IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 September 2023, accepted 27 October 2023, date of publication 1 November 2023,
date of current version 7 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3329048

APPLIED RESEARCH

FPGA Hardware Implementation of Efficient Long Short-Term Memory Network Based on Construction Vector Method

TENGFEI LI AND SHENSHEN GU
School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200444, China
Corresponding author: Shenshen Gu (gushenshen@shu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 62276163.

ABSTRACT Long Short-Term Memory (LSTM) and its variants have been widely adopted in many sequential learning tasks, such as speech recognition and machine translation. The low-latency and energy-efficiency requirements of the real-world applications make model compression and hardware acceleration for LSTM an urgent need. In this paper, we first propose a weight parameter generation method based on vector construction that can make the model have a higher compression ratio and produce less precision attenuation. Furthermore, we study in detail the influence of the size of the construction vector on the computational complexity, model compression ratio and accuracy of the construction vector, in order to obtain the optimal size design interval. Moreover, we designed a linear transformation method and a convolution method to reduce the dimension of the input sequence, so that it can be applied to training sets of different dimensions without changing the size of the model construction vector. Finally, we use high-level synthesis (HLS) to deploy the obtained LSTM inference model to the FPGA device, and use the parallel pipeline operation to realize the reuse of resources. Experiments show that, compared with the block circulant matrix method, the proposed designs generated by our framework achieve up to 2 times gains for compression with same accuracy degradation, and it has an acceptable delay. With the same compression ratio, our accuracy decay is 45% of the former.

INDEX TERMS Field-programmable gate array (FPGA), long short-term memory (LSTM), model compression, construct vector method.

I. INTRODUCTION

In recent years, with the development of computer technology and the improvement of the computing power of hardware devices, artificial intelligence technology has achieved unprecedented development [1]. Artificial Neural Networks, especially Deep Neural Networks (DNNs), are widely used in a variety of applications ranging from image classification and recognition to speech recognition and natural language processing [2]. Recurrent Neural Networks (RNNs) are an important class of neural networks that contain loops that pass signals across neurons as data is input [3]. Long Short-Term Memory (LSTM) is one of the most typical RNN models. It has achieved great success in applications in speech recognition [4], video analysis [5], and other fields. However, the improved accuracy of model identification comes at the cost of larger model size and higher computational complexity. Therefore, customized hardware acceleration is becoming more and more important for the application of LSTM. The recently used GPU, FPGA, and ASIC to accelerate inference of LSTM are good examples [6].

ASIC is an application-specific integrated circuit. Unlike programmable GPU and FPGA, ASIC cannot be changed once manufactured. Therefore, it has the problems of high development cost, long cycle and poor flexibility. Although GPUs are widely used in deep learning algorithms, they are not efficient enough to achieve lower power consumption and higher throughput. At present, in addition to improving the algorithm logic and structure itself, hardware acceleration methods are also often used to optimize the forward

The associate editor coordinating the review of this manuscript and approving it for publication was Sotirios Goudos.

reasoning process of neural network models (which are trained) to facilitate the deployment of the models on edge devices. FPGAs are the high-performance, low-power chips with unique advantages in accelerating neural network algorithms [7]. On the one hand, they can achieve high computing performance and high energy efficiency ratio. On the other hand, FPGAs are highly flexible and can be reconfigured. It integrates a large number of digital circuit logic units and memories. Developers can customize the wiring between the logic unit and the memory by burning the configuration file to achieve different arithmetic logic, and this configuration file is not one-time, the circuit logic inside the chip can be modified at any time.

In this paper, we propose a method for generating weight parameters based on the vector construction method. The memory footprint is reduced to a large extent, and the limitation of hardware resources on the FPGA is solved when the network is deployed on the hardware architecture. In order to adapt the matrix multiplication and reduce the computational complexity of the input data on the FPGA device, we perform dimensionality reduction processing on the sequence of the input network: the dimensionality reduction processing steps of the sequence are integrated into the network model, and a one-dimensional convolution kernel is used for convolutional dimensionality reduction. We use high-level synthesis (HLS) to deploy the obtained LSTM inference model on the FPGA device, and use parallel pipeline operations to achieve resource consumption, which makes up for the delay caused by the weight matrix construction.

The rest of this article is structured as follows. In section II, some details of the internal structure of the LSTM layers are introduced, and review the related work to achieve LSTM acceleration on FPGA platforms. Section III introduces a method for generating weight parameters based on the construction vector method. In section IV, we introduce the hardware system design in this work and detail the data path in the hardware design, which are further evaluated and compared with some related works in section V. Section VI is the summary of this paper.

II. BACKGROUNG

In this section, we first introduce some details of the internal structure of the LSTM layers considered, and review the related work to achieve LSTM acceleration on FPGA platforms.

A. LSTM

The long and short-term memory network proposed by Hochreiter and Schmidhuber was a kind of neural network for processing sequence data, which had an important position in the field of natural language processing [8]. With the help of cell states and ''gates'', the memory of some periods can be selectively retained to achieve both long-term and short-term memory.

In this work, we use the representative LSTM network model proposed [9]. An RNN network model based receives a sequence of input vectors $x = (x_1; x_2; x_3; \ldots x_T)$ (each x_t is a vector corresponding to time t). The mathematical expressions of LSTM are shown in (1) $\sim$ (5). As shown in Figure 1, the network structure of LSTM is obtained according to the formula. The dotted box is the internal network structure, and the outside is the input and output state.

$$f_t = \sigma(W_f[x_t; h_{t-1}; c_{t-1}] + b_f) \tag{1}$$
$$i_t = \sigma(W_i[x_t; h_{t-1}; c_{t-1}] + b_i) \tag{2}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot tanh(W_c[x_t; h_{t-1}; c_{t-1}] + b_c) \tag{3}$$
$$o_t = \sigma(W_o[x_t; h_{t-1}; c_t] + b_o) \tag{4}$$
$$h_t = o_t \cdot tanh(c_t) \tag{5}$$

where t represents the current moment, and thus $t-1$ represents the previous moment. c is the cell state, i, o, f represent the states on the input gate, output gate, and forget gate in turn. x represents the input layer state, and h represents the hidden state. $W_y(y = f, i, c)$ is model trainable parameter. $\sigma(\cdot)$ is the sigmoid function, which can be expressed as a formula $\sigma(x) = \frac{1}{1+e^{-x}}$, to map variables between 0 and 1. $\sigma(x) = 1$ means complete retention, $\sigma(x) = 0$ means complete oblivion. $tanh(\cdot)$ is the hyperbolic tangent activation function, which defined as $tanh(\cdot) = \frac{sinhx}{coshx} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

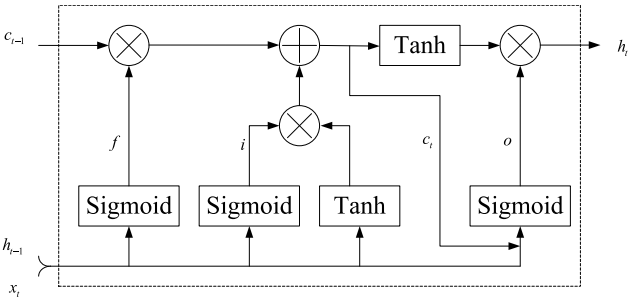

FIGURE 1. Long-short-term memory network structure.

