Randomized tensor decomposition using parallel reconfigurable systems

Misra, Ajita; Abdelgawad, Muhammad A. A.; Jing, Peng; Cheung, Ray C. C.; Yan, Hong

Published in:
Journal of Supercomputing

Published: 01/03/2025

Document Version:
Final Published version, also known as Publisher's PDF, Publisher's Final version or Version of Record

Published version (DOI):
10.1007/s11227-025-07049-5

Publication details:
Misra, A., Abdelgawad, M. A. A., Jing, P., Cheung, R. C. C., & Yan, H. (2025). Randomized tensor decomposition using parallel reconfigurable systems. *Journal of Supercomputing*, *81*(4), Article 543. https://doi.org/10.1007/s11227-025-07049-5

The Journal of Supercomputing (2025) 81:543
https://doi.org/10.1007/s11227-025-07049-5

Randomized tensor decomposition using parallel reconfigurable systems

Ajita Misra[1] · Muhammad A. A. Abdelgawad[1] · Peng Jing[1] · Ray C. C. Cheung[1] · Hong Yan[1]

Accepted: 12 February 2025

Abstract

Tensor decomposition algorithms are essential for extracting meaningful latent variables and uncovering hidden structures in real-world data tensors. Unlike conventional deterministic tensor decomposition algorithms, randomized methods offer higher efficiency by reducing memory requirements and computational complexity. This paper proposes an efficient hardware architecture for a randomized tensor decomposition implemented on a field-programmable gate array (FPGA) using high-level synthesis (HLS). The proposed architecture integrates random projection, power iteration, and subspace approximation via QR decomposition to achieve low-rank approximation of multidimensional datasets. The proposed architecture utilizes the capabilities of reconfigurable systems to accelerate tensor computation. It includes three central units: (1) tensor times matrix chain (TTMc), (2) tensor unfolding unit, and (3) QR decomposition unit to implement a three-stage algorithm. Experimental results demonstrate that our FPGA design achieves up to 14.56 times speedup compared to the well-implemented tensor decomposition using software library Tensor Toolbox on an Intel i7-9700 CPU. For a large input tensor of size $512 \times 512 \times 512$, the proposed design achieves a 5.55 times speedup compared to an Nvidia Tesla T4 GPU. Furthermore, we utilize our hardware-based high-order singular value decomposition (HOSVD) accelerator for two real applications: background subtraction of dynamic video datasets and data compression. In both applications, our proposed design shows high efficiency regarding accuracy and computational time.

Keywords High-level synthesis (HLS) · Field programmable gate array (FPGA) · Randomized algorithm · Low-rank tensor computing · High order singular value decomposition (HOSVD)

Ajita Misra, Muhammad A. A. Abdelgawad, and Peng Jing have contributed equally to this work.

Extended author information available on the last page of the article

Published online: 25 February 2025

1 Introduction

The recent advancements have resulted in exponential growth in the use of data for various applications such as computer vision, signal processing, recommender systems, and dimension reduction [1, 2]. This has led to a significant issue known as the "curse of dimensionality." An N-mode or N-th order tensor (high-dimensional array) is an N-dimensional data array [3]. In some instances where matrix representation might not be adequate, tensor computation is utilized to handle high-dimensional datasets [4]. For instance, a video can be represented as a tensor, where the dimensions corresponding to the image or pixel values account for two dimensions, while time represents the third dimension. This forms a third-order tensor that can be decomposed to extract essential features. Various studies have explored mathematical tensor decomposition models based on different combinations of data arrays. It has been shown that tensors can be expressed as outer products of vectors in canonical polyadic (CP) decomposition, which is effectively implemented on both CPU and GPU platforms [5].

Tucker decomposition is another tensor decomposition technique that is widely used for computing low-rank approximations of tensors. It is known for its accuracy and robustness, which decomposes the data tensor into factor matrices and a smaller core tensor, providing a more compact representation [6]. A particular case of Tucker decomposition is the high-order SVD (HOSVD), characterized by orthogonal factor matrices [7]. It generalizes singular value decomposition (SVD) to higher-order data. Within the framework of Tucker decomposition, several multilinear approximation algorithms exist, including higher-order orthogonal iteration (HOOI) [8], truncated HOSVD (THOSVD), and sequentially truncated HOSVD (STHOSVD) [9]. HOSVD has been observed to provide a better fit in most cases than other decomposition algorithms, making it the foundational structure for our algorithm.

Tucker decomposition can be computed using either deterministic or randomized approaches. Deterministic algorithms provide high approximation accuracy but are often prohibitive due to their significant memory and computational requirements [10]. Consequently, randomized approaches are commonly employed for large-scale datasets. These algorithms are advantageous because they (1) support parallelization and (2) process only portions of data tensors, thereby reducing memory requirements and minimizing communication between different memory levels-a common bottleneck in modern computing architectures. As a result, considerable interest has been in improving and accelerating randomized algorithms for Tucker decomposition and HOSVD [5, 7]. Although randomized tensor decomposition algorithms have been successfully applied across various fields, most existing hardware-based implementations rely on standard (deterministic) algorithms, which struggle with large datasets. We propose an efficient hardware architecture for a randomized tensor-based model to address this bottleneck. Our design enhances parallel processing capabilities and reduces communication between different design modules.

In this work, we propose a hardware accelerator for a randomized tensor-based model on FPGA using high-level synthesis (HLS). Our design integrates random

projection, power iteration, and subspace approximation via QR decomposition to compute the low-rank approximation of large-scale data tensors. The proposed accelerator addresses the tensor decomposition problem in its general form, defined as

$$\mathcal{A} \approx \mathcal{G} \times_1 \mathbf{Q}_1 \times_2 \mathbf{Q}_2 \times_3 \mathbf{Q}_3, \tag{1}$$

where a three-mode tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is decomposed to a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ multiplied by three orthonormal factor matrices $\mathbf{Q}_n \in \mathbb{R}^{I_n \times R_n}$, where $n = 1, 2, 3$ and R_n is the target rank along each mode. Unlike existing tensor decomposition accelerators, our design enhances efficiency by eliminating the need to compute the core tensor $\mathcal{G}$. Instead, our approach focuses on computing three orthonormal factor matrices $\mathbf{Q}_n$, thereby approximating the input tensor $\mathcal{A}$ as

$$\mathcal{A} \approx \mathcal{A} \times_1 (\mathbf{Q}_1 \mathbf{Q}_1^T) \times_2 (\mathbf{Q}_2 \mathbf{Q}_2^T) \times_3 (\mathbf{Q}_3 \mathbf{Q}_3^T) \tag{2}$$

The proposed design is a modified Tucker hardware implementation on the XCVU9P FPGA board, featuring three central units: (1) tensor times matrix chain (TTMc), (2) tensor unfolding unit, and (3) QR decomposition unit. These units work together to implement a three-stage architecture. Various hardware optimization techniques, such as HLS pragmas, have been employed to achieve high speed-up while maintaining moderate hardware resource utilization. Algorithmic implementations for large datasets and complex computations often suffer from high latency when executed on a CPU. This bottleneck can be alleviated by leveraging parallelism and pipelining to accelerate the implementation. Therefore, we utilize the proposed randomized-based tensor decomposition to address two critical real-life problems: video analysis and medical image compression. We employ incremental *N*-mode SVD, which uses tensor decomposition as a preprocessing step to create a hybrid model for background subtraction of dynamic videos. Qualitative analysis, higher F-scores, and lower error rates compared to state-of-the-art techniques demonstrate the system's capability to efficiently handle practical datasets. Additionally, medical datasets with multiple brain scans are efficiently compressed for storage. Extensive experiments on synthetic and real datasets demonstrate the efficiency of our proposed design for computing low-rank approximations. It achieves up to 14.56 times speedup over the well-optimized Tensor Toolbox software library on a CPU and is 5.55 times faster than a Tesla T4 GPU for large tensors ($512 \times 512 \times 512$).

Our design excels in accuracy and computational time for dynamic background subtraction and medical image compression, providing low-rank approximations with less error and significantly higher speed compared to state-of-the-art techniques.

The contributions of this paper are summarized as follows:

1. We propose an efficient FPGA-based implementation of a randomized tensor decomposition model using HLS and a random projection technique.
2. The proposed implementation features a tensor times matrix chain (TTMc) unit, a tensor unfolding unit, and a QR decomposition unit. These components form the

core of our modified Tucker decomposition architecture, which achieves superior performance and high speed.
3. Hardware analysis was conducted on various tensor sizes, comparing the results with popular, well-optimized tensor implementations on the CPU and GPU. With moderated hardware utilized resources, the proposed design achieves up to 14.56 times speedup over CPU implementations and 5.55 times speedup over GPU implementations.
4. The proposed design integrates the incremental N-mode SVD (INN-SVD) method for background subtraction, significantly enhancing performance compared to conventional models. It has also been tested for medical image compression, showing notable improvements in speed and accuracy over existing hardware implementations.

The remainder of the paper is organized as follows: Section 2 reviews related work. Section 3 introduces low-rank tensor approximation using randomized tensor decomposition. Section 4 details the proposed hardware design using HLS. Section 5 discusses the application of our design in two real-life scenarios. Section 6 presents experiments and performance comparisons. Section 7 concludes the paper and outlines future work.

1.1 Notation

The Calligraphic uppercase letters (e.g., $\mathcal{A}$, $\mathcal{B}$), bold uppercase letters (e.g., $\mathbf{A}$, $\mathbf{B}$), and bold lowercase letters (e.g., $\mathbf{a}$, $\mathbf{b}$) represent tensors, matrices, and vectors, respectively. Tensors are multidimensional arrays, with vectors and matrices being first-order and second-order tensors. The order of a tensor is the number of its modes. A fiber is a first-order tensor generated by fixing all modes except one, while a slice is a second-order tensor produced by fixing all modes except two. The symbols T and $\|\cdot\|_F$ denote the transpose and Frobenius norm, respectively.

2 Related work

Hardware implementation of tensor decomposition is a more challenging task compared with previous works on matrix decomposition acceleration [11]. High-level synthesis (HLS) is crucial for hardware accelerators, significantly reducing development time compared to hardware description languages (HDL). HLS uses higher-level languages like OpenCL, C, and C++. By bridging the gap between software and hardware, HLS inherently manages bus protocols, floating/fixed point implementations, and DRAM widths for different target boards [12, 13].

As emerging tensor decomposition algorithms are proposed and applied to different areas, more attempts have been made to accelerate tensor computation on the hardware platforms.

