TORCHAUDIO: BUILDING BLOCKS FOR AUDIO AND SPEECH PROCESSING

Yao-Yuan Yang[*†2], *Moto Hira*[*1], *Zhaoheng Ni*[*1], *Artyom Astafurov*[1], *Caroline Chen*[1], *Christian Puhrsch*[1], *David Pollack*[3], *Dmitriy Genzel*[1], *Donny Greenberg*[1], *Edward Z. Yang*[1], *Jason Lian*[†1], *Jeff Hwang*[1], *Ji Chen*[1], *Peter Goldsborough*[4], *Sean Narenthiran*[5], *Shinji Watanabe*[6], *Soumith Chintala*[1], *Vincent Quenneville-Bélair*[†1]

[1]Meta AI, [2]University of California, San Diego, [3]Solvemate Gmbh, [4]Anduril Industries, [5]Grid AI Labs, [6]Carnegie Mellon University

ABSTRACT

This document describes version 0.10 of `TorchAudio`: building blocks for machine learning applications in the audio and speech processing domain. The objective of `TorchAudio` is to accelerate the development and deployment of machine learning applications for researchers and engineers by providing off-the-shelf building blocks. The building blocks are designed to be GPU-compatible, automatically differentiable, and production-ready. `TorchAudio` can be easily installed from Python Package Index repository and the source code is publicly available under a BSD-2-Clause License (as of September 2021) at `https://github.com/pytorch/audio`. In this document, we provide an overview of the design principles, functionalities, and benchmarks of `TorchAudio`. We also benchmark our implementation of several audio and speech operations and models. We verify through the benchmarks that our implementations of various operations and models are valid and perform similarly to other publicly available implementations.

Index Terms— Open-Source Toolkit, Speech Recognition, Audio Processing, Text-to-Speech

1. INTRODUCTION

In recent years, the usage of open-source toolkits for the development and deployment of state-of-the-art machine learning applications has grown rapidly. General-purpose open-source toolkits such as `TensorFlow` [1] and `PyTorch` [2] are used extensively. However, building applications for different domains requires additional domain-specific functionality. To accelerate development, we established the `TorchAudio` toolkit, which provides building blocks for machine learning applications in the audio and speech domain.

To provide building blocks for modern machine learning applications in the audio/speech domain, we aim to enable the following three important properties for each building block: (1) GPU compute capability, (2) automatic differentiablility, (3) production readiness. GPU compute capability enables the acceleration of the training and inference process. Automatic differentiation allows to directly incorporate these functionalities into neural networks enabling full end-to-end learning. Production readiness means that the trained models can be easily ported to various environments including mobile devices running Android and iOS platforms.

The stability of the toolkit is our top priority. Our goal is not to include all state-of-the-art technologies, but rather to provide high-quality canonical implementations that users can build upon. The functionalities provided in `TorchAudio` include basic input/output of audio files, audio/speech datasets, audio/speech operations, and implementations of canonical machine learning models. To the best of our knowledge, `TorchAudio` is the only toolkit that has building blocks that span these four functionalities, with the majority satisfying the three aforementioned properties.

To understand how our implementation compares with others, we conduct an empirical study. We benchmark five audio/speech operations and three machine learning models and compare them with publicly available reference implementations. For audio/speech operations, we find that our implementations achieves or exceed parity in run time performance. For the machine learning models, we show that our implementations achieve or exceed parity in output quality.

`TorchAudio` is designed to facilitate new projects in the audio/speech domain. We have built up a thriving community on Github with many users and developers. As of September 2021, we have addressed more than 400 issues and merged more than 1250 pull requests. We have more than 40 users that contributed more than 100 lines of code, with many of these users being external contributors. We also observe extensive usage of `TorchAudio` in the open source community. Currently, there are more than 350 projects that are forked from `TorchAudio` and more than $5,420$ public repositories on Github that depend on `TorchAudio`.

