A Robust Low-rank Tensor Decomposition and Quantization based Compression Method

Yudian Ouyang[1], Kun Xie[1,2,*], Jigang Wen[3], Gaogang Xie[4], Kenli Li[1]
[1] College of Computer Science and Electronics Engineering, Hunan University, Changsha, China
[2] Hangzhou Institute of Advanced Technology, Hangzhou, China
[3] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China
[4] Institute of Computer Network Information Center, Chinese Academy of Sciences, Beijing, China
{yudian,xiekun,lkl}@hnu.edu.cn, wenjigang@hnust.edu.cn, xie@cnic.cn
* Corresponding author

Abstract—**Tensor data is widely used in fields such as smart grids, cloud systems, and deep learning. As the scale of this data increases, storage and transmission costs rise significantly. Many tensor data exhibit low-rank structures, offering the potential for data compression through low-rank decomposition techniques. Tucker decomposition, a typical low-rank decomposition technique, achieves data compression and interpretability by capturing complex data correlations and representing the original tensor with a compact core tensor and factor matrices. However, the compression ratio provided by Tucker decomposition is often insufficient, particularly for large-scale tensors. To tackle this issue, we propose a robust low-rank tensor compression method that leverages Tucker decomposition with quantization and coding. Initially, we establish a robust Tucker decomposition framework that decomposes the low-rank tensor into a core tensor and factor matrices using well-designed Tucker rank-setting rules. This framework effectively handles noise and missing values. Subsequently, we conduct an in-depth analysis of the numerical characteristics of the core tensor and factor matrices within the Tucker decomposition framework. Based on their distinct characteristics, we design tailored quantization and coding schemes to compress the core tensor and factor matrices, respectively, thereby significantly improving the compression ratio of Tucker decomposition while maintaining high accuracy. Through extensive experiments on four publicly available datasets (which can form 3 or 4 order tensors), we demonstrate that our approach can achieve compression ratios $4\times$ - $10\times$ higher than the best competitor, with recovery errors improved by 8% - 42%.**

Index Terms—**Data compression, Tensor decomposition, Quantization and coding**

I. Introduction

A tensor is a high-dimensional array extensively used for data representation across various domains, including smart grids [1], [2], cloud systems [3], [4], computer network [5], [6], and deep learning [7], [8]. However, the increasing size of tensor data can lead to significant space and bandwidth overheads when processing and transmitting large tensors [9]–[12]. These high costs burden data-intensive computations, posing challenges for applications with limited resources.

For example, network monitoring data, as a three-mode indicator-interval-period tensor, traditionally has a minute-level (5min) collection cycle. Some indicators (e.g., CPU occupancy, interface spikes, Wi-Fi interference) require a second-level (1s) collection cycle, resulting in a 300-fold expansion of data volume. Moreover, real-world tensor data often contains noise and missing values. In wireless sensor networks, sensors collect data in dynamic environments and transmit it to cloud servers for analysis and archiving. The sensed data often experiences loss or corruption during the process due to uncontrollable wireless environments, sensor node failures, resource limitations and malicious attacks [13].

Therefore, **the critical challenge is to design a robust data compression method that significantly reduces storage space and transmission bandwidth even in the presence of noise and missing values.**

Traditional compression algorithms often rely on transform coding, which follows a trilogy of a spatial-to-frequency domain transform (such as Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT)), coefficient quantization, and entropy coding [14]–[17]. DCT and DWT require data with *smooth spatiotemporal patterns* [15], [16]. However, real-world tensor data often includes multiple indicators with high diversity [2], leading to a low compression ratio. Some semantic compression techniques exploit attribute column dependencies to compress tabular data. Early methods based on clustering [18], storing values of cluster representatives and recovering discarded data via classification or regression trees [19] or saving only differences from cluster representatives [20]. [21] attempts to capture correlation using multi-column codes or special sorting orders. Recent efforts aim to uncover complex relationships across multiple columns in tabular data, employing advanced models like Bayesian networks [22] and autoencoders [23]. However, these methods typically remain two-dimensional, emphasizing column dependencies while *overlooking higher-order correlations*.

Recent studies have demonstrated that many tensor data exhibit low-rank structures [24]. This low-rank characteristic stems from the high spatiotemporal correlation in tensor data. This means that some rows, columns, or depths can be expressed as linear combinations of others. The low-rank structure presents an opportunity to compress data via low-rank decomposition.

In low-rank decomposition, the most fundamental techniques are CANDECOMP PARAFAC (CP) decomposition [25], [26] and Tucker decomposition [27]. For an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, Tucker decomposition decomposes it into a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \ldots \times R_N}$ and N factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ ($n = 1, ..., N$, and R_n is the Tucker rank). Tucker decomposition is highly interpretable and can be viewed as a multilinear generalization of principal component analysis [28]. When the core tensor $\mathcal{G}$ is a cube with all entries on the super-diagonal 1 and all other entries 0, Tucker decomposition degenerates into CP decomposition. Hence, we will focus on Tucker decomposition in this paper.

As depicted in Fig. 1, an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ with $\prod_1^N I_n$ amount of data, after undergoing Tucker decomposition, transforms into a core tensor $\mathcal{G}$ and N factor matrices $\{\mathbf{U}^{(n)}\}_1^N$ with $\prod_1^N R_n + \sum_1^N I_n R_n$ amount of data. If the tensor $\mathcal{X}$ exhibits a low-rank structure ($R_n \leq I_n$), Tucker decomposition can efficiently compress it with a compression ratio of $\prod_1^N I_n / (\prod_1^N R_n + \sum_1^N I_n R_n)$. As $R_n \ll I_n$ in practice, Tucker decomposition can reduce the data volume by nearly $N - 1$ orders of magnitude.

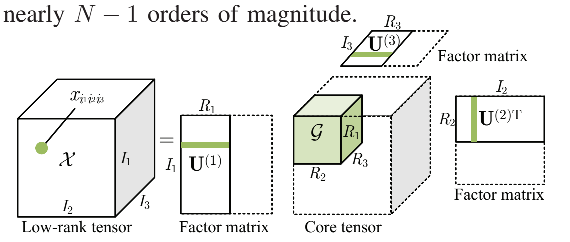


Fig. 1. Tucker decomposition for a 3-mode tensor.

While Tucker decomposition has shown promise for data compression, designing a robust data compression scheme still faces the following challenges:

- **Determining the Tucker ranks is difficult.** The Tucker ranks $R_1, ..., R_N$ directly influence the size and accuracy of compressed data. However, due to the presence of missing and noisy data, selecting appropriate ranks is challenging. Although the Bayesian optimization [29] can search for good ranks, it may involve multiple trials of Tucker decomposition to search the ranks and can suffer from extended computing resources.
- **Additional compression of the core tensor and factor matrices is required.** Even when Tucker ranks are appropriately set, the decomposed core tensor and N factor matrices still occupy an amount of data, especially for large tensors. The factor matrices and the core tensor represent the original tensor's principal components and interaction coefficients. Well-designed compression schemes are necessary since small precision loss of factor matrices and core tensor can lead to large reconstruction errors in the original tensor. To the best of our knowledge, no existing studies investigate the numerical characteristics of the core tensor and factor matrices to design algorithms for their compression.

To address these challenges, we propose a robust low-rank tensor compression framework based on TUCKER decomposition with QUANTIZATION and coding (TKQT). In TKQT, we introduce two novel metrics to facilitate the setting of Tucker ranks. We further design tailored compression methods for the core tensor and factor matrices, which reduce the data size significantly while having minimal impact on the accuracy. The framework also exhibits robustness, allowing for denoising and estimating missing values during compression and decompression procedures. The contributions of this paper are summarized as follows:

- We propose a robust Tucker decomposition framework that imposes orthogonal constraints on factor matrices and employs truncated singular value decomposition (SVD) for solving. This method enables the factor matrices to capture the principal components or main features within large-scale tensor data. We introduce two innovative metrics for setting the Tucker ranks with a single high-order SVD.
- We experimentally found that the core tensor displays a skewed data distribution, with many small and few large values. To compress it accurately, we designed a non-uniform quantization and entropy coding scheme, converting the core tensor into a small codebook and a bitstream. Compared with traditional uniform quantization, our scheme provides higher accuracy in compressing the core tensor.
- Under the orthogonal constraint, the factor matrices' values are limited in the range of $[0, 1]$. Even a slight deviation in these matrices can significantly alter the direction of the entire principal component, thereby affecting the reconstruction accuracy of the original tensor. Exploiting this value range characteristic, we propose a shift quantization and binary coding compression scheme to guarantee high-accuracy compression of the factor matrices.
- We conduct extensive experiments on four public datasets (which can form 3 or 4 order tensors) and implement 20 baselines for performance comparison. The experimental results demonstrate that our TKQT can achieve compression ratios up to $\mathbf{4\times}$ - $\mathbf{10\times}$ higher than the best competitors, with error rate improvements of $\mathbf{8\%}$ - $\mathbf{42\%}$.

Due to dataset constraints and the limited capacity of our computer, we restrict the evaluation of TKQT's effectiveness to 3 or 4 order tensors. TKQT is based on Tucker decomposition. Numerous studies have explored scalable approaches to Tucker decomposition for large-scale high-order (≥ 4) tensors [30]–[33]. Based on this research, we expect that TKQT can be extended to handle large datasets with high-order tensors in future work.

II. RELATED WORK

While many data compression tools exist, this paper focuses on tensor compression via low-rank decomposition. A variety of low-rank tensor decomposition techniques exist, such as CP decomposition, Tucker decomposition, Tensor-SVD (t-SVD) [34], Tensor-Train (TT) [35], and Tensor-Ring (TR) [36], among which CP and Tucker decomposition are the most typical and popular formats, t-SVD is specially designed for the three-mode tensor, TT and TR have low storage costs but lack interpretability.

In low-rank tensor decomposition-based data analysis, tensor completion is a well-studied area aiming to recover tensor missing values through limited sampling. This area's algorithms include auxiliary information methods, scalable approaches, dynamic methods, and neural networks enhanced tensor completion. Auxiliary information tensor completion [37]–[39] uses side or cross-domain knowledge to aid missing data recovery, excelling in mitigating data sparsity and cold-start issues in recommendation systems. Scalable approaches [40]–[43] employ parallel or distributed computing systems to decompose large-scale tensor data within limited memory efficiently, offering solutions for data explosion problems. Dynamic methods [44]–[46] are ideal for rapidly arriving data streams, using online or incremental learning for real-time completion. Neural networks enhanced algorithms [5], [6], [47] extend traditional tensor decomposition to the nonlinear domain by capturing high-dimensional and complex correlations through new interaction functions.

Compared to tensor completion, low-rank tensor compression is relatively less studied. TT and TR decompose tensors of arbitrary order into a chain product of multiple small 3-mode tensors with a low storage cost. TR can be viewed as an extension of TT. They are primarily utilized for compressing deep neural networks, including fully connected layers [48], [49], convolutional neural networks [49], and recurrent neural networks [50], [51]. Due to the lack of interpretability, TT and TR are generally not applied to data compression tasks.

