Algorithm and Hardware Co-Design of Energy-Efficient LSTM Networks for Video Recognition with Hierarchical Tucker Tensor Decomposition

Yu Gong, Miao Yin, Lingyi Huang, Chunhua Deng, Yang Sui, and Bo Yuan

Abstract—Long short-term memory (LSTM) is a type of powerful deep neural network that has been widely used in many sequence analysis and modeling applications. However, the large model size problem of LSTM networks make their practical deployment still very challenging, especially for the video recognition tasks that require high-dimensional input data. Aiming to overcome this limitation and fully unlock the potentials of LSTM models, in this paper we propose to perform algorithm and hardware co-design towards high-performance energy-efficient LSTM networks. At algorithm level, we propose to develop fully decomposed hierarchical Tucker (FDHT) structure-based LSTM, namely FDHT-LSTM, which enjoys ultra-low model complexity while still achieving high accuracy. In order to fully reap such attractive algorithmic benefit, we further develop the corresponding customized hardware architecture to support the efficient execution of the proposed FDHT-LSTM model. With the delicate design of memory access scheme, the complicated matrix transformation can be efficiently supported by the underlying hardware without any access conflict in an on-the-fly way. Our evaluation results show that both the proposed ultra-compact FDHT-LSTM models and the corresponding hardware accelerator achieve very high performance. Compared with the state-of-the-art compressed LSTM models, FDHT-LSTM enjoys both order-of-magnitude reduction (more than 1000) in model size and significant accuracy improvement (0.6% to 12.7%) across different video recognition datasets. Meanwhile, compared with the state-of-the-art tensor decomposed model-oriented hardware TIE, our proposed FDHT-LSTM architecture achieve 2:5, 1:46 and 2:41 increase in throughput, area efficiency and energy efficiency, respectively on LSTM-Youtube workload. For LSTM-UCF workload, our proposed design also outperforms TIE with 1:9 higher throughput, 1:83 higher energy efficiency and comparable area efficiency.

Index Terms—Long Short-term Memory (LSTM), Hardware Architecture, Tensor Decomposition, Hierarchical Tucker (HT), Video Recognition.

F

1 Introduction

Long short-term memory (LSTM) networks are popularly used in many real-world sequential analysis and processing tasks. Thanks to their inherent strong capability of capturing the temporal dependency and modeling the sequential correlation, the state-of-the-art LSTMs have achieved unprecedented success in many important temporal sequence-involved applications, such as video recognition [7], speech recognition [9] and natural language processing (NLP) [3].

Despite their current widespread adoptions, the large model sizes of LSTM networks make the practical deployment still very challenging. To be specific, because the input data of many real-world applications, e.g., video processing and NLP, naturally exhibit high dimensionality, the corresponding input-to-hidden weight matrices of LSTMs are typically extremely large. For instance, as analyzed in [20], even with small-size (256 states) hidden layer, the LSTM evaluated on UCF11 video recognition dataset [16] already requires more than 50 million weights. Evidently, such ultra-large model size poses a series of severe challenges for the efficient deployment of LSTMs, including but not limited to high memory footprint, long processing time, low energy efficiency, and insufficient accuracy incurred by high difficulty of training.

To address these challenges, various model compression methods, such as pruning and quantization, have been proposed to build compact LSTMs. Consider the compression ratio provided by these methods are typically limited and may be insufficient for the very large LSTMs, tensor decomposition, as a low-rank approximation approach, have attracted a lot of attention towards high-performance compression of LSTM models. In general, tensor decomposition aims to factorize a large-size tensor to a series of small tensor cores with maintaining small approximation error. After performing tensor decomposition, a very elegant mathematical phenomenon is that the resulting space and time complexity can be significantly reduced, sometimes even with order-of-magnitude reduction. Motivated by this attractive property, recently several different tensor decomposition approaches, such as tensor train (TT), tensor ring (TR) and block-term (BT), have been adopted to build compact LSTMs models [6] [17] [21] for various temporal data analysis tasks, e.g., video recognition and long-term dynamic forecasting. Compared with the original uncompressed large-size

Yu Gong, Miao Yin, Lingyi Huang, Yang Sui and Bo Yuan are with the Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ, 08854. E-mail: yg430@soe.rutgers.edu, miao.yin@rutgers.edu, lingyi.huang@rutgers.edu, yang.sui@rutgers.edu, bo.yuan@soe.rutgers.edu.

Chunhua Deng is currently with ScaleFlux Inc. This work was done when the author was with Rutgers University. E-mail: chunhua.deng518@gmail.com.

Yang Sui contributed to this work but was accidentally omitted in the published version.

LSTMs, these tensor decomposed models exhibit much smaller memory footprints and competitive classification/prediction accuracy.

Although the state-of-the-art tensor decomposed LSTMs demonstrate their promising potentials, they are still facing two inherent limitations. First, the existing approaches can only factorize the input-to-hidden layers instead of the entire model, thereby limiting the overall compression performance. Notice that, as will be shown in Section 3, the straightforward decomposition on the hidden-to-hidden layers cannot solve this problem due to the significant accuracy degradation. Second, the tensor decomposition approaches adopted in the existing LSTM compression works have inherent limitations. To be specific, TT decomposition requires that the border tensor cores be rank-1 tensors, thereby directly limiting the overall representation power. Also, using BT decomposition brings extra flatten and permutation operations, which causes significant computation overhead for the resulting BT-LSTM models. More importantly, as will be analyzed in Section 3, the existing adopted tensor decomposition approaches (TT, TR and BT) in model compression, theoretically, do not provide the best space complexity reduction. Consequently, they are not the ideal solutions for building tensor factorized high-accuracy ultra-compact LSTM models.

To overcome these limitations and develop high-performance LSTM models, in this paper [1] we propose to leverage Hierarchical Tucker (HT) [10] technique, an under-explored yet powerful tensor tool, to fully decompose the LSTM networks. The resulting compressed model, namely FDHT-LSTM, enjoys ultra-low complexity with still maintaining high accuracy. Then, to fully reap the benefits brought by the proposed efficient compression approach, we further develop the corresponding hardware architecture to accelerate the execution of FDHT-LSTM models. Overall, the contributions of this paper are summarized as follows:

- At the algorithm level, we propose to impose fully decomposed hierarchical Tucker (FDHT) structure on the LSTM networks to achieve ultra-high compression performance. The proposed FDHT structure contains two main features (as shown in Figure 1). First, the underlying LSTM is built via using HT decomposition, a powerful approach that can properly capture and model the correlation and structure in high-dimensional data. Second, the entire LSTM model, instead of one or few component layers, is fully decomposed in the HT format in a homogeneous way. In other words, the low-rank correlation among different component layers are fully exploited, and thereby bringing order-of-magnitude reduction in model size while still achieving very high accuracy.
- At the hardware level, we design and implement the corresponding FDHT-LSTM hardware architecture to fully leverage the algorithmic benefits. In order to support various complicated matrix transformations introduced by the computation flow of FDHT-LSTM, we first propose to utilize 2-D SRAM array to properly map the high-order tensor transpose on the physical 2-D memory component. Based on this philosophy, we further propose novel write and read access schemes to the 2-D SRAM array, thereby enabling conflict-free memory access with high flexibility and reconfigurability to different workloads.
- Evaluation results demonstrate the superior performance of our proposed FDHT-LSTM and the corresponding hardware accelerator. Experiments show that the FDHT-LSTM models can use very few parameters (less than 10,000 weights) to achieve very high accuracy across different video recognition datasets. Compared with the state-of-the-art compressed LSTM models, FDHT-LSTM enjoys both order-of-magnitude reduction (more than 1000) in model size and significant accuracy improvement (0.6% to 12.7%). Meanwhile, compared with the state-of-the-art tensor decomposed model-oriented hardware TIE, our proposed FDHT-LSTM architecture achieves 2:5, 1:46 and 2:41 increase in throughput, area efficiency and 2:41 energy efficiency, respectively on LSTM-Youtube workload. For LSTM-UCF workload, our proposed design also outperforms TIE with 1:9 higher throughput, 1:83 higher energy efficiency and comparable area efficiency.

1. This paper is the extension of the authors' prior work [22].

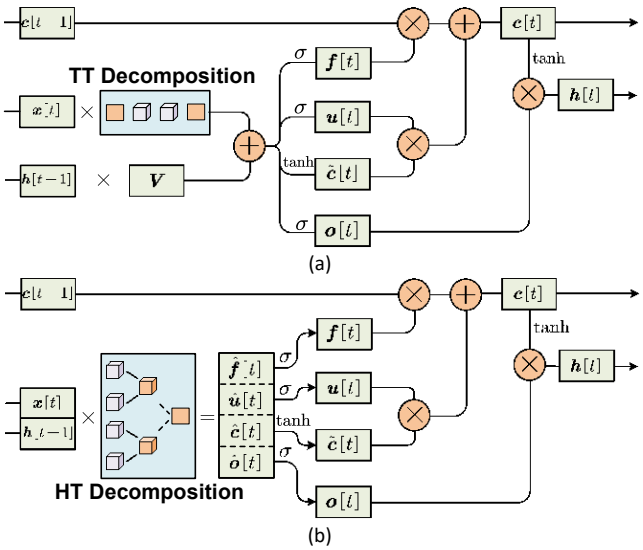


Fig. 1: Architecture of tensor decomposition-based LSTM. (a) Prior TT-LSTM. (b) The proposed FDHT-LSTM. Reproduced from [22].

The rest of this paper is organized as follows. Section 2 introduces the background of tensor decomposition and the motivation of hierarchical Tucker (HT)-based LSTM compression. The proposed fully decomposed HT-structured (FDHT) LSTM algorithm is described in Section 3. Section 4 presents the corresponding FDHT-LSTM hardware architecture. The evaluation performance at both algorithm and hardware levels is reported in Section 5. Section 7 draws the conclusions.

