Interspeech 2022
18-22 September 2022, Incheon, Korea

Pruned RNN-T for fast, memory-efficient ASR training

Fangjun Kuang, Liyong Guo, Wei Kang,
Long Lin, Mingshuang Luo, Zengwei Yao, Daniel Povey

Xiaomi Corp., Beijing, China
{kuangfangjun, guoliyong, kangwei1, linlong, luomingshuang, yaozengwei, dpovey}@xiaomi.com

Abstract

The RNN-Transducer (RNN-T) framework for speech recognition has been growing in popularity, particularly for deployed real-time ASR systems, because it combines high accuracy with naturally streaming recognition. One of the drawbacks of RNN-T is that its loss function is relatively slow to compute, and can use a lot of memory. Excessive GPU memory usage can make it impractical to use RNN-T loss in cases where the vocabulary size is large: for example, for Chinese character-based ASR.

We introduce a method for faster and more memory-efficient RNN-T loss computation. We first obtain pruning bounds for the RNN-T recursion using a simple joiner network that is linear in the encoder and decoder embeddings; we can evaluate this without using much memory. We then use those pruning bounds to evaluate the full, non-linear joiner network. The code is open-sourced and publicly available.

Index Terms: speech recognition, transducer, end-to-end, limited memory

1. Introduction

End-to-End (E2E) models have been growing in popularity in the field of automatic speech recognition (ASR). Unlike conventional ASR, which contains an acoustic model (AM) and a language model (LM) that are usually trained separately, E2E models use a single neural network model to predict words or graphemes directly from acoustic waveforms, which simplifies both training and decoding.

Three popular E2E models are connectionist temporal classification (CTC) models [1], attention-based models [2], and RNN-T models [3]. RNN-T models are naturally streaming and can be decoded frame synchronously. On one hand, it does not require the full context to predict the next token, as is required by attention-based models. On the other hand, there are no assumptions about frame independence given acoustic inputs that exist in CTC models. As a consequence, RNN-T models are very attractive in industry areas [4, 5].

The output of the RNN-T model [3] is usually a 4-D tensor of shape (N, T, U, V), where N is the batch size, T is the output length of the transcription network, U is the output length of the prediction network, and V is the vocabulary size. The output contains the probability distribution over all tokens in the utterance at each time step, which requires a lot of memory in training. For large vocabulary sizes (e.g. Chinese characters), this can severely limit the batch size and slow down training.

There are various efforts to reduce memory usage in RNN-T training. One technique is to remove paddings when combining the outputs from the prediction network and the transcription network [6]. Function merging [6] is also used to reduce memory consumption by computing gradient with respect to the logits directly. Another method is to use half-precision for training [7] at the cost of degradation in WER.

In this paper, instead of generating a probability distribution over all the tokens U at each time step, we propose pruned RNN-T, which limits the range of tokens at each time step from U to S, where $S \ll U$. Therefore, the output shape becomes (N, T, S, V), leading to less memory usage and faster training. We show that using pruned RNN-T for training can not only reduce memory consumption but can also achieve faster training, without performance degradation in WER. Furthermore, the above-mentioned techniques can be used together with pruned RNN-T to further reduce memory usage and accelerate training.

In a slight abuse of notation, when we refer to RNN-T in this paper we are speaking of the RNN-T loss itself, which is more properly speaking the transducer loss. In our experiments we use a Conformer encoder [8], not a recurrent encoder; and the decoder is stateless [9] rather than recurrent.

The code for this work is open-sourced and publicly available[1].

The remainder of the paper is structured as follows. In Section 2, we briefly describe the standard RNN-T and identify the reason for its high demand for memory in training, motivating us to propose pruned RNN-T in Section 3 to reduce memory consumption and to accelerate training. Section 4 gives the experiment setup for benchmarking different implementations of RNN-T loss and applying pruned RNN-T in ASR training. The results are given in Section 5. Finally, we conclude the paper in Section 6.

2. Motivations

Assume the acoustic input has been parameterized into a sequence of T feature frames $\mathbf{x} = \{x_t\}_{t=0}^{T-1}$. Also assume the transcript has been tokenized into a sequence of U tokens $\mathbf{y} = \{0 \le y_u < V | u = 0, 1, 2, \dots, U\}$, where V is the vocabulary size and token ID 0 is the blank token $\varnothing$; we have $\mathbf{y}_0 = 0$ as a beginning-of-sentence token, with the remaining positions corresponding to "real" words (there is no end-of-sentence token).