Different from TT and TR, Tucker decomposition offers strong interpretability and can extract complex correlations from low-rank tensor data. Gao et al. employ Tucker decomposition for feature extraction and data reduction of high-dimensional and multi-source heterogeneous data in cloud systems [3]. In similar scenarios, Zhang et al. combine various low-rank tensor decomposition models for effective feature representation and data fusion [4]. Zhao et al. and Guzmán et al. utilize Tucker decomposition for data compression in smart grid distribution systems, aiming to reduce dimensionality and noise by discarding smaller singular values in the data tensor [1], [2]. Despite these methods demonstrating the potential of Tucker decomposition to eliminate redundancy and extract critical information, their compression performance remains limited because the core tensor and factor matrices decomposed from Tucker decomposition are still large.

While low-rank decomposition is extensively studied, we found no robust data compression-focused research. Exploiting good features of Tucker decomposition while designing tailored compression algorithms for the core tensor and factor matrices, we significantly improved the compression ratio with high data accuracy even though the data have noise and missing values.

III. Preliminaries

Definition 1 (Unfolding). The n-mode unfolding of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ is a matrix denoted by $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{k \neq n} I_k}$ which map the element $x_{i_1,i_2,\ldots,i_N}$ of $\mathcal{X}$ to index (i_n, j) with

$$j = 1 + \sum_{k=1,k\neq n}^{N} (i_k - 1) \times \prod_{m=k+1}^{N} I_m \tag{1}$$

Definition 2 (Singular value decomposition). The singular value decomposition (SVD) of $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times J}$ is defined as:

$$\mathbf{X}_{(n)} = \mathbf{U}^{(n)} \mathbf{\Sigma}^{(n)} \mathbf{V}^{(n)\mathrm{T}} \tag{2}$$

where $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times I_n}$ and $\mathbf{V}^{(n)} \in \mathbb{R}^{J \times J}$ are both unitary matrices. $\mathbf{\Sigma}^{(n)} = \mathrm{diag}(\sigma_1, ..., \sigma_{I_n}) \in \mathbb{R}^{I_n \times J}$ is a diagonal matrix with the diagonal elements (i.e., singular values) arranged in descending order. The leading R_n-dimensional left singular subspace of $\mathbf{X}_{(n)}$ is defined as $\mathbf{U}_{R_n}^{(n)} = \mathbf{U}^{(n)}(:, 1 : R_n)$.

Definition 3 (n-mode matrix product). The n-mode (matrix) product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{R \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{U}$ and is of size $I_1 \times ...I_{n-1} \times R \times I_{n+1} \times ... \times I_N$. Elementwise, we have

$$(\mathcal{X} \times_n \mathbf{U})_{i_1 \ldots i_{n-1} r i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \ldots i_N} u_{r i_n} \tag{3}$$

Definition 4 (Tucker decomposition). Tucker decomposition is to decompose a tensor into a core tensor multiplied by a matrix along each mode. The N-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ can be expressed as

$$\mathcal{X} = \left[\!\left[\mathcal{G}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, ..., \mathbf{U}^{(N)} \right]\!\right] = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} ... \times_N \mathbf{U}^{(N)} \tag{4}$$

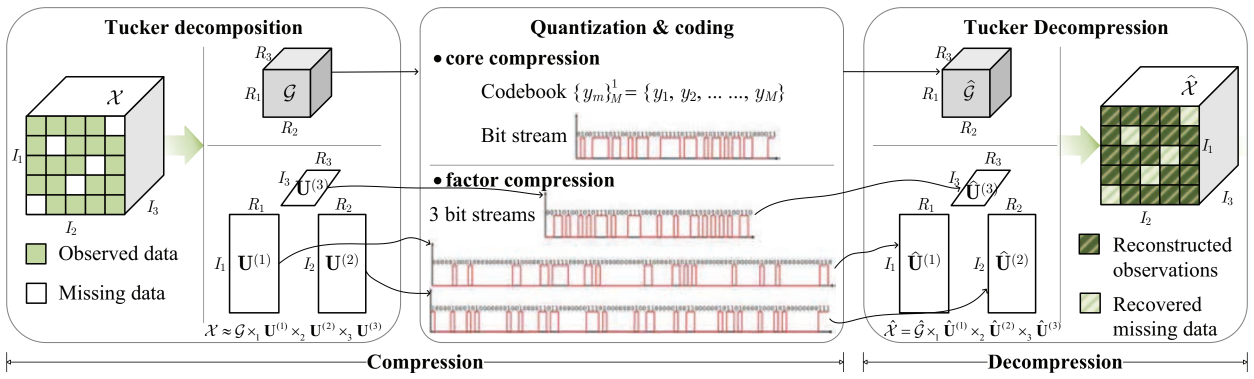
where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \ldots \times R_N}$ is the core tensor, $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ is the factor matrix corresponding to the n-th mode, $R_1, R_2, ..., R_N$ represent the Tucker ranks. Each component $x_{i_1 i_2 \cdots i_N}$ in tensor $\mathcal{X}$ is calculated by

$$x_{i_1 i_2 \cdots i_N} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \cdots r_N} u_{i_1 r_1}^{(1)} u_{i_2 r_2}^{(2)} \cdots u_{i_N r_N}^{(N)} \tag{5}$$

Tucker decomposition can be viewed as a multilinear generalization of principal component analysis [28], where the factor matrix $\mathbf{U}^{(n)}$ represents the basis matrix or principal components of the tensor in each mode for discovering the underlying latent features from the tensor, and each element of the core tensor $\mathcal{G}$ represents the degree of interaction across the different modalities or components. The number of columns of the n-th factor matrix, R_n, indicates the number of latent features (components).

IV. Problem and Overview

Real-world data can often be represented as high-dimensional tensors $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$. For instance, in smart grid systems, each Phasor Measurement Unit (PMU) can be modeled as a three-mode tensor, where I_1 denotes the number of monitoring indicators (such as voltage, current, frequency, etc.), I_2 indicates the number of timestamps captured in one period, and I_3 represents the total number of periods. Here, the element $x_{i_1 i_2 i_3}$ records the measurement of the i_1-th indicator at the i_2-th timestamp during the i_3-th period. Similarly, in



Tucker decomposition
Quantization & coding
Tucker Decompression
core compression
Codebook $\{y_m\}_M^1 = \{y_1, y_2, \ldots, y_M\}$
Bit stream
factor compression
3 bit streams
Observed data
Missing data
Reconstructed observations
Recovered missing data
$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}$
$\hat{\mathcal{X}} = \hat{\mathcal{G}} \times_1 \hat{\mathbf{U}}^{(1)} \times_2 \hat{\mathbf{U}}^{(2)} \times_3 \hat{\mathbf{U}}^{(3)}$
Compression
Decompression

Fig. 2. Solution overview.

computer networks, three-mode tensors are typically used for measurement data like traffic or latency, with I_1, I_2, and I_3 denoting the number of origin nodes, destination nodes, and time slots. The element $x_{i_1 i_2 i_3}$ signifies the traffic or latency from origin node i_1 to destination node i_2 at time slot i_3. As discussed in Section I, $\mathcal{X}$ often experiences missing values and noise, resulting in an incomplete tensor. Our objective is to **design a robust compression method based on Tucker decomposition that can significantly reduce storage and transmission overheads while eliminating noise and recovering missing values**.

We propose a robust low-rank tensor compression framework that exploits Tucker decomposition with quantization and coding. As illustrated in Fig. 2, our solution consists of two compression modules and a decompression module:

Tucker Decomposition transforms an N-mode low-rank tensor into a core tensor and N factor matrices.

Quantization and Coding comprises two essential processes, **core compression** and **factor compression**, which compress the core tensor into a bitstream with a codebook file and compress the N factor matrices into N bit streams.

Tucker Decompression decompresses the codebook and bitstreams into a core tensor and N factor matrices and then reconstructs the original tensor based on the decompressed core tensor and factor matrices while recovering missing values.

V. Tucker Compression

A. Orthogonal Tucker Decomposition

We utilize Tucker decomposition to project an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ into a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \ldots \times R_N}$ and N factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}{}_{n=1}^{N}$, where $R_n \leq I_n$. We begin with two key questions within the framework of Tucker decomposition: (a) How can we find the most informative core tensor and factor matrices among all solutions of Tucker decomposition? (b) How can we identify a robust Tucker decomposition that accurately approximates the original $\mathcal{X}$ by effectively removing noise from the data and accurately recovering the missing data?

To address the first question, we impose an orthogonal constraint on the columns of the factor matrices, i.e., $\mathbf{U}^{(n)\mathrm{T}}\mathbf{U}^{(n)} = \mathbf{I}_{R_n}$. The columns of the factor matrices reveal the underlying features of the tensor, where R_n denotes the number of latent features in the n-th mode. Orthogonal vectors are mutually independent and exhibit significant differences, allowing for the most informative latent feature representation using a compact amount of data.

For the second question, we adopt truncated SVD-based algorithms to solve the orthogonal Tucker decomposition and handle the noise. Truncated SVD retains the leading R_n-dimensional left singular subspace of the unfolding matrix $\mathbf{X}_{(n)}$ as the factor matrix $\mathbf{U}^{(n)}$. This enhances the Tucker decomposition's noise reduction capability by eliminating singular vectors associated with small singular values, which typically represent noise and data interdependencies [52]. To accurately recover data with missing entries, we initially replace the missing values in the tensor with zeros, and then update these missing values with estimations obtained in each iteration until the model converges. After obtaining the final core tensor and factor matrices, we reconstruct the original tensor and recover the missing values (Section VII).

Based on the above thoughts, to find the core tensor $\mathcal{G}$ and orthogonal factor matrices $\{\mathbf{U}^{(n)}\}_1^N$, we can minimize the error between the original tensor $\mathcal{X}$ and its approximation:

$$\begin{aligned} \underset{\mathcal{G};\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(N)}}{\arg\min} & \left\| \mathcal{X} - \left[\!\left[\mathcal{G}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right]\!\right] \right\|^2 \\ \text{s.t. } & \mathbf{U}^{(n)\mathrm{T}}\mathbf{U}^{(n)} = \mathbf{I}_{R_n} \end{aligned} \tag{6}$$

With the orthogonal constraints, the core tensor can be calculated by

$$\mathcal{G} = \mathcal{X} \times_1 \mathbf{U}^{(1)\mathrm{T}} \times_2 \mathbf{U}^{(2)\mathrm{T}} \ldots \times_N \mathbf{U}^{(N)\mathrm{T}} \tag{7}$$

Further, (6) can be deduced as an optimization problem to maximize $\|\mathcal{G}\|^2$:

$$\begin{aligned} \underset{\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(N)}}{\arg\max} & \left\| \mathcal{X} \times_1 \mathbf{U}^{(1)\mathrm{T}} \ldots \times_N \mathbf{U}^{(N)\mathrm{T}} \right\|^2 \\ \text{s.t. } & \mathbf{U}^{(n)\mathrm{T}}\mathbf{U}^{(n)} = \mathbf{I}_{R_n} \end{aligned} \tag{8}$$

To solve (8), we first employ the higher-order singular value decomposition (HOSVD) [53] to initialize the basis factor matrices $\{\mathbf{U}_0^{(n)}\}_{n=1}^N$, then utilize the higher-order orthogonal iteration (HOOI) [54] to iteratively update the orthogonal matrices $\{\{\mathbf{U}_l^{(n)}\}_{n=1}^N\}_{l=1}^L$ until convergence, where l is the index of loop. Both HOSVD and HOOI are essentially based on truncated SVD.