2 Background and Motivation

2.1 Preliminaries

2.1.1 Notation

In this paper, boldface lower-case, boldface capital and boldface calligraphic letters represent vectors, matrices and higher-order tensors, respectively, e.g. x, X and X. Additionally, letters with indices in bracket denote the entry, e.g., xpiq, Xpi; jq, Xpi$_1$; ; i$_d$q.

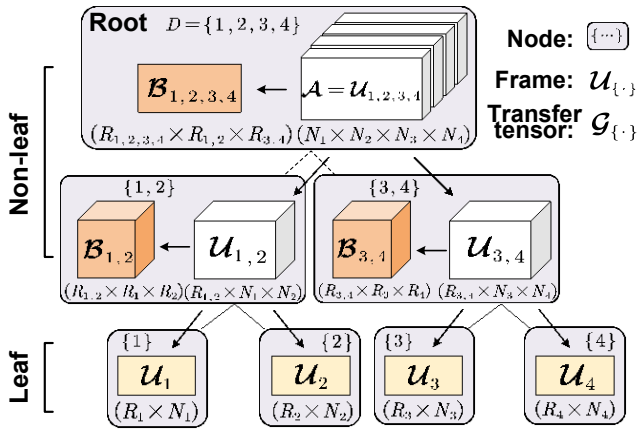


Fig. 2: Standard HT decomposition example for a 4-order tensor. The hierarchy is a binary tree with root t1; 2; 3; 4u, where nodes are denoted by rounded rectangles. t1u, t2u, t3u t4u are leaf nodes, and t1; 2u, t3; 4u are their parents. In HT format, only colored leaf frames and transfer tensors are needed to store. Reproduced from [22].

2.1.2 Tensor Contraction

Tensor contraction is the multiplication between two higher-order tensors where more than one dimension matches. For example, given two tensors $\mathcal{A} \in \mathbb{R}^{M_1 M_2 C}$ and $\mathcal{B} \in \mathbb{R}^{C N_1 N_2}$, where the 3rd dimension of $\mathcal{A}$ is identical to the 1st dimension of $\mathcal{B}$ with size C, the result of tensor contraction $\mathcal{C} = \mathcal{A} \times_1^3 \mathcal{B}$, as a size-$M_1 \times M_2 \times N_1 \times N_2$ tensor, can be calculated as:

$$\mathcal{C}(i_1, i_2, j_1, j_2) = \sum_{k=1}^{C} \mathcal{A}(i_1, i_2, k) \mathcal{B}(k, j_1, j_2). \quad (1)$$

2.1.3 Hierarchical Tucker Decomposition

As a special case of tensor decomposition, hierarchical Tucker (HT) approach recursively decomposes the original tensor into small tensor cores with hierarchical levels from top to bottom in a binary tree. As illustrated in Figure 2, the upper-level intermediate components can be factorized to lower-level components. Here the intermediate components are referred as frames, each of which corresponds to a unique node. Additionally, each node in the binary tree is represented by a dimension set. In general, given a d-order tensor $\mathcal{A} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$, we can build a binary tree to split the original dimension set $\{1, \ldots, d\}$ and assign the index to each node. To be specific, the root node $D = \{1, 2, \ldots, d\}$ is associated with the root frame $\mathcal{A} = \mathcal{U}_D$, $s \subset D$ is the node corresponding to frame $\mathcal{U}_s$, and $s_1, s_2 \subset s$ are the left and right child nodes of node s. Notice that if we define $\text{range}_s = [\min(s), \max(s)]$, then each non-leaf frame $\mathcal{U}_s \in \mathbb{R}^{R_s \times N_s \times N_s}$ can be recursively decomposed to frames ($\mathcal{U}_{s_1}$ and $\mathcal{U}_{s_2}$) and transfer tensor $\mathcal{B}_s \in \mathbb{R}^{R_s \times R_{s_1} \times R_{s_2}}$ as

$$\mathcal{U}_s = \mathcal{B}_s \times_1^2 \mathcal{U}_{s_1} \times_1^2 \mathcal{U}_{s_2}, \quad (2)$$

where R_s, R_{s_1}, R_{s_2} are referred as hierarchical ranks. Overall, by using hierarchical Tucker decomposition that recursively decomposes the frames from top to bottom, we can use the combination of the small-size 2-order leaf frames and 3-order transfer tensors to store the original large-size d-order tensor $\mathcal{X} = \mathcal{U}_D$.

2.2 Compressing LSTMs via Tensor Decomposition

In the recent years tensor decomposition has emerged as a very attractive technique to compress very large LSTM models. In [20], tensor train (TT) decomposition is applied to factorize the input-to-hidden layer of LSTMs and GRUs for video recognition. The reported experimental results show that very significant compression ratio can be achieved while maintaining high accuracy. Motivated by this success, other advanced tensor decomposition approach, such as block-term (BT) [21] decomposition and tensor ring (TR) decomposition [21], are also used to build compact LSTM models (BT-LSTM and TR-LSTM) in video processing tasks. In addition, tensor decomposed LSTMs also demonstrate high performance in other sequential analysis and prediction tasks. For instance, [23] proposes a TT-format LSTM to perform long-term forecasting in dynamic systems.

2.3 Motivation of Compressing LSTM via HT Decomposition

In this paper we propose to adopt HT decomposition, a relatively little noticed yet powerful tensor factorization approach for LSTM compression. Our choice is motivated by two reasons. First, compared with its well-explored counterparts (e.g. TT, TR and BT decomposition) in the model compression field, HT decomposition, by it nature, can provide higher space complexity reduction on the same-size tensor data with the same selected rank, thereby implying that even smaller LSTM models can be constructed in the HT format. Second, from the perspective of tensor theory, the inherent hierarchical structure of HT also enables better weight sharing and hierarchical representation from high-dimensional data, which are very important to guarantee the representation capability of tensor decomposed LSTM models. As we will report in Section 5, the HT structure-based LSTM models consistently demonstrate superior performance than the existing compressed LSTMs using other tensor decomposition methods with respect to both model accuracy and compression ratio.

3 The Proposed FDHT-LSTM: Algorithm

3.1 Compact HT-structure Linear Layer

Consider the linear layer is the foundation of LSTM models, in this subsection we first build a compact HT-structure layer that serves as the key component of the proposed FDHT-LSTM model.

3.1.1 Tensorization

In the compact HT-layer, computations are performed with high-order tensor contraction, thus we first need to transform all the original variables into tensor format. In general, for an original linear layer with weight matrix $\mathbf{W} \in \mathbb{R}^{O \times I}$ that linearly maps an input vector $\mathbf{x} \in \mathbb{R}^{I}$ to an output vector $\mathbf{y} \in \mathbb{R}^{O}$ in HT-based format, we first transform the weight matrix $\mathbf{W}$ to an d-order tensor $\mathcal{W} \in \mathbb{R}^{O_1 \times \cdots \times O_d \times I_1 \times \cdots \times I_d}$. In addition, by defining $I = \prod_{i=1}^{d} I_i$ and $O = \prod_{j=1}^{d} O_j$, we can also tensorize the affiliated input vector $\mathbf{x}$ and output vector $\mathbf{y}$ to input and output tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ and $\mathcal{Y} \in \mathbb{R}^{O_1 \times \cdots \times O_d}$, respectively.

3.1.2 Decomposing W

After tensorizing the large-size weight matrix $\mathbf{W}$ as weight tensor $\mathcal{W} \in \mathbb{R}^{O_1 \times \cdots \times O_d \times I_1 \times \cdots \times I_d}$, we can then utilize HT decompo-sition to denote the original $\mathbf{W}$ with a set of small-size matrices

and tensors. With the definition of HT decomposition in Eq. (2), $\mathcal{W}$ can be factorized as

$$\mathcal{W}((o_1,\cdots,o_d),(i_1,\cdots,i_d)) = \sum_{k=1}^{R_D}\sum_{p=1}^{R_{D_1}}\sum_{q=1}^{R_{D_2}} (\mathcal{B}_D)_{(k,p,q)} (\mathcal{U}_{D_1})_{(p,\phi_{D_1}(o,i))}(\mathcal{U}_{D_2})_{(q,\phi_{D_2}(o,i))}, \tag{3}$$

where $\phi_s(o,i)$ is a mapping function which returns the associate indices $o = (o_1,\cdots,o_d)$ and $i = (i_1,\cdots,i_d)$ for a specified frame $\mathcal{U}_s$ with a given node s and d. For example, with $d = 6$ and $s = \{3,4\}$, the output of $\phi_s(o,i)$ is (o_3,o_4,i_3,i_4). Additionally, $\mathcal{U}_{D_1}$ and $\mathcal{U}_{D_2}$ can be further recursively decomposed as

$$(\mathcal{U}_s)_{(k,\phi_s(o,i))} = \sum_{p=1}^{R_{s_1}}\sum_{q=1}^{R_{s_2}} (\mathcal{B}_s)_{(k,p,q)} (\mathcal{U}_{s_1})_{(p,\phi_{s_1}(o,i))}(\mathcal{U}_{s_2})_{(q,\phi_{s_1}(o,i))}, \tag{4}$$

where $D = \{1,2,\cdots,d\}$, $D_1 = \{1,\cdots,\lfloor d/2 \rfloor\}$ and $D_2 = \{\lceil d/2 \rceil,\cdots,d\}$ are the left and right child nodes of the root node D.