For instance, [14] proposes a high-level synthesis method for efficient systolic arrays for dense tensor kernels. Zhang et al. [15] implement Tucker decomposition

on an FPGA. Huang et al. [16] propose a hardware accelerator for singular spectrum analysis of Hankel tensors, performing Tucker decomposition, tensor reconstruction, and Hankelization. In [17], a high-performance tensor processor is proposed to accelerate both Tucker and CP decompositions. The hardware architecture of tensor-train decomposition (TTD) is presented in [18]. Additionally, several designs aim to accelerate sparse tensor data computation. For example, [19] proposes a reconfigurable memory system to reduce memory access time during sparse-MTTKRP. Srivastava et al. [20] leverage data sparsity to accelerate sparse Tucker decomposition using a hybrid FPGA-CPU platform. In [21], the authors proposed an FPGA-accelerated non-negative tensor factorization (NTF) algorithm. Several systolic architecture-based SVDs have been proposed in the literature. For instance, the authors in [22] presented an improved SVD systolic array and its implementation on FPGA, focusing on the Jacobi SVD algorithm. These works primarily address the optimization of small sub-matrix rotations in parallel. In contrast, our proposed TTMc architecture is designed to handle large-scale tensor times matrix chain operations, providing a different approach to matrix decomposition problems. However, most such hardware implementations use standard algorithms with limitations in handling complex datasets. We address this by proposing a modified Tucker decomposition with a randomization technique for better parallel processing. Our high-performance hardware architecture reduces inter-module communication and significantly speeds up compared to CPU and GPU implementations.

Designing a high-performance hardware accelerator for Tucker decomposition involves efficiently finding the dominant subspace of unfolding matrices. For a matrix $\mathbf{A}$ of size $m \times n$, the computational cost of standard SVD is $O(m^2n + n^2m + n^3)$, which is impractical for large-scale datasets or unfolded matrices of data tensors.

In the hardware domain, Jacobi-based SVD methods have been extensively employed due to their ease of parallel implementation. Tucker decomposition can also be computed using Jacobi iterations, as presented in [15], which is a fixed-point implementation. Additionally, the TTM unit presented in [15] mitigates resource constraints by reducing permutation operations, switching buffers in the PE design, and reducing the adder tree pipeline. However, this introduces latency overhead and data movement restrictions. Various SVD libraries have been developed by extending matrix operations [23], summarized in Table 1. The single-sided Jacobi method,

Table 1 Survey of the standard libraries used for SVD

SVD Library	Features
EIPACK	Initial stage of SVD development. Row major order
LINPACK	BLAS Level 1 library introduced
LAPACK	BLAS Levels 2 and 3 library introduced. Better memory management
ScaLAPACK	Extension of LAPACK. Matrix distribution as 2D cyclic block layout
PLASMA	More efficient for multicore implementation. Better performance with the tall skinny matrix
DPLASMA	More efficient for distributed systems. More dimensions can be handled

an extension of the two-sided implementation, includes modifications like dynamic, ring ordering, round-robin, and classical approaches to accelerate convergence [24].

While the aforementioned work significantly speeds up tensor decomposition compared to CPU-based implementations, there is still room for optimizing algorithms and hardware structures to enhance efficiency further. Most current hardware accelerators rely on standard tensor decomposition algorithms, which struggle with large and complex datasets. Additionally, the overall hardware architecture must be meticulously designed to minimize data traffic between modules and reduce hardware costs. The following sections introduce our more efficient hardware accelerator based on a modified Tucker decomposition. From a practical standpoint, the proposed tensor decomposition architecture is used to address key problems in real-life scenarios, such as background subtraction (BS) and medical image compression [25–27].

3 Randomized tensor decomposition

3.1 Tensor background

This section introduces the concepts of tensors used in the paper. Matrization, also known as unfolding or flattening, is the process of reshaping tensors into matrices. Several techniques for unfolding include mode-n unfolding and n-unfolding [28]. Given an N-th order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the n-unfolding matrix of the tensor $\mathcal{A}$ is denoted by $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N}$. Its components are defined as follows:

$$\mathbf{A}_{(n)}(i_n, j) = \mathcal{A}(i_1, i_2, \ldots, i_N),$$

where $j = 1 + \sum_{k=1, k \neq n}^{N} (i_k - 1) J_k$ and $J_k = \prod_{m=1, m \neq n}^{k-1} I_m$. The multiplication between tensors and matrices can be performed along different modes with matching dimensions, known as tensor-matrix multiplication (TMM) along mode n or n -mode product. This is a generalization of matrix-matrix multiplication. For a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\mathbf{B} \in \mathbb{R}^{J \times I_n}$, their multiplication along mode n is $\mathcal{A} \times_n \mathbf{B} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$, defined as:

$$\left(\mathcal{A} \times_n \mathbf{B}\right)_{i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N} = \sum_{i_n=1}^{I_n} a_{i_1, i_2, \ldots, i_N} b_{j, i_n},$$

for $j = 1, 2, \ldots, J$. Given a tensor $\mathcal{A}$ and two matrices $\mathbf{B}$ and $\mathbf{C}$ of matching sizes, we have: $\mathcal{A} \times_n \mathbf{B} \times_n \mathbf{C} = \mathcal{A} \times_n (\mathbf{BC})$. The n-unfolding of a tensor, when multiplied by several matrices along all its modes except one, is given by:

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \ldots \times_N \mathbf{B}^{(N)} \Leftrightarrow$$
$$\mathbf{A}_{(n)} = \mathbf{B}^{(n)} \mathbf{S}_{(n)} \left(\mathbf{B}^{(N)} \otimes \cdots \otimes \mathbf{B}^{(n+1)} \otimes \mathbf{B}^{(n-1)} \otimes \cdots \otimes \mathbf{B}^{(1)}\right)^T$$

Randomized algorithms are used to compute the low-rank approximation of large-scale data more efficiently in terms of memory and time compared to

deterministic methods. The low-rank approximation of the unfolding matrices $\mathbf{A} = \mathbf{A}_{(n)}, n = 1, 2, \ldots, N$ can be computed as follows:

$$\mathbf{A} \approx \left(\mathbf{Q}^{(1)}\mathbf{Q}^{(1)^T}\right)\mathbf{A}\left(\mathbf{Q}^{(2)}\mathbf{Q}^{(2)T}\right) = \mathbf{A} \times_1 \mathbf{Q}^{(1)}\mathbf{Q}^{(1)T} \times_2 \mathbf{Q}^{(2)}\mathbf{Q}^{(2)T},$$

where $\mathbf{Q}^{(1)} \in \mathbb{R}^{I_n \times \widehat{R}_n}$ and $\mathbf{Q}^{(2)} \in \mathbb{R}^{(\prod_{i \neq n} I_i) \times \widetilde{R}_n}$ $\left(\widehat{R}_n, \widetilde{R}_n < \text{rank}(\mathbf{A})\right)$ are approximation of the orthonormal bases for the column and row spaces of the matrix $\mathbf{A}$, respectively. The dimension of the second mode of the unfolding matrix $\mathbf{A} = \mathbf{A}_{(n)}$ is typically larger than that of the first mode, i.e., $I_n \ll \prod_{i \neq n} I_i$. Consequently, reduction along the second mode can be achieved by considering $\mathbf{Q} = \mathbf{Q}^{(1)}$. Inspired by the work of Che et al. [29], which introduced efficient randomized algorithms for low multilinear rank approximations of tensors, specifically HOSVD, we propose an efficient hardware implementation of randomized tensor decomposition using HLS.

3.2 Randomized tensor decomposition

This section presents the core of our proposed hardware design, a randomized tensor decomposition algorithm. The Tucker decomposition of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is defined as:

$$\mathcal{A} \approx \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_N \mathbf{U}^{(N)}, \tag{3}$$

where $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ are called the mode-n factor matrices, $n = 1, 2, \cdots, N$, and $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \cdots \times R_N}$ is the core tensor of the decomposition with multilinear rank $\left\{R_1, R_2, \cdots, R_N\right\}$ [6]. For simplicity, we illustrate our design and experimental results using third-order tensors, as generalizations to higher-order cases are straightforward. The mode-n unfolding matrix $\mathbf{A}_{(n)}$ of a tensor $\mathcal{A}$, for $n = 1, 2, 3$, can be reconstructed using the factor matrices and the mode-n unfolding of the core tensor $\mathcal{G}$ as follows:

$$\begin{aligned}
\mathbf{A}_{(1)} &\approx \mathbf{U}^{(1)}\mathbf{G}_{(1)}\left(\mathbf{U}^{(2)} \otimes \mathbf{U}^{(3)}\right)^{\top} \\
\mathbf{A}_{(2)} &\approx \mathbf{U}^{(2)}\mathbf{G}_{(2)}\left(\mathbf{U}^{(1)} \otimes \mathbf{U}^{(3)}\right)^{\top} \\
\mathbf{A}_{(3)} &\approx \mathbf{U}^{(3)}\mathbf{G}_{(3)}\left(\mathbf{U}^{(1)} \otimes \mathbf{U}^{(2)}\right)^{\top}
\end{aligned} \tag{4}$$

As the mode-n factor matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ at most has rank R_n, then $\text{rank}(\mathbf{A}_{(n)}) \leq R_n$. Applying singular value decomposition (SVD) to the mode-n unfolding matrix $\mathbf{A}_{(n)}$ for $n = 1, 2, 3$ yields the higher-order singular value decomposition (HOSVD) of $\mathcal{A}$, a special form of the Tucker decomposition. If the target rank $R_n < \text{rank}(\mathbf{A}_{(n)})$ for one or more modes, the decomposition is called truncated HOSVD, used to

initialize iterative algorithms for the best multilinear rank approximation. The low-rank decomposition of $\mathcal{A}$ can be formulated as an optimization problem:

$$\begin{aligned} &\min_{\mathcal{G},\mathbf{Q}_1,\mathbf{Q}_2,\mathbf{Q}_3} \left\|\mathcal{A} - \mathcal{G} \times_1 \mathbf{Q}_1 \times_2 \mathbf{Q}_2 \times_3 \mathbf{Q}_3\right\|_F^2, \\ &\text{subject to } \mathcal{G} \in \mathbb{R}^{\mu_1 \times \mu_2 \times \mu_3}, \end{aligned} \tag{5}$$

where $\mathbf{Q}_n \in \mathbb{R}^{I_n \times \mu_n}$ are orthonormal matrices and μ_1, μ_2, and μ_3 are positive integers. The low-rank tensor approximation based on randomized tensor decomposition is computed as follows:

- Projection step: For each mode n of tensor $\mathcal{A}$, the projection of mode-n unfolding of $\mathcal{A}$ onto the Kronecker product of random matrices is computed. These random matrices have entries that are independent and identically distributed Gaussian random variables with zero mean and unit variance. This projection captures most of the range of the mode-n unfolding of the tensor.
- Basis computation: Compute the basis for the resulting matrix from the projection step using QR decomposition.
- Final projection: Project the input tensor $\mathcal{A}$ onto the computed basis.

