect of the block size. In Section 3.4, we propose splitting each k-th mode into B_k blocks to mitigate runtime complexity. Table 4 presents the impact of varying the block size in a single mode (while keeping the others fixed) on both running time and reconstruction error. We analyze CP+PuzzleTensor on the D_8 dataset by varying the block size as [8, 8, B], where $B \in \{1, 2, 4, 8, 16\}$, ensuring a fixed compressed size of 50MB. Although Table 4 shows the results only for CP decomposition, Tucker and TT exhibit similar behavior. Notably, larger block sizes facilitate parallel computation, reducing the overall running time with a slight degradation in accuracy due to overhead. Thus, selecting moderately sized blocks offers a good trade-off between accuracy and computational efficiency.



(a) Running time of CP vs. CP+PuzzleTensor



(b) Running time of Tucker vs. Tucker+PuzzleTensor



(c) Running time of TT vs. TT+PuzzleTensor

Figure 4: Running time of CP, Tucker, and TT decompositions with and without PuzzleTensor. The results confirm that the additional shift operations by PuzzleTensor do not significantly impact computational cost, maintaining the efficiency of the base decomposition methods.

5 Discussion

Our experiments demonstrate that PuzzleTensor consistently lowers reconstruction error across diverse decomposers and compression budgets, yet several open directions warrant further study. First, the current implementation attains the state-of-the-art accuracy without exploiting sparsity; incorporating sparse algebra could reduce both memory traffic and FFT cost for highly sparse tensors, and we regard this as an immediate avenue for optimization.

Second, joint optimization of shift parameters and low-rank factors in an end-to-end framework could further enhance compression; this is model-agnostic, albeit at the cost of additional complexity—a trade-off we plan to explore in future work.

Third, one can shift at finer granularity than entire hyperslices—for example, on a fiber-by-fiber basis—but this dramatically enlarges the number of learnable parameters. With tensor order $D = 3$ and mode sizes $I_1 = I_2 = I_3 = I$, our design uses $6I$ shift parameters, whereas per-fiber shifting would require $3I^2$. Because $6I \ll 3IR$ is usually satisfied for practical ranks R but $3I^2 \ll 3IR$ is not guaranteed, aggressively fine shifts can offset any rank-reduction benefit unless the decomposition rank is also decreased substantially.

Finally, we acknowledge that factor matrices are often inspected for downstream analysis, and large shifts may complicate semantic interpretation when axis labels carry strict meaning. In practice, one can freeze non-reorderable modes while shifting only time or other flexible dimensions, or apply PuzzleTensor chiefly to domains where axes are inherently misaligned, such as video, volumetric medical images, LiDAR scans, and geophysical cubes.

Table 3: Reconstruction errors at varying numbers of layers. $L = 0$ denotes the standard baseline without PuzzleTensor. Setting $L \approx 3$ is a favorable option in most scenarios.

Number of Layers (L)	0	1	2	3	4
CP+PuzzleTensor	0.621	0.539	0.514	0.506	0.504
Tucker+PuzzleTensor	0.659	0.543	0.501	0.483	0.480
TT+PuzzleTensor	0.675	0.558	0.536	0.514	0.517

Table 4: Effect of block size on PuzzleTensor. Increasing the block size enhances computational efficiency while causing only a minor decrease in accuracy.

Block Size (B)	1	2	4	8	16
Running time (sec)	1.355	1.107	0.785	0.562	0.410
Reconstruction error	0.661	0.659	0.663	0.676	0.688

6 Conclusions

In this paper, we propose PuzzleTensor, a method-agnostic data transformation that leverages hyperslice shifts to achieve compact tensor factorization without sacrificing reconstruction accuracy. By learning shift parameters through a differentiable framework, PuzzleTensor effectively reduces the target rank, making it broadly applicable to existing tensor decomposition techniques such as CP, Tucker, and TT. Moreover, our theoretical analysis shows that PuzzleTensor yields a low-rank structure under minimal data assumptions, while extensive experiments confirm consistent improvements over baseline approaches in both synthetic and real-world settings. Promising directions for follow-up include applying PuzzleTensor to sparse, streaming tensors and developing a unified framework that co-optimizes the shift transformation and decomposition parameters for even higher compression ratios.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) funded by MSIT(2022R1A2C3007921), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT), [No.RS-2020-II200894, Flexible and Efficient Model Compression Method for Various Applications and Environments], [No.RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)], [No.RS-2021-II212068, Artificial Intelligence InnovationHub (Artificial Intelligence Institute, Seoul National University)], [No.2022-0-00641, XVoice: Multi-Modal Voice Meta Learning], [No.RS-2024-00509257, Global AI Frontier Lab], and Youlchon Foundation. The Institute of Engineering Research at Seoul National University and the ICT at Seoul National University provided research facilities for this work. U Kang is the corresponding author.