3.1.3 HT-Structure Layer

As the original weight matrix is decomposed to HT-format, the HT-structured linear layer can be readily constructed via HT-format matrix-vector multiplication instead of recovering back to original format. To be specific, the forward pass of an HT-layer is computed as:

$$\mathcal{Y}(o) = \sum_{i}\sum_{k=1}^{R_D}\sum_{p=1}^{R_{D_1}}\sum_{q=1}^{R_{D_2}} (\mathcal{B}_D)_{(k,p,q)} (\mathcal{U}_{D_1})_{(p,\phi_{D_1}(o,i))}(\mathcal{U}_{D_2})_{(q,\phi_{D_2}(o,i))}\mathcal{X}(i). \tag{5}$$

After the above computation, we need to reshape the output tensor $\mathcal{Y}$ back to the original vector-format y. Hence, we can simplify the entire computing process as a single HT-structure layer:

$$y = HTL(\mathcal{W}, x). \tag{6}$$

As depicted in Figure 3, the consecutive arrows denote the computation flow in the computation process of an HT-structure layer. Specifically, the input vector x is first tensorized, and then it is contracted with the HT-format weight tensor in a hierarchical way. Finally, the obtained output tensor is reshaped back to output vector y.

3.1.4 Benefits on Low Cost

With the proposed HT-layer, one most important benefit is huge complexity reduction. Table 1 shows the theoretical space complexity in comparison with other tensor factorized linear layer as well as the uncompressed one. Considering tensor rank R is generally smaller than I or O, HT-structure layer can provide the lowest space complexity as shown in this table. Additionally, to verity the practical ability of parameters reduction, we plot curves of number of parameters with respect to R in different tensor decomposition formats. As shown in Figure 4, to store size-$57,600 \times 256$ weight matrix used in [17], [20], [21], our HT-structure layer requires the fewest number of parameters with the same rank settings among all different types of tensor factorization methods. As will be shown in Section 5, such advantages will further be verified via empirical experimental results across various popular video recognition datasets.

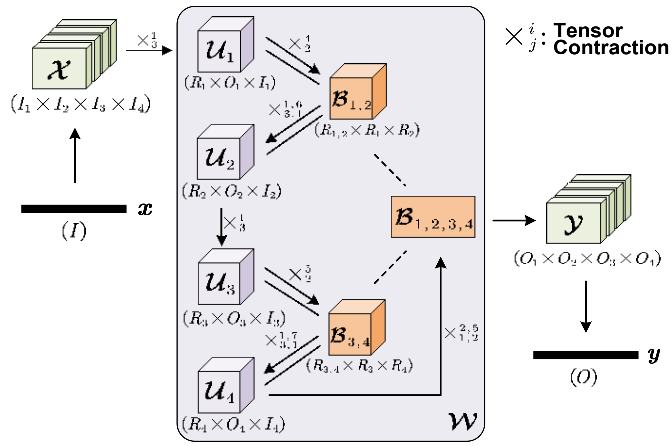


Fig. 3: Computation flow in the HT-layer for d=4. Arrows represent the directions of tensor contractions among the decomposed tensors. Here leaf frames $\{\mathcal{U}_i\}_{i=1}^{4}$ are 3-order tensors since the weight tensor is tensorized from a matrix. Reproduced from [22].

TABLE 1: Space complexities for different tensor-format linear layers. Here $R = \max_{s \in D} R_s$, $I' = \max_{k \in D} I_k$, $O' = \max_{k \in D} O_k$. Reproduced from [22].

Compressed Linear Layer	Space Complexity
Uncompressed	$O(I'O')$
Tensor Train (TT)-structure	$O(dI'O'R^2)$
Tensor Ring (TR)-structure	$O(dI'O'R^2)$
Block-Term (BT)-structure	$O(dI'O'R + R^d)$
Hierarchical Tucker (HT)-structure	$O(dI'O'R + dR^3)$

3.2 Fully Decomposing the HT-structure LSTM

3.2.1 Challenges of Fully Compressing LSTM

Recall that a typical LSTM model consists of input-to-hidden layer and hidden-to-hidden layer. Built on the top of the underlying HT structure for each component layer, performing fully decomposition on the entire LSTM models can evidently bring further performance improvement. To that end, one naive way is to simply compress each layer with HT decomposition. In other words, all weight matrices in the LSTM model are independently decomposed to HT format. Although this strategy can indeed bring full compression on the entire LSTM network, such straightforward layer-wise compression strategy suffers from huge accuracy drop. Figure 5 shows the experimental results using the layer-wise compression with HT decomposition. Here more than 2% accuracy drop can be observed as compared to the conventional input-to-hidden-only compression adopted in [17], [20], [21]. Overall, fully compressing the entire LSTM model using HT decomposition without accuracy drop is challenging and non-trivial.

3.2.2 Proposed FDHT-LSTM

To fully compress LSTM models without performance degradation, we propose a homogeneous compression method, namely fully decomposed HT (FDHT) decomposition, to obtain highly-compact LSTM models. Figure 1 illustrates the key idea of the proposed full decomposition. To be specific, in order to maximally leverage the linear correlation across all weight matrices

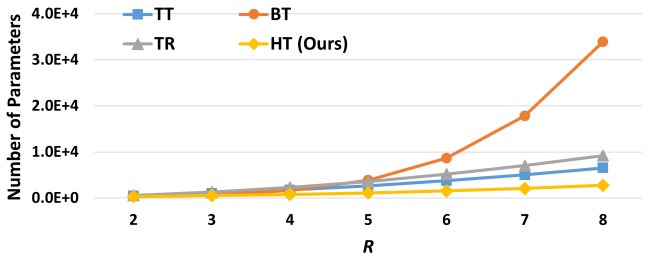


Fig. 4: Curves for number of parameters in different tensor-format layer with respect to tensor rank R. Here we use settings in the related works, i.e., d 5, pI1; ; I5q p8; 10; 10; 9; 8q, pO1; ; O5q p4; 4; 2; 4; 2q. Reproduced from [22].

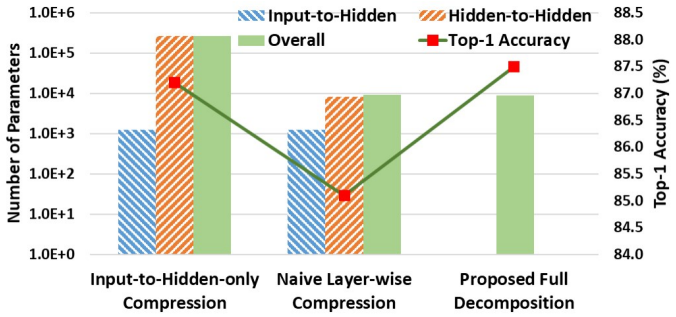


Fig. 5: Comparison between naive layer-wise compression and our proposed full decomposition in terms of number of parameters and test accuracy on UCF11 dataset. Reproduced from [22].

in the entire LSTM model, we first concatenate all the weight matrices as a single huge matrix such that the entire model can be considered as a "mega" linear layer. Then, at each time step, all the intermediate results only need one-time multiplication in the forward propagation process:

$$\begin{bmatrix} f_{rts} \\ \hat{u}_{rts} \\ \hat{c}_{rts} \\ \hat{o}_{rts} \end{bmatrix} = \begin{bmatrix} W_f \\ W_u \\ W_c \\ W_o \end{bmatrix} \begin{bmatrix} V_f \\ V_u \\ V_c \\ V_o \end{bmatrix} \begin{bmatrix} x_{rts} \\ h_{rt\,1s} \end{bmatrix} = W\, I_{rts}. \tag{7}$$

With the above interpretation, we can directly impose the desired HT structure on the integrated single huge matrix in an LSTM model. To be specific, following the scheme described in Section 3.1, we can tensorize and decompose the entire LSTM models to the HT format, and then perform forward pass as:

$$\begin{bmatrix} f_{rts} \\ c_{rts} \\ \hat{o}_{rts} \end{bmatrix} = Z_{rts} = HTL(W; I_{rts}). \tag{8}$$

Correspondingly, the outputs of the FDHT-LSTM can be calculated as follows:

$$\begin{aligned} f_{rts} &= \sigma(\hat{f}_{rts} + b_f) \\ u_{rts} &= \sigma(\hat{u}_{rts} + b_u) \\ c_{rts} &= f_{rts} \odot c_{rt\,1s} + u_{rts} \odot \tanh(\hat{c}_{rts} + b_c) \\ o_{rts} &= \sigma(\hat{o}_{rts} + b_o) \\ h_{rts} &= o_{rts} \odot \tanh(c_{rts}); \end{aligned} \tag{9}$$

where σ, $\tanh$ and $\odot$ are the sigmoid function, hyperbolic function and element-wise product, respectively.

3.2.3 HT-based Gradient Calculation

Notice that in order to ensure that the desired FDHT-LSTM model can be properly obtained, the backward propagation in the training process should also be reformulated to HT-based format. In general, given an HT-structure linear layer $W = \mathcal{U}_D$ and $\frac{\partial Y}{\partial \mathcal{U}_D} = X$, and defining s, $F(s)$ and $B(s)$ as non-root node, parent and sibling nodes of node s in the binary tree, respectively, the partial derivative of the output tensor with respect to frames can be recursively calculated as:

$$\frac{\partial Y}{\partial \mathcal{U}_s} = \mathcal{B}_{F(s)} \times_3 \mathcal{U}_{B(s)} \quad \text{where } \mathcal{B}_{F(s)} \in \mathbb{R}^{r_{1;;2B(s)} \times r_{1;;B(s)B(s)} \times r_{2;;F(s)F(s)} \times r_{3;;F(s)F(s)B(s)B(s)}}, \quad \frac{\partial Y}{\partial \mathcal{U}_{F(s)}}; \tag{10}$$

where $s = \min(s); s = \max(s)$. Notice that when $F(s)$ is equal to node D, the above recursive procedure terminates.

