INTERSPEECH 2020
October 25–29, 2020, Shanghai, China

Asteroid: the PyTorch-based audio source separation toolkit for researchers

Manuel Pariente[1], *Samuele Cornell*[2], *Joris Cosentino*[1], *Sunit Sivasankaran*[1], *Efthymios Tzinis*[3], *Jens Heitkaemper*[4], *Michel Olvera*[1], *Fabian-Robert Stöter*[5], *Mathieu Hu*[1], *Juan M. Martín-Doñas*[6], *David Ditter*[7], *Ariel Frank*[8], *Antoine Deleforge*[1], *Emmanuel Vincent*[1]

[1]Université de Lorraine, CNRS, Inria, LORIA, France
[2]Università Politecnica delle Marche, Italy
[3]University of Illinois at Urbana-Champaign, USA
[4]Universität Paderborn, Germany
[5]Inria and LIRMM, University of Montpellier, France
[6]Universidad de Granada, Spain
[7]Universität Hamburg, Germany
[8]Technion - Israel Institute of Technology, Israel
https://github.com/mpariente/asteroid

Abstract

This paper describes `Asteroid`, the `PyTorch`-based audio source separation toolkit for researchers. Inspired by the most successful neural source separation systems, it provides all neural building blocks required to build such a system. To improve reproducibility, Kaldi-style recipes on common audio source separation datasets are also provided. This paper describes the software architecture of `Asteroid` and its most important features. By showing experimental results obtained with `Asteroid`'s recipes, we show that our implementations are at least on par with most results reported in reference papers. The toolkit is publicly available at `github.com/mpariente/asteroid`.

Index Terms: source separation, speech enhancement, open-source software, end-to-end

1. Introduction

Audio source separation, which aims to separate a mixture signal into individual source signals, is essential to robust speech processing in real-world acoustic environments [1]. Classical open-source toolkits such as FASST [2], HARK [3], ManyEars [4] and openBliSSART [5] which are based on probabilistic modelling, non-negative matrix factorization, sound source localization and/or beamforming have been successful in the past decade. However, they are now largely outperformed by deep learning-based approaches, at least on the task of single-channel source separation [6–10].

Several open-source toolkits have emerged for deep learning-based source separation. These include `nussl` (Northwestern University Source Separation Library) [11], `ONSSEN` (An Open-source Speech Separation and Enhancement Library) [12], `Open-Unmix` [13], and countless isolated implementations replicating some important papers.

Both `nussl` and `ONSSEN` are written in `PyTorch` [14] and provide training and evaluation scripts for several state-of-the art methods. However, data preparation steps are not provided and experiments are not easily configurable from the command line. `Open-Unmix` does provide a complete pipeline from data preparation until evaluation, but only for the Open-Unmix model on the music source separation task. Regarding the isolated implementations, some of them only contain the model, while others provide training scripts but assume that training data has been generated. Finally, very few provide the complete pipeline. Among the ones providing evaluation scripts, differences can often be found, e.g., discarding short utterances or splitting utterances in chunks and discarding the last one.

This paper describes `Asteroid` (Audio source separation on Steroids), a new open-source toolkit for deep learning-based audio source separation and speech enhancement, designed for researchers and practitioners. Based on `PyTorch`, one of the most widely used dynamic neural network toolkits, `Asteroid` is meant to be user-friendly, easily extensible, to promote reproducible research, and to enable easy experimentation. As such, it supports a wide range of datasets and architectures, and comes with recipes reproducing some important papers. `Asteroid` is built on the following principles:

1. Abstract only where necessary, i.e., use as much native `PyTorch` code as possible.
2. Allow importing third-party code with minimal changes.
3. Provide all steps from data preparation to evaluation.
4. Enable recipes to be configurable from the command line.

We present the audio source separation framework in Section 2. We describe `Asteroid`'s main features in Section 3 and their implementation in Section 4. We provide example experimental results in Section 5 and conclude in Section 6.

2. General framework

While `Asteroid` is not limited to a single task, single-channel source separation is currently its main focus. Hence, we will only consider this task in the rest of the paper. Let x be a single channel recording of J sources in noise:

$$x(t) = \sum_{j=1}^{J} s_j(t) + n(t), \tag{1}$$

http://dx.doi.org/10.21437/Interspeech.2020-1673

where $\{s_j\}_{j=1..J}$ are the source signals and n is an additive noise signal. The goal of source separation is to obtain source estimates $\{\widehat{s}_j\}_{j=1..J}$ given x.

