Lawrence Berkeley National Laboratory
LBL Publications

Title
Efficient Tensor Completion Algorithms for Highly Oscillatory Operators

Permalink
https://escholarship.org/uc/item/4hq2h106

Journal
Numerical Linear Algebra with Applications, 33(4)

ISSN
1070-5325

Authors
Singh, N
Solomonik, E
Li, XS
et al.

Publication Date
2026

DOI
10.1002/nla.70098

Copyright Information
This work is made available under the terms of a Creative Commons Attribution License, available at https://creativecommons.org/licenses/by/4.0/

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Check for updates

Numerical Linear Algebra with Applications

WILEY

RESEARCH ARTICLE OPEN ACCESS

Efficient Tensor Completion Algorithms for Highly Oscillatory Operators

Navjot Singh[1] | Edgar Solomonik[1] | Xiaoye Sherry Li[2] | Yang Liu[2]

[1]Department of Computer Science, University of Illinois, Urbana-Champaign, IL, USA | [2]Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Correspondence: Navjot Singh (navjot2@illinois.edu)

Received: 18 October 2025 | **Revised:** 17 May 2026 | **Accepted:** 5 June 2026

ABSTRACT

We address the problem of recovering highly oscillatory operators, represented as $n \times n$ matrices with a fixed set of observed entries. Given that these matrices can be well compressed by butterfly matrix decomposition of $L = \mathcal{O}(\log n)$ levels requiring only $O(n \log n)$ degrees of freedom, we propose a novel reformulation of the butterfly structure as a compact tensor network. Specifically, we reshape the input matrix as an order $2L + 2$ dense tensor, and cast its butterfly decomposition as a tensor network consisting of order $L + 3$ dense tensors. This enables efficient utilization of the existing software infrastructure for dense and sparse tensor computations. Next, we propose several tensor completion algorithms based on the tensor reformulation of butterfly format, and compare them against algorithms using the quantized tensor train (QTT) format. These algorithms leverage popular completion methods such as alternating least squares, gradient-based optimization, and alternating directions fitting. Moreover, we propose a novel strategy that uses low-rank matrix completion to efficiently generate an initial guess for the proposed algorithms. To demonstrate the efficiency and applicability of our proposed algorithms, we perform three numerical experiments using simulated oscillatory operators in seismic applications. In these experiments, we use $\mathcal{O}(n \log n)$ observed entries in the input matrix and demonstrate an $\mathcal{O}(n \log^3 n)$ computational cost of the proposed algorithms, leading to speedups of orders of magnitude and significant accuracy improvements for large matrices compared to state-of-the-art low-rank matrix and QTT tensor completion algorithms.

1 | Introduction

Matrix or tensor completion is the task of inferring missing entries in 2D matrices or higher-order tensors from partial observations, which may be corrupted by noise. Completion problems have been widely studied for applications across data science (e.g., recommendation system [1] and computer vision [2]), imaging science (e.g., seismic inversion [3, 4], magnetic resonance imaging [5, 6], and X-ray tomography [7]), and high performance computing [8].

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2026 The Author(s). *Numerical Linear Algebra with Applications* published by John Wiley & Sons Ltd.

In many realistic applications, it is either too expensive or intrinsically impossible to collect the full matrix or tensor of measurement or simulation data. That said, the matrix or tensor can be fully reconstructed with reduced numbers of degrees of freedom given the following assumptions: (1) the matrix or tensor exhibits certain algebraic structures such as low-rankness or sparsity, and (2) the observed entries satisfy certain incoherence or randomness. Specifically, the matrix or tensor can be represented as a low-rank matrix or tensor decomposition, such as CANDECOMP/PARAFAC (CP) [9, 10], Tucker [11], tensor train (TT) [12], hierarchical Tucker [13], and tensor ring [14] decompositions. One can then define objective functions as the difference between the observed entries and the extracted ones from the low-rank representation, and use optimization algorithms to update the low-rank factors until convergence. Popular optimization algorithms include nuclear norm minimization [15], alternating least squares (ALS) [10, 16], gradient-based optimization [17], alternating direction fitting (ADF) [18] and Riemannian optimization [19], etc. More recent advances involve integrating more sophisticated low-rank structures and smooth data priors into the completion algorithm frameworks [20, 21]. It's worth mentioning that unlike other low-rank reconstruction problems that allow, for example, randomized sketching of all entries or evaluation of arbitrary entries, the completion problems can only work with a *predetermined* subset of entries.

For highly oscillatory operators such as large-scale Radon transforms or Green's functions for high-frequency seismic inversion, the aforementioned low-rank matrix and tensor completion algorithms do not approximate the underlying structure of these operators efficiently, leading to increasing rank(s) and high computational cost. Fortunately, butterfly matrices [22–26] and tensors [27] are particularly designed for highly oscillatory operators. They have been used to represent integral transforms [28–32], integral operators [26, 33–36], PDE operators, [37, 38] and neural network parameters [39–48]. Butterfly decompositions are well-suited to compress measurement operators of sizes $n \times n$ in seismic applications with $\mathcal{O}(n \log n)$ (via butterfly matrix) or even $\mathcal{O}(n)$ (via butterfly tensor of [27]) computational and memory complexities. Therefore, butterfly decompositions will be powerful tools if used in matrix or tensor completion of oscillatory operators, as they require significantly lower complexity and less number of observed entries than other existing low-rank completion algorithms. This motivates us to develop completion algorithms in the butterfly format in this current paper.

In this paper, we develop efficient algorithms to perform completion of an oscillatory matrix via butterfly decomposition. Our contribution is three-fold: (1) First, we reshape the matrix into a higher dimensional tensor by using similar tensorization techniques as quantized tensor train (QTT) decomposition [49]. Under such reshaping, the corresponding butterfly matrix decomposition can be converted into a higher-dimensional tensor network, which we call tensor reformulation of butterfly matrices. Note that the result of tensor reformulation is mathematically equivalent to butterfly matrix representation, but makes it convenient to leverage existing tensor network completion software and kernels. This is not to be confused with the butterfly tensor of [27] which directly constructs a tensor decomposition from a tensor operator without reshaping. (2) Next, we propose three tensor completion algorithms for the tensorized butterfly matrix based on ALS, ADF and gradient optimization, with an initial guess generated and converted from low-rank matrix completion algorithms. We analyze and demonstrate that for highly oscillatory operators, in the experiments presented in this manuscript, these proposed completion algorithms require only $\mathcal{O}(n \log n)$ observed entries and thus achieve $\mathcal{O}(n\log^3 n)$ computational complexities. This is in stark contrast to other completion algorithms such as QTT completion or low-rank matrix completion, which require asymptotically larger numbers of arithmetic operations. It is also worth mentioning that although ALS and gradient optimization-based algorithms have been considered for butterfly matrices [40, 47, 50], these existing algorithms assume n^2 observed entries instead of a matrix completion setting. (3) Finally, we apply the proposed butterfly completion algorithms to several operators in seismic imaging applications to numerically demonstrate their superior convergence, accuracy and efficiency compared with other existing completion algorithms. Aside from applications in tensor completion, we also argue that the proposed tensor reformulation of butterfly can be used to represent neural network layers and allows for efficient automatic differentiation.

The rest of the paper is organized as follows. In Section 2 we introduce background information about tensor decomposition, completion and existing software. In Section 3 we briefly summarize the definition of the butterfly matrix. The proposed tensor reformulation of a butterfly matrix is detailed in Section 4 followed by two proposed completion algorithms with implementation detail and complexity analysis in Section 5. Finally, we provide three sets of numerical examples to demonstrate the accuracy, convergence and efficiency of the proposed algorithms in Section 6.

1.1 | Notations

Given a matrix $\boldsymbol{T}$, we use $\boldsymbol{T}(i, j)$ or $\boldsymbol{T}(t, s)$ to denote its entries or submatrices, where i, j are indices and t, s are index sets. Alternatively, we use $\boldsymbol{t}_i$ to denote its i-th row or column, and $t_{i,j}$ to denote its entries. For an order d tensor $\boldsymbol{\mathcal{T}}$, we

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

use $\mathcal{T}(i_1, i_2, \cdots, i_d)$ to denote its entries, where $(i_1, i_2, \cdots, i_d)$ are index tuples. Alternatively, we use $t_{i_1,i_2,\cdots,i_d}$ to denote its entries. For a set Ω consisting of order d index tuples, Ω_{i_k} denotes its subset consisting of tuples whose kth elements are fixed to i_k. Throughout this chapter, for brevity of notation, we use superscripts and subscripts to label tensors, for example, $\boldsymbol{A}^k$ refers to a matrix with label k and not the kth power of $\boldsymbol{A}$, $\boldsymbol{A}_{i,j}$ refers to a matrix indexed by two indices i, j and does not refer to its row and columns. We define a block diagonal matrix with rectangular (possibly non-square) blocks using the notation $\mathrm{diag}_i\{\boldsymbol{T}_i\}_1^n$ to represent a block diagonal matrix whose blocks are the matrices $\boldsymbol{T}_i \in \mathbb{R}^{m_i \times n_i}$, for $i = 1, \ldots, n-1$. That is, it is a matrix given as

$$\mathrm{diag}_i\{\boldsymbol{T}_i\}_1^n := \begin{bmatrix} \boldsymbol{T}_1 & & & \\ & \boldsymbol{T}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{T}_{n-1} \end{bmatrix}, \tag{1.1}$$

where all off-diagonal blocks are zero. The submatrix corresponding to block $\boldsymbol{T}_i$ occupies rows from $\sum_{l=1}^{i-1} m_l + 1$ to $\sum_{l=1}^{i} m_l$, and columns from $\sum_{l=1}^{i-1} n_l + 1$ to $\sum_{l=1}^{i} n_l$ with 1−based indexing.

2 | Background on Tensor Computations

2.1 | Tensor Decompositions

Tensor decompositions are mathematical constructs used to efficiently represent and manipulate high-dimensional tensors. The key idea behind tensor decompositions is to decompose a tensor into a set of simpler tensors linked together according to a specific pattern or topology. A tensor decomposition is represented by a set of parameter tensors $\mathcal{S}$ that are contracted in a specific way according to the definition of the decomposition to reconstruct the tensor $\boldsymbol{\mathcal{X}}(\mathcal{S})$.

The simplest example of a low rank tensor decomposition is a low rank matrix decomposition of a matrix $\boldsymbol{M} \in \mathbb{R}^{I \times J}$ with rank R, given by $\boldsymbol{M} \approx \boldsymbol{S}^1 \boldsymbol{S}^{2T}$, with parameter tensors $\boldsymbol{S}^1 \in \mathbb{R}^{I \times R}$, $\boldsymbol{S}^2 \in \mathbb{R}^{J \times R}$, and parameter tensor set $\mathcal{S} = \{\boldsymbol{S}^1, \boldsymbol{S}^2\}$, where $\boldsymbol{X}(\mathcal{S}) = \boldsymbol{S}^1 \boldsymbol{S}^{2T}$.

Computing an optimal rank tensor decomposition for most problems is NP-hard [51]. Therefore, numerical optimization techniques are used to compute the tensor decomposition by formulating an objective function that is minimized with respect to the tensor decomposition parameters. We formally define the low-rank tensor decomposition optimization problem for a given tensor $\mathcal{T}$ in this subsection. The objective that is minimized to compute the low rank tensor decomposition $\boldsymbol{\mathcal{X}}(\mathcal{S})$,

$$\min_{\mathcal{S}} \frac{1}{2} \|\mathcal{T} - \boldsymbol{\mathcal{X}}(\mathcal{S})\|_F^2, \tag{2.1}$$

where $\boldsymbol{\mathcal{X}}(\mathcal{S})$ is the reconstructed tensor. Another critical parameter that is to be determined and not mentioned here is rank(s) or dimensions of the tensors in the set $\mathcal{S}$. These parameters are often determined based on the desired size and representation accuracy of the tensor decomposition.

Various types of tensor decompositions have been proposed such as CANDECOMP/PARAFAC (CP) [9], Tucker [11], tensor train [12], hierarchical Tucker [13], tensor ring [14]. We review the tensor train decomposition in this paper as it has been shown to approximate oscillatory operators [27, 52, 53] and is considered as one of the baseline algorithms in this work. Given an order $2L + 2$ tensor $\mathcal{T}$ representing a $L + 1$ dimensional operator, the tensor train decomposition or matrix product operator (MPO) representation [54] of $\mathcal{T}$ is written as

$$t_{i_0,\ldots,i_L,j_0\ldots j_L} = \sum_{r_1,\ldots,r_L} s^1_{i_0,j_0,r_1} \left(\prod_{m=1}^{L-1} s^{m+1}_{i_m,j_m,r_m,r_{m+1}} \right) s^{L+1}_{i_L,j_L,r_L} \tag{2.2}$$

Here $i_0, \ldots, i_L$ and $j_0, \ldots, j_L$ are used to index the input and output dimensions of the operator $\mathcal{T}$, and the parameter tensor set is $\mathcal{S} = \{\boldsymbol{S}^1, \boldsymbol{S}^2, \ldots, \boldsymbol{S}^{L+1}\}$. Note that the tensors $\boldsymbol{S}^1$ and $\boldsymbol{S}^{L+1}$ are order 3 tensors, and the rest are order 4 tensors. When given an operator in the matrix form, for example, $\boldsymbol{T} \in \mathbb{R}^{n \times n}$ with typically $n = 2^{L+1}$, the matrix can be reshaped and permuted into an order $2L + 2$ tensor and then represented by the tensor train decomposition in (2.2). This is called

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

the quantized tensor train decomposition (QTT) [49] and has been used to represent both sparse and dense matrices [55, 56]. Tensor train decomposition (specifically MPO) has been widely used in accelerating quantum chemistry simulations [57], quantum circuit simulations [58], and many time-dependent, steady-state, low-dimensional and high-dimensional PDEs [59–63].

2.2 | Tensor Completion

The objective function in (2.1) assumes that all entries of the tensor are available and utilized in computing the decomposition. However, in many applications of interest, only a fixed and limited subset of the tensor entries is observed. To address this, a more general objective function is considered, given as

$$\min_{\mathcal{S}} \frac{1}{2} \|\mathcal{P}_\Omega(\boldsymbol{\mathcal{T}} - \boldsymbol{\mathcal{X}}(\mathcal{S}))\|_F^2, \tag{2.3}$$

where Ω is the set of observed entries, and $\mathcal{P}_\Omega$ is a selection operator that selects the entries of tensor denoted by Ω. When Ω is a strict subset of all the entries of the tensor, the computation of the tensor decomposition and using the tensor decomposition to compute the unobserved entries of the tensor is commonly referred to as *tensor completion*. Low rank matrix completion [64–66] is one of the most established subjects in this category. Besides low rank matrix completion, many algorithms have been proposed for tensor completion in different formats, such as CP [2], Tucker [19, 67], tensor train [18, 68, 69], hierarchical Tucker [70, 71], and tensor ring decomposition [72].

It is important to note that tensor completion differs from scenarios where arbitrary entries of the tensor can be sampled at will. Instead, only a *fixed* subset of entries are provided. The arbitrary sampling problem is generally easier than the completion setting, and many algorithms have been proposed for different tensor decompositions in this context as well [73–75].

2.3 | Software for Tensor Decomposition and Completion

Computing tensor decomposition and completion involve optimization of objective functions in (2.1) and (2.3). There is a rich space of algorithms for these optimization problems, but their primary computational components are tensor contractions and linear algebra. Hence, the algorithms can be implemented efficiently by making use of the existing software libraries and infrastructure. We review several such libraries below.

Tensor Toolbox [76] is a MATLAB-based library that provides a wide variety of tensor operations and decompositions for both dense and sparse data. It supports popular models such as CP, Tucker, and block term decompositions. Due to its MATLAB foundation, Tensor Toolbox is widely used for prototyping and algorithm development. However, it is limited to single-node computation and thus is not suitable for very large-scale problems.

Cyclops Tensor Framework (CTF) [77] is a C++ library that provides software infrastructure to perform dense and sparse large-scale tensor computations in a distributed-memory setting. CTF has been used to implement various tensor decomposition algorithms [78–81]. Recently, new sparse tensor kernels to support tensor completion algorithms in Python were added to CTF [82]. Additionally, a framework to support sparse tensor contractions with tensor networks [83] has been proposed using CTF.

SPLATT (Sparse Tensor Algebra Toolbox) [84] is a high-performance C/C++ library specifically optimized for sparse tensors. It provides parallel implementations of least squares CP decomposition as well as CP completion algorithms and supports modes of parallelism including shared-memory (OpenMP), distributed-memory (MPI), and hybrid configurations. SPLATT uses a compressed sparse fiber (CSF) [85] data structure that generalizes compressed sparse row (CSR) format to tensors.

TuckerMPI is a scalable C++/MPI package designed for large-scale tensor compression based on the Tucker decomposition [86]. It is optimized for distributed computing environments and allows efficient storage and analysis of large-scale dense tensors and grid datasets, making it suitable for large-scale Tucker decomposition.

TensorLy is a Python library that provides high-level tools and efficient implementations for tensor decomposition, tensor algebra, and tensorized deep learning, with extensive backend support for frameworks such as NumPy, PyTorch, and

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TensorFlow [87]. Many algorithms for CP, Tucker, tensor train decomposition are available in Tensorly with different backends, allowing for efficient and easy-to-use tensor decomposition on different hardware.

3 | Background on Butterfly Decomposition

3.1 | Complementary Low Rank Property

Oscillatory integral operators arise in many areas of applied mathematics and physics, particularly in wave propagation, quantum mechanics, and harmonic analysis. A key challenge in working with such operators is that, when discretized, they lead to matrices that are dense and typically lack global low-rank structure. As a result, standard low-rank compression techniques fail, and naive discretization followed by direct matrix-vector multiplication incurs a high computational cost of $\mathcal{O}(n^2)$, where n is the total number of degrees of freedom. In high-dimensional problems, this cost is expensive. For example, discretizing a 3D domain with N points per dimension leads to $n = N^3$, making the matrix size $N^3 \times N^3$, and the multiplication cost scales as $\mathcal{O}(N^6)$.

This discretized matrix may be dense and high-rank, however, it exhibits a structured pattern in which certain submatrices are numerically low-rank. The property that characterizes which blocks are low-rank is known as the *complementary low-rank property* (CLR) [24]. This property is mathematically described as follows.

Let $\boldsymbol{T}$ be an $m \times n$ matrix. Consider rows to be indexed by a set χ and columns to be indexed by a set ξ, where $\chi = \{0, \dots, m-1\}$ and columns $\xi = \{0, \dots, n-1\}$, where without loss of generality we assume $m = n$. For each of the row and column sets, complete binary trees $\mathcal{T}_\chi$ and $\mathcal{T}_\xi$ are associated respectively, each comprising $L = \mathcal{O}(\log n)$ levels. We extend the complete binary trees such that each node at level L has at most c children which referred to as the leaf nodes. For simplicity, we assume that all nodes have a leaf size c, and that the number of rows and columns satisfies $n = c \cdot 2^L$. However, this definition can be readily extended to arbitrary row, column, and leaf-node sizes. Let τ^l denote the set of all sets of indices at level l in tree $\mathcal{T}_\chi$ and similarly ν^l denote the set of all sets of indices at level l in tree $\mathcal{T}_\xi$. We have that $\forall i, k \in \{0, \dots, 2^l - 1\},\ l \in \{0, \dots, L\}$ with $i \neq k$, $\tau_i^l \cap \tau_k^l = \emptyset$, $\tau_i^l = \tau_{2i}^{l-1} \cup \tau_{2i+1}^{l-1}$, and $\tau_0^0 = \{0, \dots, n-1\}$ and analogously for ν^l.

The matrix $\boldsymbol{T}$ satisfies *complementary low-rank property* (CLR) if for any row set or node τ_i^l at level l in $\mathcal{T}_\chi$ and column set or node ν_j^{L-l} at level $L-l$ in $\mathcal{T}_\xi$,

$$\boldsymbol{T}(\tau_i^l, \nu_j^{L-l}) \text{admits a low-rank approximation}$$

where the rank of each submatrix $\boldsymbol{T}(\tau_i^l, \nu_j^{L-l})$, denoted by $r_{i,j}^l$, is bounded by a small constant r (the *butterfly rank*) [23–25, 88]. This special property enables the computation a butterfly decomposition, which can reduce the storage complexity of a matrix and computational complexity of matrix-vector multiplications from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. We describe the butterfly decomposition and briefly describe the algorithms to compute it in the next subsection.

3.2 | Butterfly Matrix Decomposition

Let $\boldsymbol{T}$ be an $n \times n$ input matrix with CLR, and $n = c \cdot 2^L$, with L being the number of levels as described above. We assume the number of levels, $L = 2M$, to be even for convenience and simplicity of description. A butterfly matrix decomposition can be described for an arbitrary number of levels. By CLR, for any level $l \in \{0, \dots, L\}$ in $\mathcal{T}_\chi$,

$$\boldsymbol{T}(\tau_i^l, \nu_j^{L-l}) \approx \boldsymbol{U}_{i,j}^{L-l} \boldsymbol{V}_{i,j}^l, \quad \forall i \in \{0, \dots, 2^l - 1\},\ \forall j \in \{0, \dots, 2^{L-l} - 1\}, \tag{3.1}$$

where $\boldsymbol{U}_{i,j}^{L-l} \in \mathbb{R}^{|\tau_i^l| \times r_{i,j}^{L-l}}$ and $\boldsymbol{V}_{i,j}^l \in \mathbb{R}^{r_{i,j}^{L-l} \times |\nu_j^{L-l}|}$ are low-rank decomposition factors. Note that for $l = 0$, we have $i = 0$ only; similarly, for $l = L$, we have $j = 0$ only. Despite possibly being non-unitary, we refer to them as the column and row basis matrices. Each of the ranks $r_{i,j}^l$ is bounded by a constant r, allowing for decomposition of the input matrix into $\mathcal{O}(\log n)$ block sparse factors, with each factor having $\mathcal{O}(n)$ nonzeros. The decomposition of the matrix into these factors is known as the butterfly decomposition. We derive the form of butterfly decomposition by starting at the finest levels of both the trees and recursively constructing the bases for intermediate levels by propagating the row and column spaces derived at these boundary levels.

