RTMobile: Beyond Real-Time Mobile Acceleration of RNNs for Speech Recognition

Peiyan Dong[1], Siyue Wang[1], Wei Niu[2], Chengming Zhang[3], Sheng Lin[1], Zhengang Li[1], Yifan Gong[1], Bin Ren[2], Xue Lin[1], and Dingwen Tao[3]
[1]Northeastern University [2]The College of William and Mary [3]The University of Alabama
{dong.pe, wang.siy, lin.sheng, li.zhen, gong.yifa}@husky.neu.edu, xue.lin@northeastern.edu
wniu@email.wm.edu, bren@cs.wm.edu, czhang82@crimson.ua.edu, dingwen.tao@ieee.org

***Abstract*—Recurrent neural networks (RNNs) based automatic speech recognition has nowadays become promising and important on mobile devices such as smart phones. However, previous RNN compression techniques either suffer from hardware performance overhead due to irregularity or significant accuracy loss due to the preserved regularity for hardware friendliness. In this work, we propose *RTMobile* that leverages both a novel block-based pruning approach and compiler optimizations to accelerate RNN inference on mobile devices. Our proposed RTMobile is the first work that can achieve real-time RNN inference on mobile platforms. Experimental results demonstrate that RTMobile can significantly outperform existing RNN hardware acceleration methods in terms of both inference accuracy and time. Compared with prior work on FPGA, RTMobile using Adreno 640 embedded GPU on GRU can improve the energy-efficiency by 40× while maintaining the same inference time.**

***Index Terms*—RNN, pruning, real-time acceleration, mobile**

I. Introduction

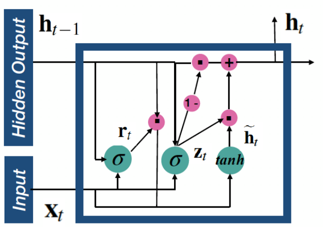
Deep neural network (DNN) has evolved to the state-of-the-art technique due to its high prediction accuracy in many artificial intelligence tasks, such as image recognition and characterization, speech recognition, and recommender system. Among various DNN architectures, recurrent neural networks (RNNs) are widely used for speech recognition tasks because they contain cycles to carry information across neurons when reading inputs. For instance, Gated Recurrent Unit (GRU) [1], the most recent representative type of RNNs, achieves great success in automatic speech recognition. In recent years, executing DNNs on mobile platforms has become more and more popular because many high-end mobile devices are emerging. Several recent studies have proposed techniques to accelerate large-scale DNNs in mobile environment. However, due to fairly high computation complexity and memory consumption when executing RNNs, it is very challenging to deploy RNNs on embedded processors and mobile devices.

DNN model compression provides an effective way to mitigate the computation and memory challenges bringing by DNNs. Many model compression techniques have been studied for recent years. For example, weight pruning can provide a notable reduction ratio in the model size. Early work [2] on non-structured weight pruning eliminates weights at arbitrary location, which leads to the pruned model to be stored in a sparse matrix format, such as compressed sparse column (CSC) format. Non-structured weight pruning, however, hurts processing throughput because the indices in the compressed weight representation result in stalls or complex workloads on highly parallel architectures, such as GPUs and FPGAs. On the other hand, structured weight pruning [3] is more hardware friendly. By exploiting filter pruning [4] and channel pruning [5], the pruned model is more regular in terms of the shape, which can eliminate storing the weight indices. However, structured pruning hurts accuracy more than non-structured pruning. Moreover, state-of-the-art model-compression-based RNN acceleration techniques such as ESE [6] and C-LSTM [7] still suffer from limited inference accuracy and processing throughput, which prevents them to be implemented on mobile devices. Furthermore, existing DNN acceleration frameworks for mobile devices such as TVM [8] *do not even support RNN*. Therefore, in order to achieve the real-time inference for RNNs on mobile devices, it is necessary to develop an end-to-end RNN acceleration framework that can achieve both high inference accuracy and high computational efficiency.

In this paper, we propose a real-time RNN acceleration framework for mobile devices named *RTMobile*. RTMobile is composed of two main components: block-based structured pruning and compiler-assisted performance optimization. Unlike traditional structured pruning methods used on DNNs, our novel block-based structured pruning approach that can provide a finer pruning granularity to maintain high inference accuracy while significantly reducing the RNN model size. We also propose several compiler-based optimization techniques to determine the block size and generate the optimal code on mobiles. Our contributions are summarized as follows.