Most state-of-the-art neural source separation systems follow the encoder-masker-decoder approach depicted in Fig. 1 [8, 9, 15, 16]. The encoder computes a short-time Fourier transform (STFT)-like representation $\mathbf{X}$ by convolving the time-domain signal x with an analysis filterbank. The representation $\mathbf{X}$ is fed to the masker network that estimates a mask for each source. The masks are then multiplied entrywise with $\mathbf{X}$ to obtain sources estimates $\{\widehat{\mathbf{S}}_j\}_{j=1..J}$ in the STFT-like domain. The time-domain source estimates $\{\widehat{s}_j\}_{j=1..J}$ are finally obtained by applying transposed convolutions to $\{\widehat{\mathbf{S}}_j\}_{j=1..J}$ with a synthesis filterbank. The three networks are jointly trained using a loss function computed on the masks or their embeddings [6, 17, 18], on the STFT-like domain estimates [7, 15, 19], or directly on the time-domain estimates [8–10, 16, 20].

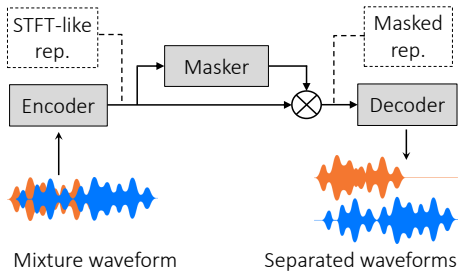

Figure 1: *Typical encoder-masker-decoder architecture.*

3. Functionality

`Asteroid` follows the encoder-masker-decoder approach, and provides various choices of filterbanks, masker networks, and loss functions. It also provides training and evaluation tools and recipes for several datasets. We detail each of these below.

3.1. Analysis and synthesis filterbanks

As shown in [20–23], various filterbanks can be used to train end-to-end source separation systems. A natural abstraction is to separate the filterbank object from the encoder and decoder objects. This is what we do in `Asteroid`. All filterbanks inherit from the `Filterbank` class. Each `Filterbank` can be combined with an `Encoder` or a `Decoder`, which respectively follow the `nn.Conv1d` and `nn.ConvTranspose1d` interfaces from `PyTorch` for consistency and ease of use. Notably, the `STFTFB` filterbank computes the STFT using simple convolutions, and the default filterbank matrix is orthogonal.

`Asteroid` supports free filters [8, 9], discrete Fourier transform (DFT) filters [19, 21], analytic free filters [22], improved parameterized sinc filters [22, 24] and the multi-phase Gammatone filterbank [23]. Automatic pseudo-inverse computation and dynamic filters (computed at runtime) are also supported. Because some of the filterbanks are complex-valued, we provide functions to compute magnitude and phase, and apply magnitude or complex-valued masks. We also provide interfaces to `NumPy` [25] and `torchaudio`[1]. Additionally, Griffin-Lim [26, 27] and multi-input spectrogram inversion (MISI) [28] algorithms are provided.

[1] https://github.com/pytorch/audio

3.2. Masker network

`Asteroid` provides implementations of widely used masker networks: TasNet's stacked long short-term memory (LSTM) network [8], Conv-Tasnet's temporal convolutional network (with or without skip connections) [9], and the dual-path recurrent neural network (DPRNN) in [16]. Open-Unmix [13] is also supported for music source separation.

3.3. Loss functions — Permutation invariance

`Asteroid` supports several loss functions: mean squared error, scale-invariant signal-to-distortion ratio (SI-SDR) [9, 29], scale-dependent SDR [29], signal-to-noise ratio (SNR), perceptual evaluation of speech quality (PESQ) [30], and affinity loss for deep clustering [6].