At level $l = 0$ in $\mathcal{T}_\chi$, the row basis matrices $\boldsymbol{V}_{0,j}^0$ each of size $r_{0,j}^L \times c$ are defined by a low-rank factorization of each corresponding submatrix $\boldsymbol{T}(\tau_0^0, \nu_j^L)$, $\forall j \in \{0, \dots, 2^L - 1\}$. Here τ_0^0 is the row index set at the root of the tree $\mathcal{T}_\chi$. Note that for levels $l \in \{1, \dots, M\}$, each submatrix $\boldsymbol{T}(\tau_i^l, \nu_j^{L-l})$, comprises row indices of its children in $\mathcal{T}_\chi$, i.e., the matrix is

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

concatenated column-wise (and split row-wise). As the matrix is concatenated column-wise, the row basis matrices $\boldsymbol{V}^{l}_{i,j}$ are formed by concatenating linear combinations of the row basis matrices corresponding to child nodes at the level $l+1$. The row basis matrices $\boldsymbol{V}^{l}_{i,j}$, with corresponding column and row index sets τ^{l}_{i} and ν^{L-l}_{j}, are represented in a nested fashion in terms of the children of row index sets, ν^{L-l+1}_{2j} and ν^{L-l+1}_{2j+1}, given as

$$\boldsymbol{V}^{l}_{i,j} = \boldsymbol{W}^{l}_{i,j}\begin{bmatrix}\boldsymbol{V}^{l+1}_{i,2j} & \\ & \boldsymbol{V}^{l+1}_{i,2j+1}\end{bmatrix}, \quad \forall l \in \{1,\dots,M\}, \tag{3.2}$$

where $\boldsymbol{W}^{l}_{i,j}$ is called the transfer matrix for corresponding set $\tau^{l}_{i}, \nu^{L-l}_{j}$. By CLR, each $\boldsymbol{W}^{l}_{i,j}$ is of size $r^{L-l}_{i,j} \times (r^{L-l+1}_{i,2j} + r^{L-l+1}_{i,2j+1})$, which is at most $r \times 2r$.

Analogously, at level $l = 0$ of $\mathcal{T}_{\xi}$, the column basis matrices $\boldsymbol{U}^{0}_{i,0}$, of size $c \times r^{0}_{i,0}$ is defined by a factorization of each corresponding submatrix $\boldsymbol{T}(\tau^{L}_{i}, \nu^{0}_{0})\ \forall i \in \{0,\dots,2^{L}-1\}$. At levels $l \in \{1,\dots,M\}$, the column basis matrices at level l are concatenation of the linear combination of corresponding basis at level $l+1$. Therefore, column basis, $\boldsymbol{U}^{l}_{i,j}$, with corresponding column and row index sets τ^{l}_{i} and ν^{L-l}_{j}, are represented in a nested fashion in terms of the children of index sets, τ^{l+1}_{2i} and τ^{l+1}_{2i+1}, given as

$$\boldsymbol{U}^{l}_{i,j} = \begin{bmatrix}\boldsymbol{U}^{l+1}_{2i,j} & \\ & \boldsymbol{U}^{l+1}_{2i+1,j}\end{bmatrix}\boldsymbol{P}^{l}_{i,j}, \quad \forall l \in \{1,\dots M\}, \tag{3.3}$$

where $\boldsymbol{P}^{l}_{i,j}$ is also called the transfer matrix for corresponding set $\tau^{l}_{i}, \nu^{L-l}_{j}$. Each $\boldsymbol{P}^{l}_{i,j}$ is of size $(r^{l-1}_{2i,j} + r^{l-1}_{2i+1,j}) \times r^{l}_{i,j}$ which is bounded by $2r \times r$ by CLR property.

At level $M = L/2$ of both the trees (half level), from (3.1), we have a low-rank representation of a block $\boldsymbol{T}(\tau^{M}_{i}, \nu^{M}_{j})$, given by

$$\boldsymbol{T}(\tau^{M}_{i}, \nu^{M}_{j}) \approx \boldsymbol{U}^{M}_{i,j}\boldsymbol{V}^{M}_{i,j}, \tag{3.4}$$

where $\boldsymbol{U}^{M}_{i,j}$ is the column basis matrix at level M and $\boldsymbol{V}^{M}_{i,j}$ is the row basis matrix at level M of both the trees for a given i, j. The column basis matrix $\boldsymbol{U}^{M}_{i,j}$ can be computed recursively using (3.3) corresponding to the given τ^{M}_{i} and ν^{M}_{j}. The column basis for τ^{M}_{i} and ν^{M}_{j} is given as

$$\boldsymbol{U}^{M}_{i,j} = \overline{\boldsymbol{U}}_{i}\left(\prod_{l=1}^{M}\overline{\boldsymbol{P}}^{l}_{i,j}\right),$$
$$\text{where } \overline{\boldsymbol{U}}_{i} = \operatorname{diag}_{k}\left\{\boldsymbol{U}^{0}_{k,0}\right\}^{2^{M}(i+1)}_{2^{M}i}, \tag{3.5}$$

and the transfer factors $\overline{\boldsymbol{P}}^{l}_{i,j}$ for $l \in \{1,\dots,M\}$ are block diagonal matrices. They consist of transfer matrices $\boldsymbol{P}^{l}_{i,j}$, given as

$$\overline{\boldsymbol{P}}^{l}_{i,j} = \operatorname{diag}_{k}\left\{\boldsymbol{P}^{l}_{k,j}\right\}^{2^{M-l}(i+1)}_{2^{M-l}i}. \tag{3.6}$$

The row basis matrix $\boldsymbol{V}^{M}_{i,j}$ can be computed recursively using (3.2) and described analogously.

Since (3.4) is defined for an arbitrary i and j, collecting all the submatrices of $\boldsymbol{T}$ at level M, we have the following approximation of $\boldsymbol{T}$,

$$\boldsymbol{T} \approx \boldsymbol{K} = \boldsymbol{S}^{1}\left(\prod_{m=1}^{M}\boldsymbol{S}^{m+1}\right)\left(\prod_{m=M+1}^{L}\boldsymbol{S}^{m+1}\right)\boldsymbol{S}^{L+2}, \tag{3.7}$$

where

$$\boldsymbol{S}^{1} = \operatorname{diag}_{i}\left\{\boldsymbol{U}^{0}_{i,0}\right\}^{2^{L}}_{0},$$
$$(\boldsymbol{S}^{m+1})^{T} = \operatorname{diag}_{j}\left\{\begin{bmatrix}\operatorname{diag}_{i}\left\{(\boldsymbol{P}^{m}_{i,2^{m-1}j})^{T}\right\}^{2^{L-m}}_{0} \\ \operatorname{diag}_{i}\left\{(\boldsymbol{P}^{m}_{i,2^{m-1}j+1})^{T}\right\}^{2^{L-m}}_{0}\end{bmatrix}\right\}^{2^{m-1}}_{0}, \ \forall m \in \{1,\dots,M\}, \tag{3.8}$$

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

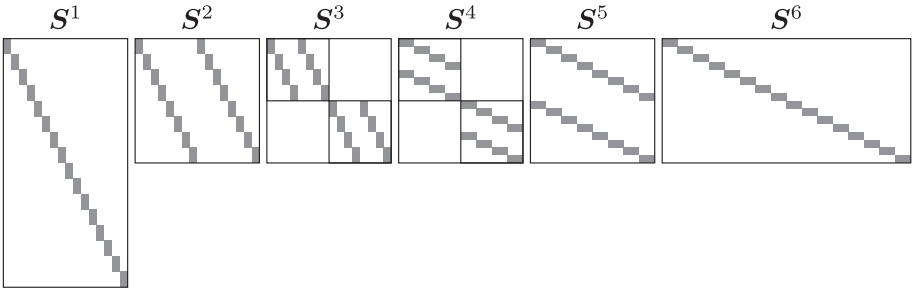


FIGURE 1 | Illustration of a 4-level butterfly matrix decomposition $\boldsymbol{X} = \boldsymbol{S}^1\boldsymbol{S}^2\boldsymbol{S}^3\boldsymbol{S}^4\boldsymbol{S}^5\boldsymbol{S}^6$.

and $\boldsymbol{S}^{L+2}$, and $\boldsymbol{S}^{m+1}$, $m \in \{M+2, \dots, L\}$ are defined analogously.

For reference, we plot an example of 4-level butterfly matrix decomposition of (3.8) in Figure 1. In $\boldsymbol{S}^2$ and $\boldsymbol{S}^3$, each approximately $2r \times r$ block (shown in gray) represents one $\boldsymbol{P}^l_{i,j}$ in $\boldsymbol{S}^4$ and $\boldsymbol{S}^5$, each approximately $r \times 2r$ block (shown in gray) represents one $\boldsymbol{W}^l_{i,j}$.

Each $\boldsymbol{U}^L_{i,0}$ and $(\boldsymbol{V}^0_{0,j})^T$ is of size $c \times r$ with $i, j \in \{0, \dots, 2^L - 1\}$. Each $\boldsymbol{P}^m_{i,j}$ is of size $2r \times r$ with $i \in \{0, \dots, 2^{L-m} - 1\}$ and $j \in \{0, \dots, 2^m - 1\}$ for $m \in \{1, \dots, M\}$. Similarly, each $\boldsymbol{W}^l_{i,j}$ is of size $r \times 2r$, with $i \in \{0, \dots, 2^l - 1\}$ and $j \in \{0, \dots, 2^{L-l} - 1\}$ for $l \in \{1, \dots, H\}$. Therefore, the outermost blocks require $O(cr \cdot 2^L)$ memory, and all of the inner blocks require $O(r^2 L \cdot 2^{L+1})$ memory. Assuming r and c to be constant, and $L = O(\log n)$, the full butterfly matrix decomposition can be represented in $O(n \log n)$ memory. We present a brief overview of algorithms to compute the butterfly decomposition below.

3.2.1 | Computation of Butterfly Decomposition

When any entry of $\boldsymbol{T}$ can be sampled in $\mathcal{O}(1)$, a butterfly decomposition can be efficiently approximated using SVD [24] in $\mathcal{O}(n^{1.5})$, or using Chebyshev interpolation [88, 89] or proxy points [26, 33, 90] in $\mathcal{O}(n \log n)$ complexity. When $\boldsymbol{T}$ can be applied to arbitrary random vectors in $\mathcal{O}(n \log n)$ complexity, a butterfly decomposition can be approximated using sketching algorithms [25] in $\mathcal{O}(n^{1.5} \log n)$ complexity. Note that none of these methods can be applied when only a fixed subset of entries is given (i.e., the matrix completion setting). In this setting, numerical optimization algorithms must be used and are described in Section 5.

3.3 | Butterfly Decomposition in Machine Learning

Due to its ability to approximate Fourier integral operators with a small number of parameters, butterfly decomposition has recently been integrated into deep neural networks to efficiently learn structured linear transforms [40, 91]. Building on this, the Kaleidoscope hierarchy, which is a product of learnable orthogonal butterfly matrices, was introduced into the Mel-Frequency Spectral Coefficients (MFSC) pipeline [92], achieving performance comparable to hand-crafted transforms. These Kaleidoscope matrices are capable of learning latent permutations and accelerating inference [41]. To further enhance their practicality and compatibility with GPU architectures, the sparsity structure of the butterfly factors was modified in various works [39, 42, 47]. Empirical evaluations in these works show that restructuring maintains similar accuracy while improving efficiency by enhancing the ease of implementation.

Integrating butterfly matrices into machine learning frameworks like PyTorch has led to new developments in neural network architectures and computational kernels. Although butterfly decompositions theoretically enable matrix-vector products in $O(n \log n)$ complexity, achieving these speedups in practice poses significant challenges. Traditional block-sparse representations often lead to inefficient high-level implementations, as they require manual looping over sparse blocks, which impedes performance. Realizing theoretical speedups for both the forward pass, that is, applying the butterfly matrix, and backward pass, which is obtaining the gradients of block sparse matrices, demands carefully optimized software kernels. While recent efforts have developed efficient GPU kernels for the forward pass [43, 44], kernels for computing gradients (i.e., the backward pass) remain underdeveloped. To address this limitation, some works have modified the sparsity structure of the butterfly factors to better align with hardware efficiency [42]. In contrast, we propose a new representation of the butterfly decomposition as a tensor network, enabling differentiation with respect to the factors by leveraging automatic differentiation with tensor contractions. This reformulation is detailed in the next section.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

4 | Tensor Network for Butterfly Decomposition

In this section, we propose a reformulation that recasts the butterfly decomposition as a tensor decomposition. Each butterfly factor is modeled as a dense tensor, and the product of block sparse matrices in the butterfly decomposition is expressed through tensor contractions. These contractions are naturally supported by modern auto-differentiation frameworks via einsum-like primitives, eliminating the need for hand-crafted software kernels.

4.1 | Butterfly Decomposition Rewritten as a Tensor Network

To describe the tensor decomposition reformulation, we will continue to use the same assumptions as in the previous section. We consider an $m \times n$ input matrix $\boldsymbol{T}$, where $n = c \cdot 2^L$, and L is the number of levels in the butterfly decomposition. The row and column sets, $\chi = \{0, \dots, m-1\}$ for rows and $\xi = \{0, \dots, n-1\}$ for columns of the matrix, and $m = n$. These sets are organized into hierarchical trees $\mathcal{T}_\chi$ and $\mathcal{T}_\xi$ respectively as described in Section 3. τ^l denotes the set of all nodes at level l in tree $\mathcal{T}_\chi$ and v^l denotes the set of all nodes at level l in tree $\mathcal{T}_\xi$. Additionally, we assume that for each level $l \in \{0, \dots, L\}$, the ranks of all the blocks are equal, that is, $\forall i, j,\ r^l_{i,j} = R_l$. For the ease of later complexity analysis, we further assume $R_l = r$.

We reshape the matrix $\boldsymbol{T}$, indexed by (i, j)–where i denotes the row and j denotes the column–into a tensor. The tensor is formed by associating an $(L+1)$-tuple to each of i and j, based on the nodes τ^l_i and v^l_j that contain i and j respectively, at each level l. For generating the tuple corresponding to rows, let τ^0_0 be the root node ρ. For each level l from 0 to $L-1$:

- Assign $i_l = 0$ if i is in the left child of ρ else assign $i_l = 1$.
- Update the root ρ to be its child where i is found.

This recursive process generates a binary tuple $(i_0, \dots, i_{L-1})$ for each index i. The last index of the tuple, i_L, is generated by finding the position out of $\{0, \dots, c-1\}$ of i at the leaf level. The reshaping of index i into the tuple $(i_0, \dots, i_L)$ is formally defined by

$$i_l = \begin{cases} i \bmod c, & \text{if } l = L, \\ \left\lfloor \frac{i}{c \cdot 2^{L-l-1}} \right\rfloor \bmod 2, & \text{if } l = 0, \dots, L-1. \end{cases} \tag{4.1}$$

A transformation to the index $j \to (j_0, \dots, j_L)$ is defined analogously. This leads to reshaping of the matrix $\boldsymbol{T}$ into an order $2L+2$ tensor, $\boldsymbol{\mathcal{T}}$, by representing each entry of the matrix (i, j) by a $(2L+2)$– tuple $(\underbrace{i_0, \dots, i_L}_{i}, \underbrace{j_0, \dots, j_L}_{j})$.

With this approach, each of the block sparse butterfly factor matrices, $\boldsymbol{S}^l\ \forall l \in \{1, \dots, L+2\}$, given in (3.8) are transformed into higher-order dense tensors $\boldsymbol{\mathcal{S}}^l$. This is done by stacking up the non-zero blocks of the block-sparse matrices in $\boldsymbol{S}^l$ in (3.8) as slices of a higher-order tensor $\boldsymbol{\mathcal{S}}^l$.

To define this transformation, we define a function ψ. When given an integer $i \in \{0, \dots, 2^l - 1\}$, it outputs a binary tuple as follows,

$$\psi(i, l) = \left(b_0, b_1, \dots, b_{l-1}\right) \in \{0, 1\}^l, \tag{4.2}$$

where each $b_m \in \{0, 1\}$ is given by

$$b_m = \left\lfloor \frac{i}{2^m} \right\rfloor \bmod 2, \quad \text{for } m = 0, 1, \dots, l-1. \tag{4.3}$$

We can describe the transformation for the butterfly factor matrices $\boldsymbol{S}^l$ to tensors $\left\{\boldsymbol{\mathcal{S}}^l : l \in \{1, \dots, L+2\}\right\}$, using (4.1), (4.2), and (4.3) with $\boldsymbol{\mathcal{S}}^1$ and $\boldsymbol{\mathcal{S}}^{L+2}$ being order $L+2$ and $\left\{\boldsymbol{\mathcal{S}}^l : l \in \{2, \dots L+1\}\right\}$ being order $L+3$ tensors. The slices of the higher-order tensors are indexed as follows

$$\forall i,\ \boldsymbol{\mathcal{S}}^1(i_0, \dots, i_{L-1}, :, :) = \boldsymbol{U}^0_{i,0}, \quad (i_0, \dots, i_{L-1}) = \psi(i, L),$$
$$\forall i, j,\ \boldsymbol{\mathcal{S}}^{m+1}(i_0 \dots i_{L-m}, j_0, \dots, j_L, :, :) = \boldsymbol{P}^m_{i,j},$$

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

$$(i_0, \dots, i_{L-m}) = \psi(i, L-m), \quad (j_0, \dots j_m) = \psi(j, m), \quad \forall m \in \{1, \dots, M\}, \tag{4.4}$$

where $\boldsymbol{U}^L_{i,0}$ are the column basis in (3.5), and $\boldsymbol{P}^m_{i,j}$ are column transfer matrices in (3.6). The tensors $\left\{\boldsymbol{S}^l : l \in \{M+2, \dots, L+2\}\right\}$, are defined analogously.

Using the above assignment, each entry of the matrix $\boldsymbol{T}$ in (3.7) is mapped to a tensor $\boldsymbol{\mathcal{T}}$. The input tensor $\boldsymbol{\mathcal{T}}$ is approximated by a butterfly matrix $\boldsymbol{K}$, which is reshaped into $\boldsymbol{\mathcal{K}}$, and is represented by the tensor network with parameters $\mathcal{S} = \left\{\boldsymbol{S}^l : l \in \{1, \dots, L+2\}\right\}$. Using the transformations above, each $c \times c$ block in (3.7) is

$$\begin{aligned}\boldsymbol{K}(\psi(i, L), :, \psi(j, L), :) = \boldsymbol{S}^1(\psi_L(i, L), :, :)\left(\prod_{m=1}^{M} \boldsymbol{S}^{m+1}(\psi(i, L-1), \psi(j, m-1), :, :)\right) \\ \left(\prod_{l=M+1}^{L} \boldsymbol{S}^{l+1}(\psi_L(i, l-1), \psi_L(j, L-l), :, :)\right)\boldsymbol{S}^{L+2}(\psi_L(j, L), :, :),\end{aligned} \tag{4.5}$$

where for convenience, we slightly abuse notation to write

$$\boldsymbol{K}(i_0, \dots, i_{L-1}, :, j_0, \dots, j_{L-1}, :) = \boldsymbol{K}(\psi(i, L), :, \psi(j, L), :) \tag{4.6}$$

and similarly for the parameter tensors. This leads to the following elementwise expression,

$$\begin{aligned} t_{i_0,\dots,i_L,j_0\dots j_L} &\approx k_{i_0,\dots,i_L,j_0\dots j_L} \\ &= \sum_{r_1,\dots,r_{L+1}} s^1_{i_0\dots i_L r_1}\left(\prod_{m=1}^{L} s^{m+1}_{i_0,\dots,i_{L-m},j_0,\dots,j_{m-1},r_m,r_{m+1}}\right) s^{L+2}_{j_0\dots j_{L-1} r_{L+1} j_L}. \end{aligned} \tag{4.7}$$

As an example, Figure 2a shows a 2-level butterfly matrix decomposition $\boldsymbol{X} = \boldsymbol{S}^1\boldsymbol{S}^2\boldsymbol{S}^3\boldsymbol{S}^4$ and how an entry, (i, j), of the butterfly matrix is generated as highlighted in blue. Figure 2b shows the tensor diagram of the tensor network representation of the butterfly matrix decomposition in Figure 2a. Each gray circle represents the order $L+2$ core tensors $\boldsymbol{S}^1$ and $\boldsymbol{S}^4$, or the order $L+3$ core tensors $\boldsymbol{S}^2$ and $\boldsymbol{S}^3$. Each black dot represents an *identity node*, where all edges connected to it represent the same (external) edge.

