Iterative Compression of End-to-End ASR Model using AutoML

Abhinav Mehrotra[1], Łukasz Dudziak[1], Jinsu Yeo[2], Young-yoon Lee[2], Ravichander Vipperla[1],
Mohamed S. Abdelfattah[1], Sourav Bhattacharya[1], Samin Ishtiaq[1], Alberto Gil C. P. Ramos[1],
SangJeong Lee[2], Daehyun Kim[2], Nicholas D. Lane[1,3]

[1] Samsung AI Center, Cambridge, UK
[2] On-device Lab, Samsung Research, Seoul, South Korea
[3] University of Cambridge, UK

Abstract

Increasing demand for on-device Automatic Speech Recognition (ASR) systems has resulted in renewed interests in developing automatic model compression techniques. Past research have shown that AutoML-based Low Rank Factorization (LRF) technique, when applied to an end-to-end Encoder-Attention-Decoder style ASR model, can achieve a speedup of up to $3.7\times$, outperforming laborious manual rank-selection approaches. However, we show that current AutoML-based search techniques only work up to a certain compression level, beyond which they fail to produce compressed models with acceptable word error rates (WER). In this work, we propose an iterative AutoML-based LRF approach that achieves over $5\times$ compression without degrading the WER, thereby advancing the state-of-the-art in ASR compression.

Index Terms: ASR Compression, AutoML, Reinforcement Learning

1. Introduction

Rapid technological improvements in deep learning-based acoustic modeling, language modeling, and noise-resilience techniques resulted in a drastic drop in the WER of modern ASR systems. Evidently, ASR is serving as the backbone in audio-based input modality on a variety of devices including mobile phones, smart speakers, and IoT appliances. Due to existing data security and privacy concerns in cloud-based ASR systems, a clear shift in preference towards on-device deployment of the state-of-the-art ASR models is emerging. Mobile and IoT devices, however, suffer from a limited resource budget and require efficient deployment of ASR models with significantly lower memory, compute and power demands.

Popular techniques in reducing resource demands of well-trained and parameter-heavy models include LRF, pruning, and reduced-precision representations [1, 2]. In our previous work, we have shown that automated Reinforcement Learning (RL)-based search can be applied to identify low ranks in LSTM weight-matrices, which allowed for a $1.23\times$ relative speedup gain over a manual search procedure [3]. However, we observe that the conventional AutoML search fails to find ranks that manifest higher compression ratio (e.g., $\geq 3.7\times$), without degrading the WER. This is due to inability of RL to differentiate between better and worse choices since most of the visited points result in similar rewards. Alternatively, iterative approaches have been successfully used to manually compress image recognition models, often delivering better results than their one-shot counterparts [4, 5, 6, 7].
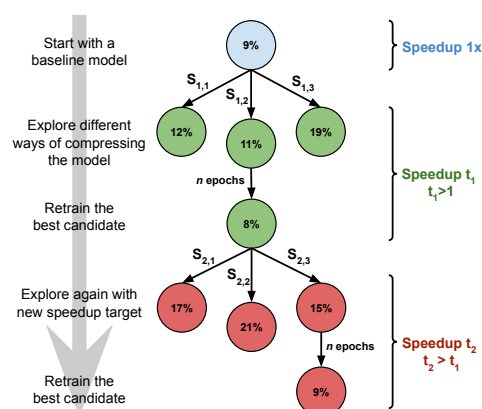

Figure 1: *A high-level overview of the proposed method. Percentages in nodes are example word error rates for different models. $s_{n,i}$ values represent different schemes of compressing a model to achieve target speedup t_n.*

In this work, we propose a unified approach by combining iterative compression with AutoML-based rank searching to push the boundaries of ASR model compression. We present an RL-based iterative search that performs incremental compression by following a sequence of increasing speedup targets (*trajectory*) to reach the desired final speedup, while maintaining the WER. The basic idea behind our method is illustrated in Figure 1. The main goals are: (i) obtain smaller models than the one achievable by current AutoML-based one-shot compression, while maintaining low WER; and (ii) understand if there is a fundamental difference between iterative and one-shot LRF-based compression techniques.