References

[1] Donald A Adjeroh and Supriya D Sawant. 2009. Error-resilient transmission for 3D DCT coded video. *IEEE Transactions on Broadcasting* 55, 2 (2009), 178–189.

[2] Anastasia Aidini, Grigorios Tsagkatakis, and Panagiotis Tsakalides. 2021. Tensor decomposition learning for compression of multidimensional signals. *IEEE Journal of Selected Topics in Signal Processing* 15, 3 (2021), 476–490.

[3] Rafael Ballester-Ripoll, Peter Lindstrom, and Renato Pajarola. 2019. TTHRESH: Tensor compression for multidimensional visual data. *IEEE transactions on visualization and computer graphics* 26, 9 (2019), 2891–2903.

[4] Venkat Chandrasekaran, Sujay Sanghavi, Pablo A Parrilo, and Alan S Willsky. 2009. Sparse and low-rank matrix decompositions. *IFAC Proceedings Volumes* 42, 10 (2009), 1493–1498.

[5] Dongjin Choi, Jun-Gi Jang, and U Kang. 2019. S3CMTF: Fast, accurate, and scalable method for incomplete coupled matrix-tensor factorization. *PLOS ONE* 14, 6 (06 2019), 1–20.

[6] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301.

[7] Marco Cuturi. 2011. Fast global alignment kernels. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. 929–936.

[8] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. 2000. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SIAM journal on Matrix Analysis and Applications* 21, 4 (2000), 1324–1342.

[9] Johan Håstad. 1990. Tensor rank is NP-complete. *Journal of algorithms* 11, 4 (1990), 644–654.

[10] Cole Hawkins, Xing Liu, and Zheng Zhang. 2022. Towards compact neural networks via end-to-end training: A Bayesian tensor approach with automatic rank determination. *SIAM Journal on Mathematics of Data Science* 4, 1 (2022), 46–71.

[11] Junhui Hou, Lap-Pui Chau, Nadia Magnenat-Thalmann, and Ying He. 2014. Scalable and compact representation for motion capture data using tensor decomposition. *IEEE Signal Processing Letters* 21, 3 (2014), 255–259.

[12] Jun-Gi Jang and U Kang. 2021. Fast and Memory-Efficient Tucker Decomposition for Answering Diverse Time Range Queries. In *KDD*.

[13] Jun-Gi Jang, Jeongyoung Lee, Yong-chan Park, and U Kang. 2023. Fast and accurate dual-way streaming parafac2 for irregular tensors-algorithm and application. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 879–890.

[14] Jun-Gi Jang, Yong-chan Park, and U Kang. 2024. Fast and accurate parafac2 decomposition for time range queries on irregular tensors. In *CIKM*. 962–972.

[15] Hyunsik Jeon, Jongjin Kim, Jaeri Lee, Jong-eun Lee, and U Kang. 2023. Aggregately diversified bundle recommendation via popularity debiasing and configuration-aware reranking. In *PAKDD*. Springer, 348–360.

[16] Hyunsik Jeon, Jong-eun Lee, Jeongin Yun, and U Kang. 2024. Cold-start Bundle Recommendation via Popularity-based Coalescence and Curriculum Heating. In *Proceedings of the ACM Web Conference 2024*. 3277–3286.

[17] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. 2019. Multivariate LSTM-FCNs for time series classification. *Neural networks* 116 (2019), 237–245.

[18] Venera Khoromskaia and Boris N Khoromskij. 2022. Ubiquitous nature of the reduced higher order SVD in tensor-based scientific computing. *Frontiers in Applied Mathematics and Statistics* 8 (2022), 826988.

[19] Henk AL Kiers. 2000. Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics: A Journal of the Chemometrics Society* 14, 3 (2000), 105–122.