- We propose a novel RNN acceleration framework for mobile devices, namely, RTMobile. To the best of our knowledge, **RTMobile is the first work that achieves real-time RNN inference on mobile devices**.
- We propose a fine-grained Block-based Structured Pruning algorithm (BSP) for both high inference accuracy and high computational efficiency.
- We develop a series of compiler-based optimization techniques to further accelerate RNN inference on mo-

Corresponding author: Dingwen Tao, Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35487, USA.



$$\mathbf{z}_t = \sigma(\mathbf{W}_{iz}\mathbf{x}_t + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_z),$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_{ir}\mathbf{x}_t + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_r),$$
$$\widetilde{\mathbf{h}}_t = \theta(\mathbf{W}_{ig}\mathbf{x}_t + \mathbf{W}_{hg}(\mathbf{r}_t \odot \mathbf{h}_{t-1})),$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{h}}_t$$

Fig. 1: A Single GRU model.

bile platforms, including matrix reorder, load redundant elimination, and a new compact data format for pruned model storage (called BSPC, i.e., Block-based Structured Pruning Compact format).

- We compare RTMobile with multiple state-of-the-art methods based on a representative RNN (GRU) using a well-known speech recognition dataset. Evaluation results demonstrate that RTMobile is **the first work that can compress the GRU model by over 10x without losing accuracy**. Experiments also illustrate that RTMobile can obtain about **50x energy-efficiency improvement** over prior work with the same inference time.

II. Background and Motivation

In this section, we present some background information about GRU, DNN model compression, and DNN mobile acceleration framework, and discuss our research motivation.

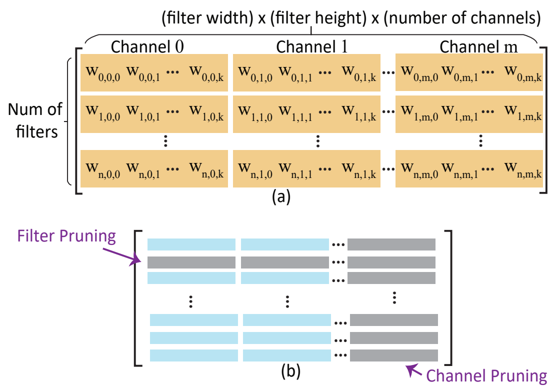
A. Gated Recurrent Unit

The ***Gated Recurrent Unit (GRU)*** is a variation from the LSTM, proposed by Cho et al. [1]. It combines the forget and input gates into a single "update gate". It also merges the cell state and hidden state, and makes some other changes. The resulting GRU model is simpler than standard LSTM models, and has been growing increasingly popular. Fig. 1 shows a single GRU, whose functionality is derived by using the following equations iteratively from $t = 1$ to T, where symbols $\mathbf{z}$, $\mathbf{r}$, $\widetilde{\mathbf{h}}$, $\mathbf{h}$ are respectively the update gate, output gate, cell state, and cell output. As GRU is a more advanced version of RNN than LSTM, we mainly focus on GRU model in this work.

B. DNN Model Compression Techniques

As a representative technique in DNN model compression, DNN weight pruning removes the redundant or less important weights to reduce the storage and computational costs for the inference phase. There exist two mainstreams of weight pruning, i.e., non-structured pruning and structured pruning.

a) Non-structured pruning: Non-structured weight pruning is fine-grained and prunes weights at arbitrary locations. With the successful applications of the powerful ADMM optimization framework, existing research works [2], [9] achieve a very high weight reduction ratio while maintaining promising accuracy. However, non-structured methods lead to sparse and irregular weight matrices, which require indices to be stored in a compressed format. Though saving the storage cost, the decoding of each stored index requires a search over the whole activation vector. Consequently, it suffers from limited acceleration in actual hardware implementation [6].