The cell state t carries some information about the network. The core of LSTM is to update c with the help of the ''gate'' operation to control the change of information. The input of LSTM is the input layer state x_t at the current time, the cell state x_{t-1} at the previous time, and the hidden state h_{t-1}. After passing through three gates, the hidden state h_t at the current time and the updated cell state c_t are output. The details of the three gates are as follows:

1) Forgetting gate (equation (1))

Its function is to selectively discard information that is not important to the result, and retain part of the information in the input c_{t-1} at time t.

2) Input gate (equation (2)(3))

If there are discards, there will be new additions. The input gate is to consider the information that needs to be supplemented and determine how many components in x_t remain in c_t. After having the state of the input gate and the

forgetting gate, the cell state c_t can be updated by using (3) to obtain new information. The former term in (3) is the c_{t-1} related component retained in c_t after passing through the forget gate, and the latter term is the x_t related component that exists in c_t after passing through the input gate.

3) Output gate (equation (4)(5))

The purpose of this gate is to use the cell state c_t to determine how many components of the output o_t go into the hidden state h_t. First, the updated c_t is calculated by (4) to obtain the state on the output layer. Then, use (5) to determine how much of the component o_t is retained in the output state h_t.

In one gate or unit, $W_*[x_t; h_{t-1}]$ can be combined in one matrix-vector multiplication by concatenating the matrix and vector as $W_*[x_t^T; h_{t-1}^T]^T$. The four gate/cell matrices can be concatenated and calculated through one matrix-vector multiplication as $W_{ifco}[x_t^T; h_{t-1}^T]^T$ as show in (6).

$$W_{ifco}[x_t^T, y_{t-1}^T]^T = [W_i, W_f, W_c, W_o]^T[x_t^T, y_{t-1}^T]^T \quad (6)$$

B. RELATED WORK

Massively parallel elements are used for RNN acceleration to achieve low latency and high throughput on FPGA [10]. [11] presents different design strategies that balances memory bandwidth and internal storage utilization to optimize performance per power for RNN workloads. The architectures mentioned in both works are fixed because the designer cannot choose any trade-off between resource usage and latency.

In order to improve the prediction speed and energy utilization, a load balance-aware pruning method is proposed in [12]. Using the RNN model based on weight pruning, the model size can be compressed by 20 times, but the network model structure will become irregular after pruning. Then design hardware architectures that work directly on sparse LSTM models. In [13], the irregular network constraints are alleviated by combining block circulant matrices for weight matrix representation in RNN instead of sparse matrices to compress weight matrices, thereby achieving model compression and acceleration. Various LSTM variants are implemented on FPGAs through a proposed synthesis framework called C-LSTM.

III. MODEL DESIGN EXPLORATION

Whether it is using deep learning technology to effectively solve the problem of foreign body intrusion in transmission lines, providing basic guarantee for abnormal detection of transmission lines [26], or using a point cloud segmentation method based on advanced dynamic graph convolutional neural network in coal mining face under harsh environment, aiming to provide key data basis for environment perception in coal mines [27]. It is of great help to promote the green and sustainable production of coal resources. This is undoubtedly the perfect combination of artificial intelligence technology in the industrial field. A big disadvantage of deep neural network (DNN) is that it is too computationally intensive [14]. This largely hinders the commercialization of deep learning-based methods, especially on some edge devices. Most edge devices are not designed for computing-intensive tasks. If they are simply deployed, power consumption and latency will become problems. Under certain processing methods, getting rid of redundancy will not have a great impact on the prediction accuracy [15].

From a hardware implementation perspective, model compression plays a crucial role in saving computation and reducing memory footprint, which means lower latency and higher energy efficiency. Model compression is a software method with low application cost. It is not contradictory to the hardware acceleration method and can be added to each other. In terms of subdivision, model compression can be divided into many methods, such as pruning [16], quantization [17] and so on. Deep neural networks can be divided into two stages: training and inference. The training phase is to learn the parameters in the model according to the data (for the neural network, it is mainly the weights in the network). In the inference phase, new data is fed into the model, and the result is obtained after calculation. Over-parameterization means that we need a large number of parameters in the training phase to capture the tiny information in the data. Once the training is completed to the inference phase, we do not need so many parameters. This assumption supports that we can simplify the model before deployment. The two methods of pruning and quantization in model compression are based on this premise. There are many benefits after model simplification, including but not limited to:

1) The amount of computation is reduced, resulting in less computation time and less power consumption.

2) The memory footprint becomes smaller and can be run on lower-end devices. An additional performance benefit is that DRAM, which is slow and power-hungry, can now be put in SRAM.

3) Smaller packages are beneficial to application release and update.

A. PRUNING

Although the LSTM network model solves the problem of gradient disappearance, it has a large amount of parameters due to the introduction of many gating units. However, due to the robustness of the neural network, even if many parameters are pruned, the accuracy of the neural network can still be obtained by retraining [18]. First, we need to train the model to understand which weights are necessary. Then prune those weights that do not contribute to the prediction accuracy. Finally, we retrain the model.

In [12], a method for sparse pruning is proposed. Due to the discreteness of non-zero weights, it takes a lot of resources to store location information. Storing location information for each non-zero weight, the relative distance from the previous non-zero weight is attached. Processing elements (PEs) are a set of basic computing units with corresponding weights for input speech vectors and are

responsible for the computing tasks of the LSTM. In addition, when performing matrix-vector multiplication, it is ensured that each PE unit processes the same amount of data during pruning, and a FIFO structure is introduced to achieve load balancing. [7] adopts the method of grouping to prune, divides each c weights of the matrix into a group, and only retains the k largest non-zero weights in each group. The location information only needs $\log_2 c$ bits to store the location information, and the number of non-zero elements in each group is the same, which is in line with load balancing. There is also a block circulant matrix pruning method to reduce the amount of parameters of the network [13]. The maximum pruning rate was 16 times. When more than 16 times, the model accuracy will be greatly reduced. Our analysis shows that although the sparse matrix is convenient, if the distance between the two non-zero weights is unstable, a large storage resource is required to store the location information. Although Top-k pruning can only achieve 8 times the compression rate, there are many zero elements in the matrix, which can reduce a lot of computing resources. Ciculant matrix pruning can achieve 16 times the pruning rate.

