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

Yen-Ju Lu, Xuankai Chang, Chenda Li, Wangyou Zhang, Samuele Cornell, Zhaoeng Ni, Yoshiki Masuyama, Brian Yan, Robin Scheibler, Zhong-Qiu Wang, Yu Tsao, Yanmin Qian, and Shinji Watanabe

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

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

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 eg2/ folder. Under the eg2/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_dtar.sh and enh_st.sh are similar to it.
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.
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

```python
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

```python
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.
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

```python
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

```python
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=8.0,
    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


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