Whenever the sources are of the same nature, a permutation-invariant (PIT) loss shall be used [7, 31]. `Asteroid` provides an optimized, versatile implementation of PIT losses. Let $\boldsymbol{s} = [s_j(t)]_{j=1\ldots J}^{t=0\ldots T}$ and $\widehat{\boldsymbol{s}} = [\widehat{s}_j(t)]_{j=1\ldots J}^{t=0\ldots T}$ be the matrices of true and estimated source signals, respectively. We denote as $\widehat{\boldsymbol{s}}_\sigma = [\widehat{s}_{\sigma(j)}(t)]_{j=1\ldots J}^{t=0\ldots T}$ a permutation of $\boldsymbol{s}$ by $\sigma \in \mathcal{S}_\mathcal{J}$, where $\mathcal{S}_\mathcal{J}$ is the set of permutations of $[1, ..., J]$. A PIT loss $\mathcal{L}_{\text{PIT}}$ is defined as

$$\mathcal{L}_{\text{PIT}}(\theta) = \min_{\sigma \in \mathcal{S}_\mathcal{J}} \mathcal{L}(\widehat{\boldsymbol{s}}_\sigma, \boldsymbol{s}), \tag{2}$$

where $\mathcal{L}$ is a classical (permutation-dependent) loss function, which depends on the network's parameters θ through $\widehat{\boldsymbol{s}}_\sigma$.

We assume that, for a given permutation hypothesis σ, the loss $\mathcal{L}(\widehat{\boldsymbol{s}}_\sigma, \boldsymbol{s})$ can be written as

$$\mathcal{L}(\widehat{\boldsymbol{s}}_\sigma, \boldsymbol{s}) = \mathcal{G}\big(\mathcal{F}(\widehat{\boldsymbol{s}}_{\sigma(1)}, \boldsymbol{s}_1), ..., \mathcal{F}(\widehat{\boldsymbol{s}}_{\sigma(J)}, \boldsymbol{s}_J)\big) \tag{3}$$

where $\boldsymbol{s}_j = [s_j(0), \ldots, s_j(T)]$, $\widehat{\boldsymbol{s}}_j = [\widehat{s}_j(0), \ldots, \widehat{s}_j(T)]$, $\mathcal{F}$ computes the pairwise loss between a single true source and its hypothesized estimate, and $\mathcal{G}$ is the *reduce* function, usually a simple mean operation. Denoting by $\mathbf{F}$ the $J \times J$ pairwise loss matrix with entries $\mathcal{F}(\widehat{\boldsymbol{s}}_i, \boldsymbol{s}_j)$, we can rewrite (2) as

$$\mathcal{L}_{\text{PIT}}(\theta) = \min_{\sigma \in \mathcal{S}_\mathcal{J}} \mathcal{G}\big(\mathbf{F}_{\sigma(1)1}, ..., \mathbf{F}_{\sigma(J)J}\big) \tag{4}$$

and reduce the computational complexity from $J!$ to J^2 by precomputing $\mathbf{F}$'s terms. Taking advantage of this, `Asteroid` provides `PITLossWrapper`, a simple yet powerful class that can efficiently turn any pairwise loss $\mathcal{F}$ or permutation-dependent loss $\mathcal{L}$ into a PIT loss.

3.4. Datasets

`Asteroid` provides baseline recipes for the following datasets: wsj0-2mix and wsj0-3mix [6], WHAM [32], WHAMR [33], LibriMix [34] FUSS [35], Microsoft's Deep Noise Suppression challenge dataset (DNS) [36], SMS-WSJ [37], Kinect-WSJ [38], and MUSDB18 [39]. Their characteristics are summarized and compared in Table 1. wsj0-2mix and MUSDB18 are today's reference datasets for speech and music separation, respectively. WHAM, WHAMR, LibriMix, SMS-WSJ and Kinect-WSJ are recently released datasets which address some shortcomings of wsj0-2mix. FUSS is the first open-source dataset to tackle the separation of arbitrary sounds. Note that wsj0-2mix is a subset of WHAM which is a subset of WHAMR.

Table 1: *Datasets currently supported by* `Asteroid`*. * White sensor noise. ** Background environmental scenes.*

	wsj0-mix	WHAM	WHAMR	Librimix	DNS	SMS-WSJ	Kinect-WSJ	MUSDB18	FUSS
Source types	speech	speech	speech	speech	speech	speech	speech	music	sounds
# sources	2 or 3	2	2	2 or 3	1	2	2	4	0 to 4
Noise		✓	✓	✓	✓	*	✓		✓**
Reverb			✓			✓	✓	✓	✓
# channels	1	1	1	1	1	6	4	2	1
Sampling rate	16k	16k	16k	16k	16k	16k	16k	16k	16k
Hours	30	30	30	210	100 (+aug.)	85	30	10	55 (+aug.)
Release year	2015	2019	2019	2020	2020	2019	2019	2017	2020

3.5. Training

For training source separation systems, `Asteroid` offers a thin wrapper around `PyTorch-Lightning` [40] that seamlessly enables distributed training, experiment logging and more, without sacrificing flexibility. Regarding the optimizers, we rely on native `PyTorch` and `torch-optimizer` [2].