Furthermore, with Eq. (10) the gradients for leaf frames and transfer tensors can also be recursively calculated as:

$$\frac{\partial L}{\partial U_s} = \frac{\partial Y}{\partial U_s} \frac{\partial L}{\partial Y} = \mathcal{B}_{s(s)} \times_{3;;d-1} \mathcal{U}_{s(s)} \times_{1;;s-1;;s+1;;d} \frac{\partial L}{\partial Y}; \tag{11}$$

$$\frac{\partial L}{\partial \mathcal{U}_s} = \frac{\partial Y}{\partial \mathcal{U}_{s_1 s_2}} \frac{\partial L}{\partial Y} = \mathcal{B}_{s_1} \times_{2;;s-1;;s+1;;d} \mathcal{U}_{s(s)} \times_{3;;s-1;;s+2;;d} \mathcal{U}_{s(s)} \times_{1;;d} \frac{\partial L}{\partial Y}. \tag{12}$$

In general, our proposed "integrate-then-decompose" compression strategy can bring significant performance benefit for the FDHT-LSTM models. As shown in Figure 5, compared with the naive layer-wise compression strategy that suffers significant accuracy degradation, our proposed approach ensures high accuracy without performance loss. Furthermore, compared to the input-to-hidden-only compression counterpart, our FDHT-LSTM enjoys much smaller model size. More detailed and comprehensive empirical evaluation results across different datasets will be reported in Section 5.

4 The Proposed FDHT-LSTM: Hardware Architecture

Based on the above described FDHT-LSTM structure and algorithm, in this section we further develop the corresponding hardware architecture that can fully leverage the algorithmic benefits to support the efficient execution of the FDHT-LSTM model.

4.1 Analysis of Computation Flow: A 2-D Matrix Perspective

As described in Section 3, the overall computation scheme of FDHT-LSTM can be interpreted as a series of tensor contraction along a binary tree (see Figure 3). Consider tensor contraction (Eq. 1) is essentially a high-dimensional operation; while the underlying arithmetic and memory hardware components can only support 2-D operation, a re-analysis of the computational flow, from 2-D matrix perspective, is very necessary and desired.

Figure 6 illustrates the overall computational flow of the proposed FDHT-LSTM in the 2-D matrix format. Here $\mathcal{B}^1_{1234}$, $\mathcal{B}^1_{12}$, $\mathcal{B}^1_{34}$, $\mathcal{U}^1_1$, $\mathcal{U}^1_2$, $\mathcal{U}^1_3$ and $\mathcal{U}^1_4$ are the weight matrices of the flatten high-order tensors, $\mathcal{X}^1$ is the reshaped input matrix

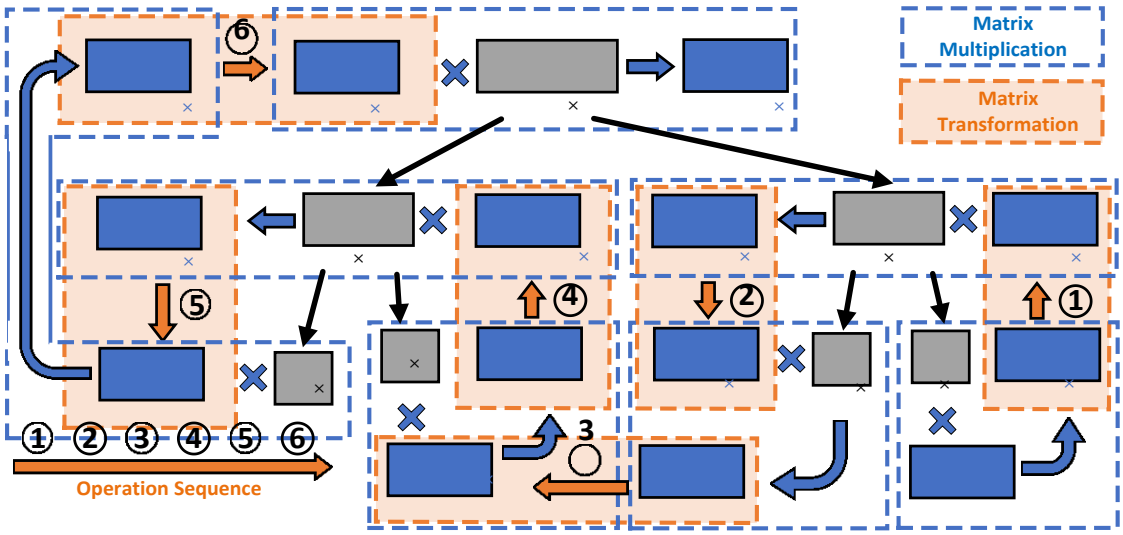


Fig. 6: Computation flow of high-order FDHT-LSTM from the perspective of 2-D matrix multiplication and transformation.

data, and T is the intermediate result. Notice that here T cannot be multiplied with the next weight matrix directly due to the mismatch of matrix dimension. Therefore, T needs to be first transformed to T^{1} to ensure functional validity.

As shown in Figure 6, a total of six matrix transformations (from T to T^{1}) is required in the entire computation flow of FDHT-LSTM. After further analysis, we can obtain two important observations: First, these transformations are not simple matrix reshape but essentially permutation that requires complex matrix operations such as row/column merge and concatenation etc.; Second the six transformations can be categorized to three types. To be specific, Type-I (Transformation ① and ④) is to convert $T \; P \; R^{pA_pB_1B_2q}$ to $T^{1} \; P \; R^{pAB_1qB_2}$, and the mapping principle for this transformation is as:

$$T pm; nq \text{ æ } T^{1} pp; qq; \tag{13}$$

where $m \; a, n \; pb_1 \; 1q \; B_2 \quad b_2, p \; pa \; 1q \; B_1 \quad b_1, q \; b_2, a \; 1; 2; :::; A, b_1 \; 1; 2; :::; B_1$ and $b_2 \; 1; 2; :::; B_2$.

Type-II (Transformation ② and ⑤) is to convert $T \; P \; R^{pA_1A_2A_3qpB_1B_2q}$ to $T^{1} \; P \; R^{pA_1A_3B_1qpA_2B_2q}$, and the mapping principle for this transformation is as:

$$T pm; nq \text{ æ } T^{1} pp; qq; \tag{14}$$

where $m \; pa_1 \; 1q \; A_2 \; A_3 \quad pa_2 \; 1q \; A_3 \quad a_3, n \; pb_1 1q B_2 \quad b_2, p \; pa_1 1q A_3 B_1 \quad pa_3 1q B_1 \quad b_1, q \; pa_2 \; 1q \; b_2, a_1 \; 1; 2; :::; A_1, a_2 \; 1; 2; :::; A_2, a_3 \; 1; 2; :::; A_3, b_1 \; 1; 2; :::; B_1$ and $b_2 \; 1; 2; :::; B_2$.

Type-III (Transformation ③ and ⑥) is to convert $T \; P \; R^{pA_1A_2A_3qB}$ to $T^{1} \; P \; R^{pA_1A_3BqA_2}$, and the mapping principle for this transformation is as:

$$T pm; nq \text{ æ } T^{1} pp; qq; \tag{15}$$

where $m \; pa_1 \; 1q \; A_2 \; A_3 \quad pa_2 \; 1q \; A_3 \quad a_3, n \; b, p \; pa1qA_3B \; pa_3 1qB \; b, q \; a_2, a_1 \; 1; 2; :::; A_1, a_2 \; 1; 2; :::; A_2, a_3 \; 1; 2; :::; A_3$ and $b \; 1; 2; :::; B$.

4.2 Access Conflict on Conventional SRAM Architecture

Without losing the generality, we use the basic matrix transformation $M \; P \; R^{pA_1A_2qpB_1B_2q}$ æ $M^{1} \; P \; R^{pA_1B_1qpA_2B_2q}$ to elaborate the challenges incurred by the complicated transformation above, where $A_1 \; A_2 \; B_1 \; B_2 \; 2$. As shown in Figure 7, when using conventional SRAM architecture, in the writing phase, each row of the matrix is written to one address (one row) in this SRAM in one cycle. For example, data 01; 02; 03 and 04 is written to the address 01 of the SRAM. However, the access conflict happens in the read phase. When reading the SRAM, the desired data is 01; 02 and 04; 05. These four data are located at two different addresses 01 and 02, and these two addresses can not be accessed simultaneously, thereby causing access conflict.

The basic solution to the access conflict problem is relocating one address multiple times (as shown in Figure 7). More specifically, in cycle 1, address 01 is located and data 01; 02; 03; 04 is read out from the SRAM. Data 01; 02 is saved in the register file, while 03; 04 is the undesired data and discarded. Similarly, in cycle 2, address 02 is located; 05; 06; 07; 08 is read out from SRAM, and 05; 06 is stored in the register file and 07; 08 is discarded. In the assemble unit, 01; 02 and 05; 06 form the data array 01; 02; 05; 06. However, such a solution suffers two main limitations:

1. Additional memory overhead is needed to support matrix transformation. Since the matrix transformation cannot be executed in an on-the-fly way when using conventional memory architecture, an additional register file is necessary to save the desired data. Thus, this solution significantly increases the cost of the register file, reducing the area and energy performance of the overall hardware.

2. The write and reading scheme becomes much more complicated because of frequent re-locating of the memory address. Also, notice that the example we raise here is only a simple case, and in practical workload, the matrix transformation is very large and complicated. Hence it is not feasible to read all the data to register files and assemble them when using conventional SRAM architecture.