This paper is organized as follows. We start with reviewing existing open source audio/speech toolkits in Section 2. Next, we talk about `TorchAudio`'s design principles in Section 3, and then we introduce the structure of `TorchAudio` and the functionalities available in Section 4. Finally, we showcase the empirical study we conducted in Section 5.

2. RELATED WORK

Arguably, modern deep learning applications for audio/speech processing are mainly developed within either the `numpy` [3], `TensorFlow` [1], or `PyTorch` [2] ecosystem. Users tend to commit to one ecosystem when building their applications to avoid complicated dependencies and conversions between ecosystems that increase the cost of maintenance. There are many excellent open-source toolkits that provide audio/speech related building blocks and functionalities within each ecosystem. For example, `librosa` [4]

[*] Equal contribution on the writing of this paper. The rest of authors are in alphabetical order.

[†] Work done while at Meta AI.

is built for numpy, DDSP [5] and TensorFlowASR[1] are created for TensorFlow, and TorchAudio is designed to work with PyTorch. TorchAudio is the go-to toolkit for basic audio/speech functionalities inside the PyTorch ecosystem.

TorchAudio provides important low-level functionalities like audio input/output, spectrogram computation, and unified interface for accessing dataset. In the PyTorch ecosystem, there are many useful audio/speech toolkits available including Asteroid [6], ESPnet [7], Espresso [8], fairseq [9], NeMo [10]. Pytorch-Kaldi [11], and SpeechBrain [12]. These toolkits provide ready-to-use models for various applications, including speech recognition, speech enhancement, speech separation, speaker recognition, text-to-speech, etc. One common thing with these toolkits is that they all have TorchAudio as a dependency so that they do not have to re-implement the basic operations[2].

3. DESIGN PRINCIPLES

TorchAudio is designed to provide building blocks for the development of audio applications within the PyTorch ecosystem. The functionalities in TorchAudio are built to be compatible with PyTorch core functionalities like neural network containers and data loading utilities. This allows users to easily incorporate functionalities in TorchAudio into their use cases. For simplicity and ease of use, TorchAudio does not depend on any other Python packages except PyTorch.

We ensure that most of our functionalities satisfy three key properties: (1) GPU compute capability, (2) automatic differentiability, and (3) production readiness. To ensure GPU compute capability, we implement all computational-heavy operations, such as convolution and matrix multiplication, using GPU-compatible logic. To ensure automatic differentiability, we perform a gradient test on all necessary functionalities. Lastly, for production readiness, we make sure that most of our functionalities are compilable into *TorchScript*[3]. TorchScript is an intermediate representation that can be saved from Python code, serialized, and then later loaded by a process where there is no Python dependency, such as systems where only C++ is supported[4] and mobile platforms including Android[5] and iOS[6].

Since users depend on TorchAudio for foundational building blocks, we must maintain its stability and availability to a wide range of platforms, particularly as its user base grows. We support all platforms that PyTorch supports, which includes major platforms (Windows, MacOS, and Linux) with major Python versions from 3.6 to 3.9. The features are classified into three release status: *stable*, *beta*, and *prototype*. Once a feature reaches the stable release status, we aim to maintain backward compatibility whenever it is possible. Any backward-compatibility-breaking change on stable features can be released only after two release cycles have elapsed from when it was proposed. On the other hand, the beta and prototype features can be less stable in terms of its API, but they allow users to benefit from accessing newly implemented features earlier. The difference between beta and prototype is that prototype features are generally even less stable and will only be accessible from the source code or nightly build but not from PyPI or Conda. Before the release of a new version, we use release candidates to collect user feedback before the official release.

We apply modern software development practices to ensure the quality of the codebase. For code readability, the code linting complies with PEP-8 recommendations, with only small modifications. All developments are conducted openly on GitHub and all functions are thoroughly documented using Sphinx[7]. We also have implemented more than 6000 test cases using the pytest[8] framework and use CircleCI[9] for continuous integration testing.

We strive to be selective about the features we implement. We aim to maintain only the most essential functionalities to keep the design of the package lean. This way, it is easier to keep each releases stable and the maintenance cost down. We provide models that researchers would employ as baselines for comparison. For example, Tacotron2 [13] is usually used as the baseline when developing text-to-speech systems. Therefore, we included the implementation of Tacotron2.