3.6. Evaluation

Evaluation is performed using `pb_bss_eval`[3], a sub-toolkit of `pb_bss`[4] [41] written for evaluation. It natively supports most metrics used in source separation: SDR, signal-to-interference ratio (SIR), signal-to-artifacts ratio (SAR) [42], SI-SDR [29], PESQ [43], and short-time objective intelligibility (STOI) [44].

4. Implementation

`Asteroid` follows Kaldi-style recipes [45], which involve several stages as depicted in Fig. 2. These recipes implement the entire pipeline from data download and preparation to model training and evaluation. We show the typical organization of a recipe's directory in Fig. 3. The entry point of a recipe is the `run.sh` script which will execute the following stages:

- **Stage 0**: Download data that is needed for the recipe.
- **Stage 1**: Generate mixtures with the official scripts, optionally perform data augmentation.
- **Stage 2**: Gather data information into text files expected by the corresponding `DataLoader`.
- **Stage 3**: Train the source separation system.
- **Stage 4**: Separate test mixtures and evaluate.

In the first stage, necessary data is downloaded (if available) into a storage directory specified by the user. We use the official scripts provided by the dataset's authors to generate the data, and optionally perform data augmentation. All the information required by the dataset's `DataLoader` such as filenames and paths, utterance lengths, speaker IDs, etc., is then gathered into text files under `data/`. The training stage is finally followed by the evaluation stage. Throughout the recipe, log files are saved under `logs/` and generated data is saved under `exp/`.

[2]https://github.com/jettify/pytorch-optimizer
[3]https://pypi.org/project/pb-bss-eval/
[4]https://github.com/fgnt/pb_bss

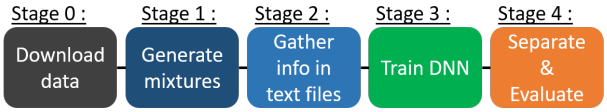

Figure 2: *Typical recipe flow in* `Asteroid`*.*

```
├── data/                    # Output of stage 2
├── exp/                     # Store experiments
├── local/
│   ├── conf.yml             # Training config
│   └── other_scripts.py     # Dataset specific
├── run.sh                   # Entry point
├── model.py                 # Model definition
├── train.py                 # Training scripts
└── eval.py                  # Evaluation script
```

Figure 3: *Typical directory structure of a recipe.*

As can be seen in Fig. 4, the model class, which is a direct subclass of `PyTorch`'s `nn.Module`, is defined in `model.py`. It is imported in both training and evaluation scripts. Instead of defining constants in `model.py` and `train.py`, most of them are gathered in a YAML configuration file `conf.yml`. An argument parser is created from this configuration file to allow modification of these values from the command line, with `run.sh` passing arguments to `train.py`. The resulting modified configuration is saved in `exp/` to enable future reuse. Other arguments such as the experiment name, the number of GPUs, etc., are directly passed to `run.sh`.

5. Example results

To illustrate the potential of `Asteroid`, we compare the performance of state-of-the-art methods as reported in the corresponding papers with our implementation. We do so on two common source separation datasets: wsj0-2mix [6] and WHAMR [33]. wsj0-2mix consists of a 30 h training set, a 10 h validation set, and a 5 h test set of single-channel two-speaker mixtures without noise and reverberation. Utterances taken from the Wall Street Journal (WSJ) dataset are mixed together at random SNRs between -5 dB and 5 dB. Speakers in the test set are different from those in the training and validation sets. WHAMR [33] is a noisy and reverberant extension of wsj0-2mix. Experiments are conducted on the 8 kHz *min* version of both datasets. Note that we use wsj0-2mix separation, WHAM's clean separation, and WHAMR's anechoic clean separation tasks inter-