There are 3 components in the standard transducer model [3]: An encoder (a.k.a transcription network), a decoder (a.k.a prediction network), and a joiner (a.k.a joint network).

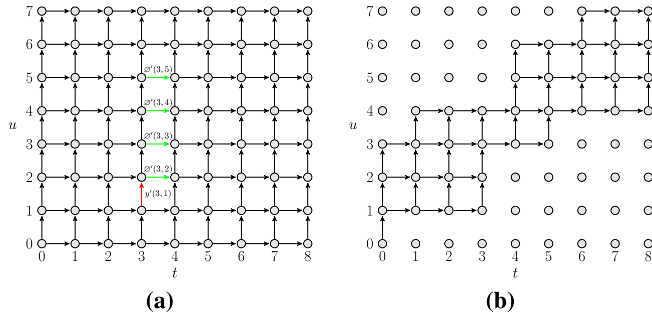
The encoder network functions as an acoustic model, transforming acoustic frames into a high-level representation. The encoder output is a 2-D tensor $\mathbf{X}$ with shape $(T, E)^2$, where E is the output dimension of the encoder. The decoder network is similar to a language model that tries to predict a distribution by conditioning on the last non-blank token. The output of the de-

[1] https://github.com/danpovey/fast_rnnt and https://github.com/k2-fsa/k2

[2] We assume the batch size is 1.

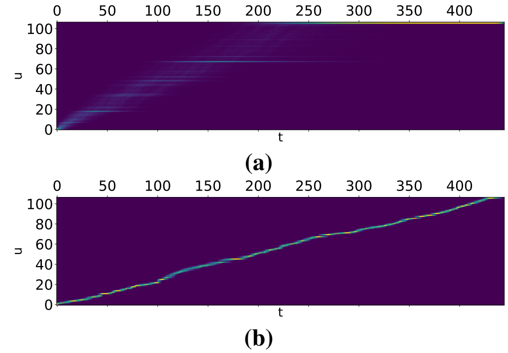
Copyright © 2022 ISCA

10.21437/Interspeech.2022-10340



(a) **(b)**

Figure 1: *Output log-probability lattice defined by the joiner output* $\mathbf{L}$ *in RNN-T. The vertical transition leaving node* (t, u) *has the log-probability* $y(t, u)$*, while the horizontal leaving node* (t, u) *has the log-probability* $\varnothing(t, u)$*. Log-probabilities for transitions that are not shown in the figure are set to minus infinity.* ***(a)*** *Lattice for the standard RNN-T.* ***(b)*** *Lattice for the pruned RNN-T.*



(a)

(b)

Figure 2: *Visualizations of node gradient in the standard transducer lattice for an utterance during training. The joiner consists of an adder and a softmax layer.* ***(a)*** *The model parameters are randomly initialized.* ***(b)*** *The model has been trained for several epochs.*

coder is a 2-D tensor $\mathbf{Y}$ of shape $(U+1, D)$, where D is the output dimension of the decoder. The joiner combines the outputs from the encoder network and the decoder network and produces a log-probability distribution $\mathbf{L}$ of shape $(T, U+1, V)$, where $L(t, u, v)$ is the log-probability for the token v to appear at position t, given $y_{0..u}$.

Similar to [3] (but in log-space, and with zero-based t index), we define

$$y(t, u) = L(t, u, y_{u+1}) \tag{1}$$

$$\varnothing(t, u) = L(t, u, \varnothing) \tag{2}$$

where $y(t, u)$ is the log-probability of the vertical transition leaving the node at position (t, u) in Figure 1(a), while $\varnothing(t, u)$ is the log-probability of the horizontal transition leaving the node at position (t, u) in Figure 1(a).

