# Online CP Decomposition for Sparse Tensors

Shuo Zhou, Sarah M. Erfani and James Bailey
School of Computing and Information Systems, The University of Melbourne, Australia
{zhous@student., sarah.erfani@, baileyj@}unimelb.edu.au

***Abstract*—Tensor decomposition techniques such as CANDECOMP/PARAFAC (CP) decomposition have achieved great success across a range of scientific fields. They have been traditionally applied to dense, static data. However, today's datasets are often highly sparse and dynamically changing over time. Traditional decomposition methods such as Alternating Least Squares (ALS) cannot be easily applied to sparse tensors, due to poor efficiency. Furthermore, existing online tensor decomposition methods mostly target dense tensors, and thus also encounter significant scalability issues for sparse data. To address this gap, we propose a new incremental algorithm for tracking the CP decompositions of online sparse tensors on-the-fly. Experiments on nine real-world datasets show that our algorithm is able to produce quality decompositions of comparable quality to the most accurate algorithm, ALS, whilst at the same time achieving speed improvements of up to 250 times and 100 times less memory. Compared to other state-of-the-art online decomposition approaches, our method also demonstrates significant improvements in terms of efficiency and scalability.**

## I. Introduction

Multi-dimensional datasets are common i*2220D100494254509700*2220D100494254509700n a wide range of applications such as chemometrics [1], signal processing [2], machine learning [3] and data mining [4]. A natural choice for representing such data is a *Tensor*, which is a multi-way array that can preserve the complex relationships among different dimensions. *Tensor Decomposition* (TD) is a fundamental technique for analyzing tensors, and it has been extensively studied and widely applied for varying tasks, including subspace learning in computer vision [5], community detection in time-evolving networks [6], and feature extraction for spatial-temporal time series [7]. However, existing TD techniques are usually designed for dense and static tensors, which makes them less suitable when the data is highly sparse and dynamically changing over time.

Many real-world tensors are very large and have high sparsity. Thus, compared to their overall size, the number of non-zero entries is small. As a consequence, it is difficult to apply existing TD techniques, since these have often been designed for dense tensors, and thus have poor efficiency and scalability when applied to sparse scenarios. Moreover, sparse tensors are often not static. Instead, they are often changing over time, as large volumes of new data is generated and added to the the existing tensor. One typical type of dynamic update is appending new data along a specific dimension such as time, with the other dimensions remaining unchanged. An example of such a scenario is a time-evolving social network, as shown in Figure 1. The network can be represented by a

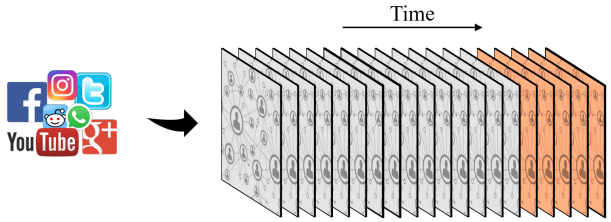



Fig. 1. An example of online sparse tensors

$user \times user \times time$ tensor and each slice at its time dimension is a collection of user interactions such as tagging a friend in a photo or visiting a friend's homepage. Overall, this tensor is highly sparse by its nature, and rapidly growing as new data (slices) are added. We refer to such tensors as *online sparse tensors*.

The research problem we address in this paper is how to efficiently decompose online sparse tensors. We are particularly interested in CANDECOMP/PARAFAC (CP) decomposition, since it is one of the most popular TD techniques having numerous applications across different domains [8]. Specifically, given the existing CP decomposition of a sparse tensor and a batch of new sparse data slices, which get added to the tensor's time mode, *we would like to efficiently obtain the new CP decomposition of the newly formed tensor without computing it from scratch*.

To the best of our knowledge, this is an unresolved question and existing approaches are not suitable for decomposing online sparse tensors. Alternating Least Squares (ALS) has been widely considered as the workhorse for CP decomposition [8] due to its simplicity and ease of implementation. However, it is infeasible for decomposing an online large scale sparse tensor, due to poor runtime efficiency. In addition, batch methods like ALS require availability of the full data, while the size of an online tensor that grows over time is potentially unbounded, so we may not be able to fit it into the memory. Even though distributed and parallel algorithms [9], [10], [11] may be used to accelerate ALS, the number of non-zeros in the new data can be too small, so it is wasteful to employ computational resources to decompose the full tensor from scratch. Lastly, there are existing approaches that have been designed for incremental, online decompositions [12], [13], but these are designed for online *dense* tensors and their complexities are usually linear to the size of new data. This means that they require the same amount of time and space for processing both dense and sparse online tensors, and no optimization

is employed with respect to sparsity. This significantly limits their applicability to online sparse tensors which can be very large, but contain few non-zero elements.

Overall, due to their poor efficiency and scalability, existing methods are not well suited for decomposing online sparse tensors. To address this gap, we propose a new algorithm, OnlineSCP. The contributions of our work are as follows:

- We propose a new algorithm having linear complexity to the number of non-zeros in the new data, for tracking the CP decomposition of an online sparse tensor.
- Via experiments on nine real-world datasets, our method demonstrates high decomposition quality, as well as significant efficiency improvements in both time and memory usagae. Empirical analysis on real and synthetic datasets shows that our method is considerably more scalable than state-of-the-art techniques.

## II. Related Work

Tensors are a generalization of matrices and CP decomposition is a powerful tool for data simplification, feature extraction and knowledge discovery on tensors. While much tensor-based work can be found in the literature, few studies have been conducted on decomposing online sparse tensors. In this section we review relevant literature in this area.

In order to decompose a large-scale sparse tensor efficiently, Bader and Kolda [14] proposed an algorithm based on ALS, tailoring it to sparse tensors, with support from special data structures and customized tensor-related operations. Following the same idea, Smith *et al.* [15] proposed a new hierarchical, fiber-centric data structure called a Compressed Sparse Fiber (CSF). Compared to the coordinate (COO) format used in [14], CSF is more memory-efficient and much easier to parallelize. Li *et al.* [16] also applied CSF as the storage format for sparse tensors, while a variant of CSF (vCSF) was used for recording intermediate results, which leads to more memory-savings and speedup. In addition, with advances in distributed computing, techniques like MapReduce have also been used to further speed up ALS for large scale sparse tensors [9], [10], [11]. Inspired by the success of parallel tensor decomposition algorithms, GPUs have been applied for accelerating sparse tensor computations [17].

In terms of online tensor decomposition, the problem was first proposed by Sun *et al.* [18], [19], wherein they refer to this problem as *Incremental Tensor Analysis* (ITA). Three algorithms have been proposed in their work: dynamic tensor analysis (DTA), stream tensor analysis (STA) and window-based tensor analysis (WTA), using techniques such as incremental update, approximation and sliding windows. However, these techniques are online versions of Tucker decomposition. Although CP decomposition can be viewed as a special case of Tucker with super-diagonal core tensor, none of the above methods provides a way to enforce this constraint. As a result, these algorithms are not suitable for tracking the CP decompositions of online sparse tensors.

Limited research can be found for online CP decompositions. An early exploration was done by Nion and Sidiropoulos [12], introducing two adaptive algorithms that specifically focus on CP decomposition: Simultaneous Diagonalization Tracking (SDT), which incrementally tracks the SVD of the unfolded tensor; and Recursive Least Squares Tracking (RLST), which recursively updates the decomposition factors by minimizing the mean squared error. However, the limitation of this work is that it can only be applied to third-order tensors. Zhou *et al.* [13] proposed a general approach that incrementally tracks the CP decompositions of online tensors with arbitrary dimensions. However, both [12] and [13] are specifically designed for dense online tensors, which means that their efficiency quickly drops to an unacceptable level and may potentially run out of memory for sparse online tensors having large non-temporal modes. A more recent technique proposed by Gujral *et al.* [20] can operate with both dense and sparse online tensors via sub-sampling, though multiple repetitions are required for a stable result.

Lastly, it is worth mentioning studies that address related, but somewhat different questions to our research problem. Zhou *et al.* [21] proposed a dynamic learning schema for tracking CP decomposition of sparse tensors that are incrementally changing at the element (not slice) level. Shin *et al.* [22] also studied dynamic sparse tensors with element-wise updating and proposed an efficient algorithm for dense-subtensor detection. Song *et al.* [23] addressed the tensor completion problem based on CP decomposition for dynamic tensors that are growing across all modes (rather than just a single mode, which will be our focus).