(filter width) x (filter height) x (number of channels)
Channel 0
Channel 1
Channel m
Num of filters
$W_{0,0,0}$ $W_{0,0,1}$ ⋯ $W_{0,0,k}$
$W_{0,1,0}$ $W_{0,1,1}$ ⋯ $W_{0,1,k}$
$W_{0,m,0}$ $W_{0,m,1}$ ⋯ $W_{0,m,k}$
$W_{1,0,0}$ $W_{1,0,1}$ ⋯ $W_{1,0,k}$
$W_{1,1,0}$ $W_{1,1,1}$ ⋯ $W_{1,1,k}$
$W_{1,m,0}$ $W_{1,m,1}$ ⋯ $W_{1,m,k}$
$W_{n,0,0}$ $W_{n,0,1}$ ⋯ $W_{n,0,k}$
$W_{n,1,0}$ $W_{n,1,1}$ ⋯ $W_{n,1,k}$
$W_{n,m,0}$ $W_{n,m,1}$ ⋯ $W_{n,m,k}$
(a)
Filter Pruning
Channel Pruning
(b)

Fig. 2: (a) To support GEMM computation, the weight tensor representation of a CONV layer is transformed into the weight matrix representation. (b) How different structured weight pruning schemes are implemented on the weight matrix representation.

b) Structured pruning: To overcome the limitations of non-structured pruning, recent works [3]–[5] consider to incorporate regularity in weight pruning with a main focus on convolutional (CONV) layers of DNNs. Previous works mainly focus on two types of structured pruning: filter pruning and channel pruning. Filter pruning, also known as row pruning, removes the entire filter(s), while channel pruning removes the whole channel(s). Figure 2 illustrates an example of transforming convolutional computation into general matrix multiplication (GEMM) by converting weight tensors and feature map tensors to matrices [10]. In general, structured pruning directly reduces the dimension of a weight matrix and preserves a full matrix format, thereby facilitating hardware implementations. On the downside, the coarse-grained nature of structured pruning hurts the accuracy more severely.

C. DNN Acceleration on Mobile Devices

Many efforts target accelerating DNN execution on mobile devices in the past few years, including MCDNN [11], DeepMon [12], TFLite [13], TVM [8], and Alibaba Mobile Neural Network [14]. However, most of them do not deeply exploit model compression techniques as RTMobile does. *In particular, none of the existing frameworks can even support RNN acceleration on mobile devices.*

D. Research Motivation

Based on the survey of recent research works, we conclude the following insights: (i) non-structured pruning has the advantage of very high compression ratio but is typically not compatible with GPU acceleration for inference; and (ii) structured pruning facilitates hardware implementations but is often subjected to accuracy degradation, especially when it is applied to time-based RNNs. To overcome the limitations of current methods, a more flexible and fine-grained pruning policy is needed. This work specifically focuses on RNN models that have not been extensively studied.

III. Related Work

Many existing studies have implemented model compression algorithms for RNN acceleration on FPGAs [6], [7], [15]–[18]. However, the majority of these works focus on constructing new RNN architectures [16] rather than software and hardware co-design framework. Instead, our RTMobile proposes architecture designs in both software and hardware level. In this work, we mainly discuss and compare RTMobile with two most recent and related approaches, i.e., **ESE** [6] and **C-LSTM** [7], which not only address the RNN model compression problem on algorithm/software but also take into account the energy efficiency on hardware (i.e., FPGAs).

A. ESE

ESE proposes an optimized LSTM compression framework on FPGA, which sparsifies the model through parameter pruning [6]. Compared with both CPU- and GPU-based implementations, ESE achieves higher energy efficiency on FPGA. However, the design of ESE has three main limitations: (1) ESE's irregular pruning method used for model compression causes large overhead when performing read/write operations on hardware; (2) the irregularity of weight matrix in ESE results in inefficient implementations of indices that consume extra storage cost, thus the computing power of the FPGA is not fully exerted; and (3) ESE only marginally improves compression rate taking into account indices.

B. C-LSTM

In order to solve the problem caused by irregular pruning, Wang et al. [7] propose an approach (called C-LSTM) to employ a structured compression technique using block-circulant matrices to compress the LSTM model. With regular structure of the block-circulant matrices, C-LSTM can further reduces both computational and storage complexity compared with ESE. However, the coarse-grained nature of structured pruning also cause relatively significant degradation on the model accuracy. Moreover, the advanced ADMM-based neural network pruning method, which can effectively handle both model compression and accuracy, is not supported in the C-LSTM training because it requires the most advanced optimizer in stochastic gradient decent (e.g., Adam optimizer).

