

Accelerating Transducers through Adjacent Token Merging

Yuang Li^{1*}, Yu Wu², Jinyu Li² Shujie Liu²

¹University of Cambridge, ²Microsoft

y1807@eng.cam.ac.uk, {Wu.Yu, jinyuli, shujliu}@microsoft.com

Abstract

Recent end-to-end automatic speech recognition (ASR) systems often utilize a Transformer-based acoustic encoder that generates embedding at a high frame rate. However, this design is inefficient, particularly for long speech signals due to the quadratic computation of self-attention. To address this, we propose a new method, Adjacent Token Merging (A-ToMe), which gradually combines adjacent tokens with high similarity scores between their key values. In this way, the total time step could be reduced, and the inference of both the encoder and joint network is accelerated. Experiments on LibriSpeech show that our method can reduce 57% of tokens and improve the inference speed on GPU by 70% without any notable loss of accuracy. Additionally, we demonstrate that A-ToMe is also an effective solution to reduce tokens in long-form ASR, where the input speech consists of multiple utterances.

Index Terms: speech recognition, transducer, adaptive subsampling

1. Introduction

The area of end-to-end (E2E) automatic speech recognition (ASR) has seen significant progress in recent years [1, 2, 3, 4, 5, 6, 7], and three main approaches have emerged: Connectionist Temporal Classification (CTC) [8], Attention-based Encoder-Decoder (AED) [9], and Recurrent Neural Network Transducers (RNN-T) [10]. These methods differ in how they handle the alignment between speech and text tokens. AED uses cross-attention, while CTC and RNN-T use redundant symbols like “blank”. The encoder of all these models processes fine-grained acoustic embedding at a high frame rate, leading to high computational costs. Given that the frequency of acoustic tokens is much higher than that of text tokens, such as phonemes or word pieces, significant redundancy exists. Hence, reducing the sequence length within the encoder is crucial for improving the efficiency of E2E ASR.

Adaptive subsampling techniques have been extensively researched in the field of Natural Language Processing, with token pruning being one of the most popular approaches [11, 12, 13, 14]. Token pruning involves removing tokens with low importance scores, which are usually determined by the cumulative attention score in a multi-head attention mechanism. The amount of pruned tokens can be determined through a fixed configuration [13], a learned threshold [14], or through evolutionary search [12]. These methods are often evaluated on sequence-level classification tasks rather than sequence-to-sequence tasks. For ASR, the majority of research focused on fixed-length subsampling such as progressively downsampling

through convolutional layers [15, 16]. Squeezeformer [17] further promoted the performance by using upsampling layer followed by downsampling. However, fixed-length subsampling can be sub-optimal as the duration of acoustic units varies considerably depending on the context and speaker. To address this issue, Meng et al. [18] proposed using a CIF [19] module with the supervision of phoneme boundaries to achieve an adaptive rate in the Distill-Hubert [20]. Cuervo et al. [21] proposed a two-level CPC network with a boundary predictor and an average pooling layer between the two levels.

In this study, we concentrate on a recently introduced adaptive subsampling technique called Token Merging [22]. The method was originally developed for use in Vision Transformers for classification tasks. It operates by merging tokens at any location that has high cosine similarity scores between their key values within the attention mechanism. However, it cannot be directly applied to the ASR task as preserving the temporal order of tokens is crucial. To address this issue, we propose a modified technique called Adjacent Token Merging (A-ToMe), which only merges tokens that are adjacent to each other. Furthermore, instead of merging a specific number of tokens, we introduce two different configurations to handle varying input lengths: fixed merge ratio and fixed merge threshold. Unlike previous studies, the proposed method does not explicitly predict boundaries. Instead, it gradually combines similar tokens to achieve a variable frame rate as the layers get deeper.

Experiments were conducted on the LibriSpeech [23] dataset using Transformer transducer [24] as the baseline. We adjusted the number of merged tokens by changing the merge ratio or threshold. In most instances, when the total merge ratio was below 60%, the model was able to maintain comparable word-error-rates (WERs) to the baseline while achieving a relative inference speedup of up to 35% and 70% on CPU and GPU respectively. Although the WER slightly increased as the number of merged tokens increased above 70%, the performance remained significantly better than that of the convolutional subsampling. Furthermore, we extended our experiments to long-form ASR where history utterances are concatenated with current utterances to provide context information and showed that A-ToMe is even more crucial for accelerating the encoder when the input speech becomes longer. Finally, we found that the model trained with a fixed threshold can adapt to multiple thresholds during inference which can promote future research in the direction of on-demand token reduction.