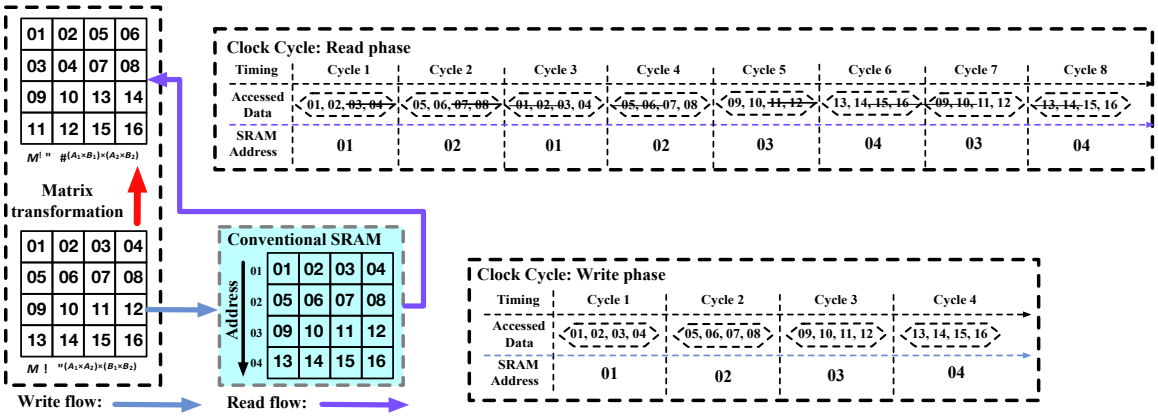


Fig. 7: Data access in conventional SRAM architecture.

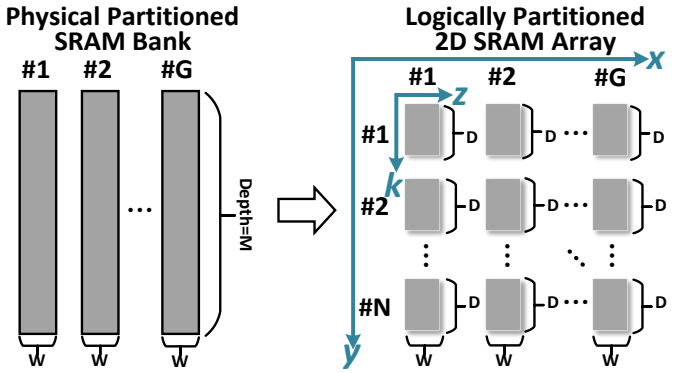


Fig. 8: The physical SRAM bank and logically partitioned 2-D SRAM array. Here W and M are the width and depth of one SRAM bank, respectively. D is the depth of one segment of SRAM bank, where M D N. Here D can be reconfigured according to workload.

4.3 2-D SRAM Array

In order to implement the desired matrix transformation, which is essentially the 2-D mapping of high-order tensor transpose, we propose to read and write the data in the 2-D SRAM array. In other words, we aim to increase the dimension of the SRAM from physical 2-D to "virtually" 4-D, to better support the operation in the high-order tensor operation. As illustrated in Figure 8, our proposed SRAM array physically consists of G banks of SRAM, where each of them has width of W and depth of M. Based on such fixed arrangement, the controller will logically partition the SRAM array to more fine-grained format when supporting different workloads with various matrix sizes. To be specific, according to the demand of the workload, each SRAM bank is logically partitioned into N segments along the depth dimension, and each segment has a depth of D where M N D. Notice that here G, M and W are the fixed dimensions of the entire SRAM module, and D and N are reconfigurable and determined by the main controller, thereby providing the controller sufficient capability to adjust the granularity of the memory operation for different workloads.

To achieve this goal without introducing any memory access conflict, delicate design of memory read/write scheme is required and will be described next.

4.4 Matrix Transformation on 2-D SRAM Array

Based on the 2-D SRAM array, the corresponding conflict-free memory access scheme can be developed. Without loss of generality, in this subsection we first study the read and write schemes for a basic transformation M P $R^{pA_1A_2qpB_1B_2q}$ æ M^{1} P $R^{pA_1B_1qpA_2B_2q}$. The obtained outcome will further serve as the foundation of implementing the specific matrix transforma-tions (Eq.13 – Eq.15) used in FDHT-LSTM.

To facilitate the notation, as illustrated in Figure 8, the bank and segment indices of the 2-D SRAM array are denoted as x and y, respectively, where x 1; 2; :::; G and y 1; 2; :::; N. In addition, z and k are the row and depth indices in each SRAM segment, respectively, where z 1; 2; :::; W and k 1; 2; :::; D. Following such notation, one data entry in one SRAM segment, one row in one SRAM segment, one SRAM segment, and one SRAM bank can be represented as Spx; y; k; zq, Spx; y; k; :q, Spx; y; :; :q and Spx; :; :; :q, respectively. Also, we use Mpa; :q and Mp:; bq to represent the a-th row vector and the b-th column vector of M, respectively. In addition, Mpa; b : cq and Mpa : b; cq denote the part of Mpc; :q and Mp:; cq that are with column and row indices from a to b.

4.4.1 Memory Write Scheme

Next we describe memory write scheme for the example transformation M P $R^{pA_1A_2qpB_1B_2q}$ æ M^{1} P $R^{pA_1B_1qpA_2B_2q}$. Here D, as the granularity of 2-D SRAM array, is config-ured as B_2 to support the operation for this example. Initially, Mp1; 1 : B_2q is written along the z dimension into Sp1; 1; 1; :q as the first row of SRAM segment Sp1; 1q. Then, if we interpret the B_2 data entries in one row as a group, and one row of M contains B_1 groups, these B_1 data groups are scheduled to be sequentially written into Sp1; 1; :; :q in a row-wise way. For instance, the data entries of Mp1; B_2 1 : 2 B_2q are written into Sp1; 1; 2; :q, and the data entries of Mp1; pB_1 1q B_2 1 : B_1 B_2q are written into Sp1; 1; B_1; :q. After that, a row of data entries of M is stored in one segment of 2-D SRAM array. Then, A_2 rows of M are written into A_2 SRAM segments along the x dimension, e.g., Sp1; 2; :; :q stores the data of Mp2; :q and Sp1; S_2; :; :q stores the data of MpA_2; :q. By using this way, Mp1 : A_2; :q is written into one SRAM bank Sp1; :; :; :q. Writing the rest of the data in M to SRAM follows the similar way, e.g., Sp2; :; :; :q stores the data of MpA_2 1 : 2 A_2; :q, and SpA_1; :; :; :q stores the data of MppA_1 1q A_2 1 : A_1 A_2; :q. Figure 9 shows the detailed scheme for an example matrix transformation, where

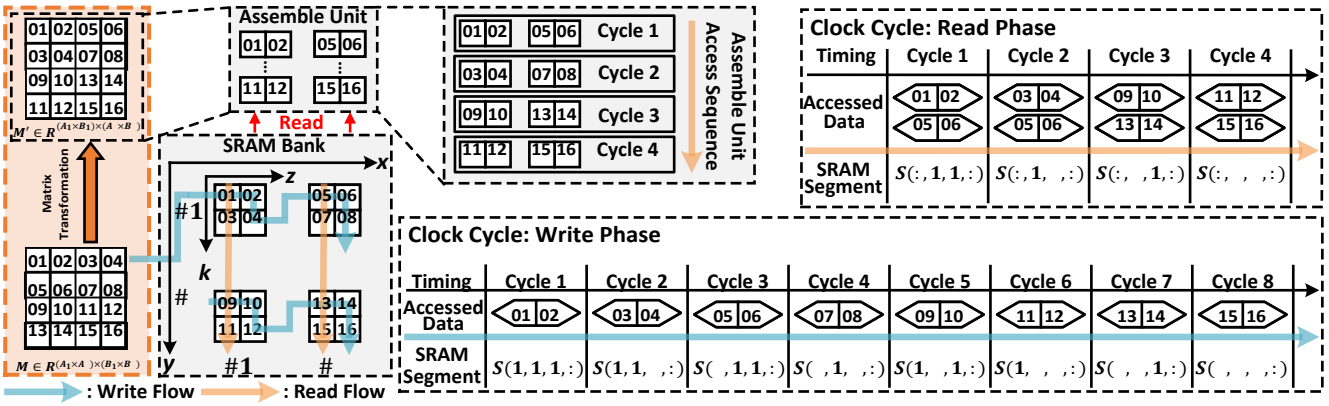


Fig. 9: The proposed write and read scheme to 2-D SRAM array for a basic matrix transformation.

G 2, M 4, W 2, and A_1 A_2 B_1 B_2 2. It is seen that with such writing scheme the 2-D SRAM array stores the re-arranged data, which will be ready to realize the desired transformation after proper reading scheme.

4.4.2 Memory Read Scheme

In the reading phase, the data in G banks are read simultaneously in a row-wise way. To be specific, the data in Sp:; 1; :; kq are first read in the first cycle, and then they are sent to assemble unit to form M 1 p1; :q. In the following cycles, such operation is repeated row by row along the k and y dimensions. Figure 9 shows the details of reading scheme for the example matrix transformation.

In general, the proposed 2-D SRAM with a corresponding write and read scheme can solve the challenges mentioned above:

1. The matrix transformation can be executed in an on-the-fly way. In the reading phase, all the to-be-read data is the desired data and only a small register file is needed to save the data. In our FDHT-LSTM, the size of the register file is only 448Bytes.

2. Since in the writing and reading phase, the desired memory address increases sequentially, it is very easy to design and implement the address controller. Also, because the 2-D SRAM architecture is logically split, it is very easy to be reconfigured and very flexible for different matrix transformations with various shapes.

4.5 Realizing Matrix Transformation for FDHT-LSTM

Recall that the read/write access schemes described in Section 4.4 are designed for a basic matrix transformation that is different from the ones described in Eq. 13 – Eq. 15. In this subsection we leverage the approach developed in Section 4.4 to further realize the three matrix transformations specifically adopted in the computation flow of FDHT-LSTM.