model.py

```python
from asteroid.filterbanks import make_enc_dec
from asteroid.masknn import DPRNN

# Encoder-Masker-Decoder with DPRNN
class Model(torch.nn.Module):
  def __init__(self):
    super().__init__()
    # Create matching encoder/decoder
    self.enc, self.dec = make_enc_dec(
      fb_name='free', n_filters=512,
      kernel_size=16, stride=8
      )
    # Create Dual-Path RNN mask network
    self.masker = DPRNN(in_chan=512, n_src=3)

  def forward(self, x):
    tf_rep = self.enc(x)
    est_masks = self.masker(tf_rep)
    masked_tf_rep = est_masks * tf_rep
    return self.dec(masked_tf_rep)
```

conf.yml

```yaml
# Filterbank config
filterbank:
  n_filters: 512
  kernel_size: 16
  stride: 8
# Network config
masknet:
  n_blocks: 8
  n_repeats: 3
# Training config
training:
  epochs: 200
  batch_size: 8
# Optim config
optim:
  optimizer: adam
  lr: 0.001
# Data config
data:
  task: sep_clean
  mode: min
```

train.py

```python
from torch.utils.data import DataLoader as Loader
import pytorch_lightning as pl
from asteroid.data.wham_dataset import WhamDataset
from asteroid.engine.system import System
from asteroid.losses import PITLossWrapper, neg_sisdr
from model import Model

# Define training and validation DataLoader
train_loader = Loader(WhamDataset(train_dir, 'sep_clean'))
val_loader = Loader(WhamDataset(val_dir, 'sep_clean'))
# Define model and optimizers
model = Model(model_args)
optimizer = Adam(model.parameters(), lr=1e-3, ...)
# Define Loss function.
loss_func = PITLossWrapper(neg_sisdr)
# Train source separation system
system = System(model, loss_func, optimizer,
                train_loader, val_loader)
trainer = pl.Trainer(distributed_backend='dp', gpus=4)
trainer.fit(system)
```

Figure 4: *Simplified code example.*

changeably as the datasets only differ by a global scale.

Table 2 reports SI-SDR improvements (SI-SDR$_i$) on the test set of wsj0-2mix for several well-known source separation systems. In Table 3, we reproduce Table 2 from [33] which reports the performance of an improved TasNet architecture (more recurrent units, overlap-add for synthesis) on the four main tasks of WHAMR: anechoic separation, noisy anechoic separation, reverberant separation, and noisy reverberant separation. On all four tasks, `Asteroid`'s recipes achieved better results than originally reported, by up to 2.6 dB.

Table 2: *SI-SDR$_i$ (dB) on the wsj0-2mix test set for several architectures. ks stands for for kernel size, i.e., the length of the encoder and decoder filters.*

	Reported	Using `Asteroid`
Deep Clustering [46]	9.6	9.8
TasNet [8]	10.8	15.0
Conv-TasNet [9]	15.2	16.2
TwoStep [15]	16.1	15.2
DPRNN (ks $=$ 16) [16]	16.0	17.7
DPRNN (ks $=$ 2) [16]	18.8	19.3
Wavesplit [10]	20.4	-

Table 3: *SI-SDR$_i$ (dB) on the four WHAMR tasks using the improved TasNet architecture in [33].*

Noise	Reverb	Reported [33]	Using `Asteroid`
		14.2	**16.8**
✓		12.0	**13.7**
	✓	8.9	**10.6**
✓	✓	9.2	**11.0**

In both Tables 2 and 3, we can see that our implementations outperform the original ones in most cases. Most often, the aforementioned architectures are trained on 4-second segments. For the architectures requiring a large amount of memory (e.g., Conv-TasNet and DPRNN), we reduce the length of the training segments in order to increase the batch size and stabilize gradients. This, as well as using a weight decay of 10^{-5} for recurrent architectures increased the final performance of our systems.

`Asteroid` was designed such that writing new code is very simple and results can be quickly obtained. For instance, starting from stage 2, writing the TasNet recipe used in Table 3 took less than a day and the results were simply generated with the command in Fig. 5, where the GPU ID is specified with the `--id` argument.

```
n=0
for task in clean noisy reverb reverb_noisy
    do
    ./run.sh --stage 3 --task $task --id $n
    n=$(($n+1))
done
```

Figure 5: *Example command line usage.*

6. Conclusion

In this paper, we have introduced `Asteroid`, a new open-source audio source separation toolkit designed for researchers and practitioners. Comparative experiments show that results obtained with `Asteroid` are competitive on several datasets and for several architectures. The toolkit was designed such that it can quickly be extended with new network architectures or new benchmark datasets. In the near future, pre-trained models will be made available and we intend to interface with `ESPNet` to enable end-to-end multi-speaker speech recognition.