2. Methodology

2.1. Transformer transducer

RNN-T [10] is composed of an encoder, a prediction network, and a joint network. The Transformer transducer [24] extends

*Work done during an internship at Microsoft.

the RNN-T by using a Transformer-based [25] encoder that can effectively extract high-level representations \mathbf{h}_t from acoustic features \mathbf{X} (Equation 1). The prediction network generates embedding \mathbf{z}_u based on previously predicted non-blank symbols $\mathbf{Y}_{<u}$ (Equation 2). The joint network, implemented as a feed-forward network (FFN), combines the output of the encoder and the prediction network, and its output is converted to token probabilities through a Softmax function (Equation 3).

$$\mathbf{h}_t = f_{enc}(\mathbf{X}) \quad (1)$$

$$\mathbf{z}_u = f_{pred}(\mathbf{Y}_{<u}) \quad (2)$$

$$P(k|\mathbf{h}_t, \mathbf{z}_u) = \text{softmax}(f_{joint}(\mathbf{h}_t, \mathbf{z}_u)) \quad (3)$$

The “blank” token is used to help the alignment between the acoustic tokens from the encoder output with the text tokens. As there are many more acoustic tokens than text tokens, without token reduction, most output symbols are blank and will be removed in the final prediction.

2.2. Adjacent token merging

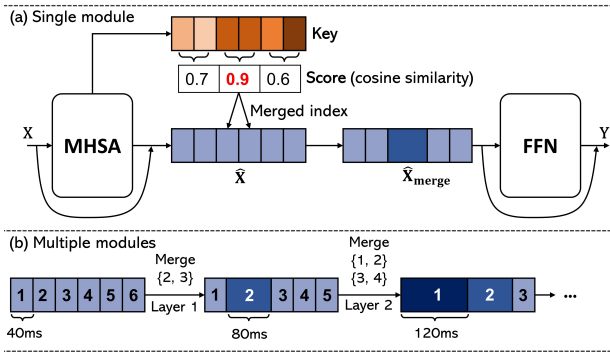


Figure 1: (a) A-ToMe module inside a Transformer layer. (b) The adaptive frame rate is achieved by stacking multiple modules.

As shown in Figure 1 (a), the proposed A-ToMe module is inserted between the multi-head self-attention (MHSA) and FFN of a Transformer layer. This module utilizes the key values used in the self-attention calculation to determine the cosine similarity score between each pair of neighboring tokens. Tokens with high similarity scores are merged by taking their average, and the upper boundary for the merge ratio per layer is 50%, which is equivalent to average pooling. Figure 1 (b) illustrates that multiple layers with the A-ToMe module can be stacked to achieve an adaptive frame rate, as merged tokens can be re-merged in subsequent layers. With n modules and the original token length of l , the highest possible token length is $2^n \times l$. The A-ToMe is simple and efficient, requiring no additional parameters, and it can be implemented in parallel using PyTorch’s [26] built-in functions without any loops.

To determine the number of tokens to merge, we employed two strategies that work for inputs of varying lengths: **1) Fixed merge threshold:** Tokens with a similarity score above a pre-defined threshold are merged. This strategy prevents dissimilar tokens from being merged at an earlier stage of the network, minimizing the loss of information. By adjusting the threshold, the number of merged tokens can be controlled, however, it is not possible to predict the exact number of merged tokens before inference. **2) Fixed merge ratio:** The similarity scores are

ranked and a fixed ratio of tokens with the highest scores are merged. As the layer becomes deeper, the number of tokens decreases, leading to a corresponding decrease in the number of merged tokens. The advantage is that the number of output tokens can be pre-calculated based on the merge ratio. In Section 3.2, we demonstrate that the fixed merge ratio can also be interpreted as using a higher threshold for deeper layers.