We present an extensive set of experiments by considering a state-of-the-art E2E ASR model and show that the proposed iterative search can achieve at least $5\times$ compression ratio, while producing a 7% relative gain in WER. Compared to the current state-of-the-art one-shot AutoML compression the results show $1.35\times$ relative gain in speedup without degrading the model's accuracy. Furthermore, we investigate the impact of different trajectories on achieving a $5\times$ compression ratio with our iterative compression approach. Our results suggest that taking incremental compression steps, while performing iterative search, helps in discovering better compression ranks. Finally, we show that once the optimal ranks are found, they can be applied to the baseline model in an one-shot manner to obtain a compressed model without any loss in WER. However, the question, *how to find those ranks without intermediate retraining*, remains an open problem.

Algorithm 1: Iterative compression with AutoML.

Input: (i) Trajectory $[t_i]_{i=1}^K$, where $t_i \in \mathbb{R}^+$, (ii) parameter matrices $\mathbb{M} = \{M_i\}_{i=1}^L$.

Output: Compressed and retrained model

1 **for** $i \leftarrow 1$ **to** K **do**
2     $S \leftarrow$ construct_search_space$(\mathbb{M}, t_i)$
    $s \leftarrow$ automatic_rank_search$(S, \mathbb{M}, t_i)$ **for** $j \leftarrow 1$ **to** L
    do
3        $U, \Sigma, V^T \leftarrow LRF(M_j)$
4        $\hat{M}_j \leftarrow U_{[:s_j]} \Sigma_{s_j} V_{[s_j:]}^T$
5        $M_j \leftarrow \hat{M}_j$
6   **end**
7   **if** $i == K$ **then**
8        $s^* \leftarrow s$
9   **end**
10   Retrain the model using $\mathbb{M}$ for initialization
11 **end**
12 **for** $j \leftarrow 1$ **to** L **do**
13     $U, \Sigma, V^T \leftarrow LRF(M_j)$
14     $M_j \leftarrow (U_{[:s_j^*]}, \Sigma_{s_j^*} V_{[s_j^*:]}^T)$
15 **end**
16 Retrain the model using $\mathbb{M}$ for initialization

---

2. AutoML-based Iterative Compression

Automatic Compression Scheme. Given a layer parameter matrix M, we use LRF as our main compression method [8, 9, 10] and approximate M using its truncated singular value decomposition (SVD), i.e., $M = U\Sigma V^T \approx U_{[:k]} \Sigma_k V_{[k:]}^T = \hat{M}$. Given a ASR model, we compress all parameter matrices independently by choosing different ranks and call the set of selected ranks across all layers a *compression scheme* (s). We further define *speedup* (or compression ratio) of the compressed model as:

$$\text{speedup} = \frac{\text{FLOPS of baseline}}{\text{FLOPS of compressed model}}$$

We automatize the process of searching for a good compression scheme by using an algorithm based on REINFORCE [11] with two reward functions, $\mathcal{R}$ and $\mathcal{R}_v$, used to optimize for lower WER and smaller model size simultaneously. Further details can be found in [3].

Iterative Compression. Iterative model compression is performed by repeatedly applying AutoML-based rank-search for a *trajectory*, followed by retraining the model with the identified best compression scheme. A trajectory ($t = [t_1, t_2, ...t_K]$) is defined as a sequence of increasing intermediate speedup goals to reach the final target speedup. For instance, example trajectories for an overall $3\times$ speedup goal can be $[2\times, 3\times]$ and $[1.5\times, 2\times, 2.5\times, 3\times]$, resulting in 2 and 4 iterations of AutoML search respectively. Note that *one-shot AutoML-based search* is a special case of the iterative searching, where the trajectory consists of a single step, e.g., $[3\times]$.