For notational convenience, we rewrite (4.7) as

$$k_{i_0,\dots,i_L,j_0\dots j_L} = \sum_{r_1,\dots,r_{L+1}} \left(\prod_{m=0}^{L+1} s^{m+1}_{i_0,\dots,i_{L-m},j_0,\dots,j_{m-1},r_m,r_{m+1}}\right). \tag{4.8}$$

Here we define $r_{L+2} = j_L$, $r_0 = i_L$, and extend the $L+2$ order tensors $\boldsymbol{S}^1$ and $\boldsymbol{S}^{L+2}$ to $L+3$ order tensors by defining $i_{-1} = 1$ and $j_{-1} = 1$.

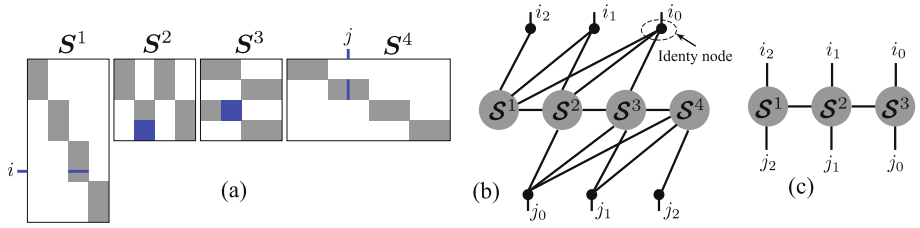


FIGURE 2 | (a) A 2-level butterfly matrix decomposition $\boldsymbol{X} = \boldsymbol{S}^1\boldsymbol{S}^2\boldsymbol{S}^3\boldsymbol{S}^4$. For a matrix entry (i, j) with $i = 2c + a$ and $j = c + b$, $a, b \le c$, x_{ij} is computed by the product of the subblocks highlighted in blue. (b) The diagram of reshaping the 2-level butterfly matrix decomposition $\boldsymbol{X}$ as an order $2L+2$ tensor $\boldsymbol{\mathcal{X}}$ with tensor cores $\boldsymbol{S}^1, \boldsymbol{S}^2, \cdots, \boldsymbol{S}^{L+1}, \boldsymbol{S}^{L+2}$. Each gray circle represents one tensor core $\boldsymbol{S}^l$ of order $L+3$ for $1 < l < L+2$ or order $L+2$ otherwise. Each black node denotes an identity node, representing an external edge i_0, i_1, i_2 or j_0, j_1, j_2. The (i, j) entry in (a) now becomes $(i_0, i_1, i_2, j_0, j_1, j_2) = (1, 0, a, 0, 1, b)$. (c) The diagram of the QTT representation of the same tensor in (b): $\boldsymbol{\mathcal{X}}$ with tensor cores $\boldsymbol{S}^1, \boldsymbol{S}^2, \boldsymbol{S}^3$ where each tensor core has order 3 or 4, independent of L.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

It is worth mentioning that the reformulation of the butterfly decomposition described in (3.7) as a tensor network representation described in (4.7) is a valid tensor network only when the ranks of all the blocks at each level are the same, and the size of the leaf nodes is also the same. This is due to the fact that only in this case, we are able to stack the dense sub-blocks in butterfly factors and represent them as tensors as in (4.7). Also, this tensor network reformulation is not to be confused with approaches that leverage tensor low-rank compression algorithms (e.g., Tucker decomposition) to construct tensorized butterfly algorithms (see [27] for an example), which leads to a different tensor diagram compared with Figure 2b.

4.2 | Tensor Reformulation of Operations on Butterfly Matrices

Once the butterfly decomposition is represented as a tensor network, the operations involved with butterfly matrices can be expressed using einsum-like primitives. This enables efficient and portable implementation on heterogeneous architectures by leveraging existing tensor contraction libraries. For example, the $O(n \log n)$ complexity of applying a butterfly matrix $\boldsymbol{X}(\mathcal{S})$ to a vector $\boldsymbol{v}$ is realized by first reshaping the matrix and vector as described in (4.1). The matrix-vector product $\boldsymbol{u} = \boldsymbol{X}(\mathcal{S})\boldsymbol{v}$ can then be computed by performing the following dense tensor contractions,

$$\begin{aligned} u_{i_0 \dots i_L} &= \sum_{j_0 \dots j_L} x(\mathcal{S})_{i_0 \dots i_L j_0 \dots j_L} v_{j_0 \dots j_L}, \\ &= \sum_{r_1, \dots r_{L+1}} \sum_{j_0 \dots j_L} \left(\prod_{m=0}^{L+1} s^{m+1}_{i_0 \dots i_{L-m} j_0 \dots j_{m-1} r_m r_{m+1}} \right). \end{aligned} \tag{4.9}$$

Some algorithms require gradients of the objective function with respect to the butterfly factors for learning with butterfly matrices. The reformulation of butterfly decomposition as a tensor decomposition enables easy and efficient computation of the gradients via tensor contractions. We derive the gradients of butterfly factors with respect to an arbitrary objective function below.

Consider an objective function $\phi(\boldsymbol{X}(\mathcal{S}))$, where $\boldsymbol{X}(\mathcal{S})$ is a butterfly matrix with factors $\mathcal{S}$. Expressions for gradients with respect to the factors can be derived by reshaping the butterfly matrix and other parameters involved as in (4.1). The Gradient with respect to the resulting tensorized factor $\boldsymbol{\mathcal{S}}^1$ is computed via chain rule and is given by

$$\begin{aligned} \frac{\partial \phi}{\partial s^1_{i_0 \dots i_L r_1}} &= \sum_{j_0, \dots, j_L} z_{i_0 \dots i_L j_0 \dots j_L} \sum_{r_2, \dots, r_{L+1}} \left(\prod_{m=1}^{L+1} s^{m+1}_{i_0 \dots i_{L-m} j_0 \dots j_{m-1} r_m r_{m+1}} \right), \\ z_{i_0 \dots i_L j_0 \dots j_L} &= \frac{\partial \phi}{\partial x_{i_0 \dots i_L j_0 \dots j_L}}. \end{aligned} \tag{4.10}$$

Similarly, the gradient with respect to $\boldsymbol{\mathcal{S}}^{L+2}$ is

$$\frac{\partial \phi}{\partial s^{L+2}_{j_0 \dots j_{L-1} r_{L+1} j_L}} = \sum_{i_0, \dots, i_L} z_{i_0 \dots i_L j_0 \dots j_L} \sum_{r_1, \dots, r_L} \left(\prod_{m=0}^{L} s^{m+1}_{i_0 \dots i_{L-m} j_0 \dots j_{m-1} r_m r_{m+1}} \right), \tag{4.11}$$

and gradients with respect to $\boldsymbol{\mathcal{S}}^{l+1}, l \in \{1, \dots, L\}$,

$$\frac{\partial \phi}{\partial s^{l+1}_{i_0 \dots i_{L-l} j_0 \dots j_{l-1} r_l r_{l+1}}} = \sum_{\substack{i_{L-l+1}, \dots, i_L \\ j_l, \dots, j_L}} z_{i_0 \dots i_L j_0 \dots j_L} \sum_{\substack{r_1, \dots, r_{l-1} \\ r_{l+2} \cdots r_{L+1}}} \left(\prod_{\substack{m=0 \\ m \neq l}}^{L+1} s^{m+1}_{i_0 \dots i_{L-m} j_0 \dots j_{m-1} r_m r_{m+1}} \right). \tag{4.12}$$

These expressions can be used to perform a backward pass for learning with butterfly matrices in neural networks using dense tensor contractions or to obtain gradients for numerical optimization algorithms for tensor completion in the butterfly matrix format using sparse tensor contractions which is detailed in the next section.

5 | Tensor Completion with Butterfly Decomposition

Many applications require computation of a butterfly decomposition when only a fixed subset of matrix entries or indices is given. This computation is formulated as a tensor completion problem. Let $\boldsymbol{T} \in \mathbb{C}^{n \times n}$, $n = c \cdot 2^L$, be the input matrix, with observed entries set $\overline{\Omega}$, then the objective function is given as,

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

$$\phi(\overline{S}) = \frac{1}{2}\left\|\mathcal{P}_\Omega(\boldsymbol{T} - \boldsymbol{X}(\overline{S})\right\|_F^2, \tag{5.1}$$

where $\boldsymbol{X}(\overline{S})$ is a butterfly matrix with the butterfly factors $\overline{S}$ as a set of block sparse matrices as given in (3.8). Since the numerical optimization of this objective function would require computations with block sparse matrices which require complicated software kernels, we leverage the reformulation of butterfly matrix as a tensor network introduced in Section 4. We transform the input matrix $\boldsymbol{T}$ into an order $2L+2$ tensor, $\boldsymbol{\mathcal{T}} \in \mathbb{C}^{2\times\cdots\times c\times 2\times\cdots\times c}$ by using the transformation of each index $(i,j) \in \overline{\Omega}$ into a tuple $(i_0,\dots,i_L,j_0,\dots j_L) \in \Omega$ according to (4.1). This transformation results in the corresponding tensor completion problem, with objective,

$$\phi(S) = \frac{1}{2}\left\|\mathcal{P}_\Omega(\boldsymbol{\mathcal{T}} - \boldsymbol{\mathcal{X}}(S))\right\|_F^2, \tag{5.2}$$

where $\boldsymbol{\mathcal{X}}(S)$ is the reconstructed tensor. The reconstructed tensor is described by the tensorized butterfly decomposition dense tensor parameters, S, as described in (4.7).

Tensor completion problem is a well-studied problem from numerical optimization perspective with many existing algorithms such as alternating least squares (ALS) [10, 93], adaptive moment estimation (ADAM) optimization [17], alternating direction fitting (ADF) [18], and Riemannian optimization [19]. Next, we present the implementation and analyze the computational complexity of the proposed butterfly completion algorithms using ALS, ADAM and ADF in Section 5.1. Several practical considerations required to implement these algorithms are discussed in Section 5.2.

5.1 | Optimization for Tensor Completion of Butterfly Matrices

The proposed ALS, ADAM and ADF optimization methods for butterfly completion are described in this subsection. We focus on the explanation of the ALS algorithm and its complexity analysis, and briefly summarize the ADAM and ADF algorithms as many computational kernels are shared between them.

5.1.1 | Alternating Least Squares (ALS)

ALS has proven to be a competitive method for tensor completion problems for various tensor decompositions [10]. The proposed ALS-based butterfly completion algorithm proceeds by minimizing one factor at a time, summarized in Algorithm 1. For minimizing factors of the tensorized butterfly, the resulting least squares to solve for the first parameter tensor $\boldsymbol{\mathcal{S}}^1$ is given for each $(i_0,\dots,i_L)$ and $(j_0,\dots,j_L) \in \Omega_{i_0,\dots,i_L}$, as

$$\begin{aligned}
\sum_{r_1} s^1_{i_0\dots i_L r_1} a_{i_0\dots i_{L-1} j_0\dots j_L r_1} &= t_{i_0\dots i_L j_0\dots j_L},\\
a_{i_0\dots i_{L-1} j_0\dots j_L r_1} &= \sum_{r_2,\dots,r_{L+1}} \left(\prod_{m=1}^{L+1} s^{m+1}_{i_0\dots i_{L-m} j_0\dots j_{m-1} r_m r_{m+1}}\right),
\end{aligned} \tag{5.3}$$

Similarly, the least squares to solve for the last parameter tensor, $\boldsymbol{\mathcal{S}}^{L+2}$, is given for each $(j_0,\dots,j_L)$ and $(i_0,\dots,i_L) \in \Omega_{j_0,\dots,j_L}$, as

$$\begin{aligned}
\sum_{r_{L+1}} s^{L+2}_{j_0\dots j_L r_{L+1}} a_{i_0\dots i_L j_0\dots j_{L-1} r_{L+1}} &= t_{i_0\dots i_L j_0\dots j_L},\\
a_{i_0\dots i_L j_0\dots j_{L-1} r_{L+1}} &= \sum_{r_1,\dots,r_L} \left(\prod_{m=0}^{L} s^{m+1}_{i_0\dots i_{L-m} j_0\dots j_{m-1} r_m r_{m+1}}\right).
\end{aligned} \tag{5.4}$$

The least squares to solve for the remaining parameters $\boldsymbol{\mathcal{S}}^{l+1}$, $l \in \{1,\dots,L\}$, for each $(i_0,\dots,i_{L-l},j_0,\dots,j_{l-1})$ and $(i_{L-l+1},\dots,i_L,j_l,\dots,j_L) \in \Omega_{i_0,\dots,i_{L-l},j_0,\dots,j_{l-1}}$, is given as

$$\begin{aligned}
\sum_{r_l,r_{l+1}} s^{l+1}_{i_0\dots i_{L-l} j_0\dots j_{l-1} r_l r_{l+1}} a_{i_0\dots i_L j_0\dots j_L r_l r_{l+1}} &= t_{i_0\dots i_L j_0\dots j_L},\\
a_{i_0\dots i_L j_0\dots j_L r_l r_{l+1}} &= \sum_{\substack{r_1,\dots,r_{l-1}\\ r_{l+2}\dots,r_{L+1}}} \left(\prod_{\substack{m=0\\ m\neq l}}^{L+1} s^{m+1}_{i_0\dots i_{L-m} j_0\dots j_{m-1} r_m r_{m+1}}\right).
\end{aligned} \tag{5.5}$$

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

ALGORITHM 1 | ALS for butterfly completion.

1: **Input:** Observed matrix $\boldsymbol{T}_\Omega$ (or its tensorization), rank r, initial guess $\boldsymbol{\mathcal{S}}^i, i = 1, \dots, L+2$, max iterations $t_{\max}$, and convergence threshold ϵ.
2: **Output:** Updated butterfly cores $\boldsymbol{\mathcal{S}}^i, i = 1, \dots, L+2$.
3: **for** $t = 1$ to $t_{\max}$ and not converged **do**
4: **for** $l = 1$ to $L+2$ **do**
5: **if** $l = 0$ **then**
6: **for** each $(i_0, \dots, i_L)$ **do** ▷ one row of a frontal slice of $\boldsymbol{\mathcal{S}}^1$
7: Initialize $\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{K} = \boldsymbol{0}$
8: **for** each $(j_0, \dots, j_L) \in \Omega_{i_0,\dots,i_L}$ **do**
9: $\boldsymbol{v} = \left(\prod_{m=1}^{L} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right) \boldsymbol{s}^{L+2}(j_0, \dots, j_L, :)$
10: Update $\boldsymbol{y}$ += $t_{i_0 \dots i_L j_0 \dots j_L} \boldsymbol{v}$
11: Update $\boldsymbol{K}$ += $\boldsymbol{v}\boldsymbol{v}^T$
12: **end for**
13: Solve $\boldsymbol{K}\boldsymbol{s} = \boldsymbol{y}$ and update $\boldsymbol{s}^1(i_0, \dots, i_L, :) \leftarrow \boldsymbol{s}$
14: **end for**
15: **else if** $l = L+2$ **then**
16: **for** each $(j_0, \dots, j_L)$ **do** ▷ one column of a frontal slice of $\boldsymbol{\mathcal{S}}^{L+2}$
17: Initialize $\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{K} = \boldsymbol{0}$
18: **for** each $(i_0, \dots, i_L) \in \Omega_{j_0,\dots,j_L}$ **do**
19: $\boldsymbol{v} = \boldsymbol{s}^1(i_0, \dots, i_L, :)^T \left(\prod_{m=1}^{L} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right)$
20: Update $\boldsymbol{y}$ += $t_{i_0 \dots i_L j_0 \dots j_L} \boldsymbol{v}$
21: Update $\boldsymbol{K}$ += $\boldsymbol{v}\boldsymbol{v}^T$
22: **end for**
23: Solve $\boldsymbol{K}\boldsymbol{s} = \boldsymbol{y}$ and update $\boldsymbol{s}^{L+2}(j_0, \dots, j_L, :) \leftarrow \boldsymbol{s}$
24: **end for**
25: **else**
26: **for** each $(i_0, \dots, i_{L-l}, j_0, \dots, j_{l-1})$ **do** ▷ one frontal slice of $\boldsymbol{\mathcal{S}}^{l+1}$
27: Initialize $\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{K} = \boldsymbol{0}$
28: **for** each $(i_{L-l+1}, \dots, i_L, j_l, \dots, j_L) \in \Omega_{i_0,\dots,i_{L-l},j_0,\dots,j_{l-1}}$ **do**
29: $\boldsymbol{u} = \boldsymbol{s}^1(i_0, \dots, i_L, :)^T \left(\prod_{m=1}^{l-1} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right)$
30: $\boldsymbol{v} = \left(\prod_{m=l+1}^{L} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right) \boldsymbol{s}^{L+2}(j_0, \dots, j_L, :)$
31: Update $\boldsymbol{y}$ += $t_{i_0 \dots i_L j_0 \dots j_L} (\boldsymbol{v} \otimes \boldsymbol{u})$
32: Update $\boldsymbol{K}$ += $(\boldsymbol{v} \otimes \boldsymbol{u})(\boldsymbol{v} \otimes \boldsymbol{u})^T$
33: **end for**
34: Solve $\boldsymbol{K}\boldsymbol{s} = \boldsymbol{y}$and update $\boldsymbol{\mathcal{S}}^{l+1}(i_0, \dots, i_{L-l}, j_0, \dots, j_{l-1}, :, :) := \text{reshape}(\boldsymbol{s}, r, r)$
35: **end for**
36: **end if**
37: **end for**
38: **end for**

All of the above least squares problems can be solved by solving for fibers of $\boldsymbol{\mathcal{S}}^{(1)}$ and $\boldsymbol{\mathcal{S}}^{L+2}$, and slices of $\boldsymbol{\mathcal{S}}^l, l \in \{2, \dots, L+1\}$ in parallel, similar to tensor train completion [18]. We solve the least squares equations by using normal equations that result in small symmetric positive semi-definite systems of size $r \times r$ for $\boldsymbol{\mathcal{S}}^1$ and $\boldsymbol{\mathcal{S}}^{L+2}$ or $r^2 \times r^2$ for $\boldsymbol{\mathcal{S}}^l, \forall l \in \{2, \dots, L+1\}$, that can be solved in parallel efficiently [82, 94]. We summarize the normal equations method in Algorithm 1. This algorithm proceeds by iterating over each fiber of $\boldsymbol{\mathcal{S}}^1$ and $\boldsymbol{\mathcal{S}}^{L+2}$ and slice of $\boldsymbol{\mathcal{S}}^l, l \in \{2, \dots, L+1\}$. This corresponds to lines 16, 6, and 26 and this loop can be parallelized as there is no dependency. These steps and their complexities are explained as follows.

When solving for $\boldsymbol{\mathcal{S}}^1$ and $\boldsymbol{\mathcal{S}}^{L+2}$, the algorithm then iterates over each observed entry in Ω and computes the corresponding fibers of the system tensor $\boldsymbol{\mathcal{A}}$ in a vector $\boldsymbol{v}$ of size r by computing L matrix-vector products with matrices of size $r \times r$ and vectors of size r, computed in lines 9 and 19 with cost $\mathcal{O}(Lr^2)$. The right-hand side for each fiber is formed in vector $\boldsymbol{y}$ by

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

accumulating scalar products of $\boldsymbol{v}$ with corresponding input tensor entries $t_{i_0 \dots i_L j_0 \dots j_L}$ in lines 10 and 20 with cost $\mathcal{O}(r)$. The semi-definite systems $\boldsymbol{K}$ are computed by accumulating the outer product of vectors $\boldsymbol{v}$ with themselves in lines 11 and 21 with cost $\mathcal{O}(r^2)$. Lines 9, 10, 11, 19, 20, and 21 are executed for each nonzero; therefore, the cost of these operations is $\mathcal{O}(|\Omega| L r^2)$. A positive semi-definite (PSD) solve is performed in lines 13 and 23 to form the updated fiber with cost $\mathcal{O}(r^3)$. In practice, a regularization parameter is used to ensure the positive definiteness of the system. Since lines 13 and 23 are executed for all the fibers, the total cost is $\mathcal{O}(nr^3)$.

When solving for $\boldsymbol{S}^l, l \in \{2, \dots, L+1\}$, the algorithm then iterates over each observed entry in Ω and computes the corresponding fibers of the system tensor $\boldsymbol{\mathcal{A}}$ that are a Kronecker product of two vectors $\boldsymbol{v}$ and $\boldsymbol{u}$ formed in lines 29 and 30 by performing at most L matrix-vector products, each with cost $\mathcal{O}(r^2)$. Similar to the above case, right-hand sides in line 31 and left-hand sides in line 32 are formed by accumulating scalar and outer products of vectors, but of size $\mathcal{O}(r^2)$; each with a cost $\mathcal{O}(r^2)$ and $\mathcal{O}(r^4)$, respectively. Each slice is then updated by performing a PSD solve in line 34, each with a cost $\mathcal{O}(r^6)$ and reshaping the result into a matrix in row-major ordering. The resulting total cost for updating all the slices is $\mathcal{O}\big(|\Omega|(Lr^2 + r^4) + nr^6\big)$. Since the solves are performed for each factor, substituting $L = \mathcal{O}(\log n)$, the total cost for one iteration of ALS is $\mathcal{O}\big(|\Omega|(r^2 \log^2 n + r^4 \log n) + nr^6 \log n\big)$.

The ALS algorithm runs until maximum number of iterations are reached or convergence is achieved. The convergence can be defined by (1) measuring the norm of the difference of factor tensors between iterations, (2) computing the gradient of objective function $\phi(\mathcal{S})$ in (5.2) or (3) comparing the relative difference of the reconstruction and ground truth using a tolerance ϵ defined as

$$\left\| \mathcal{P}_\Omega(\boldsymbol{\mathcal{T}} - \boldsymbol{\mathcal{X}}(\mathcal{S})) \right\|_F \Big/ \left\| \mathcal{P}_\Omega(\boldsymbol{\mathcal{T}}) \right\|_F < \epsilon. \tag{5.6}$$

Unless otherwise stated, we use (5.6) as the convergence metric in all the completion algorithms under consideration.