2.3. Long-form speech encoder

ASR performance improves with longer sequences as more contextual information becomes available [27, 28, 29, 30]. In this paper, we adopted a simple approach to utilize historical utterances by concatenating the acoustic features from historical utterances $\{\mathbf{X}_{i-n}, \dots, \mathbf{X}_{i-1}\}$ and the features \mathbf{X}_i of the current utterance in order before the encoder (Equation 4). While the outputs contain both historical and current embeddings, the joint network only considers \mathbf{H}_i , the embedding corresponding to the current utterance.

$$[\mathbf{H}_{i-n}; \dots; \mathbf{H}_{i-1}; \mathbf{H}_i] = f_{enc}([\mathbf{X}_{i-n}; \dots; \mathbf{X}_{i-1}; \mathbf{X}_i]) \quad (4)$$

This approach increases the computational intensity and memory consumption of the encoder due to the quadratic complexity of MHSA. Therefore, A-ToMe can be more important in this case. Additionally, different merging configurations can be applied to current and historical tokens, considering that current tokens may be more crucial. For instance, we can limit merging to only historical tokens or use a higher merge ratio for historical tokens than for current tokens.

3. Experiments

3.1. Experimental setup

Evaluations were performed on the LibriSpeech dataset [23], which comprises 960 hours of speech. We report the WERs on dev-clean, dev-other, test-clean, and test-other subsets. Moreover, we measured the average inference latency per utterance for the test-clean subset on GPU and CPU. For GPU latency, we used beam search with a beam size of 16 on NVIDIA Tesla V100 32GB GPU. For CPU latency, we used a single core of Intel Xeon CPU E5-2673 and employed greedy search instead of beam search for efficiency. The WERs reported were obtained using the beam search decoding method.

The encoder of the Transformer transducer has a four-layer VGG-like convolutional network that reduces the frame rate by a factor of four, followed by 18 Transformer layers. Each Transformer layer consists of an MHSA with an attention dimension of 512 and eight heads, and an FFN with a hidden dimension of 2048. The encoder takes as input 80-dimensional filterbank features with a frame length of 25 ms and a stride of 10 ms. The prediction network is a two-layer LSTM [31] with a hidden dimension of 1024, and the joint network has an embedding dimension of 512. 5000-wordpiece vocabulary [32] is used as the target. The whole model contains 94 million parameters, with the majority located in the encoder (59 million). We used a multitask loss function [33] including RNN-T, AED, and CTC losses with weights of 1, 0.3, and 0.3 respectively. The model was trained from scratch for 400,000 steps with AdamW [34] optimizer. SpecAug [35] was adopted for better generalizations.

The Transformer encoder incorporates A-ToMe every three layers, specifically at layers 2, 5, 8, 11, 14, and 17. In addition to presenting results for the un-merged model, we also report the outcomes of a traditional subsampling technique, achieved

Table 1: The comparison between different merging configurations. The average of merged tokens, token length, and latency/speed is calculated based on the test-clean subset. The ratio of merged tokens and the average token length refer to the tokens after the encoder.

Method	Merged Tokens (%)	Token Length (ms)	Latency (s) / Speed		WER Dev		WER Test	
			CPU	GPU	clean	other	clean	other
baseline	0	40	3.66 / 1.00×	1.07 / 1.00×	2.57	5.79	2.79	6.01
subsampling×2	50	80	2.99 / 1.22×	0.67 / 1.59×	2.71	6.08	2.90	6.29
subsampling×4	75	160	2.16 / 1.70×	0.50 / 2.16×	3.07	6.77	3.15	6.86
A-ToMe (fixed merge ratio)								
ratio/layer=10%	46	74	2.86 / 1.28×	0.74 / 1.46×	2.63	5.86	2.79	5.90
ratio/layer=15%	61	103	2.43 / 1.51×	0.62 / 1.73×	2.67	6.02	2.88	6.02
ratio/layer=20%	73	148	2.06 / 1.78×	0.53 / 2.04×	2.80	5.95	2.88	6.20
A-ToMe (fixed merge threshold)								
threshold=0.90	42	69	3.09 / 1.18×	0.78 / 1.37×	2.79	5.74	2.90	6.17
threshold=0.85	57	93	2.70 / 1.35×	0.63 / 1.70×	2.66	5.78	2.89	5.96
threshold=0.80	72	143	2.20 / 1.66×	0.54 / 1.98×	2.70	5.97	3.01	6.04

by adding extra convolutional layers to the VGG-like downsampler, as a baseline comparison. In the long-form ASR experiments, only a small batch size can be used which leads to slow convergence if the transducer is trained from scratch. Hence, we fine-tuned the utterance-based models for 200,000 steps with history utterances.

