The Journal of Open Source Software

Software Design and User Interface of ESPnet-SE++: Speech Enhancement for Robust Speech Processing

Yen-Ju Lu[1*], **Xuankai Chang**[2*], **Chenda Li**[3], **Wangyou Zhang**[3], **Samuele Cornell**[2,4], **Zhaoheng Ni**[5], **Yoshiki Masuyama**[2,6], **Brian Yan**[2], **Robin Scheibler**[7], **Zhong-Qiu Wang**[2], **Yu Tsao**[8], **Yanmin Qian**[3], and **Shinji Watanabe**[2¶]

1 Johns Hopkins University, USA **2** Carnegie Mellon University, USA **3** Shanghai Jiao Tong University, Shanghai **4** Universita' Politecnica delle Marche, Italy **5** Meta AI, USA **6** Tokyo Metropolitan University, Japan **7** LINE Corporation, Japan **8** Academia Sinica, Taipei ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.05403

Software

- Review
- Repository
- Archive

Editor: Fabian-Robert Stöter

Reviewers:

- @joimort
- @justusschock

Submitted: 07 September 2022
Published: 20 November 2023

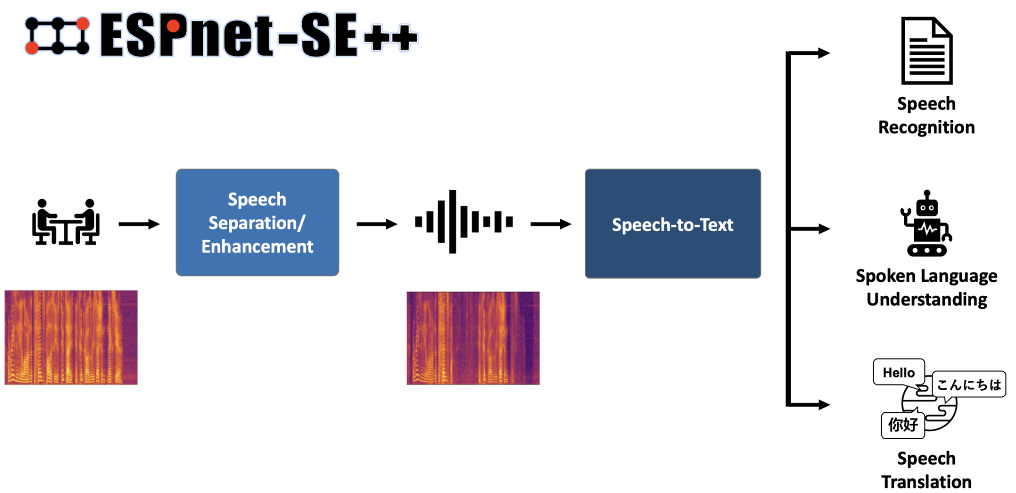

Figure 1: The Joint-task Systems of SSE with ASR, ST, and SLU in ESPnet-SE++.

Summary

This paper presents the software design and user interface of ESPnet-SE++, a new speech separation and enhancement (SSE) module of the ESPnet toolkit. ESPnet-SE++ significantly expands the functionality of ESPnet-SE (Li et al., 2021) with several new models(Chen et al., 2017; Dang et al., 2022; Hershey et al., 2016; Hu et al., 2020; Li et al., 2022; Lu, Cornell, et al., 2022; Luo et al., 2019; Takahashi et al., 2019; Tan et al., 2021), loss functions (Boeddeker et al., 2021; Le Roux et al., 2019; Luo & Mesgarani, 2018; Scheibler, 2022), and training recipes as shown in (Lu, Chang, et al., 2022). Crucially, it features a new, redesigned interface, which allows for a flexible combination of SSE front-ends with many downstream tasks, including automatic speech recognition (ASR), speaker diarization (SD), speech translation (ST), and spoken language understanding (SLU).