Although not detailed here, we remark that many software kernels of Algorithm 1 can be reused to develop a QTT completion algorithm. The QTT representation of the same $\boldsymbol{T}$ matrix (and its tensorization $\boldsymbol{\mathcal{T}}$) as (4.8) is defined as

$$t_{i_0,\dots,i_L,j_0\dots j_L} = \sum_{r_1,\dots,r_L} \left(\prod_{m=0}^{L} s^{m+1}_{i_m,j_m,r_m,r_{m+1}} \right). \tag{5.7}$$

Comparing with (4.8), QTT can be viewed as a more aggressive compression format than butterfly, consisting of order 4 tensor cores instead of butterfly's order $L+3$ tensor cores. The diagrams of butterfly and QTT of the same tensor with $L = 3$ are shown in Figure 2b and c. The QTT completion algorithm (based on extension of the tensor train completion in [18] will be used as one of our reference algorithms in numerical experiments of Section 6 and is discussed in Algorithm 5. We also remark that Algorithm 1 reduces to low-rank matrix completion algorithm when $L = 0$, which is used as one of our reference algorithms in numerical experiments, as well as the initial guess to the butterfly completion algorithms (see Section 5.2).

5.1.2 | ADAM Optimization

Adaptive moment estimation (ADAM) [17] is a first-order optimization algorithm that combines momentum and adaptive learning rates, making it effective for training deep models and non-convex optimization tasks. ADAM is used in tensor completion problems to efficiently update the factors by obtaining gradients via sparse tensor contractions. The proposed ADAM-based butterfly completion algorithm is described in Algorithm 3. For the factors in tensorized butterfly, the gradients are computed by computing the sparse tensor contractions in Equations (4.10), (4.12), and (4.11), with $\mathcal{Z} = \mathcal{P}_\Omega(\boldsymbol{\mathcal{X}}(\mathcal{S}) - \boldsymbol{\mathcal{T}})$. The cost of computing the gradients is $O(|\Omega| L r^2)$ and a detailed description of the sparse tensor contractions for gradients is summarized in Algorithm 3. Once gradients are computed, updates can be computed via the ADAM optimization method as summarized in Algorithm 2. ADAM uses scalar operations on parameters with the total cost of single iteration being $O(|\Omega| L^2 r^2 + Lnr^2)$. Replacing $L = O(\log n)$ in the previous expression, the total cost of a single ADAM iteration is $O(|\Omega| r^2 \log^2 n + nr^2 \log n)$. Comparing with the total cost of a single ALS iteration, ADAM has an $\mathcal{O}(r^2)$ reduction for the leading order term. That said, ADAM oftentimes requires a much larger number of iterations to converge compared with ALS. Just like ALS, we also implement a QTT completion algorithm with ADAM optimization with the same computational kernels as used for butterfly completion.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

ALGORITHM 2 | ADAM_Update($\mathcal{S}, \mathcal{G}, \mathcal{M}, \mathcal{V}, \alpha, \beta_1, \beta_2, \epsilon, t$).

Input: A tensor (or array) of parameters $\mathcal{S}$, its gradient $\mathcal{G}$, and the current first and second moment estimates $\mathcal{M}$ and $\mathcal{V}$, learning rate α, decay rates for moments β_1, β_2, small constant σ, current iteration t.
Output: Updated parameters $\mathcal{S}$, first and second moment estimates $\mathcal{M}$ and $\mathcal{V}$
for each entry (s, g, m, v) in $(\mathcal{S}, \mathcal{G}, \mathcal{M}, \mathcal{V})$ **do**
    $m \leftarrow \beta_1 m + (1-\beta_1)g$ ▷ Update biased first moment estimate
    $v \leftarrow \beta_2 v + (1-\beta_2)(g * g)$ ▷ Update biased second moment estimate
    $\hat{m} \leftarrow \frac{m}{1-\beta_1^t}, \quad \hat{v} \leftarrow \frac{v}{1-\beta_2^t}$ ▷ Update bias-corrected estimates
    $s \leftarrow s - \frac{\alpha \hat{m}}{\sqrt{\hat{v}}+\sigma}$
end for

ALGORITHM 3 | ADAM for butterfly completion.

Input: Observed matrix $\boldsymbol{T}_\Omega$ (or its tensorization), rank r, initial guess $\boldsymbol{\mathcal{S}}^i$, $i = 1, \dots, L+2$,, max iterations $t_{\max}$, learning rate α, decay rates for moments β_1, β_2, small constant σ, and convergence threshold ϵ.
Output: Updated butterfly cores $\boldsymbol{\mathcal{S}}^i$, $i = 1, \dots, L+2$.
Initialize gradient tensors, first and second moment estimate tensors, $\boldsymbol{\mathcal{G}}^l = 0 \in \mathcal{G}$, $\boldsymbol{\mathcal{M}}^l = 0 \in \mathcal{M}$, and $\boldsymbol{\mathcal{V}}^l = 0 \in \mathcal{V}$, $l = 1, \dots, L+2$
for $t = 1$ to $t_{\max}$ and not converged **do**
    for $l = 1$ to $L+2$ **do**
        Initialize $\boldsymbol{\mathcal{G}}^l = 0$
        if $l = 0$ **then**
            for each $(i_0, \dots, i_L) \in \Omega$ **do** ▷ one row of a frontal slice of $\boldsymbol{\mathcal{S}}^1$
                for each $(j_0, \dots, j_L) \in \Omega_{i_0 \dots i_L}$ **do**
                    Compute $\boldsymbol{g}^1(i_0, \dots, i_L, :) += (x_{i_0 \dots i_L j_0 \dots j_L} - t_{i_0 \dots i_L j_0 \dots j_L})\boldsymbol{v}$
                    where $\boldsymbol{v} = \left(\prod_{m=1}^{L} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right) \boldsymbol{s}^{L+2}(j_0, \dots, j_L, :)$
                end for
            end for
        else if $l = L+2$ **then**
            for each $(j_0, \dots, j_L) \in \Omega$ **do** ▷ one column of a frontal slice of $\boldsymbol{\mathcal{S}}^{L+2}$
                for each $(i_0, \dots, i_L) \in \Omega_{j_0 \dots j_L}$ **do**
                    Compute $\boldsymbol{g}^{L+2}(j_0, \dots, j_L, :) += (x_{i_0 \dots i_L j_0 \dots j_L} - t_{i_0 \dots i_L j_0 \dots j_L})\boldsymbol{u}$
                    where $\boldsymbol{u} = \boldsymbol{s}^1(i_0, \dots, i_L, :)^T \left(\prod_{m=1}^{L} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right)$
                end for
            end for
        else
            for each $(i_0, \dots, i_{L-l}, j_0, \dots, j_{l-1}) \in \Omega$ **do** ▷ one frontal slice of $\boldsymbol{\mathcal{S}}^{l+1}$
                for each $(i_{L-l+1}, \dots i_L, j_l, \dots, j_L) \in \Omega_{i_0 \dots i_{L-l} j_0 \dots j_{l-1}}$ **do**
                    Compute $\boldsymbol{g}^{l+1}(i_0, \dots, i_{L-l}, j_0, \dots, j_{l-1}, :, :) += (x_{i_0 \dots i_L j_0 \dots j_L} - t_{i_0 \dots i_L j_0 \dots j_L})\boldsymbol{v} \otimes \boldsymbol{u}$
                    where $\boldsymbol{u} = \boldsymbol{s}^1(i_0, \dots, i_L, :)^T \left(\prod_{m=1}^{l-1} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right)$

                    $\boldsymbol{v} = \left(\prod_{m=l+1}^{L} \boldsymbol{\mathcal{S}}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :)\right) \boldsymbol{s}^{L+2}(j_0, \dots, j_L, :)$
                end for
            end for
        end if
        $(\mathcal{S}, \mathcal{M}, \mathcal{V}) \leftarrow$ ADAM_Update$(\mathcal{S}, \mathcal{G}, \mathcal{M}, \mathcal{V}, \alpha, \beta_1, \beta_2, \epsilon, t)$
    end for
end for

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

5.1.3 | Alternating Direction Fitting (ADF)

Gradient-based methods like ADAM are computationally cheaper than ALS, however, these methods converge more slowly, taking much more time to reach an accurate decomposition. An alternative to solving least squares systems that arise in the ALS method described above is the ADF algorithm [95]. ADF can be viewed as a variant of ALS that aims at reducing the cost per iteration while maintaining a similar convergence property to get the best of both worlds [18]. The ADF method relies on each of the linear systems being solved for ALS subiteration having orthonormal columns. Once this property is satisfied, ADF replaces these least-squares solves with gradient-based updates of individual core entries with accelerated convergence using an over-relaxation step size computation scheme. Our tensor network reformulation of the butterfly decomposition allows us to make the observation that this property can be satisfied for the butterfly decomposition, enabling a straightforward implementation of the ADF algorithm.

The step-size for each slice of the cores is determined by using the formula described in algorithm 5 in [18]. The key comparison point with respect to ALS is that ALS forms normal equations in Line 32 (and Lines 11 and 21) of Algorithm 1, leading to linear systems of sizes $r^2 \times r^2$ requiring $\mathcal{O}(r^4)$ cost for each slice. Instead, ADF does not compute these systems at all and works with gradient and sparse butterfly reconstruction contractions that cost $\mathcal{O}(|\Omega| r^2 \log^2 n)$ per iteration with an additional cost of orthogonalizing cores at each subiteration, which requires $\mathcal{O}(n)$ parallel QR decompositions of matrices of size $2r \times r$. The total per iteration cost of ADF for butterfly thus becomes $\mathcal{O}(|\Omega| r^2 \log^2 n + nr^2 \log n)$. The implementation details of the orthogonalization for ADF for butterfly completion are described in Appendix A.3. The proposed ADF-based butterfly completion algorithm is described as Algorithm 6.

5.2 | Initial Guess Generation

The ALS, ADAM, and ADF methods require an initial guess for the factors. Although random initial guesses are used for many existing tensor and matrix completion algorithms for data science and scientific computing applications, we remark that they do not work well for the proposed butterfly completion algorithms (and the reference QTT completion algorithms), particularly for applications involving highly oscillatory matrices. This is particularly because the larger number of parameters leading to a more complex optimization landscape. Additionally, as we will demonstrate in numerical experiments, we use $|\Omega| = \mathcal{O}(nr \log n)$ observed entries, making full recovery significantly more challenging and requiring an initial guess that is closer to the true solution in Frobenius norm than a random guess. We propose to first use the low-rank matrix completion algorithm (i.e., Algorithm 1 with $L = 0$ and $r = R$, R being a constant much smaller than the actual rank of the matrix). Neither $|\Omega| = \mathcal{O}(nr \log n)$ nor $r = R$ is sufficient for the low-rank completion algorithm to recover the underlying matrix with sufficient accuracy, that is, the relative error over observed entries defined in (5.6) stays over 0.5. However, the resultant low-rank matrix provides a rough approximation of the target matrix and can serve as a better initial guess than a random guess for the butterfly completion algorithm. Our experimental observations suggest that obtaining a low rank decomposition with relative test error of about 0.9 is enough to finally obtain a good butterfly matrix approximation. It is worth mentioning that the low-rank completion algorithm only requires $\mathcal{O}(t_{\max}|\Omega| R^2 + nR^3)$ operations.

Once the initial low-rank matrix is computed, the low-rank matrix of rank R needs to be converted to a butterfly matrix of L levels and rank r, that is, $\boldsymbol{S}^i, i = 1, \ldots, L+2$, to use it as the initial guess. This low-rank to butterfly conversion can be performed efficiently using a randomized algorithm based on a randomized algorithm that uses matrix-vector products of the butterfly matrix [25]. In what follows, we briefly describe the randomized algorithm shown in Algorithm 4.The input to Algorithm 4 is the low rank matrix $\boldsymbol{X} = \boldsymbol{AB}$ of rank R, desired numbers of butterfly levels L and rank r, and an oversampling integer p. The algorithm sketches the relevant blocks of $\boldsymbol{X}$ using Kronecker-structured sketching matrices. Specifically, it employs Gaussian random matrices in Kronecker product with appropriately sized identity matrices. We express these operations in tensor form in Algorithm 4.

The algorithm proceeds by first reshaping both the low rank factors to tensors using the reshape presented in (4.1). We then generate a random Gaussian tensor of appropriate size. The tensor is a reshaping of matrix of size $n \times (r + p)$ using (4.1). A tensor contraction with cost $\mathcal{O}(nrR)$ is computed in Line 5 to sketch the factors using the generated random Gaussian tensor. The column and row basis for outer-most levels can then be computed with QR factorization with column pivoting over the blocks of the resulting tensors denoted by QRCP in Line 6. Note that these blocks can be executed in parallel, and the QR factorization costs $\mathcal{O}(nr^2)$ operations, and the blocks of factor tensors are updated to the upper triangular factors obtained from the QR factorization. Next, for each inner level $l = 1, \ldots, H$, a new Gaussian random tensor of order $L + 2 - l$ is generated. The tensors implicitly represent Gaussian random matrices obtained as the Kronecker product of identity matrices and appropriately sized Gaussian random matrices. The sketching and QR factorization are performed

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

ALGORITHM 4 | Randomized_LR_to_Butterfly.

1: **Input:** Factors $\boldsymbol{A}$ and $\boldsymbol{B}$ of a $n \times n$ low-rank matrix with rank R, butterfly levels L and butterly rank r, and oversampling factor p
2: **Output:** Butterfly cores $\boldsymbol{\mathcal{S}}^i, i = 1, \dots, L+2$.
3: Reshape $\boldsymbol{A}$ of shape $n \times R$ to $\boldsymbol{\mathcal{A}}$ of shape $2 \times \cdots \times 2 \times c \times R$ and $\boldsymbol{B}$ of shape $n \times R$ to $\boldsymbol{\mathcal{B}}$ of shape $2 \times \cdots \times 2 \times c \times R$ by using (4.1)
4: Generate random tensor $\boldsymbol{\mathcal{O}}$ of order $L+2$ and shape $2 \times \cdots \times 2 \times c \times (r+p)$
5: $m_{i_0 \dots i_L q} = \sum_z a_{i_0 \dots i_L z} \sum_{j_0,\dots,j_L} b_{j_0 \dots j_L z} o_{j_0 \dots j_L q}$
$n_{j_0 \dots j_L q} = \sum_z b_{j_0 \dots j_L z} \sum_{i_0,\dots,i_L} a_{i_0 \dots i_L z} o_{i_0 \dots i_L q}$
6: $\boldsymbol{S}^1(i_0, \dots, i_{L-1}, :, :), \boldsymbol{V}(i_0, \dots, i_{L-1}, :, :) \leftarrow \text{QRCP}\big(\boldsymbol{M}(i_0, \dots, i_{L-1}, :, :), r\big)$
$\boldsymbol{S}^{L+2}(j_0, \dots, j_{L-1}, :, :), \boldsymbol{W}(j_0, \dots, j_{L-1}, :, :) \leftarrow \text{QRCP}\big(\boldsymbol{N}(j_0, \dots, j_{L-1}, :, :), r\big)$ ▷ QR with column pivoting
7: **for** $l = 1$ to H **do**
8: Generate random tensor $\boldsymbol{\mathcal{O}}$ of order $L+2-l$ and shape $2 \times \cdots \times 2 \times c \times (r+p)$
9: $m_{i_0 \dots i_{L-l} j_0 \dots j_{l-1} r_l q} = \sum_z v_{i_0 \dots i_{L-l} j_0 \dots j_{l-2} r_l z} \sum_{j_l,\dots,j_L} b_{j_0 \dots j_L z} o_{j_l \dots j_L q}$
$n_{i_0 \dots i_{l-1} j_0 \dots j_{L-l} r_{L+2-l} q} = \sum_z w_{i_0 \dots i_{l-2} j_0 \dots j_{L-l} r_{L+2-l} z} \sum_{i_l,\dots,i_L} a_{i_0 \dots i_L z} o_{i_l \dots i_L q}$
10: $\boldsymbol{S}^{l+1}(i_0, \dots, i_{L-l}, j_0, \dots, j_{l-1}, :, :), \boldsymbol{V}(i_0, \dots i_{L-l}, j_0, \dots, j_{l-1})$
$\leftarrow \text{QRCP}(\boldsymbol{M}(i_0, \dots, i_{L-l}, j_0, \dots, j_{l-1}), r)$
$\boldsymbol{S}^{L+2-l}(i_0, \dots, i_{l-1}, j_0, \dots, j_{L-l}, :, :), \boldsymbol{W}(i_0, \dots, i_{l-1}, j_0, \dots, j_{L-l}, :, :)$
$\leftarrow \text{QRCP}(\boldsymbol{N}(i_0, \dots, i_{l-1}, j_0, \dots, j_{L-l}, :, :), r)$
11: **end for**
12:

$$s^{H+1}_{i_0 \dots i_H j_0 \dots j_{H-1} r_H r_{H+1}} = \sum_q s^{H+1}_{i_0 \dots i_{H-1} j_0 \dots j_{H-1} r_H q} \sum_z v_{i_0 \dots i_{H-1} j_0 \dots j_{H-1} q z} w_{i_0 \dots i_{H-1} j_0 \dots j_{H-1} r_{H+1} z}$$

at Line 9 and 10. These operations respectively cost $\mathcal{O}(nRr^2)$, $\mathcal{O}(nr^3)$ for each level l. Once all factor cores have been computed, $\boldsymbol{\mathcal{S}}^{H+1}$ is updated by contracting the resulting tensors in Line 12, which requires $\mathcal{O}(nRr \log n)$ operations. Note that this computation is equivalent to the computation done for the middle-level in algorithm 4.1 in [25] and sketching is not required for the middle level factor when the matrix is given as a low-rank decomposition. Overall, the complexity of Algorithm 4 is $\mathcal{O}(nRr \log n + nRr^2 \log n + nr^3 \log n)$.

5.3 | Full Tensor-Based Butterfly Completion

The overall workflow and cost of the proposed tensor completion algorithms for butterfly matrices are as follow. Assuming a butterfly rank r and given $|\Omega| = \mathcal{O}(nr \log n)$ (which is the empirical estimate of the minimal number entries required by the butterfly completion), we first generate the initial guess by low-rank completion with a constant number of iterations and rank estimate R, followed by the low-rank to butterfly conversion (Algorithm 4). Unless otherwise stated, we use $R = r$. Hence the initial guess generation requires a total of $\mathcal{O}(nr^4 \log n)$ operations. Next, we run the butterfly completion (ALS-based Algorithm 1, ADAM-based Algorithm 3, or ADF-based) for at most t_{max} iterations. Here t_{max} is also set to a constant. Each iteration requires $\mathcal{O}(|\Omega| r^4 \log^2 n + nr^6 \log n) = \mathcal{O}(nr^5 \log^3 n)$ operations for ALS and $O(|\Omega| r^2 \log^2 n + nr^2 \log n) = O(nr^3 \log^3 n)$ operations for ADAM and ADF. As a baseline, we also implement several QTT tensor completion algorithms that are ALS, ADAM or ALS-based. All of the above algorithms are implemented in Python and are available online.[1]

A remark on sampling complexity. We remark that identification of the sampling complexity of various matrix and tensor completion algorithms is a still active research area. Ideally, the number of observed entries $|\Omega|$ should scale linearly with the number of degrees of freedom of the underlying matrix or tensor representation. While practical algorithms for completion have a long history of development, theoretical guarantees regarding their optimal sampling complexity have only recently been established. For low-rank matrix completion, earlier works can reach the optimal sampling complexity $|\Omega| = \mathcal{O}(rn)$ only for nuclear norm-based algorithms [96, 97], but sub-optimal complexity for ALS-based [98] algorithms.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

The optimal complexity for ALS-based algorithms has only been recently proved [99]. For QTT (or TT) completion, a very recent work proves the optimal $|\Omega| = \mathcal{O}(r^2 \log n)$ sampling complexity for Riemannian optimization-based completion algorithms [100]. As our paper represents the first completion algorithms for butterfly representation, we do not attempt to establish a theoretical bound for the sampling complexity, but rather focus on empirically demonstrating the effectiveness of the proposed butterfly completion algorithms for highly oscillatory matrices. We empirically show that the sampling complexity of $|\Omega| = \mathcal{O}(rn \log n)$ is sufficient for several numerical examples in Section 6 and leave its theoretical justification as a future work.

6 | Numerical Experiments

This section provides several numerical examples to demonstrate the convergence, accuracy and efficiency of the proposed completion algorithms when applied to incomplete data from high-frequency Green's function for wave equations (Section 6.1), inverse scattering applications (Section 6.2) and Radon transforms (Aection 6.3). Specifically, the proposed algorithms are butterfly completion algorithms that are ALS-based (Algorithm 1), ADAM-based (Algorithm 3), and ADF-based (Algorithm 6). As for the baseline algorithms, we use the ALS-based low-rank matrix completion algorithm (Algorithm 1 with $L = 0$) and the QTT tensor completion algorithms that are ALS-based (Algorithm 5), ADAM-based and ADF-based. A convergence tolerance of $\epsilon = 10^{-3}$ is used for all the algorithms. All the proposed and baseline algorithms are implemented in Python and executed using one CPU processor of the Perlmutter machine at NERSC in Lawrence Berkeley National Laboratory, which has AMD EPYC 7763 processors and 128 GB of 2133 MHz DDR4 memory.