Let the mode-1 unfolding of $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ be $\mathbf{A}_{(1)} \in \mathbb{R}^{I_1 \times I_2 I_3}$. The SVD of $\mathbf{A}_{(1)}$ is $\mathbf{A}_{(1)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{I_1 \times I_1}$ and $\mathbf{V} \in \mathbb{R}^{I_2 I_3 \times I_2 I_3}$ are orthogonal, and $\mathbf{\Sigma} \in \mathbb{R}^{I_1 \times I_2 I_3}$ contains the singular values. If $\mathcal{B} = \mathcal{A} \times_1 \mathbf{Q}_1 \times_2 \mathbf{Q}_2 \times_3 \mathbf{Q}_3$, where $\mathbf{Q}_n \in \mathbb{R}^{I_n \times I_n}$ are orthogonal for $n = 1, 2, 3$, then the mode-1 unfolding of $\mathcal{B}$ is:

$$\mathbf{B}_{(1)} = \big(\mathbf{Q}_1\mathbf{U}\big)\mathbf{\Sigma}\big(\mathbf{V}\big(\mathbf{Q}_3 \otimes \mathbf{Q}_2\big)\big)^\top. \tag{6}$$

Therefore, the singular values of the mode-n unfolding of $\mathcal{B}$ are the same as that of the mode-n unfolding of $\mathcal{A}$ with $n = 1, 2, 3$. Algorithm 1 outlines the steps of the randomized tensor decomposition method, which can be summarized as follows: (1) The three product tensors $\mathcal{B}_n$ for $n = 1, 2, 3$ are generated using the power scheme and projection onto matrices with random Gaussian entries (Steps 1 and 2 of Algorithm 1). These random matrices $\mathbf{G}_{n,m}$ have dimensions $L_{n,m} \times I_m$, where $L_{1,2}L_{1,3} \geq R_1$, $L_{2,1}L_{2,3} \geq R_2$, and $L_{2,1}L_{3,2} \geq R_2$. (2) The mode-n unfolding of $\mathcal{B}_n$, which captures the maximum range of the data tensor, is computed (Step 3). (3) The mode-n orthonormal matrix $\mathbf{Q}_n$ is obtained using the QR decomposition of the mode-n unfolding of $\mathcal{B}_n$. In this context, we focus on the first mode of tensor $\mathcal{B}_n$, which captures the maximum range of the data tensor and highlights the main features for the background subtraction task.

Algorithm 1 HOSVD with random projection with $N = 3$

Input : $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and target multilinear rank R_n for $n = 1, 2, 3$.
Output: Three orthonormal matrices $\mathbf{Q}_n$, $n = 1, 2, 3$, such that:
$\|\mathcal{A} \times_1 (\mathbf{Q}_1\mathbf{Q}_1^T) \times_2 (\mathbf{Q}_2\mathbf{Q}_2^T) \times_3 (\mathbf{Q}_3\mathbf{Q}_3^T)\| \leq \sum_{n=1}^{3} O(\epsilon(\mathbf{A}_n))$

1 Generate six real matrices $\mathbf{G}_{n,m} \in \mathbb{R}^{L_{n,m} \times I_m}$ whose elements are independent and identically distributed (i.i.d.) Gaussian random variables of zero mean and unit variance, where $m, n = 1, 2, 3$ and $m \neq n$;
2 Compute the product tensors $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$, as follows,

$$\mathcal{B}_1 = \mathcal{A} \times_2 \mathbf{G}_{1,2} \times_3 \mathbf{G}_{1,3}$$
$$\mathcal{B}_2 = \mathcal{A} \times_1 \mathbf{G}_{2,1} \times_3 \mathbf{G}_{2,3}$$
$$\mathcal{B}_3 = \mathcal{A} \times_1 \mathbf{G}_{3,1} \times_2 \mathbf{G}_{3,2}$$

3 Get the mode-n unfolding $\mathbf{B}_{n,(n)}$ of each tensor $\mathcal{B}_n$;
4 Compute the QR decomposition of $\mathbf{B}_{n,(n)}$, as $\mathbf{B}_{n,(n)} = \mathbf{Q}_n\mathbf{R}_n$
5 **Return** $\mathbf{Q}_1 = \mathbf{Q}_1(:, 1 : R_1)$, $\mathbf{Q}_2 = \mathbf{Q}_2(:, 1 : R_2)$, and $\mathbf{Q}_2 = \mathbf{Q}_3(:, 1 : R_3)$.

Compared to conventional and other randomized HOSVD algorithms, Algorithm 1 avoids computing SVD of large unfolded matrices, reducing memory costs for random projection. This gives our accelerators a competitive edge in handling large datasets with low complexity and hardware resources. To compute HOSVD of a three-mode tensor $\mathcal{A}$ with dimension I for each mode, ST-HOSVD [9] requires $\mathcal{O}\left(I^4 + RI^3 + R^2I^2\right) + \left(RI^2 + R^2I + R^3\right)$ operations. With very large I, the complexity of ST-HOSVD increases rapidly due to the SVD of unfolded matrices. In contrast, Algorithm 1 reduces operations to $6\left(IL + LI^3 + L^2I^2\right) + \mathcal{O}\left(IL^2 + IL^4\right)$ by random projection, where $L = L_{n,1} = L_{n,2} = L_{n,3}$ and $L \ll I$. Additionally, Algorithm 1 avoids generating a large Gaussian random matrix by performing TTMc on two smaller random matrices, decreasing storage costs. For more details on the error bound and computational complexity analysis of the randomized tensor decomposition method, please refer to [29].

4 Hardware design for tensor decomposition using HLS

This section details the hardware implementation of the modified Tucker decomposition using Vivado HLS. The design process involves developing C++ files with specific constraints, headers, directives, and pragmas for optimization. After validating the algorithm, the HLS tool synthesizes the C++ code into RTL design, which can then be packaged as an IP for use in the Vivado design suite, system generator, or Xilinx platform studio.

4.1 The top design

Using HLS, we develop our design with Algorithm 1, forming top-level functions and submodules. The algorithm includes three key components: a multiplication

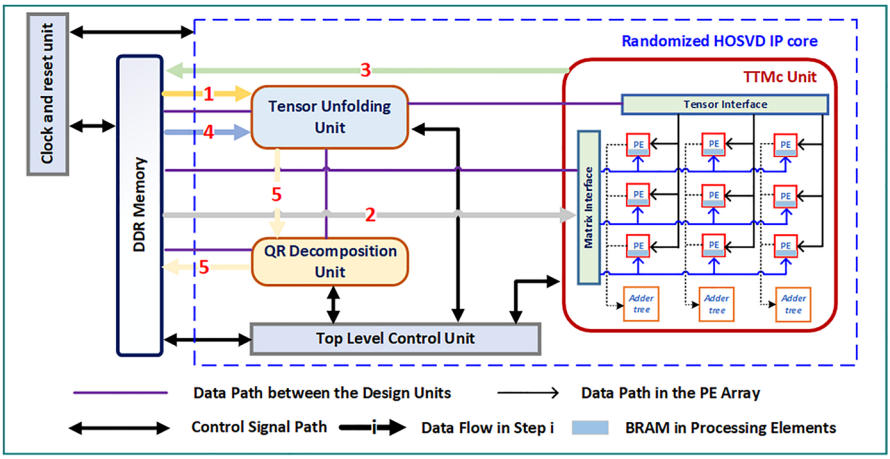

Fig. 1 Top-level design with dataflow for the proposed HOSVD design

unit, a matrix decomposition unit, and a tensor unfolding unit. The multiplication unit handles tensor-matrix operations to form product tensors $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$. The matrix decomposition unit performs QR decomposition for orthonormal factor matrices. The tensor unfolding unit converts tensors into their unfolding representation for TTMc operations and unfolds product tensors before QR decomposition. These units form the highest-level subfunctions. Figure 1 illustrates the hardware architecture of our HOSVD design, including the randomized HOSVD core, external DDR memory, clock, and reset unit. Data movement is shown in steps 1 to 5. Due to limited on-chip BRAMs, the input tensor and projection matrices are stored in external DDR memory and loaded to on-chip memory for processing. The projection matrices use Gaussian entries. Once the input data is stored in external memory (Step 1), the tensor is sent to the tensor unfolding unit, converting it to mode-n unfolding. Step 2 involves the TTMc multiplication using matrix and tensor entries to compute the product tensors. Once the multiplication unit obtains the fractions of product tensors, they are sent back to the memory and concatenated in Step 3. Then, the product tensor is forwarded to the tensor unfolding unit to convert to mode-1 unfolding, as shown in Step 4. This reshaped matrix is sent for QR decomposition and stored in memory, as depicted in Step 5. Input tensors are limited to three dimensions. For higher dimensions, tensors are first converted to a third-order tensor by unfolding along feature modes larger than three. Therefore, the overall iteration is limited to three. In each iteration, a column-wise orthogonal matrix is obtained. The top-level control selects the projection matrix entries and the other data entries for each unit.

4.2 TTMc unit

In this module, we have computed the tensor times matrix chain (TTMc). As demonstrated in [17], TTMc outperforms other tensor multiplication units. Consequently,

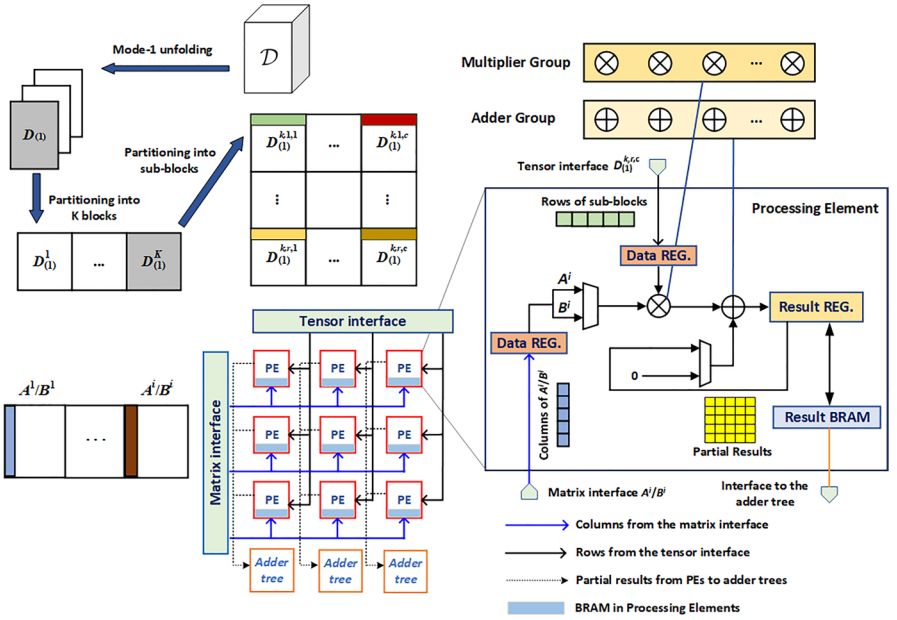

Fig. 2 Hardware architecture of the tensor times matrix chain (TTMc) unit

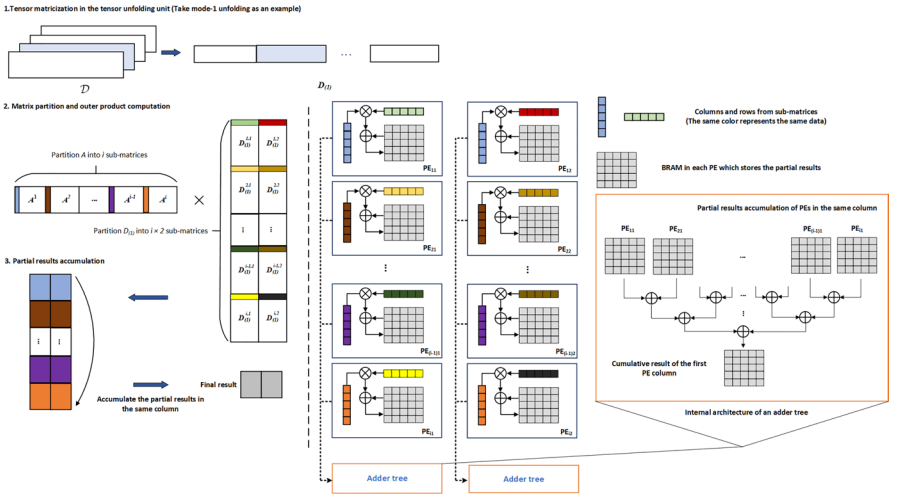

Fig. 3 Data partition schemes and dataflow of the proposed PE array

we extend this implementation using high-level synthesis (HLS) for higher-order singular value decomposition (HOSVD) design. Figure 2 illustrates the overall architecture of the TTMc unit and the internal architecture of the processing element. Figure 3 further details the matrix partition scheme and computation flow

for matrix multiplication based on the proposed PE array. The flow of the TTMc operations is highlighted in Algorithm 2. For instance, consider the TTMc operation $\mathcal{Y} = \mathcal{D} \times_1 \mathbf{A} \times_2 \mathbf{B}$. Due to the limited on-chip memory for storing large input tensors, our proposed design partitions a large input tensor $\mathcal{D}$ into K blocks. In each step, the TTMc unit processes one block to compute a partial result of the resultant tensor, $\mathbf{Y}^k_{(2)}$. All partial results are then concatenated to form the final resultant tensor, $\mathbf{Y}_{(2)}$.