For sophisticated downstream applications, we provide examples and tutorials[10] on important downstream applications, therefore, it is easy for users to adapt TorchAudio to various applications and use cases.

4. PACKAGE STRUCTURE AND FUNCTIONALITIES

Here, we provide an overview of the functionalities and structure of TorchAudio. TorchAudio supports all four categories of functionalities mentioned previously, including audio input/output (I/O), audio/speech datasets, audio/speech operations, and audio/speech models.

4.1. Audio input/output (I/O)

Audio I/O is implemented under the backend submodule to provide a user-friendly interface for loading audio files into PyTorch tensors and saving tensors into audio files. We ported SoX[11] into TorchAudio and made it torchscriptable (meaning it is production-ready and can be used for on-device stream ASR support) for this purpose. Optionally, TorchAudio also provides interfaces for different backends such as soundfile and kaldi-io[12].

4.2. Audio/speech datasets

The access to audio/speech datasets is implemented under the dataset submodule. The goal of this submodule is to provide a user-friendly interface for accessing commonly used datasets. Currently, we support 11 datasets, including Librispeech [14], VCTK [15], LJSpeech [16], etc. These datasets are designed to greatly simplify data pipelines and are built to be compatible with the data loading utilities provided in PyTorch, i.e. torch.utils.data.DataLoader[13]. This allows users to access a wide range of off-the-shelf features including customizing data loading order, automatic batching, multi-process data loading, and automatic memory pinning.

[1]https://github.com/TensorSpeech/TensorFlowASR
[2]We say a package depends on TorchAudio if they have import torchaudio in their repository.
[3]https://pytorch.org/docs/stable/jit.html
[4]https://github.com/pytorch/audio/tree/main/examples/libtorchaudio
[5]https://github.com/pytorch/android-demo-app/tree/master/SpeechRecognition
[6]https://github.com/pytorch/ios-demo-app/tree/master/SpeechRecognition
[7]https://www.sphinx-doc.org/en/master/
[8]https://pytest.org/
[9]https://circleci.com/
[10]https://pytorch.org/tutorials/
[11]https://sourceforge.net/projects/sox/
[12]https://github.com/vesis84/kaldi-io-for-python
[13]https://pytorch.org/docs/stable/data.html

4.3. Audio/speech operations

We implemented common audio operations under three submodules – `functional`, `transform`, and `sox_effects`.

`functional`. In this submodule, we provide 49 different commonly used operations. The operations can be further categorized into four categories including general utility, complex utility, filtering and feature extraction. In general utilities, we provide utilities such as creating of the discrete cosine transformation matrix. In complex utilities, we provide functions such as computing the complex norms. For filtering, we include things like bandpass filters. Finally, for feature extraction, we have algorithms such as the computation of spectral centroid.

`transform`. There are 26 different transforms implemented under the submodule – `torch.nn.Module`. The objects here are designed to work as neural network building blocks. They are designed to interface with `PyTorch`'s neural network containers (i.e. `torch.nn.Sequential`, `torch.nn.ModuleList`, etc.) As such, they can be seamlessly integrated with all the neural network features in `PyTorch`. The functionalities implemented in this submodule include Spectrogram/InverseSpectrogram, mel-frequency cepstrum coefficients (MFCC), minimum variance distortionless response (MVDR) beamforming, RNN-Transducer Loss, etc.

`sox_effects`. `SoX` is a popular command line program that provides a many audio processing functionality. To make these functionalities more accessible, we modified from the source code of `Sox` version 14.4.2 and ported it into `TorchAudio`. This module is fully torchscriptable and supports 58 different sound effects, such as resampling, pitch shift, etc.

4.4. Machine learning models

We implement various canonical machine learning models in `torchaudio.models` across a wide range of audio-related applications. For speech recognition, we have DeepSpeech [17], HuBERT [**?**], Wav2letter [18], and Wav2Vec 2.0 [19]. For speech separation, we have Conv-TasNet [20]. For text-to-speech and neural vocoder, we have Tacotron2 [21] paired with WaveRNN [21]. These models are selected to support basic downstream tasks.