Statement of need

ESPnet is an open-source toolkit for speech processing, including several ASR, text-to-speech (TTS) (Hayashi et al., 2020), ST (Inaguma et al., 2020), machine translation (MT), SLU (Arora et al., 2022), and SSE recipes (Watanabe et al., 2018). Compared with other open-source SSE toolkits, such as Nussl (Manilow et al., 2018), Onssen (Ni, 2019), Asteroid (Pariente et al., 2020), and SpeechBrain (Ravanelli et al., 2021), the modularized design in ESPnet-SE++ allows for the joint training of SSE modules with other tasks. Currently, ESPnet-SE++ supports 20 SSE recipes with 24 different enhancement/separation models.

ESPnet-SE++ Recipes and Software Structure

ESPNet-SE++ Recipes for SSE and Joint-Task

For each task, ESPnet-SE++, following the ESPnet2 style, provides common scripts which are carefully designed to work out-of-the-box with a wide variety of corpora. The recipes for different corpora are under the `egs2/` folder. Under the `egs2/TEMPLATE` folder, the common scripts `enh1/enh.sh` and `enh_asr1/enh_asr.sh` are shared for all the SSE and joint-task recipes. The directory structure can be found in TEMPLATE/enh_asr1/README.md.

Common Scripts

`enh.sh` contains 13 stages, and the details for the scripts can be found in TEMPLATE/enh1/README.md.

```
- stage 1 to stage 4: data preparation stages
    - stage 1: Call the local/data.sh script from the recipe.
    - stage 2: Optional offline augmentation of input dataset
    - stage 3: Create a temporary data dump folder, segment audio files.
    - stage 4: Possibly remove too short and too long utterances

- stage 5 to stage 6: SSE training steps
    - stage 5: Collect dataset statistics for dataloading or for normalization
    - stage 6: SSE task training

- stage 7 to stage 8: Evaluation stages for speech enhancement.
    - stage 7: Evaluation stages: inferencing and storing the enhanced audios
    - stage 8: Scoring

- stage 9 to stage 10: Evaluation stages for speech recognition or understanding.
    - stage 9: Decoding with a pretrained ASR/SLU model
    - stage 10: Scoring with a pretrained ASR model

- stage 11 to stage 13: model uploading steps
```

`enh_asr.sh` contains 17 stages, and the details for the scripts can be found in TEMPLATE/enh_asr1/README.md. The `enh_diar.sh` and `enh_st.sh` are similar to it.

```
- stage 1 to stage 5: data preparation stages

- stage 6 to stage 9: language model training steps

- stage 10 to stage 11: joint-task training steps

- stage 12 to stage 13: Inference stages: Decoding and enhancing

- stage 14 to stage 15: Scoring recognition and SSE results

- stage 16 to stage 17: model uploading steps
```

Training Configuration

SSE Task Training Configuration

An example of an enhancement task for the CHiME-4 `enh1` recipe is configured as `conf/tuning/train_enh_dprnn_tasnet.yaml`. This file includes the specific types of `encoder`, `decoder`, `separator`, and their respective settings. Furthermore, the file also defines the training setup and `criterions`.

Joint-Task Training Configuration

An example of joint-task training configuration is the CHiME-4 `enh_asr1` recipe, configured as `conf/tuning/train_enh_asr_convtasnet.yaml`. This joint-task comprises of a front-end SSE model and a back-end ASR model. The configuration file includes specifications for the `encoder`, `decoder`, `separator`, and `criterions` of both the SSE and ASR models,using prefixes such as `enh_` and `asr_`.

ESPNet-SE++ Software Structure for SSE Task

The directory structure for the SSE python files can be found in TEMPLATE/enh1/README.md. Additionally, the UML diagram for the enhancement-only task in ESPNet-SE++ is provided below.