HOSVD: We first compute the leading R_n-dimensional left singular subspace of $\mathbf{X}_{(n)}$ to initialize the basis factor matrices $\mathbf{U}_0^{(n)}$, where $n = 1, .., N$. Since the truncated subspace clips out the smallest $(I_n - R_n)$ singular values that represent the noise of the data, HOSVD has the effect of noise reduction.

HOOI: Then, we rewrite (8) as (9) to solve the n-th factor matrices alternately

$$\arg\max_{\mathbf{U}^{(n)}} \left\|\mathbf{U}^{(n)\mathrm{T}}\mathbf{W}\right\| \quad \text{s.t. } \mathbf{U}^{(n)\mathrm{T}}\mathbf{U}^{(n)} = \mathbf{I}_{R_n} \tag{9}$$

where $\mathbf{W} = \mathbf{X}_{(n)}(\mathbf{U}^{(N)} \otimes \cdots \otimes \mathbf{U}^{(n+1)} \otimes \mathbf{U}^{(n-1)} \cdots \otimes \mathbf{U}^{(1)})$ and $\otimes$ denotes Kronecker product. The solution $\mathbf{U}^{(n)}$ of (9) can be set as the leading R_n-dimensional left singular vectors of the matrix $\mathbf{W}$. By iteratively updating $\{\{\mathbf{U}_l^{(n)}\}_{n=1}^N\}_{l=1}^L$, we can obtain a set of orthogonal factor matrices $\{\mathbf{U}^{(n)}\}_{n=1}^N$. During the iteration, the core tensor $\mathcal{G}_l$ can be easily obtained by (7). HOOI removes data noise using the same truncation operation as HOSVD.

At each iteration, we replace the missing entries in $\mathcal{X}$ with the current latest approximation , i.e.

$$\mathcal{X}_{l+1} = \mathcal{P}_\Omega(\mathcal{X}_l) + \mathcal{P}_{\bar{\Omega}}([\![\mathcal{G}_l; \mathbf{U}_l^{(1)}, \mathbf{U}_l^{(2)}, ..., \mathbf{U}_l^{(N)}]\!]) \tag{10}$$

where $\mathcal{X}_0 = \mathcal{X}$. Ω, $\bar{\Omega}$, and $\mathcal{P}_\Omega$ denote the set of all observed elements, all missing elements, and the projection on Ω, respectively. By (10), we solve Tucker decomposition using only observations and iteratively correct for missing values.

B. *Rank Selection Rules*

The Tucker ranks $R_1, ...R_N$ play a significant role as they directly influence the size and accuracy of the compressed data. Choosing appropriate ranks for Tucker decomposition is challenging. Setting ranks too small will lead to poor reconstruction accuracy for $\mathcal{X}$, while setting ranks too large will unnecessarily increase the size of the core tensor and factor matrices with little improvement in accuracy.

Bayesian Optimization (BO) [29] is one of the state-of-the-art method for determining the Tucker ranks. It transforms the selection of Tucker ranks into an optimization problem with N-parameter, aiming to minimize reconstruction loss and model complexity [55]. With a defined range of ranks and a trial budget, BO iteratively searches for the best ranks among all possible combinations. It employs a surrogate model to estimate the performance of candidate rank combinations and an acquisition function to guide the search. While BO can find the globally optimal rank given enough search space and trials, it is computationally expensive and time-consuming because it requires a Tucker decomposition for each trial.

The number of non-zero singular values of the unfolding matrices determines the ranks of Tucker decomposition. However, real-world tensors, typically not perfectly low-rank due to missing values and noise, exhibit all singular values larger than 0. Many small singular values may tend towards 0, complicating rank selection via SVD.

Let $\sigma_0, ..., \sigma_{I_n-1}$ denote the singular values of the unfolding matrix $\mathbf{X}_n \in \mathbb{R}^{I_n \times J}$. For a low-rank matrix $\mathbf{X}_n$, there exists a rank R_n such that $\sigma_0 \gg \sigma_{R_n} \approx \cdots \approx \sigma_{I_n-1} \approx 0$. Our goal is to find R_n as the rank of the n-th tensor mode, $n = 1, .., N$. Specifically, we propose two metrics based on singular values:

$$\Delta(i) = \frac{\sigma_{i-1}}{\sigma_i} - 1, \quad \Phi(i) = \frac{\sigma_i}{\sigma_0} \tag{11}$$

where $\Delta(i) \geq 0$ and $0 \leq \Phi(i) \leq 1$. A smaller $\Delta(i)$ indicates a smaller gap between two consecutive singular values. A smaller $\Phi(i)$ suggests a smaller ratio of σ_i compared to σ_0.

Before presenting our rank selection rules based on these two metrics, we illustrate the $\Delta(i)$ and $\Phi(i)$ of various datasets (described in Section VIII) in Fig. 3. This reveals two distinct cases. In the first case, the $\Delta(i)$ curves exhibit a stable point, which can be set as the rank since $\Delta(i)$ reflects the decreased rate of singular values. In the second case, the $\Delta(i)$ curves lack a clear stabilization point due to noise-induced fluctuations, and the rank can then be set to the point where $\Delta(i)$ achieves its minimum value. We define the following rules:

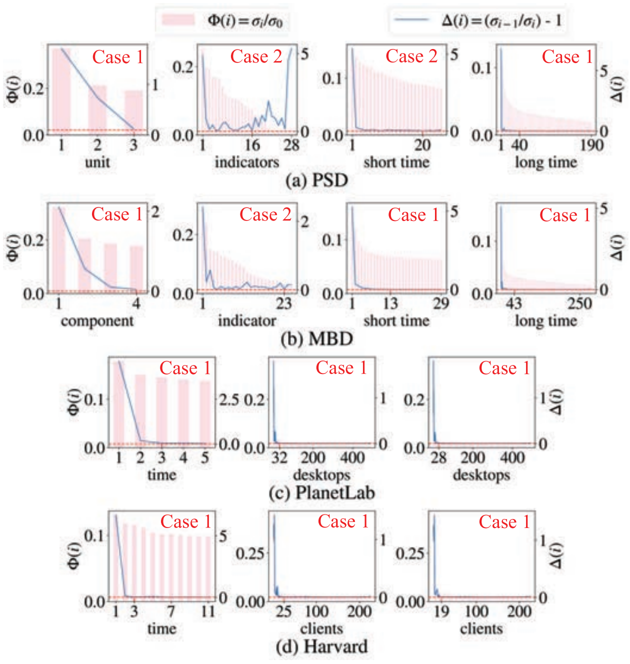


Fig. 3. Analysis of the Tucker ranks.

Case 1: The $\Delta(i)$ curve has a stable point. If $\Delta(i) < \epsilon$, and the next several consecutive $\Delta(i)$ values do not fluctuate drastically, we set $R_n = i$. Here, ϵ represents a small threshold, typically set to 0.01.

Case 2: If no stable point can be found in the $\Delta(i)$ curve, the i that minimizes $\Delta(i)$ within a certain range is chosen as the rank. Specifically, the range is determined by $\Phi(I_n - 1)$. If $\Phi(I_n - 1) < \epsilon$, we conduct the search at $I_n/2$. Otherwise, it means a relatively high noise level, and we perform the search at $2I_n/3$.

As shown in Fig. 3, most of the curves align with Case 1, such as (c) and (d), exhibiting a stable point that indicates the rank. In contrast, the curves of the indicator mode in (a) and (b) adhere to Case 2. The rank settings for the four data sets, determined according to our rules, are marked in the figure. In Section VIII-C, experimental comparisons with BO demonstrate that our proposed rank selection method closely approximates the optimal rank while significantly reducing computation time compared to BO.

VI. Quantization and Coding

A. Core Compression

The core tensor, representing the interaction among the original tensor's principal components or latent features, must be compressed as much as possible while preserving key information. Although quantization and coding are potential solutions, designing an appropriate quantization method to preserve accuracy is challenging.

We investigate the numerical characteristic of the core tensor in Fig. 4 and observe that the core tensor has a skewed data distribution, with a dense distribution of small values and few large values.

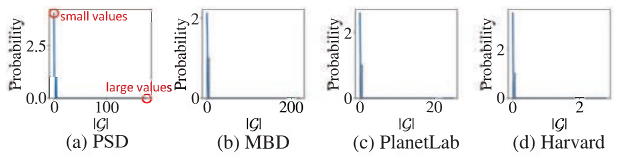
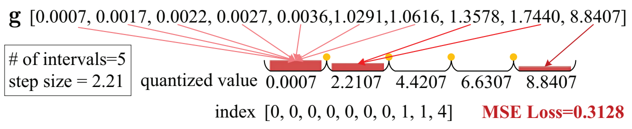


Fig. 4. The distribution of the core tensor.

The straightforward quantization method is uniform quantization, which quantizes the data into uniform intervals, each of which has a fixed size = (maximum value - minimum value) / (number of quantization intervals - 1). Fig. 5 gives the result of a vector $\mathbf{g}$ with skewed data distribution under uniform quantization. As shown in the figure, when the data has a skewed distribution, the uniform quantization tends to quantize the densely distributed small values into a single interval (index = 0), and the quantized values (2.2107) computed by a fixed step size are usually far away from the real data values (1.3578), which ultimately leads to a large quantization loss (0.3128).



g [0.0007, 0.0017, 0.0022, 0.0027, 0.0036,1.0291,1.0616, 1.3578, 1.7440, 8.8407]
# of intervals=5
step size = 2.21
quantized value 0.0007 2.2107 4.4207 6.6307 8.8407
index [0, 0, 0, 0, 0, 0, 0, 1, 1, 4]
MSE Loss=0.3128

Fig. 5. Uniform quantization.

To solve the above problems, we design a non-uniform quantization and entropy coding scheme to compress the core tensor. The procedures include: 1) preprocessing; 2) non-uniform quantization; 3) setting the sign bit; 4) entropy coding.

Step 1 Preprocessing. Let $\mathbf{g} \in \mathbb{R}^{R_1 \ldots R_n} = |\text{vec}(\mathcal{G})|$ and $\mathbf{s} \in \{0,1\}^{R_1 \ldots R_n} = (\text{vec}(\mathcal{G}) < 0)$, we first extract the absolute values and signs of the elements of the vectorized core tensor. We give a toy example of quantization coding for $\mathcal{G} \in \mathbb{R}^{7\times7\times3}$ in Fig. 7, with an initial size of $7\times7\times3\times4 = 588$ Byte, where each entry occupies 4 Byte.

Step 2: Non-uniform Quantization. This step uses data distribution to determine quantization values and decision boundaries. Given the probability density function (PDF) of the core tensor, we can assign higher quantization loss weights to more frequent values, allowing denser regions of small values to have finer quantization intervals, thus minimizing overall loss. Let $f(\mathbf{g})$ be the PDF of $\mathbf{g}$, the optimal quantizer problem $\mathcal{Q}^*$ is defined by minimizing the quantization error:

$$\mathcal{Q}^*(\mathbf{g}) = \arg\min_{\mathcal{Q}} \int (\mathcal{Q}(\mathbf{g}) - \mathbf{g})^2 f(\mathbf{g}) d\mathbf{g} \tag{12}$$

In (12), we use $f(\mathbf{g})$ to assign varied weights to the errors $(\mathcal{Q}(\mathbf{g}) - \mathbf{g})^2$, giving higher weights to more frequent values, consequently reducing quantization errors.

By assigning the number of quantization intervals M, we can convert (12) into (13), which aims to find the set of optimal decision boundaries $\{b_m\}_0^M$ and quantized values $\{y_m\}_1^M$ (a.k.a. codebook).