To accelerate matrix multiplication, the input matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{D}^k_{(1)}$ are partitioned into smaller sub-matrices to enhance parallelism, as illustrated in Steps 3 and 4 of Algorithm 2. The proposed PE array efficiently handles parallel matrix multiplication by partitioning the input matrix according to the number of PEs in both vertical and horizontal directions. Figure 3 demonstrates how to map matrix multiplication onto an array with i rows and two columns. The size of the PE array can be increased using the same method. While matrix $\mathbf{A}$ is partitioned only in the column direction, matrix $\mathbf{D}$ is partitioned in both column and row directions. PEs share columns from sub-matrices A_i in the ith row. In the PE array, each PE operates independently, with its internal architecture shown in the right part of Fig. 2. Each PE has two input interfaces: one for columns from the matrix interface and one for rows from the tensor interface. The sub-matrices are fed into the corresponding PE, column by column (or row by row), to compute and accumulate outer products. After processing all columns and rows of the sub-matrices, each PE obtains a partial product, which is stored in the local BRAM. As illustrated in the right part of Fig. 3, an adder tree is then implemented to accumulate the partial products from all PEs within the same column. The accumulated products from each PE column are concatenated to form the final result. The proposed matrix partition scheme is tailored to the random projection of randomized HOSVD, where the Gaussian matrix $\mathbf{G}_{n,m} \in \mathbb{R}^{L_{n,m} \times I_m}$ is unbalanced, meaning $L_{n,m}$ is small. This type of matrix multiplication cannot be efficiently handled with conventional TTM architecture, which partitions two matrices in the same way. However, as shown in Fig. 3, the proposed partition scheme ensures that all PEs are utilized, thereby maximizing the efficiency of the entire PE array.

Algorithm 2 HLS pseudocode of TTMc unit

Input : $\mathbf{A} \in \mathbb{R}^{R_1 \times I}, \mathbf{B} \in \mathbb{R}^{R_2 \times J}$, and $\mathcal{D} \in \mathbb{R}^{I \times J \times K}$
Output: $\mathbf{Y}_{(2)} \in \mathbb{R}^{R_2 \times R_1 K}$

Get the mode-1 unfolding $\mathbf{D}_{(1)}$ of a tensor $\mathcal{D}$;
Partitioning $\mathbf{D}_{(1)}$ into K blocks, where each block (i.e., $\mathbf{D}^k_{(1)}$) has size $I \times J$, as $\left[\mathbf{D}^1_{(1)}, \ldots \mathbf{D}^k_{(1)}, \ldots, \mathbf{D}^K_{(1)}\right]$.
Partitioning $\mathbf{A}$ and $\mathbf{B}$ into i and j blocks as $\left[\mathbf{A}^1, \ldots, \mathbf{A}^i\right]$ and $\left[\mathbf{B}^1, \ldots, \mathbf{B}^j\right]$.
Partitioning $\mathbf{D}^k_{(1)}$ into sub-blocks as $\begin{bmatrix} \mathbf{D}^{k,1,1}_{(1)} & \cdots & \mathbf{D}^{k,i,1}_{(1)} \\ \vdots & \ddots & \vdots \\ \mathbf{D}^{k,1,j}_{(1)} & \cdots & \mathbf{D}^{k,i,j}_{(1)} \end{bmatrix}$.
for $(k = 1, k \leq K, k++)$ **do**
  for $(r = 1, r \leq i, r++)$ **do**
    for $(c = 1, c \leq j, c++)$ **do**
      $\mathbf{Y}^{k,r} = \mathbf{A}^i \mathbf{D}^{k,r,c}_{(1)}$
    end
    $\mathbf{Y}^k_{(2)} = \mathbf{B}^r (\mathbf{Y}^{k,r})^T$
  end
end
Concatenate the result $\mathbf{Y}_{(2)} = \left[\mathbf{Y}^1_{(2)}, \ldots . \mathbf{Y}^K_{(2)}\right]$

4.3 QR decomposition unit

In this work, we employ QR decomposition based on the modified Gram–Schmidt (MGS) method to decompose the product matrix obtained after tensor-matrix multiplication [30–33]. A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ can be decomposed using QR factorization into $\mathbf{Q} \in \mathbb{R}^{m \times n}$ and $\mathbf{R} \in \mathbb{R}^{n \times n}$, where $\mathbf{R}$ is an upper triangular matrix, such that $\mathbf{A} = \mathbf{QR}$. Once the factors are obtained, the dominant subspace is derived from the orthonormal basis $\mathbf{Q}$. This is achieved by taking $\mathbf{Q}_n = \mathbf{Q}_n(:, 1 : k)$, where $k = R_n$ represents the target rank required for the computation for $n = 1, 2, 3$. It is important to note that in the proposed design, only the orthonormal basis $\mathbf{Q}$ is required for the proposed randomized HOSVD, allowing us to omit the computation of $\mathbf{R}$ to simplify the algorithm and reduce hardware utilization.

The internal architecture of the QR decomposition unit is illustrated in Fig. 4 and summarized in Algorithm 3. The matrix $\mathbf{A}$ is initially stored in the BRAM and accessed column by column. During each iteration of the outer loop in Algorithm 3, a column is selected, and its dot product with subsequent columns is computed using a set of multipliers and adders. Subsequently, the parameters ir_{ii} and s_{ij} are computed. This allows for the extraction of one column of the orthonormal basis $\mathbf{Q}$, specifically q_i. The remaining columns of $\mathbf{A}$ are then updated through scalar

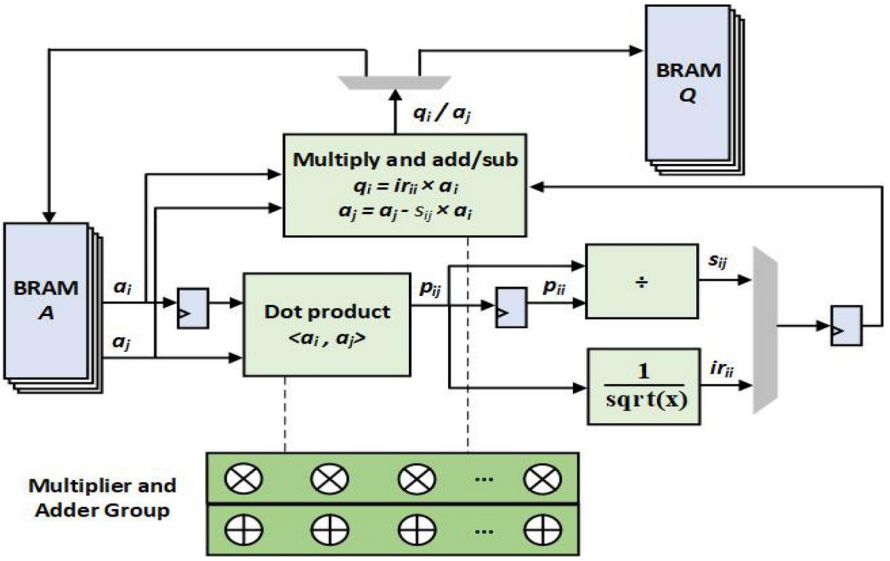

Fig. 4 Architecture of the proposed QR decomposition unit

multiplication and vector subtraction (or addition) operations. Finally, the updated columns a_j are written back to the BRAM.

Algorithm 3 Modified Gram–Schmidt QR decomposition

Input : Input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$
Output: Orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{m \times n}$
for $i = 1$ *to* n **do**
  for $j = i$ *to* n **do**
    $p_{ij} = \langle \boldsymbol{a_i}, \boldsymbol{a_j} \rangle$
    if $j = i$ **then**
      $p_{ii} = p_{ij}$;
      $ir_{ii} = \frac{1}{\sqrt{p_{ii}}}$;
      $\boldsymbol{q_i} = ir_{ii} \times \boldsymbol{a_i}$;
    end
    else
      $s_{ij} = \frac{p_{ij}}{p_{ii}}$;
      $\boldsymbol{a_j} = \boldsymbol{a_j} - s_{ij} \times \boldsymbol{a_i}$;
    end
  end
end
Return $\mathbf{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n]$

The QR implementation based on the Householder transformation has been employed in previous works [34, 35]. However, it is difficult to parallelize and, therefore, not recommended for hardware design. As shown in Algorithm 3, the

Table 2 Hardware utilization for the orthogonality step using different techniques

Algorithm	BRAM	DSP	FF	LUT
2 sided Jacobi	100	39	11,198	11,110
1 sided Jacobi [15]	–	256	13,964	11,134
QR based on MGS	59	48	13,248	10,047

modified Gram–Schmidt (MGS) QR decomposition can be easily parallelized since the computation is handled by column. Additionally, the pipelining technique can be applied with minimal overhead to further improve throughput. Although the SVD can be used for the orthogonality step, the QR decomposition is more computationally efficient. We compared three widely used techniques for the orthogonality step: (1) MGS QR, (2) 2-sided Jacobi SVD, and (3) 1-sided Jacobi SVD to determine which algorithm is more suitable for the proposed randomized HOSVD design. Table 2 shows a comparison between conventional SVD methods (2-sided Jacobi and 1-sided Jacobi) and QR decomposition regarding hardware resource utilization. Figure 5 illustrates the timing comparison of these methods with varying matrix dimensions.