## III. Preliminaries

This section introduces tensor notations (summarized in Table I) and some basic operations that we are using throughout this work. In addition, a brief introduction to CP decomposition is provided.

### A. Notation and Tensor Algebra

A tensor is a multi-way array, which is a generalization of vector and matrix. In this paper, we denote vectors by boldface lowercase letters, e.g., $\mathbf{a}$, matrices by boldface uppercase letters, e.g., $\mathbf{A}$, and tensors by boldface Euler script letters, e.g., $\boldsymbol{\mathcal{X}}$. The *order* of a tensor, also known as the number of *ways* or *modes*, is its number of dimensions. For example, vectors and matrices are $1^{st}$-order and $2^{nd}$-order tensors, respectively; and a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an $N^{th}$-order one consisting of real numbers and the cardinality of its $i$-th order, $i \in [1, N]$ is $I_i$. We refer to tensors with more than 3 modes as *higher*-order ones.

Let $\mathbf{A}^\top$, $\mathbf{A}^{-1}$, $\mathbf{A}^\dagger$ and $\left\|\mathbf{A}\right\|$ denote the transpose, inverse, Moore-Penrose pseudoinverse and Frobenius norm of $\mathbf{A}$, respectively. Additionally, for indexing matrix $\mathbf{A}$, we denote its $(i, j)$-th element by $a_{ij}$, $i$-th row vector by $\mathbf{a}_{i\cdot}$, and $j$-th column vector by $\mathbf{a}_{\cdot j}$.

TABLE I
LIST OF SYMBOLS

| | |
|---|---|
| $a$, $\mathbf{a}$, $\mathbf{A}$, $\mathcal{X}$ | scalar, vector, matrix, tensor |
| $\mathbf{a}_{i\cdot}$, $\mathbf{a}_{\cdot j}$ | the $i$-th row and $j$-th column vectors of $\mathbf{A}$ |
| $a_{ij}$ | the $(i,j)$-th element of $\mathbf{A}$ |
| $\mathbf{A}^\top$, $\mathbf{A}^{-1}$, $\mathbf{A}^\dagger$ | transpose, inverse and pseudoinverse of $\mathbf{A}$ |
| $\|\|\mathbf{A}\|\|$ | Frobenius norm of $\mathbf{A}$ |
| $\mathbf{A}^{(n)}$ | the $n$-the matrix of a sequence of matrices |
| $\odot$ | Khatri-Rao product |
| $\circledast$ | Hadamard product |
| $\oslash$ | element-wise division |
| $\bigodot_{i=1}^{N}\mathbf{A}^{(i)}$ | $\mathbf{A}^{(N)}\odot\cdots\odot\mathbf{A}^{(i)}\odot\cdots\odot\mathbf{A}^{(1)}$ |
| $\circledast_{i=1}^{N}\mathbf{A}^{(i)}$ | $\mathbf{A}^{(N)}\circledast\cdots\circledast\mathbf{A}^{(i)}\circledast\cdots\circledast\mathbf{A}^{(1)}$ |
| $\mathbf{X}_{(n)}$ | mode-$n$ unfolding of $\mathcal{X}$ |
| $[\![\bullet]\!]$ | CP decomposition operator |
| $R$ | number of components |
| $N$ | order of tensor |
| $I_n$ | length of the $n$-th mode of $\mathcal{X}$ |
| $S$, $J$ | $\prod_{n=1}^{N-1} I_n$, $\sum_{n=1}^{N-1} I_n$ |
| $t$, $\Delta t$ | length of time of existing and incoming tensors |
| $\|\Omega^+\|$, $\|\Delta\Omega^+\|$ | number of non-zeros in full and incoming tensors |

The *Khatri-Rao* product [8] of two matrices $\mathbf{A}\in\mathbb{R}^{I\times K}$ and $\mathbf{B}\in\mathbb{R}^{J\times K}$ is denoted by $\mathbf{A}\odot\mathbf{B}$, the output of which is a $IJ\times K$ matrix defined as

$$\mathbf{A}\odot\mathbf{B}=\begin{bmatrix} a_{11}\mathbf{b}_{\cdot 1} & a_{12}\mathbf{b}_{\cdot 2} & \cdots & a_{1K}\mathbf{b}_{\cdot K}\\ a_{21}\mathbf{b}_{\cdot 1} & a_{22}\mathbf{b}_{\cdot 2} & \cdots & a_{2K}\mathbf{b}_{\cdot K}\\ \vdots & \vdots & \ddots & \vdots\\ a_{I1}\mathbf{b}_{\cdot 1} & a_{I2}\mathbf{b}_{\cdot 2} & \cdots & a_{IK}\mathbf{b}_{\cdot K}\end{bmatrix}.$$

The *Hadamard* product of two equal size matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted by $\mathbf{A}\circledast\mathbf{B}$, which is their element-wise product. Similarly, we denote their element-wise division by $\mathbf{A}\oslash\mathbf{B}$.

Furthermore, let $\mathbf{A}^{(1)},\mathbf{A}^{(2)},\ldots,\mathbf{A}^{(N)}$ represent a list of $N$ matrices. The Khatri-Rao product series $\mathbf{A}^{(N)}\odot\cdots\odot\mathbf{A}^{(i)}\odot\cdots\odot\mathbf{A}^{(1)}$ is denoted by $\bigodot_{i=1}^{N}\mathbf{A}^{(i)}$ and the Hadamard product sequence $\mathbf{A}^{(N)}\circledast\cdots\circledast\mathbf{A}^{(i)}\circledast\cdots\circledast\mathbf{A}^{(1)}$ is denoted by $\circledast_{i=1}^{N}\mathbf{A}^{(i)}$.

*Tensor unfolding*, or *matricization*, is a procedure that transforms a tensor into a matrix [8]. Generally speaking, given an $N^{th}$-order tensor $\mathcal{X}\in\mathbb{R}^{I_1\times\cdots\times I_N}$, its mode-$n$ unfolding $\mathbf{X}_{(n)}\in\mathbb{R}^{I_n\times\prod_{i\neq n}^{N} I_i}$ can be obtained by permuting the dimensions of $\mathcal{X}$ as $[I_n,I_1,\ldots,I_{n-1},I_{n+1},\ldots,I_N]$ and then reshaping the permuted tensor into a matrix of size $I_n\times\prod_{i\neq n}^{N} I_i$ [13].

### B. CANDECOMP/PARAFAC Decomposition

CP decomposition is one of the most popular techniques for analyzing tensors. Basically, given an $N^{th}$-order tensor $\mathcal{X}\in\mathbb{R}^{I_1\times\cdots\times I_N}$, its CP decomposition is denoted by $[\![\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}]\!]$, which is consisted of $N$ *loading* matrices. Each loading matrix, $\mathbf{A}^{(n)}, n\in[1,N]$, is of size $I_n\times R$, where $R$ is usually a very small number compared to $I_n$. $R$ is called the rank of $\mathcal{X}$, indicating the number of latent factors used to approximate this tensor. By using these loading matrices, the mode-$n$ unfolding of $\mathcal{X}$ can be approximated as

$$\begin{aligned}\mathbf{X}_{(n)}&\approx\mathbf{A}^{(n)}(\mathbf{A}^{(N)}\odot\cdots\mathbf{A}^{(n+1)}\odot\mathbf{A}^{(n-1)}\cdots\odot\mathbf{A}^{(1)})^\top\\&=\mathbf{A}^{(n)}(\textstyle\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})^\top.\end{aligned}\tag{1}$$

It means that to find the CP decomposition of $\mathcal{X}$, we need to solve an optimization problem that minimizes the error between $\mathcal{X}$ and $[\![\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}]\!]$. One commonly used loss function is

$$\mathcal{L}=\frac{1}{2}\left\|\mathbf{X}_{(n)}-\mathbf{A}^{(n)}(\textstyle\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})^\top\right\|^2.$$