In this work, we propose a method for generating weight parameters based on the construction vector method. The purpose of pruning the weight parameters is achieved. The specific principle and process are described in detail in the next section.

B. WEIGHT MATRIX REASONING BASED ON CONSTRUCTION VECTOR METHOD

When the algorithm model is implemented in hardware, the design of the model algorithm in the early stage will have a significant impact on the consumption of FPGA hardware computing resources and storage occupancy in the later stage. The biggest impact is the scale of the problem involved (dimension). The scale of the problem will directly affect the scale of the network model parameters. When the scale of the problem increases, in order to reduce the possibility of network underfitting, the first is to increase the trainable parameters of the network. When the network does not overfit under the appropriate circumstances, increasing the network size can improve the performance of the network [18]. Therefore, the two requirements make the network model have better performance while occupying a small amount of hardware resources, which has become a great challenge.

Taking the formula $y_{m\times 1} = W_{m\times n}x_{n\times 1}$ as an example, the parameter reduction method used in [9] is to use the block circulant matrix processing method for the weight matrix $W_{m\times n}$ obtained by training. The idea is to divide the matrix into multiple cyclic blocks. Then each cyclic block only keeps the first row of parameters and store these parameters into BRAM. Finally, the parameters of the remaining lines are filled by the first line loop. Therefore, when the number of loop blocks increases, the number of effective parameters increases, and the structure between the parameters is more complicated. So the calculation operations of transplanting and restoring the parameters to a matrix are also more complicated.

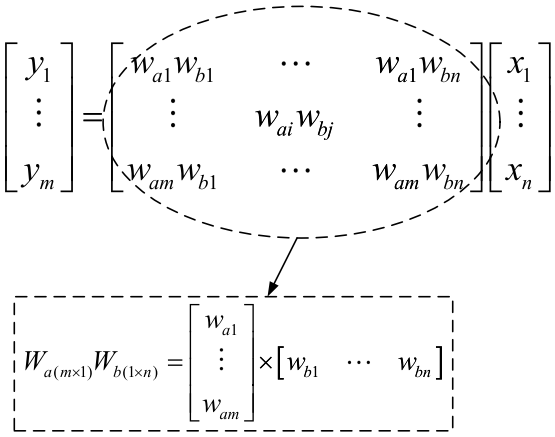

FIGURE 2. Schematic diagram of weight parameter generation.

Aiming at the optimization method of network weight parameters, this paper proposes a weight parameter generation method based on the vector construction method. The construction method is shown in Figure 2. The weight matrix $W_{m\times n}$ can be constructed by the matrix multiplication of the two construction vectors W_a and W_b. In the training phase, each combination parameter $W_{ai}W_{bi}$ of the constructed matrix is directly trained. Therefore, compared with the block cyclic matrix in [9], each element of the construction matrix in this paper is an effective parameter, so the accuracy of the model is greatly improved compared to the former.

After the matrix training is completed, the final construction vectors W_a and W_b are obtained. The parameter structure is simple, the parameter quantity is obviously reduced, and it can be directly stored in the BRAM of the FPGA without any compression processing. When performing inference calculations on FPGA devices, only one step of matrix multiplication can be used to obtain the weight centering. Compared with the method of [9], after the model is trained, this paper omits the operations of matrix partitioning and matrix restoration, porting, and computing stages are all convenient. By the operation of constructing a matrix during hardware calculation, the reduction in the number of parameters directly leads to a significant increase in the number of multipliers consumed. So the resource consumption and time consumption increase. A step-by-step parallel processing method is designed later in this paper to improve the sharing of multiplier resources rate.

C. FACTORS AFFECTING MODEL ACCURACY

The dimension of the input data directly determines the number of weight parameters in the front part of the network. At the same time, the choice of the dimension of the construction vector in the previous section also has a direct impact on the accuracy and parameter quantity of the model. This section discusses the setting method of these two factors.

1) CONSTRUCT VECTOR DESIGN

When constructing the weight matrix $W_{m\times n}$ in the previous section, two one-dimensional vectors of $m*1$ and $1*n$ are used to construct it. Therefore, the number of effective parameters is $m + n$. When the dimension of the weight matrix is small, this method greatly reduces the parameters with better accuracy. However, when the dimension of the weight $W_{m\times n}$ required by the problem increases, the number of effective parameters $m+n$ of the one-dimensional construction vector is much smaller than the $m * n$ required by $W_{m\times n}$. The expressive ability of the model must be greatly reduced. As the scale of the problem increases, the ''dimension'' of the construction vector can be gradually increased. In theory, the higher the ''dimension'' of the construction vector, the better the network performance corresponding to the final constructed weight matrix. The construction vectors are W_a and W_b as shown in Figure 3, and the dimensions are $(m, 2)$ and $(2, n)$ respectively. The construction vector is a matrix in form, and it is called a construction matrix. The result of the matrix multiplication of the two construction matrices W_aW_b is the constructed weight matrix $W_{m\times n}$, and the weight matrix is directly trained during training.

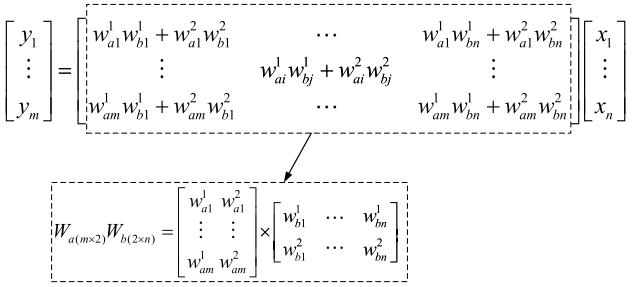

FIGURE 3. Schematic diagram of the construction vectorx (1).

The number of effective parameters of the weight matrix is increased from $m + n$ of the one-dimensional construction vector to $2(m + n)$, which is twice that of the former. The expressive ability of the network is further improved, and the model performance is further improved. When the required dimension of the weight matrix $W_{m\times n}$ is very large, the dimension of the construction matrix can be appropriately increased, as shown in Figure 4. The dimension of the construction matrix is increased to (m, k) and (k, n), and the actual number of parameters is $k(m + n)$, which is much smaller than $m * n$. Therefore, it is necessary to choose an appropriate dimension for the construction vector through experiments, so that the acceptable performance accuracy can still be obtained while greatly reducing the actual number of parameters of the model.