The Journal of Open Source Software

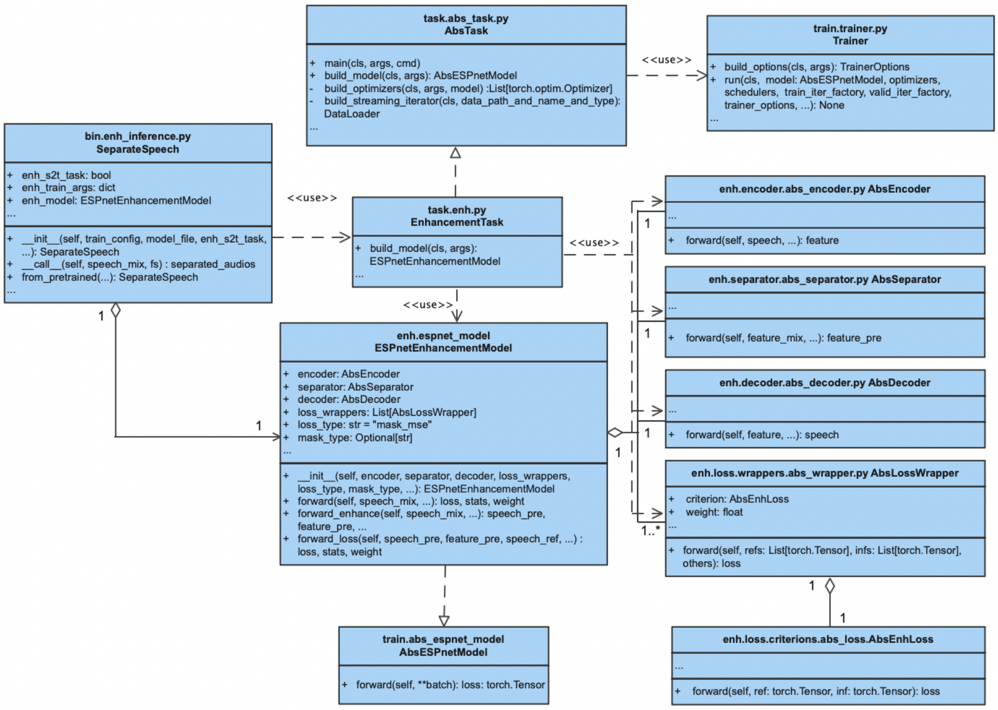

Figure 2: UML Diagram for Speech Separation and Enhancement in ESPnet-SE++

SSE Executable Code `bin/*`

bin/enh_train.py

As the main interface for the SSE training stage of `enh.sh`, `enh_train.py` takes the training parameters and model configurations from the arguments and calls

```
EnhancementTask.main(...)
```

to build an SSE object `ESPnetEnhancementModel` for training the SSE model according to the model configuration.

bin/enh_inference.py

The `inference` function in `enh_inference.py` creates a

```
class SeparateSpeech
```

object with the data-iterator for testing and validation. During its initialization, this class instantiate an SSE object `ESPnetEnhancementModel` based on a pair of configuration and a pre-trained SSE model.

bin/enh_scoring.py

```
def scoring(..., ref_scp, inf_scp, ...)
```

The SSE scoring functions calculates several popular objective scores such as SI-SDR (Le Roux et al., 2019), STOI (Taal et al., 2011), SDR and PESQ (Rix et al., 2001), based on the reference signal and processed speech pairs.

SSE Control Class **tasks/enh.py**

```
class EnhancementTask(AbsTask)
```

`EnhancementTask` is a control class which is designed for SSE tasks. It contains class methods for building and training an SSE model. Class method `build_model` creates and returns an SSE object `ESPnetEnhancementModel`.

SSE Modules **enh/espnet_model.py**

```
class ESPnetEnhancementModel(AbsESPnetModel)
```

`ESPnetEnhancementModel` is the base class for any ESPnet-SE++ SSE model. Since it inherits the same abstract base class `AbsESPnetModel`, it is well-aligned with other tasks such as ASR, TTS, ST, and SLU, bringing the benefits of cross-tasks combination.

```
def  forward(self, speech_mix, speech_ref, ...)
```

The `forward` function of `ESPnetEnhancementModel` follows the general design in the ESPnet single-task modules, which processes speech and only returns losses for Trainer to update the model.

```
def  forward_enhance(self, speech_mix, ...)
def  forward_loss(self, speech_pre, speech_ref, ...)
```

For more flexible combinations, the `forward_enhance` function returns the enhanced speech, and the `forward_loss` function returns the loss. The joint-training methods take the enhanced speech as the input for the downstream task and the SSE loss as a part of the joint-training loss.