However, it is very difficult to minimize $\mathcal{L}$ jointly over $\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}$, since $\mathcal{L}$ is not convex w.r.t. $\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}$. To address this issue, a widely applied approach is ALS. The main idea is to divide the above optimization problem into $N$ sub-problems and the $n$-th one, $n\in[1,N]$ fixes all variables but $\mathbf{A}^{(n)}$, and then minimizes the convex objective $\mathcal{L}$ w.r.t. $\mathbf{A}^{(n)}$, that is

$$\begin{aligned}\mathbf{A}^{(n)}&\leftarrow\arg\min_{\mathbf{A}^{(n)}}\frac{1}{2}\left\|\mathbf{X}_{(n)}-\mathbf{A}^{(n)}(\textstyle\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})^\top\right\|^2\\&=\mathbf{X}_{(n)}((\textstyle\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})^\top)^\dagger\\&=\frac{\mathbf{X}_{(n)}(\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})}{\circledast_{i\neq 1}^{N}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})}.\end{aligned}\tag{2}$$

ALS has gained great popularity due to its simplicity and ease of implementation, but its efficiency has been criticized. The major computational bottleneck is the calculation of $\mathbf{X}_{(n)}(\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})$, which is often referred to as *matricized tensor times Khatri-Rao products* (MTTKRP) [15]. Specifically, the first term, $\mathbf{X}_{(n)}$, is a $I_n\times\prod_{i\neq n}^{N} I_i$ flat matrix and the Khatri-Rao products produce a super tall matrix with size $\prod_{i\neq n}^{N} I_i\times R$, resulting in an overall time complexity as $\mathcal{O}(R\prod_{i=1}^{N} I_i)$, which is linear to the size of $\mathcal{X}$. In addition, in terms of space usage, ALS is challenged by the so-called intermediate data explosion problem [11], since it requires allocating a huge amount of temporal memory space to store the matricized tensor and the Khatri-Rao products.

Overall, both the time and space complexities of naive MTTKRP calculation are linear to the size of the input tensor. Such high cost makes typical ALS impractical for large-scale tensors. More importantly, for a sparse tensor, compared to its size, the amount of valuable information, i.e., the number of non-zeros, is much reduced. As a result, it is not desirable or feasible to explicitly calculate MTTKRP for sparse tensors.

To address the efficiency issue and improve the performance of MTTKRP for sparse tensors, a number of methods have been proposed by taking sparsity into consideration and using techniques such as distributed and parallel computing [10], [11], [15], [16], and GPU acceleration [17]. The most popular method that has been widely recognized as a gold standard is [14] (summarized in Algorithm 1), which dramatically reduces the complexity of MTTKRP to be linear in the number of non-zeros in a sparse tensor.

**Algorithm 1:** MTTKRP via Sparse Tensor-Vector Products

**Input:** non-zeros $\mathbf{z}$, indices of non-zeros $\mathbf{j}^{(1)}, \ldots, \mathbf{j}^{(N)}$, loading matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$, mode $n$
**Output:** mode-$n$ MTTKRP $\mathbf{P}^{(n)}$

**for** $r \leftarrow 1$ **to** $R$ **do**
  $\mathbf{t} \leftarrow \mathbf{z}$
  **for** $i \leftarrow 1$ **to** $N$ **do**
    $\mathbf{t} \leftarrow \mathbf{t} \circledast \mathbf{a}^{(i)}_{\mathbf{j}^{(i)}r}$
  **end**
  $\mathbf{p}^{(n)}_{:r} \leftarrow \text{ACCUMARRAY}(\mathbf{j}^{(n)}, \mathbf{t})$
**end**

## IV. Our Approach

This section introduces our approach, OnlineSCP, an algorithm to track the CP decomposition of a sparse online tensor. We first discuss the main idea of our algorithm, followed by key improvements that have been applied and a brief complexity analysis and a comparison to state-of-the-art methods.

Formally speaking, the research question we solve is defined as follows:

**Problem Definition.** Given *(i)* an existing CP decomposition $[\![\widetilde{\mathbf{A}}^{(1)}, \ldots, \widetilde{\mathbf{A}}^{(N)}]\!]$ of $R$ components that approximates a sparse tensor $\widetilde{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times t}$ at time $t$, *(ii)* a new incoming sparse tensor $\Delta\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times \Delta t}$ that is appended to $\mathcal{X}$ at its last mode and forms a new tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times (t+\Delta t)}$, where $\Delta t \ll t$. The objective is to find the CP decomposition $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ of $\mathcal{X}$.

### A. The Principle of OnlineSCP

To address the online SCP (sparse CP decomposition) problem, our method follows the same alternating update schema as ALS, such that only one loading matrix is updated at one time by fixing all others. Specifically, we update the time mode first, then move on to the rest of the non-temporal modes as

$$\mathbf{A}^{(N)} \rightarrow \mathbf{A}^{(1)} \rightarrow \mathbf{A}^{(2)} \cdots \rightarrow \mathbf{A}^{(N-1)}.$$

In order to take advantage of the fact that the tensor is only growing at its time mode, similar to existing online works [12], [13], the main assumption of our method is that the new incoming data, $\Delta\mathcal{X}$, will not have significant impact on the existing model, but only contribute to the update of its corresponding local features. By this, we mean when new data arrives, the first $t$ rows in the loading matrix of the temporal mode, $\mathbf{A}^{(N)}$, will remain unchanged; while the loading matrices for non-temporal modes, $\mathbf{A}^{(1)}, \ldots \mathbf{A}^{(N-1)}$, will receive a minor update based on their existing values.

At a high level, as presented in Algorithms 2 and 3, our algorithm consists of two procedures. Before the start of the learning phase, it initializes two small *helper* matrices for storing historical information by using the initial tensor and its decomposition (details in Algorithm 2). After that, in the online learning phase, the new incoming tensors can be efficiently processed by an incremental update schema (details in Algorithm 3).

Compared to ALS, the major speedup gained by our approach comes from several aspects: (i) only a small fraction of new values are added to the loading matrix of the temporal mode while the remainder is kept unchanged (as discussed in §IV-B); (ii) for non-temporal modes, we break down the costly MTTKRP calculation into historical and new data parts, the historical one is avoided by using helper matrices and the new data one is efficiently obtained by Algorithm 1 (as discussed in §IV-C); (iii) the helper matrices can also be incrementally updated at a small cost (as discussed in §IV-D).

**Algorithm 2:** Initialization Procedure of OnlineSCP

**Input:** initial data $\mathcal{X}_0$, its decomposition $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$
**Output:** helper matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$

$\mathbf{U}^{(N)} \leftarrow {\mathbf{A}^{(N)}}^\top \mathbf{A}^{(N)}$
$\mathbf{Q} \leftarrow \mathbf{U}^{(N)}$
**for** $n \leftarrow 1$ **to** $N-1$ **do**
  $\mathbf{Q} \leftarrow \mathbf{Q} \circledast ({\mathbf{A}^{(n)}}^\top \mathbf{A}^{(n)})$
**end**

**Algorithm 3:** Update Procedure of OnlineSCP

**Input:** loading matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N-1)}$, helper matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$, new data $\Delta\mathcal{X}$
**Output:** updated loading matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N-1)}$, updated helper matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$, coefficient of new data $\Delta\mathbf{A}^{(N)}$

$\widetilde{\mathbf{Q}} \leftarrow \mathbf{Q}$
// process temporal mode
$\mathbf{Q}^{(N)} \leftarrow \mathbf{Q} \oslash \mathbf{U}^{(N)}$
$\Delta\mathbf{P}^{(N)} \leftarrow$ MTTKRP by Algorithm 1
$\Delta\mathbf{A}^{(N)} \leftarrow \Delta\mathbf{P}^{(N)}(\mathbf{Q}^{(N)})^{-1}$
// update helper matrices
$\mathbf{U}^{(N)} \leftarrow \mathbf{U}^{(N)} + ({\Delta\mathbf{A}^{(N)}}^\top \Delta\mathbf{A}^{(N)})$
$\mathbf{Q} \leftarrow \mathbf{Q}^{(N)} \circledast \mathbf{U}^{(N)}$
// process non-temporal mode
**for** $n \leftarrow 1$ **to** $N-1$ **do**
  $\widetilde{\mathbf{A}}^{(n)} \leftarrow \mathbf{A}^{(n)}$
  $\mathbf{U}^{(n)} \leftarrow {\mathbf{A}^{(n)}}^\top \mathbf{A}^{(n)}$
  $\mathbf{Q}^{(n)} \leftarrow \mathbf{Q} \oslash \mathbf{U}^{(n)}$, $\widetilde{\mathbf{Q}}^{(n)} \leftarrow \widetilde{\mathbf{Q}} \oslash \mathbf{U}^{(n)}$
  $\Delta\mathbf{P}^{(n)} \leftarrow$ MTTKRP by Algorithm 1
  $\mathbf{A}^{(n)} \leftarrow (\mathbf{A}^{(n)}\widetilde{\mathbf{Q}}^{(n)} + \Delta\mathbf{P}^{(n)})(\mathbf{Q}^{(n)})^{-1}$
  // update helper matrices
  $\mathbf{Q} \leftarrow \mathbf{Q}^{(n)} \circledast ({\mathbf{A}^{(n)}}^\top \mathbf{A}^{(n)})$
  $\widetilde{\mathbf{Q}} \leftarrow \widetilde{\mathbf{Q}}^{(n)} \circledast ({\widetilde{\mathbf{A}}^{(n)}}^\top \mathbf{A}^{(n)})$
**end**