2) RESEARCH ON THE UPPER LIMIT OF CONSTRUCTING VECTOR DIMENSION

The purpose of designing the construction vector is mainly to reduce the number of model weight parameters, and make the performance approach the original model. Therefore, first of all, it is necessary to ensure that the number of

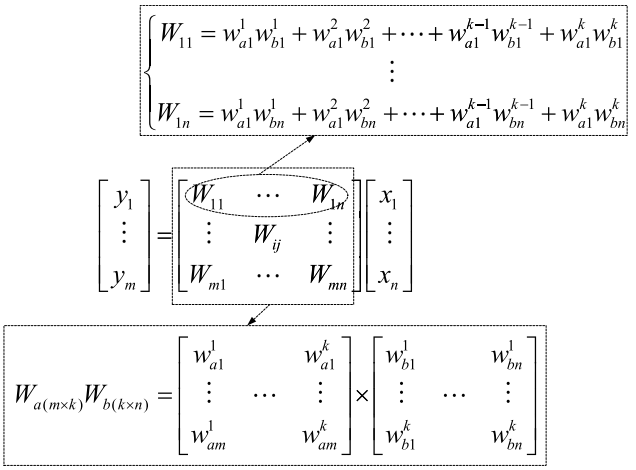

FIGURE 4. Schematic diagram of the construction vector (2).

parameters of the construction vector itself is smaller than the required number of weight matrix parameters. Suppose the input sequence is $(x_1, x_2, x_3, \ldots x_p)^T$, the number of hidden neurons required by the LSTM unit is q, then the weight matrix corresponding to each gate is $W_{p\times q}$, the corresponding number of parameters is $p \times q$, and the construction vectors designed according to this requirement are $W_{a(p\times m)}$ and $W_{b(m\times q)}$, then, design the constructor of the vector must satisfy the following formula:

$$\begin{cases} m \prec min(p, q) \\ pm + mq \prec pq \end{cases} \Longrightarrow \begin{cases} m \prec min(p, q) \\ m \prec \dfrac{pq}{p+q} \end{cases} \tag{7}$$

where min $(min(p, q), \frac{pq}{p+q})$ is the upper limit of the construction vector dimension. The closer the selected value is to the upper limit, the closer the parameters of the construction vector itself are to the parameters of the weight matrix, and the closer the performance of the model is to the original model.

3) DIMENSIONALITY REDUCTION OF INPUT DATA

In the previous section, in the formula $y_{m\times 1} = W_{m\times n}x_{n\times 1}$ of the example, the dimension of the input vector $x_{n\times 1}$ is $(n, 1)$. In order to adapt the matrix multiplication, the dimension of the weight matrix $W_{m\times n}$ should also match the input vector. The dimension of the weight matrix is (m, n), the number of elements is $m * n$. When the dimension of the input vector increases, the parameter quantity of the weight matrix also increases sharply. Therefore, we perform dimensionality reduction processing on the sequence input to the network. This paper adopts two dimensionality reduction methods: 1) linear transformation method; 2) using one-dimensional convolution kernel for convolution dimension reduction.

a: LINEAR TRANSFORMATION METHOD

As shown in Figure 5, the entire network model is divided into two parts. The first part is the dimension transformation layer, which is used to change the dimension of the input sequence $x_{n\times 1}$. This layer consists of a layer of linear layers.

The weight of this linear layer can be specified manually. It can also be learned during training. The weight parameters of the liner layer also contribute to the overall expression ability of the model. The transformed vector $\overline{x}_{p\times 1}$ is finally used as the input of the LSTM network.

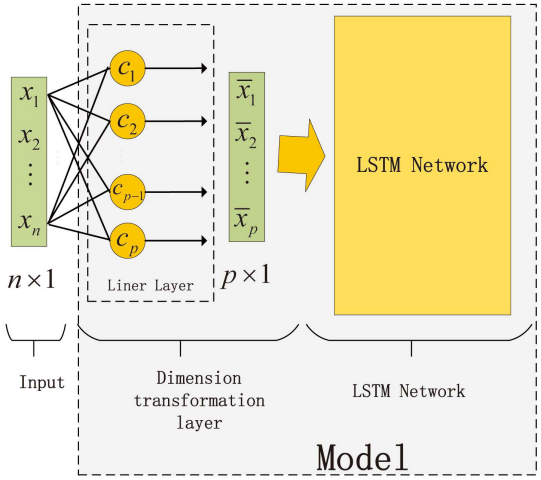

FIGURE 5. Linear transformation method.

b: DIMENSIONALITY REDUCTION USING ONE-DIMENSIONAL CONVOLUTION KERNEL

As shown in Figure 6, the figure takes a $3 * 1$ convolution kernel $(a_1, a_2, a_3)^T$ as an example. The filter starts from the first three elements of the sequence $x_{n\times 1}$, and moves the filter in turn to obtain the corresponding convolution value. The convolution operation is to multiply the corresponding elements to average, the dimension $p = n - k + 1$ of the vector $\overline{x}_{p\times 1}$ obtained after convolution. Where k is the dimension of the convolution kernel, and the parameters of the convolution kernel are set to 1 and -1 cyclic padding, that is $(1, -1, \ldots, 1, -1)^T$. So the $3 * 1$ convolution kernel parameter in this example should be $(1, -1, 1)^T$.

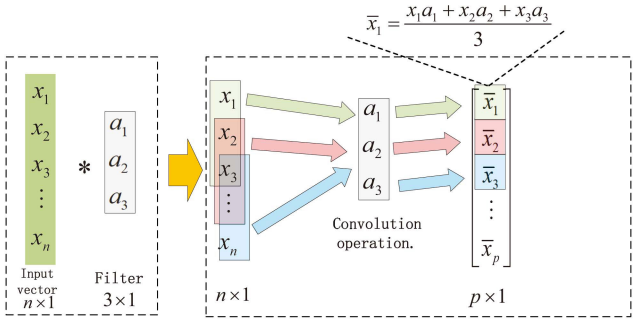

FIGURE 6. Dimensionality reduction using one-dimensional convolution kernel.