C. ADMM

The pruning problem can be formulated as the minimization of $f(W, b) + g(W)$ by following:

$$\begin{aligned} &\underset{\{\mathbf{W}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W_i}, \mathbf{b_i}\}_{\mathbf{i=1}}^{\mathbf{N}}\big) + \mathbf{g}\big(\{\mathbf{W_i}\}_{\mathbf{i=1}}^{\mathbf{N}}\big), \\ &\text{subject to} \quad \mathbf{W}_i \in \mathcal{S}_i,\ i = 1, \ldots, N, \end{aligned} \tag{1}$$

where N is the total number of weight tensor in recurrent neural network, $f(W, b)$ is the loss function, and $g(W)$ is an indicator function that is zero when the constraint S = $\{$ the number of nonzero weights is less than certain threshold $\}$ is satisfied, but $+\infty$ otherwise.

The augmented Lagrangian formation of problem (1) is

$$L_p = \underset{\{\mathbf{W}_i\}\}}{\text{minimize}} \quad f\big(\{\mathbf{W_i}, \mathbf{b_i}\}_{\mathbf{i=1}}^{\mathbf{N}}\big) + \sum_{\mathbf{i=1}}^{\mathbf{N}} \frac{\rho_{\mathbf{i}}}{\mathbf{2}} \|\mathbf{W_i} - \mathbf{Z_i} + \mathbf{U_i}\|_{\mathbf{F}}^{\mathbf{2}}, \tag{2}$$

where ρ_i is a penalty value, $\mathbf{Z}_i$ is pruning mask and $\mathbf{U}_i$ is dual variable.

The ADMM algorithm [19] is to iteratively update the indicated pruning mask and retrain the neural network under this mask, until a good mask and neural network converge. It proceeds by repeating iteration $k = 0, 1, \ldots$ as following:

$$\mathbf{W_i^{k+1}} := \underset{\mathbf{W_i}}{\arg\min} \quad L_{\mathbf{p}}(\{\mathbf{W_i}\}, \{\mathbf{Z_i^k}\}, \{\mathbf{U_i^k}\}), \tag{3}$$

$$\mathbf{Z_i^{k+1}} := \underset{\mathbf{Z_i}}{\arg\min}\ L_{\mathbf{p}}(\{\mathbf{W_i^{k+1}}\}, \{\mathbf{Z_i}\}, \{\mathbf{U_i^k}\}), \tag{4}$$

$$\mathbf{U_i^{k+1}} := \mathbf{U_i^k} + \mathbf{W_i^{k+1}} - \mathbf{Z_i^{k+1}}. \tag{5}$$

The pruning mask can be trained by Algorithm 1.

IV. Proposed RTMobile Framework

In this section, we describe in detail RTMobile, our proposed mobile acceleration framework for RNNs.

A. Block-based Structured Pruning

To better facilitate the compression rate and ensure the structured model architecture for hardware implementations, we propose *Block-based Structured Pruning (BSP)* algorithm. In general, training a BSP compressed model can be separated into two main steps: *Step 1)* row-based column block pruning and *Step 2)* column-based row pruning.

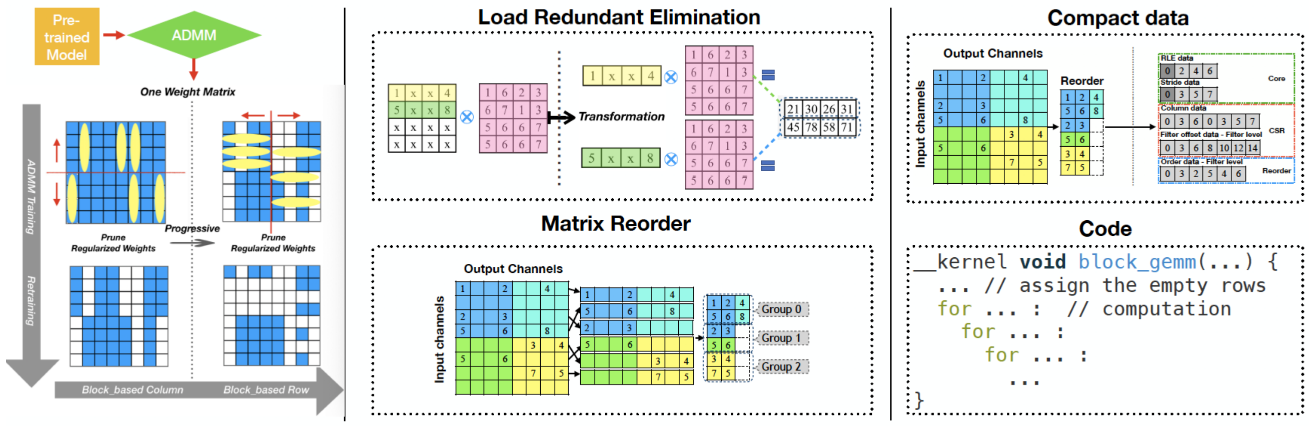
The training process starts with splitting the whole weight matrix $\boldsymbol{W}$ into Num_r rows horizontally. For each row, we divide it into Num_c blocks and then perform the structured pruning using ADMM method (discussed in Section III-C). Then, we perform column-based row pruning over the entire weight matrix $\boldsymbol{W}$ in the step 2. Given the constraint of block number after dividing by Num_c and Num_r, the pruned model can achieve a satisfactory performance overhead on hardware.