5. EMPIRICAL EVALUATIONS

This section provides a comparison of our implementations with others. We present the performance benchmarks for five implementations of audio/speech operations and three implementations of machine learning models. We benchmark the audio/speech operations using run time as the main performance metric. For machine learning models, the performance measures are chosen based on the task they are performing. The experiments are conducted on Amazon AWS p4d.24xlarge instance with NVIDIA A100 GPUs. The implementation for the empirical evaluation can be found in `https://github.com/yangarbiter/torchaudio-benchmark`.

5.1. Audio/Speech Operations

The pioneering `librosa` [4] is arguably one of the most commonly used toolkits in the `Python` community. We select five operations that are implemented in both `TorchAudio` and `librosa` and compare their run time. For `librosa`, we use version 0.8.0 installed from `PyPI`. The inputs are all floating point data type. We consider five different operations including phase vocoder, Griffin-Lim algorithm, MFCC, spectral centroid, and spectrogram. For MFCC, spectral centroid, spectrogram, we measure the run time of 100 runs, and for Griffin-Lim algorithm, phase vocoder, we measure the run time of 10 run. Operations in `TorchAudio` can be compiled into torchscript during runtime and can be accelerated with a GPU, thus, we also include the just-in-time compiled (jitted) and GPU accelerated versions of the `TorchAudio` operations.

The benchmark results are shown in Figure 1. We have two findings. First, we can see that in terms of CPU operations, `TorchAudio` is able to have even or slightly better speed. In addition, the support of GPU operation allows `TorchAudio` to take advantage of hardware for acceleration. Second, we found that for the jitted version, it runs slightly slower on CPU, but slightly faster on GPU. However, this difference is marginal.

5.2. Audio/Speech Applications

Here, we benchmark the performance of three models, including WaveRNN [21], Tacotron2 [13], and and Conv-TasNet [20]. We compare each model independently with a popular open-source implementation available online.

	PESQ (↑)	STOI (↑)	MOS (↑)
fatchord's WaveRNN[14]	3.43	0.96	3.78 ± 0.08
`TorchAudio` WaveRNN	3.68	0.97	3.88 ± 0.08
`Nvidia` WaveGlow	3.78	0.97	3.72 ± 0.05
`TorchAudio` WaveRNN	3.63	0.97	3.86 ± 0.05
ground truth	-	-	4.09 ± 0.05

Table 1. The average PESQ, STOI, and MOS of the vocoders. In addition, we also show the standard error of the mean for the MOS metric due to larger variance. These numbers are higher the better. The `TorchAudio` WaveRNN on the upper part of the table is trained on the dataset split by fatchord's WaveRNN implementation, and the `TorchAudio` WaveRNN on the lower part of the table is trained on the dataset split from Nvidia.

WaveRNN. For WaveRNN, we consider the task to reconstruct a waveform from the corresponding mel-spectrogram. We measure the reconstruction performance with the wide band version of the *perceptual evaluation of speech quality (PESQ)*, and *short-time objective intelligibility (STOI)* on the LJSpeech [16] dataset. When evaluating with PESQ, we resample the waveform to 16000 Hz and measure the reconstruction performance with a publicly available implementation[15]. We compare the performance of our model with the popular WaveRNN implementation from fatchord[16] as well as the WaveGlow [22] from Nvidia[17] The results are shown in Table 1. Note that the pretrained models for fatchord and Nvidia are trained on a different dataset split, therefore we train and evaluate our WaveRNNs on each of their datasets, respectively.