D. BI-DIRECTIONAL RECURRENT LSTM NETWORK STRUCTURE

As shown in Figure 7, the bi-directional recurrent LSTM network has strong expressive ability in learning the logical

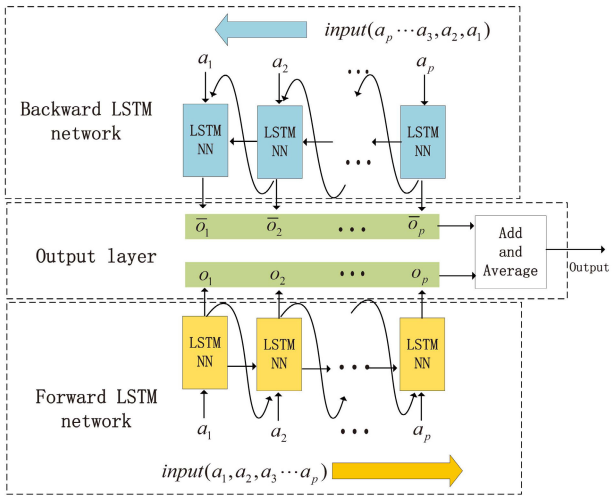

FIGURE 7. Bi-directional recurrent LSTM network structure.

relationship between sequences. The bi-directional cyclic LSTM network architecture is divided into four parts, namely forward LSTM network, backward LSTM network, output layer, and input layer [19]. Among them, the input layer was introduced in the input sequence dimensionality reduction process in the previous section. After processing the input layer, the input sequence is input into the forward LSTM network in sequential order, and the output sequence of the forward network is obtained. The input sequence is reversed and input to the backward LSTM network in turn to obtain the output sequence corresponding to the reverse network. The output sequence is weighted and averaged to obtain the final network output.

E. PERFORMANCE COMPARISON

The previous section discussed that the dimension of the designed construction vector and the dimension of the sequence, for example, have the greatest impact on the number of parameters inside the model. In this section, based on models of different structures and scales, a variety of construction vectors of different dimensions are used to compare and verify the model. The purpose is to find the appropriate dimension of the construction vector, which can greatly reduce the actual number of parameters of the model while still obtaining acceptable performance accuracy. As shown in lines 1-5 of Table 2, the network based on it has three layers of LSTM layers, and the number of hidden units of LSTM units in each layer is 256, denoted as 256-256-256. Based on this model, we designed 7 construction vectors of different sizes, and compared with their phone error rate (PER). The rule for determining the dimension of the construction vector is determined by (7), and the upper limit of the dimension is marked next to the layer size. The same is true for models with a layer size of 1024-1024. We compare each model with the baseline model that does nothing.

The column size of construct vectors in Table 2 represents the size of the construction vector used by each layer

of LSTM. In the baseline model where the layer size is 256-256-256, the weight matrix of the LSTM gate is W, and the dimension of the input vector is n. Then "2-6-4" means that the weight matrix $W_{n\times256}$ of the first layer (input layer) LSTM gate is constructed by constructing vectors $W_{a(n\times2)}$ and $W_{b(2\times256)}$, and the weight matrix $W_{256\times256}$ of the second layer (middle layer) LSTM gate is constructed by constructing vectors $W_{a(256\times6)}$ and $W_{b(6\times256)}$ construction, the weight matrix $W_{256\times256}$ of the third layer (output layer) LSTM gate is constructed by constructing vectors $W_{a(256\times4)}$ and $W_{b(4\times256)}$.

Since the restoration of the construction vector to the weight matrix requires matrix multiplication, when deployed to the FPGA, the restoration of the weight matrix requires multipliers and adders that consume a certain amount of resources. Each gate of the LSTM unit corresponds to the same weight matrix size and calculation. For the three-layer unit, we only select one of the gates in each layer, calculate the number of multiply-add resources consumed when constructing the matrix by constructing the vector, and sum to represent the number of multiply-add resources for the entire model. New model represents the phone error rate of the model corresponding to the construction vector of this size, baseline model is the phone error rate of the model without using the construction vector, and degradation represents the PER degradation of the pruned model compared to the baseline model. Matrix compression ratio represents the compression ratio of the weight parameters of each layer of LSTM relative to the weight parameters of the baseline model under the construction vector of the corresponding size. x represents the dimension of the vector, let in the baseline model, the shape of the weight matrix of the LSTM gate is $W_{p\times q}$. When the construction vector is used to construct $W_{p\times q}$, the corresponding calculation formula of matrix compression ratio r_c is shown in (8), where n represents the dimension of the input vector.

$$r_c = \frac{p \times q}{p \times n + n \times q} \tag{8}$$

[9] uses a block cyclic matrix for pruning, the matrix compression ratios of different shapes are the same when the block size is unchanged. Because the construction vector method is used in this paper, even if the construction vector size is the same, the compression ratios corresponding to matrices of different shapes are different. As can be seen from the first column, the weight matrices of the input layer LSTM and the intermediate layer LSTM gate are different. When the compression ratio is close, the size of the construction vector is quite different, and the range of change is also different.

For the model whose layer size is 256-256-256, it can be seen from Table 2 that with the increase of the size of the construction vector, the parameter compression ratio is smaller, the computational complexity also increases sharply, and the performance of the corresponding model is better. It can be seen from columns 1 and 2 that the size of the construction vector of the input layer has a greater impact on degradation. From columns 4(1) and 4(3), it can be seen that the size of the construction vector of the output layer has less impact on degradation. It can be seen from the columns 4(1) and 4(2) that the different construction vector sizes of the three-layer LSTM layers will also affect the performance. When the small-sized construction vector is placed in the middle layer, the accuracy of the model is lower than that in the output layer. Worse, the reason for this phenomenon is: the small size of the construction vector of the previous layer will cause the layer to fail to capture more input and more information of the layer, and the error will continue to accumulate when the output of the layer passes through the next layer. Therefore, this paper suggests that the size of the construction vector of the front layer should be set larger (that is, the compression ratio should be set smaller), and the latter layer can be set smaller.

TABLE 1. Shape of weight parameters on each gate.

Layer size	Shape of weight parameters on each gate		
	input layer	middle layer	output layer
256-256-256	(n,256)	(256,256)	(256,256)
1024-1024	(n,1024)	-	(1024,1024)

As the size of the construction vector becomes larger, the time and resource consumption of constructing the weight matrix $W_{p\times q}$ is more obvious. As an option, when the application is sensitive to time delay and hardware resources, the construction vector size can be set to 2-5-5. An accuracy loss of no more than 0.31% can be obtained. When the accuracy is required, the recommended setting is 2-6-6, and an accuracy loss of no more than 0.18% can be obtained.

For the models corresponding to 1024-1024, it can be seen from columns 6, 7, and 8 that when the size of the construction vector is small, the change in the size of the construction vector makes the change in degradation more obvious. When construction vector size is large, the changes in degradation will no longer be noticeable. Also as an option, when the application is sensitive to time delay and hardware resources, the size of the construction vector can be set to 3-20, which can obtain the accuracy loss of no more than 0.3% and compression ratio of 1:25.6. If there is a requirement for accuracy, the recommended setting is 3-27, which can obtain an accuracy loss of no more than 0.15% and a compression ratio of 1:18.9.