The modified Gram–Schmidt QR decomposition demonstrates a considerable advantage over the other methods in terms of both resources and timing. The QR algorithm uses significantly fewer resources than the Jacobi-based methods. The BRAM resource is almost halved through the QR decomposition, which is a substantial benefit given the BRAM-based design. The utilization of DSP is also kept low. Although our design consumes more FF resources than the 2-sided Jacobi method, the LUT resource can be reduced by approximately 9.6%. Additionally, the timing of the 2-sided SVD method increases rapidly with an increase in tensor dimensions. As the dimension increases, the timing of the QR decomposition grows relatively slowly, resulting in a much faster performance compared to the conventional Jacobi method. This demonstrates a significant advantage of our design in handling large datasets. Notably, QR decomposition on FPGA is at least 6.9 times faster than the two-sided Jacobi method on a hardware platform and at least three times faster than the software counterpart of QR decomposition. Additionally, the maximum operational frequency for QR is 288 MHz, compared to 207 MHz for one-sided Jacobi and 154 MHz for two-sided Jacobi, demonstrating the efficiency of the QR decomposition unit.

4.4 Tensor unfolding unit

This unit is responsible for tensor unfolding and reshaping matrices. The TTMc unit partitions mode-1 entries of the input tensor, which are then processed by this unit. After multiplication, the output obtained in mode-2 unfolding of $\mathcal{Y}$, denoted as $\mathbf{Y}_{(2)}$, is converted to mode-1 unfolding $\mathbf{Y}_{(1)}$ before QR decomposition. Generally, the tensor unfolding unit is designed to permute and reshape the incoming tensor $\mathcal{A}$ into the unfolded matrix $\mathbf{A}_{(k)}$ along a specific mode k. This unit is crucial in our accelerator as it enables efficient tensor-times-matrix operations and QR factorization. The

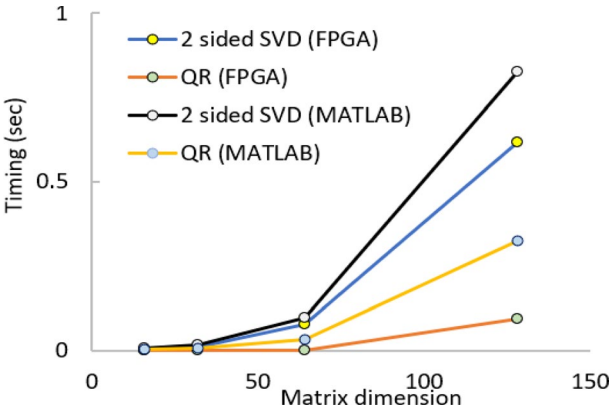

Fig. 5 Timing comparison of two-sided Jacobi vs QR decomposition

top part of Fig. 3 illustrates the operations in the tensor unfolding unit, using the mode-1 unfolding of a tensor $\mathcal{D}$ as an example. The unfolded matrix $\mathbf{D}_{(1)}$ is then sent to the PE array for TTMc operations. In our proposed accelerator, the large data tensor $\mathcal{A}$ is stored in external DDR memory, while the unfolded matrix $\mathbf{A}_{(k)}$ required for further processing is stored in on-chip memory. A careful data movement strategy is implemented: initially, tensor data is read from the external memory into a local buffer within the tensor unfolding unit. These data are then reshaped and written to the on-chip memory for subsequent processing steps.

4.5 Hardware optimization techniques

Specific FPGA optimization techniques are required to implement an architecture accelerator. HLS offers a set of directives and guidelines for these techniques [36, 37]. In the proposed randomized HOSVD design, the TTMc unit and QR decomposition unit are fully optimized to enhance the performance of the entire accelerator. In the QR decomposition unit, array partition is applied to BRAM **A** and BRAM **Q** so that computation can be handled by column. For example, each row of matrix **A** is stored in a separate BRAM, allowing parallel access to each column of **A**. Additionally, pipelining structures and loop unrolling are utilized for the dot product and column update to maximize throughput. The TTMc unit exhibits the highest hardware usage among all units. Loop unrolling is applied to the innermost loop for matrix multiplication within the PE, with an unrolling factor of 32 to conserve resources. To accommodate input tensors with larger dimensions, a larger PE array with more PEs in both columns and rows can be implemented, albeit with increased hardware costs. This trade-off will be further discussed in Sect. 6. Furthermore, partitioning and cyclic reshaping are applied to matrices **A**, **B**, and **D**, allowing columns and rows of sub-matrices to be accessed in a single cycle. The pipelining structures

are utilized only to compute outer products in each processing element (PE), as a fully pipelined design would exceed the resource limitations of FPGA devices. Additionally, array partitioning is applied to the BRAMs that store partial products, maximizing the parallel capacity of the PE array.

4.6 Analysis of hardware utilization and power consumption

In this section, we provide a theoretical analysis of the operations required by the proposed accelerator, considering the size of the input tensors and the target multilinear rank. We also explore how hardware utilization and power consumption vary with different design parameters. While floating-point arithmetic consumes more resources compared to fixed-point precision, our design specifically targets applications such as medical image compression, where high approximation accuracy is essential.

For the required operations, suppose the accelerator takes an input tensor of size $I_1 \times I_2 \times I_3$ with a multilinear rank set as $\{R_1, R_2, R_3\}$. For clarity, we set $I_1 = I_2 = I_3 = I$ and $R_1 = R_2 = R_3 = R$. The operations of our design are the sum of three parts: TTMc, tensor unfolding, and QR decomposition. For a three-mode input tensor, the proposed design computes three product tensors $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ involving three TTMc operations. Each TTMc operation requires $2\left(R^{\frac{1}{2}}I^3 + RI^2\right)$ operations. Tensor unfolding is performed on each product tensor, and unfolding one product tensor $\mathcal{B}_n$ requires $\mathcal{O}(IR)$ operations. Additionally, three QR decompositions are required, with each decomposition involving $\mathcal{O}(IR^2)$ operations. Thus, the total operations for decomposing a three-mode tensor are $6\left(R^{\frac{1}{2}}I^3 + RI^2\right) + \mathcal{O}(IR + IR^2)$.

The hardware utilization of our design is mainly determined by two units: the TTMc unit and the QR decomposition unit. For the TTMc unit, hardware resources depend on the size of the PE array, that is, the number of PEs in each row c and the number of PEs in each column r. The multipliers, adders, registers, and BRAMs utilized in the TTMc unit are proportional to $r \times c$, resulting in an area of approximately $\mathcal{O}(r \times c)$. In the QR decomposition unit, loop unrolling is applied to the dot product, scalar-vector multiplication, and vector subtraction operations. Hardware utilization here is determined by the unrolling factor (p), with an area of approximately $\mathcal{O}(p)$. While the hardware utilization increases with higher parallelism of the design, the power consumption also increases.

5 The proposed design for real applications

In this section, we utilize our proposed design in real-life applications. We explore background subtraction and MRI compression and investigate the performance of existing models.

5.1 Incremental *N*-mode SVD for Background Subtraction

Background subtraction is a key task in computer vision applications like intelligent transportation and human activity surveillance [25]. It isolates moving objects (foreground) from static scenes (background). Real-life scenarios often have dynamic backgrounds due to changes like illumination and water currents, complicating foreground extraction. After subtraction, the clean background aids high-level computer vision tasks. Batch processing in background subtraction typically requires the entire dataset to obtain the factors.

Several background subtraction techniques have been proposed. Traditional methods used principal component analysis (PCA) techniques were inefficient for dynamic backgrounds [38]. To better understand the inherent properties of the dataset and extend it to non-static backgrounds, it is essential to view the data as a tensor. Sobral et al. [39] demonstrated this approach with incremental tensor learning, but it suffers from significant computational complexity as each frame is processed individually. Another promising method employs a Gaussian model, as shown in [40]. In [41], features are represented as a graph, and semi-supervised learning is applied to extract the foreground. Inspired by [26], which uses randomized HOSVD to obtain factors followed by batch processing with incremental SVD, we propose an iterative tensor learning technique. This hybrid approach combines hardware and software implementations to accelerate dynamic background subtraction for large datasets while maintaining high accuracy. To enhance the method proposed in [26], we utilized our hardware design to perform the most complex step: obtaining the factor matrices of tensor data. Our implementation involves two main steps. First, tensor factors of the input frames are obtained on an FPGA using our modified Tucker decomposition hardware design. These factors are then sent to the CPU for background subtraction using *N*-mode incremental SVD. By utilizing our hardware design for the factor extraction step, we significantly accelerate the entire background subtraction process. This step is particularly time-consuming and can potentially become a throughput bottleneck.

The initial number of frames, K, is set to 15 to build an initial small tensor for consideration, as recommended in [26]. This choice is made for two reasons: (1) the initial subspace should capture the changes in those initial frames, so K should not be too small, and (2) to avoid high computational complexity if K is too large. This tensor is first decomposed using the proposed hardware design of the HOSVD. Once the factors are obtained, the subtraction model is implemented as shown in Algorithm 1 from [39]. A median filter is employed for binary mask post-processing [42]. By using random projection and power scheme in our design, we achieve better performance than [39], as shown in the quantitative performance analysis. Algorithm 4 outlines the update steps as follows:

$$[\mathbf{U}^*, \sim] = \mathrm{QR} \begin{bmatrix} f_p \mathbf{C}_{(p)} & \mathbf{U}_{(p)}^T \mathbf{B}_p \\ 0 & \mathrm{orth}(\mathbf{D}_{(k)}^T \mathbf{B}_{(p)}) \end{bmatrix}. \tag{7}$$

$$[\mathbf{Q}, \sim] = \mathrm{QR}\left[f_p \mathbf{D}_{(p)} \sqrt{\frac{f_p n_a n_b}{f_p n_a + n_b}}(\mathbf{A}_k - \mathbf{B})\right]. \tag{8}$$

$$\mathbf{U}^{(p)'} = \left[\mathbf{U}_{(p)} \ \mathbf{Q}\right] \mathbf{U}^{*}. \tag{9}$$

$$\mathbf{U}_{N'} = \begin{bmatrix} \mathbf{U}_{(N)} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} \end{bmatrix} \mathrm{orth} \begin{bmatrix} \mathbf{C}_{(N)}\mathbf{C}_{(N)}^{T} & \mathbf{B'}^{T} \\ \mathbf{B'} & \mathbf{B}_{(N)}\mathbf{B}_{(N)}^{'T} \end{bmatrix}, \tag{10}$$

Algorithm 4 N-mode incremental SVD

Input : Input frames $\mathcal{A}_K$, incoming frames $\mathcal{B}$, core $\mathcal{C}$ and initial factor matrices obtained from HOSVD $\mathbf{U}_k$

repeat
  if $p \neq N$ **then**
    Concatenate tensor as $\mathcal{D} = \left[\mathcal{A}_K \ \mathcal{B}\right]$;
    Compute projection matrix $\mathbf{U}^{(p)'}$ using Equations (7-10);
  end
  else
    Concatenate tensor as $\mathcal{D} = \begin{bmatrix} \mathcal{A}_K \\ \mathcal{B} \end{bmatrix}$;
    Compute projection matrix $\mathbf{U}_{N'}$ using Equation (11);
    Update the core tensor $\mathcal{C}' = \left[\mathcal{C} \times_{p \neq n} \mathbf{U}^{(p)'} \ \mathcal{B} \times_{p \neq n} \mathbf{U}^{(p)'}\right] \times_N \mathbf{U}_{N'}$;
  end
until *all modes covered*;

Here, f refers to the forgetting factor used for adaptive learning, $\mathbf{E}$ is the identity matrix, and n_a and n_b are the numbers of elements in $\mathcal{A}$ and $\mathbf{B}$, respectively, serving as a basis for normalization. Once the orthogonal factors are updated, they are used to update the core tensor based on the initial tensor $\mathcal{A}$. Since this multiplication would be a computationally expensive task due to the size of $\mathcal{A}$, it is modified in Eq. (11), where the core factor is updated directly to enhance computational efficiency:

$$\mathcal{C}' = \left[\mathcal{C} \times_{p \neq N} \hat{\mathbf{U}}_k^{'} \quad \mathcal{B} \times_{p \neq N} \mathbf{U}^{(p)'}\right] \times_N \mathbf{U}_N^{'}, \tag{11}$$

where $\mathbf{U}_k^{'}$ represents the upper block matrix of the third orthogonal factor $\mathbf{U}_N$.