### B. Updating the Temporal Mode

Specifically, the temporal loading matrix, $\mathbf{A}^{(N)}$, is a $(t+\Delta t)\times R$ matrix as

$$\mathbf{A}^{(N)} \leftarrow \begin{bmatrix} \widetilde{\mathbf{A}}^{(N)} \\ \Delta\mathbf{A}^{(N)} \end{bmatrix},$$

where its first $t$ rows is $\widetilde{\mathbf{A}}^{(N)} \in \mathbb{R}^{t\times R}$ and $\Delta\mathbf{A}^{(N)}$ is a small matrix of size $\Delta t \times R$. $\Delta\mathbf{A}^{(N)}$ can be easily obtained by projecting $\Delta\mathcal{X}$ to the last mode via other non-temporal loading matrices as

$$\begin{aligned}\Delta\mathbf{A}^{(N)} &= \Delta\mathbf{X}_{(N)}((\odot_{i=1}^{N-1}\mathbf{A}^{(i)})^\top)^\dagger \\ &= \frac{\Delta\mathbf{X}_{(N)}(\odot_{i=1}^{N-1}\mathbf{A}^{(i)})}{\circledast_{i=1}^{N-1}({\mathbf{A}^{(i)}}^\top\mathbf{A}^{(i)})}.\end{aligned}$$

Let $\mathbf{U}^{(N)} = {\mathbf{A}^{(N)}}^\top\mathbf{A}^{(N)}$, $\mathbf{Q} = \circledast_{i=1}^{N}({\mathbf{A}^{(i)}}^\top\mathbf{A}^{(i)})$, and $\Delta\mathbf{P}^{(N)} = \Delta\mathbf{X}_{(N)}(\odot_{i=1}^{N-1}\mathbf{A}^{(i)})$, recall that there is no loading matrices has been updated so far, we can rewrite the above equation with helper matrices $\mathbf{U}^{(N)}$ and $\mathbf{Q}$ as

$$\Delta\mathbf{A}^{(N)} \leftarrow \frac{\Delta\mathbf{P}^{(N)}}{\mathbf{Q}\oslash\mathbf{U}^{(N)}}, \tag{3}$$

where the MTTKRP, $\Delta\mathbf{P}^{(N)}$, can be efficiently calculated by Algorithm 1 at linear complexity to the number of non-zeros in $\Delta\mathcal{X}$, which we refer to as $|\Delta\Omega^+|$.

### C. Updating Non-Temporal Modes

Without loss of generality, here we show how to derive the incremental update rule for loading matrix at the first mode, $\mathbf{A}^{(1)}$. By using helper matrix $\mathbf{Q}$ defined as above, the update given by the typical ALS is

$$\mathbf{A}^{(1)} \leftarrow \frac{\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})}{\circledast_{i=2}^{N}({\mathbf{A}^{(i)}}^\top\mathbf{A}^{(i)})} = \frac{\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}. \tag{4}$$

Even though the MTTKRP operation, $\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})$, can be accelerated by Algorithm 1 given that $\mathcal{X}$ is sparse, such cost is still not accepatable since $\mathcal{X}$ is potentially a large-scale tensor and the number of non-zeros in $\mathcal{X}$ will keep growing with the increase of time. As a matter of fact, by noticing that $\mathcal{X}$ is an online tensor that new data is only appended at its last mode, there are some patterns can be found in its mode-$n$ unfolding and the corresponding Khatri-Rao product. As a result, we make use of such patterns to derive an efficient update rule as follows.

Recall that $\mathbf{A}^{(N)}$ has been updated as $[\widetilde{\mathbf{A}}^{(N)}; \Delta\mathbf{A}^{(N)}]$ and let $\mathbf{B}^{(1)} = \odot_{i=2}^{N-1}\mathbf{A}^{(i)}$, Eq. (4) can be rewritten as

$$\begin{aligned}\mathbf{A}^{(1)} &\leftarrow \frac{\left[\widetilde{\mathbf{X}}_{(1)}, \Delta\mathbf{X}_{(1)}\right]\left(\begin{bmatrix}\widetilde{\mathbf{A}}^{(N)}\\ \Delta\mathbf{A}^{(N)}\end{bmatrix}\odot\mathbf{B}^{(1)}\right)}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} \\ &= \frac{\widetilde{\mathbf{X}}_{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\mathbf{B}^{(1)}) + \Delta\mathbf{X}_{(1)}(\Delta\mathbf{A}^{(N)}\odot\mathbf{B}^{(1)})}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} \\ &= \frac{\widetilde{\mathbf{P}}^{(1)} + \Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}.\end{aligned}$$

In this way, the MTTKRP is divided into two parts: the one that is related to the historical data, $\widetilde{\mathbf{P}}^{(1)}$; and the other that is only depending on the new incoming data, $\Delta\mathbf{P}^{(1)}$. In fact, since $[\![\widetilde{\mathbf{A}}^{(1)},\ldots,\widetilde{\mathbf{A}}^{(N)}]\!]$ is the existing decomposition of $\widetilde{\mathcal{X}}$, we have

$$\widetilde{\mathbf{X}}_{(1)} \approx \widetilde{\mathbf{A}}^{(1)}(\odot_{i=2}^{N}\widetilde{\mathbf{A}}^{(i)})^\top = \widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\widetilde{\mathbf{B}}^{(1)})^\top, \tag{5}$$

where $\widetilde{\mathbf{B}}^{(1)} = \odot_{i=2}^{N-1}\widetilde{\mathbf{A}}^{(i)}$. Based on this, $\widetilde{\mathbf{X}}_{(1)}$ in $\widetilde{\mathbf{P}}^{(1)}$ can be replaced by Eq. (5) and the final update rule for $\mathbf{A}^{(1)}$ is

$$\begin{aligned}\mathbf{A}^{(1)} &\leftarrow \frac{\widetilde{\mathbf{X}}_{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\mathbf{B}^{(1)}) + \Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} \\ &\approx \frac{\widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\widetilde{\mathbf{B}}^{(1)})^\top(\widetilde{\mathbf{A}}^{(N)}\odot\mathbf{B}^{(1)}) + \Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} \\ &= \frac{\widetilde{\mathbf{A}}^{(1)}({\widetilde{\mathbf{A}}^{(N)}}^\top\widetilde{\mathbf{A}}^{(N)})\circledast({\widetilde{\mathbf{B}}^{(1)}}^\top\mathbf{B}^{(1)}) + \Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} \\ &= \widetilde{\mathbf{A}}^{(1)}\frac{\widetilde{\mathbf{Q}}\oslash\mathbf{U}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} + \frac{\Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}},\end{aligned} \tag{6}$$

where $\widetilde{\mathbf{Q}} = ({\widetilde{\mathbf{A}}^{(N)}}^\top\widetilde{\mathbf{A}}^{(N)})\circledast(\circledast_{i=1}^{N-1}({\widetilde{\mathbf{A}}^{(i)}}^\top\mathbf{A}^{(i)}))$.

Overall, the above update rule significantly reduces the computational cost by limiting the expensive MTTKRP operation to the new data only. We can interpret such update schema as follows: *the new loading matrix is a weighted combination of existing loading and a hyper loading learned from the new data. The weight between them is determined by the ratio of information contains in the historical data w.r.t. the full data*.