In addition, we also consider a subjective evaluation metric, the mean opinion score (MOS). For the comparison with fatchord's WaveRNN, we use the full test set, which consists of 50 audio samples;

[14]`https://github.com/fatchord/WaveRNN`
[15]`https://github.com/ludlows/python-pesq`
[16]`https://github.com/fatchord/WaveRNN`
[17]`https://github.com/NVIDIA/DeepLearningExamples`

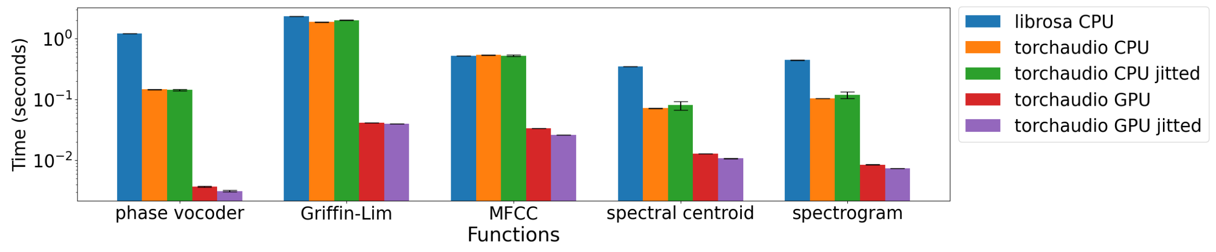

Fig. 1. The mean and standard error of the running time over five repeated experiments. We compare over five different audio processing functions using the implementations from `TorchAudio` (including just-in-time compiled (jitted) and GPU accelerated versions) and `librosa`.

for the comparison with Nvidia's WaveGlow, we randomly selected 100 audio examples from the test set to generate the audio sample. These generated samples are then sent to Amazon's Mechanical Turk, a human rating service, to have human raters score each audio sample. Each evaluation is conducted independently from each other, so the outputs of two different models are not directly compared when raters are assigning a score to them. Each audio sample is rated by at least 4 raters on a scale from 1 to 5 with 1 point increments, where 1 is lowest means that the rater perceived the audio as unlike a human speech, and 5 means that the rater perceived the audio as similar to a human speech.

The results in Table 1 show that our implementation is able to achieve similar performance comparing with both fatchord's and Nvidia's models. This verifies the validity of our implementation. In addition, our implementation also utilizes the latest DistributedDataParallel[18] for multi-GPU training, which gives additional running time benefit over fatchord's implementation.

Tacotron2. For Tacotron2, we measure the performance with the *mel cepstral distortion (MCD)* metric, and we compare our implementation with Nvidia's implementation[19]. The evaluation is conducted on the LJSpeech dataset. We train our implemented Tacotron2 using the training testing split from Nvidia's repository and compare it with Nvidia's pretrained model. We train our Tacotron2 similarly to the default parameter provided in Nvidia's repository, which has 1500 epochs, an initial learning rate of 10^{-3}, a weight decay of 10^{-6}, clips the norm of the gradient to 1.0, and uses the Adam optimizer. Table 2 shows that the difference in MCD between the two implementations is very small (around 1%), which verifies the validity of our implementation.

	MCD (↓)
Nvidia[20]	2.294
`TorchAudio`	2.305

Table 2. The MCD of different implementation of Tacotron2. The numbers are lower the better.

Conv-TasNet. We train the Conv-TasNet model on the *sep_clean* task of Libri2Mix dataset. The sample rate is 8000 Hz. We follow the same model configuration and training strategy in the Asteroid training pipeline: 1) We use the negative value of scale-invariant scale-to-distortion ratio (Si-SDR) as the loss function with a permutation invariant training (PIT) criterion; 2) We clip the gradient with a threshold of 5; 3) We use 1e-3 as the initial learning rate and halve it if the validation loss stops decreasing for 5 epochs. We evaluate the performance by Si-SDR improvement (Si-SDRi) and signal-to-distortion ratio improvement (SDRi). Table 3 shows that our implementation slightly outperforms that in Asteroid on both Si-SDRi and SDRi metrics.

	Si-SDRi (dB) (↑)	SDRi (dB) (↑)
Asteroid[21]	14.7	15.1
`TorchAudio`	15.3	15.6

Table 3. The scale-invariant signal-to-distortion ratio (Si-SDR) and SDR improvements of Conv-TasNet (numbers are higher the better).