The training process continues iteratively until all the blocks are pruned. We identify that by doing so, the training performance is stable, and the whole weight matrix after pruning is decentralized. Our BSP approach is shown in Algorithm 1.

B. Compiler-assisted RNN Acceleration Framework

After the block-based structured pruning, RTMobile relies on a compiler-assisted RNN acceleration framework to achieve efficient RNN inference on mobile devices. This compiler framework consists of three key optimizations that work on each RNN layer (as shown in Figure 3): matrix reorder, load redundancy elimination, and a compact data storage format for pruned RNN matrices, BSPC (i.e., Block-based Structured Pruning Compact format). These optimizations aim to address three key challenges in pruned RNN execution: *thread divergence* and *load imbalance* among threads, *redundant memory access*, and *unnecessary zero storage*.

a) Matrix reorder: The matrix is executed by multiple CPU/GPU threads simultaneously. Without a further reorder, these threads may execute rows with significantly divergent computations, causing severe load imbalance issue that hurts thread-level parallelism. Therefore, RTMobile introduces a matrix reorder optimization to group the rows with the same (or similar) computation patterns together. After this reorder, the rows in each group are assigned to multiple threads to achieve balanced processing.



```
__kernel void block_gemm(...) {
  ... // assign the empty rows
  for ... :  // computation
    for ... :
      for ... :
        ...
}
```

Fig. 3: Systematic overview of RTMobile acceleration framework.

Algorithm 1 Block-based Structure Pruning (BSP)

```
Require:
    $W^l \in R^{m \times n}$ is the weight matrix at $l_{th}$ layer $p_c$ and $p_r$ are the percentage of target column and row sparsities. $Num_c$ and $Num_r$ represent the number of blocks after division. b is the target block size.
    (Experiments on mobile devices showed that when $Num_c \leq 4$ and $Num_r \leq 32$, BSP can guarantees the suitable overhead of mobile device.)
Ensure: A pruning mask $Z^l$ for $W^l$ consists of 0's and 1's representing the weights to be pruned or retrained.
    Step 1. Do Row-based Column Block Pruning:
 1: Divide $W^l$ into $Num_r$ number of rows where the weight matrix are represented by $W^l_{:,0}$, $W^l_{:,1}$,...$W^l_{:,Num_r}$.
 2: for block $i \in [1, Num_r]$ do
 3:     Compute the $L_2$ norm of each column in $W^l_i$;
 4:     Find the threshold $\gamma^*$ such that for columns whose $L_2$ norm less than the threshold should be pruned to achieve the target sparsity level $p_c$;
 5:     for column $k = 0, 1, ...(n-1)$ do
 6:         $Z^l_i[:, k] = 0$ if $L_2 W^l_i[:, k] < \gamma$ :
 7:     end for
 8: end for
    Step 2. Do Column-based Row Block Pruning:
 9: Divide $W^{l_{new}}$ into $Num_c$ number of columns where the weight matrix are represented by $W^{l_{new}}_{0,:}$, $W^{l_{new}}_{1,:}$,... $W^{l_{new}}_{Num_c,:}$.
10: for block $j \in [1, Num_c]$ do
11:     Compute the $L_2$ norm of each row in ${W_j}^{l_{new}}$;
12:     Find the threshold $\zeta^*$ such that for rows whose $L_2$ norm less than the threshold should be pruned to achieve the target sparsity level $p_r$;
13:     for row $r = 0, 1, ...(m-1)$ do
14:         $Z^l_j[r, :] = 0$ if $L_2 {W_j}^{l_{new}}[r, :] < \zeta$ :
15:     end for
16: end for
```

b) Redundant load elimination: Within a group, each thread processes multiple continuous rows, offering us an opportunity to eliminate the redundant memory load operations. This optimization is specifically enabled by our block-based structured pruning, because after such pruning the preserved weights in two neighbor rows may share the same pattern and require the same data in the input feature maps. It is difficult to explore this optimization opportunity for existing unstructured weight pruning due to its irregularity.