3.2. Utterance-based ASR results

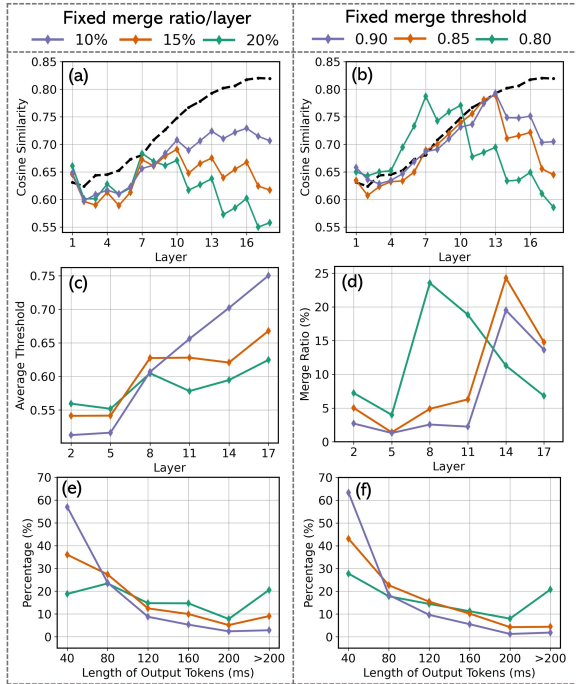


Figure 2: Visualizing different merge configurations for better understanding on the test-clean subset. (a, b) The change in cosine similarity between adjacent tokens from shallow to deep layers. (c) The average threshold at different layers when a fixed merge ratio is applied. (d) The average merge ratio at different layers when a fixed merge threshold is used. (e, f) Distribution of encoder output token lengths in percentages.

As shown in Table 1, the convolutional subsampling resulted in a significant increase in WERs, particularly on the dev-other and test-other subsets. For instance, when a subsampling rate of $\times 2$ and $\times 4$ was employed, there was a relative degradation in WER of 5% and 14% respectively, on the test-other subset. Fixed merge ratios led to a slight increase in WER as the number of merged tokens increased, but the impact was much smaller than that of convolutional subsampling. When the total merged tokens reached 73% (comparable to subsampling $\times 4$), the WER on test-other only increased by 3% relative. Moreover, when the merge ratio per layer was 10% and the total merged tokens were 46%, there was no noticeable degradation compared to the baseline. In terms of speed, A-ToMe contributed to much lower E2E latencies compared to the baseline by accelerating the encoder and reducing forward steps in decoding. The speed on the CPU became 1.28 to 1.78 times faster when using merge ratios per layer between 10% and 20%. Evaluations on the GPU directly reflected the computations since operations were not parallelized. For GPU performance, A-ToMe was even more effective with a speed of 1.46 to 2.01 times faster since the bottleneck here is the iterative forward steps during decoding, which increases with the number of tokens and is hard to parallelize. Compared to fixed ratio merging, using fixed merge thresholds performed slightly better when the number of merged tokens was high. For example, when using a threshold of 0.85, approximately 57% of tokens were merged with a negligible performance drop. However, the performance with the threshold of 0.9 was less satisfactory and the speedup on the CPU was more limited as fewer tokens at lower layers were merged.

Figure 2 provides visualizations to better understand A-ToMe. As shown by the dashed line in Figure 2 (a, b), without merging, the cosine similarity between adjacent tokens kept increasing as the layer became deeper, indicating considerable redundant information. With fixed merge ratios, the cosine similarity was kept at a relatively low level throughout the network, whereas with fixed merge thresholds, cosine similarity was reduced mostly at deep layers. This is because at low layers few tokens were merged as similarity scores were below the threshold. We can see from Figure 2 (d) that with thresholds of 0.85 and 0.90, most tokens were merged at layers 14 and 17. For a lower threshold of 0.8, more tokens can be merged at shallower layers like layer 8. This also means that enforcing a fixed merge ratio is similar to using a lower threshold for shallow

Table 2: Long-form ASR utilizing A-ToMe. 'Merge history' represents the merging of only previous utterances, while 'Merge all' indicates the merging of both previous and current utterances.