### D. Incremental Update of $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$

As mentioned before, by definition we have

$$\begin{aligned}\mathbf{Q} &= ({\mathbf{A}^{(1)}}^\top\mathbf{A}^{(1)})\cdots\circledast({\mathbf{A}^{(n)}}^\top\mathbf{A}^{(n)})\circledast\cdots({\mathbf{A}^{(N)}}^\top\mathbf{A}^{(N)}), \\ \widetilde{\mathbf{Q}} &= ({\widetilde{\mathbf{A}}^{(1)}}^\top\mathbf{A}^{(1)})\cdots\circledast({\widetilde{\mathbf{A}}^{(n)}}^\top\mathbf{A}^{(n)})\circledast\cdots({\widetilde{\mathbf{A}}^{(N)}}^\top\widetilde{\mathbf{A}}^{(N)}).\end{aligned}$$

Their values are gradually changing with the updating of loading matrices. However, the main difference between the $\mathbf{Q}$ values before and after updating $\mathbf{A}^{(n)}$ is just the ${\mathbf{A}^{(n)}}^\top\mathbf{A}^{(n)}$ term. As a result, we do not need to calculate the $\mathbf{Q}$ values from scratch for each update. Instead, we initialize both $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$ as $\circledast_{i=1}^{N}({\widetilde{\mathbf{A}}^{(i)}}^\top\widetilde{\mathbf{A}}^{(i)})$. After processing the time mode, $\widetilde{\mathbf{Q}}$ remains unchanged, while $\mathbf{Q}$ can be updated as

$$\mathbf{Q} \leftarrow \mathbf{Q}\oslash({\widetilde{\mathbf{A}}^{(N)}}^\top\widetilde{\mathbf{A}}^{(N)})\circledast({\mathbf{A}^{(N)}}^\top\mathbf{A}^{(N)}).$$

Similarly, the update after processing the $n$-th non-temporal mode is

$$\begin{aligned}\mathbf{Q} &\leftarrow \mathbf{Q}\oslash({\widetilde{\mathbf{A}}^{(N)}}^\top\widetilde{\mathbf{A}}^{(N)})\circledast({\mathbf{A}^{(N)}}^\top\mathbf{A}^{(N)}), \\ \widetilde{\mathbf{Q}} &\leftarrow \widetilde{\mathbf{Q}}\oslash({\widetilde{\mathbf{A}}^{(N)}}^\top\widetilde{\mathbf{A}}^{(N)})\circledast({\widetilde{\mathbf{A}}^{(N)}}^\top\mathbf{A}^{(N)}).\end{aligned}$$

After processing all modes, the final $\mathbf{Q}$ is stored and used to initialize $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$ for the next batch of new data.

Additionally, since $\mathcal{X}$ is an online tensor and the length of the time mode $t$ can be potentially quite large, we choose to avoid directly computing ${\widetilde{\mathbf{A}}^{(N)}}^\top\widetilde{\mathbf{A}}^{(N)}$ and ${\mathbf{A}^{(N)}}^\top\mathbf{A}^{(N)}$ by

TABLE II
COMPLEXITY COMPARISON BETWEEN ONLINESCP AND EXISTING METHODS.

| | Time | Memory | Source |
|---|---|---|---|
| OnlineSCP | $\mathcal{O}((J+\Delta t)R^2+\lvert\Delta\Omega^+\rvert NR)$ | $\mathcal{O}((J+\Delta t)R+\lvert\Delta\Omega^+\rvert)$ | |
| ALS | $\mathcal{O}((J+t+\Delta t)R^2+\lvert\Omega^+\rvert NR)$ | $\mathcal{O}((J+t+\Delta t)R+\lvert\Omega^+\rvert)$ | [24] |
| OnlineCP | $\mathcal{O}((J+\Delta t)R^2+NRS\Delta t)$ | $\mathcal{O}((J+\Delta t)R+SR\Delta t)$ | [13] |
| SDT | $\mathcal{O}((S+\Delta t)R^2)$ | $\mathcal{O}((J+S+t)R+SR\Delta t)$ | [12] |
| RLST | $\mathcal{O}(SR^2)$ | $\mathcal{O}((S+J+\Delta t)R+SR\Delta t)$ | [12] |

storing $\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)}$ into a small $R\times R$ matrix $\mathbf{U}^{(N)}$, and $\mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$ can be easily obtained as

$$\mathbf{U}^{(N)} \leftarrow \mathbf{U}^{(N)} + (\Delta\mathbf{A}^{(N)\top}\Delta\mathbf{A}^{(N)}).$$

### E. Complexity Analysis

Let $R$ be the decomposition rank, $N$ be the order of tensor, $t$ be the time length of existing data, $\Delta t$ be the time length of new data, $|\Omega^+|$ be the number of non-zeros in the full data, $|\Delta\Omega^+|$ be the number of non-zeros in the new tensor, $S=\prod_{i=1}^{N-1} I_i$, and $J=\sum_{i=1}^{N-1} I_i$, where $I_i$ is the length of the $i$-th mode.

To process a new chunk of data, the overall time complexity for our algorithm is dominated by $\mathcal{O}((J+\Delta t)R^2+|\Delta\Omega^+|NR)$, where $(J+\Delta t)R^2$ is corresponding to the $(\mathbf{A}^{(n)\top}\mathbf{A}^{(n)})$-like calculations (line 5, 9, 13, 14 in Algorithm 3) and the actual update for loading matrices (line 4, 12 in Algorithm 3); $|\Delta\Omega^+|NR$ is cost for MTTKRP operation (line 3, 11 in Algorithm 3).

In terms of space consumption, since the update of the time mode is independent to historical data, our method only needs to store the loading matrices for non-temporal modes and two $R\times R$ helper matrices in memory. In addition, extra $\mathcal{O}(|\Delta\Omega^+|)$ memory needs to be used for the MTTKRP calculation. As a result, the overall space cost is $\mathcal{O}((J+\Delta t)R+|\Delta\Omega^+|)$.

We summarize the complexity of OnlineSCP and make a comparison to state-of-the-art methods in Table II. It should be noted that the complexitie of SDT and RLST are measured based on the exponential window strategy [12], which considers all updated-to-date data in a single window assessing the importance of data slices at different timestamps using a forgetting factor $\lambda$. In addition, as they only work on $3^{rd}$-order tensors, when other methods are compared to them, $N$ should be set to 3. Another remark is that the complexity of ALS is based on one iteration only, in reality it usually takes a few iterations until convergence.

## V. EMPIRICAL ANALYSIS

In this section, we compare the performance of our proposed OnlineSCP algorithm with state-of-the-art techniques. We first examine their effectiveness and efficiency, in terms of both time and space, on nine real-world datasets. In addition, we make further investigation on the scalability of our method and baselines using several synthetic datasets.

### A. Experiment Specifications

*1) Datasets:* Nine real-world datasets of varying characteristics have been used in our experiments and their detail can be found in Table III. Facebook [25] and Youtube [28] are time-evolving social network data obtained from KONECT [31]. Both MovieLens [26] and LastFM [27] contain user rating data. The ratings in MovieLens are explicit, scaling from 1 to 5, whereas LastFM contains implicit ratings that are represented by the number of times a user listened a particular artist's songs for a given time interval. The rest of the datasets are publicly available on FROSTT [32]. NELL-1 and NELL-2 come from Never Ending Language Learning (NELL) project [30] and represent (noun, verb, noun) triples. NIPS [3] is a paper authorship network data and Enron [29] records email transactions within senior managers in Enron. Both of them are $4^{th}$-order tensors.

*2) Baselines:* In our experiments, four baselines have been selected for performance comparison.

(i) ALS [8]: an implementation of the ALS algorithm for sparse tensors provided by Tensor Toolbox [24]. Since it is a batch method, to make it work with online tensors, the CP decomposition of the last time step is used as the initialization for decomposing the current tensor.

(ii) OnlineCP [13]: an incremental ALS-like algorithm, which can track the decompositions of both $3^{rd}$-order and higher-order online tensors.

(iii) SDT [12]: an adaptive algorithm based on incrementally tracking the SVD of the unfolded tensor on the time mode.

(iv) RLST: another online approach proposed in [12]. Instead of tracking the SVD, recursive updates are performed to minimize the mean squared error on new data.