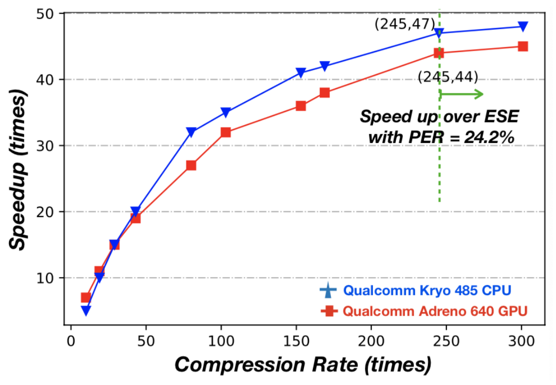


Fig. 4: Speedup using RTMobile with different compression rates on mobile platform.

c) BSPC format: Our proposed block-based structured pruning also guides us to design a more compact data structure than traditional CSR format (called BSPC format) to store RNN weight matrices. This is because within each block the preserved weights only exist in certain rows and columns, enabling to further compact the index array in CSR. The BSPC format also includes the matrix reorder information to match the corresponding input feature map with the weight matrix. The BSPC format significantly reduces the memory footprint and hence alleviate the memory-bound issue in RNN execution.

In addition to the above optimizations, our compiler framework also includes an auto-tuning component to perform an offline search of the best execution configurations like the matrix tiling size, unrolling size, memory placement, etc. In particular, we employ it to find the best block size that results in an optimal combination of accuracy and performance.

V. Experimental Evaluation

In this section, we evaluate RTMobile by comparing it with several state-of-the-art methods. There are three evaluation objectives: 1) comparing RTMobile with other model compression methods and demonstrating that our method outperforms others in both compression rate and accuracy; 2) showing RTMobile has both higher computational efficiency and energy efficiency than a well-known deployment on FPGA (ESE

TABLE I: **Results of Different Model Compression Methods on GRU Using TIMIT Dataset:** PER is *phone error rate*, the lower the better. Baseline PER is for dense (non-pruned) models and pruned PER is for pruned compressed models. PER Degrad. represents for the PER degradation, i.e., $PER_{pruned} - PER_{baseline}$. The rest columns show the column compression rate, row compression rate, the number of preserved parameters, and the overall compression rate, respectively.

Method	PER (%) (baseline - pruned)	PER Degrad.	Column Compress. Rate	Row Compress. Rate	Para. No.	Overall Compress. Rate
ESE [6]	20.40 - 20.70	0.30	–	–	0.37M	8×
C-LSTM [7]	24.15 - 24.57	0.42	–	–	0.41M	8×
C-LSTM [7]	24.15 - 25.48	1.33	–	–	0.20M	16×
BBS [20]	23.50 - 23.75	0.25	–	–	0.41M	8×
Wang [21]	–	0.91	–	–	0.81M	4×
E-RNN [18]	20.02 - 20.20	0.18	–	–	1.20M	8×
BSP (ours)	18.80 (w/o pruning)	0	1	1	9.6M	1×
BSP (ours)	18.80 - 18.80	0	10	1	0.96M	10×
BSP (ours)	18.80 - 19.40	0.60	16	1.25	0.48M	19×
BSP (ours)	18.80 - 19.60	0.80	16	2	0.33M	29×
BSP (ours)	18.80 - 20.60	1.80	16	5	0.22M	43×
BSP (ours)	18.80 - 21.50	2.70	20	8	0.12M	80×
BSP (ours)	18.80 - 23.20	4.40	16	16	0.09M	**103**×
BSP (ours)	18.80 - 24.20	5.40	20	10	0.06M	153×
BSP (ours)	18.80 - 24.20	5.40	20	16	0.04M	**245**×
BSP (ours)	18.80 - 25.50	6.70	20	20	0.03M	**301**×

TABLE II: **Performance and Energy Evaluation on Mobile GPU and CPU:** GOP refers to Giga Operations. GPU/CPU energy efficiency is calculated as $InferenceFrames/(Power \times InferenceTime)$, i.e., the number of frames inferred per unit energy consumption. This table normalizes our method's GPU/CPU energy efficiency by the ESE FPGA implementation's. As our compression rate reaches 245×, our GPU inference time becomes slightly faster than ESE's (82.7us). Our GPU implementation uses 16-bit floating point.