During each step within a trajectory, the best compression scheme s found by the automated search is used to approximate individual parameter matrices of the model, i.e., $\hat{M}_i = U_{[:k]} \Sigma_k V_{[k:]}^T$, where i is the index over all layers. These approximations are then used as the initialization weights for layers during model retraining for a fixed number of epochs. To leverage computational and memory gains, we only store factorized components U_i, V_i^T for the final model identified at the end of a trajectory search. However, we keep using the full size matrices $\hat{M}_i$ during intermediate iterations to give the training algorithm
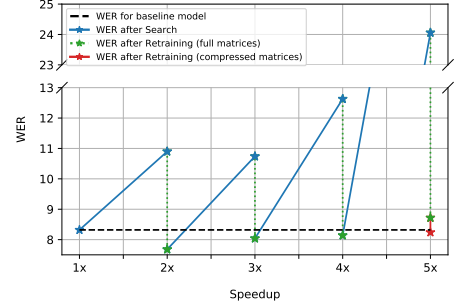

Figure 2: *WER for MoChA model at each step of iterative compression approach. The solid lines indicate the change in WER during each compression step, and the dotted lines show the gain in performance after retraining. The red line indicate the performance gain after retraining target compressed model.*

more capacity in recovering from degraded WER due to low-rank approximations. The overall iterative search procedure is described in Algorithm 1.

At the first search step of a trajectory, we identify good range of ranks (for a specific layer) based on the cumulative eigenvalue energy range, for obtaining the required speedup. For the subsequent steps we use the same energy range when the Δt_i (i.e., the speedup increase from the previous step) is similar to Δt_{i-1}. However, we lower (or increase) the range to be more aggressive (or moderate) for high (low) values of $\Delta t_i / \Delta t_{i-1}$. For example, energy range considered during the search for $2\times$ speedup, is reused when we perform the search for $3\times$ speedup in the next step. However, we lower the energy range if we perform the search for $4\times$ speedup after $2\times$ compression.

3. Evaluation

Model Architecture. We use an end-to-end encoder-attention-decoder ASR model [12, 13, 14] with unidirectional LSTMs (unit size 1024) and monotonic chunk-wise attention (MoChA) [15] as our baseline. We identified 11 layers consisting of total 17 matrices which could be compressed and considered 5 different compression levels for each of them, which constitutes a search space of $5^{17} \approx 762 \times 10^9$ possible combinations.

Model Training. The model is trained on LibriSpeech dataset [16] using MFCC features as input and operates on a vocabulary of 10,000 byte-pair-encoded sub-word units [17, 18].

Baseline Compression Approach. We use the state-of-the-art AutoML-based one-shot compression [3] as the baseline for evaluating the performance of our approach.

3.1. Defining Trajectories

To perform iterative compression, we need to define a trajectory that informs the iterative search algorithm about the intermediate speedup through which it should traverse. Since a trajectory is based on the target speedup, we performed initial experiments to explore the maximum possible compression. Based on our empirical results, we found that some layers of the model used in the experiment reach the maximum compression with the target speedup of 5x (discussed in Section 3.2). Therefore, to evaluate our iterative compression approach we defined a series of trajectories to obtain 5x speedup, which is $1.35\times$ improvement over the state-of-the-art AutoML-based one-shot approach [3]. Conclusively, we define the following trajectories: (i) Base-2x-5x (ii) Base-2x-3x-5x (iii) Base-2x-3x-4x-5x

Part	Layer	Search	Orig. dimensions	Orig. FLOPS [M]	Rank	New FLOPS [M]	Speedup
Encoder	LSTM 0	✓	40×4096, 1024×4096	0.16, 4.29	Full, 105	0.16, 0.55	1x, 7.8x
	LSTM 1	✓	1024×4096, 1024×4096	4.29, 4.29	72, 85	0.37, 0.44	11.4x, 9.6x
	LSTM 2	✓	1024×4096, 1024×4096	4.29, 4.29	82, 63	0.42, 0.32	10.0x, 13.0x
	LSTM 3	✓	1024×4096, 1024×4096	4.29, 4.29	102, 81	0.52, 0.41	8.0x, 10.1x
	LSTM 4	✓	1024×4096, 1024×4096	4.29, 4.29	93, 103	0.48, 0.53	8.8x, 8.0x
	LSTM 5	✓	1024×4096, 1024×4096	4.29, 4.29	80, 82	0.41, 0.42	10.2x, 10.0x
Attention	enc. ctx	✓	1024×1024	1.05	16	0.03	32.0x
	chunk enc. ctx	✓	1024×1024	1.05	24	0.05	21.3x
	dec. trans.	✓	1000×1024	1.02	15	0.03	33.7x
	chunk dec. trans.	✓	1000×1024	1.02	17	0.03	29.8x
Decoder	kernel	✓	2645×4000	10.58	130	0.86	12.2x
Output	readout	✗	2645×1000	2.64	N/A	2.64	1x
	output prob.	✗	500×10025	5.01	N/A	5.01	1x
	embedding	✗	10025×621	0	N/A	0	1x
			Overall:	**63.96**	**N/A**	**12.73**	**5.0x**