[20] Jongjin Kim, Hyunsik Jeon, Jaeri Lee, and U Kang. 2023. Diversely regularized matrix factorization for accurate and aggregately diversified recommendation. In *PAKDD*. Springer, 361–373.

[21] Jongjin Kim and U Kang. 2025. Sequentially diversified and accurate recommendations in chronological order for a series of users. In *WSDM*. 811–819.

[22] Junghun Kim, Ka Hyun Park, Jun-Gi Jang, and U Kang. 2024. Fast and accurate domain adaptation for irregular tensor decomposition. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1383–1394.

[23] Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[24] Taehyung Kwon, Jihoon Ko, Jinhong Jung, Jun-Gi Jang, and Kijung Shin. 2024. Compact lossy compression of tensors via neural tensor-train decomposition. *Knowledge and Information Systems* (2024), 1–43.

[25] Taehyung Kwon, Jihoon Ko, Jinhong Jung, and Kijung Shin. 2023. Neukron: Constant-size lossy compression of sparse reorderable matrices and tensors. In *Proceedings of the ACM Web Conference 2023*. 71–81.

[26] Jaeri Lee, Jeongin Yun, and U Kang. 2024. Towards True Multi-interest Recommendation: Enhanced Scheme for Balanced Interest Training. In *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 394–402.

[27] SeungJoo Lee, Yong-chan Park, and U Kang. 2024. Accurate Coupled Tensor Factorization with Knowledge Graph. In *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 1009–1018.

[28] Xingyi Liu and Keshab K Parhi. 2023. Tensor Decomposition for Model Reduction in Neural Networks: A Review [Feature]. *IEEE Circuits and Systems Magazine* 23, 2 (2023), 8–28.

[29] Yipeng Liu, Maarten De Vos, and Sabine Van Huffel. 2015. Compressed sensing of multichannel EEG signals: the simultaneous cosparsity and low-rank optimization. *IEEE Transactions on Biomedical Engineering* 62, 8 (2015), 2055–2061.

[30] Alan V Oppenheim. 1999. *Discrete-time signal processing*. Pearson Education India.

[31] Ivan V Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing* 33, 5 (2011), 2295–2317.

[32] Yong-chan Park, Jun-Gi Jang, and U Kang. 2021. Fast and accurate partial fourier transform for time series data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1309–1318.

[33] Yong-chan Park, Jongjin Kim, and U Kang. 2024. Fast Multidimensional Partial Fourier Transform with Automatic Hyperparameter Selection. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2328–2339.

[34] James C Schatzman. 1996. Accuracy of the discrete Fourier transform and the fast Fourier transform. *SIAM Journal on Scientific Computing* 17, 5 (1996), 1150–1166.

[35] Parikshit Shah, Nikhil Rao, and Gongguo Tang. 2015. Sparse and low-rank tensor decomposition. *Advances in neural information processing systems* 28 (2015).

[36] Shaden Smith, Jee W Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. 2017. FROSTT: The formidable repository of open sparse tensors and tools.

[37] Shaden Smith, Niranjay Ravindran, Nicholas D Sidiropoulos, and George Karypis. 2015. SPLATT: Efficient and parallel sparse tensor-matrix multiplication. In *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 61–70.

[38] Jiahao Su, Jingling Li, Xiaoyu Liu, Teresa Ranadive, Christopher Coley, Tai-Ching Tuan, and Furong Huang. 2022. Compact neural architecture designs by tensor representations. *Frontiers in artificial intelligence* 5 (2022), 728761.

[39] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. 2007. Less is more: Compact matrix decomposition for large sparse graphs. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 366–377.

[40] Nico Vervliet, Otto Debals, Laurent Sorber, and Lieven De Lathauwer. 2014. Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis. *IEEE Signal Processing Magazine* 31, 5 (2014), 71–79.

[41] Hongcheng Wang and Narendra Ahuja. 2004. Compact representation of multidimensional data using tensor rank-one decomposition. *vectors* 1, 5 (2004).

[42] Jiang Wang, Zicheng Liu, Ying Wu, and Junsong Yuan. 2012. Mining actionlet ensemble for action recognition with depth cameras. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 1290–1297.

[43] Zi Yang, Junnan Shan, and Zheng Zhang. 2022. Hardware-efficient mixed-precision CP tensor decomposition. *arXiv preprint arXiv:2209.04003* (2022).

[44] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. 2018. Learning compact recurrent neural networks with block-term tensor decomposition. In *CVPR*. 9378–9387.