Compression Rate	GOP	GPU Time / Frame (us)	GPU GOP/s	GPU Energy Efficiency (normalized with ESE)	CPU Time / Frame (us)	CPU GOP/s	CPU Energy Efficiency (normalized with ESE)
1× (baseline)	0.5800	3590.12	161.55	0.88	7130.00	81.35	0.25
10×	0.0580	495.26	117.11	6.35	1210.20	47.93	1.48
19×	0.0330	304.11	108.51	10.35	709.33	46.52	2.52
29×	0.0207	233.89	88.29	13.45	464.73	44.43	3.85
43×	0.0143	186.05	76.86	16.91	344.77	41.48	5.19
80×	0.0080	130.00	61.54	24.2	218.01	36.70	8.20
103×	0.0060	109.76	54.66	28.67	202.72	29.59	8.82
153×	0.0039	97.11	40.16	32.4	170.74	22.84	10.47
245×	**0.0028**	**81.64**	**34.30**	**38.54**	**151.28**	**18.51**	**11.82**
301×	0.0020	79.13	25.27	39.76	145.93	13.71	12.25

[6])[1]; and 3) studying the relationship between compression rate and inference execution time.

A. Experiment Setup

Experimental Platform. We conduct our experiments using a Samsung Galaxy S10 with the latest Qualcomm Snapdragon 855 mobile platform, which consists of a Qualcomm Kryo 485 Octa-core CPU and a Qualcomm Adreno 640 GPU.

Model Architecture. We evaluate RTMobile and compare it with the state-of-the-art methods on the popular GRU RNN model, which has been widely used in previous studies [6], [7], [18]. The GRU model contains 2 GRU layers and about 9.6M overall number of parameters.

Evaluation Dataset. We conduct our experiments on the TIMIT dataset [22], which is widely adopted for evaluating automatic speech recognition systems. The TIMIT dataset contains broadband recordings from 630 speakers reading ten phonetically rich sentences in eight major dialects of American English, each reading ten phonetically rich sentences.

B. Evaluation Results and Discussion

Compression Rate and Accuracy. Table I illustrates the results (including phone error rate and number of preserved parameters) of RTMobile with different compression rates and the comparison with other state-of-the-art methods, including ESE [6], C-LSTM [7], BBS [20], Wang [21] and E-RNN [18]. For a fair comparison, we train all models using the same TIMIT dataset [22]. Benefit from the most advanced *PyTorch-Kaldi Speech Recognition Toolkit* [23], the baseline GRU model for our RTMobile can achieve higher recognition accuracy than the other methods before pruning, e.g., our PER is 5.35% lower than C-LSTM's (18.80% v.s. 24.15%).

We observe that our proposed BSP method can guarantee no accuracy degradation when the compression rate is not higher than 10×, which is superior than ESE and C-LSTM from both compression rate and inference accuracy. We also observe that BSP can stably keep a high accuracy compared to the other methods when the compression rate is relatively high. For instance, when the compression rate is **103**×, the BSP pruned model can even outperform the C-LSTM baseline model in terms of both compression rate and accuracy. The C-LSTM baseline model (with 3.25M parameters) has 36× more parameters than our BSP pruned model, but its PER is 0.95% higher than ours (24.15% vs. 23.20%). In addition, we use BSP to further prune the model until the rate of **301**× and observe that our method can well adapt to *ultra-high compression rate* scenario. For example, our model with **245**× compression rate can still maintain the same-level PER as the C-LSTM baseline

[1]We compare RTMobile on mobile with ESE on FPGA because (1) *none of the existing RNN acceleration works supports mobile device*, and (2) ESE provides one of the highest inference accuracy among prior works.

model (24.20% vs. 24.15%) and reduce the parameter number by over 80× (0.04M vs. 3.25M).