Table 1: *Break-down of the Base–2–3–4–5x compression scheme. LSTM layers contain two matrices (for projecting input and recurrent state) hence two values are reported. Layers which do not contribute much to the running-time were omitted. However, we included the embedding layer due to its considerable size (even though it has marginal impact on run-time).*
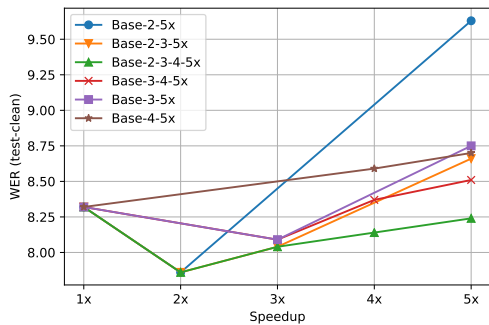

Figure 3: *WER for MoChA model for $5\times$ compression with different granularities of steps.*

(iv) Base-3x-5x (v) Base-3x-4x-5x (vi) Base-4x-5x. Note that we used the increment of speedup during each step of a trajectory as a factor of 1x, but this could be any real number as per the requirements.

3.2. Iterative Model Compression

To perform an iterative search for a trajectory, energy range are explored between 0.3 to 0.99, such that they could find good ranks to compress the model satisfying the speedup threshold of the first step of the trajectory. These ranks are used to compress the model, which is then retrained[1] to recover any drop in WER. This model is then used as a base to perform the compression search for the next step of the trajectory. The processes is continued with the compression-retraining phases repeated until all steps of the trajectory are processed.

Following this methodology, we performed the iterative search for all trajectories defined in Section 3.1. The results show that *Base-2×-3×-4×-5×* trajectory was able to find the best ranks for 5.0x compression along with a 7% relative gain in accuracy (i.e., reduced WER). Thus, our approach outperforms the current AutoML-based one-shot compression with $1.35\times$ gain in speedup.

The entire process of the best trajectory (i.e., Base-2×-3×-4×-5×) is depicted in Figure 2, with solid and dotted lines representing search/compression and retraining steps respectively, and Table 1 shows a breakdown of the obtained compression

---

[1]Through the initial experiments we found that around 50 epochs were required to recover the model performance after applying SVD while increasing speedup by 1x.
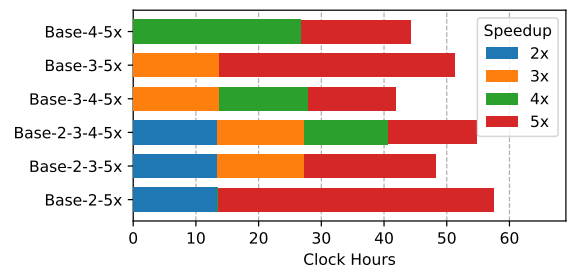

Figure 4: *Computation time for 5x compression of MoChA model with different granularity of step. Each color indicates the specific speedup of a step and the length represents the clock hours consumed during search for that step (excluding training time).*

scheme. We observe that at 5x compression our model becomes mainly limited by the size of the layers which were not subject to the search, with the other parts being compressed well beyond 5x (i.e., around 13x) to balance them out. Interestingly, layers related to attention turned out to be the most compressible with around 28x speedup.