[45] Qibin Zhao, Masashi Sugiyama, Longhao Yuan, and Andrzej Cichocki. 2019. Learning efficient tensor representations with ring-structured networks. In *ICASSP*. IEEE, 8608–8612.

[46] Bingyin Zhou, Fan Zhang, and Lizhong Peng. 2012. Compact representation for dynamic texture video coding using tensor method. *IEEE Transactions on Circuits and Systems for Video Technology* 23, 2 (2012), 280–288.

A Proofs

A.1 Proof of Theorem 2

Proof. Because each dimension I_k is evenly divided into B_k subparts, each sub-block has a shape of $I_1/B_1 \times \cdots \times I_D/B_D$. Now, considering each sub-block individually, for the k-mode, there exist a total of I_k/B_k hyperslices, each of which has $(D-1)$ shift parameters (corresponding to the possible movement directions). Extending this to all $k = 1, \ldots, D$, the total number of shift parameters is given by

$$(D-1) \cdot \sum_{k=1}^{D} I_k/B_k.$$

Since the total number of sub-blocks is $\prod_{k=1}^{D} B_k$ and this process is repeated L times, the total number of parameters is given by

$$L(D-1) \cdot \sum_{k=1}^{D} I_k/B_k \cdot \prod_{k=1}^{D} B_k,$$

which completes the proof. □

Table 5: Reconstruction errors at large compression sizes (lower is better; best within each pair is highlighted).

Dataset	D_8			S_8			Uber			Action		
Compressed Size (MB)	116	128	140	116	128	140	34	38	42	220	245	270
CP	0.2488	0.1912	0.1204	0.2051	0.1687	0.1019	0.2189	0.1598	0.1149	0.1935	0.1520	0.1028
CP+PuzzleTensor	0.2185	0.1774	0.1089	0.1764	0.1493	0.0822	0.2048	0.1515	0.1084	0.1825	0.1463	0.0944
Tucker	0.2929	0.2060	0.1493	0.2205	0.1793	0.1176	0.2343	0.1564	0.1040	0.2127	0.1565	0.1178
Tucker+PuzzleTensor	0.2597	0.1788	0.1267	0.1886	0.1407	0.0955	0.2171	0.1376	0.0912	0.1901	0.1426	0.0969
TT	0.2857	0.2088	0.1481	0.2093	0.1556	0.1070	0.2165	0.1497	0.0959	0.2083	0.1438	0.0956
TT+PuzzleTensor	0.2514	0.1739	0.1276	0.1646	0.1364	0.0864	0.1922	0.1329	0.0881	0.1859	0.1372	0.0843

Dataset	PEMS-SF			Activity			Stock			NYC		
Compressed Size (MB)	418	465	512	425	470	515	720	800	880	470	520	570
CP	0.1761	0.1373	0.0953	0.1621	0.1353	0.0893	0.1644	0.1401	0.0958	0.1739	0.1375	0.0955
CP+PuzzleTensor	0.1697	0.1316	0.0902	0.1573	0.1267	0.0836	0.1559	0.1276	0.0871	0.1694	0.1311	0.0909
Tucker	0.1928	0.1452	0.1136	0.1777	0.1419	0.1069	0.1753	0.1456	0.1072	0.1974	0.1431	0.1179
Tucker+PuzzleTensor	0.1804	0.1328	0.0887	0.1708	0.1240	0.0851	0.1674	0.1288	0.0850	0.1866	0.1395	0.1055
TT	0.1966	0.1358	0.0924	0.1839	0.1305	0.0904	0.1883	0.1275	0.0939	0.1980	0.1346	0.0934
TT+PuzzleTensor	0.1744	0.1289	0.0802	0.1722	0.1188	0.0793	0.1668	0.1163	0.0797	0.1843	0.1276	0.0861