5.2 MRI Compression

Medical image compression is a significant technique utilized to facilitate the storage of large image files in health organizations. Magnetic resonance imaging (MRI)

is employed to examine various body details by scanning the target organ and converting the signals into images [27]. In neuroimaging, brain scans are analyzed to investigate concussions, nerve connectivity, and brain fluid content. MRI datasets can be three-dimensional or four-dimensional, as multiple brain scans are stacked over time. Consequently, these datasets can range from gigabytes to terabytes, consuming significant memory space. Tensor decomposition can be applied to compress these medical images, effectively reducing their data size. Our proposed design has been utilized for this task, and experimental results on a real MRI dataset have demonstrated its efficiency in MRI compression.

6 Experiments

This section evaluates the performance of the proposed hardware design and compares it with a well-optimized counterpart and other implementations. The performance is evaluated from the perspective of the primary metrics of hardware design, latency, and utilized resources. To enhance latency, tensor elements are accessed in column-major order rather than the traditional row-major method. Additionally, various hardware optimization techniques, such as pipelining and loop unrolling, are employed to further reduce latency.

6.1 Experimental setup

The proposed design has been implemented on an FPGA, specifically the XCVU9P-FLGA2577-3-E device, using Xilinx Vivado HLS 2020.1. This device offers substantial hardware resources, including 4320 BRAMs (each consisting of 18K bits), 6840 DSP48E slices, 2,364,480 flip-flops (FFs), and 1,182,240 lookup tables (LUTs). Well-optimized CPU and GPU implementations using two popular tensor computing libraries, Tensor Toolbox [43] and Tensorly [44], have been used for comparison. Therefore, we compared our design with four baseline software implementations: (1) Tensor Toolbox library using MATLAB R2020b, (2) Tensorly with Numpy [45] backend, (3) Tensorly with PyTorch [46] backend on the CPU device, and (4) Tensorly with PyTorch backend on the GPU device. The CPU configuration used is an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz with 16 GB RAM, while the GPU configuration is a Nvidia Tesla T4 GPU with 16 GB memory. For latency testing of the CPU and GPU implementations, we performed the tensor decomposition ten times with the same input size and reported the average latency as the final result.

6.2 TTMc performance results

The tensor-matrix multiplication unit is one of the key components in our proposed tensor decomposition design, as detailed in Algorithm 2 and Fig. 2. In this section, we evaluate the performance of our TTMc unit and compare it with existing tensor multiplication units designed for hardware implementation. While few hardware

designs specifically target FPGA devices, our implementation demonstrates a significant advantage over previously proposed works.

The TTM unit presented in [15] addresses resource constraints by reducing permutation operations, switching buffers in the PE design, and minimizing the adder tree pipeline. However, this approach introduces latency overhead and data movement restrictions. The comparison of timing and resources with other implementations has not been presented in [15]. The TTM implementation in [34] performs mode-3 unfolding along the third dimension of the smaller tensor, obtained from the original tensor given as the input. This implementation is limited to sparse tensors and is specifically algorithm-oriented. Details regarding the TTM architecture for the power iteration used to obtain the smaller tensor are not provided, making the efficiency of this architecture heavily dependent on the input matrix. Consequently, the generality of this architecture is compromised. In contrast, our architecture can be extended to various applications, including those involving dense entries.

Table 3 presents a timing comparison of the TTMc unit between our proposed architecture and the related work in [34]. We evaluate the multiplication of a three-dimensional tensor with a matrix, varying the sizes of both. The initial size is fixed at 32 in all dimensions, increasing to 256 in the endmost dimension for both the tensor and matrix. Our TTMc unit can handle two consecutive TTM operations, which is an advantage for better scalability and flexibility in different tensor decomposition algorithms. Despite the additional control logic required for input matrix selection, the proposed TTMc achieves superior timing compared to the TTM in [34], as shown in Table 3. It efficiently handles varying-sized data, providing a speedup of 1.2 to 1.5 times.

6.3 Overall performance analysis

The section provides the performance evaluation of the developed hardware architecture. In Table 4, we compare the proposed design with various tensor decomposition implementations. We use input tensors of the size ($I \times I \times I$) and the target multilinear rank (R, R, R). Table 4 shows the timing performance of well-optimized CPU implementations based on widely-used libraries, including the Tensor Toolbox [43] and the Tensorly library [44], with Numpy and PyTorch backends.

According to Table 4, our design consistently outperforms CPU implementations across all input dimensions, achieving up to 14.56 times speedup compared to the Tensor Toolbox, the fastest CPU implementation. For small input tensors, our design shows a speedup comparable to GPU implementations. As the input size increases, the computational time for both CPU and GPU implementations rises rapidly, especially when the size grows from 256 to 512, making it more costly to process real-world datasets. In contrast, our design maintains much lower latency with large input tensors, achieving up to 5.55 times speedup compared to GPU implementations. This efficiency is primarily due to our random projection method, which significantly reduces the dimensions of unfolded product tensors. The column number of the unfolded product tensor is closely related to the target rank rather than the

input tensor size. Consequently, with a highly pipelined MGS hardware architecture, orthogonal factors can be computed efficiently column by column. Table 4 also compares our design with the state-of-the-art Tucker decomposition accelerator [15] in terms of timing. For an input dimension of 256, our design shows comparable latency to [15]. However, for larger input tensors of size 512, our design demonstrates approximately 1.6 times speedup, demonstrating its significant advantage in handling large tensors.

In Table 5, hardware utilization is compared for an input size of 256. Despite more FF resources used in our design, the resources of DSPs and LUTs can both be saved. Moreover, we compare the proposed design with an ASIC-based hardware implementation of tensor singular value decomposition proposed in [47]. We conduct our comparison by only varying the third dimension, allowing us to explore asymmetrical tensors. The work in [47] focuses on the speedup from software implementation using one-sided Jacobian SVD, but it lacks accuracy analysis and is limited to small tensor sizes of $64 \times 64 \times 32$. Consequently, it cannot be extended to large-scale, high-dimensional datasets or real-life applications that handle large-scale data. The comparison results are shown in Table 6. Our proposed design achieves up to 11.9 and 12.8 times faster results than the tensor toolbox and Tensorly libraries, respectively, and demonstrates better timing performance than the ASIC design.

While increasing the number of processing elements (PEs) can reduce computational time and accelerate the design, it also demands more hardware resources. Therefore, a trade-off between speed and hardware cost needs to be considered. The number of parallel operations cannot be further increased for the following reasons: (1) the resource-latency trade-off caused by the size increase and (2) the resource constraint caused by the speedup, which does not have a rapid increase.

Figure 6a shows the speedup of our proposed design over the well-optimized CPU implementation under different core ranks for $\mathcal{A} \in \mathbb{R}^{I \times I \times I}$, and $I = 32$. It can be observed that the speedup decreases almost linearly with an increase in the desired rank of the factor. It can also be seen that the difference in the maximum and minimum speed up concerning change in rank is not very large. When the core ranks for the tested tensor grow to 16, our design still achieves a promising speedup of no less than 16. This means that our design is suitable for low-rank applications and guarantees a low latency in more demanding tensor decomposition tasks where high accuracy is required. This advantage can be verified again when evaluating the

Table 3 Timing analysis of TTMc unit

Tensor dimension	Matrix dimension	TTM [34] (ms)	TTMc (our work) (ms)
$32 \times 32 \times 32$	32×32	0.148	0.096
$32 \times 32 \times 64$	32×64	0.281	0.205
$32 \times 32 \times 128$	32×128	0.546	0.425
$32 \times 32 \times 256$	32×256	1.077	0.832

Table 4 Timing comparisons of various implementations

I	R	Tensor Toolbox	Tensorly (Numpy) (s)	Tensorly (Pytorch) (s)	Tensorly (GPU) (s)	FPGA [15] (s)	Ours (s)
64	8	0.097	0.163	0.132	0.069	–	0.007
128	16	0.381	0.580	0.498	0.056	–	0.044
256	16	1.034	1.234	1.183	0.345	≈ 0.1	0.098
512	16	2.403	4.891	4.472	2.010	≈ 0.6	0.362

Table 5 Comparison of FPGA-based tensor decompositions in terms of hardware utilization

Design	BRAM	DSP	FF	LUT
Our work	2147	64	281,373	225,808
Tucker decomposition [15]	–	2048	231,036	339,378

speedup and accuracy of the functional-based, Hilbert tensor, and real-life image processing tasks in the following subsections.

Figure 6b illustrates the percentage of utilized hardware resources by our proposed design under changing the size of the data tensor from $32 \times 32 \times 32$ to $256 \times 256 \times 256$. As can be seen, by increasing the tensor size, the utilization of FF and LUT resources increases almost linearly but takes up a low percentage. The BRAM resource shows rapid growth, thus becoming the main constraint of our design. However, even under the large size of $256 \times 256 \times 256$, BRAM utilization does not exceed 60% of the available BRAM on the board. This demonstrates the efficiency of our proposed design in terms of hardware resource utilization.

6.4 Approximation accuracy

Various synthetic input formats have been considered to evaluate the performance of our proposed design regarding approximation accuracy. We use the relative l_2 norm error (RLNE) as an accuracy metric, defined as follows,

$$\text{RLNE} = \frac{\|\mathcal{A}' - \mathcal{A}\|_F}{\|\mathcal{A}\|_F}, \tag{12}$$

where $\mathcal{A}'$ is the approximated tensor.