IV. HARDWARE ARCHITECTURE

In this section, we first present the challenges in hardware design, then present the hardware system design in this work and detail the data path in the hardware design [20], [21].

A. MOTIVATION

In the previous work, the design of the weight matrix based on the construction vector method significantly reduces the number of parameters, thereby greatly reducing the memory footprint. But it also introduces some new challenges,

TABLE 2. Comparation of matrix compression ratio and PER between diffierent models.

ID	Layer size	Size of construct vectors	Matrix compression ratio	Computational complexity (million)	PER(%)		
					New model	Baseline model	Degradation
1	256-256-256 (dimension of construct vectors <33)	1-4-4	1:33.8-1:32-1:32	53	25.21	22.34	2.87
2		2-4-4	1:16.9-1:32-1:32	54	23.82		1.48
3		2-5-5	1:16.9-1:25.6-1:25.6	67	22.65		0.31
4		(1) 2-6-4	1:16.9-1:21-1:32	67	22.55		0.21
		(2) 2-4-6	1:16.9-1:32-1:21	67	22.92		0.58
		(3) 2-6-6	1:16.9-1:21-1:21	81	22.52		0.18
5		2-7-7	1:16.9-1:18-1:18	94	22.47		0.13
6	1024-1024 (dimension of construct vectors <37)	2-16	1:38-1:32	1685	23.88	21.56	2.32
7		3-16	1:12.5-1:32	1689	22.90		1.34
8		3-20	1:12.5-1:25.6	2109	21.83		0.27
9		3-24	1:12.5-1:21.3	2528	21.74		0.18
10		3-27	1:12.5-1:18.9	2843	21.70		0.14
11		3-30	1:12.5-1:17	3158	21.67		0.11

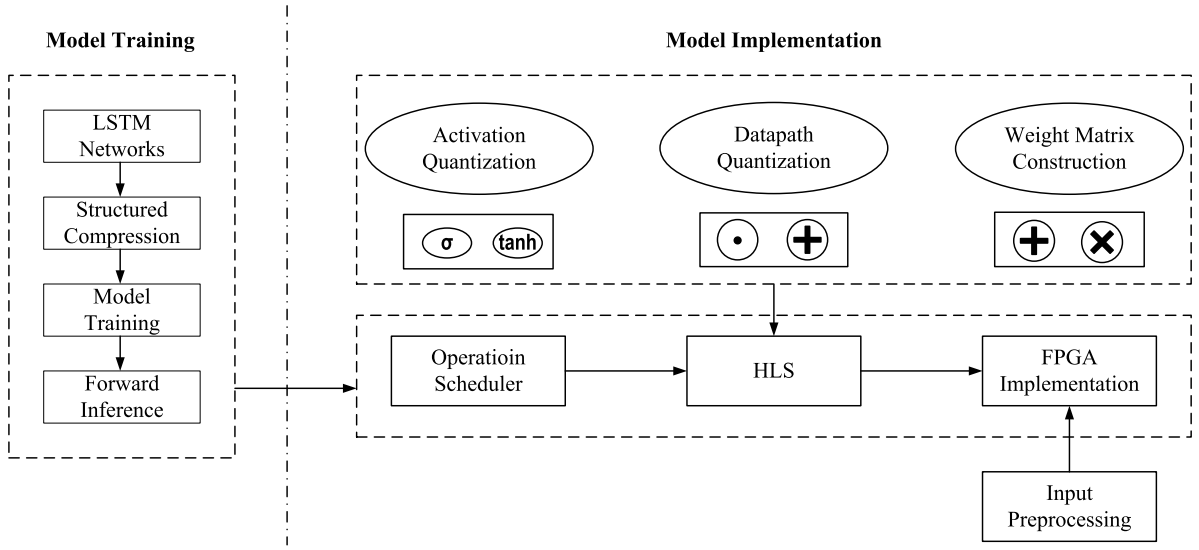

FIGURE 8. Overall system architecture.

the general-purpose processing unit in the hardware architecture cannot effectively meet these challenges [22].

First of all, in the design of hardware computing, the number of multipliers consumed by the weight matrix operation of constructing vectors increases significantly, so the resource consumption and time consumption increase. Here we design a step-by-step parallel processing method to improve the sharing of multiplier resources rate, which will be described in detail in the datapath section below. For the preprocessing of the input data, we can do it in the software part. That is, the delay caused by the construction of the weight matrix is compensated to a certain extent by reducing the dimensionality of the input speech signal with time series.

B. SYNTHESIS OVERVIEW

In order to increase the broad adaptability of the architecture designed in this paper, we propose the synthesis framework shown in the figure below to use matrix structuring based on the construction vector method to assist the model training phase of LSTM. The model can be mapped through HLS to the FPGA device, the inference design on the FPGA device is generated. As shown in Figure 8, the architecture designed in this paper mainly includes two parts: LSTM model training and FPGA device implementation.

1) MODEL TRAINING

The model training phase provides a well-trained inference model and obtains the inference model mapped to the FPGA device through HLS. The sigmoid function, tanh function, weight matrix, vector addition, multiplication, and dot multiplication operations in the above figure are all defined as primitive operators, which are combined into a multi-stage coarse-grained pipeline so that we can analyze the performance. At times, the optimal performance is sought in some resource-constrained situations [23], [24]. Because the hardware computing resources on the FPGA are limited, we will tailor the original model accordingly, otherwise it cannot be deployed on hardware devices.

2) MODEL IMPLEMENTATION

The architecture implementation designed is shown on the right side of Figure 8, which is mainly composed of two parts:

model implementation and comprehensive framework. For the LSTM algorithm model studied in this work, the sigmoid function and the hyperbolic tangent tanh function are the activation functions, and the point multiplication operation is distributed to the operation unit of the FPGA device through HLS. For the weight matrix based on the construction vector method mentioned above, computing units are allocated and parallelized. It should be noted that, according to the comparison, such an architecture design has a relatively high versatility.