4.5.1 Memory Access Scheme for Type-I Transformation

In this case D is set as 1. During the memory writing phase, consider B_2 data as a group, then the data in each group are written to one row of SRAM segment along the z dimension. Since each row of T contains B_1 data groups, they are written into one row of SRAM array along the x dimension sequentially. In other words, one row of SRAM array Sp:; 1; :; :q stores one row of T as Tp1; :q. Following this strategy, the A rows of T are written into the SRAM array row by row along the y dimension in a sequential way. Then during the memory reading phase, consider

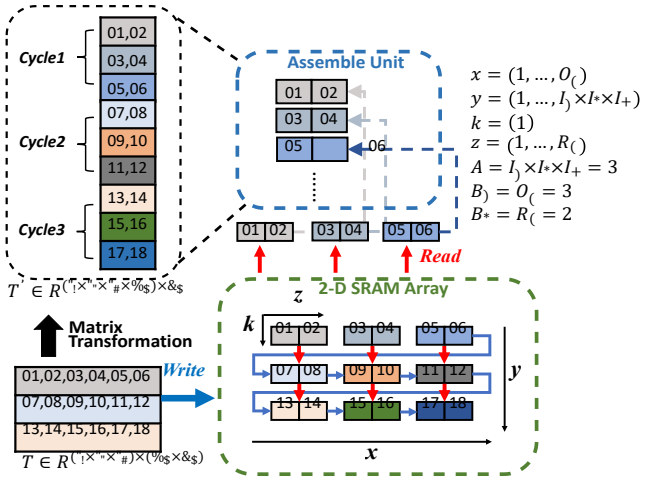


Fig. 10: Example memory write and read schemes for Transformation 1.

each row of SRAM array Sp:; 1; :; :q has B_1 B_2 data; hence the assemble unit will partition one row data to form a data array with B_1 columns and B_2 rows. By repeating such reading operation along the y dimension, T 1 can be read out and sent to PE array for further processing. Figure 10 illustrates the details of memory write and read schemes to an example Transformation (1), where A I_1 I_2 I_3 3, B_1 O_4 3 and B_2 R_4 2.

4.5.2 Memory Access Scheme for Type-II Transformation

In this case, the transformation can be rewritten as T P $R^{pA_1A_2qppA_3B_1qB_2q}$ æ T^{1} P $R^{pA_1pA_3B_1qqpA_2B_2q}$, and then it becomes the basic format example described in Section 4.4 with D A_3 B_1. Then, during the writing phase, we rearrange the data of T along B_2, A_3B_1, A_2 and A_1 dimensions on the z, k, x and y dimensions of 2-D SRAM array, respectively. In the memory reading phase, A_2 B_2 data entries in the SRAM array are read simultaneously and combined as one row of T . 1

In general, by reading along the k and y dimensions of 2-D SRAM array, the row data of T 1 as A_1 A_3 B_1 entries can be obtained. Figure 11 illustrates the details of memory access scheme to an example Transformation (2), where B_2 R_3 2, A_3B_1 O_4R_{34} 3, A_2 I_3 3 and A_1 I_1I_2 2.

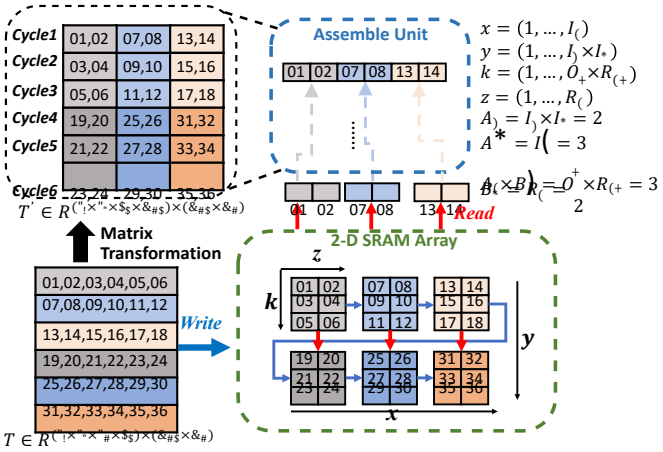


Fig. 11: Example memory write and read schemes for Transformation 2.

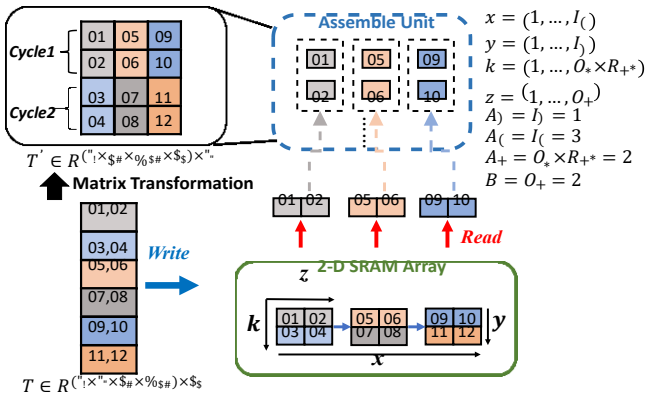


Fig. 12: Example memory write and read schemes for Transformation 3.

4.5.3 Memory Access Scheme for Type-III Transformation

In the writing phase of this case, consider each row of T as a group, and the data in each group is written along the dimension z in each SRAM segment, e.g., Sp1; 1; 1; :q stores the data in Tp1; :q. Consequently, in order to keep A_3 in the column index of T^1, D should be set as A_3 to ensure that each A_3 rows of T are written along the k dimension in each SRAM segment, e.g., Tp1 : A_3; :q is stored in Sp1; 1; :; :q. In addition, the data of T along A_2 and A_1 dimensions are re-arranged on the x and y dimensions of 2-D SRAM array, respectively, to guarantee A_2 appear in the row index of T^1. During the reading phase, A_2 B data entries in the SRAM array are read simultaneously, and they are reshaped into an array with A_2 columns and B rows in the assemble units. Figure 12 illustrates the details of memory read and write schemes of the Transformation 3, where B O_3 2, A_3 O_4 R_{34} 2, and A_1 I_1 1.

4.6 Overall Architecture

Based on the proposed 2-D SRAM array and the corresponding read/write access schemes, the hardware architecture that supports the execution of FDHT-LSTM models can be developed. As illustrated in Figure 13, the datapath of the accelerator is a PE Array that consists of multiply-accumulators (MACs) to perform

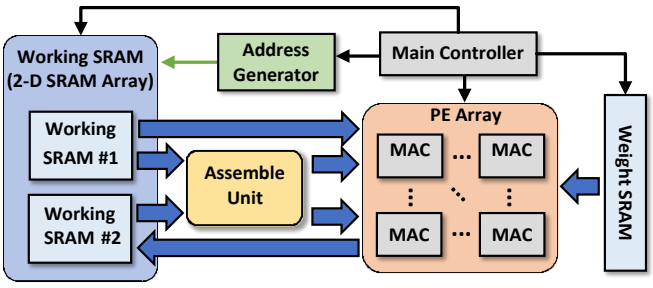


Fig. 13: The overall hardware architecture.

TABLE 2: Compression settings for HT-layer. Here because YTC dataset is smaller than UCF11 dataset, we choose smaller R (non-leaf) for LSTM-YTC to achieve higher compression ratio with still preserving high accuracy.

Model	LSTM-UCF11	LSTM-YTC
Size	p57600; 256q	p57600; 256q
d	4	4
I	r16; 16; 16; 15s	r16; 16; 16; 15s
O	r4; 4; 4; 4s	r4; 4; 4; 4s
R (Leaf)	14	14
R (Non-leaf)	12	11
Compr. Ratio	6,726	7,117

matrix multiplication. Notice that because FDHT-LSTM operates on the the dense matrices rather than the sparse ones, the multiplier utilization is very high. In addition, the weight of the factorized tensor cores and the intermediate results are stored in the weight SRAM and working SRAM, respectively. Notice that here only working SRAM is in the format of 2-D array since the desired matrix transformation is only involved with the intermediate results. The read/write access schemes described in Section 4.5 are mapped into an individual address generator, which receives the configuration information from main controller for different workloads.

5 Evaluation

5.1 Performance of FDHT-LSTM Model

We evaluate the performance of our proposed FDHT-LSTM approach on various video recognition tasks and compare it with other state-of-the-art tensor decomposition-based LSTM compression methods. In particular, in order to verify the benefits of our full decomposition strategy and fairly compare the HT decomposition with other tensor decomposition for LSTM compression, we also conduct experiments on only performing HT decomposition on the input-to-hidden layer of LSTM models.

5.1.1 Experimental Settings

For fair comparison, we set d, the order of the target tensors of the tensorized input, output and weights, identical to the settings in prior works [20] and [21]. For the tensor ranks, we also follow the prior works to select r towards reaching a good balance between compression ratio and model accuracy. For the setting of other hyper-parameters, dropout rate and batch size are set as 0.25 and 16, respectively, and ADAM is selected as the underlying training optimizer with weight decay of 0.001.

TABLE 3: Compression performance of different tensor decomposed-based LSTM models on UCF11 dataset.

Model	Number of Parameters		Top-1 Acc. (%)
	Input-to-Hidden	Overall	
LSTM	58.98M	59.24M	69.7
TT-LSTM [20] (ICML'17)	3,360	265.50K (223)	79.6
BT-LSTM [21] (CVPR'18)	3,387	265.53K (223)	85.3
TR-LSTM [17] (AAAI'19)	1,725	263.87K (225)	86.9
HT-LSTM (Ours)	1,245	263.39K (225)	87.2
FDHT-LSTM (Ours)	N/A	8,808 (6,726)	87.5

TABLE 4: Compression performance of different tensor decomposed-based LSTM models on Youtube celebrities face dataset.