6.1 | Green's Function for Helmholtz Equation

In this subsection, we consider a square matrix discretized from the 3D free-space Green's function for high-frequency Helmholtz equations. Let $n = c \cdot 2^L$ denote the matrix size with $c = 4$ and L representing the number of butterfly or QTT levels. The matrix $\overline{\boldsymbol{T}}$ is given by $\bar{t}_{i,j} = \frac{\exp(-\mathrm{i}\omega\rho)}{\rho}$, where ω is the wavenumber, $\rho = |x^i - y^j|$, $x^i = (\frac{i_1}{\sqrt{n}}, \frac{i_2}{\sqrt{n}}, 0)$, $y^j = (\frac{j_1}{\sqrt{n}}, \frac{j_2}{\sqrt{n}}, 1)$. Here, $i_1 = \lfloor (i-1)/\sqrt{n} \rfloor$, $i_2 = (i-1) \bmod \sqrt{n}$, and similarly for j_1 and j_2. In this test, $\omega = \frac{\sqrt{n}\pi}{5}$, that is, one wavelength is discretized into 10 points. We apply the completion algorithms to the permuted matrix $\boldsymbol{T} = \boldsymbol{P}\overline{\boldsymbol{T}}\boldsymbol{P}^T$ with $\boldsymbol{P}$ presenting the KD-tree-based reordering that yields the CLR property of $\boldsymbol{T}$. The completion algorithms are trained on a set of entries denoted by Ω and tested on a set of entries denoted by Ω_{test}, both consisting of randomly generated row and column index pairs. In other words, we define the training error using (5.6), and test error using (5.6) with $\Omega \to \Omega_{\text{test}}$.

First, we consider an example of $n = 4096$ and compare the reconstruction quality of the butterfly completion algorithm (Butterfly-ALS of Algorithm 1) and low-rank matrix completion algorithm. The real part of the full matrix $\boldsymbol{T}$ and the observed submatrix $\boldsymbol{T}_\Omega$ are shown in Figure 3a and b, where the number of observed entries is empirically set to $|\Omega| = 66n \log n = 3244032$. The low-rank completion algorithm uses rank estimate $r = 400$, max iterations $t_{max} = 20$, and the reconstructed full matrix $\boldsymbol{X}$ is shown in Figure 3c. The butterfly completion algorithm (Algorithm 1) uses rank estimate $r = 11$, max iterations $t_{max} = 20$ and use the initial guess with $\boldsymbol{R} = 11$. The reconstructed full matrix $\boldsymbol{X}$ is shown in Figure 3d. As one can see, the reconstruction quality of Algorithm 1 is significantly better than that of the low-rank completion algorithm, given the same set of observed entries.

Next, we examine the convergence of the completion algorithms using the real part of the Green's function as $\boldsymbol{T}$. We first consider an example of $n = 16,384$ ($L = 12$). As the ground truth, the matrix rank, butterfly rank and QTT rank with tolerance $\epsilon = 10^{-3}$ are 366, 11, and 96, respectively. For all algorithms in this example, we set the rank estimates in the completion algorithms as the above true numerical ranks. Specifically, for low-rank completion, we use $r = 366$ and $|\Omega| = 10rn = 59,965,440$; for the ALS-based (Algorithm 1), gradient-based (Algorithm 3) and ADF-based (Algorithm 6) butterfly completion, we use $r = 11$, $R = 11$, and $|\Omega| = 6rn \log n = 15,138,816$; for the ALS-based (Algorithm 5), gradient-based and ADF-based QTT completion, we set $r = 96$, $R = 11$, and $|\Omega| = 16r^2 \log n = 2,240,000$. The convergence history for the training and test errors are shown in Figure 4. We set max iterations $t_{max} = 20$ for all algorithms. As we set $|\Omega|$ and r to approximately the minimum numbers required by each completion algorithm, all algorithms show convergence. Specifically, gradient-based methods such as Butterfly-ADAM and QTT-ADAM show slowest convergence; ADF-based algorithms show slightly worse convergence behaviors than ALS-based algorithms. For low-rank matrix completion, we also plot the convergence history for $|\Omega| = 4rn = 23,986,176$ (labeled as "low-rank(low)"), which appears to be insufficient. For QTT-ADF, we also plot the convergence history for $|\Omega| = 8r^2 \log n = 1,120,000$ (labeled as "QTT-ADF(low)"), which also turns to be an insufficient number for convergence. Moreover, although QTT completion requires the lowest

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

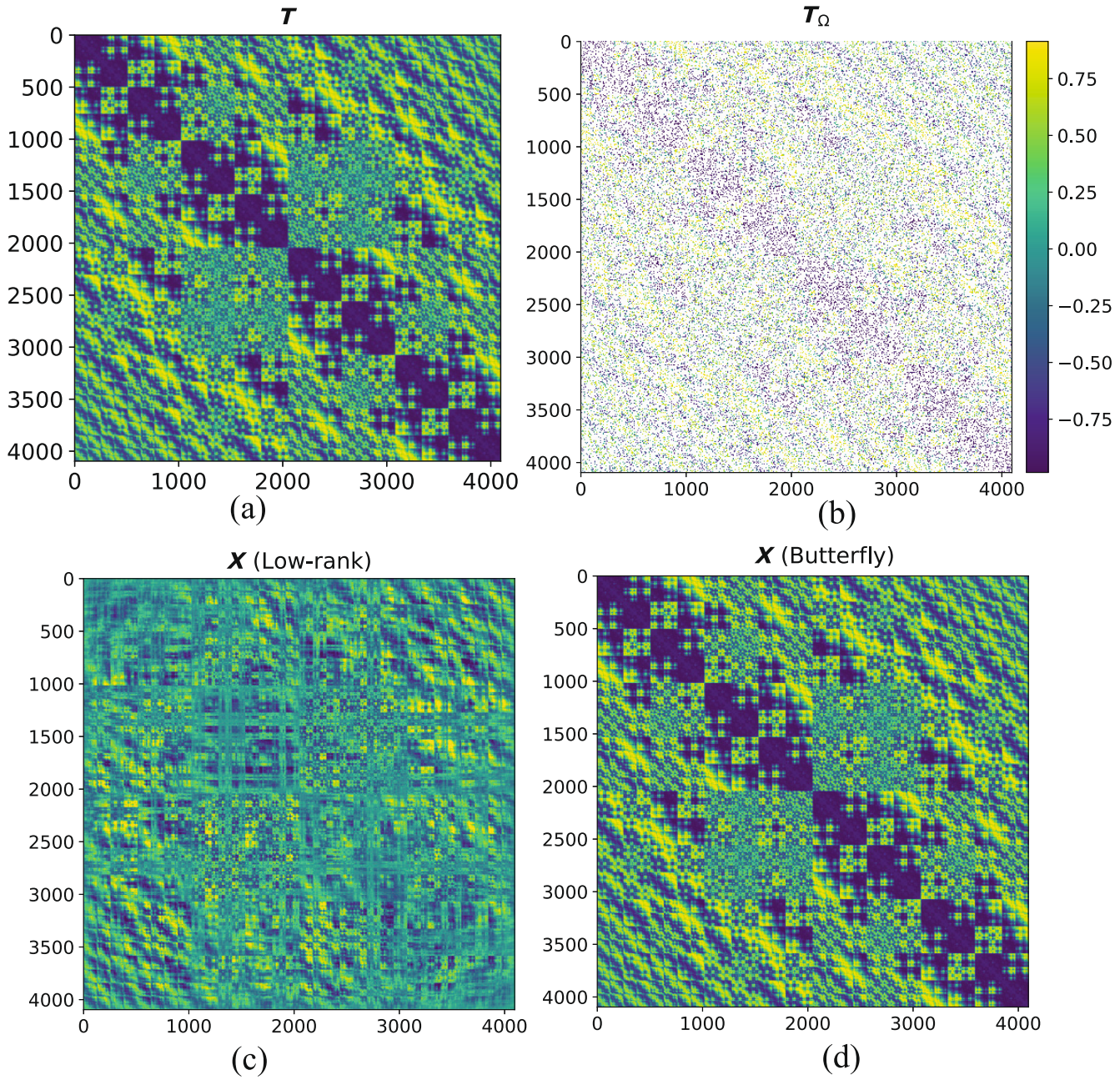


FIGURE 3 | The real part of a Green's function matrix with $n = 4096$, $c = 4$, $L = 10$. (a) The ground truth data $\boldsymbol{T}$. (b) The observed data $\boldsymbol{T}_\Omega$ with $|\Omega| = 66n \log n = 3,244,032$. (c) The low-rank reconstruction with rank $r = 400$. (d) The butterfly reconstruction using Algorithm 1 with rank $r = 11$.

$|\Omega|$, the cost per iteration (see Figure 6 for more detail) is most expensive (e.g., at $L = 12$, QTT-ADF is 10× slower than Butterfly-ALS, and QTT-ALS is 50× slower than Butterfly-ALS) and the test errors are less satisfactory than butterfly completion. Next, an even larger problem is tested as $n = 65,536$ ($L = 14$). We only test the low-rank matrix completion algorithms with $r = 200$ or 400 and Butterfly-ALS/Butterfly-ADF with $r = 11$. We skip Butterfly-ADAM due to its slow convergence and QTT-completion due to its high memory usage and computational cost attributed to large QTT ranks. We set $|\Omega| = 69,206,016$ for both low-rank and butterfly completion algorithms. As can be seen from Figure 5, the low-rank matrix completion algorithms diverge due to both insufficient rank estimate and $|\Omega|$ (as will be seen in Figure 6, it requires a projected 4 h per iteration for low-rank completion to converge). In contrast, the butterfly algorithm converges within 20 iterations. Again, the convergence of Butterfly-ADF is slightly worse than Butterfly-ALS.

Next, we investigate the computational complexity of these completion algorithms with the minimum number of nonzeros it took for the algorithms to converge. As $L \in \{8, \dots, 14\}$, we set the number of observed entries to $|\Omega| = 10rn$ for the

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

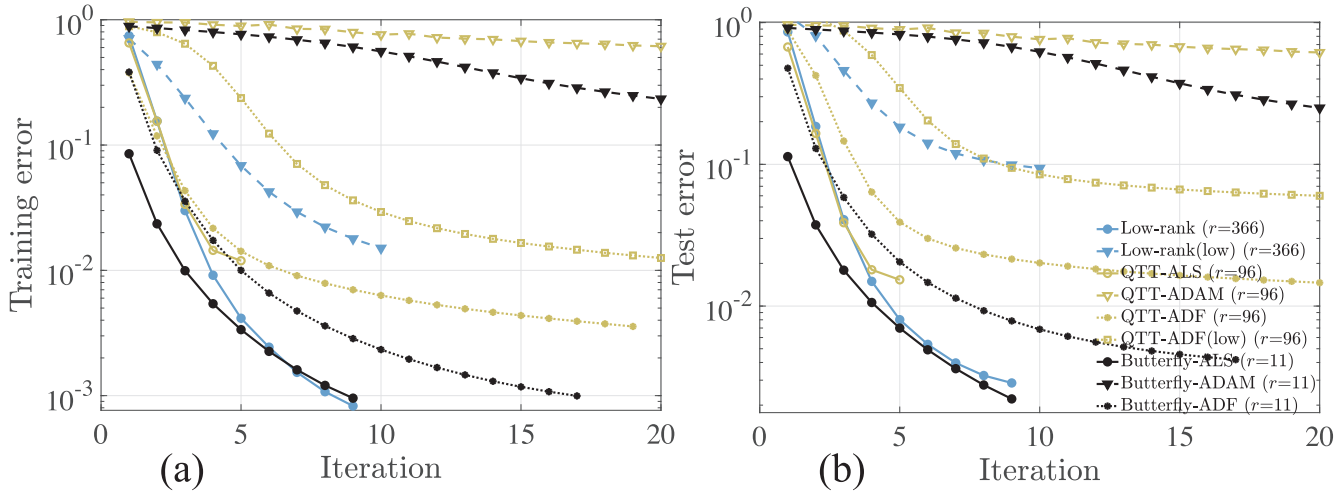


FIGURE 4 | The convergence of a Green's function example with $n = 16,384$, $c = 4$, $L = 12$ using the low-rank matrix completion with $r = 366$, $|\Omega| = 10rn = 59,965,440$ (i.e., $|\Omega|/n^2 = 0.22$), and $|\Omega| = 4rn = 23,986,176$ (i.e., $|\Omega|/n^2 = 0.09$, labeled as "Low-rank(low)"), butterfly completion (ALS, ADAM, ADF) with $r = 11$, $|\Omega| = 6rn \log n = 15,138,816$ (i.e., $|\Omega|/n^2 = 0.056$), and QTT completion (ALS, ADAM, ADF) with $r = 96$, $|\Omega| = 16r^2 \log n = 2,240,000$ (i.e., $|\Omega|/n^2 = 0.0083$). In addition, "QTT-ADF(low)" denotes ADF-based QTT completion with $|\Omega| = 8r^2 \log n = 1,120,000$. (a) Training error. (b) Test error.

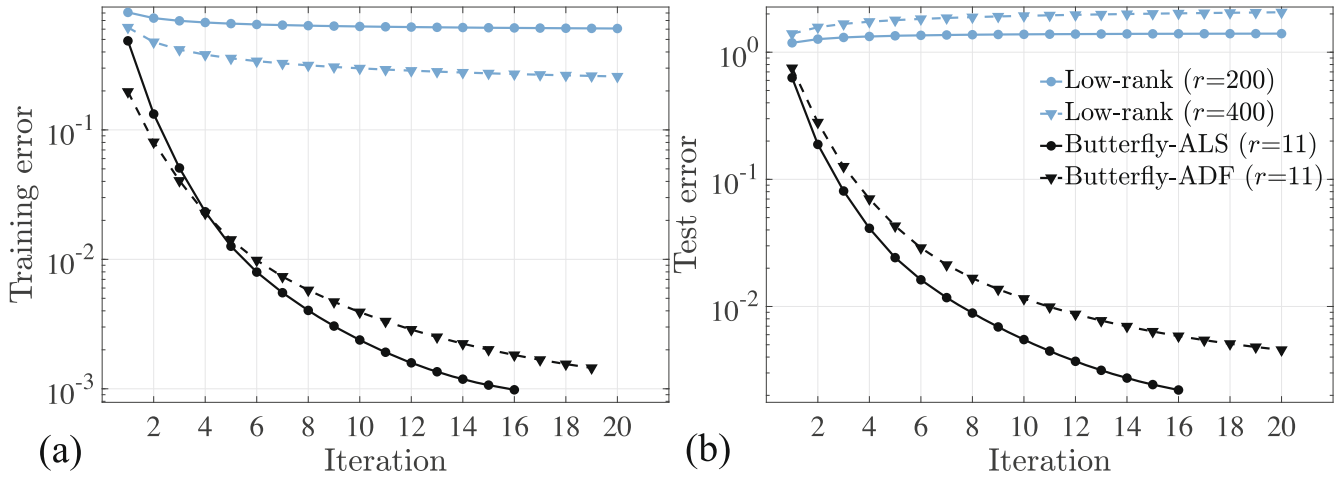


FIGURE 5 | The convergence of a Green's function example with $n = 65,536$, $c = 4$, $L = 14$ using butterfly completion (ALS and ADF) with rank $r = 11$. (a) Training error with $|\Omega| = 6rn \log n = 69,206,016$. (b) Test error with $|\Omega_{\text{test}}| = rn \log n = 11,534,336$. The results using low-rank matrix completion with rank $r = 200, 400$ and the same $|\Omega|$ as butterfly completion are also plotted.

low-rank matrix completion algorithm, $|\Omega| = 6n \log n$ for the butterfly completion algorithms, and $|\Omega| = 16r^2 \log n$ for the QTT algorithms. The rank estimates are set to the empirically smallest numbers such that the completion algorithms converge for a given n. More specifically, we set $r = 57, 134, 366$ for the low-rank algorithm, $r = 11$ for the butterfly algorithms and $r = 30, 54, 96$ for the QTT algorithm. Note that for $n = 65536$ ($L = 14$), both low-rank matrix completion and QTT completion algorithms run out of memory or are too expensive to finish. We plot the CPU time per iteration for each algorithm in Figure 6a and the number of observed entries $|\Omega|$ in Figure 6b. For the time per iteration, Butterfly-ADF is about 2× slower than Butterfly-ADAM as it requires two gradient-like computation per iteration (See Line 7 and 8 of Algorithm 6). Surprisingly, we note that Butterfly-ALS and Butterfly-ADAM almost overlap with each other. This is likely due to the fact that the butterfly rank $r = 11$ is a small constant independent of n. As a result, we conclude that ALS is the best algorithm among the three for butterfly completion of highly oscillatory matrices due to its fastest convergence (see Figure 4 and Figure 5) and time per iteration. On the other hand, QTT-ADF is much faster than QTT-ALS due to relatively large QTT ranks. Therefore, we conclude that ADF is the best algorithm among the three for QTT completion for highly oscillatory matrices. Note that although QTT completion requires less $|\Omega|$ than butterfly completion (see Figure 6b), the time per iteration at $L = 12$ is 3349 s for QTT-ADF but only 373 s for Butterfly-ALS. This time difference will be even

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

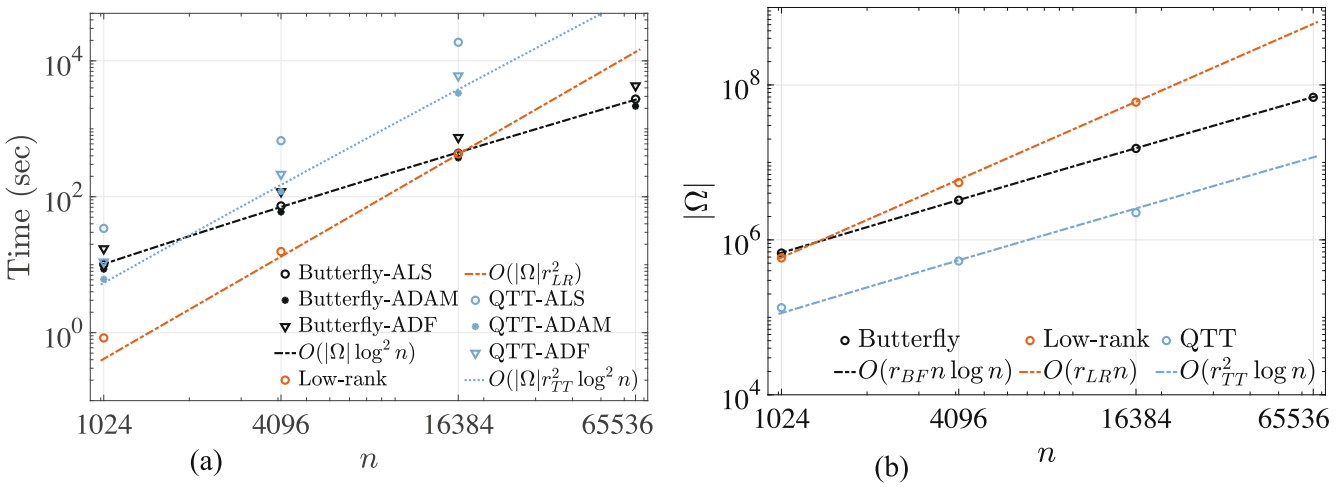


FIGURE 6 | (a) Computational time per iteration and (b) $|\Omega|$ for a Green's function example with $c = 4$, $L = 8 - 14$ using the low-rank matrix completion, butterfly completion (ALS, ADAM, ADF), and QTT completion (ALS, ADAM, ADF). Butterfly completion: $|\Omega| = 6rn \log n$, $r = 11$ (i.e., the actual butterfly rank for $\boldsymbol{T}$ with tolerance $\epsilon = 10^{-3}$). Low-rank completion: $|\Omega| = 10rn$ with $r = 57, 134, 366$ (i.e., the actual rank for $\boldsymbol{T}$ with tolerance $\epsilon = 10^{-3}$). Note that in theory low-rank representation requires $r = \mathcal{O}(n)$ for sufficiently large n, but these numerical tests exhibit $r = \mathcal{O}(\sqrt{n})$ up to $n = 16384$. QTT completion: $|\Omega| = 16r^2 \log n$ with $r = 30, 54, 96$ (i.e., the actual QTT rank for $\boldsymbol{T}$ with tolerance $\epsilon = 10^{-3}$). The QTT ranks scale approximately as $r = \mathcal{O}(n^{0.5})$.