It usually uses the forward-backward algorithm [3] to compute the RNN-T loss. Let the forward variable $\alpha(t, u)$ be the log-probability outputting $y_{0..u}$ after seen $x_{0..t}$. It can be computed recursively using (3)

$$\alpha(t, u) = \text{LogAdd}\left(\alpha(t-1, u) + \varnothing(t-1, u)\right., \\ \left.\alpha(t, u-1) + y(t, u-1)\right)) \tag{3}$$

where LogAdd is defined as:

$$\text{LogAdd}(x, y) = \log(e^x + e^y) \tag{4}$$

$\alpha(0, 0)$ is initialized to 0 and the total log-likelihood of the sequence is $\alpha(T-1, U) + \varnothing(T-1, U)$.

The RNN-T loss computation can be quite memory- and compute-intensive because it has to compute $y(t, u)$ and $\varnothing(t, u)$ for all $t \leq T$ and $u \leq U$, so the output shape of the joint network has to be (N, T, U, V) if the batch size is N. This is much larger than the output required by CTC [1] and attention-based [2] models which involves shapes like (N, T, V) or (N, U, V).

Figure 2 shows the node gradient at each position (t, u) in Figure 1(a). It shows that: (1) At each time step, there is only a small range of nodes with a non-zero gradient; (2) Positions of nodes with non-zero gradient change monotonically from the lower left to the upper right. Therefore, instead of generating a probability distribution over all U tokens at each time step, we can limit the range of tokens from U to S, where $S \ll U$. That is, at time step t, we only compute the log-probabilities at positions $(t, p_t), (t, p_t + 1), \ldots, (t, p_t + S - 1)$ and the log-probabilities at other positions are set to minus infinity. Figure 1(b) shows the lattice for pruned RNN-T when S is 4.

As a consequence, the output shape of the joint network in the pruned RNN-T becomes (T, S, V). Since $S \ll U$, it reduces not only memory consumption but also computation, thus leading to faster training.

3. Pruned RNN-T

The basic idea of pruned RNN-T is to only evaluate the joiner network for the (t, u) pairs that significantly contribute to the final loss. We do this by computing the core recursion of (3) twice. The first time we do it with a "trivial" joiner network that is very fast to evaluate; we use this output to work out which indexes are important, and evaluate the full joiner network with a subset of (t, u) pairs.

3.1. Trivial joiner network

We formulate the trivial joiner network in such a way that the computation of $y(t, u)$ and $\varnothing(t, u)$ can be implemented by matrix multiplication and some simple lookups. (We omit the subscript trivial here for brevity; let it be understood that in this Sec. 3 we are talking about a separate version of these variables). In the trivial version of the joiner we project the encoder embedding and the decoder embedding respectively to un-normalized logprobs $L_{\text{enc}}(t, v)$ and $L_{\text{dec}}(t, u)$ respectively, and the joiner consists of simply adding these together and normalizing the log-probabilities:

$$L_{\text{trivial}}(t, u, v) \overset{def}{=} L_{\text{enc}}(t, v) + L_{\text{dec}}(u, v) - L_{\text{normalizer}}(t, u), \tag{5}$$

where

$$L_{\text{normalizer}}(t, u) \overset{def}{=} \log \sum_v \exp\left(L_{\text{enc}}(t, v) + L_{\text{dec}}(u, v)\right). \tag{6}$$

Equation (6) can be thought of as log-space matrix multiplication, and it can be implemented by conventional matrix multiplication after first applying offsets to ensure that overflow will not occur.

Thus, with the trivial joiner network we can compute

$y(t,u)$ and $\varnothing(t,u)$ without ever materializing any "large" matrices.

3.2. Pruning bounds

We introduce a constant S, e.g. $S=4$ or $S=5$, that represents the number of u indexes that we will evaluate for any t index[3]. For each t we will evaluate $L(t,u,v)$ only for integer positions $p_t \leq u < p_t + S$, where p_t (representing a position on the u axis) will be computed from the result of the recursion of the trivial joiner network. We will construct the $y(t,u)$ and $\varnothing(t,u)$ matrices with only these elements set, and all others set to $-\infty$, before doing the recursion (3).

3.2.1. Globally optimal pruning bounds

Ideally, we would like to find the sequence of integer pruning bounds $p = p_0, p_t, \ldots, p_{T-1}$ that would maximize the total retained probability based on the "trivial" joiner network, treating all $P(t,u,\cdot)$ for $u < p_t$ or $u \geq p_t + S$ as $-\infty$. That is, we are searching for the pruning bounds that would maximize the data likelihood given the trivial joiner, after setting all pruned logprobs to $-\infty$. This is a difficult optimization problem. Instead, we solve it by finding the *locally optimal* pruning bounds for each frame t, and then applying some continuity constraints to the result.