*3) Evaluation Metrics:* The performance of each method is demonstrated by its effectiveness and efficiency.

In terms of effectiveness, we measure the *fitness* of each algorithm as

$$fitness \triangleq \left(1-\frac{\left\|\hat{\mathcal{X}}-\mathcal{X}\right\|}{\left\|\mathcal{X}\right\|}\right),$$

where $\mathcal{X}$ is the original data, $\hat{\mathcal{X}}$ is the estimation and $\left\|\bullet\right\|$ denotes the Frobenius norm.

In order to validate the efficiency performance of an algorithm, both the *running time* and *memory usage* for processing one batch of new data are measured.

Since ALS is the most popular method for CP decomposition and in order to better interpret the results, we report

TABLE III
DETAIL OF REAL-WORLD DATASETS
(K: THOUSANDS, M: MILLIONS)

| Datasets | Description | Size | $nnz^*$ | Density | Batch Size | Source |
|---|---|---|---|---|---|---|
| Facebook-Links | user × user × day | 64K × 64K × 886 | 671K | $2 \times 10^{-7}$ | 4 | [25] |
| Facebook-Wall | wall owner × poster × day | 47K × 47K × 2K | 738K | $2 \times 10^{-7}$ | 8 | [25] |
| MovieLens | user × movie × time | 6K × 4K × 1K | 1M | $4 \times 10^{-5}$ | 5 | [26] |
| LastFM | user × artist × time | 1K × 1K × 168 | 3M | $2 \times 10^{-2}$ | 1 | [27] |
| NIPS | paper × author × year × word | 2K × 3K × 17 × 14K | 3M | $2 \times 10^{-6}$ | 70 | [3] |
| Youtube | user × user × day | 3M × 3M × 226 | 18M | $8 \times 10^{-9}$ | 1 | [28] |
| Enron | sender × receiver × word × day | 6K × 6K × 244K × 1K | 54M | $5 \times 10^{-9}$ | 6 | [29] |
| NELL-2 | entity × relation × entity | 12K × 9K × 30K | 77M | $2 \times 10^{-5}$ | 144 | [30] |
| NELL-1 | entity × relation × entity | 3M × 2M × 25M | 144M | $9 \times 10^{-13}$ | 127,476 | [30] |

* number of non-zeros

the *relative performance* of an online method to ALS over all three metrics as

$$relative\ fitness \triangleq \frac{fitness(baseline)}{fitness(ALS)},$$
$$speedup \triangleq \frac{time(ALS)}{time_(baseline)},$$
$$relative\ memory \triangleq \frac{memory(baseline)}{memory(ALS)}.$$

*4) Experimental Setup:* Given a dataset, the first $50\%$ data along the last mode is decomposed by ALS and this corresponding CP decomposition is used for initializing all methods. After that, the second half is further divided into 100 batches and each of them is sequentially appended to the existing data. All methods process one batch of data at a time. After learning a new batch, the updated decompositions of all methods are used to calculate the fitness, along with the time and memory consumption for this batch.

With respect to parameter settings, the decomposition rank $R$ is fixed to 5 for all datasets, due to the poor efficiency of baselines. For the ALS algorithm used in the initialization stage, the tolerance $\epsilon$ is set to $1 \times 10^{-8}$ and the maximum number of iterations is set to 100, to ensure quality starting points for all methods. In the online learning phase, $\epsilon$ of ALS is altered to $1 \times 10^{-4}$ and we enforce it to run one iteration at one timestamp, for a fair comparison to other methods. For SDT and RLST, exponential window strategy is used with a forgetting factor $\lambda = 1$, as all other methods treat all data as equally important. In addition, it should be noted that the original implementation of SDT and RLST can only handle one slice of data at a time, so we modified them by executing a for loop to process a batch. Lastly, no parameters need to be tuned for our method and OnlineCP.

The experiments are conducted on Spartan [33], a research platform with multiple computing nodes and each of them has 12 CPU cores and 251 GB RAM. Due to the fact that Spartan is a highly utilized system, we limit the computing resources for each algorithm-dataset pair as 4 CPU cores, 32 GB memory and 12-hour maximum running time. The whole experiment is replicated 5 times with Matlab and the final results are averaged over these repetitions.

### B. Experimental Results

The experimental results on real-world datasets can be found in Table IV, V and VI.

Among all online methods, our proposal, OnlineSCP, is the only method that is able to produce results for all datasets, given the limited computational resources as stated before. In contrast, SDT and RLST only manage to process MovieLens and LastFM, which are two relatively small datasets in terms of slice size (6K × 6K for MovieLens and 1K × 1K for LastFM). OnlineCP can process one more dataset compared to them, NELL-2, which has a slice size of 108M. Ideally OnlineCP should also work on NIPS dataset as one slice of NIPS data is slightly smaller than that of NELL-2. However, OnlineCP has a specifically designed procedure to speedup calculations on higher-order tensors by costing more memory, hence it fails on this $4^{th}$-order tensor.

*1) Results on Effectiveness:* Compared to existing online methods, our algorithm is able to produce the highest quality decompositions on-the-fly. The fitness of OnlineSCP is comparable to ALS being more than 90% as good in most cases. However, the performance of SDT and RLST is considerably worse on this criteria. Specifically, compared to ALS, the relative fitness of SDT is under one third for both MovieLens and LastFM, and RLST can only reach 17% relative fitness on LastFM. The relative fitness of OnlineCP is close to ours on MovieLens and LastFM, but 1% lower on NELL-2 dataset.

*2) Results on Time Efficiency:* Lack of optimization on sparsity is the main contributing factor to the poor time efficiency of existing online methods on the MovieLens dataset, e.g., OnlineCP only speeds up ALS by 2 times, and SDT and RLST are much slower than ALS, while our method significantly reduces the time consumption by more than 40 times. Similarly, huge time efficiency difference can be observed between OnlineCP and our approach on NELL-2 dataset, where our method is more than 250 times faster than ALS, whilst the speedup of OnlineCP is only about 30 times. On LastFM dataset, SDT and RLST perform slightly better with around 12 and 7 times faster than ALS, respectively. However, they have been significantly outperformed by OnlineCP (x70 speedup) and our approach (x75 speedup).

TABLE IV
MEAN RELATIVE **FITNESS** TO ALS OVER ALL BATCHES.
THE HIGHER THE BETTER (BOLDFACE MEANS THE BEST RESULTS)

| Dataset | OnlineCP | SDT | RLST | OnlineSCP |
|---|---|---|---|---|
| Facebook-Links | n/a* | n/a | n/a | **1.00** |
| Facebook-Wall | n/a | n/a | n/a | **0.92** |
| MovieLens | **1.00** | 0.31 | **1.00** | **1.00** |
| LastFM | **0.91** | 0.22 | 0.17 | **0.91** |
| NIPS | n/a | n/a | n/a | **0.96** |
| Youtube | n/a | n/a | n/a | **1.00** |
| Enron | n/a | n/a | n/a | **0.93** |
| NELL-2 | 0.97 | n/a | n/a | **0.98** |
| NELL-1 | n/a | n/a | n/a | **0.84** |

* n/a means the method is failed since it is not applicable/running out of memory/cannot finish within 12 hours

TABLE V
MEAN RELATIVE **SPEEDUP** TO ALS OVER ALL BATCHES.
THE HIGHER THE BETTER (BOLDFACE MEANS THE BEST RESULTS)

| Dataset | OnlineCP | SDT | RLST | OnlineSCP |
|---|---|---|---|---|
| Facebook-Links | n/a | n/a | n/a | **3.90** |
| Facebook-Wall | n/a | n/a | n/a | **5.65** |
| MovieLens | 2.03 | 0.05 | 0.02 | **42.06** |
| LastFM | 70.68 | 12.41 | 6.53 | **74.83** |
| NIPS | n/a | n/a | n/a | **102.08** |
| Youtube | n/a | n/a | n/a | **3.14** |
| Enron | n/a | n/a | n/a | **160.24** |
| NELL-2 | 30.12 | n/a | n/a | **258.50** |
| NELL-1 | n/a | n/a | n/a | **27.81** |