Inference Time and Energy Efficiency. Table II presents the evaluation results of RTMobile's inference time, Giga Operations Per Second (GOP/s), and energy efficiency (normalized with ESE method) on mobile GPU and CPU, respectively. The table illustrates that, when the compression rate is higher than **245**×, RTMobile can outperform in energy efficiency by about 40× compared with ESE while maintaining the same inference time (ESE's inference time is 82.7 us) on the mobile GPU (ESE uses a large FPGA platform of 41W power, and thus it is easier to achieve higher energy efficiency than speed). Please note that this is a clear feat, as it is typically perceived that FPGA is more energy-efficient than general-purpose computing devices. This is because of two main reasons. First, comparing to ESE's activation calculation by look-up tables that results in limited parallelization and irregular memory accesses (two key performance factors on FPGA), RTMobile's compiler optimizations significantly improve both the parallelization and memory performance. Second, RTMobile has a much better compression rate (with a negligible accuracy loss), resulting in a more significant computation reduction. Although our compression rates are significant, we must emphasize that the inefficiency in FPGA implementation in ESE (especially activation) plays an equally important, if not more, role. As can be seen from the table, our GPU energy efficiency (frames in unit energy) is almost the same as ESE (which uses compression) even when we do not have any pruning. With increase in the compression rate, the computation pattern becomes I/O and memory bounded, the memory access pattern becomes more irregular, which leads to lower CPU/GPU GOP/s.

Relationship between Compression Rate and Inference Time. Figure 4 further illustrates the relationship between inference time and compression rate. The inference time is in the form of speedups over our own dense CPU/GPU baselines, respectively. The speedup grows as compression rate increases. The speedup becomes stable when the compression rate reaches to a certain range (e.g., compression rate reaches **250**×). When the compression rate is **245**×, our inference time on mobile GPU is as same as ESE's on FPGA.

VI. Conclusion

In this paper, we propose *the first RNN acceleration framework for mobiles*, called RTMobile. We develop a novel block-based pruning algorithm and three compiler optimizations to achieve real-time inference without any accuracy degradation. Experimental results demonstrate that RTMobile significantly outperforms the existing RNN hardware acceleration methods in terms of compression rate, inference accuracy, execution time, and energy efficiency.

Acknowledgment

This work is partly supported by the National Science Foundation CCF-1901378, CNS-1909172, CCF-1919117, OAC-1948447, and OAC-2034169.

References

[1] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[2] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *ECCV*, 2018, pp. 184–199.

[3] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016, pp. 2074–2082.

[4] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *IJCAI*, 2018, pp. 2234–2240.

[5] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*. IEEE, 2017, pp. 1398–1406.

[6] 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 *FPGA*, 2017, pp. 75–84.

[7] 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 *FPGA*. ACM, 2018, pp. 11–20.

[8] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *OSDI*, 2018.

[9] A. Ren, T. Zhang, S. Ye, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: an algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *ASPLOS*, 2019.

[10] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[11] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *MobiSys*. ACM, 2016, pp. 123–136.

[12] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *MobiSys*. ACM, 2017, pp. 82–95.

[13] https://www.tensorflow.org/mobile/tflite/.

[14] https://github.com/alibaba/MNN/.

[15] S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu, "Fpga acceleration of recurrent neural network based language model," in *FCCM*. IEEE, 2015, pp. 111–118.

[16] 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 *FPL*. IEEE, 2016, pp. 1–4.

[17] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," in *ASP-DAC*. IEEE, 2017, pp. 629–634.

[18] Z. Li, C. Ding, S. Wang, W. Wen, Y. Zhuo, X. Lin, X. Qian, and Y. Wang, "E-rnn: design optimization for efficient recurrent neural networks in fpgas," in *HPCA*. IEEE, 2019.

[19] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[20] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu, and L. Zhang, "Efficient and effective sparse lstm on fpga with bank-balanced sparsity," in *FPGA*. ACM, 2019, pp. 63–72.

[21] S. Wang, P. Lin, R. Hu, H. Wang, J. He, Q. Huang, and S. Chang, "Acceleration of lstm with structured pruning method on fpga," *IEEE Access*, vol. 7, pp. 62 930–62 937, 2019.

[22] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, "Timit acoustic-phonetic continuous speech corpus," *Linguistic data consortium*, vol. 10, no. 5, p. 0, 1993.

[23] M. Ravanelli, T. Parcollet, and Y. Bengio, "The pytorch-kaldi speech recognition toolkit," in *In Proc. of ICASSP*, 2019.