History	Merge	WER Dev		WER Test	
		clean	other	clean	other
0	-	2.57	5.79	2.79	6.01
1	-	2.54	5.45	2.61	5.64
1	history	2.55	5.40	2.69	5.66
1	all	2.48	5.47	2.74	5.61
2	-	2.35	5.20	2.57	5.38
2	history	2.39	5.17	2.64	5.50
2	all	2.42	5.31	2.67	5.49

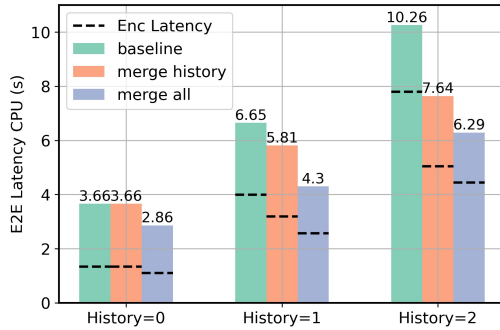


Figure 3: E2E latency on CPU with varying numbers of historical utterances. The region below the dashed line indicates the duration spent by the encoder, while the area above the dashed line represents the time consumed by the prediction and joint network during the decoding process.

layers (Figure 2 (c)). Figure 2 (e, f) shows the adaptive token length achieved by A-ToMe. With aggressive merging configurations such as the threshold of 0.8, more than 20% of tokens had lengths of more than 200 ms. For a lower merge ratio like 10% per layer, more than 50% of tokens remained unchanged at 40 ms. These visualizations highlight two main strengths of our approach: 1) variable token lengths instead of fixed lengths like 80 ms and 160 ms, and 2) gradual subsampling instead of subsampling all at once.

3.3. Long-form ASR results

In the long-form ASR experiment, we investigated two configurations that significantly merge history tokens while preserving more current information. The first configuration involves merging only historical tokens, with a fixed merge ratio of 20% per layer. The second configuration involves merging current tokens with a ratio of 10% per layer, and historical tokens with a ratio of 20% per layer. As shown in Table 2, ASR performance improved as more context was added. Without merging, the WER on test-other decreased from 6.01% to 5.38% when two historical utterances were used. When there was only one history utterance, the two merging configurations had similar WERs and were comparable to the results of the unmerged model. When there were two historical utterances, A-ToMe slightly affected the performance, and merging only historical tokens yielded slightly better results than merging both current

and historical tokens. It is worth noting that using a merge ratio of 20% per layer on historical tokens had a smaller impact on WERs than using it on current tokens. Figure 3 illustrates comparisons of E2E latency on the CPU. As the number of historical utterances increased from zero to two, there was a significant increase in latency from 3.66 seconds to 10.26 seconds when A-ToMe was not used. The encoder latency was primarily affected, whereas the rest of the model was less impacted since history tokens are removed after the encoder. Furthermore, the speed gain from A-ToMe improves as sequences become longer, with a shift from primarily benefiting the computation after the encoder to benefiting the encoder itself.

3.4. On-demand inference with different threshold

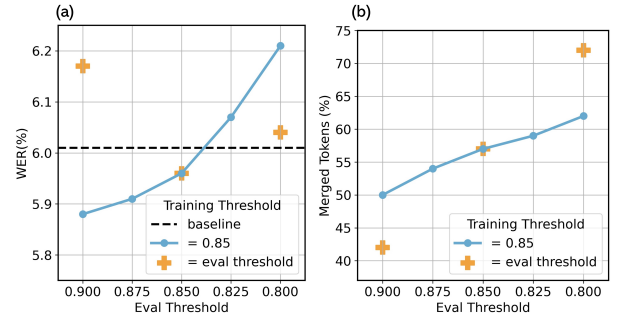


Figure 4: (a) The impact of applying distinct thresholds during inference compared to training on WER (test-other). (b) The proportion of merged tokens with different threshold setups.