TABLE VI
MEAN RELATIVE **MEMORY USAGE** TO ALS OVER ALL BATCHES.
THE LOWER THE BETTER (BOLDFACE MEANS THE BEST RESULTS)

| Dataset | OnlineCP | SDT | RLST | OnlineSCP |
|---|---|---|---|---|
| Facebook-Links | n/a | n/a | n/a | **0.35** |
| Facebook-Wall | n/a | n/a | n/a | **0.32** |
| MovieLens | 17.41 | 60.87 | 46.38 | **0.02** |
| LastFM | 0.36 | 1.24 | 0.94 | **0.01** |
| NIPS | n/a | n/a | n/a | **0.01** |
| Youtube | n/a | n/a | n/a | **0.38** |
| Enron | n/a | n/a | n/a | **0.04** |
| NELL-2 | 1.49 | n/a | n/a | **0.01** |
| NELL-1 | n/a | n/a | n/a | **0.06** |

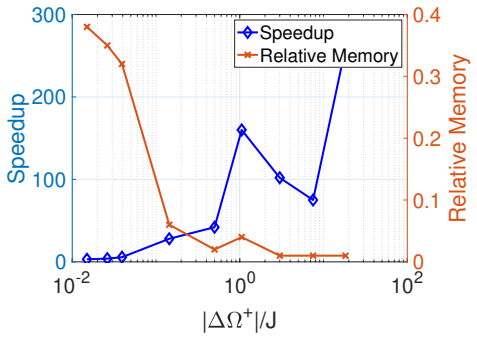



Fig. 2. The efficiency of OnlineSCP is correlated to the ratio between the number of non-zeros in data and the overall length of its non-temporal modes

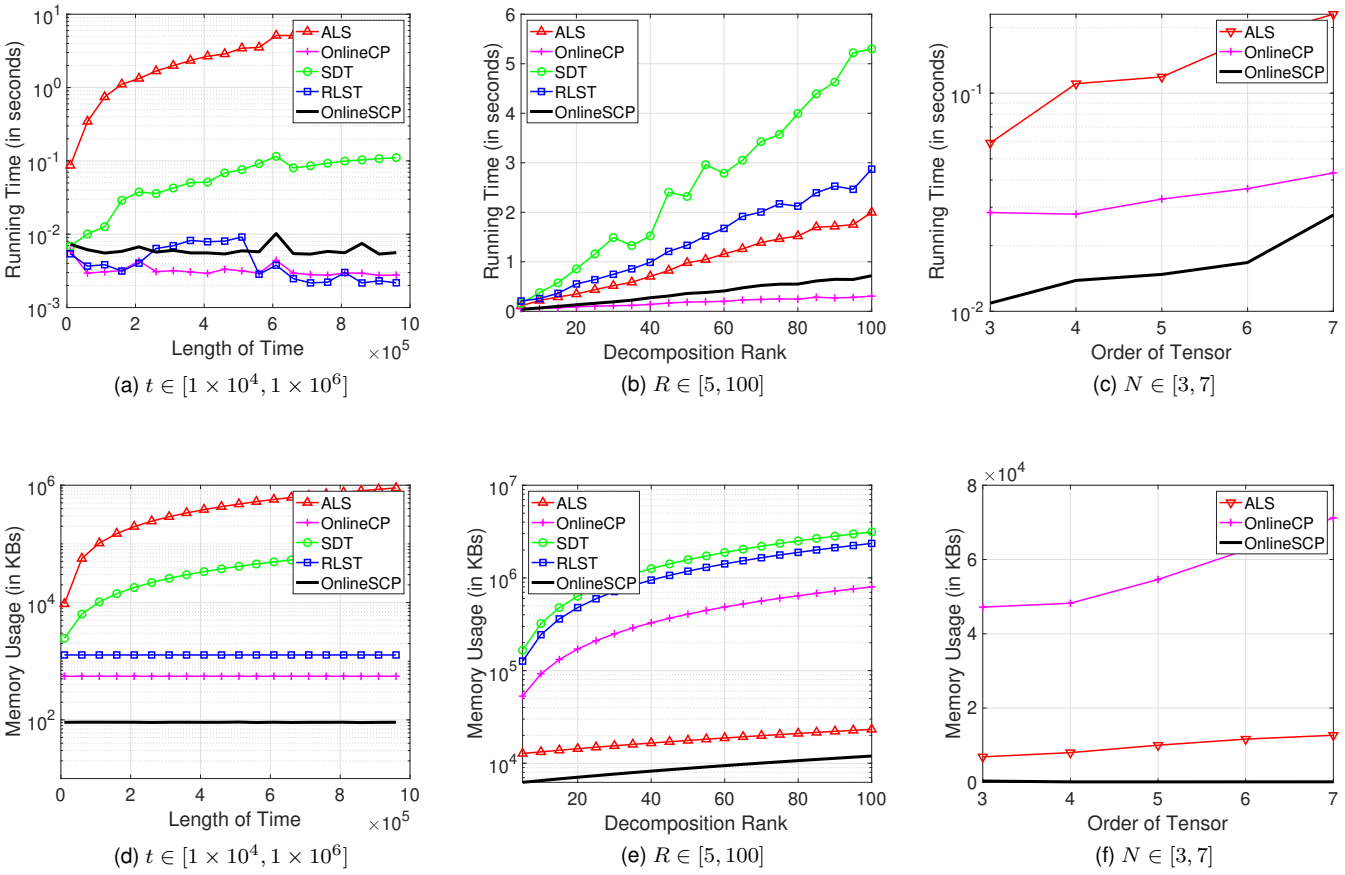
*3) Results on Space Efficiency:* As shown in the complexity comparison (Table II, all existing online methods have a space complexity that is linear to the slice size, which can be much greater than necessary if the data is highly sparse. In fact, this is the main reason why they fail on most of the real-world datasets given the time and memory limits. Sometimes their memory usages can be even worse than ALS, which is a batch method that stores all information with sparse indexing. For example, on MovieLens, the space usage of OnlineCP, SDT and RLST is 17, 60 and 46 times to the memory used by ALS. In contrast, we only use 2% of memory compared to ALS. In addition,

*4) Discussion on the Efficiency Variance of OnlineSCP:* We can see that the efficiency performance of our OnlineSCP method varies from dataset to dataset. For example, compared to ALS, the speedups of OnlineSCP on the social network datasets, Facebook-Links, Facebook-Wall and Youtube are only 3.9, 5.65 and 3.14 times, respectively; while it can improve the time consumption on NIPS, Enron and NELL-2 by more than 100 times. Similar patterns can be found for memory usage as well, where for social network data, around one third of memory used by ALS is needed for OnlineSCP, while for others the relative memory can be as less as 1% w.r.t. ALS.

One can understand this behavior based on the complexity analysis. Specifically, the complexity of OnlineSCP, both in time and space, consists of two parts: 1) one part related to the overall length of the non-temporal modes, $J$, 2) and the other part that depends on the number of non-zeros in the new data, $|\Delta\Omega^+|$. That is, if $J$ is a relatively large number compared to $|\Delta\Omega^+|$, then the speedup obtained by the incremental learning phase will be significantly exploited. In fact, by plotting $|\Delta\Omega^+|/J$ against the speedup (or relative memory) of our method w.r.t. ALS, we can see a clear correlation between this ratio and the efficiency performance as shown in Figure 2. This means that our method tends to work better for tensors that have relatively smaller overall length of the non-temporal modes. However, this does not necessarily means that our method is not suitable for large-scale sparse online tensors. In fact, it should be highlighted that in this experiment setting, each new data batch contains around 0.5% of overall data, so the maximum speedup can be gained in the MTTKRP calculation is about 200 times, compared to the ALS algorithm. In practice, as long as the time used by our algorithm for one new batch of data is less than the data generation time, one can always use a smaller batch size to gain even greater efficiency.

## C. Scalability

As demonstrated by results on real-world datasets, our proposed method, OnlineSCP, significantly improves on existing methods, especially for the efficiency aspect. To better show the merits of our method, we conduct a set of scalability experiments on a group of synthetic tensors that have been randomly generated. Three main factors are considered for




(a) $t \in [1 \times 10^4, 1 \times 10^6]$ (b) $R \in [5, 100]$ (c) $N \in [3, 7]$

(d) $t \in [1 \times 10^4, 1 \times 10^6]$ (e) $R \in [5, 100]$ (f) $N \in [3, 7]$

Fig. 3. Scalability comparison w.r.t. different factors

scalability testing, including: the length of existing time, $t$, the decomposition rank, $R$, and the order of tensor, $N$.