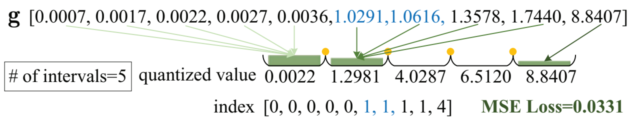
$$\mathcal{L}_q = \sum_{m=1}^{M} \int_{bm-1}^{bm} (\mathbf{g} - y_m)^2 f(\mathbf{g}) d\mathbf{g} \tag{13}$$

Taking the partial derivatives of (13) for y_m and b_m, respectively, we can obtain:

$$\begin{aligned} y_m &= \frac{\int_{b_{m-1}}^{b_m} \mathbf{g} f(\mathbf{g}) d\mathbf{g}}{\int_{b_{m-1}}^{b_m} f(\mathbf{g}) d\mathbf{g}}, \; 1 \le m \le M \\ b_m &= \begin{cases} 0, & m = 0 \\ \frac{1}{2}(y_m + y_{m+1}), & 1 \le m \le M-1 \\ +\infty, & m = M \end{cases} \end{aligned} \tag{14}$$

The optimal solution of (14) can be easily solved by Lloyd's algorithm [56]. With the optimal decision boundaries $\{b_m\}_0^M$ and codebook $\{y_m\}_1^M$, we can compute the quantized index vector $\mathbf{z} \in \{0, ..., M-1\}^{R_1 \ldots R_n}$ corresponding to $\mathbf{g}$. If $g_i \in [b_{m-1}, b_m)$, than $z_i = m - 1$ and $\mathcal{Q}(g_i) = y_m$. After the elements in $\mathbf{g}$ have been mapped to indexes through the decision boundaries $\{b_m\}_0^M$, we only need to save the index vector $\mathbf{z}$ and the codebook $\{y_m\}_1^M$ to get the quantization of $\mathbf{g}$ by $\mathcal{Q}(\mathbf{g}) = \{y_m\}_1^M[\mathbf{z}]$.

Fig. 6 illustrates that our method can spread the dense values over different intervals and compute quantized values (1.2981) that are closer to the original values (1.3578) based on the data distribution. Our non-uniform quantization, with a lower loss (0.0331), outperforms traditional uniform quantization. This advantage is further shown in Section VIII-C using four datasets.



g [0.0007, 0.0017, 0.0022, 0.0027, 0.0036,1.0291,1.0616, 1.3578, 1.7440, 8.8407]
# of intervals=5
quantized value 0.0022 1.2981 4.0287 6.5120 8.8407
index [0, 0, 0, 0, 0, 1, 1, 1, 1, 4]
MSE Loss=0.0331

Fig. 6. Non-uniform quantization.

We specify $M = 2^{bit_c}$ in the deployment and the number of bits bit_c is a important parameter in core compression. A larger bit_c results in larger number of quantization intervals and a lower loss. Generally, a larger core requires a more quantization interval budget. Therefore, in our experiments, we determine the value of bit_c based on the size of the core: $bit_c = \text{Round}\left(\log_{10}\prod_{n=1}^{N} R_n\right) + 2$.

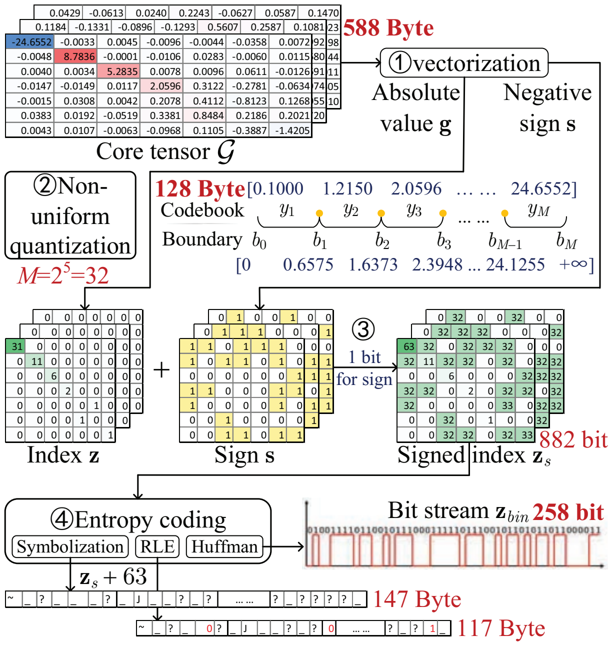


Fig. 7. Core compression.

[A toy example]: As shown in Step 2 of Fig. 7, we set the quantization intervals $M = 2^5 = 32$ and solve the optimal quantization boundaries $\{b_m\}_0^{32}$ and codebook $\{y_m\}_1^{32}$ for a core tensor of size $7\times7\times3$ by (14). Then, each absolute value is mapped into a quantized index according to the boundaries. For $g_1 = 24.6552 \in [b_{31}, b_{32})$, the corresponding index $z_1 = 31$, and its quantized value $Q(g_1) = y_{32} = 24.6552$. In this case, the codebook requires $32 \times 4 = 128$ Byte for storage.

Step 3 Setting the sign bit. Since the elements of $\mathbf{z}$ are integers in $[0, M)$, each element of $\mathbf{z}$ can be written as a bit_c bit binary digit. Considering that the core tensor $\mathcal{G}$ has positive and negative values, we set the highest $(bit_c + 1)$-th bit as 1 to indicate a negative sign and 0 to indicate a positive sign. The signed index vector can be calculated by $\mathbf{z}_s = \mathbf{z} + M\mathbf{s}$. Since $\mathbf{z}_s$ occupies bit_c+1 bits per element, storing $\mathbf{z}_s$ requires $R_1...R_n(bit_c + 1)$ bits.

[A toy example]: As shown in Step 3 of Fig. 7, $s_1 = 1$ means that g_1 is originally a negative value. As $bit_c = 5$, we write z_1 as $z_1 = 31 = (11111)_2$, then $z_{s1} = (111111)_2 = 63$. In this example $\mathbf{z}_s$ consumes $7 \times 7 \times 3 \times (5 + 1) = 882$ bits.

Step 4 Entropy coding. After non-uniform quantization, many small values of the core tensor are divided into the 0-th interval; thus, $\mathbf{z}_s$ has many repeated elements. We further adopt entropy coding to reduce the size of $\mathbf{z}_s$. We use the displayable symbols in the ASCII table instead of the integer values in $\mathbf{z}_s$, followed by run-length encoding (RLE) and Huffman coding to get a more compact bit stream $\mathbf{z}_{bin}$.

[A toy example]: As shown in Step 4 of Fig. 7, we symbolize $\mathbf{z}_s$ from $0-63$ integers to $63-126$ ASCII displayable symbols. In RLE, we record the frequency for symbols with the number of consecutive occurrences ≥ 3 and subtract the recorded frequencies by 3. Lastly, the RLE string is compressed into a 258 bit bitstream by Huffman coding.

With the above procedures, we finally compress the core tensor $\mathcal{G}$ into a bit stream $\mathbf{z}_{bin}$ and a codebook file $\{y_m\}_1^M$, which significantly reduces the size of the core tensor. In the toy example we give, the compression ratio of the core tensor is $588 \times 8/(128 \times 8 + 258) = 3.67$. In actual storage and transmission, the compressed data size can be further reduced by rounding off excessive decimal places when writing the codebook.

B. Factor Compression

Under the orthogonality constraint (6) on the Tucker decomposition, factor matrices are unitary, representing the tensor's principal components with absolute values in $[0, 1]$. Small deviations can shift these components, affecting tensor reconstruction. Designing an accurate compression algorithm that considers data features is challenging. We propose a binary-based shift quantization and coding algorithm to compress factor matrices in four steps.

Step 1 Preprocessing. Same as the core tensor, for the factor matrix $\mathbf{U}^{(n)}$, we treat the absolute values $\left|\mathbf{U}^{(n)}\right|$ and signs $(\mathbf{U}^{(n)} < 0)$ separately.

Step 2 Shift quantization. We left shifted $(1 \ll bit_f)$ the elements of $\left|\mathbf{U}^{(n)}\right|$ by bit_f bits to magnify the values by $2^{bit_f} - 1$ fold, and then round the decimal portion to quantize the elements from floating-point to integer. As shown in Step 2 of Fig. 8, most of the precision of the original decimal has been preserved in the integer after the left shift.

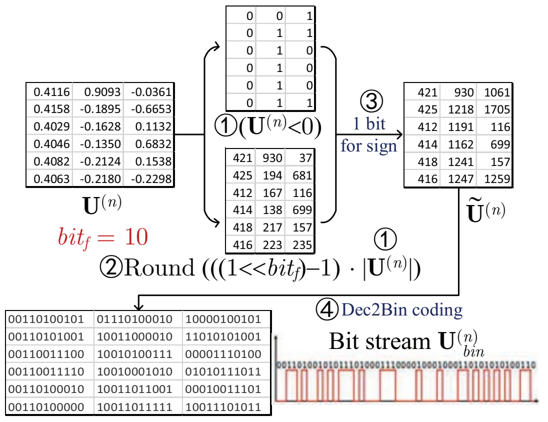
Step 3 Setting the sign bit. After Step 2, we can represent each element of $\left|\mathbf{U}^{(n)}\right|$ as an integer with bit_f bits. We keep the highest bit to represent the sign. As shown in Step 3 of Fig. 8, the $(bit_f + 1)$-th bit is set to 1 for negative numbers and 0 for positive numbers.

Integrating steps 1 to 3, we can obtain the quantization of the factor matrix $\mathbf{U}^{(n)}$ by (15). No codebook is needed here because the quantized values are all integers in $\left[0, 2^{bit_f+1}\right)$.

$$\begin{aligned}\tilde{\mathbf{U}}^{(n)} &= \underbrace{(1 \ll bit_f) \cdot (\mathbf{U}^{(n)} < 0)}_{\text{negative flag}} \\ &\quad + \underbrace{\text{Round}\left(((1 \ll bit_f) - 1) \cdot \left|\mathbf{U}^{(n)}\right|\right)}_{\text{left-shift and quantization}}\end{aligned} \tag{15}$$

Step 4 Binary coding. As shown in Step 4 of Fig. 8, we encode the quantized matrix $\tilde{\mathbf{U}}^{(n)}$ into a bit stream $\mathbf{U}_{bin}^{(n)}$ of length $I_n R_n(bit_f + 1)$ by decimal to binary coding. In the end, the elements of $\mathbf{U}^{(n)}$ transform from a 4 Byte floating-point number to a $bit_f + 1$ bit binary digit, with a theoretical compression ratio of $32/(bit_f + 1)$.