ESPNet-SE++ Software Structure for Joint-Task

The directory structure for the SSE python files can be found in TEMPLATE/enh_asr1/README.md. Furthermore, the UML diagram for the joint-task in ESPNet-SE++ is displayed below.

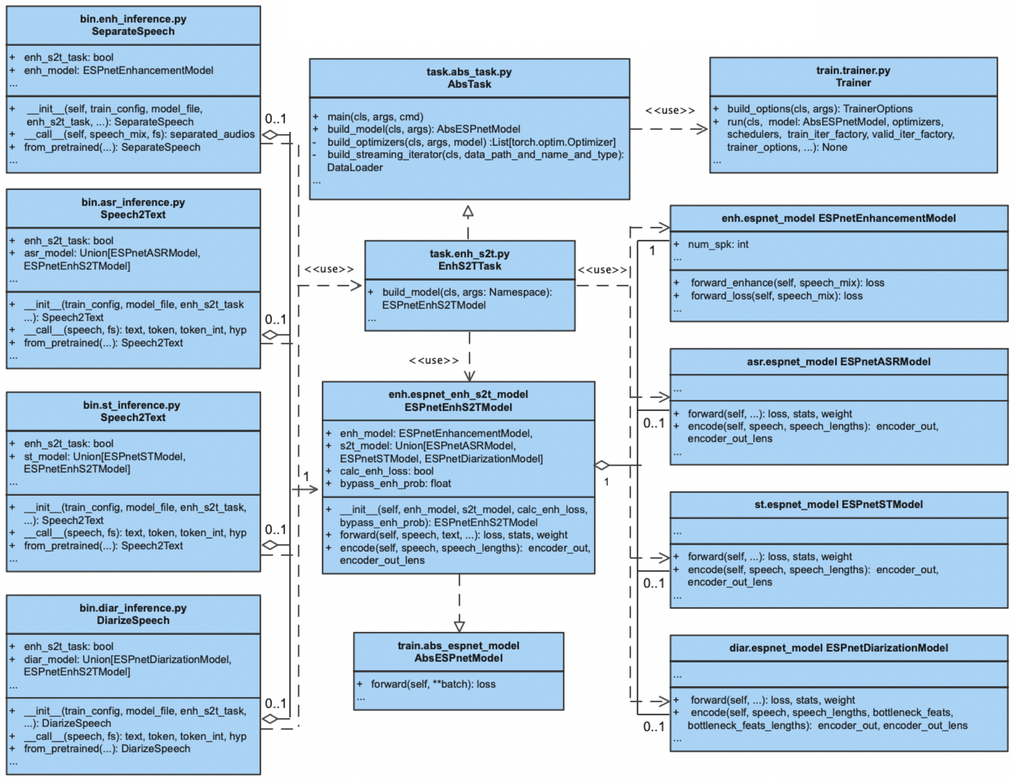

Figure 3: UML Diagram for Joint-Task in ESPnet-SE++

Joint-Task Executable Code `bin/*`

bin/enh_s2t_train.py

Similarly to the interface of SSE training code `enh_train.py`, `enh_s2t_train.py` takes the training and modular parameters from the scripts, and calls

```
tasks.enh_s2t.EnhS2TTask.main(...)
```

to build a joint-task object for training the joint-model based on a configuration with both SSE and s2t models setting with or without pre-trained checkpoints.

bin/asr_inference.py, bin/diar_inference.py, and bin/st_inference.py

The `inference` function in `asr_inference.py`, `diar_inference.py`, and `st_inference.py` builds and call a

```
class Speech2Text
class DiarizeSpeech
```

object with the data-iterator for testing and validation. During their initialization, the classes build a joint-task object `ESPnetEnhS2TModel` with pre-trained joint-task models and configurations.

Joint-task Control Class `tasks/enh_s2t.py`

```
class EnhS2TTask(AbsTask)
```

`class EnhS2TTask` is designed for joint-task model. The subtask models are created and sent into the `ESPnetEnhS2TModel` to create a joint-task object.