We compare the accuracy of our proposed hardware design to observe its similarity with the well-optimized software implementation counterpart. The input data tensors considered include random Gaussian entries, function-based tensors, and Hilbert tensors. Random Gaussian entries are used to explore the applicability of the design with unknown input, and they have been used in the previous sections to show the performance of the design. Additionally, constraint-based inputs, such as function-based and Hilbert tensors, are used to evaluate the feasibility of extending

Table 6 Timing analysis of overall design with asymmetrical tensor

Size	Our work (ms)	Tensor toolbox (ms)	Tensorly (Pytorch) (ms)	Software [47] (ms)	Hardware [47] (ms)
$64 \times 64 \times 4$	5.644	26.07	41.20	41.46	6.052
$64 \times 64 \times 8$	5.912	30.07	44.60	45.43	6.118
$64 \times 64 \times 16$	6.026	41.10	47.70	56.75	6.282
$64 \times 64 \times 32$	6.420	76.75	82.60	86.77	6.675

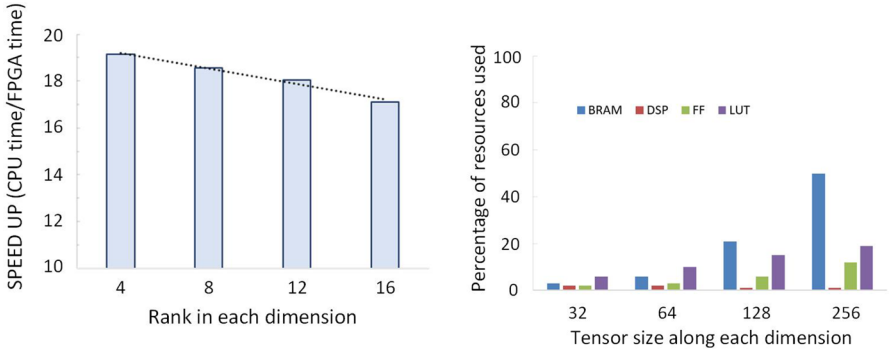

Fig. 6 **a** Speed-up analysis of our proposed design over the well-optimized CPU implementation under different core ranks. **b** Analysis of the percentage of resources used by our proposed design under changing the size of the data tensor

the design to real-life scenarios. Function-based tensors, which are common in signal-processing applications dealing with fluctuations and nonlinearity, are defined in Eq. 13. It is considered that i, j, and k are the three dimensions, and x is a positive integer. According to the value of x, the tensor multilinear rank is changed as follows: $x = 1 : R = \{17, 17, 17\}$, $x = 2 : R = \{26, 26, 26\}$, $x = 3 : R = \{34, 34, 34\}$, $x = 4 : R = \{41, 41, 41\}$, and $x = 5 : R = \{48, 48, 48\}$. To evaluate our design under the truncation (desire rank (32, 32, 32)), we set x to 5 to demonstrate the accuracy of our design for approximating data with low rank.

$$\mathcal{A}(i, j, k) = \frac{1}{\sqrt[x]{i^x + j^x + z^x}} \tag{13}$$

Hilbert tensors are derived from Hilbert matrices, which are a specific type of Hankel matrix. These tensors are widely used in signal distribution approximation. The entries in Hilbert tensors are defined as follows:

$$\mathcal{A}(i,j,k) = \frac{1}{i+j+k-N+1} \tag{14}$$

Table 7 presents the evaluation performance of our proposed design in terms of approximation error. The results indicate that the approximation error is minimal for both types of data input tensors, demonstrating that the accuracy and fit of the design are maintained. Consequently, the proposed design has been extended to real-life datasets, as detailed in Sect. 5.

6.5 Tensor-based background subtraction

We leverage our hardware design to perform the factor extraction step, significantly accelerating the background subtraction process. This step is typically very time-consuming and can become a throughput bottleneck. Our proposed method is more memory-efficient than conventional methods, as it stacks the first K frames from the entire dataset to extract the background. This approach eliminates the need to rely on the entire dataset, enhancing overall computational efficiency. The core and tensor factors are updated incrementally with incoming frames. We used the UCSD dataset [48] to evaluate the performance of our proposed method for the background subtraction task. This dataset consists of 18 categories of video frames that incorporate camera motion and moving backgrounds, and the videos have a large dynamic environment, with 30 up to 246 frames in each sequence. The sizes of tested video frames are slightly different, with the smallest size of 232×152. Therefore, the test video frames are resized to fit the input of our hardware implementation. We assessed our design by comparing the output images from our background subtraction method with the ground truth images using the following metrics:

$$\text{Precision} = \frac{Result \cap Groundtruth}{|Groundtruth|}, \tag{15}$$

$$\text{Recall} = \frac{Result \cap Groundtruth}{|Result|}, \tag{16}$$

$$\text{F}-\text{score} = \frac{2 \times Recall \times Precision}{Recall + Precision}, \tag{17}$$

The F-score measurement is used as a performance metric since it considers both recall and precision for the images. The quantitative analysis has been done according to established methods [39, 40] and [41].

Figure 7 illustrates the performance evaluation of our proposed background subtraction method across various video categories: Bottle-31, Chopper-46, Peds-24, Jump-49, and Birds-56. The Bottle and Birds videos feature water movements along

with changes in object positions. The Chopper video includes blade movements and changes in load positions, while the Peds and Jump videos capture people's movements. These complexities make background extraction particularly challenging. The first row displays the original video frames, the second row shows the ground truth images, and the third row presents the results obtained using our proposed method. The frames shown are keyframes from different videos. As demonstrated, our method produces outputs that closely match the ground truth.

Table 8 presents the quantitative performance of our proposed methods compared to previous works on the dynamic background subtraction task. Our approach replaces the standard HOSVD with a hardware-aided randomized HOSVD, achieving a comparable F-score. This indicates that the proposed randomized HOSVD accelerator offers a significant speedup and maintains high accuracy in dynamic background subtraction tasks. In [39], HOSVD is utilized for incremental multi-feature tensor subspace learning, achieving an F-score of 0.714 but suffering from high latency due to the time-consuming process of extracting multiple features from each frame. Additionally, our method demonstrates higher efficiency than [41], which requires convolutional neural network (CNN) training before model deployment. Among the previous works listed in Table 8, the method in [41] achieves the highest average F-score of 0.781, while our design attains an F-score of 0.792. Consequently, our method surpasses state-of-the-art dynamic background subtraction methods in both efficiency and accuracy.

6.6 Performance of MRI compression

To evaluate our proposed MRI image compression design, we used 30 human cranium scans, each with dimensions of 128×128. The data tensor was formed into a size of $128 \times 128 \times 30$. The target ranks for the approximated data were set to $R_1 = 64$, $R_2 = 64$, and $R_3 = 17$. Due to limited on-chip memory, the tensor data were initially stored in external DDR memory. It was then loaded into the on-chip BRAM memory for TTMc and QR decomposition, and finally, the results were written back to the external DDR memory. Our proposed design achieved approximately 7.9 times speedup over the well-optimized software implementation of the modified HOSVD algorithm. Specifically, the decomposition and reconstruction took around 0.0176 s with our design, compared to 0.1391 s with the tensor toolbox and 0.324 s with the software implementation of [15], which uses the HOOI algorithm. Furthermore, our design is 18.4 times faster than the HOOI software counterpart, which is also implemented using the tensor toolbox on MATLAB. Additionally, the HOOI

Table 7 The relative least normalized error of the reconstruction by the proposed design under different data tensors

Tenor data	Tensor size	Multilinear rank	Target rank	RLNE
Function-based	$256 \times 256 \times 256$	$32 \times 32 \times 32$	$48 \times 48 \times 48$	0.0826
Hilbert	$256 \times 256 \times 256$	$32 \times 32 \times 32$	$18 \times 18 \times 18$	0.0321

requires 7 SVD instances before convergence for a three-dimensional tensor input. In the overall design, SVD turns out to be very computationally expensive, and our algorithm mitigates this issue by performing QR decomposition only three times for a three-dimensional tensor.

The reconstruction error of the decomposed data is 0.1338 for a compression ratio of 6.6. Figure 8 illustrates the qualitative results of the approximated MRI data using our proposed design. The first row shows the ground truth of six frontal cranium scans, while the second row displays the corresponding approximated scans. As can be seen, our proposed design achieves a high-accuracy approximation, closely matching the ground truth images.

Moreover, we compare our implementation with another hardware tensor decomposition, Tensor train decomposition [18], and the performance comparison is shown in Table 9. We keep the compression ratio around 7.1 for a fair comparison between the two implementations. The speed up is compared concerning CPU implementation using tensor toolbox on Intel i7 core for both algorithms. It is demonstrated that our design gives a better speed up and less error than the TTD engine. For the TTD, the SVD core is equipped with multiple processing elements and can orthogonalize multiple pairs of columns in one cycle. Despite the increased resource utilization for the SVD core, this design shows a lower speedup ratio than our work. Also, the speedup ratio in [18] is achieved by compressing each image separately. Our design performs better by compressing the dataset as a large single tensor.

7 Conclusion and future work

In this paper, a novel approach integrating tensor decomposition on a reconfigurable platform has been proposed. Tucker decomposition has been modified to incorporate randomization and power scheme steps, improving the design's overall efficiency. The proposed design combines random projection, power scheme, and subspace approximation using QR decomposition to decompose and compute the low-rank

Fig. 7 Performance analysis of hybrid background subtraction for Bottle-31, Chopper-46, Peds-24, Jump-49, and Birds-56. The first row is the original frame, the second row is the ground truth images for the corresponding frames, and the third row contains the results obtained using our work

Table 8 Quantitative analysis of the proposed tensor-based background subtraction

Video	Hybrid INN-SVD	INN-SVD-BS [26]	GMM-BS [40]	iHoSVD-BS [39]	Graph-BGS [41]
Chopper	0.7910	0.7950	0.7450	0.7830	0.6956
Peds	0.7617	0.6770	0.6310	0.6390	0.8465
Bottle	0.8059	0.8020	0.7650	0.7890	0.8741
Jump	0.7765	0.7760	0.7650	0.7250	0.7727
Birds	0.8230	0.8150	0.7360	0.6340	0.7143
Average	0.7920	0.7730	0.7280	0.7140	0.7810

approximation of tensor data. The design architecture includes three main units: (1) tensor times matrix chain (TTMc), (2) tensor unfolding unit, and (3) QR decomposition unit to implement a three-stage algorithm. It is observed that the FPGA performance provides up to 14.56 times speedup compared with the fastest CPU implementation and up to 5.55 times speedup compared with the GPU implementation while maintaining accuracy in determining the factors. The design has been tested with the medical dataset for compression and also integrated with incremental learning-based background modeling. Since the INN-SVD method directly involves the large dataset and is the crucial component to obtain the factors used further in the increment of the tensor model, hardware acceleration has been employed. Furthermore, using incremental mode-based background subtraction, a better F-score, which serves as the accuracy parameter, is observed compared to other previously established models. Since it uses a fixed tensor size in the initial stage, the memory usage is very efficient and suitable for extension to real-time lengthy video processing systems. This paper thereby presents an efficient hardware-supported background modeling methodology. The proposed design has proven to give better results than the other hardware systems for MRI compression in terms of accuracy and speed up. Randomization schemes can be studied for tensor projections, and a

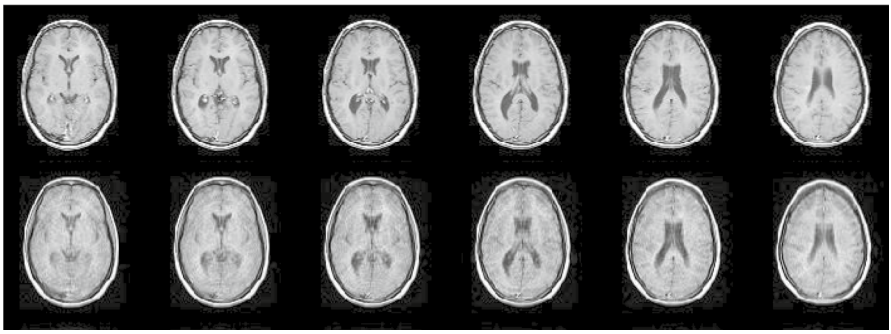

Fig. 8 Brain MRI dataset decomposition analysis. The original dataset is $128 \times 128 \times 30$. The rank for decomposition is kept as $64 \times 64 \times 17$. The first row contains the original images, and the second contains the reconstructed images

Table 9 Comparative analysis for MRI decomposition performance

Hardware	Compression ratio	Speed up	Error
Our work	7.1	25.4	0.1427
TTD engine [18]	7.1	20.4	0.157

fully integrated hardware-software co-simulation can be developed for future work. Furthermore, our current design uses a separate multiplier and adder in the TMMc unit. We will explore the integration of fused multiply-add (FMA) units to optimize the performance of the PE architecture. In this work, we have focused on decomposing three-dimensional tensors. We currently convert tensors with dimensions greater than three into three-mode tensors by unfolding along the feature mode. Future work will aim to enhance the architecture to support higher-order tensor decomposition directly.