Model	Number of Parameters		Top-1 Acc. (%)
	Input-to-Hidden	Overall	
LSTM	58.98M	59.24M	33.2
TT-LSTM [20] (ICML'17)	3,392	265.54K (223)	75.5
HT-LSTM (Ours)	810	262.95K (225)	88.1
FDHT-LSTM (Ours)	N/A	8,324 (7,117)	88.2

TABLE 5: Experimental comparison among FDHT-LSTM and other non-tensor decomposition models, e.g., DML-PV [5], VGGFACE + RRNN [14] and VGG16-GCR [15], on Youtube celebrities face dataset.

Model	Number of Parameters	Top-1 Acc. (%)
DML-PV	220K	82.8
VGGFACE + RRNN	¥42M	84.6
VGG16-GCR	138M	82.9
HT-LSTM (Ours)	263K	88.1
FDHT-LSTM (Ours)	8,324	88.2

5.1.2 Results on UCF11 Dataset

The UCF11 dataset [16] is a set of 1,600 human action video clips which can be categorized as 11 classes. The resolution of each clip is 320 240. We follow the prior works [21] [17] to pre-process the video data. To be specific, the resolution of each video clip is downgraded to 160 120, then we randomly sample 6 frames from each clip and combine them as a single sequential data point. In the original uncompressed LSTM model, there are 4 input-to-hidden layers and 4 hidden-to-hidden layers. We also follow the prior works to set the input size and the number of hidden states in the baseline model as 160 120 3 57; 600 and 256, respectively. For the compressed FDHT-LSTM model, the concatenated vector I can be reshaped as a tensor of shape as 16 16 16 15, and the hidden state is reshaped to a tensor of shape as 4 4 4 4. In addition, all leaf and non-leaf ranks are set as 14 and 12, respectively.

The compression performance of the proposed FDHT-LSTM and other state-of-the-art tensor decomposed LSTM are summarized in Table 3. It is seen that compared with the orig-inal uncompressed LSTM model with 59 million parameters, our proposed FDHT-LSTM only needs 8,808 parameters while resulting in a 17.8% accuracy increase. Compared with other tensor decomposition-based LSTM compression methods, i.e., TT-LSTM, BT-LSTM and TR-LSTM, FDHT-LSTM enjoys order-of-magnitude reduction in model parameters and at least 0.6% higher accuracy.

5.1.3 Results on Youtube Celebrities Face Dataset

Youtube celebrities face dataset [13] contains 1,910 video clips with different resolutions which are categorized as 47 labels. Similar to the case of UCF11 dataset, we also follow the prior works [21] [17] for data pre-processing. To be specific, we scale down all the video clips to 160 120 and randomly sample 6 frames to form a single sequence. Other experimental settings are also consistent with the case of UCF11 dataset except that all the non-leaf ranks are set as 11.

Table 4 summarizes the compression performance on this dataset for different tensor decomposed LSTMs. It is seen that compared to the baseline LSTM model our FDHT-LSTM can provide as high as 7,117 compression ratio with test accuracy increase. Compared with the state-of-the-art TT-LSTM, our FDHT-LSTM still enjoys 6,894 smaller model size and 12.7% higher accuracy.

Besides, we also compare the performance of FDHT-LSTM with several other tensor decomposition-free models for video recognition, and the results are summarized in Table 5. It is seen that our FDHT-LSTM can outperform the state-of-the-art work [14] with 3.6% higher accuracy and much smaller model size.

Rank Selection. As analyzed before, for compressing large LSTM models, HT decomposition can improve accuracy with very high compression ratio because it can effectively reduces the structural redundancy. Therefore, for high rank setting such as R 32, the accuracy is not as good as low rank setting because considerable unnecessary redundancy still exists and it is not friendly for training. On the other hand, for extremely low rank setting, e.g., R 2, the model size is too small and thus the model capacity will be hurt. Therefore, as shown in Table 6, in order to achieve good balance between compression rate and model accuracy, R 12 and R 11 are set for LSTM-UCF11 and LSTM-Youtube, respectively.

5.2 Performance of FDHT-LSTM Hardware

5.2.1 Configuration of Experiment and Design Example

A bit-accurate cycle-accurate simulator is developed to model the high-level behavior of the proposed FDHT-LSTM architecture. Then, we build a Verilog-based RTL model and verify the functional validation. This model is synthesized with CMOS 28nm library. Table 7 shows the detailed configuration information of the design example based on the proposed FDHT-LSTM architecture. Here the overall hardware consists of 16 PEs, and each PE is equipped with 16 16-bit multipliers and 16 24-bit accumulators, hence 256 MAC operations are performed in each clock cycle. The working memory is composed of 14 banks of SRAM, and

TABLE 6: Accuracy comparison with various R on LSTM-UCF11 and LSTM-Youtube.

Models	R	Top-1 Acc.(%)
LSTM-UCF11	2	81.4
	12	87.5
	32	79.3
LSTM-Youtube	2	78.9
	11	88.2
	32	77.4

TABLE 7: Architectural configuration for design example.

System Parameter		Value
PE		16
Quantization		16-bit
Memory Parameter		
Working Memory	Width (W)	256-bit
	Depth (M)	2048
	# of Banks (G)	14
	Total Size	1.7MB
Weight Memory	Width	16-bit
	Depth	8808
	Size	17.2KB
PE Parameter		
Amount of Multiplier		16
Amount of Accumulator		16

each SRAM bank has the width of 256 bits and the depth of 2048, thereby leading to 875KB budgeted capacity that is sufficient for storing the activation and intermediate result in most applications. Since the working SRAM contains two copies to serve as a ping-pong buffer, the total size of the working SRAM is 875KB 2 1:70MB. For the weight SRAM, it has 16-bit width with depth of 8808 to store the weight parameters of the compressed FDHT-LSTM models. Thanks to the ultra-strong compression capability of FDHT technique, the budgeted 17.2KB weight SRAM can support the storage of many large-scale LSTM models.

5.2.2 Comparison with GPU

Though the decomposed model reduces the weight size significantly and enjoys the parallel processing, the computation flow of FDHT-LSTM contains complicated matrix transformation, which is not naturally supported by GPU in an efficient way. We implement the decomposed LSTM-Youtube and LSTM-UCF11 on NVIDIA RTX A6000, and compare the inference speed with our FDHT hardware accelerator. The result shows that FDHT-LSTM hardware achieves 2.30 and 1.85 inference speedup than GPU for executing LSTM-Youtube and LSTM-UCF11, respectively.

5.2.3 Comparison with EIE and TIE

In this subsection, we compare the proposed architecture with two prior compressed model-oriented hardware accelerators EIE [12] and TIE [6]. Table 9 summarizes the implementation results for the three listed ASIC designs. Table 8 shows the compression performance of FDHT-LSTM, EIE and TIE, where top-1 accuracy is the accuracy that the recognition result with the highest probability is exactly the expected answer. It is seen that our FDHT-LSTM outperforms EIE and TIE with respect to compression rate and accuracy performance on both UCF11 and Youtube datasets.

TABLE 8: Compression performance of EIE, TIE and our FDHT-LSTM on UCF11 and Youtube dataset. Top-1 accuracy is the accuracy that the recognition result with the highest probability is exactly the expected answer.

	Design	Compres. Rate	Top-1 Acc.(%)
UCF11	EIE	117	82.9
	TIE	4954	79.6
	FDHT-LSTM (Ours)	6726	87.5
Youtube	EIE	111	84.2
	TIE	4608	78.8
	FDHT-LSTM (Ours)	7117	88.2

Comparison with EIE. EIE is a sparse model-oriented hardware architecture. Consider EIE is implemented with a different technology node, as reported in Table 9, the performance metrics of EIE are projected to 28nm technology for fair comparison. Here the projection follows the scaling rule adopted in [12]: linear, quadratic and constant scaling for clock frequency, silicon area and power consumption, respectively. In addition, considering the different budgeted arithmetic and memory resource, Figure 14 compares our proposed FDHT-LSTM with EIE with respect to different hardware performance metrics. It is seen that the FDHT-LSTM achieves 1:69, 8:99 and 6:22 increase in throughput, area efficiency and energy efficiency on UCF11, and 2:06, 10:94 and 7:58 increase in throughput, area efficiency and energy efficiency on Youtube respectively.

Comparison with TIE. TIE is the state-of-the-art tensor train (TT) decomposed model-oriented hardware accelerator, which is the most related work to our FDHT-LSTM architecture. Figure 14 shows the performance comparison between the two designs in terms of processing throughput, area efficiency and energy efficiency on two LSTM workloads [2]. It is seen that for executing LSTM-Youtube workload, our proposed FDHT-LSTM architecture achieves 2:5, 1:46 and 2:41 increase in throughput, area efficiency and 2:41 energy efficiency, respectively. For LSTM-UCF workload, our FDHT-LSTM design also outperforms TIE with 1:9 higher throughput, 1:83 higher energy efficiency and comparable area efficiency.

5.2.4 Flexibility

The proposed FDHT-LSTM architecture can provide high flexibility to support different compressed LSTM models with different topology settings. Considering the rank R is the most important parameters that directly determines the storage requirement and computational costs of the underlying models, we study the flexibility of the proposed hardware architecture with respect to different R's. As reported in Figure 15, FDHT-LSTM hardware demonstrates strong flexibility.

2. Because TIE is based on the TT decomposition that only compresses the input-to-hidden layer of the original LSTM, the reported performance of TIE is evaluated on that single layer.