3.2.2. Locally optimal pruning bounds

The total data log-prob is given by $L_{\text{tot}} = \alpha(T-1,U) + \varnothing(T-1,U)$; let it be understood that we are talking about the "trivial" version of these variables. Define $y'(t,u)$ and $\varnothing'(t,u)$ as the derivatives of L_{tot} with respect to $y(t,u)$ and $\varnothing(t,u)$. We will need to compute these later anyway, in the neural network backprop; we do this "early" in the forward pass so that we can use the derivatives to compute the pruning bounds.

You can think of $y'(t,u)$ and $\varnothing'(t,u)$ as "occupation counts" in the interval $[0,1]$, which correspond to the probability of taking the upward and rightward transitions in Figure 1(a). Now consider the case where $S=4$ and we want to compute how much the total data log-probability would be decreased if we were to choose $p_t = 2$, for example. The total amount of retained probability mass can be lower-bounded by

$$\varnothing'(t,2) + \varnothing'(t,3) + \varnothing'(t,4) + \varnothing'(t,5) - y'(t,1), \quad (7)$$

indicated by the colored transitions in Figure 1(a). Our "locally-optimal" p_t is thus

$$p_t = \operatorname{argmax}_{p=0}^{U-S+1}\left(-y'(t,p-1) + \sum_{u=p}^{p+S-1} \varnothing'(t,u)\right). \quad (8)$$

[4] To explain (8), which corresponds to "green arcs minus red arc" in Figure 1(a): we want to get the total probability mass of all the paths that will be included if we use this pruning bound; and we can do by summing up the probabilities of exactly one link in each included path; (8) is not the only way to do this. The arcs summed in (8) include some probability mass that would actually be pruned out because it arises from lower u values, and we cancel this by subtracting $y'(t,p-1)$, which is red in the diagram. This may slightly over-compensate, to the extent that some of that subtracted probability mass makes it all the way through the pruned region.

It is possible for (8) to give a sequence of p_t values that are "inconsistent", i.e. that do not admit any complete path. For consistency, in addition to $0 \leq p_t \leq U - S + 1$, we require:

$$p_t \leq p_{t+1} \quad (9)$$
$$p_{t+1} - p_t < S. \quad (10)$$

We modify the pruning bounds p_t after computing them with (8) to ensure that they satisfy these constraints while otherwise changing them as little as possible. We won't expand due to length constraints[5].

3.3. Loss function

The loss function will be a combination of the data log-probability from the trivial joiner and the one with the full joiner network. If the full-joiner logprob is unscaled, we found it best to scale the trivial-joiner logprob by 0.5 in the loss function; it seems to have a regularizing effect.

3.3.1. Smoothed trivial joiner

Since (5) makes it natural to separate the encoder (acoustic) and decoder (language-model/LM) parts of $L_{\text{trivial}}(t,u,v)$, we decided to try interpolating the trivial joiner with even-more-trivial versions of the joiner network: specifically, versions where we use encoder-only and decoder-only versions of the probabilities. Let $\mathrm{LogSoftmax}$ be the log-softmax operation, applied along the appropriate axis (the v axis). So the version of the log-likelihoods we use in the recursion would be:

$$\begin{aligned} L_{\text{smoothed}}(t,u,v) = &\left(1 - \alpha^{\text{lm}} - \alpha^{\text{acoustic}}\right) L_{\text{trivial}}(t,u,v) \\ &+ \alpha^{\text{lm}} L_{\text{lm}}(t,u,v) \\ &+ \alpha^{\text{acoustic}} L_{\text{lm}}(t,u,v) \end{aligned} \quad (11)$$

where:

$$L_{\text{trivial}}(t,u,v) \stackrel{def}{=} \mathrm{LogSoftmax}_v\left(L_{\text{enc}}(t,v) + L_{\text{dec}}(u,v)\right) \quad (12)$$