The precision loss in factor compression is confined to the $\text{Round}()$ function in Step 2, and this loss can be controlled by bit_f. In (15), the left shift operation amplifies $\left|\mathbf{U}^{(n)}\right|$ by a factor of $2^{bit_f} - 1$. Let's set $\tau = \left\lfloor \log_{10}(2^{bit_f} - 1) \right\rfloor$, which



0.4116	0.9093	-0.0361
0.4158	-0.1895	-0.6653
0.4029	-0.1628	0.1132
0.4046	-0.1350	0.6832
0.4082	-0.2124	0.1538
0.4063	-0.2180	-0.2298

0	0	1
0	1	1
0	1	0
0	1	0
0	1	0
0	1	1

421	930	37
425	194	681
412	167	116
414	138	699
418	217	157
416	223	235

421	930	1061
425	1218	1705
412	1191	116
414	1162	699
418	1241	157
416	1247	1259

00110100101	01110100010	10000100101
00110101001	10011000010	11010101001
00110011100	10010100111	00001110100
00110011110	10010001010	01010111011
00110100010	10011011001	00010011101
00110100000	10011011111	10011101011

Fig. 8. Factor compression.

means $2^{bit_f} - 1 \geq 10^{\tau}$. The $\mathrm{Round}()$ function truncates the digits after the τ-th decimal place of $\left|\mathbf{U}^{(n)}\right|$. Therefore, we set $bit_f = 10$ in our experiments and have $2^{10} - 1 = 1023 \geq 10^3$, ensuring high accuracy. This means the precision loss is bounded by 10^{-3}. The ablation experiments in Section VIII-C also confirm that $bit_f = 10$ effectively controls precision loss while achieving a high compression ratio.

VII. Tucker Decompression

In practical application scenarios, such as cloud systems, after receiving compressed files $\{y_m\}_1^M$, $\mathbf{z}_{bin}$, and $\{\mathbf{U}^{(n)}_{bin}\}_1^N$ from the edge node or terminals, the cloud data center can decompress the core tensor $\hat{\mathcal{G}}$ and factor matrices $\{\hat{\mathbf{U}}^{(n)}\}_1^N$ by decoding and de-quantization, and reconstruct the original tensor $\hat{\mathcal{X}}$ based on Tucker decomposition.

We first decompress the codebook file $\{y_m\}_1^M$ and the bitstream $\mathbf{z}_{bin}$ into a core tensor $\hat{\mathcal{G}}$. The decoding algorithm can obtain $\mathbf{z}_s$ directly from $\mathbf{z}_{bin}$. Based on $\mathbf{z}_s$, the index vector $\mathbf{z}$ and the symbol vector $\mathbf{s}$ can be computed by

$$\begin{aligned} \mathbf{s} &= (\mathbf{z}_s \gg bit_c)\&1 \\ \mathbf{z} &= !(1 \ll bit_c)\&\mathbf{z}_s \end{aligned} \tag{16}$$

The decompression of the core tensor is

$$\hat{\mathcal{G}} = \mathrm{Reshape}\left((\mathbf{1} - 2\mathbf{s}) \cdot \{y_m\}_1^M[\mathbf{z}], (R_1, ..., R_N)\right) \tag{17}$$

Then, we decompress the factor bitstreams $\{\mathbf{U}^{(n)}_{bin}\}_1^N$ into the quantized matrices $\{\tilde{\mathbf{U}}^{(n)}\}_1^N$ by binary to decimal coding. After that, we de-quantize $\{\tilde{\mathbf{U}}^{(n)}\}_1^N$ into $\{\hat{\mathbf{U}}^{(n)}\}_1^N$ by the right-shift operation of (18). $\hat{\mathbf{U}}^{(n)}$ is the decompression of factor matrix $\mathbf{U}^{(n)}$.

$$\hat{\mathbf{U}}^{(n)} = \underbrace{\left(1 - 2((\tilde{\mathbf{U}}^{(n)} \gg bit_f)\&1)\right)}_{\text{negative sign}} \cdot \underbrace{\frac{\left(!(1 \ll bit_f)\&\tilde{\mathbf{U}}^{(n)}\right)}{((1 \ll bit_f) - 1)}}_{\text{right-shift and de-quantization}} \tag{18}$$

Finally, we obtain the core tensor $\hat{\mathcal{G}}$ and factor matrices $\{\hat{\mathbf{U}}^{(n)}\}_1^N$ from the observed elements of the tensor and use them to approximate the entire tensor by (19).

$$\hat{\mathcal{X}} = \hat{\mathcal{G}} \times_1 \hat{\mathbf{U}}^{(1)} \times_2 \hat{\mathbf{U}}^{(2)} ... \times_N \hat{\mathbf{U}}^{(N)} \tag{19}$$

Tucker reconstruction allows not only the estimate of the observed entries but also the recovery of the missing values by

$$\hat{x}_{i_1 i_2 \cdots i_N} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_N=1}^{R_N} \hat{g}_{r_1 r_2 \cdots r_N} \hat{u}^{(1)}_{i_1 r_1} \cdots \hat{u}^{(N)}_{i_N r_N} \tag{20}$$

Algorithm 1 The compression method of TKQT.

```
Input: An $N$-mode data tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$,
Output: A cookbook $\{y_m\}_1^M$,
        a bit stream of core tensor $\mathbf{z}_{bin}$,
        $N$ bit streams of factor matrices $\{\mathbf{U}^{(n)}_{bin}\}_1^N$,
        an approximation of original tensor $\hat{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$.
    ▷ Tucker decomposition
 1: for $n = 1, ..., N$ do                                          ▷ HOSVD
 2:     $[\mathbf{U}^{(n)}_0, \Sigma^{(n)}_0, \_] = \mathrm{SVD}(\mathbf{X}_{(n)})$;
 3:     Set rank $R_n$ by $\{\Delta(i)\}_0^{I_n-1}$, $\{\Phi(i)\}_0^{I_n-1}$, $\Sigma^{(n)}_0$;
 4:     $\mathbf{U}^{(n)}_0 = \mathbf{U}^{(n)}_0[:, 1:R_n]$;  ▷ $R_n$ leading left singular vectors
 5:     $\mathcal{G}_0 \leftarrow \mathcal{X}$, $\{\mathbf{U}^{(n)}_0\}_1^N$ by (7).
 6: end for
 7: for $l = 1, 2, ...$ do                                           ▷ HOOI
 8:     $\mathcal{X}^l \leftarrow \mathcal{X}^{l-1}$, $\mathcal{G}_l$, $\{\mathbf{U}^{(n)}_l\}_1^N$ by (10);
 9:     for $n = 1, ..., N$ do
10:         $\mathbf{W} = \mathbf{X}_{(n)l}(\mathbf{U}^{(N)}_l \otimes \cdots \otimes \mathbf{U}^{(n+1)}_l \otimes \mathbf{U}^{(n-1)}_l \cdots \otimes \mathbf{U}^{(1)}_l)$;
11:         $\mathbf{U}^{(n)}_l \leftarrow R_n$ leading left singular vectors of $\mathbf{W}$.
12:     end for
13:     $\mathcal{G}_l \leftarrow \mathcal{X}_l$, $\{\mathbf{U}^{(n)}_l\}_1^N$ by (7);
14:     if $\left\|\mathcal{G}^l - \mathcal{G}^{l-1}\right\|^2 < 1e-6$ then
15:         break;
16:     end if
17: end for
18: obtain $\mathcal{G}$ and $\{\mathbf{U}^{(n)}\}_1^N$.
    ▷ Core compression
19: Set $bit_c = \mathrm{Round}\left(\log_{10}\prod_{n=1}^N R_n\right) + 2$, $M = 2^{bit_c}$
20: Compute $\mathbf{g} = |\mathrm{vec}(\mathcal{G})|$ and $\mathbf{s} = (\mathrm{vec}(\mathcal{G}) < 0)$
21: Compute cookbook $\{y_m\}_1^M$ and index vector $\mathbf{z}$ by (14);
22: Compute signed index vector $\mathbf{z}_s \leftarrow \mathbf{z} + M\mathbf{s}$;
23: $\mathbf{z}_{bin} \leftarrow$ Entropy coding of $\mathbf{z}_s$;
    ▷ Factor compression
24: for $n = 1, ..., N$ do
25:     $\tilde{\mathbf{U}}_n \leftarrow$ Shift quantize $\mathbf{U}_n$ by (15);
26:     $\mathbf{U}^{(n)}_{bin} \leftarrow$ Decimal to binary coding of $\tilde{\mathbf{U}}_n$;
27: end for
28: transmission and storage $\{y_m\}_1^M$, $\mathbf{z}_{bin}$, $\{\mathbf{U}^{(n)}_{bin}\}_1^N$.
    ▷ Tucker reconstruction
29: $\mathbf{z}_s \leftarrow$ Entropy decoding of $\mathbf{z}_{bin}$;
30: $\mathbf{z}$, $\mathbf{s} \leftarrow \mathbf{z}_s$ by (16)
31: $\hat{\mathcal{G}} \leftarrow \{y_m\}_1^M$, $\mathbf{z}$, $\mathbf{s}$ by (17)
32: $\{\tilde{\mathbf{U}}^{(n)}\}_1^N$ Decimal to binary decoding of $\{\mathbf{U}^{(n)}_{bin}\}_1^N$;
33: $\{\hat{\mathbf{U}}^{(n)}\}_1^N$ Shift de-quantize $\{\tilde{\mathbf{U}}^{(n)}\}_1^N$ by (18);
34: $\hat{\mathcal{X}} \leftarrow \hat{\mathcal{G}}$,$\{\hat{\mathbf{U}}^{(n)}\}_1^N$ by (19);
35: return $\hat{\mathcal{X}}$.
```

Algorithm 1 summarizes the compression and decompression process of TKQT. The computational complexity of HOSVD is $\mathcal{O}(N\max\{I_n^N R_n\}_1^N)$. The HOOI iteratively executes the HOSVD L times, but the number of iterations L can be viewed as a constant due to the HOSVD initialization. For Lloyd's algorithm, RLE, and Huffman coding, the complexity is $\mathcal{O}(M\prod_{n=1}^N R_n)$, $\mathcal{O}(\prod_{n=1}^N R_n)$, and $\mathcal{O}(M'logM')$, respectively, where $M' = 2M+c$ and $c \leq 10$ is a constant introduced by RLE. The computational complexity of shift quantization is $\mathcal{O}(\sum_{n=1}^{N} I_n R_n)$. Since N and M can be regarded as constants and $\prod_{n=1}^{N} R_n \ll \prod_{n=1}^{N} I_n$ in practice, the final computational complexity of TKQT is $\mathcal{O}(\max\{I_n^N R_n\}_1^N)$.

VIII. Experiments

A. Experimental Setup

Datasets The four datasets utilized for evaluation are:

- PSD [57] is a collection of 15 power system datasets that measure 26 synchrophasors (or indicators) in a simulated smart grid with four PMUs. Each dataset contains at least 4966 records. These are reshaped into 15 4D tensors of size $4 \times 29 \times 26 \times 191$.
- MBD [58] contains run-time performance data from a cluster (including five system components: one master and four slaves) of a big data batch processing system. It measures 26 indicators every half minute over a total of 3 days, forming a 4D tensor of size $5 \times 26 \times 30 \times 288$.
- PlanetLab [59] measures the round-trip times (RTTs) between 490 desktops over 18 time slots, forming three 3D tensors of size $490 \times 490 \times 6$.
- Harvard [60] contains RTT data collected from 226 Azureus clients over 72 hours, forming six 3D tensors of size $226 \times 226 \times 12$.

To simulate missing data, we randomly divide measurements into two portions (9:1), with 90% observed and 10% missing. For the PSD and MBD datasets, which feature a wide range of values, we normalize different indicators and units separately [58]. The RTT datasets are standardized by dividing by their maximum values.