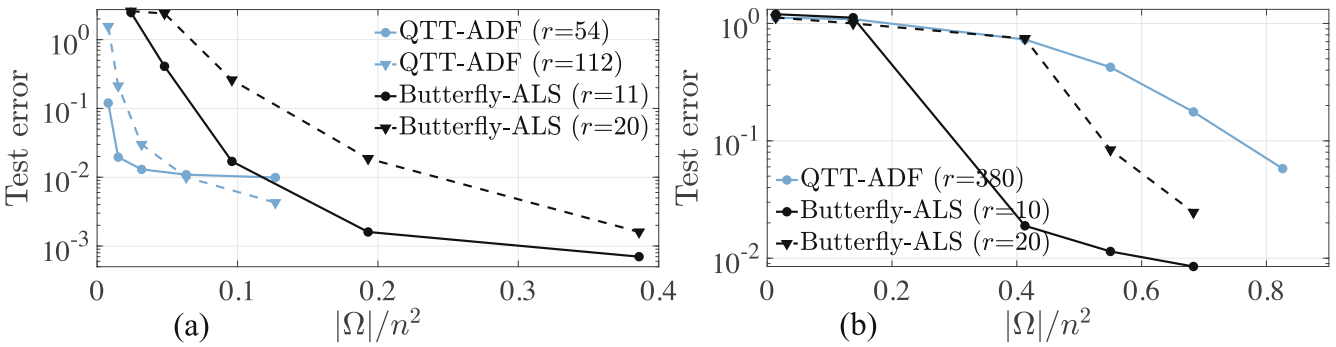


FIGURE 7 | Test errors with varying observation counts $|\Omega|$ and rank estimates r for butterfly completion with ALS and QTT completion with ADF. (a) A Green's function example with $c = 4$, $L = 10$. The actual rank for $\boldsymbol{T}$ with tolerance $\epsilon = 10^{-3}$ is 10 for butterfly decomposition and 54 for QTT decomposition. (b) A Radon transform example with $c = 4$, $L = 8$. The actual rank for $\boldsymbol{T}$ with tolerance $\epsilon = 10^{-3}$ is 10 for butterfly decomposition and 380 for QTT decomposition. We remark that for Radon transforms, QTT exhibits full ranks and requires $|\Omega| = n^2$ for accurate recovery.

larger for larger L. In terms of asymptotic behaviors, $|\Omega|$ scales as $\mathcal{O}(n \log n)$ for both butterfly (r stays as a small constant) and QTT completion ($r = \mathcal{O}(n^{0.5})$) algorithms, and super-linearly for low-rank completion. As a result, the butterfly completion algorithms empirically achieve the lowest total asymptotic computational complexity, $\mathcal{O}(n\log^3 n)$, among all the completion algorithms.

Moreover, we study the effect of rank estimate and $|\Omega|$ on the test error for QTT-ADF and Butterfly-ALS (i.e., the best performing algorithms out of the three QTT completion algorithms and the tree butterfly completion algorithms) using an example of $n = 4096$ ($L = 10$). For QTT completion, the actual rank with tolerance $\epsilon = 10^{-3}$ is 54 and we set the rank estimates to $r = 54, 112$. For butterfly completion, the actual rank with tolerance $\epsilon = 10^{-3}$ is 11 and we set the rank estimates to $r = 11, 20$. The test errors with varying observation ratios $|\Omega|/n^2$ are plotted in Figure 7a. For most values of observation ratios, the use of rank estimate higher than necessary leads to over-parameterization and hence worse test errors. For this Green's function example, QTT completion typically requires lower observation ratios than butterfly completion (e.g., QTT-ADF reaches an error of 0.013 at $|\Omega|/n^2 = 0.03$ while Butterfly-ALS reaches an error of 0.017 at $|\Omega|/n^2 = 0.1$), but struggles to achieve the same level of test errors as the ratio increases.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

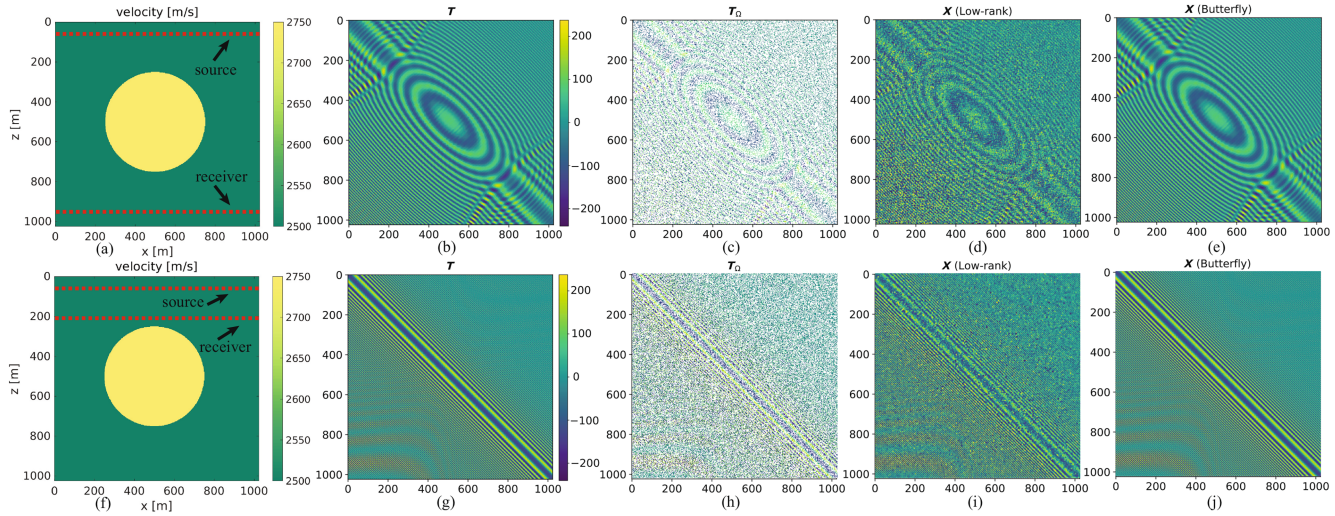


FIGURE 8 | Top: the transmission problem. Bottom: the reflection problem. From left to right: the velocity model under consideration, the full target matrix $\boldsymbol{T}$, the observed matrix entries, the reconstructed matrix X by the low-rank completion algorithm and the reconstructed matrix X by the butterfly completion algorithm. Here we set $n = 1024$, $c = 4$, $L = 8$, $|\Omega| = 27n \log n$ for the transmission problem and $|\Omega| = 48n \log n$ for the reflection problem.

6.2 | Inverse Wave Scattering

Next, we consider a more realistic inverse scattering application governed by the following 2D Helmholtz equation

$$\nabla^2 p(\mathbf{x}) + \frac{\omega^2}{v^2(\mathbf{x})} p(\mathbf{x}) = -f(\mathbf{x}), \tag{6.1}$$

in a physical domain of 1000 m × 1000 m. Here $\mathbf{x} = (x_1, x_2)$, $f(\mathbf{x})$ is the excitation function, $p(\mathbf{x})$ is the pressure wave field, ω is the angular frequency, and $v(\mathbf{x})$ is the velocity profile. Here $v(\mathbf{x}) = 2500$ m/s everywhere except for a circular target region where $v(\mathbf{x}) = 2750$ m/s (see the first column of Figure 8). In the inverse scattering application, one is interested in generating an array of point sources $f(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_i^s)$ at locations $\mathbf{x}_i^s$, $i = 1, 2, \ldots, n$ and measuring the pressure wave fields $p(\mathbf{x}_j^o)$ at locations $\mathbf{x}_j^o$, $j = 1, 2, \ldots, n$; from this measured $n \times n$ Green's function matrix, one can infer the unknown velocity ($v(\mathbf{x}) = 2750$ m/s is the ground truth to be inferred) inside the target region. Here, we demonstrate how to reconstruct the $n \times n$ matrix in some compressed formats from incomplete measurement, which can be potentially used to infer the unknown velocity profile. We consider both the transmission problem where the source and observer arrays (with equally spaced points) are located at $z = 50$ m and $z = 950$ m (see the top panel of Figure 8), and the reflection problem where the source and observer arrays are located at $z = 50$ m and $z = 200$ m (see the bottom panel of Figure 8).

To generate the Green's function matrix $\boldsymbol{T}$, we leverage the simulation package WAVEFORM [101] to simulate the responses $p(\mathbf{x}_j^o)$, $j = 1, 2, \ldots, n$ for each excitation function $f(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_i^s)$. The frequency is set to $\omega = 500\pi$ and n is set to 1024 (i.e., $c = 4$, $L = 8$). Once the matrix is fully generated, we assume only $|\Omega|$ entries of the matrix are observed and our goal is to fully recover $\boldsymbol{T}$. For the transmission problem, we empirically set $|\Omega| = 27n \log n$, $r = 9$ for the butterfly completion Algorithm 1 and $r = 120$ for the low-rank completion. For the reflection problem, we empirically set $|\Omega| = 48n \log n$, $r = 12$ for the butterfly completion Algorithm 1 and $r = 120$ for the low-rank completion. The reconstructed matrices are plotted in the last two columns of Figure 8. Clearly, the butterfly completion algorithm achieves significantly better reconstruction quality than the low-rank completion algorithm.

6.3 | Radon Transform

In this subsection, we consider a matrix $\boldsymbol{T}$ discretized from a 1D Radon transform:

$$t_{i,j} = \exp(2\pi \mathrm{i} \phi(x^i, y^j))$$
$$\phi(x, y) = x \cdot y + c(x)|y|$$
$$c(x) = (2 + \sin(2\pi x))/8$$

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

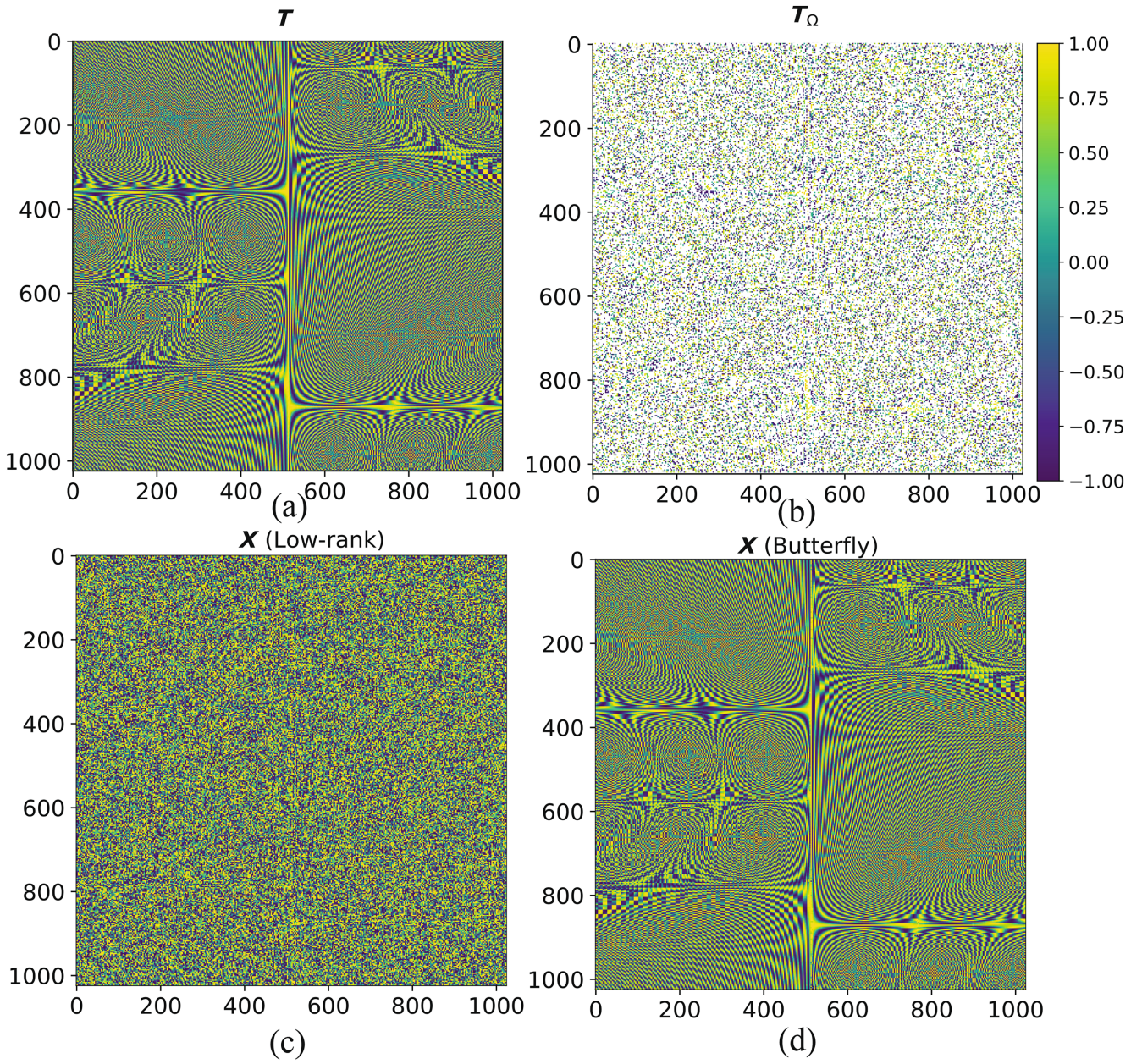


FIGURE 9 | The real part of a Radon transform matrix with $n = 1024$, $c = 4$, $L = 8$. (a) The ground truth data $\boldsymbol{T}$. (b) The observed data $\boldsymbol{T}_\Omega$ with $|\Omega| = 30n \log n = 307,200$. (c) The low-rank reconstruction using low-rank completion with rank $r = 400$. (d) The butterfly reconstruction using Algorithm 1 with rank $r = 10$. Note that only the real part of the data is plotted.

with $x^i = \frac{i}{n}$, $y^j = j - \frac{n}{2}$, $i, j = 1, 2, \ldots, n$. For seismic migration, $c(x)$ represents a 1D velocity model at spatial location x, which can be noisy or inaccurate, leading to $\boldsymbol{T}$ with noisy or missing data. This matrix is known to exhibit full rank when represented as low-rank or QTT algorithms, but show constant rank when represented as butterfly.

First, we consider an example of $n = 1024$ and compare the reconstruction quality of the butterfly completion algorithm (Algorithm 1) and low-rank matrix completion algorithm. The real part of the full matrix $\boldsymbol{T}$ and the observed submatrix $\boldsymbol{T}_\Omega$ are shown in Figure 9a and b, where the number of observed entries is empirically set to $|\Omega| = 30n \log n = 307,200$. The low-rank matrix completion algorithm uses rank estimate $r = 400$, max iterations $t_{max} = 20$, and the reconstructed full matrix $\boldsymbol{X}$ is shown in Figure 9c. The butterfly completion algorithm (Algorithm 1) uses rank estimate $r = 10$, max iterations $t_{max} = 20$ and the initial guess with $R = 10$. The reconstructed full matrix $\boldsymbol{X}$ is shown in Figure 9d. As one can see, the reconstruction quality of Algorithm 1 is significantly better than that of the low-rank completion algorithm, given the same set of observed entries. This is no surprise as the low-rank completion algorithm requires $r = \mathcal{O}(n)$ and $|\Omega| = \mathcal{O}(n^2)$ for a good reconstruction.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

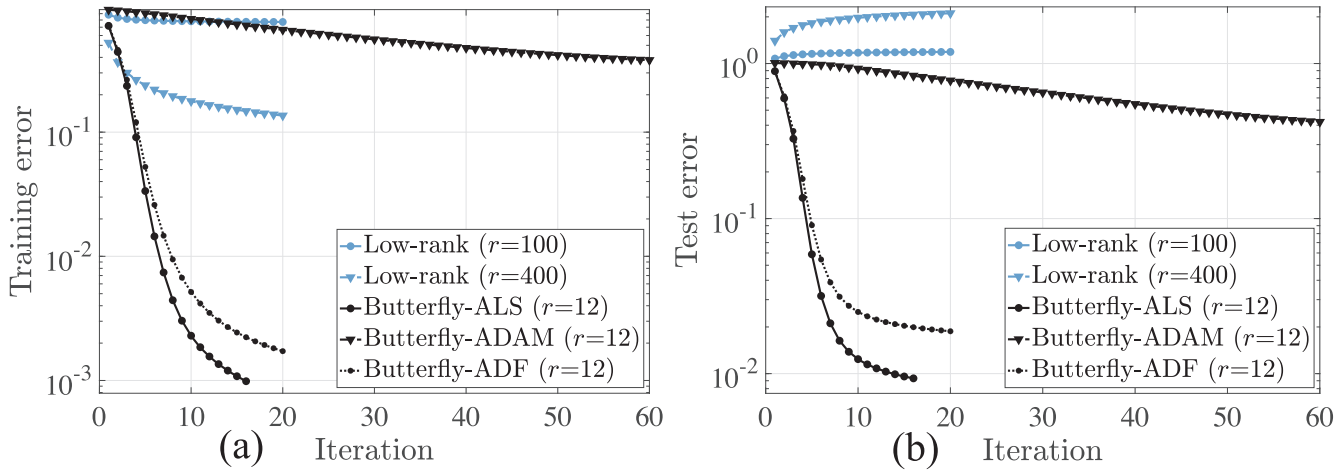


FIGURE 10 | The convergence of a Radon transform example with $n = 4096$, $c = 4$, $L = 10$ using butterfly completion (ALS, ADAM, ADF) with rank $r = 12$ (the actual butterfly rank for $\boldsymbol{T}$ with tolerance $\epsilon = 10^{-3}$ is 10), $|\Omega| = 7rn \log n = 3,440,640$ and $|\Omega_{\text{test}}| = rn \log n = 491,520$. (a) Training error. (b) Test error. The results using low-rank matrix completion with rank $r = 100, 400$ and the same $|\Omega|$ as butterfly completion are also plotted. Note that the actual rank for $\boldsymbol{T}$ with tolerance $\epsilon = 10^{-3}$ is $r = 4096$ (i.e., full rank) for low-rank decomposition.

Next, we examine the convergence of the completion algorithms. We first consider an example of $n = 4096$ ($L = 10$) using the ALS-based (Algorithm 1) and gradient-based (Algorithm 3) and ADF-based (Algorithm 6) butterfly completion with rank $r = 12$ (the actual butterfly rank is 10 with tolerance $\epsilon = 10^{-3}$). Note that the QTT completion algorithm is not considered as the full QTT rank will yield the algorithm prohibitively expensive (see later discussion of Figures 11 and 7b for more detail). The number of observed entries is empirically set to $|\Omega| = 7rn \log n$ for training and $|\Omega_{\text{test}}| = n \log n$ for testing. As the reference, we also tested the low-rank matrix completion with rank $r = 100$ or 200, but using the same $|\Omega|$ as butterfly completion. The convergence history for the training and test errors are shown in Figure 10. We set max iterations $t_{max} = 20$ for all the ALS-based and ADF-based algorithms and $t_{max} = 60$ for the gradient-based algorithm. As expected, the low-rank matrix completion algorithms do not converge due to both insufficient numbers of r and $|\Omega|$. In contrast, the butterfly completion algorithms converge with a small rank estimate $r = 12$. Specially, the ALS-based butterfly algorithm converges in 16 iterations, and the ADF-based butterfly algorithm is slightly worse, while the gradient-based butterfly algorithm takes more than 60 iterations to converge.

Next, we investigate the computational complexity of the low-rank matrix, QTT and butterfly completion algorithms for the radon transform matrices. As $L \in \{6, \dots, 14\}$, we set the number of observed entries to $|\Omega| = n^2$ for low-rank completion and QTT completion as they need all the entries to be observed to converge to sufficient accuracy, and $|\Omega| = 6rn \log n$ for the butterfly algorithms. For QTT completion, we only test QTT-ADF as the actual QTT ranks with tolerance $\epsilon = 10^{-3}$ for $L = 6, 8, 10$ are 140, 380, 1140, which are almost full-rank and scale as $\mathcal{O}(n)$. The rank estimate is $r = 10$ for the butterfly algorithms, $r = n$ for the low-rank algorithm, and $r = 140, 380$ for the QTT completion algorithm. We plot the CPU time per iteration for each algorithm in Figure 11a and the number of observed entries $|\Omega|$ in Figure 11b. Similar to the Green's function example, the time per iteration for ALS-based and ADAM-based butterfly algorithms almost overlap with each other, and ADF-based butterfly algorithm is about 2× slower. Clearly, the butterfly algorithms achieve the quasi-linear asymptotic complexity as $\mathcal{O}(n \log^3 n)$. In stark contrast, low-rank and QTT completion algorithms scale at least as $\mathcal{O}(n^4)$ due to the unfavorable $|\Omega| = n^2$ and $r = \mathcal{O}(n)$ scaling. Note that at $n = 1024$, it requires 5 h per iteration for QTT completion. Obviously, if all the entries of the matrix are available, one should use low-rank or QTT decomposition instead of low-rank or QTT completion.