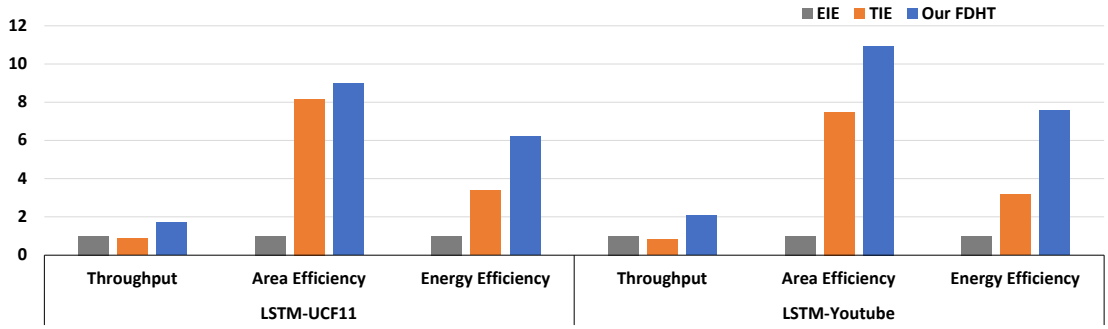


Fig. 14: Hardware performance comparison among FDHT-LSTM, TIE and EIE.

TABLE 9: Comparison on hardware performance.

Design	EIE [12]	TIE [6]	Our FDHT
Compression Approach	Pruning	Tensor Train (TT)	Hierarchical Tucker (HT)
CMOS Technology	28nm (projected)	28nm	28nm
Frequency (MHz)	1280	1000	1000
Quantization Scheme	4-bit index 16-bit weight	16-bit	16-bit
Area (mm)	15.7	1.74	2.96
Power (mW)	590	154.8	160.4

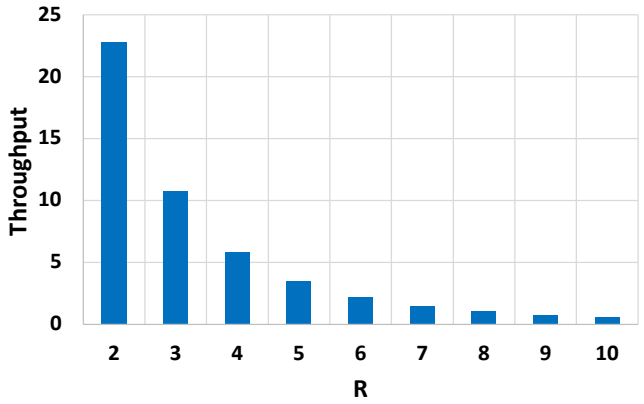


Fig. 15: Flexibility of FDHT with respect to different decomposition ranks.

6 Related Work

Customized hardware accelerators for deep neural networks have been extensively studied in recent years. To accelerate the execution of convolutional neural networks (CNNs) in a real-time and energy-efficient way, a series of CNN hardware architecture have been proposed in [4] [24] [2]. In particular, fully leveraging the sparsity is a very important optimization strategy towards high-performance CNN hardware accelerators [18] [1] [8].

Different from computation-bounded CNNs, LSTMs are inherently memory-bounded and require very large model size because of its component linear layer. Therefore, the focus of efficient LSTM hardware design is to efficiently integrate upper-level model compression algorithm to the lower-level architecture optimization. Motivated by this design philosophy, several compressed model-oriented LSTM hardware have been developed in [19] [11] [12] [6]. To be specific, EIE [12] and ESE [11] are built on pruned model and the focus of their architectural optimization is to properly leverage sparsity opportunities. C-LSTM [19] aims to use FFT-based compression compression to accelerate LSTM, and hence its underlying architecture mainly consists of optimized FFT modules. TIE is the first tensor decomposition-based DNN hardware architecture, which is the most closed work to our design. By leveraging the tensor train decomposition, TIE [6] can achieve competitive hardware performance with good resource utilization.

7 Conclusion

In this paper, we propose algorithm and hardware co-design for FDHT-LSTM, a ultra-compact high-performance compressed LSTM network. By fully decomposing the entire LSTM models via hierarchical Tucker decomposition, the entire LSTM model size can be significantly reduced. Meanwhile, an energy-efficient customized hardware architecture with delicate design of memory access scheme is developed. The proposed algorithm and hardware are empirically evaluated on different video recognition datasets and LSTM workloads. The experimental results show that our proposed FDHT-LSTM networks and the corresponding hardware accelerator can achieve very high model performance and hardware performance as compared to the state-of-the-art designs.

Acknowledgments

This work was partially supported by National Science Foundation under Grant CCF-1955909.

References

[1] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, et al. Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. IEEE transactions on neural networks and learning systems, 30(3):644–656, 2018.

[2] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 1–12. IEEE, 2016.

[3] Elham Azari and Sarma Vrudhula. An Energy-Efficient Reconfigurable LSTM Accelerator for Natural Language Processing. In 2019 IEEE International Conference on Big Data (Big Data), pages 4450–4459, Los Angeles, CA, USA, Dec. 2019. IEEE.

[4] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. ACM SIGARCH Computer Architecture News, 44(3):367–379, 2016.

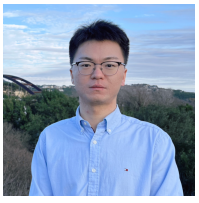
[5] Gong Cheng, Peicheng Zhou, and Junwei Han. Duplex metric learning for image set classification. IEEE Transactions on Image Processing, 27(1):281–292, 2017.
[6] Chunhua Deng, Fangxuan Sun, Xuehai Qian, Jun Lin, Zhongfeng Wang, and Bo Yuan. TIE: energy-efficient tensor train-based inference engine for deep neural network. In Proceedings of the 46th International Symposium on Computer Architecture, pages 264–278, Phoenix Arizona, June 2019. ACM.
[7] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. page 10.
[8] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and TN Vijaykumar. Sparten: A sparse tensor accelerator for convolutional neural networks. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pages 151–165, 2019.
[9] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with Deep Bidirectional LSTM. In 2013 IEEE Workshop on Automatic Speech Recognition and Understanding, pages 273–278, Olomouc, Czech Republic, Dec. 2013. IEEE.
[10] Wolfgang Hackbusch and Stefan Kư hn. A new scheme for the tensor representation. Journal of Fourier Analysis and Applications, 15(5):706–722, 2009.
[11] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 75–84, 2017.
[12] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pages 243–254, Seoul, South Korea, June 2016. IEEE.
[13] Minyoung Kim, Sanjiv Kumar, Vladimir Pavlovic, and Henry Rowley. Face tracking and recognition with visual constraints in real-world videos. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8. IEEE, 2008.
[14] Yang Li, Wenming Zheng, Zhen Cui, and Tong Zhang. Face recognition based on recurrent regression neural network. Neurocomputing, 297:50–58, 2018.
[15] Bo Liu, Liping Jing, Jia Li, Jian Yu, Alex Gittens, and Michael W Mahoney. Group collaborative representation for image set classification. International Journal of Computer Vision, 127(2):181–206, 2019.
[16] Jingen Liu, Jiebo Luo, and Mubarak Shah. Recognizing realistic actions from videos "in the wild". In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1996–2003. IEEE, 2009.
[17] Yu Pan, Jing Xu, Maolin Wang, Jinmian Ye, Fei Wang, Kun Bai, and Zenglin Xu. Compressing recurrent neural networks with tensor ring for action recognition. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 4683–4690, 2019.
[18] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. ACM SIGARCH Computer Architecture News, 45(2):27–40, 2017.
[19] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. C-lstm: Enabling efficient lstm using structured compression techniques on fpgas. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 11–20, 2018.
[20] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In International Conference on Machine Learning, pages 3891–3900. JMLR. org, 2017.
[21] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 9378–9387, 2018.
[22] Miao Yin, Siyu Liao, Xiao-Yang Liu, Xiaodong Wang, and Bo Yuan. Towards extremely compact rnns for video recognition with fully decomposed hierarchical tucker structure. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12085–12094, 2021.
[23] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. Long-term forecasting using higher order tensor rnns. arXiv preprint arXiv:1711.00073, 2017.
[24] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays, pages 161–170, 2015.

Yu Gong received the master's degree in Shanghai Jiao Tong University, and bachelor's degree in Wuhan University of Technology. Currently, he is a Ph.D. student in Electrical and Computer Engineering at Rutgers University. His research interests include high-performance architecture for AI and hardware-software co-design.

Miao Yin is a Ph.D. student in Electrical and Computer Engineering at Rutgers, The State University of New Jersey. His research is focused on algorithm-hardware co-designed energy-efficient AI system (e.g., on-device image and action recognition) based on higher-order tensor decomposition and advanced optimization. He serves as a PC member of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

Lingyi Huang received the master's degree in electrical engineering from New York University and the bachelor's degree in electrical engineering from Xidian University. He is currently pursuing the PhD degree in computer engineering at Rutgers University.

Chunhua Deng is a senior staff design engineer at ScaleFlux Inc. He received his Ph.D. degree from the Department of Electrical and Computer Engineering at Rutgers University. He received his bachelor's degree and master's degree from China University of Petroleum, Beijing Institute of Technology, respectively. His research interests include machine learning, computer architecture, and VLSI design.



Yang Sui is a Ph.D. student in Electrical and Computer Engineering at Rutgers, The State University of New Jersey. His research is focusing on algorithm-hardware co-designed efficient AI with model compression based on pruning, low-rank decomposition, and advanced algorithms.

Bo Yuan received the bachelor's and master's degrees from Nanjing University, Nanjing, China, in 2007 and 2010, respectively, and the PhD degree from the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, Minnesota, in 2015. His research interests include algorithm and hardware co-design and implementation for machine learning and signal processing systems, error-resilient low-cost computing techniques for embedded and IoT systems and machine learning for domain-specific applications. He is the recipient of Global Research Competition Finalist Award in Broadcom Corporation and doctoral dissertation fellowship with the University of Minnesota. He serves as technical committee track chair and technical committee member for several IEEE/ACM conferences. He is the associated editor of the Springer Journal of Signal Processing System.