3.3. Comparison of Trajectories

The trajectories used in the experiment have different granularity of steps, which could impact the performance of the end model and computation time for the overall search. Therefore, we analyse the model accuracy and computation time for each step of all trajectories. As shown in Figure 3, we observe that a low granularity of steps yields models with better size and WER. This could be due to smaller distortion of weights in each step, compared to weights distortion in higher granularity steps. At the same time, the overall compute time required to perform the compression process with fine grained steps does not increase significantly even though the number of steps performed is higher. As shown in Figure 4, this is due to the fact that even though the trajectories with bigger step sizes have fewer steps, searching for a bigger step size takes significantly longer compute time. Note that for a fair comparison between trajectories of different length, we keep the same overall retraining budget (i.e., 250 epochs) for all experiments, which is allocated proportionally to the granularity of steps. Therefore, the overall training time would be same for all trajectories.
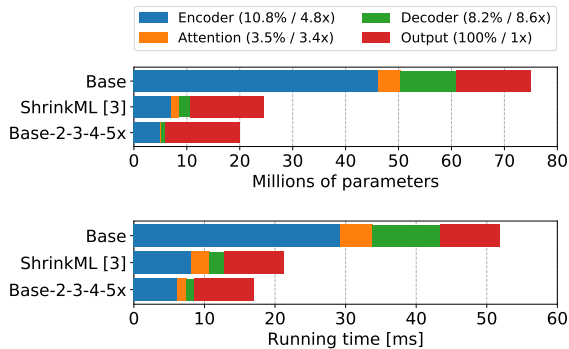
Figure 5: *Model compression (top) and runtime measurements on Exynos 9810 processor (bottom). Values reported in the legend are compression ratio and observed speedup of each part.*

3.4. On-Device Measurements

We compare the running-time of our 5x compressed model (with Base-2×-3×-4×-5× trajectory) against the baseline model and 3.7x compressed model obtained by one-shot approach [3]. The measurements were performed on a mobile phone equipped with an Exynos 9 Octa 9810 chipset to check in practice what is the achieved speedup of our model. The models were run using a standard TFLite runtime. As shown in Fig. 5, improvements in running-time (bottom) remain proportional to what we would expect from theoretical estimates (top). However, it is quite visible that the speedup achieved in practice does not match exactly the theoretical one. We attribute that to the fact that most of the layers which were compressed became very small and therefore more sensitive to any kind of overhead present in the TFLite runtime. However, we did not study the precise reason behind this mismatch and instead leave it for future work.

Overall, the running-time of the compressed model is 17ms, which indicates for the $5\times$ theoretical speedup we were able to observe only $3.0\times$ practical reduction over the baseline model. Compared to the current one-shot AutoML-based model [3], we achieved $1.2\times$ practical reduction in running-time for the $1.35\times$ theoretical speedup.

3.5. Why does Iterative outperform single-shot?

To better understand differences between iterative and one-shot compression, we investigated if the one-shot compression approach fails to find the optimal ranks or the iterative compression along with retraining is a must to obtain higher compression. To examine this, we explored two other strategies for achieving 5x speedup:

(i) Base–5x: search for 5x speedup using one-shot AutoML approach – i.e., analogical to [3];

(ii) Base–Iterative Ranks: apply the best set of ranks found with our iterative approach directly on the base model.

The idea behind (ii) was to answer the question: could one-shot approach match performance of the iterative compression if it had access to a better searching method? After compressing these models we retrained them with the same budget of 250 epochs. We used two retraining strategies:

(i) Compressed: in this approach we keep the model in compressed form (i.e., keeping weight matrices as U, V) to constrain the training to maintain the compressed size;

Approach	WER (test-clean)
Baseline	8.32
Iterative Approach	8.24
Base–Iterative Ranks (Compressed)	9.17
Base–Iterative Ranks (Cyclic)	**8.19**
Base–5x (Compressed)	10.25
Base–5x (Cyclic)	8.92

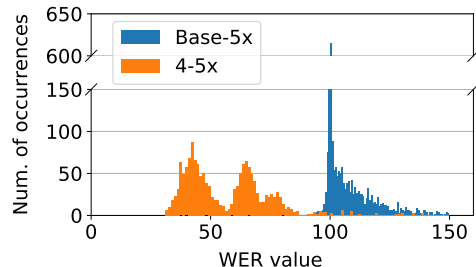Table 2: *Evaluation of different compression strategies.*

Figure 6: *Iterative vs one-shot AutoML search for $5\times$ speedup.*

(ii) Cyclic: in this approach we tried to mimic the training scheme from iterative compression. At every 50 epochs, we compress the model but keep the full weight matrices (i.e., as $\hat{M}$) to allow training to recover from compression. However, in the final step we train the model in the compressed form to constrain the compressed size. Moreover, after every 50 epochs the learning rate is reset to its initial value.