Acknowledgements This work is supported by Hong Kong Innovation and Technology Commission (InnoHK Project CIMDA), Hong Kong Research Grants Council (Project 11204821), and City University of Hong Kong (Projects 9610034 and 9610460).

Funding Open access publishing enabled by City University of Hong Kong Library's agreement with Springer Nature.

References

1. Wu T (2022) Online tensor low-rank representation for streaming data clustering. IEEE Trans Circuits Syst Video Technol 33(2):602–617
2. De Lathauwer L, Vandewalle J (2004) Dimensionality reduction in higher-order signal processing and rank-(r1, r2, ..., rn) reduction in multilinear algebra. Linear Algebra Appl 391:31–55
3. Kolda TG, Bader BW (2009) Tensor decompositions and applications. SIAM Rev 51(3):455–500
4. Bi X, Tang X, Yuan Y, Zhang Y, Qu A (2021) Tensors in statistics. Annu Rev Stat Appl 8:345–368
5. Acosta-Quiñonez R, Torres-Roman D, Rodriguez-Avila R (2021) HOSVD prototype based on modular SW libraries running on a high-performance CPU + GPU platform. J Syst Archit 113:101897
6. Tucker LR (1966) Some mathematical notes on three-mode factor analysis. Psychometrika 31(3):279–311
7. De Lathauwer L, De Moor B, Vandewalle J (2000) A multilinear singular value decomposition. SIAM J Matrix Anal Appl 21(4):1253–1278
8. Sheehan BN, Saad Y (2007) Higher order orthogonal iteration of tensors (HOOI) and its relation to PCA and GLRAM. In: Proceedings of the 2007 SIAM International Conference on Data Mining. SIAM, pp 355–365

9. Vannieuwenhoven N, Vandebril R, Meerbergen K (2012) A new truncation strategy for the higher-order singular value decomposition. SIAM J Sci Comput 34(2):1027–1052
10. Kolda TG, Sun J (2008) Scalable tensor decompositions for multi-aspect data mining. In: 2008 Eighth IEEE International Conference on Data Mining. IEEE, pp 363–372
11. Diel P, Muñoz-Montoro AJ, Carabias-Orti JJ, Ranilla J (2024) Efficient FPGA implementation for sound source separation using direction-informed multichannel non-negative matrix factorization. J Supercomput 80:13411–13433
12. Nane R, Sima V-M, Pilato C, Choi J, Fort B, Canis A, Chen YT, Hsiao H, Brown S, Ferrandi F et al (2015) A survey and evaluation of FPGA high-level synthesis tools. IEEE Trans Comput Aided Des Integr Circuits Syst 35(10):1591–1604
13. Cong J, Liu B, Neuendorffer S, Noguera J, Vissers K, Zhang Z (2011) High-level synthesis for FPGAs: from prototyping to deployment. IEEE Trans Comput Aided Des Integr Circuits Syst 30(4):473–491
14. Srivastava N, Rong H, Barua P, Feng G, Cao H, Zhang Z, Albonesi D, Sarkar V, Chen W, Petersen P et al (2019) T2s-tensor: productively generating high-performance spatial hardware for dense tensor computations. In: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, pp 181–189
15. Zhang K, Zhang X, Zhang Z (2019) Tucker tensor decomposition on FPGA. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, pp 1–8
16. Huang W-P, Kwan BP, Ding W, Min B, Cheung RC, Qi L, Yan H (2019) High performance hardware architecture for singular spectrum analysis of Hankel tensors. Microprocess Microsyst 64:120–127
17. Huang W-P, Cheung RC, Yan H (2021) An efficient parallel processor for dense tensor computation. IEEE Trans Very Large Scale Integr (VLSI) Syst 29(7):1335–1347
18. Qu Z, Deng L, Wang B, Chen H, Lin J, Liang L, Li G, Zhang Z, Xie Y (2021) Hardware-enabled efficient data processing with tensor-train decomposition. IEEE Trans Comput Aided Des Integr Circuits Syst 41(2):372–385
19. Wijeratne S, Kannan R, Prasanna V (2021) Reconfigurable low-latency memory system for sparse matricized tensor times Khatri–Rao product on FPGA. In: 2021 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, pp 1–7
20. Srivastava N, Jin H, Smith S, Rong H, Albonesi D, Zhang Z (2020) Tensaurus: a versatile accelerator for mixed sparse-dense tensor computations. In: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, pp 689–702
21. Min B, Huang W-P, Cheung RC, Yan H (2019) A high performance hardware architecture for non-negative tensor factorization. Microelectron J 85:25–33
22. Ahmedsaid A, Amira A, Bouridane A (2003) Improved SVD systolic array and implementation on FPGA. In: Proceedings 2003 IEEE International Conference on Field-Programmable Technology (FPT)(IEEE Cat. No. 03EX798). IEEE, pp 35–42
23. Dongarra J, Gates M, Haidar A, Kurzak J, Luszczek P, Tomov S, Yamazaki I (2018) The singular value decomposition: anatomy of optimizing an algorithm for extreme scale. SIAM Rev 60(4):808–865
24. Wang Y, Lee J-J, Ding Y, Li P (2020) A scalable FPGA engine for parallel acceleration of singular value decomposition. In: 2020 21st International Symposium on Quality Electronic Design (ISQED). IEEE, pp 370–376
25. Garcia-Garcia B, Bouwmans T, Silva AJR (2020) Background subtraction in real applications: challenges, current models and future directions. Comput Sci Rev 35:100204
26. Khan S, Xu G, Yan H (2017) Tensor learning using n-mode SVD for dynamic background modelling and subtraction. In: 2017 Second Russia and Pacific Conference on Computer Technology and Applications (RPC). IEEE, pp 6–10
27. Mokhtari F, Laurienti PJ, Rejeski WJ, Ballard G (2019) Dynamic functional magnetic resonance imaging connectivity tensor decomposition: a new approach to analyze and interpret dynamic brain connectivity. Brain Connect 9(1):95–112
28. Cichocki A, Lee N, Oseledets I, Phan A-H, Zhao Q, Mandic DP et al (2016) Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. Found Trends Mach Learn 9(4–5):249–429
29. Che M, Wei Y, Yan H (2021) Randomized algorithms for the low multilinear rank approximations of tensors. J Comput Appl Math 390:113380

30. Souza Junior C, Bispo J, Cardoso JM, Diniz PC, Marques E (2020) Exploration of FPGA-based hardware designs for QR decomposition for solving stiff ode numerical methods using the harp hybrid architecture. Electronics 9(5):843
31. Tseng T-T, Shen C-A (2018) Design and implementation of a high-throughput configurable preprocessor for mimo detections. Microelectron J 72:14–23
32. Leon SJ, Björck Å, Gander W (2013) Gram–Schmidt orthogonalization: 100 years and more. Numer Linear Algebra Appl 20(3):492–532
33. Golub GH, Van Loan CF (2013) Matrix Computations. JHU Press, Baltimore
34. Jiang W, Zhang K, Lin CY, Xing F, Zhang Z (2020) Sparse tucker tensor decomposition on a hybrid FPGA–CPU platform. IEEE Trans Comput Aided Des Integr Circuits Syst 40(9):1864–1873
35. Tomás AE, Quintana-Ortí ES (2020) Tall-and-skinny QR factorization with approximate householder reflectors on graphics processors. J Supercomput 76(11):8771–8786
36. Xilinx, U.: Vivado HLS Optimization Methodology Guide. Apr (2018)
37. Fine Licht J, Besta M, Meierhans S, Hoefler T (2020) Transformations of high-level synthesis codes for high-performance computing. IEEE Trans Parallel Distrib Syst 32(5):1014–1029
38. Javed S, Oh SH, Heo J, Jung SK (2014) Robust background subtraction via online robust PCA using image decomposition. In: Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, pp 105–110
39. Sobral A, Baker CG, Bouwmans T, Zahzah E-h (2014) Incremental and multi-feature tensor subspace learning applied for background modeling and subtraction. In: Image Analysis and Recognition: 11th International Conference, ICIAR 2014, Vilamoura, Portugal, October 22–24, 2014, Proceedings, Part I 11. Springer, pp 94–103
40. Silva C, Bouwmans T, Frélicot C (2015) An extended center-symmetric local binary pattern for background modeling and subtraction in videos. In: International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISAPP 2015
41. Giraldo JH, Bouwmans T (2021) GraphBGS: background subtraction via recovery of graph signals. In: 2020 25th International Conference on Pattern Recognition (ICPR). IEEE, pp 6881–6888
42. George G, Oommen RM, Shelly S, Philipose SS, Varghese AM (2018) A survey on various median filtering techniques for removal of impulse noise from digital image. In: 2018 Conference on Emerging Devices and Smart Systems (ICEDSS). IEEE, pp 235–238
43. Bader BW, Kolda TG et al (2015) Matlab tensor toolbox version 2.6. http://www.sandia.gov/tgkolda/TensorToolbox
44. Kossaifi J, Panagakis Y, Anandkumar A, Pantic M (2016) Tensorly: tensor learning in Python. arXiv preprint arXiv:1610.09555
45. Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ et al (2020) Array programming with NumPy. Nature 585(7825):357–362
46. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol 32
47. Deng C, Yin M, Liu X-Y, Wang X, Yuan B (2019) High-performance hardware architecture for tensor singular value decomposition. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, pp 1– 6
48. Mahadevan V, Vasconcelos N (2009) Spatiotemporal saliency in dynamic scenes. IEEE Trans Pattern Anal Mach Intell 32(1):171–177

Authors and Affiliations

Ajita Misra[1] · Muhammad A. A. Abdelgawad[1] · Peng Jing[1] · Ray C. C. Cheung[1] · Hong Yan[1]

✉ Ajita Misra
ajita.msra@gmail.com

Muhammad A. A. Abdelgawad
mabdelgaw2-c@my.cityu.edu.hk

Peng Jing
pjing4-c@my.cityu.edu.hk

Ray C. C. Cheung
r.cheung@cityu.edu.hk

Hong Yan
h.yan@cityu.edu.hk

[1] Department of Electrical Engineering and Centre for Intelligent Multidimensional Data Analysis, City University of Hong Kong, Kowloon, Hong Kong