Joint-Task Modules `enh/espnet_enh_s2t_model.py`

```
class ESPnetEnhS2TModel(AbsESPnetModel)
```

The `ESPnetEnhS2TModel` takes a front-end `enh_model`, and a back-end `s2t_model` (such as ASR, SLU, ST, and SD models) as inputs to build a joint-model.

```
def __init__(
    self,
    enh_model: ESPnetEnhancementModel,
    s2t_model: Union[ESPnetASRModel, ESPnetSTModel, ESPnetDiarizationModel],
    ...
):
```

The `forward` function of the class follows the general design in ESPnet2:

```
def  forward(self, speech_mix, speech_ref, ...)
```

which processes speech and only returns losses for Trainer to update the model.

ESPnet-SE++ User Interface

Building a New Recipe from Scratch

Since ESPnet2 provides common scripts such as `enh.sh` and `enh_asr.sh` for each task, users only need to create `local/data.sh` for the data preparation of a new corpus. The generated data follows the Kaldi-style structure (Povey et al., 2011):

```
data/
  train/
    - text       # The transcription for each utterance.
    - spk1.scp   # Wave file path to the clean utterances.
    - noise1.scp # [Optional] Wave file path to the noise references.
    - wav.scp    # Wave file path to the noisy utterances.
    - utt2spk    # Mapping utterance-id to speaker-id.
    - spk2utt    # Mapping speaker-id to utterance-id.
    - segments   # [Optional] Specifying the start and end time of each utterance.
  dev/
    ...
  test/
    ...
```

The detailed instructions for data preparation and building new recipes in espnet2 are described in the link.

Inference with Pre-trained Models

Pretrained models from ESPnet are provided on HuggingFace and Zenodo. Users can download and infer with the models.`model_name` in the following section should be `huggingface_id` or one of the tags in the table.csv in espnet_model_zoo . Users can also directly provide a Zenodo URL or a HuggingFace URL.

Inference API

The inference functions are from the `enh_inference` and `enh_asr_inference` in the executable code `bin/`

```
from espnet2.bin.enh_inference import SeparateSpeech
from espnet2.bin.enh_asr_inference import Speech2Text
```

Calling `SeparateSpeech` and `Speech2Text` with unprocessed audios returns the separated speech and their recognition results.

SSE

```
import soundfile
from espnet2.bin.enh_inference import SeparateSpeech
separate_speech = SeparateSpeech.from_pretrained(
    "model_name",
    # load model from enh model or enh_s2t model
    enh_s2t_task=True,
    # for segment-wise process on long speech
    segment_size=2.4,
    hop_size=0.8,
    normalize_segment_scale=False,
    show_progressbar=True,
    ref_channel=None,
    normalize_output_wav=True,
)
# Confirm the sampling rate is equal to that of the training corpus.
# If not, you need to resample the audio data before inputting to speech2text
speech, rate = soundfile.read("long_speech.wav")
waves = separate_speech(speech[None, ...], fs=rate)
```

Joint-Task

```
import soundfile
from espnet2.bin.asr_inference import Speech2Text
speech2text = Speech2Text.from_pretrained(
    "model_name",
    # load model from enh_s2t model
    enh_s2t_task=True,
    # Decoding parameters are not included in the model file
    maxlenratio=0.0,
    minlenratio=0.0,
    beam_size=20,
    ctc_weight=0.3,
    lm_weight=0.5,
    penalty=0.0,
    nbest=1
)
# Confirm the sampling rate is equal to that of the training corpus.
# If not, you need to resample the audio data before inputting to speech2text
speech, rate = soundfile.read("speech.wav")
nbests, waves = speech2text(speech)
text, *_ = nbests[0]
```

The details for downloading models and inference are described in espnet_model_zoo.

Demonstrations

The demonstrations of ESPnet-SE can be found in the following google colab links:

- ESPnet SSE Demonstration: CHiME-4 and WSJ0-2mix
- ESPnet-SE++ Joint-Task Demonstration: L3DAS22 Challenge and SLURP-Spatialized

Development plan

The development plan of the ESPnet-SE++ can be found in Development plan for ESPnet2 speech enhancement. In addition, we will explore the combinations with other front-end tasks, such as using ASR as a front-end model and TTS as a back-end model for speech-to-speech conversion.