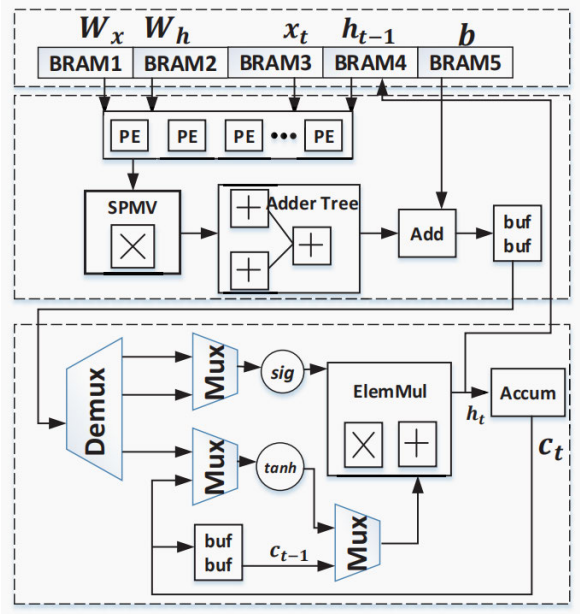

FIGURE 9. Overall system hardware architecture.

C. DATAPATH

The overall hardware architecture of LSTM is shown in Figure 9. It is mainly composed of memory, calculation function, cache and controller. There are 5 BRAMs: BRAM1 stores the weight matrix W_x, BRAM2 stores the weight matrix W_h, BRAM3 stores the variable X_t, BRAM4 stores h_{t-1}, and BRAM5 stores the offset b. The weight matrices W_x and W_h are taken out from BRAM1 and BRAM2 respectively, and the input feature vectors X_t and h_{t-1} are taken out from BRAM3 and BRAM4 respectively. The output is obtained by matrix-vector multiplication and accumulation. The SPMV unit performs P multiplication operations concurrently in one cycle. Here, the selection of the P value determines the utilization efficiency of resources, and more importantly, determines the degree of parallelization of the algorithm and the degree of delay. The output of the SPMV unit is passed to the adder tree component. This component performs addition operations to determine the sum of the outputs of the P operations. This component has a register in addition to the adder, and the current output is added to the previous output. Then, the output is passed to the add component, which adds the bias to the result output by the adder tree component, and saves it in our reserved ping-pong buffer. The function part consists of activation function and dot product operation, obtains the outputs h and c, and writes them into BRAM4 and buff for subsequent operations.

The activation function used in the long short-term memory neural network is a transcendental function. It is very resource-intensive for resource utilization to be implemented on FPGA. To balance resource utilization and algorithm accuracy, we employ quantized piecewise linear functions to estimate them. We use $y = Kx + B$ type linear functions to represent, and only need to store the slope K and intercept B of each linear function. Therefore, the complex calculation of the activation function will turn into a call involving only the slope and intercept, and the use of one multiplication and addition. Practical tests show that the accuracy reduction caused by the piecewise linear function to estimate the activation function is negligible.

The multiplication and addition units of (3) are deployed in the function module, which are respectively realized by the DSP unit. Since the computations of $f_t \odot c_{t-1}$ and $i_t \odot g_t$ are performed at different times, the addition computation is performed by an accumulator within the DSP unit. After the accumulate register is reset, the accumulation result c_t is stored in it. Then, the new result c_t is passed to buff for use in subsequent operations.

V. RESULTS AND DISCUSSION

To evaluate the performance of the proposed scheme, we use the same TIMIT dataset to evaluate the performance of LSTM on FPGA hardware architecture, which is the same as that used in previous E-RNN and C-LSTM work. The TIMIT dataset is a speech dataset constructed for the acquisition of acoustic speech knowledge (model training) and the evaluation of automatic speech recognition systems (ASR) (model testing) [25]. It is widely used in various speech experiments. The TIMIT dataset consists of recordings of 630 speakers speaking 10 sentences from 8 major dialect regions in the United States, for a total of 6300 utterances.

TABLE 3. On-chip resourse of hardware platform.

FPGA	DSP	BRAM	LUT	FF
Xilinx ZYNQ ZCU102	1728	912	274080	548160
ADM-PCIE-7V3	3600	1470	859200	429600

A. EXPERIMENTAL PLATFORM SETTINGS

We evaluate the design proposed in this work on the Xilinx ZYNQ ZCU102 platform. Mounted on a Xilinx ZCU102 board, the ZYNQ UltraScale+ ZU9EG MPSoC integrates a powerful processing system (PS) and programmable logic (PL) in the same device. The on-chip resources of the Xilinx ZYNQ ZCU102 hardware platform are shown in Table 3. The CPU used in this experiment is Intel Core i9-7960X Intel,

TABLE 4. Comparation between block-circulant matrice-based RNN and construct vectors-based RNN.

	Baseline RNN*	E-RNN (Block size:16)	E-RNN (Block size:8)	OUR (*Construct vectors size:3-20)	OUR (*Construct vectors size:3-24)	OUR (*Construct vectors size:3-27)
RNN Cell	LSTM-1024-1024					
Quantization	12bit fixed					
Matrix compression ratio	/*	1:15.9	1:7.9	1:12.5-1:25.6	1:12.5-1:21.3	1:12.5-1:18.9
Platform	ZCU102	ADM-PCIE-7V3		ZCU102		
DSP	1505	2865	3082	1457	1524	1615
BRAM	947	958	1154	610	667	835
LUT	223488	506928	635808	187084	196531	208051
FF	326496	237569	224680	290304	322560	357120
Frequency(MHz)	200					
PER degradation	/*	0.31	0.14	0.27	0.18	0.14
Latency(us)	57.0	12.9	8.3	12.4	15.7	18.8
Power(W)	41	25	24	15	17	21

* Baseline RNN:In order to adapt to the ZCU102 hardware platform we used, baseline RNN is compressed so that it can be deployed on the experimental platform.
*/ In the columns of PER degradation and matrix compression ratio, the indicators of the other models are based on the comparison of the baseline RNN model, so this item is default.
* Construct vectors: represents the size of the construct vector dimension.
* Since each layer of LSTM in [9] uses a block circulant matrix of the same block size, the parameter compression ratio of each layer is the same. In this paper, the construction vector method is used, so in order to obtain a reasonable compression ratio, the size of the construction vector of each layer depends on the shape of the weight matrix. The table specifically indicates the matrix compression ratio of each layer.

and the graphics card is NVIDIA GeForce RTX 3080 Ti. Xilinx Vivado 2018.3 is used as a synthesis backend to synthesize advanced LSTM designs given in C/C++ to FPGA devices. The hardware implementation of LSTM on FPGA runs at 200MHz.

Since we do not have the ADM-PCIE-7V3 FPGA board used in the previous work in this work, we use the ZYNQ ZCU102, which does not affect the experiment if the hardware resources are sufficient. Relevant resources about ADM-PCIE-7V3 are also listed in Table 4, and its experimental results are obtained from experiments in ESE and E-RNN.