Finally, we study the effect of rank estimate and $|\Omega|$ on the test error for QTT-ADF and Butterfly-ALS using an example of $n = 1024$ ($L = 8$). For QTT-ADF, the actual rank with tolerance $\epsilon = 10^{-3}$ is 380 and we set the rank estimates to $r = 380$. For Butterfly-ALS, the actual rank with tolerance $\epsilon = 10^{-3}$ is 10 and we set the rank estimates to $r = 10, 20$. The test errors with varying observation ratios $|\Omega|/n^2$ are plotted in Figure 7b. For most values of observation ratios, the use of rank estimate higher than necessary leads to over-parameterization and hence worse test errors. It is clear that QTT-ADF requires almost all matrix entries and hence does not represent an effective completion algorithm for Radon transform matrices.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

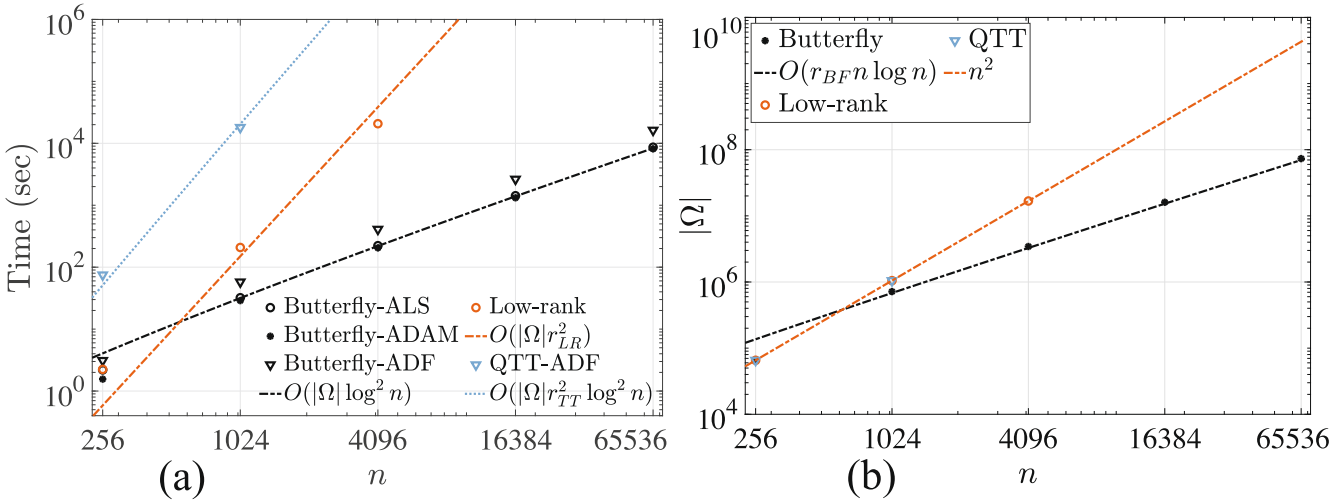


FIGURE 11 | (a) Computational time per iteration and (b) $|\Omega|$ for the Radon transform example with $c = 4$, $L = 6 - 14$ using the low-rank matrix completion, butterfly completion (ALS, ADAM, ADF) and QTT completion (ADF). Butterfly completion: $|\Omega| = 6n\log n$, $r = 10$. Low-rank completion: $|\Omega| = n^2$, $r = n$ (this requires all the entries as $\boldsymbol{T}$ is full-rank). QTT completion: $|\Omega| = n^2$ (this requires all the entries as the actual TT-rank for $\boldsymbol{T}$ is $r = 140, 380, 1140$ for $L = 6, 8, 10$).

7 | Conclusion

In this work, we proposed three low-complexity butterfly completion algorithms, respectively based on ALS, ADAM and ADF, for reconstructing highly oscillatory operators discretized as $n \times n$ matrices requiring only $\mathcal{O}(n\log n)$ observed matrix entries and $\mathcal{O}(n\log^3 n)$ computational cost empirically. The proposed algorithms are based on the reformulation of butterfly matrices into higher order tensors, which allows for efficient implementation via tensor computations. Numerical results demonstrate the superior accuracy, convergence, and efficiency of the proposed algorithms for highly oscillatory operators such as Radon transforms and Green's functions in seismic applications, when compared with other completion algorithms such as low-rank matrix completion and QTT completion. While our experiments utilized only synthetic datasets, the oscillatory operators studied here, such as high-frequency Green's functions and Radon transforms, form the mathematical foundation of several critical imaging tasks, where physical or cost constraints limit data acquisition. The proposed butterfly completion algorithm is potentially useful for addressing missing-trace problem for high-resolution reverse time migration of seismic exploration, the frequency-domain gaps inherent in low-dose medical CT, and the "missing wedge" problem in electron tomography.

Future work includes exploiting butterfly completion for stochastic low-rank decompositions, high-performance and parallel implementations of the butterfly contraction operations, as well as integrating the tensorized reformulation of the butterfly decomposition into neural network architectures to facilitate efficient learning with butterfly matrices.

Acknowledgments

This research has been supported by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Multifaceted Integrated Capability Centers (MMICCs) program, under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAP0024170.

Funding

This work was supported by the US Department of Energy (Grant No. DESC0023296).

Data Availability Statement

The data that support the findings of this study are openly available in Github at https://github.com/navjo2323/butterfly-matrix-completion.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Endnotes

[1]https://github.com/navjo2323/butterfly-matrix-completion.

[2]https://github.com/navjo2323/butterfly-matrix-completion.

References

1. S. Dzhenzher, T. Garaev, O. Nikitenko, A. Petukhov, A. Skopenkov, and A. Voropaev, "Low Rank Matrix Completion and Realization of Graphs: Results and Problems," arXiv preprint arXiv:2501.13935, 2025, https://arxiv.org/abs/2501.13935.

2. J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor Completion for Estimating Missing Values in Visual Data," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, no. 1 (2012): 208–220.

3. R. Kumar, C. Da Silva, O. Akalin, et al., "Efficient Matrix Completion for Seismic Data Reconstruction," *Geophysics* 80, no. 5 (2015): V97–V114, https://doi.org/10.1190/geo2014-0369.1.

4. D. Liu, M. D. Sacchi, and W. Chen, "Efficient Tensor Completion Methods for 5-d Seismic Data Reconstruction: Low-Rank Tensor Train and Tensor Ring," *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022): 1–17, https://doi.org/10.1109/TGRS.2022.3179275.

5. S. G. Lingala, Y. Hu, E. DiBella, and M. Jacob, "Accelerated Dynamic Mri Exploiting Sparsity and Low-Rank Structure: Kt Slr," *IEEE Transactions on Medical Imaging* 30, no. 5 (2011): 1042–1054.

6. Z. Yi, Y. Liu, Y. Zhao, et al., "Joint Calibrationless Reconstruction of Highly Undersampled Multicontrast Mr Datasets Using a Low-Rank Hankel Tensor Completion Framework," *Magnetic Resonance in Medicine* 85, no. 6 (2021): 3256–3271.

7. E. J. Candès, Y. C. Eldar, T. Strohmer, and V. Voroninski, "Phase Retrieval via Matrix Completion," *SIAM Journal on Imaging Sciences* 6, no. 1 (2013): 199–225, https://doi.org/10.1137/110848074.

8. E. Hutter and E. Solomonik, "Application Performance Modeling via Tensor Completion," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 65:1–65:14. (Association for Computing Machinery, 2023), https://doi.org/10.1145/3581784.3607069.

9. R. A. Harshman, "Foundations of the PARAFAC Procedure: Models and Conditions for an "Explanatory" Multi-Modal Factor Analysis," *UCLA Working Papers in Phonetics* 16, no. 1 (1970): 84.

10. T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review* 51, no. 3 (2009): 455–500.

11. L. De Lathauwer, B. De Moor, and J. Vandewalle, "A Multilinear Singular Value Decomposition," *SIAM Journal on Matrix Analysis and Applications* 21, no. 4 (2000): 1253–1278.

12. I. V. Oseledets, "Tensor-Train Decomposition," *SIAM Journal on Scientific Computing* 33, no. 5 (2011): 2295–2317.

13. W. Hackbusch and S. Kühn, "A New Scheme for the Tensor Representation," *Journal of Fourier Analysis and Applications* 15, no. 5 (2009): 706–722.

14. Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, "Tensor Ring Decomposition," *Neural Networks* 89 (2017): 47–61, https://doi.org/10.1016/j.neunet.2016.06.005.

15. E. Candes and B. Recht, "Exact Matrix Completion via Convex Optimization," *Communications of the ACM* 55, no. 6 (2012): 111–119.

16. G. Ballard and T. G. Kolda, *Tensor Decompositions for Data Science* (Cambridge University Press, 2025).

17. D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR)*, (2015), https://arxiv.org/abs/1412.6980.

18. L. Grasedyck, M. Kluge, and S. Kramer, "Variants of Alternating Least Squares Tensor Completion in the Tensor Train Format," *SIAM Journal on Scientific Computing* 37, no. 5 (2015): A2424–A2450.

19. D. Kressner, M. Steinlechner, and B. Vandereycken, "Low-Rank Tensor Completion by Riemannian Optimization," *BIT Numerical Mathematics* 54 (2014): 447–468.

20. J. Xue, Y. Zhao, S. Huang, W. Liao, J. C.-W. Chan, and S. G. Kong, "Multilayer Sparsity-Based Tensor Decomposition for Low-Rank Tensor Completion," *IEEE Transactions on Neural Networks and Learning Systems* 33, no. 11 (2021): 6916–6930.

21. S. Yuan and K. Huang, "A Generalizable Framework for Low-Rank Tensor Completion With Numerical Priors," *Pattern Recognition* 155 (2024): 110678.

22. E. Candes, L. Demanet, and L. Ying, "A Fast Butterfly Algorithm for the Computation of Fourier Integral Operators," *Multiscale Modeling and Simulation* 7 (2009): 1727–1750.

23. Y. Li and H. Yang, "Interpolative Butterfly Factorization," *SIAM Journal on Scientific Computing* 39, no. 2 (2017): A503–A531, https://doi.org/10.1137/16M1074941.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

24. Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying, "Butterfly Factorization," *Multiscale Modeling and Simulation* 13, no. 2 (2015): 714–732.

25. Y. Liu, X. Xing, H. Guo, E. Michielssen, P. Ghysels, and X. S. Li, "Butterfly Factorization via Randomized Matrix-Vector Multiplications," *SIAM Journal on Scientific Computing* 43, no. 2 (2021): A883–A907.

26. E. Michielssen and A. Boag, "A Multilevel Matrix Decomposition Algorithm for Analyzing Scattering From Large Structures," *IEEE Transactions on Antennas and Propagation* 44, no. 8 (1996): 1086–1093.

27. P. M. Kielstra, T. Shi, H. Luo, J. Qian, and Y. Liu, "A Linear-Complexity Tensor Butterfly Algorithm for Compressing High-Dimensional Oscillatory Integral Operators," *Multiscale Modeling and Simulation* 23, no. 2 (2025): 864–893, https://doi.org/10.1137/24M1707821.

28. J. Bremer, Z. Chen, and H. Yang, "Rapid Application of the Spherical Harmonic Transform via Interpolative Decomposition Butterfly Factorization," *SIAM Journal on Scientific Computing* 43, no. 6 (2021): A3789–A3808.

29. M. O'Neil, F. Woolfe, and V. Rokhlin, "An Algorithm for the Rapid Evaluation of Special Function Transforms," *Applied and Computational Harmonic Analysis* 28, no. 2 (2010): 203–226 Special Issue on Continuous Wavelet Transform in Memory of Jean Morlet, Part I.

30. M. Tygert, "Fast Algorithms for Spherical Harmonic Expansions, III," *Journal of Computational Physics* 229, no. 18 (2010): 6181–6192.

31. H. Yang, "A Unified Framework for Oscillatory Integral Transforms: When to Use NUFFT or Butterfly Factorization?," *Journal of Computational Physics* 388 (2019): 103–122.

32. L. Ying, "Sparse Fourier Transform via Butterfly Algorithm," *SIAM Journal on Scientific Computing* 31, no. 3 (2009): 1678–1694.

33. H. Guo, Y. Liu, H. Jun, and E. Michielssen, "A Butterfly-Based Direct Integral-Equation Solver Using Hierarchical LU Factorization for Analyzing Scattering From Electrically Large Conducting Objects," *IEEE Transactions on Antennas and Propagation* 65, no. 9 (2017): 4742–4750.

34. Y. Liu, H. Guo, and E. Michielssen, "An HSS Matrix-Inspired Butterfly-Based Direct Solver for Analyzing Scattering From Two-Dimensional Objects," *IEEE Antennas and Wireless Propagation Letters* 16 (2017): 1179–1183.

35. Y. Liu, J. Song, R. Burridge, and J. Qian, "A Fast Butterfly-Compressed Hadamard-Babich Integrator for High-Frequency Helmholtz Equations in Inhomogeneous Media With Arbitrary Sources," *Multiscale Modeling and Simulation* 21, no. 1 (2023): 269–308.

36. S. B. Sayed, Y. Liu, L. J. Gomez, and A. C. Yucel, "A Butterfly-Accelerated Volume Integral Equation Solver for Broad Permittivity and Large-Scale Electromagnetic Analysis," *IEEE Transactions on Antennas and Propagation* 70, no. 5 (2022): 3549–3559, https://doi.org/10.1109/TAP.2021.3137193.

37. L. Claus, P. Ghysels, Y. Liu, et al., "Sparse Approximate Multifrontal Factorization With Composite Compression Methods," *ACM Transactions on Mathematical Software* 49, no. 3 (2023): 1–28.

38. Y. Liu, P. Ghysels, L. Claus, and X. S. Li, "Sparse Approximate Multifrontal Factorization With Butterfly Compression for High-Frequency Wave Equations," *SIAM Journal on Scientific Computing* 43, no. 5 (2021): S367–S391.

39. B. Chen, T. Dao, K. Liang, et al., "Pixelated Butterfly: Simple and Efficient Sparse Training for Neural Network Models," in *International Conference on Learning Representations*, (2022), https://openreview.net/forum?id=Nfl-iXa-y7R.

40. T. Dao, A. Gu, M. Eichhorn, A. Rudra, and C. Ré, "Learning Fast Algorithms for Linear Transforms Using Butterfly Factorizations," in *International Conference on Machine Learning (ICML)*, (2019), https://arxiv.org/abs/1903.05895.

41. T. Dao, N. S. Sohoni, A. Gu, et al., "Kaleidoscope: An Efficient, Learnable Representation for All Structured Linear Maps," *arXiv preprint arXiv:2012.14966*, (2020).

42. T. Dao, B. Chen, N. S. Sohoni, et al., "Monarch: Expressive Structured Matrices for Efficient and Accurate Training," in *Proceedings of the 39th International Conference on Machine Learning, Volume 162 of Proceedings of Machine Learning Research*, ed. K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu and S. Sabato, 4690–4721, 17–23 Jul 2022. PMLR, https://proceedings.mlr.press/v162/dao22a.html.

43. H. Fan, T. Chau, S. I. Venieris, et al., "Adaptable Butterfly Accelerator for Attention-Based Nns via Hardware and Algorithm Co-Design, (2022)," https://arxiv.org/abs/2209.09570.

44. A. Gonon, L. Zheng, P. Carrivain, and Q.-T. Le, "Make Inference Faster: Efficient GPU Memory Management for Butterfly Sparse Matrix Multiplication," *arXiv preprint arXiv:2405.15013*, (2024).

45. Y. Khoo and L. Ying, "SwitchNet: A Neural Network Model for Forward and Inverse Scattering Problems," *SIAM Journal on Scientific Computing* 41, no. 5 (2019): A3182–A3201, https://doi.org/10.1137/18M1222399.

46. M. Li, L. Demanet, and L. Zepeda-Núñez, "Wide-Band Butterfly Network: Stable and Efficient Inversion via Multi-Frequency Neural Networks," *Multiscale Modeling and Simulation* 20, no. 4 (2022): 1191–1227.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

47. R. Lin, J. Ran, K. H. Chiu, G. Chesi, and N. Wong, "Deformable Butterfly: A Highly Structured and Sparse Linear Transform," in *Advances in Neural Information Processing Systems*, vol. 34, ed. M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Curran Associates, Inc., 2021), 16145–16157, https://proceedings.neurips.cc/paper_files/paper/2021/file/86b122d4358357d834a87ce618a55de0-Paper.pdf.

48. W. Liu, Z. Qiu, Y. Feng, et al., "Parameter-Efficient Orthogonal Finetuning via Butterfly Factorization," in *International Conference on Learning Representations (ICLR)*, (2024), https://arxiv.org/abs/2311.06243.

49. B. N. Khoromskij, "O (d log N)-quantics Approximation of N-d Tensors in High-Dimensional Numerical Modeling," *Constructive Approximation* 34, no. 2 (2011): 257–280.

50. Q.-T. Le, L. Zheng, E. Riccietti, and R. Gribonval, "Butterfly Factorization with Error Guarantees," *SIAM Journal on Matrix Analysis and Applications* 46, no. 4 (2025): 2253–2309, https://arxiv.org/abs/2411.04506.

51. C. J. Hillar and L.-H. Lim, "Most Tensor Problems Are NP–Hard," *Journal of the ACM* JACM 60, no. 6 (2013): 1–39.

52. J. Chen and M. Lindsey, "Direct Interpolative Construction of the Discrete Fourier Transform as a Matrix Product Operator," *arXiv preprint arXiv:2404.03182*, (2024).

53. S. Dolgov, B. Khoromskij, and D. Savostyanov, "Superfast Fourier Transform Using QTT Approximation," *Journal of Fourier Analysis and Applications* 18, no. 5 (2012): 915–953.

54. C. Hubig, I. P. McCulloch, and U. Schollwöck, "Generic Construction of Efficient Matrix Product Operators," *Physical Review B* 95, no. 3 (2017): 035129.

55. I. Oseledets, E. Tyrtyshnikov, and N. Zamarashkin, "Tensor-Train Ranks for Matrices and Their Inverses," *Computational Methods in Applied Mathematics* 11, no. 3 (2011): 394–403.

56. I. V. Oseledets, "Approximation of $2^d \times 2^d$ Matrices Using Tensor Decomposition," *SIAM Journal on Matrix Analysis and Applications* 31, no. 4 (2010): 2130–2145.

57. S. Keller, M. Dolfi, M. Troyer, and M. Reiher, "An Efficient Matrix Product Operator Representation of the Quantum Chemical Hamiltonian," *Journal of Chemical Physics* 143, no. 24 (2015): 244118.

58. P. Gelß, S. Klus, S. Knebel, Z. Shakibaei, and S. Pokutta, "Low-Rank Tensor Decompositions of Quantum Circuits," *SIAM Journal on Mathematics of Data Science* 6, no. 4 (2024): 1078–1108, https://doi.org/10.1137/23M1503296.

59. E. Corona, A. Rahimian, and D. Zorin, "A Tensor-Train Accelerated Solver for Integral Equations in Complex Geometries," *Journal of Computational Physics* 334 (2017): 145–169, https://doi.org/10.1016/j.jcp.2016.12.051.

60. S. V. Dolgov, B. N. Khoromskij, and I. V. Oseledets, "Fast Solution of Parabolic Problems in the Tensor Train/Quantized Tensor Train Format With Initial Application to the Fokker–Planck Equation," *SIAM Journal on Scientific Computing* 34, no. 6 (2012): A3016–A3038.

61. P. Gelß, R. Klein, S. Matera, and B. Schmidt, "Solving the Time-Independent Schrödinger Equation for Chains of Coupled Excitons and Phonons Using Tensor Trains," *Journal of Chemical Physics* 156, no. 2 (2022): 24109.

62. G. Manzini, P. M. D. Truong, R. Vuchkov, and B. Alexandrov, "The Tensor-Train Mimetic Finite Difference Method for Three-Dimensional Maxwell's Wave Propagation Equations," *Mathematics and Computers in Simulation* 210 (2023): 615–639.

63. E. Ye and N. F. Loureiro, "Quantized Tensor Networks for Solving the Vlasov–Maxwell Equations," *Journal of Plasma Physics* 90, no. 3 (2024): 805900301.

64. T. Hastie, R. Mazumder, J. D. Lee, and R. Zadeh, "Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares. *The*," *Journal of Machine Learning Research* 16, no. 1 (2015): 3367–3402.

65. P. Jain, P. Netrapalli, and S. Sanghavi, "Low-Rank Matrix Completion Using Alternating Minimization," in *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, 665–674, (2013).

66. B. Vandereycken, "Low-Rank Matrix Completion by Riemannian Optimization," *SIAM Journal on Optimization* 23, no. 2 (2013): 1214–1236.

67. M. Filipović and A. Jukić, "Tucker Factorization With Missing Data With Application to Low-n-Rank Tensor Completion," *Multidimensional Systems and Signal Processing* 26, no. 3 (2015): 677–692.

68. J.-F. Cai, J. Li, and D. Xia, "Provable Tensor-Train Format Tensor Completion by Riemannian Optimization," *Journal of Machine Learning Research* 23, no. 123 (2022): 1–77.

69. M. Steinlechner, "Riemannian Optimization for High-Dimensional Tensor Completion," *SIAM Journal on Scientific Computing* 38, no. 5 (2016): S461–S484.

70. C. Da Silva and F. J. Herrmann, "Optimization on the Hierarchical Tucker Manifold–Applications to Tensor Completion," *Linear Algebra and Its Applications* 481 (2015): 131–173.

71. H. Rauhut, R. Schneider, and Ž. Stojanac, "Tensor Completion in Hierarchical Tensor Representations," in *Compressed Sensing and Its Applications: MATHEON Workshop 2013*, 419–450. (Springer, 2015).

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

72. W. Wang, V. Aggarwal, and S. Aeron, "Efficient Low Rank Tensor Ring Completion," in *Proceedings of the IEEE International Conference on Computer Vision*, 5697–5705, (2017).

73. S. Dolgov, D. Kressner, and C. Strossner, "Functional Tucker Approximation Using Chebyshev Interpolation," *SIAM Journal on Scientific Computing* 43, no. 3 (2021): A2190–A2210.

74. I. Oseledets and E. Tyrtyshnikov, "TT–Cross Approximation for Multidimensional Arrays," *Linear Algebra and Its Applications* 432, no. 1 (2010): 70–88.

75. A. K. Saibaba, "HOID: Higher Order Interpolatory Decomposition for Tensors Based on Tucker Representation," *SIAM Journal on Matrix Analysis and Applications* 37, no. 3 (2016): 1223–1249.

76. B. W. Bader and T. G. Kolda, "Efficient MATLAB Computations With Sparse and Factored Tensors," *SIAM Journal on Scientific Computing* 30, no. 1 (2008): 205–231.

77. E. Solomonik, D. Matthews, J. Hammond, and J. Demmel, "Cyclops Tensor Framework: Reducing Communication and Eliminating Load Imbalance in Massively Parallel Contractions," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 813–824. (IEEE, 2013).

78. L. Ma and E. Solomonik, "Efficient Parallel CP Decomposition With Pairwise Perturbation and Multi-Sweep Dimension Tree," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 412–421. (IEEE, 2021).