As can be seen in Table 2, the results show that our iterative approach is significantly better than *Base-5×* approach, but when iterative ranks are applied to the baseline model (i.e., *Base–Iterative Ranks* approach) we could obtain a compressed model with almost similar performance. This indicates that there exists optimal compression schemes for large speedups that can be used to directly compress the baseline model without losing any accuracy. However, as shown in Figure 6, the distribution of WER values for one-shot search has a strong peak, which indicates that most of the points visited by RL yield in similar rewards. This limits the algorithm to learn well as it can hardly differentiate between better and worse choices as they all become "equally bad".

Furthermore, the analysis of the two *Base–Iterative Ranks* strategies tells us that aggressive compression schemes, even if known to be good, can not be simply applied to the baseline model in a one-shot manner. Instead, a different training scheme (e.g., cyclic learning rate scheduling) has to be used to fully utilize their potential, which suggests a need for more research into training techniques for small models.

4. Conclusions

In this paper we presented an AutoML-based iterative compression approach that overcomes the limitations in existing one-shot compression approaches and is found to produce highly-compressed models without any loss in WER. We applied the search on MoChA-based ASR model and demonstrated that the iterative approach could find a $5\times$ compressed model with 7% relative gain in WER. We further showed that a baseline model can be compressed directly by using the ranks identified in the iterative search procedure. This highlights that during an one-shot search, AutoML fails to explore good ranks, since most of the compression schemes encountered by the search could not be differentiate between better and worse choices, thereby undermining the RL agent's learning ability.

5. References

[1] R. Pang, T. Sainath, R. Prabhavalkar, S. Gupta, Y. Wu, S. Zhang, and C.-C. Chiu, "Compression of End-to-End Models," in *Interspeech'18*. ISCA, 2018.

[2] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[3] Ł. Dudziak, M. S. Abdelfattah, R. Vipperla, S. Laskaridis, and N. D. Lane, "ShrinkML: End-to-End ASR Model Compression Using Reinforcement Learning," in *Interspeech'19*, 2019.

[4] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *International Conference on Computer Vision*, 2017. [Online]. Available: http://arxiv.org/abs/1708.06519

[5] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations*, 2017. [Online]. Available: http://arxiv.org/abs/1611.06440

[6] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding small, trainable neural networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: http://arxiv.org/abs/1803.03635

[7] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *International Conference on Learning Representations*, 2019. [Online]. Available: http://arxiv.org/abs/1810.05331

[8] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *INTERSPEECH*, 2013.

[9] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6655–6659.

[10] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '16. New York, NY, USA: ACM, 2016, pp. 176–189. [Online]. Available: http://doi.acm.org/10.1145/2994551.2994564

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representation (ICLR)*, 2017.

[12] C. Kim, S. Kim, K. Kim, M. Kumar, J. Kim, K. Lee, C. Han, A. Garg, E. Kim, M. Shin, S. Singh, L. Heck, and D. Gowda, "End-to-end training of a large vocabulary end-to-end speech recognition system," in *ASRU*, 2019.

[13] K. Kim*, K. Lee*, D. Gowda, J. Park, S. Kim, E. S. Kim, Y.-Y. Lee, J. Yeo, D. Kim, S. Jung, J. Lee, M. Han, and C. Kim, "Attention based on-device streaming speech recognition with large speech corpus," in *ASRU*, 2019.

[14] A. Zeyer, K. Irie, R. Schlüter, and H. Ney, "Improved training of end-to-end attention models for speech recognition," in *Interspeech*, 2018, pp. 7–11.

[15] C.-C. Chiu and C. Raffel, "Monotonic chunkwise attention," in *International Conference on Learning Representations*, 2018.

[16] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *ICASSP*, 2015, pp. 5206–5210.

[17] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *ACL*, Berlin, Germany, Aug. 2016, pp. 1715–1725.

[18] D. Gowda, A. Garg, K. Kim, M. Kumar, and C. Kim, "Multi-task multi-resolution char-to-BPE cross-attention decoder for end-to-end speech recognition," in *INTERSPEECH*, 2019.