A.2 Proof of Theorem 3

Proof. Assuming that each dimension I_k is evenly divided into B_k subparts, each sub-block has a shape of $I_1/B_1 \times \cdots \times I_D/B_D$. Each sub-block undergoes the following process: (1) A $(D-1)$-dimensional FFT is applied to each mode-k hyperslice, (2) an element-wise multiplication is performed with the phase tensor, and (3) an inverse FFT is applied. Each step has a computational cost of

$$O\left(\frac{I_{\neq k}}{B_{\neq k}} \log\left(\frac{I_{\neq k}}{B_{\neq k}}\right)\right), \quad O\left(\frac{I_{\neq k}}{B_{\neq k}}\right), \quad \text{and} \quad O\left(\frac{I_{\neq k}}{B_{\neq k}} \log\left(\frac{I_{\neq k}}{B_{\neq k}}\right)\right),$$

respectively, where $I_{\neq k} = \prod_{j \neq k} I_j$ and $B_{\neq k} = \prod_{j \neq k} B_j$. Applying this to all I_k/B_k slices, the total complexity becomes

$$O\left(\frac{I_1 \cdots I_D}{B_1 \cdots B_D} \log\left(\frac{I_{\neq k}}{B_{\neq k}}\right)\right).$$

Extending this to all dimensions, we obtain

$$O\left(\sum_{k=1}^{D} \frac{I_1 \cdots I_D}{B_1 \cdots B_D} \log\left(\frac{I_{\neq k}}{B_{\neq k}}\right)\right) = O\left(\frac{I_1 \cdots I_D}{B_1 \cdots B_D} \sum_{k=1}^{D} \log\left(\frac{I_{\neq k}}{B_{\neq k}}\right)\right).$$

Since

$$\sum_{k=1}^{D} \log\left(\frac{I_{\neq k}}{B_{\neq k}}\right) = \log\left(\frac{I_1^{D-1} \cdots I_D^{D-1}}{B_1^{D-1} \cdots B_D^{D-1}}\right) = (D-1) \log\left(\frac{I_1 \cdots I_D}{B_1 \cdots B_D}\right),$$

the total complexity simplifies to

$$O\left((D-1) \cdot \frac{I_1 \cdots I_D}{B_1 \cdots B_D} \log\left(\frac{I_1 \cdots I_D}{B_1 \cdots B_D}\right)\right).$$

Because there exist $B_1 \cdots B_D$ sub-blocks in total and the entire process is repeated L times, the final time complexity is

$$O\left(L(D-1) \cdot (I_1 \cdots I_D) \cdot \log\left(\frac{I_1 \cdots I_D}{B_1 \cdots B_D}\right)\right).$$

This completes the proof. □

Table 6: Hyperparameter settings for sub-block decomposition. Each dataset is truncated so that its dimensions are multiples of the chosen block sizes. $l(n)$ is defined as $l(4) = 0$ and $l(n) = n - 5$ for $n = 5, \cdots, 8$.

Dataset	Original Size	Truncated Size	Block Size
$\{D_n\}_{n=4,\cdots,8}$	$(2^n, 2^n, 2^n)$	$(2^n, 2^n, 2^n)$	$[2^{l(n)}, 2^{l(n)}, 2^{l(n)}]$
$\{S_n\}_{n=4,\cdots,8}$	$(2^n, 2^n, 2^n)$	$(2^n, 2^n, 2^n)$	$[2^{l(n)}, 2^{l(n)}, 2^{l(n)}]$
Uber	(183, 24, 1140)	(180, 24, 1140)	[9, 1, 20]
Action	(100, 570, 567)	(100, 570, 560)	[4, 19, 16]
PEMS-SF	(963, 144, 440)	(960, 144, 440)	[32, 6, 11]
Activity	(337, 570, 320)	(336, 570, 320)	[14, 19, 10]
Stock	(1317, 88, 916)	(1312, 88, 912)	[41, 4, 38]
NYC	(265, 265, 28, 35)	(264, 264, 28, 34)	[22, 22, 2, 2]

B Additional Experiments

We further evaluate PuzzleTensor at substantially larger compression budgets, as reported in Table 5. Across every dataset and decomposition family, attaching PuzzleTensor yields the lowest reconstruction error, often with margins exceeding 5-10% relative to its direct baseline. These results confirm that the proposed shift-based transformation remains effective even when aggressive compression leaves little redundancy to exploit.

C Hyperparameters

We set the number of layers in PuzzleTensor to $L = 3$, and use the Adam optimizer [23] with the learning rate $\eta = 0.001$ for the training. Table 6 summarizes the block sizes $[B_1, \ldots, B_D]$ used for sub-block decomposition across eight datasets. Note that each dataset is truncated so that each dimension is evenly divisible by the corresponding block count; leftover sub-blocks are not shifted for computational efficiency.