On-demand compute reduction [36, 37] involves training a model that can be adapted to various computational requirements at inference without retraining. We conducted preliminary experiments to examine the on-demand capability of A-ToMe. Figure 4 (a) shows the WERs on test-other when the model was evaluated with different thresholds, even though it was trained with a fixed threshold of 0.85. Figure 4 (b) illustrates the percentage of merged tokens while the threshold was adjusted. By modifying the threshold, we can control the number of merged tokens during inference while maintaining good performance, especially at high thresholds. However, we observed that the performance was not as good with low thresholds, such as 0.8. Additionally, when using an on-demand setup with thresholds between 0.8 and 0.9, the percentage of merged tokens had a narrower range than the traditional setup where the same threshold was used for training and evaluation.

4. Conclusion

In this paper, we proposed a novel adaptive subsampling method called Adjacent Token Merging that progressively reduces the number of tokens in the encoder of the Transformer transducer. We emphasized the importance of variable frame rates and gradual subsampling. Experiments on utterance-based and long-form ASR showed that our approach could accelerate inference substantially while having minimal impact on recognition performance. Additionally, our approach can provide more flexibility in designing efficient ASR models and on-demand neural networks, which will facilitate future research. Moving forward, we plan to investigate more sophisticated merging strategies, and we will adapt our approach for streaming ASR.