79. L. Ma and E. Solomonik, "Accelerating Alternating Least Squares for Tensor Decomposition by Pairwise Perturbation," *Numerical Linear Algebra with Applications* 29, no. 4 (2022): e2431.

80. N. Singh and E. Solomonik, "Alternating Mahalanobis Distance Minimization for Accurate and Well-Conditioned CP Decomposition," *SIAM Journal on Scientific Computing* 45, no. 6 (2023): A2781–A2812.

81. N. Singh, L. Ma, H. Yang, and E. Solomonik, "Comparison of Accuracy and Scalability of Gauss–Newton and Alternating Least Squares for CANDECOMC/PARAFAC Decomposition," *SIAM Journal on Scientific Computing* 43, no. 4 (2021): C290–C311.

82. N. Singh, Z. Zhang, X. Wu, N. Zhang, S. Zhang, and E. Solomonik, "Distributed-Memory Tensor Completion for Generalized Loss Functions in Python Using New Sparse Tensor Kernels," *Journal of Parallel and Distributed Computing* 169 (2022): 269–285.

83. R. Kanakagiri and E. Solomonik, "Minimum Cost Loop Nests for Contraction of a Sparse Tensor With a Tensor Network," in *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, 169–181, (2024).

84. S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 61–70. (IEEE, 2015).

85. S. Smith and G. Karypis, "Tensor-Matrix Products With a Compressed Sparse Tensor," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, 1–7, (2015).

86. G. Ballard, A. Klinvex, and T. G. Kolda, "Tuckermpi: A Parallel C++/MPI Software Package for Large-Scale Data Compression Based on the Tucker Tensor Decomposition," *ACM Transactions on Mathematical Software (TOMS)* 46 (2019): 1–31, https://doi.org/10.1145/3378445.

87. J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor Learning in Python," *Journal of Machine Learning Research* 20, no. 26 (2019): 1–6, https://jmlr.org/papers/v20/18-277.html.

88. Q. Pang, K. L. Ho, and H. Yang, "Interpolative Decomposition Butterfly Factorization," *SIAM Journal on Scientific Computing* 42, no. 2 (2020): A1097–A1115, https://doi.org/10.1137/19M1294873.

89. E. Candes, L. Demanet, and L. Ying, "Fast Computation of Fourier Integral Operators," *SIAM Journal on Scientific Computing* 29, no. 6 (2007): 2464–2493.

90. E. Michielssen and A. Boag, "Multilevel Evaluation of Electromagnetic Fields for the Rapid Solution of Scattering Problems," *Microwave and Optical Technology Letters* 7, no. 17 (1994): 790–795.

91. Q.-T. Le, L. Zheng, E. Riccietti, and R. Gribonval, "Fast Learning of Fast Transforms, With Guarantees," in *ICASSP 2022–2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3348–3352, (2022), https://doi.org/10.1109/ICASSP43922.2022.9747791.

92. S. B. Davis and P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28, no. 4 (1980): 357–366, https://doi.org/10.1109/TASSP.1980.1163420.

93. P. Paatero, "A Weighted Non-Negative Least Squares Algorithm for Three-Way 'PARAFAC' Factor Analysis," *Chemometrics and Intelligent Laboratory Systems* 38, no. 2 (1997): 223–242.

94. S. Smith, J. Park, and G. Karypis, "An Exploration of Optimization Algorithms for High Performance Tensor Completion," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 359–371. (IEEE, 2016).

95. L. Grasedyck, M. Kluge, and S. Krämer, *Alternating Directions Fitting (ADF) of Hierarchical Low Rank Tensors* (Verlag nicht ermittelbar, 2013).

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

96. C. Mu, B. Huang, J. Wright, and D. Goldfarb, "Square Deal: Lower Bounds and Improved Relaxations for Tensor Recovery," in *International Conference on Machine Learning* (PMLR, 2014), 73–81.

97. B. Recht, "A Simpler Approach to Matrix Completion," *Journal of Machine Learning Research* 12, no. 12 (2011).

98. P. Jain and P. Netrapalli, "Fast Exact Matrix Completion With Finite Samples," in *Conference on Learning Theory* (PMLR, 2015), 1007–1034.

99. J. A. Kelner, J. Li, A. Liu, A. Sidford, and K. Tian, "Matrix Completion in Almost-Verification Time," in *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, 2102–2128. (IEEE, 2023).

100. S. Budzinskiy and N. Zamarashkin, "Tensor Train Completion: Local Recovery Guarantees via Riemannian Optimization," *Numerical Linear Algebra with Applications* 30, no. 6 (2023): e2520.

101. C. Da Silva and F. Herrmann, "A Unified 2D/3D Large-Scale Software Environment for Nonlinear Inverse Problems," *ACM Transactions on Mathematical Software (TOMS)* 45, no. 1 (2019): 1–35, https://doi.org/10.1145/3291042.

Appendix A: Efficient Implementation of ALS for Butterfly Completion

Implementation of the ALS algorithm for butterfly completion as described in Algorithm 1 efficiently can be a challenging task when the number of levels of the butterfly decomposition increase. This is due to computations with high-order parameter tensors with their order depending on the number of levels in the butterfly decomposition. In this section, we describe a method to efficiently implement ALS for butterfly completion and QTT completion with a single software kernel that involves sparse tensor computations. This method also allows for efficient sampling of entries of a butterfly matrix (section 3.4 of [38]). Our implementation is in Python and is open-source.[2]

ALS for QTT Completion

Firstly, we present ALS for QTT format in Algorithm 5. Similar to butterfly completion, Algorithm 5 proceeds by taking the observed input tensor, rank r (assumed to be the same for all tensors here, but can be easily generalized to different ranks), initial guess for the tensors, and a parameter for controlling the maximum number of iterations if the convergence criterion is not satisfied. For the outermost tensors, $\boldsymbol{\mathcal{S}}^1$ and $\boldsymbol{\mathcal{S}}^{L+1}$, the algorithm solves for each fiber (potentially in parallel) requiring an $r \times r$ positive semi-definite (PSD) solve. The steps to compute the left and right-hand sides of the systems are similar to Algorithm 1, however, the L matrix-vector products are indexed differently and are given in lines 19 and 9. Updating the outermost tensors requires at most L matrix-vector products for each nonzero with a total cost $\mathcal{O}(|\Omega| L r^2)$ (including the outer product) and solving for each fiber of the tensor requires $O(r^3)$ as there are only 4 fibers to be solved for. For the inner tensors, $\left\{\boldsymbol{\mathcal{S}}^i : i \in \{2, \dots, L\}\right\}$, the algorithm solves for each slice requiring $r^2 \times r^2$ PSD solve. The left and right-hand sides of the systems are computed in a similar manner as in Algorithm 1 except for the matrix-vector products which are given in lines 29 and 30. Similar to Algorithm 1, the result is reshaped into a matrix in row-major ordering to update the inner tensors. The total cost of performing the computations in lines 29 and 30 is $\mathcal{O}(|\Omega| L r^2)$, and the total cost of forming the left-hand sides $\mathcal{O}\left(|\Omega|(L r^2 + r^4)\right)$, which includes the outer products of Kronecker products in line 32. The solve can be performed in a total cost of $\mathcal{O}(r^6)$. Since the number of tensors to be solved is L, the total cost of one iteration of Algorithm 5 is $\mathcal{O}\left(|\Omega| L(L r^2 + r^4) + L r^6\right)$.

It should be noted that the Algorithm 5 is exactly the same as Algorithm 1 except for the indexing of the matrix-vector products used to form the left and right-hand sides. Therefore, by introducing an indexing on the input tensor entries such that the outer tensors are reshaped into matrices and the inner tensors are reshaped into order 3 tensors, one can use exactly the same software kernel for tensor train, QTT and butterfly completion. This also means that the computational benefits of storing the sparse tensor entries in the compressed sparse fiber (CSF) format [85] to amortize the matrix-vector products involved in each solve can be translated to the butterfly completion algorithms. We describe this indexing in the next subsection.

Indexing for Butterfly Completion Algorithms

To reshape the higher-order tensors in tensorized butterfly factorization such that the outermost tensors are represented as matrices and the inner tensors are represented as order 3 tensors, we use the inverse of the function defined in (4.2). Precisely, we use a function $\psi^{-1}(\cdot)$ that takes a p-bit binary number as input and outputs an integer given as

$$\psi^{-1}(b_0, \dots, b_{p-1}) = \sum_{i=0}^{p-1} b_i \; 2^{p-1-i}. \tag{A1}$$

Therefore, a new list of factors $\mathcal{Z} = \left\{\mathcal{Z}^l : l \in \{1, \dots, L+2\}\right\}$ can be formulated based on the above reshaping, by reshaping the $\boldsymbol{\mathcal{S}}^l$ introduced in (4.4) as

$$\begin{aligned} \boldsymbol{z}^1\left(k_1, :\right) &= \boldsymbol{s}^1(i_0, \dots, i_L, :) \\ \boldsymbol{z}^{L+2}\left(k_{L+2}, :\right) &= \boldsymbol{s}^{L+2}(j_0, \dots, j_L, :) \\ \boldsymbol{Z}^{m+1}\left(k_{m+1}, :, :\right) &= \boldsymbol{S}^{m+1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1}, :, :), \end{aligned} \tag{A2}$$

where $m \in \{1, \dots, L\}$, $k_1 = \psi^{-1}(i_0, \dots, i_L)$, $k_{L+2} = \psi^{-1}(j_0, \dots, j_L)$, and $k_{m+1} = \psi^{-1}(i_0, \dots i_{L-m}, j_0, \dots, j_{m-1})$.

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

ALGORITHM 5 | ALS for QTT completion.

Input: Observed matrix $\mathcal{T}_\Omega$, rank r, initial guess $\mathcal{S}^i, i = 1, \dots, L+1$, max iterations $t_{\max}$.
Output: Updated QTT cores $\mathcal{S}^i, i = 1, \dots, L+1$.
for $t = 1$ to $t_{\max}$ **do**
  for $l = 1$ to $L+1$ **do**
    if $l = 0$ **then**
      for each (i_0, j_0) **do**
        Initialize $\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{K} = \boldsymbol{0}$
        for each $(i_1, \dots, i_L, j_1, \dots, j_L) \in \Omega_{i_0, j_0}$ **do**
          $\boldsymbol{v} = \left(\prod_{m=1}^{L-1} \boldsymbol{S}^{m+1}(i_m, j_m, :, :)\right) \boldsymbol{s}^{L+1}(i_L, j_L, :)$
          Update $\boldsymbol{y}$ += $t_{i_0 \dots i_L j_0 \dots j_L} \boldsymbol{v}$
          Update $\boldsymbol{K}$ += $\boldsymbol{v}\boldsymbol{v}^T$
        end for
        Solve $\boldsymbol{K}\boldsymbol{s} = \boldsymbol{y}$ and update $\boldsymbol{s}^1(i_0, j_0, :) \leftarrow \boldsymbol{s}$ ▷ $\boldsymbol{K} \in \mathbb{R}^{r \times r}$
      end for
    else if $l = L+1$ **then**
      for each (i_L, j_L) **do**
        Initialize $\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{K} = \boldsymbol{0}$
        for each $(i_0, \dots, i_{L-1}, j_0, \dots, j_{L-1}) \in \Omega_{i_L, j_L}$ **do**
          $\boldsymbol{u} = \boldsymbol{s}^1(i_L, j_L, :)^T \left(\prod_{m=1}^{L-1} \boldsymbol{S}^{m+1}(i_m, j_m, :, :)\right)$
          Update $\boldsymbol{y}$ += $t_{i_0 \dots i_L j_0 \dots j_L} \boldsymbol{u}$
          Update $\boldsymbol{K}$ += $\boldsymbol{u}\boldsymbol{u}^T$
        end for
        Solve $\boldsymbol{K}\boldsymbol{s} = \boldsymbol{y}$ and update $\boldsymbol{s}^{L+1}(i_L, j_L, :) \leftarrow \boldsymbol{s}$ ▷ $\boldsymbol{K} \in \mathbb{R}^{r \times r}$
      end for
    else
      for each (i_{l-1}, j_{l-1}) **do**
        Initialize $\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{K} = \boldsymbol{0}$
        for each $(i_0, \dots, i_{l-2}, i_l, \dots, i_L, j_0, \dots, j_{l-2}, j_l, \dots, j_L)$ **do**
          $\boldsymbol{u} = \boldsymbol{s}^1(i_0, j_0, :)^T \left(\prod_{m=1}^{l-2} \boldsymbol{S}^{m+1}(i_m, j_m, :, :)\right)$
          $\boldsymbol{v} = \left(\prod_{m=l}^{L-1} \boldsymbol{S}^{m+1}(i_m, j_m, :, :)\right) \boldsymbol{s}^{L+1}(i_L, j_L, :)$
          Update $\boldsymbol{y}$ += $t_{i_0 \dots i_L j_0 \dots j_L} (\boldsymbol{v} \otimes \boldsymbol{u})$
          Update $\boldsymbol{K}$ += $(\boldsymbol{v} \otimes \boldsymbol{u})(\boldsymbol{v} \otimes \boldsymbol{u})^T$
        end for
        Solve $\boldsymbol{K}\boldsymbol{s} = \boldsymbol{y}$ ▷ $\boldsymbol{K} \in \mathbb{R}^{r^2 \times r^2}$
        Update $\boldsymbol{S}^l(i_{l-1}, j_{l-1}, :, :) := \text{reshape}(\boldsymbol{s}, r, r)$
      end for
    end if
  end for
end for

By using order 2 and order 3 tensor representations $\mathcal{Z}$ that adhere to the tensor train format, any tensor-train completion algorithm can be used to optimize the factors. Note that similar reshapes can be defined for the QTT format as well to utilize the implementation of tensor-train completion algorithms. We use the above reshapes for implementing Algorithms 1 and 5 with a tensor-train completion optimization algorithm. In our implementation of ALS for tensor-train completion, we transpose the sparse input tensor according to the index being solved for. This results in increasing the computational efficiency by using BLAS-3 instead of BLAS-2 operations to form the left-hand sides (see [82]) without constructing the CSF trees. The software kernel can further be optimized by using the computational trees which would allow for further increase in computational efficiency by reusing the matrix-vector products needed to form left and right-hand sides (see [94]).

Orthogonalization Sweeps for Butterfly ADF

To apply ADF [18, 95] to the butterfly factors $\mathcal{S}$, we require orthogonalization sweeps that produce orthonormal matrices in the local least-squares systems. In the higher-order tensor representation $\mathcal{S}^l$ (4.4), orthogonalization at level $l+1$ proceeds by folding the outermost mode that is not shared with the adjacent factor into the rank dimension, forming a tall (or wide) matrix, and computing its QR (or LQ) factorization. The resulting triangular factor $\boldsymbol{R}$ (or $\boldsymbol{L}$) is then absorbed into the adjacent factor so that the overall product is preserved. In the reshaped representation where we only have order 2 and 3 tensors (A2), the modes of $\mathcal{Z}^{m+1}$ are encoded in the binary digits of the index $k_{m+1} = \psi^{-1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1})$, so folding a mode corresponds to pairing matrices that differ at a specific bit position in k_{m+1} and concatenating them along the appropriate rank axis. We describe the QR factorization and absorption steps below

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

ALGORITHM 6 | ADF for butterfly completion.

Input: Observed matrix $\boldsymbol{T}_\Omega$ (or its tensorization $\boldsymbol{\mathcal{T}}_\Omega$), rank r, initial guess $\boldsymbol{\mathcal{S}}^i, i = 1, \dots, L+2$, max iterations $t_{\max}$, and convergence threshold ϵ.
Output: Updated butterfly cores $\boldsymbol{\mathcal{S}}^i, i = 1, \dots, L+2$.
$\mathcal{Z} \leftarrow RIGHTTOLEFTSWEEP(\mathcal{Z}, L)$ ▷ LQ sweep: concentrate weight in $\boldsymbol{\mathcal{Z}}^1$
$\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{T}}_\Omega - RECONSTRUCT_\Omega(\mathcal{Z})$ ▷ Initial residual on Ω
for iter $= 1, \dots, t_{\max}$ **do**
  for $l = 1, 2, \dots, L+2$ **do**
    $\boldsymbol{\mathcal{N}} \leftarrow GRADIENT(\boldsymbol{\mathcal{G}}, \Omega, \mathcal{Z}, l, L)$ ▷ As in Algorithm 5.3
    $\mathcal{Z} \leftarrow RECONSTRUCT_\Omega(\mathcal{Z}|_{\boldsymbol{\mathcal{Z}}^l \leftarrow \boldsymbol{\mathcal{N}}})$
    $\alpha_s \leftarrow \|\boldsymbol{\mathcal{N}}_{(s)}\|_F^2 \,/\, \|z\|_2^2$ ▷ Overrelaxation for each slice asin [31]
    $\boldsymbol{\mathcal{Z}}^l_s \leftarrow \boldsymbol{\mathcal{Z}}^l_s + \alpha_s \boldsymbol{\mathcal{N}}_s$ ▷ Slice-wise update
    $\boldsymbol{\mathcal{G}}_s \leftarrow \boldsymbol{\mathcal{G}}_s - \alpha_s \boldsymbol{\mathcal{Z}}_s$ ▷ Slice-wise update
    if $l \le L+1$ **then** ▷ Batched QR and absorb into next level
      $\boldsymbol{\mathcal{Z}}^l, \boldsymbol{R} \leftarrow QR(\boldsymbol{\mathcal{Z}}^l, L, l)$
      $\boldsymbol{\mathcal{Z}}^{l+1} \leftarrow ABSORB(\boldsymbol{R}, \boldsymbol{\mathcal{Z}}^{l+1}, L, l)$
    end if
  end for
  if $\|g\|_2 / \|\boldsymbol{\mathcal{T}}_\Omega\|_F < \epsilon$ **then**
    break
  end if
end for

for a left-to-right sweep; the right-to-left sweep is analogous and can be implemented by transposing each factor, performing the same QR-based procedure in reverse order, and transposing back.

QR Factorization For the outer factor $\boldsymbol{\mathcal{Z}}^1$, we perform independent reduced QR factorizations of each of the 2^L blocks of size $(c \times r)$, yielding orthonormal blocks $\boldsymbol{Q}_{k_1}$ and upper-triangular remainders $\boldsymbol{R}_{k_1}$. Here recall that $k_1 = \psi^{-1}(i_0, \dots, i_L)$. For an inner factor $\boldsymbol{\mathcal{Z}}^{m+1}$ with $m \in \{1, \dots, L\}$, the mode to be folded is i_{L-m}, since this is the outermost row index that appears in $\boldsymbol{\mathcal{Z}}^{m+1}$ but not in the next factor $\boldsymbol{\mathcal{Z}}^{m+2}$ (compare their index sets in (A2)). In the encoding $k_{m+1} = \psi^{-1}(i_0, \dots, i_{L-m}, j_0, \dots, j_{m-1})$, the index i_{L-m} occupies bit position m from the least significant bit. We therefore pair the 2^{L+1} matrices $\boldsymbol{Z}^{m+1}(k_{m+1}, :, :) \in \mathbb{R}^{r \times r}$ over bit position m in k_{m+1}: for each pair $(k^{(0)}, k^{(1)})$ differing only at bit m (with $k^{(0)}$ having bit m equal to 0 and $k^{(1)}$ having bit m equal to 1), we concatenate along the left-rank axis to form a $(2r \times r)$ matrix and compute its reduced QR factorization. The two halves of $\boldsymbol{Q}$ are assigned back to indices $k^{(0)}$ and $k^{(1)}$, yielding 2^{L+1} matrices with pairwise orthonormal columns, while the remainder $\boldsymbol{R} \in \mathbb{R}^{r \times r}$ must be absorbed into the adjacent factor $\boldsymbol{\mathcal{Z}}^{m+2}$.

Absorption of $\boldsymbol{R}$

Each remainder $\boldsymbol{R}$ is left-multiplied into the matrices of $\boldsymbol{\mathcal{Z}}^{m+2}$ that share the same remaining indices. Comparing the index sets of $\boldsymbol{\mathcal{Z}}^{m+1}$ and $\boldsymbol{\mathcal{Z}}^{m+2}$, the transition from level $m+1$ to $m+2$ drops i_{L-m} and introduces j_m. The new index j_m occupies bit position 0 (the least significant bit) in $k_{m+2} = \psi^{-1}(i_0, \dots, i_{L-m-1}, j_0, \dots, j_m)$. Each remainder $\boldsymbol{R}$ from the QR step therefore multiplies both matrices of the pair $(k^{(0)}, k^{(1)})$ differing at bit 0 in k_{m+2}:

$$\boldsymbol{Z}^{m+2}(k^{(b)}, :, :) \leftarrow \boldsymbol{R}\ \boldsymbol{Z}^{m+2}(k^{(b)}, :, :), \quad b \in \{0, 1\}. \tag{A3}$$

When the target is the right outer factor $\boldsymbol{Z}^{L+2}$, each block $\boldsymbol{Z}^{L+2}_{k_{L+2}}$ of size $(r \times c)$ is instead updated as $\boldsymbol{Z}^{L+2}_{k_{L+2}} \leftarrow \boldsymbol{R}_{k_{L+2}}\ \boldsymbol{Z}^{L+2}_{k_{L+2}}$.

The complete ADF algorithm for butterfly matrix completion is summarized in Algorithm 6. The gradient computation is the same as that used in Algorithm 3 and the optimal step size for each slice of the computed tensors are computed as suggested in [18].

10991506, 2026, 4, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/nla.70098, Wiley Online Library on [10/11/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License