Conclusions

In this paper, we introduce the software structure and the user interface of ESPnet-SE++, including the SSE task and joint-task models. ESPnet-SE++ provides general recipes for training models on different corpus and a simple way for adding new recipes. The joint-task implementation further shows that the modularized design improves the flexibility of ESPnet.

Acknowledgement

This work used the Extreme Science and Engineering Discovery Environment (XSEDE) (Towns et al., 2014), which is supported by NSF grant number ACI-1548562. Specifically, it used the Bridges system (Nystrom et al., 2015), which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC).

References

Arora, S., Dalmia, S., Denisov, P., Chang, X., Ueda, Y., Peng, Y., Zhang, Y., Kumar, S., Ganesan, K., & Yan, W., B. (2022). ESPnet-SLU: Advancing spoken language understanding through ESPnet. *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 7167–7171. https://doi.org/10.1109/icassp43922.2022.9747674

Boeddeker, C., Zhang, W., Nakatani, T., Kinoshita, K., Ochiai, T., Delcroix, M., Kamo, N., Qian, Y., & Haeb-Umbach, R. (2021). Convolutive transfer function invariant SDR training criteria for multi-channel reverberant speech separation. *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8428–8432. https://doi.org/10.1109/icassp39728.2021.9414661

Chen, Z., Luo, Y., & Mesgarani, N. (2017). Deep attractor network for single-microphone speaker separation. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 246–250. https://doi.org/10.1109/icassp.2017.7952155

Dang, F., Chen, H., & Zhang, P. (2022). DPT-FSNet: Dual-path transformer based full-band and sub-band fusion network for speech enhancement. *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6857–6861. https://doi.org/10.1109/icassp43922.2022.9746171

Hayashi, T., Yamamoto, R., Inoue, K., Yoshimura, T., Watanabe, S., Toda, T., Takeda, K., & Zhang, X., Y. Tan. (2020). ESPnet-TTS: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit. *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 7654–7658. https://doi.org/10.1109/icassp40776.2020.9053512

Hershey, J. R., Chen, Z., Le Roux, J., & Watanabe, S. (2016). Deep clustering: Discriminative embeddings for segmentation and separation. *2016 IEEE International Conference on*

Acoustics, Speech and Signal Processing (ICASSP), 31–35. https://doi.org/10.1109/icassp.2016.7471631

Hu, Y., Liu, Y., Lv, S., Xing, M., Zhang, S., Fu, Y., Wu, J., Zhang, B., & Xie, L. (2020). DCCRN: Deep complex convolution recurrent network for phase-aware speech enhancement. *Proceedings of Interspeech*, 2472–2476. https://doi.org/10.21437/interspeech.2020-2537

Inaguma, H., Kiyono, S., Duh, K., Karita, S., Soplin, N. E. Y., Hayashi, T., & Watanabe, S. (2020). ESPnet-ST: All-in-one speech translation toolkit. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 302–311. https://doi.org/10.18653/v1/2020.acl-demos.34

Le Roux, J., Wisdom, S., Erdogan, H., & Hershey, J. R. (2019). SDR – half-baked or well done? *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 626–630. https://doi.org/10.1109/icassp.2019.8683855

Li, C., Shi, J., Zhang, W., Subramanian, A. S., Chang, X., Kamo, N., Hira, M., Hayashi, T., Boeddeker, C., & Chen, S., Z. Watanabe. (2021). ESPnet-SE: End-to-end speech enhancement and separation toolkit designed for ASR integration. *2021 IEEE Spoken Language Technology Workshop (SLT)*, 785–792. https://doi.org/10.1109/slt48900.2021.9383615

Li, C., Yang, L., Wang, W., & Qian, Y. (2022). SkiM: Skipping memory lstm for low-latency real-time continuous speech separation. *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 681–685. https://doi.org/10.1109/icassp43922.2022.9746372