5. References

- [1] Y. Miao, M. Gowayyed, and F. Metze, “EESN: End-to-end speech recognition using deep RNN models and WFST-based decoding,” in *Proc. ASRU*, 2015, pp. 167–174.
- [2] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition,” in *Proc. ICASSP*, 2016, pp. 4960–4964.
- [3] S. Watanabe, T. Hori, S. Kim, J. R. Hershey, and T. Hayashi, “Hybrid CTC/attention architecture for end-to-end speech recognition,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 8, pp. 1240–1253, 2017.
- [4] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach *et al.*, “Streaming end-to-end speech recognition for mobile devices,” in *Proc. ICASSP*, 2019, pp. 6381–6385.
- [5] J. Li, R. Zhao, H. Hu, and Y. Gong, “Improving RNN transducer modeling for end-to-end speech recognition,” in *Proc. ASRU*, 2019.
- [6] G. Saon, Z. Tüske, D. Bolanos, and B. Kingsbury, “Advancing RNN transducer technology for speech recognition,” in *Proc. ICASSP*, 2021, pp. 5654–5658.
- [7] J. Li, “Recent advances in end-to-end automatic speech recognition,” *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.
- [8] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks,” in *Proc. ICML*, Pittsburgh, Pennsylvania, USA, Jun. 2006, pp. 369–376.
- [9] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” in *Proc. NeurIPS*, Montreal, Canada, Dec. 2015, pp. 577–585.
- [10] A. Graves, “Sequence transduction with recurrent neural networks,” in *Proc. ICML*, Edinburgh, Scotland, Jun. 2012.
- [11] S. Goyal, A. R. Choudhury, S. Raje, V. Chakravarthy, Y. Sabharwal, and A. Verma, “Power-bert: Accelerating bert inference via progressive word-vector elimination,” in *Proc. ICML*, Vienna, Austria, Jul. 2020, pp. 3690–3699.
- [12] G. Kim and K. Cho, “Length-adaptive transformer: Train once with length drop, use anytime with search,” in *Proc. ACL-IJCNLP*, Jul. 2021, pp. 6501–6511.
- [13] H. Wang, Z. Zhang, and S. Han, “Spatten: Efficient sparse attention architecture with cascade token and head pruning,” in *Proc. HPCA*, Feb. 2021, pp. 97–110.
- [14] S. Kim, S. Shen, D. Thorsley, A. Gholami, W. Kwon, J. Hassoun, and K. Keutzer, “Learned token pruning for transformers,” in *Proc. KDD*, Washington DC, USA, Aug. 2022, pp. 784–794.
- [15] M. Burchi and V. Vielzeuf, “Efficient conformer: Progressive downsampling and grouped attention for automatic speech recognition,” in *Proc. ASRU*, Cartagena, Colombia, Dec. 2021, pp. 8–15.
- [16] W. Huang, W. Hu, Y. T. Yeung, and X. Chen, “Conv-transformer transducer: Low latency, low frame rate, streamable end-to-end speech recognition,” in *Proc. Interspeech*, Shanghai, China, Oct. 2020, pp. 5001–5005.
- [17] S. Kim, A. Gholami, A. E. Shaw, N. Lee, K. Mangalam, J. Malik, M. W. Mahoney, and K. Keutzer, “Squeezeformer: An efficient transformer for automatic speech recognition,” in *Proc. NeurIPS*, New Orleans, Louisiana, USA, Nov. 2022.
- [18] Y. Meng, H.-J. Chen, J. Shi, S. Watanabe, P. Garcia, H.-y. Lee, and H. Tang, “On compressing sequences for self-supervised speech models,” in *Proc. SLT*, Doha, Qatar, Jan. 2023, pp. 1128–1135.
- [19] L. Dong and B. Xu, “Cif: Continuous integrate-and-fire for end-to-end speech recognition,” in *Proc. ICASSP*, Barcelona, Spain, May 2020, pp. 6079–6083.
- [20] H.-J. Chang, S.-w. Yang, and H.-y. Lee, “Distilhubert: Speech representation learning by layer-wise distillation of hidden-unit bert,” in *Proc. ICASSP*, Singapore, May 2022, pp. 7087–7091.
- [21] S. Cuervo, A. Lancucki, R. Marxer, P. Rychlikowski, and J. K. Chorowski, “Variable-rate hierarchical cpc leads to acoustic unit discovery in speech,” in *Proc. NeurIPS*, New Orleans, Louisiana, USA, Nov. 2022.
- [22] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, “Token merging: Your ViT but faster,” in *Proc. ICLR*, Kigali, Rwanda, May 2023.
- [23] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “Librispeech: an asr corpus based on public domain audio books,” in *Proc. ICASSP*, South Brisbane, Queensland, Australia, Apr. 2015, pp. 5206–5210.
- [24] Q. Zhang, H. Lu, H. Sak, A. Tripathi, E. McDermott, S. Koo, and S. Kumar, “Transformer transducer: A streamable speech recognition model with transformer encoders and RNN-T loss,” in *Proc. ICASSP*, Barcelona, Spain, May 2020, pp. 7829–7833.
- [25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proc. NeurIPS*, Long Beach, California, USA, Dec. 2017.
- [26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: an imperative style, high-performance deep learning library,” in *Proc. NeurIPS*, Vancouver, Canada, Dec. 2019, pp. 8026–8037.
- [27] A. Narayanan, R. Prabhavalkar, C.-C. Chiu, D. Rybach, T. N. Sainath, and T. Strohman, “Recognizing long-form speech using streaming end-to-end models,” in *Proc. ASRU*, Sentosa, Singapore, Dec. 2019, pp. 920–927.
- [28] A. Schwarz, I. Sklyar, and S. Wiesler, “Improving RNN-T ASR accuracy using context audio,” in *Proc. Interspeech*, Brno, Czech Republic, Sep. 2021.
- [29] T. Hori, N. Moritz, C. Hori, and J. L. Roux, “Advanced long-context end-to-end speech recognition using context-expanded transformers,” in *Proc. Interspeech*, Brno, Czech Republic, Sep. 2021.
- [30] R. Masumura, N. Makishima, M. Ihori, A. Takashima, T. Tanaka, and S. Orihashi, “Hierarchical transformer-based large-context end-to-end ASR with large-context knowledge distillation,” in *Proc. ICASSP*, Toronto, Canada, Jun. 2021, pp. 5879–5883.
- [31] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [32] T. Kudo and J. Richardson, “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing,” in *Proc. EMNLP*, Brussels, Belgium, Oct. 2018, p. 66.
- [33] J.-J. Jeon and E. Kim, “Multitask learning and joint optimization for Transformer-RNN-Transducer speech recognition,” in *Proc. ICASSP*, Toronto, Canada, Jun. 2021, pp. 6793–6797.
- [34] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *Proc. ICLR*, New Orleans, Louisiana, USA, May 2019.
- [35] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, “SpecAugment: A simple data augmentation method for automatic speech recognition,” in *Proc. Interspeech*, Graz, Austria, Sep. 2019, pp. 2613–2617.
- [36] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once for all: Train one network and specialize it for efficient deployment,” in *Proc. ICLR*, Addis Ababa, Ethiopia, Apr. 2020.
- [37] A. Vyas, W.-N. Hsu, M. Auli, and A. Baevski, “On-demand compute reduction with stochastic wav2vec 2.0,” *Proc. Interspeech*, Sep. 2022.