7. Acknowledgements

Experiments presented in this paper were partially carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`). High Performance Computing resources were partially provided by the EXPLOR centre hosted by the University de Lorraine.

8. References

[1] E. Vincent, T. Virtanen, and S. Gannot, *Audio Source Separation and Speech Enhancement*, 1st ed. Wiley, 2018.

[2] Y. Salaün, E. Vincent, N. Bertin, N. Souviraà-Labastie, X. Jaureguiberry, D. T. Tran, and F. Bimbot, "The Flexible Audio Source Separation Toolbox Version 2.0," ICASSP Show & Tell, 2014.

[3] K. Nakadai, H. G. Okuno, H. Nakajima, Y. Hasegawa, and H. Tsujino, "An open source software system for robot audition HARK and its evaluation," in *Humanoids*, 2008, pp. 561–566.

[4] F. Grondin, D. Létourneau, F. Ferland, V. Rousseau, and F. Michaud, "The ManyEars open framework," *Autonomous Robots*, vol. 34, pp. 217–232, 2013.

[5] B. Schuller, A. Lehmann, F. Weninger, F. Eyben, and G. Rigoll, "Blind enhancement of the rhythmic and harmonic sections by nmf: Does it help?" in *ICA*, 2009, pp. 361–364.

[6] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe, "Deep clustering: discriminative embeddings for segmentation and separation," in *ICASSP*, 2016, pp. 31–35.

[7] D. Yu, M. Kolbæk, Z. Tan, and J. Jensen, "Permutation invariant training of deep models for speaker-independent multi-talker speech separation," in *ICASSP*, 2017, pp. 241–245.

[8] Y. Luo and N. Mesgarani, "TasNet: Time-domain audio separation network for real-time, single-channel speech separation," in *ICASSP*, 2018, pp. 696–700.

[9] ——, "Conv-TasNet: Surpassing ideal time–frequency magnitude masking for speech separation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 27, no. 8, pp. 1256–1266, 2019.

[10] N. Zeghidour and D. Grangier, "Wavesplit: End-to-end speech separation by speaker clustering," *arXiv preprint arXiv:2002.08933*, 2020.

[11] E. Manilow, P. Seetharaman, and B. Pardo, "The Northwestern University Source Separation Library," in *ISMIR*, 2018.

[12] Z. Ni and M. I. Mandel, "Onssen: an open-source speech separation and enhancement library," *arXiv preprint arXiv:1911.00982*, 2019.

[13] F.-R. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, "Open-Unmix - a reference implementation for music source separation," *J. Open Source Soft.*, vol. 4, no. 41, p. 1667, 2019.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[15] E. Tzinis, S. Venkataramani, Z. Wang, C. Subakan, and P. Smaragdis, "Two-step sound source separation: Training on learned latent targets," in *ICASSP*, 2020, pp. 31–35.

[16] Y. Luo, Z. Chen, and T. Yoshioka, "Dual-path RNN: Efficient long sequence modeling for time-domain single-channel speech separation," in *ICASSP*, 2020, pp. 46–50.

[17] Y. Isik, J. Le Roux, Z. Chen, S. Watanabe, and J. R. Hershey, "Single-channel multi-speaker separation using deep clustering," in *Interspeech*, 2016, pp. 545–549.

[18] Z. Chen, Y. Luo, and N. Mesgarani, "Deep attractor network for single-microphone speaker separation," in *ICASSP*, 2017.

[19] J. Heitkaemper, D. Jakobeit, C. Boeddeker, L. Drude, and R. Haeb-Umbach, "Demystifying TasNet: A dissecting approach," in *ICASSP*, 2020, pp. 6359–6363.

[20] F. Bahmaninezhad, J. Wu, R. Gu, S.-X. Zhang, Y. Xu, M. Yu, and D. Yu, "A comprehensive study of speech separation: Spectrogram vs waveform separation," in *Interspeech*, 2019.

[21] I. Kavalerov, S. Wisdom, H. Erdogan, B. Patton, K. Wilson, J. Le Roux, and J. R. Hershey, "Universal sound separation," in *WASPAA*, 2019, pp. 175–179.