Metrics Three metrics utilized for evaluation are:

- **Reconstruction errors (REs)**: We calculate Normalized Mean Absolute Error (NMAE) and Normalized Root Mean Square Error (NRMSE) by comparing the ground truth x_{ijk} with their reconstructed values $\hat{x}_{ijk}$ in observed set (Obs.), missing set (Miss.) and all set (All), respectively.

$$\mathrm{NMAE} = \sum\nolimits_{(i,j,k)} |x_{ijk} - \hat{x}_{ijk}| \Big/ \sum\nolimits_{(i,j,k)} |x_{ijk}|$$
$$\mathrm{NRMSE} = \sqrt{\sum\nolimits_{(i,j,k)} (x_{ijk} - \hat{x}_{ijk})^2} \Big/ \sqrt{\sum\nolimits_{(i,j,k)} x_{ijk}^2} \tag{21}$$

- **Compression ratio (CR)**:

$$\mathrm{CR} = \text{Size of original data}/\text{Size of compressed data}. \tag{22}$$

Competitors We compare our TKQT with 20 methods.

- **Tensor decomposition techniques:** TT [35] and TR [36]. TT decomposition represents a tensor as a sequence of smaller tensors, organized in a linear structure. TR decomposition is an extension of TT decomposition, representing tensors as a ring of smaller tensors.
- **The standard compression methods:** 7z [61], Brotli [62], Zstd [63], Bzip2 [64], Gzip [65] and SZ3 [66], where the first 5 are standard compression tools. SZ3 is an error-bound compression method, and the error-bound is set to $1e{-}3$. These methods can only compress the observed elements directly (REs of Obs. ≈ 0) and cannot recover missing values (REs of Miss. $= 1$). Therefore, we only compare the compression ratio with them.
- **Transform coding based compression methods:** DCT9, DCT99, DCT999, DWT9, DWT99, DWT999. We reproduce the scheme in [15], using DCT and DWT to convert the tensor data from the frequency domain to the spatial domain, then retaining the first 90%, 99% and 99.9% large coefficients as the compressed data, and finally inverting the quantized coefficients to obtain the decompressed data.
- **Neural network based tensor completion models:** LTP [5] and CoSTCo [47] are missing data recovery models designed for network measurement and recommender system, respectively. These models have good missing data recovery accuracy owing to the powerful fitting ability of neural networks, but they cannot compress data. Therefore, we only compare the reconstruction errors with them.
- **Variant of TKQT**: TK, TKQT-wo-c, TKQT-wo-f, and TKQT-uc are variants of TKQT that without quantization and coding, without core compression, without factor compression, and with uniform quantization for the core tensor, respectively. The variants allow for both missing data recovery and compression.

Deployment We implement TKQT on Python, where HOSVD and HOOI are deployed by tensorly [67]. For TT and TR, we call tensorly to solve them based on SVD, and their ranks are obtained by Bayesian optimization. Since the performance of TT and TR is affected by the sequence of tensor modes, we rearrange the tensor modes and trial all possible combinations, ultimately reporting the best result for TT and TR. Specifically, we find experimentally that for a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, the most effective ordering is $[J_1, J_3, ..., J_{N-1}, J_N, ..., J_4, J_2]$, where the sequence $J_1, J_2, ..., J_N$ is the descending order of the sequence $I_1, I_2, ..., I_N$.

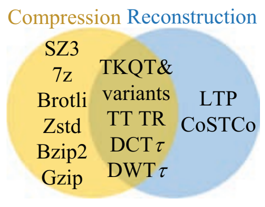


Fig. 9. The competitors.

B. Performance Comparison

Fig. 10 shows the compression ratios of TKQT with the comparison methods, where for TKQT, we additionally consider the CR of its two components, TK and quantization (QT). $\mathrm{CR}_{\mathrm{TKQT}} = \mathrm{CR}_{\mathrm{TK}} \times \mathrm{CR}_{\mathrm{QT}}$. Fig. 11, Fig. 12, and Table I exhibit the reconstruction errors of different methods on the four PMUs of PSD, the five nodes of MBD, and the two RTTs datasets, respectively.

TKQT has a superior CR among all the methods, which is **4**× - **10**× of the best competitor TT. Meanwhile, the REs of TKQT are **8**% - **42**% better than that of TT. TKQT is slightly inferior to TK in reconstruction performance but has a better

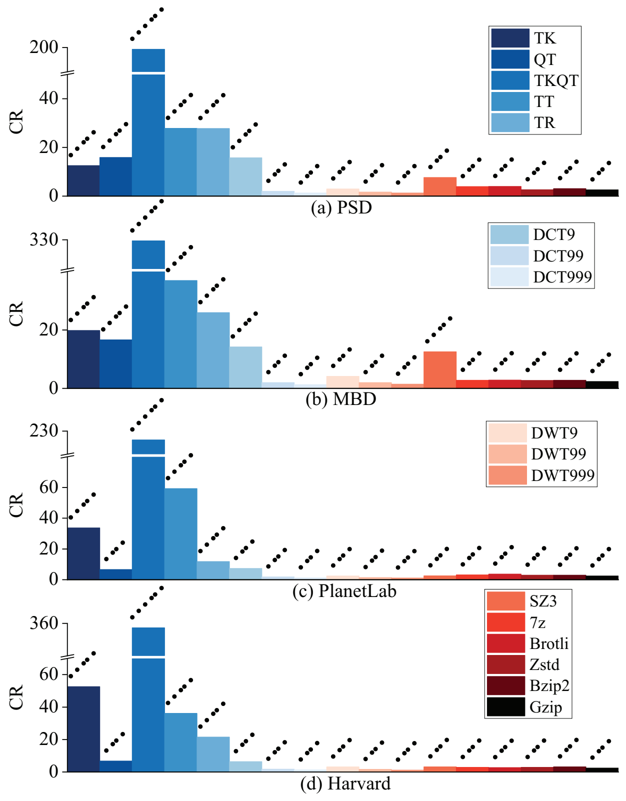


Fig. 10. The compression ratio of TKQT and comparison methods.

TABLE I
THE RECOVERY ERROR OF TKQT AND COMPARISON METHODS

	NMAE			NRMSE		
PlanetLab	Obs.	Miss.	All	Obs.	Miss.	All
TKQT	**0.0789**	**0.0823**	**0.0793**	**0.1179**	**0.1741**	**0.1246**
TK	0.0487	**0.0555**	**0.0494**	0.0749	0.1642	**0.0880**
TT	0.1215	0.1391	0.1232	0.1406	0.2175	0.1501
TR	0.1197	0.1651	0.1243	0.1397	0.2654	0.1568
DCT9	0.2539	0.5954	0.2881	0.2676	0.6292	0.3224
DCT99	0.0905	0.9277	0.1742	0.0929	0.9438	0.3108
DCT999	0.0291	0.9829	0.1245	0.0299	0.9879	0.3132
DWT9	0.2395	0.9423	0.3098	0.2598	0.9602	0.3908
DWT99	0.0670	0.9799	0.1583	0.0823	0.9874	0.3214
DWT999	**0.0183**	0.9936	0.1158	**0.0273**	0.9964	0.3157
LTP	0.0723	0.0729	0.0724	0.1479	0.1543	0.1486
CoSTCo	0.0718	0.0728	0.0719	0.1433	**0.1489**	0.1439
Harvard	Obs.	Miss.	All	Obs.	Miss.	All
TKQT	**0.0894**	**0.1031**	**0.0907**	**0.1344**	**0.4344**	**0.1839**
TK	0.0593	**0.0761**	**0.0610**	0.1090	0.4383	**0.1688**
TT	0.1312	0.1736	0.1354	0.1461	0.4837	0.2024
TR	0.1347	0.1884	0.1401	0.1513	0.4746	0.2039
DCT9	0.2535	0.6108	0.2892	0.2501	0.7014	0.3196
DCT99	0.0937	0.9292	0.1771	0.0857	0.9534	0.3009
DCT999	0.0304	0.9826	0.1255	0.0277	0.9900	0.3018
DWT9	0.2818	0.9365	0.3472	0.2701	0.9571	0.3882
DWT99	0.0768	0.9754	0.1665	0.0816	0.9871	0.3097
DWT999	**0.0210**	0.9922	0.1180	**0.0253**	0.9967	0.3037
LTP	0.1164	0.1167	0.1164	0.3858	**0.3921**	0.3865
CoSTCo	0.1527	0.1539	0.1528	0.3857	0.4033	0.3874

compression ratio. The maximum error gap between TKQT and TK is $\mathbf{0.04}$, but the CR of TKQT is $\mathbf{7\times}$ - $\mathbf{17\times}$ higher than that of TK. All the above observations illustrate that **TKQT can achieve a significantly better trade-off between compression ratio and reconstruction accuracy.**

Compared to the best competitor TT, the superiority of TKQT in CR can mainly be attributed to the gains brought by compression on the core tensor and factor matrices, especially for higher-order tensors. Because of the different data sizes and ranks, TK will show different compression ratios on different datasets. When the CR of TK is lower, the redundant information in the core tensor and factor matrices tends to be more, with QT working better. We discuss TKQT and its variants in more detail in Section VIII-C. The superiority of TKQT in terms of reconstruction performance can be attributed to the interpretability of TK (as described in Section III). When applied to low-rank data, TK decomposition can even achieve lower errors than neural network tensor completion algorithms (LTP and CoSTCo).

Among the three tensor decomposition technologies, TK has relatively high computational complexity and low compression ratio due to the fact that TT and TR has factors with linear complexity, whereas TK has an N-order core tensor that requires further compression. However, TT and TR cannot achieve the reconstruction accuracy of TK. Additionally, as shown in Section VIII-C, the computational complexity of TK is merely a constant multiple of that of TT and TR.

Standard compression methods generally yield low compression ratios. While SZ3 achieves the highest CR among these methods by somewhat sacrificing data precision, its CR still falls short of TKQT's. Moreover, it cannot handle missing values and noise. For transform-based compression methods, a larger quantization threshold results in lower REs and CR. However, both DCT and DWT underperform tensor decomposition techniques in terms of reconstruction and compression due to their inability to utilize multidimensional data information. LTP and CoSTCo require complex and time-consuming parameter training and lack inherent compression capabilities. In contrast, TKQT leverages the interpretability of TK and the well-designed quantization coding schemes to achieve REs comparable to or lower than those of LTP and CoSTCo, and compression ratios of $\mathbf{166}$ - $\mathbf{357}$, all at a low computational cost.

C. Model Analysis

Computational complexity analysis. The computational complexity of TKQT lies mainly in Tucker decomposition. For a matrix of $m \times n$, the computational complexity of its r-truncated SVD is $\mathcal{O}(rmn)$. Let $I = \max\{I_1, ..., I_N\}$ and $R = \max\{R_1, ..., R_N\}$, the computational complexity of TK, TT and TR is shown below.

1) TK requires 1 HOSVD and L HOOIs, and each HOSVD and HOOI performs N SVDs on unfolding matrices, with a computational complexity of $\mathcal{O}(\text{TK}) = \mathcal{O}((L+1)NI^N R)$.