$$L_{\text{acoustic}}(t,u,v) \stackrel{def}{=} \mathrm{LogSoftmax}_v\left(L_{\text{enc}}(t,v) + L_{\text{dec}}^{\text{avg}}(u,v)\right) \quad (13)$$

$$L_{\text{lm}}(t,u,v) \stackrel{def}{=} \mathrm{LogSoftmax}_v\, L_{\text{dec}}(u,v). \quad (14)$$

and $L_{\text{dec}}^{\text{avg}}(u,v)$ takes the role of a unigram language-model prior:

$$L_{\text{dec}}^{\text{avg}}(u,v) \stackrel{def}{=} \log \frac{1}{U+1} \sum_{u=0}^{U} \mathrm{Softmax}_v\, L_{\text{dec}}(u,v) \quad (15)$$

The reason for the asymmetry between the encoder and decoder here is that we want the decoder log-probs to be independently interpretable as language-model probabilities; this will be more convenient in case we need to access the language model probabilities independently for some reason later on.

4. Experiment Settings

The experiment contains 2 parts. In the first part, we benchmark the speed and memory usage of pruned RNN-T and several other open-source implementations for computing RNN-T

[3] This is called s_range in the code.

[4] The experiments in this paper were actually done with an earlier, less-accurate version of this computation

[5] Search for _adjust_pruning_lower_bound in https://github.com/k2-fsa/k2 for more details.

loss. In the second part, we apply pruned RNN-T for ASR training using the LibriSpeech corpus [10]. Note that we don't use any kind of language models during decoding.

4.1. Benchmarks of RNN-T loss computation

We compare the speed and peak memory usage of pruned RNN-T with the following open-source implementations: warp-transducer[6], torchaudio [11], optimized_transducer[7], and SpeechBrain [12].

Because padding matters in the RNN-T loss computation and to make the benchmark more realistic, instead of generating random data with random shapes we get the shapes for tokens and acoustics using the test-clean dataset from the LibriSpeech corpus. Two commonly used settings are benchmarked: (1) Fixed batch size. In this setting, the batch size is fixed and utterances in a batch have various lengths of durations. (2) Dynamic batch size. In this setting, we sort utterances by durations before batching them up to minimize paddings and the maximum number of frames in a batch before padding is limited to 10k at 100 frames per second.

The number of model output units is 500. We use an NVIDIA V100 GPU with 32 GB RAM to run the benchmarks. The code is open-sourced and publicly available[8].

4.2. Pruned RNN-T for ASR training

We use pruned RNN-T for ASR training with the LibriSpeech corpus, which consists of 960 hours of 16 kHz read English speech for training and two subsets, test-clean and test-other, for testing, each of which has approximately 5 hours speech data.

The inputs of the neural network model are 80-dimension log Mel filter bank features with a window size 25 ms and a window shift 10 ms. SpecAugment [13] and speed perturbation [14] with factors 0.9 and 1.1 are used to make the training more stable. The outputs of the model are 500 sentence pieces [15] with byte pair encoding (BPE) [16].

The encoder of the RNN-T model is a Conformer [8] with 12 layers. Each encoder layer has 8 self-attention[2] heads. The attention dimension and the feed-forward dimension are 512 and 2048, respectively. We use a stateless decoder [9], which consists of an embedding layer followed by a 1-D convolutional layer with a kernel size 2. The embedding dimension is 512. We use 8 NVIDIA V100 32GB GPUs for training.

To ensure convergence, for the first few thousand mini-batches we disable the pruned part of the loss by giving it a zero scale in the loss function; after the trivial loss starts to learn something meaningful, it will provide reasonable pruning bounds and we can enable the pruned loss. Due to space constraints we are not showing convergence results, but the pruned transducer converges very similarly to the conventional transducer.

5. Results

5.1. Benchmark results

Table 1 and Table 2 compare the speed and peak memory usage for different implementations. Pruned RNN-T has a clear advantage in not only speed but also memory usage in both benchmark settings. The memory efficiency means that we can use a larger batch size and vocabulary size during training, further increasing speed.