[22] M. Pariente, S. Cornell, A. Deleforge, and E. Vincent, "Filterbank design for end-to-end speech separation," in *ICASSP*, 2020.

[23] D. Ditter and T. Gerkmann, "A multi-phase gammatone filterbank for speech separation via TasNet," in *ICASSP*, 2020, pp. 36–40.

[24] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with SincNet," in *SLT*, 2018, pp. 1021–1028.

[25] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Computing in Science and Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[26] D. Griffin and J. Lim, "Signal estimation from modified short-time Fourier transform," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 2, pp. 236–243, 1984.

[27] N. Perraudin, P. Balazs, and P. Søndergaard, "A fast Griffin-Lim algorithm," in *WASPAA*, 2013, pp. 1–4.

[28] D. Gunawan and D. Sen, "Iterative phase estimation for the synthesis of separated sources from single-channel mixtures," *IEEE Signal Process. Letters*, vol. 17, no. 5, pp. 421–424, 2010.

[29] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "SDR — half-baked or well done?" in *ICASSP*, 2019, pp. 626–630.

[30] J. M. Martín-Doñas, A. M. Gomez, J. A. Gonzalez, and A. M. Peinado, "A deep learning loss function based on the perceptual evaluation of the speech quality," *IEEE Signal Process. Letters*, vol. 25, no. 11, pp. 1680–1684, 2018.

[31] M. Kolbæk, D. Yu, Z.-H. Tan, and J. Jensen, "Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 25, no. 10, pp. 1901–1913, 2017.

[32] G. Wichern, J. Antognini, M. Flynn, L. R. Zhu, E. McQuinn, D. Crow, E. Manilow, and J. Le Roux, "WHAM!: extending speech separation to noisy environments," in *Interspeech*, 2019.

[33] M. Maciejewski, G. Wichern, E. McQuinn, and J. Le Roux, "WHAMR!: Noisy and reverberant single-channel speech separation," in *ICASSP*, 2020, pp. 696–700.

[34] J. Cosentino, S. Cornell, M. Pariente, A. Deleforge, and E. Vincent, "LibriMix: An open-source dataset for generalizable speech separation," *arXiv preprint arXiv:2005.11262*, 2020.

[35] S. Wisdom, H. Erdogan, D. P. W. Ellis, R. Serizel, N. Turpault, E. Fonseca, J. Salamon, P. Seetharaman, and J. R. Hershey, "What's all the fuss about free universal sound separation data?" *in preparation*, 2020.

[36] C. K. A. Reddy, E. Beyrami, H. Dubey, V. Gopal, R. Cheng *et al.*, "The Interspeech 2020 deep noise suppression challenge: Datasets, subjective speech quality and testing framework," *arXiv preprint arXiv:2001.08662*, 2020.

[37] L. Drude, J. Heitkaemper, C. Boeddeker, and R. Haeb-Umbach, "SMS-WSJ: Database, performance measures, and baseline recipe for multi-channel source separation and recognition," *arXiv preprint arXiv:1910.13934*, 2019.

[38] S. Sivasankaran, E. Vincent, and D. Fohr, "Analyzing the impact of speaker localization errors on speech separation for automatic speech recognition," 2020.

[39] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "The MUSDB18 corpus for music separation," 2017.

[40] W. Falcon *et al.*, "Pytorch lightning," https://github.com/PytorchLightning/pytorch-lightning, 2019.

[41] L. Drude and R. Haeb-Umbach, "Tight integration of spatial and spectral features for BSS with deep clustering embeddings," in *Interspeech*, 2017, pp. 2650–2654.

[42] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 14, no. 4, pp. 1462–1469, 2006.

[43] A. W. Rix, J. G. Beerends, M. P. Hollier, and A. P. Hekstra, "Perceptual evaluation of speech quality (PESQ) — a new method for speech quality assessment of telephone networks and codecs," in *ICASSP*, vol. 2, 2001, pp. 749–752.

[44] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "An algorithm for intelligibility prediction of time–frequency weighted noisy speech," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 19, no. 7, pp. 2125–2136, 2011.

[45] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek *et al.*, "The Kaldi speech recognition toolkit," in *ASRU*, 2011.

[46] Z. Wang, J. Le Roux, and J. R. Hershey, "Alternative objective functions for deep clustering," in *ICASSP*, 2018, pp. 686–690.