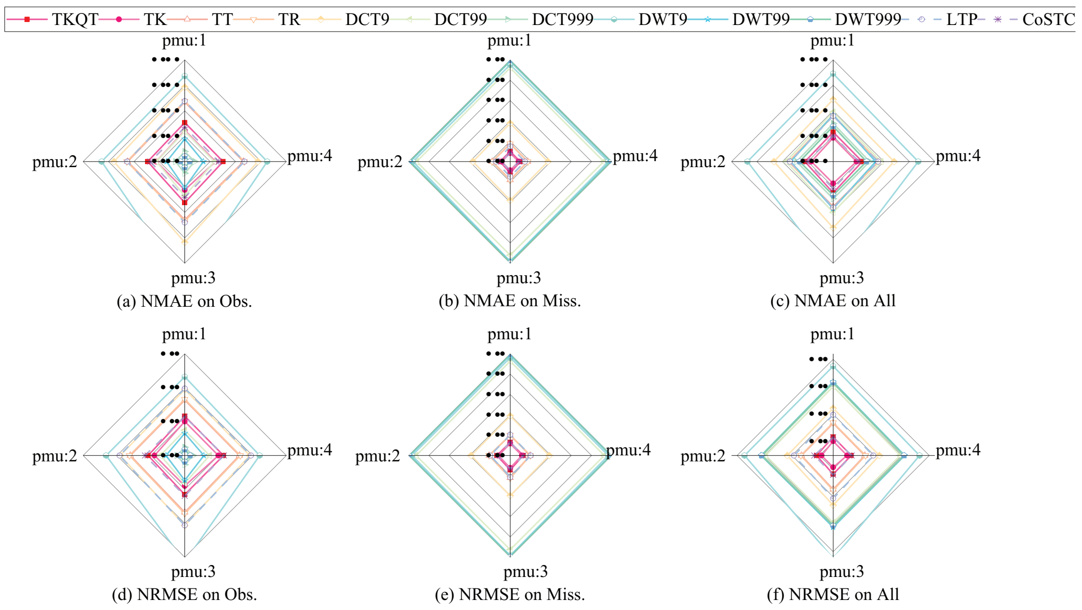


Fig. 11. The recovery error on the four PMUs of PSD.

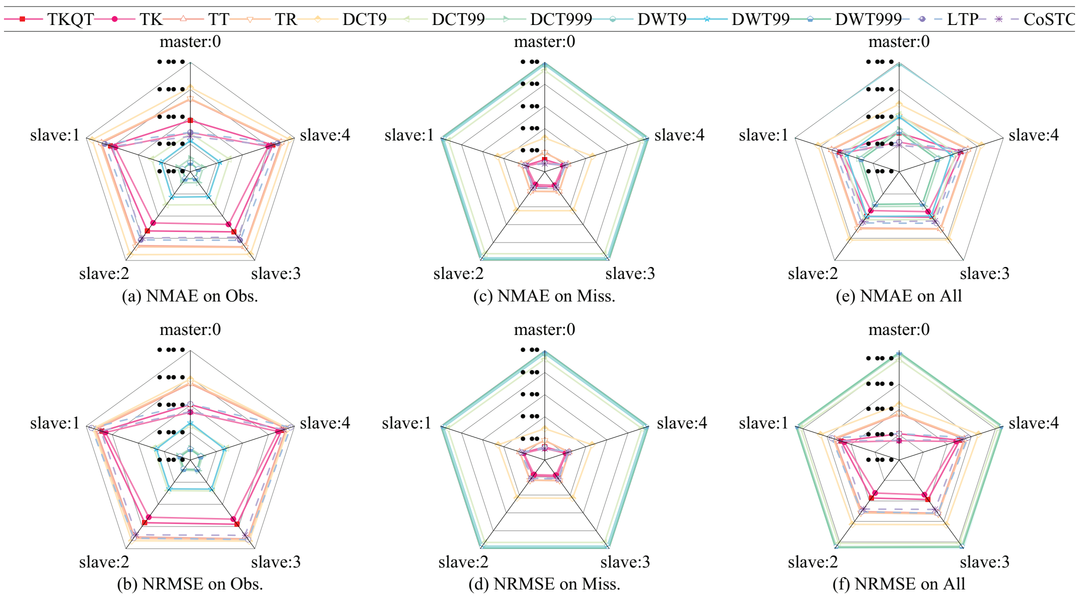


Fig. 12. The recovery error on the five nodes of MBD.

2) TT performs $N-1$ SVDs on increasingly smaller unfolding matrices with a computational complexity of $\mathcal{O}(\text{TT}) = \mathcal{O}(I^N R + I^{N-1}R^2 + ... + I^2R^2)$.
3) TR is similar to TT with a computational complexity of $\mathcal{O}(\text{TR}) = \mathcal{O}(I^N R^2 + I^{N-1}R^2 + ... + I^2R^2)$.

By comparing the highest order terms, we can obtain that $\mathcal{O}(\text{TT}) : \mathcal{O}(\text{TR}) : \mathcal{O}(\text{TK}) \approx 1 : R : (L+1)N$. In general, TK have a larger computational complexity compared to TT and TR. However, $(L+1)N$ can be counted as a constant compared to the highest order term $I^N R$. Therefore, the difference in computational complexity between TK and TT/TR is small.

Ablation experiment. Table II reports the REs and CR of TKQT and its four variants, leading to the following conclusions:

- Comparing TK and TKQT-wo-c, it is observable that TKQT-wo-c achieves REs close to that of TK (bounded by 10^{-3}). This suggests that the proposed factor compression algorithm can achieve high accuracy compression on factor matrices when $bit_f = 10$ is set.

- TKQT exhibits improvements of 39%, 19%, 5%, and 18% in REs (NMAE of Obs.) while slightly lowering CR compared to TKQT-uc. This indicates that the proposed non-uniform quantization compression better captures the numerical characteristics of the core tensor, thus increasing accuracy compared to uniform quantization.
- TKQT-wo-c enhances the CR by $1.18\times$, $2.45\times$, $1.21\times$, and $3.77\times$ compared to TK. Meanwhile, TKQT-wo-f improves the CR by $4.61\times$, $4.29\times$, $1.09\times$, and $1.13\times$ compared to TK. It is evident that for higher-order tensors ($N \geq 4$), core compression yields larger compression gains, while for 3D tensors, factor compression enhances compression ratios even more.

TABLE II
COMPRESSION PERFORMANCE OF TKQT AND ITS VARIANTS

	NMAE			NRMSE			CR
PSD	Obs.	Miss.	All	Obs.	Miss.	All	
TK	0.0702	0.0839	0.0715	0.0948	0.1175	0.0973	12.54
TKQT	0.0915	0.1025	0.0926	0.1132	0.1322	0.1152	199.31
TKQT-wo-c	0.0704	0.0841	0.0717	0.0949	0.1176	0.0973	14.82
TKQT-wo-f	0.0914	0.1024	0.0925	0.1132	0.1322	0.1151	57.86
TKQT-uc	0.1509	0.1527	0.1511	0.1811	0.1844	0.1814	238.44
MBD	Obs.	Miss.	All	Obs.	Miss.	All	CR
TK	0.1437	0.1553	0.1449	0.1637	0.1770	0.1650	19.84
TKQT	0.1617	0.1716	0.1627	0.1755	0.1873	0.1767	329.61
TKQT-wo-c	0.1438	0.1555	0.1450	0.1637	0.1771	0.1651	23.99
TKQT-wo-f	0.1616	0.1716	0.1626	0.1755	0.1872	0.1767	85.00
TKQT-uc	0.1991	0.2004	0.1992	0.2137	0.2146	0.2138	370.40
PlanetLab	Obs.	Miss.	All	Obs.	Miss.	All	CR
TK	0.0497	0.0560	0.0503	0.0764	0.1756	0.0913	36.16
TKQT	0.0810	0.0841	0.0813	0.1223	0.1824	0.1295	239.42
TKQT-wo-c	0.0506	0.0569	0.0512	0.0769	0.1757	0.0917	156.01
TKQT-wo-f	0.0805	0.0836	0.0808	0.1220	0.1822	0.1292	39.33
TKQT-uc	0.0853	0.0872	0.0855	0.1331	0.1823	0.1388	239.45
Harvard	Obs.	Miss.	All	Obs.	Miss.	All	CR
TK	0.0593	0.0761	0.0610	0.1090	0.4383	0.1688	52.58
TKQT	0.0894	0.1031	0.0907	0.1344	0.4344	0.1839	357.26
TKQT-wo-c	0.0600	0.0767	0.0616	0.1092	0.4385	0.1690	198.07
TKQT-wo-f	0.0891	0.1028	0.0904	0.1343	0.4342	0.1838	59.64
TKQT-uc	0.1084	0.1203	0.1096	0.1584	0.4360	0.2008	358.07

The validity of the rule-based ranks. We compare the proposed rule-based ranks with BO [55] and SV80. The optimization objective of BO is set to Score = NMAE + NRMSE + 1/CR, the set of candidate ranks is set to the full set, and the trial budget is set to 100. SV80 is a straightforward singular value-based rank-setting method that takes as rank the minimum number of singular values that exceed 80% of the sum of all singular values.

As shown in Table III, BO can find suitable ranks for Tucker decomposition while balancing recovery accuracy and compression ratio. TK (our rule-based ranks) performs close to BO, demonstrating the effectiveness of our proposed rank selection approach. SV80 tends to choose larger ranks, resulting in low compression ratios and reduced accuracy in recovering missing values due to overfitting noise in the data. Table III also displays the time required by these three methods to select the ranks for a tensor unit. It can be observed that TK is speedy, being able to set suitable ranks within 1 second, while BO takes $750\times$ - $917\times$ longer than TK.

TABLE III
COMPRESSION PERFORMANCE OF DIFFERENT RANK SELECTION SCHEMES

Method	NMAE		NRMSE		CR	Score	Time (s)
	Obs.	Miss.	Obs.	Miss.			
PSD	Rank: TK-[3, 16, 20, 40] SV80-[3, 13, 18, 126]						
BO	0.0719	0.0807	0.1033	0.1180	19.59	**0.2286**	**90.534**
TK	0.0702	0.0839	0.0948	0.1175	12.54	**0.2485**	**0.1072**
SV80	0.0681	0.1156	0.0905	0.1729	5.20	0.3663	0.0992
MBD	Rank: TK-[4, 23, 13, 43] SV80-[3, 13, 21, 192]						
BO	0.1186	0.1282	0.1433	0.1555	22.08	**0.3094**	**261.28**
TK	0.1198	0.1297	0.1414	0.1542	21.58	**0.3099**	**0.2850**
SV80	0.0847	0.1345	0.0959	0.1643	3.45	0.4846	0.2628
PlanetLab	Rank: TK-[3, 32, 28] SV80-[4, 299, 299]						
BO	0.0520	0.0584	0.0727	0.1596	38.29	**0.1642**	**401.62**
TK	0.0497	0.0560	0.0764	0.1756	36.16	**0.1693**	**0.5343**
SV80	0.0205	0.1063	0.0387	0.3207	1.79	0.6970	0.5886
Harvard	Rank: TK-[3, 25, 19] SV80-[8, 136, 136]						
BO	0.0518	0.0756	0.0833	0.4424	29.47	**0.2441**	**103.09**
TK	0.0593	0.0761	0.1090	0.4383	52.58	**0.2489**	**0.1242**
SV80	0.0306	0.1596	0.0462	0.5194	2.87	0.5553	0.1307

IX. CONCLUSION

This paper introduces a robust low-rank tensor compression framework, TKQT, based on Tucker decomposition coupled with quantization and coding. The framework incorporates several innovative techniques: We propose two rank determination rules that accurately determine Tucker ranks with low complexity; We design a high-accuracy core tensor compression algorithm based on non-uniform quantization and entropy coding; We propose a high-accuracy factor matrix compression algorithm based on shift quantization and binary coding. We implemented and compared our method against 20 state-of-the-art schemes using four publicly available datasets. Experimental results demonstrate that our method achieves the best balance between compression ratio and reconstruction accuracy. Given the ubiquity of tensor data in many applications, our method will offer new insights for compressing tensor data with high accuracy and an impressive compression ratio. Although we currently limit our evaluation of TKQT's effectiveness to datasets with 3 or 4 order tensors, our TKQT is based on Tucker decomposition and is expected to be applicable to larger datasets with high-order tensors. In future research, we will explore the scalability of TKQT and seek additional datasets to further verify its performance.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant 62025201, the National Natural Science Foundation of China/Research Grants Council (RGC) Joint Research Scheme under Grant 62321166652, and the Key Research and Development Program of Hunan Province under Grant 2023GK2001.