Lu, Y. J., Chang, X., Li, C., Zhang, W., Cornell, S., Ni, Z., Masuyama, Y., Yan, B., Scheibler, R., Wang, Z. Q., Tsao, Y., & Qian Y. Watanabe, S. (2022). ESPnet-SE++: Speech enhancement for robust speech recognition, translation, and understanding. *Proceedings of Interspeech*, 5458–5462. https://doi.org/10.21437/interspeech.2022-10727

Lu, Y. J., Cornell, S., Chang, X., Zhang, W., Li, C., Ni, Z., Wang, Z., & Watanabe, S. (2022). Towards low-distortion multi-channel speech enhancement: The ESPNET-se submission to the L3DAS22 challenge. *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 9201–9205. https://doi.org/10.1109/icassp43922.2022.9747146

Luo, Y., Han, C., Mesgarani, N., Ceolini, E., & Liu, S. (2019). FaSNet: Low-latency adaptive beamforming for multi-microphone audio processing. *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 260–267. https://doi.org/10.1109/asru46091.2019.9003849

Luo, Y., & Mesgarani, N. (2018). TaSNet: Time-domain audio separation network for real-time, single-channel speech separation. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 696–700. https://doi.org/10.1109/icassp.2018.8462116

Manilow, E., Seetharaman, P., & Pardo, B. (2018). The northwestern university source separation library. *International Society for Music Information Retrieval (ISMIR)*, 297–305. https://doi.org/10.1163/1872-9037_afco_asc_1322

Ni, M. I., Zhaoheng Mandel. (2019). ONSSEN: An open-source speech separation and enhancement library. *arXiv Preprint arXiv:1911.00982*.

Nystrom, N. A., Levine, M. J., Roskies, R. Z., & Scott, J. R. (2015). Bridges: A uniquely flexible HPC resource for new communities and data analytics. *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, 1–8. https://doi.org/10.1145/2792745.2792775

Pariente, M., Cornell, S., Cosentino, J., Sivasankaran, S., Tzinis, E., Heitkaemper, J., Olvera, M., Stöter, F. R., Hu, M., Martín-Doñas, J. M., Ditter, D., Frank, A., Deleforge, A.,

& Vincent, E. (2020). Asteroid: The PyTorch-based audio source separation toolkit for researchers. *Proceedings of Interspeech*, 2637–2641. https://doi.org/10.21437/interspeech.2020-1673

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky´, J., & Stemmer, K., G. Vesely. (2011). The Kaldi speech recognition toolkit. *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. https://doi.org/10.15199/48.2016.11.70

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

Rix, A. W., Beerends, J. G., Hollier, M. P., & Hekstra, A. P. (2001). Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, *2*, 749–752. https://doi.org/10.1109/icassp.2001.941023

Scheibler, R. (2022). SDR — medium rare with fast computations. *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 701–705. https://doi.org/10.1109/icassp43922.2022.9747473

Taal, C. H., Hendriks, R. C., Heusdens, R., & Jensen, J. (2011). An algorithm for intelligibility prediction of time–frequency weighted noisy speech. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(7), 2125–2136. https://doi.org/10.1109/tasl.2011.2114881

Takahashi, N., Parthasaarathy, S., Goswami, N., & Mitsufuji, Y. (2019). Recursive speech separation for unknown number of speakers. *Interspeech 2019*, 1348–1352. https://doi.org/10.21437/interspeech.2019-1550

Tan, K., Zhang, X., & Wang, D. (2021). Deep learning based real-time speech enhancement for dual-microphone mobile phones. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *29*, 1853–1863. https://doi.org/10.1109/taslp.2021.3082318

Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., Roskies, R., Scott, J. R., & Wilkins-Diehr, N. (2014). XSEDE: Accelerating scientific discovery. *Computing in Science & Engineering*, *16*(5), 62–74. https://doi.org/10.1109/mcse.2014.80

Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., Soplin, N. E. Y., Heymann, J., Wiesner, M., Chen, N., Renduchintala, A., & Ochiai, T. (2018). ESPnet: End-to-end speech processing toolkit. *Proceedings of Interspeech*, 2207–2211. https://doi.org/10.21437/interspeech.2018-1456