B. COMPARISON OF EXPERIMENTAL RESULTS

The latency achieved by this design framework can be measured by multiplying the total number of clock cycles (N_{cc}) by the clock cycles T (5ns) from the Xilinx SDx tool. As shown in the detailed comparison results in the Table 4, we implemented the experiments in the inference phase of the LSTM network on the selected FPGA platform. The bit length was quantized to 12 bits, and the model accuracy was not lost due to quantization. The baseline RNN in the Table 4 is the basic RNN model, and the experimental results of the E-RNN part are from its corresponding paper.

In Table 4, we select the models corresponding to the three construct vectors size with a layer size of 1024 × 1024 to the corresponding models of the two block sizes in [9]. From separately comparing their multiple performance metrics, our model (Construct vectors size: 3-24) has a higher compression ratio than E-RNN(Block size: 8). When the compression ratio is about 1:15, our model (Construct vectors size: 3-27) has higher accuracy than the model of E-RNN (Block size: 16). The reason for this is that E-RNN uses the block loop method to prune the weight matrix obtained by training after the training is completed. After the block is divided, the parameters of each block are all carried out by the first row of the block. It is obtained by cyclic filling, so the uncertainty of the influence of different filling methods on degradation is relatively large. However, the weight matrix we obtained is constructed by constructing a vector, then training is performed. Each parameter of the matrix obtained after construction is determined and can be traces to follow. Therefore, every parameter of constructing the vector actually participates in the training process, so our method has higher accuracy at higher compression ratios, while occupying less BRAM storage. Due to the use of constructing vectors, the hardware device consumes more computing resources when constructing the weight matrix. The parallel processing method designed in this paper improves the sharing rate of resources and keeps the resource consumption rate within an acceptable range.

VI. CONCLUSION

In this paper, we propose the method of parameter generation based on the construction vector method to prune and compression model weights, and introduce the hardware architecture for implementing the LSTM network model. Besides, this paper studies the influence of the size of the construction vector on the computational complexity, model compression ratio and accuracy of the construction vector in detail, so as to obtain the optimal size design interval.

In order to reduce the dimensionality of the input sequence, we propose linear transformation methods and convolution methods. Moreover, we use HLS to deploy the obtained LSTM inference model into the Xilinx ZCU102 FPGA device. Overall, compared with the block circulant matrix method, the proposed designs generated by our framework achieve up to 2 times gains for compression with same accuracy degradation. Our accuracy decay is 45% of the former at the same compression ratio. Finally, for applications with different requirements for time delay, precision attenuation and other indicators, we also give the best choice of construction vector size.

This paper lacks the exploration of industrial application, and the research is limited to the laboratory level. There are three directions for further improvement in future research: 1) reducing data precision; 2) design more fine-grained accelerators; 3) exploring industrial applications.

REFERENCES

[1] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, nos. 1–3, pp. 239–255, Dec. 2010.

[2] R. Sarikaya, G. E. Hinton, and A. Deoras, "Application of deep belief networks for natural language understanding," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 22, no. 4, pp. 778–784, Apr. 2014.

[3] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[5] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using LSTMs," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 843–852.

[6] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–4.

[7] M. Wang, Z. Wang, J. Lu, J. Lin, and Z. Wang, "E-LSTM: An efficient hardware architecture for long short-term memory," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 280–291, Jun. 2019.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[9] Z. Li, C. Ding, S. Wang, W. Wen, Y. Zhuo, C. Liu, Q. Qiu, W. Xu, X. Lin, X. Qian, and Y. Wang, "E-RNN: Design optimization for efficient recurrent neural networks in FPGAs," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 69–80.

[10] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "FPGA-based low-power speech recognition with recurrent neural networks," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2016, pp. 230–235.

[11] A. X. M. Chang and E. Culurciello, "Hardware accelerators for recurrent neural networks on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[12] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 75–84.

[13] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2018, pp. 11–20.

[14] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1135–1143.

[15] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[16] K. Xu, D. Zhang, J. An, L. Liu, L. Liu, and D. Wang, "GenExp: Multi-objective pruning for deep neural network based on genetic algorithm," *Neurocomputing*, vol. 451, pp. 81–94, Sep. 2021.

[17] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[19] R. Zhao, R. Yan, J. Wang, and K. Mao, "Learning to monitor machine health with convolutional bi-directional LSTM networks," *Sensors*, vol. 17, no. 2, pp. 1–18, Jan. 2017.

[20] A. Wuraola and N. Patel, "Resource efficient activation functions for neural network accelerators," *Neurocomputing*, vol. 482, pp. 163–185, Apr. 2022.

[21] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik, "Action recognition in video sequences using deep bi-directional LSTM with CNN features," *IEEE Access*, vol. 6, pp. 1155–1166, 2018.

[22] H. Y. Kim and C. H. Won, "Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models," *Expert Syst. Appl.*, vol. 103, pp. 25–37, Aug. 2018.

[23] Z. Hajduk, "Reconfigurable FPGA implementation of neural networks," *Neurocomputing*, vol. 308, pp. 227–234, Sep. 2018.

[24] E. Bank-Tavakoli, S. A. Ghasemzadeh, M. Kamal, A. Afzali-Kusha, and M. Pedram, "POLAR: A pipelined/overlapped FPGA-based LSTM accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 3, pp. 838–842, Mar. 2020.

[25] Z. Wang, J. Lin, and Z. Wang, "Accelerating recurrent neural networks: A memory-efficient approach," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2763–2775, Oct. 2017.

[26] Y. Wu, S. Zhao, Z. Xing, Z. Wei, Y. Li, and Y. Li, "Detection of foreign objects intrusion into transmission lines using diverse generation model," *IEEE Trans. Power Del.*, vol. 38, no. 5, pp. 3551–3560, Oct. 2023.

[27] Z. Xing, S. Zhao, W. Guo, F. Meng, X. Guo, and S. Wang, "Coal resources under carbon peak: Segmentation of massive laser point clouds for coal mining in underground dusty environments using integrated graph deep learning model," *Energy*, vol. 285, Dec. 2023, Art. no. 128771.

TENGFEI LI received the bachelor's degree in electrical engineering and automation and the master's degree in electrical engineering from the Anhui University of Science and Technology, in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree with Shanghai University. His research interests include neural networks, deep compression models, and especially efficient hardware implementations of deep learning.

SHENSHEN GU received the B.E. degree in computer science from the Shanghai University of Engineering Science, in 2002, the M.E. degree in computer science from Shanghai University, in 2005, and the Ph.D. degree in automation and computer-aided engineering from The Chinese University of Hong Kong, in 2009. Then, he joined the School of Mechatronics Engineering and Automation, Shanghai University, where he is currently a Professor. His research interests include optimization, optimal control, and neural networks.