References

[1] H. Zhao and L. Ma, "Power distribution system stream data compression based on incremental tensor decomposition," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2469–2476, 2019.

[2] B. S. Guzmán, E. Barocio, P. Korba, A. Obushevs, and F. R. S. Sevilla, "Data compression for advanced monitoring infrastructure information in power systems based on tensor decomposition," *Sustainable Energy, Grids and Networks*, vol. 32, p. 100917, 2022.

[3] Y. Gao, G. Zhang, C. Zhang, J. Wang, L. T. Yang, and Y. Zhao, "Federated tensor decomposition-based feature extraction approach for industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 8541–8549, 2021.

[4] S. Zhang, L. T. Yang, J. Feng, W. Wei, Z. Cui, X. Xie, and P. Yan, "A tensor-network-based big data fusion framework for cyber–physical–social systems (cpss)," *Information Fusion*, vol. 76, pp. 337–354, 2021.

[5] Y. Ouyang, K. Xie, X. Wang, J. Wen, and G. Zhang, "Lightweight trilinear pooling based tensor completion for network traffic monitoring," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 2128–2137.

[6] K. Xie, H. Lu, X. Wang, G. Xie, Y. Ding, D. Xie, J. Wen, and D. Zhang, "Neural tensor completion for accurate network monitoring," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1688–1697.

[7] A.-H. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavskỳ, V. Glukhov, I. Oseledets, and A. Cichocki, "Stable low-rank tensor decomposition for compression of convolutional neural network," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*. Springer, 2020, pp. 522–539.

[8] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 8018–8027.

[9] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving IoT from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[10] A. Jonathan, A. Chandra, and J. Weissman, "Rethinking adaptability in Wide-Area stream processing systems," in *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, 2018.

[11] I. Psaras, O. Ascigil, S. Rene, G. Pavlou, A. Afanasyev, and L. Zhang, "Mobile data repositories at the edge," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[12] T. Lu, W. Xia, X. Zou, and Q. Xia, "Adaptively compressing {IoT} data on the resource-constrained edge," in *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[13] K. Xie, X. Ning, X. Wang, D. Xie, J. Cao, G. Xie, and J. Wen, "Recover corrupted data in sensor networks: A matrix completion solution," *IEEE Transactions on Mobile Computing*, vol. 16, no. 5, pp. 1434–1448, 2016.

[14] L. Liu, X. Chen, Z. Lu, L. Wang, and X. Wen, "Mobile-edge computing framework with data compression for wireless network in energy internet," *Tsinghua Science and Technology*, vol. 24, no. 3, pp. 271–280, 2019.

[15] A. Moon, J. Kim, J. Zhang, and S. W. Son, "Evaluating fidelity of lossy compression on spatiotemporal data from an iot enabled smart farm," *Computers and Electronics in Agriculture*, vol. 154, pp. 304–313, 2018.

[16] H.-C. Chen, K. T. Putra, S.-S. Tseng, C.-L. Chen, and J. C.-W. Lin, "A spatiotemporal data compression approach with low transmission cost and high data fidelity for an air quality monitoring system," *Future Generation Computer Systems*, vol. 108, pp. 488–500, 2020.

[17] K. T. Putra, H.-C. Chen, Prayitno, M. R. Ogiela, C.-L. Chou, C.-E. Weng, and Z.-Y. Shae, "Federated compressed learning edge computing framework with ensuring data privacy for pm2. 5 prediction in smart city sensing applications," *Sensors*, vol. 21, no. 13, p. 4586, 2021.

[18] H. Jagadish, J. Madar, and R. T. Ng, "Semantic compression and pattern extraction with fascicles," in *VLDB*, vol. 99, 1999, pp. 186–97.

[19] S. Babu, M. Garofalakis, and R. Rastogi, "Spartan: A model-based semantic compression system for massive data tables," *ACM SIGMOD Record*, vol. 30, no. 2, pp. 283–294, 2001.

[20] H. Jagadish, R. T. Ng, B. C. Ooi, and A. K. Tung, "Itcompress: An iterative semantic compression algorithm," in *Proceedings. 20th International Conference on Data Engineering*. IEEE, 2004, pp. 646–657.

[21] V. Raman and G. Swart, "How to wring a table dry: Entropy compression of relations and querying of compressed relations," in *Proceedings of the 32nd international conference on Very large data bases*. Citeseer, 2006, pp. 858–869.

[22] Y. Gao and A. Parameswaran, "Squish: Near-optimal compression for archival of relational datasets," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1575–1584.

[23] A. Ilkhechi, A. Crotty, A. Galakatos, Y. Mao, G. Fan, X. Shi, and U. Cetintemel, "Deepsqueeze: deep semantic compression for tabular data," in *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, 2020, pp. 1733–1746.

[24] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, and G. Zhang, "Accurate recovery of internet traffic data: A tensor completion approach," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[25] J. D. Carroll and J. J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[26] R. A. Harshman *et al.*, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis," *UCLA Working Papers in Phonetics*, 1970.

[27] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[28] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "MPCA: Multilinear principal component analysis of tensor objects," *IEEE transactions on Neural Networks*, vol. 19, no. 1, pp. 18–39, 2008.

[29] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[30] J. Zhang, J. Oh, K. Shin, E. E. Papalexakis, C. Faloutsos, and H. Yu, "Fast and memory-efficient algorithms for high-order tucker decomposition," *Knowledge and Information Systems*, vol. 62, no. 7, pp. 2765–2794, 2020.

[31] A. Traoré, M. Berar, and A. Rakotomamonjy, "Singleshot: a scalable tucker tensor decomposition," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[32] J. Oh, K. Shin, E. E. Papalexakis, C. Faloutsos, and H. Yu, "S-hot: Scalable high-order tucker decomposition," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017, pp. 761–770.

[33] T. G. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining," in *2008 Eighth IEEE international conference on data mining*. IEEE, 2008, pp. 363–372.

[34] Z. Zhang, G. Ely, S. Aeron, N. Hao, and M. Kilmer, "Novel methods for multilinear data completion and de-noising based on tensor-svd," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 3842–3849.

[35] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[36] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, "Tensor ring decomposition," *arXiv preprint arXiv:1606.05535*, 2016.

[37] Z. Chen, Z. Xu, and D. Wang, "Deep transfer tensor decomposition with orthogonal constraint for recommender systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4010–4018.

[38] B. Hui, D. Yan, H. Chen, and W.-S. Ku, "Time-sensitive poi recommendation by tensor completion with side information," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 205–217.

[39] H. Lamba, V. Nagarajan, K. Shin, and N. Shajarisales, "Incorporating side information in tensor completion," in *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016, pp. 65–66.

[40] K. Xie, Y. Chen, X. Wang, G. Xie, J. Cao, J. Wen, G. Yang, and J. Sun, "Accurate and fast recovery of network monitoring data with gpu-accelerated tensor completion," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1601–1614, 2020.

[41] B. Yang, A. S. Zamzam, and N. D. Sidiropoulos, "Large scale tensor factorization via parallel sketches," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 365–378, 2020.

[42] J.-G. Jang and U. Kang, "Dpar2: Fast and scalable parafac2 decomposition for irregular dense tensors," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2454–2467.

[43] K. Yang, Y. Gao, Y. Shen, B. Zheng, and L. Chen, "Dismastd: An efficient distributed multi-aspect streaming tensor decomposition," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1080–1091.

[44] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu, "Multi-aspect streaming tensor completion," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 435–443.

[45] D. Lee and K. Shin, "Robust factorization of real-world tensor streams with patterns, missing values, and outliers," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 840–851.

[46] M. Najafi, L. He, and S. Y. Philip, "Outlier-robust multi-aspect streaming tensor completion and factorization." in *IJCAI*, 2019, pp. 3187–3194.

[47] H. Liu, Y. Li, M. Tsang, and Y. Liu, "CoSTCo: A neural tensor completion model for sparse tensors," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 324–334.

[48] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," *Advances in neural information processing systems*, vol. 28, 2015.

[49] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9329–9338.

[50] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3891–3900.

[51] Y. Pan, J. Xu, M. Wang, J. Ye, F. Wang, K. Bai, and Z. Xu, "Compressing recurrent neural networks with tensor ring for action recognition," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4683–4690.

[52] J. C. S. de Souza, T. M. L. Assis, and B. C. Pal, "Data compression in smart distribution systems via singular value decomposition," *IEEE transactions on smart grid*, vol. 8, no. 1, pp. 275–284, 2015.

[53] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[54] ——, "On the best rank-1 and rank-$(r_1, r_2, ..., r_n)$ approximation of higher-order tensors," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.

[55] T. Kim, J. Lee, and Y. Choe, "Bayesian optimization-based global optimal rank selection for compression of convolutional neural networks," *IEEE Access*, vol. 8, pp. 17 605–17 618, 2020.

[56] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[57] T. Morris, "Industrial control system (ICS) cyber attack datasets," https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets, Apr. 2014, accessed 16 July 2023.

[58] Z. He, P. Chen, X. Li, Y. Wang, G. Yu, C. Chen, X. Li, and Z. Zheng, "A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[59] R. Zhu, B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network latency estimation for personal devices: A matrix completion approach," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 724–737, 2016.

[60] J. Ledlie, P. Gardner, and M. I. Seltzer, "Network coordinates in the wild." in *NSDI*, vol. 7, 2007, pp. 299–311.

[61] I. Pavlov, "History of 7-zip changes," https://7-zip.org/history.txt, May 2023, accessed 6 June 2023.

[62] Z. S. Jyrki Alakuijala, "Releases - google/brotli," https://github.com/google/brotli/releases, Aug. 2020, accessed 6 June 2023.

[63] Y. Collet, "Releases - facebook/zstd," https://github.com/facebook/zstd/releases, Apr. 2023, accessed 6 June 2023.

[64] J. Seward, "bzip2 and libbzip2, version 1.0.8," https://sourceware.org/bzip2/manual/manual.html, Jul. 2019, accessed 30 June 2023.

[65] J. Meyering, "gzip-1.12 released," https://lists.gnu.org/archive/html/info-gnu/2022-04/msg00003.html, Apr. 2022, accessed 30 June 2023.

[66] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao *et al.*, "Sz3: A modular framework for composing prediction-based error-bounded lossy compressors," *IEEE Transactions on Big Data*, vol. 9, no. 2, pp. 485–498, 2022.

[67] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor learning in python," *Journal of Machine Learning Research*, vol. 20, no. 26, pp. 1–6, 2019.