Table 1: *Speed and memory usage for different RNN-T loss implementations using fixed batch size 30.*

	Average time per batch (ms)	Peak memory usage (GB)
torchaudio	544	18.48
optimized_transducer	377	7.32
warp-transducer	276	18.63
SpeechBrain	459	18.63
pruned RNN-T	**64**	**3.73**

Table 2: *Speed and memory usage for different RNN-T loss implementations using dynamic batch size where utterances are sorted by durations before batching them up and the maximum number of frames in a batch before padding is limited to 10k at 100 frames per second.*

	Average time per batch (ms)	Peak memory usage (GB)
torchaudio	601	12.66
optimized_transducer	568	10.65
warp-transducer	211	12.76
SpeechBrain	264	12.76
pruned RNN-T	**38**	**2.59**

Table 3: *WERs on the LibriSpeech test-clean and test-other datasets for two models, where one is trained with pruned RNN-T and the other is trained using optimized_transducer. Beam search with beam size 4 is used for decoding. No external LMs are used during decoding.*

	test		train hours
	clean	other	per epoch
pruned RNN-T	2.56	6.27	1.17
optimized_transducer	2.61	6.46	2.33

5.2. Results for ASR training

Table 3 compares WERs on the LibriSpeech test-clean and test-other datasets for models trained with pruned RNN-T vs. optimized_transducer. Pruned RNN-T has slightly better WER than the model trained with unpruned RNN-T loss.

6. Conclusions

In this paper, we propose pruned RNN-T to reduce the memory usage and computation for RNN-T loss by limiting the range of tokens at each time step. Benchmark results show that pruned RNN-T is the fastest and consumes the least memory among commonly used open-source implementations: torchaudio, warp-transducer, SpeechBrain, and optimized_transducer. Furthermore, we demonstrate that using pruned RNN-T in ASR training can achieve competitive WERs with standard RNN-T on the LibriSpeech corpus.

[6]https://github.com/b-flo/warp-transducer/tree/espnet_v1.1

[7]https://github.com/csukuangfj/optimized_transducer

[8]https://github.com/csukuangfj/transducer-loss-benchmarking

7. References

[1] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[4] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang *et al.*, "Streaming end-to-end speech recognition for mobile devices," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6381–6385.

[5] R. Botros, T. N. Sainath, R. David, E. Guzman, W. Li, and Y. He, "Tied & reduced rnn-t decoder," *arXiv preprint arXiv:2109.07513*, 2021.

[6] J. Li, R. Zhao, H. Hu, and Y. Gong, "Improving rnn transducer modeling for end-to-end speech recognition," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 114–121.

[7] X. Zhang, F. Zhang, C. Liu, K. Schubert, J. Chan, P. Prakash, J. Liu, C.-F. Yeh, F. Peng, Y. Saraf *et al.*, "Benchmarking lf-mmi, ctc and rnn-t criteria for streaming asr," in *2021 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2021, pp. 46–51.

[8] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[9] M. Ghodsi, X. Liu, J. Apfel, R. Cabrera, and E. Weinstein, "Rnn-transducer with stateless prediction network," in *ICASSP*, 2020, pp. 7049–7053.

[10] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5206–5210.

[11] Y.-Y. Yang, M. Hira, Z. Ni, A. Chourdia, A. Astafurov, C. Chen, C.-F. Yeh, C. Puhrsch, D. Pollack, D. Genzel, D. Greenberg, E. Z. Yang, J. Lian, J. Mahadeokar, J. Hwang, J. Chen, P. Goldsborough, P. Roy, S. Narenthiran, S. Watanabe, S. Chintala, V. Quenneville-Blair, and Y. Shi, "Torchaudio: Building blocks for audio and speech processing," *arXiv preprint arXiv:2110.15018*, 2021.

[12] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, L. Lugosch, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, J.-C. Chou, S.-L. Yeh, S.-W. Fu, C.-F. Liao, E. Rastorgueva, F. Grondin, W. Aris, H. Na, Y. Gao, R. D. Mori, and Y. Bengio, "SpeechBrain: A general-purpose speech toolkit," 2021, arXiv:2106.04624.

[13] 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," *Interspeech 2019*, Sep 2019.

[14] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Sixteenth annual conference of the international speech communication association*, 2015.

[15] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Nov. 2018, pp. 66–71.

[16] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units." Association for Computational Linguistics (ACL), 2016, pp. 1715–1725.