As shown in the real-world experiments, existing online methods, OnlineCP, SDT and RLST, are specifically designed for dense tensors and do not scale well with large $S$. As a result, we set $S = 1000 \times 1000$ unless mentioned otherwise. Even though this is not a large-scale slice size, it is sufficient for us to see the general trends for both time and space consumption. The density of all synthetic tensors are fixed to $1 \times 10^{-3}$ and we repeat all experiments 10 times and report the averaged results over these 10 runs.

*1) Scalability v.s. $t$:* In order to assess the scalability performance with respect to the length of existing time, $t$, we vary $t$ from $10^4$ to $10^6$ and set $R = 5$ and $S = 100 \times 100$, since too large $R$ and $S$ will significantly slow down ALS. At each specific time length, we require all methods to process one slice of new data and report their running time and memory usage in Figure 3a and 3d. Both of them are presented in a log scale. It is clear that OnlineCP, RLST and our OnlineSCP are much better than ALS and SDT. Both the time and memory usage are independent of the existing time for the former, while a linear trend can be found for ALS and SDT. The time efficiency of our method is less than RLST and OnlineCP, since the size of data is not big enough to show the advantage of sparse operation compared to highly optimized matrix calculation. However, the memory usage of OnlineSCP is the least among all methods.

*2) Scalability v.s. $R$:* For scalability regarding to the decomposition rank, $R$, a $1000 \times 1000 \times 100$ tensor is used as the existing data and a new $1000 \times 1000$ slice is input to each method with $R$ chosen from 5 to 100. The results can be found in Figure 3b and 3e (in log scale). As shown in Figure 3b, the running time of ALS, SDT and RLST grows much faster than OnlineCP and OnlineSCP. However, the high efficiency of OnlineCP is gained at the cost of a linear complexity for memory computation due to the intermediate data explosion issue. The growth rates of memory in ALS and OnlineSCP are considerably lower than the others since the sparse MTTKRP is calculated column by column and only a fixed amount of memory that depends on the number of non-zeros is used. The memory growth for these is mainly caused by the storage for loading matrices.

*3) Scalability v.s. $N$:* To validate the scalability of each algorithm with respect to the order of tensor, we conduct experiments on five tensors with orders ranging from 3 to 7, $t = 100$, $R = 5$ and the overall size of non-temporal mode $S \approx 1 \times 10^6$. It should be noted that SDT and RLST are omitted in this experiment since they work with $3^{rd}$-

order tensors only. Figure 3c and 3f demonstrate results for this part of experiment. Regarding to the running time, there is no surprise to see that all three methods are gradually increasing with the growth of tensor order, since more loading matrices need to be updated and the time-consuming MTTKRP calculation need to be done more frequently. In terms of memory usage, the space consumption of ALS and OnlineSCP is stable and an upwards trend can be found in OnlineCP, since it has a specific procedure to calculate all Khatri-Rao product series at a time, in order to speedup the decomposition for higher-order tensors.

Overall, our method demonstrates excellent performance in terms of scalability w.r.t. different factors, compared to state-of-the-art methods. The time complexity of OnlineSCP is only linear to the decomposition rank and the order of tensor, and independent of the existing time. Compared to other online methods, the growth of time in our method is less noticeable since the scaling factor in OnlineSCP is the number of non-zeros in the new data, while for other methods, they have to be scaled by the overall size of non-temporal modes. Additionally, our method is also the most memory-efficient and scalable method among all algorithms and it always consumes the least amount of memory, which is also linear to the number of non-zeros in the new data.

## VI. Conclusions and Future Work

To conclude, this paper addressed the CP decomposition problem for online sparse tensors. We proposed an efficient algorithm, OnlineSCP, to incrementally track the up-to-date CP decomposition when a new batch of data is appended to the time mode of a sparse tensor. The complexity of our algorithm is only linear to the number of non-zeros in the new data, which is significantly better than existing online approaches that are designed for dense tensors. As evaluated on both real-world and synthetic datasets, our algorithm demonstrated considerable improvements over state-of-the-art techniques, in terms of decomposition quality, time and space efficiency, and scalability.

One possible strategy to further speedup our method is replacing the current MTTKRP procedure with more advanced hardware acceleration techniques such as GPU acceleration. Another potential future direction is imposing constraints such as non-negativity.

## References


[1] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011.

[2] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[3] A. Globerson, G. Chechik, F. Pereira, and N. Tishby, "Euclidean Embedding of Co-occurrence Data," *JMLR*, vol. 8, pp. 2265–2295, 2007.

[4] T. G. Kolda, B. W. Bader, and J. P. Kenny, "Higher-order web link analysis using multilinear algebra," in *ICDM*, 2005.

[5] W. Hu, X. Li, X. Zhang, X. Shi, S. Maybank, and Z. Zhang, "Incremental tensor subspace learning and its applications to foreground segmentation and tracking," *IJCV*, vol. 91, no. 3, pp. 303–327, 2011.

[6] A. Anandkumar, R. Ge, D. J. Hsu, and S. M. Kakade, "A tensor approach to learning mixed membership community models." *JMLR*, vol. 15, no. 1, pp. 2239–2312, 2014.

[7] G. Ma, L. He, C.-T. Lu, P. S. Yu, L. Shen, and A. B. Ragin, "Spatio-temporal tensor analysis for whole-brain fmri classification," in *SDM*, 2016.

[8] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[9] J. H. Choi and S. Vishwanathan, "Dfacto: Distributed factorization of tensors," in *NIPS*, 2014.

[10] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *ICDE*, 2015.

[11] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *SIGKDD*, 2012.

[12] D. Nion and N. D. Sidiropoulos, "Adaptive algorithms to track the parafac decomposition of a third-order tensor," *IEEE Transactions on Signal Processing*, vol. 57, no. 6, pp. 2299–2310, 2009.

[13] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online cp decompositions for higher order tensors," in *SIGKDD*, 2016.

[14] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2007.

[15] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in *IPDPS*, 2015.

[16] J. Li, J. Choi, I. Perros, J. Sun, and R. Vuduc, "Model-driven sparse cp decomposition for higher-order tensors," in *IPDPS*, 2017.

[17] B. Liu, C. Wen, A. D. Sarwate, and M. M. Dehnavi, "A unified optimization approach for sparse tensor operations on gpus," in *CLUSTER*, 2017.

[18] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," in *SIGKDD*, 2006.

[19] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: Theory and applications," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2, no. 3, p. 11, 2008.

[20] E. Gujral, R. Pasricha, and E. E. Papalexakis, "Sambaten: Sampling-based batch incremental tensor decomposition," in *SDM*, 2018.

[21] S. Zhou, S. M. Erfani, and J. Bailey, "Sced: A general framework for sparse tensor decomposition with constraints and elementwise dynamic learning," in *ICDM*, 2017.

[22] K. Shin, B. Hooi, J. Kim, and C. Faloutsos, "Densealert: Incremental dense-subtensor detection in tensor streams," in *SIGKDD*, 2017.

[23] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu, "Multi-aspect streaming tensor completion," in *SIGKDD*, 2017.

[24] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.6," Available online, February 2015

[25] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proceedings of the 2nd ACM workshop on Online social networks*. ACM, 2009, pp. 37–42.

[26] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *WSDM*, 2017.

[27] O. C. Herrada, "Music recommendation and discovery in the long tail," Ph.D. dissertation, Universitat Pompeu Fabra, 2009.

[28] A. E. Mislove, "Online social networks: measurement, analysis, and applications to distributed information systems," Ph.D. dissertation, Rice University, 2009.

[29] J. Shetty and J. Adibi, "The enron email dataset database schema and brief statistical report," *Information sciences institute technical report, University of Southern California*, vol. 4, 2004.

[30] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell, "Toward an architecture for never-ending language learning." in *AAAI*, 2010.

[31] J. Kunegis, "Konect: the koblenz network collection," in *WWW*, 2013.

[32] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. "FROSTT: The formidable repository of open sparse tensors and tools". Available online, 2017.

[33] L. Lafayette and B. Wiebelt, "Spartan and nemo: Two hpc-cloud hybrid implementations," in *e-Science*, 2017.