6. CONCLUSION

This paper provides a brief summary of the `TorchAudio` toolkit. With increasing number of users, this project is under active development so that it can serve our goal to accelerate the development and deployment of audio-related machine learning applications. The roadmap for future work can be found on our Github page[22].

7. ACKNOWLEDGEMENT

We thank the release engineers, Nikita Shulga and Eli Uriegas, for helping with creating releases. We thank all contributors on Github, including but not limited to Bhargav Kathivarapu, Chin-Yun Yu, Emmanouil Theofanis Chourdakis, Kiran Sanjeevan, Prabhat Roy, and Tomas Osorio[23]. We thank Abdelrahman Mohamed, Alban Desmaison, Andrew Gibiansky, Anjali Chourdia, Brian Vaughan, Ching-Feng Yeh, Christoph Boeddeker, Fabian-Robert Stöter, Jan Schlüter, Jay Mahadeokar, Keunwoo Choi, Manuel Pariente, Mirco Ravanelli, Piotr Żelasko, Samuele Cornell, Yangyang Shi, Yossi Adi, and Zhi-Zheng Wu for the meaningful discussions about the design of `TorchAudio`. We also thank the Meta marketing team that helped with promoting this work.

[18] `https://pytorch.org/tutorials/intermediate/ddp_tutorial.html`
[19] `https://github.com/NVIDIA/DeepLearningExamples`
[20] `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/SpeechSynthesis/Tacotron2`
[21] `https://github.com/asteroid-team/asteroid/tree/master/egs/librimix/ConvTasNet`
[22] `https://github.com/pytorch/audio`
[23] Due to page limit, we only include the name of people who contributed 1000+ lines as of September 2021.

8. REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020.

[4] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*. Citeseer, 2015, vol. 8, pp. 18–25.

[5] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts, "Ddsp: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.

[6] Manuel Pariente, Samuele Cornell, Joris Cosentino, Sunit Sivasankaran, Efthymios Tzinis, Jens Heitkaemper, Michel Olvera, Fabian-Robert Stöter, Mathieu Hu, Juan M. Martín-Doñas, David Ditter, Ariel Frank, Antoine Deleforge, and Emmanuel Vincent, "Asteroid: the PyTorch-based audio source separation toolkit for researchers," in *Proc. Interspeech*, 2020.

[7] Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai, "ESPnet: End-to-end speech processing toolkit," in *Proceedings of Interspeech*, 2018, pp. 2207–2211.

[8] Yiming Wang, Tongfei Chen, Hainan Xu, Shuoyang Ding, Hang Lv, Yiwen Shao, Nanyun Peng, Lei Xie, Shinji Watanabe, and Sanjeev Khudanpur, "Espresso: A fast end-to-end neural speech recognition toolkit," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019.

[9] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 2019, pp. 48–53.

[10] Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, et al., "Nemo: a toolkit for building ai applications using neural modules," *arXiv preprint arXiv:1909.09577*, 2019.

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

[12] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, et al., "Speechbrain: A general-purpose speech toolkit," *arXiv preprint arXiv:2106.04624*, 2021.

[13] Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al., "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4779–4783.

[14] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[15] Junichi Yamagishi, Christophe Veaux, Kirsten MacDonald, et al., "Cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit (version 0.92)," 2019.

[16] Keith Ito and Linda Johnson, "The lj speech dataset," `https://keithito.com/LJ-Speech-Dataset/`, 2017.

[17] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al., "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[18] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *arXiv preprint arXiv:1609.03193*, 2016.

[19] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[20] Yi Luo and Nima Mesgarani, "Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation," *IEEE/ACM transactions on audio, speech, and language processing*, vol. 27, no. 8, pp. 1256–1266, 2019.

[21] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu, "Efficient neural audio synthesis," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2410–2419.

[22] Ryan Prenger, Rafael Valle, and Bryan Catanzaro, "Waveglow: A flow-based generative network for speech synthesis," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 3617–3621.