SpeechBrain: A General-Purpose Speech Toolkit

Preprint · June 2021

CITATIONS
0

READS
602

21 authors, including:

Mirco Ravanelli
Concordia University Montreal
90 PUBLICATIONS **2,295** CITATIONS
SEE PROFILE

Titouan Parcollet
Université d´Avignon et des Pays du Vaucluse
58 PUBLICATIONS **494** CITATIONS
SEE PROFILE

Loren Lugosch
McGill University
15 PUBLICATIONS **608** CITATIONS
SEE PROFILE

Cem Subakan
University of Illinois, Urbana-Champaign
32 PUBLICATIONS **218** CITATIONS
SEE PROFILE

SpeechBrain: A General-Purpose Speech Toolkit

Mirco Ravanelli[1,2], **Titouan Parcollet**[3,16], **Peter Plantinga**[4], **Aku Rouhe**[5], **Samuele Cornell**[6], **Loren Lugosch**[1,7], **Cem Subakan**[1], **Nauman Dawalatabad**[8], **Abdelwahab Heba**[9], **Jianyuan Zhong**[1], **Ju-Chieh Chou**[10*], **Sung-Lin Yeh**[11*], **Szu-Wei Fu**[12], **Chien-Feng Liao**[12], **Elena Rastorgueva**[13†], **François Grondin**[14], **William Aris**[14], **Hwidong Na**[15], **Yan Gao**[16], **Renato De Mori**[3,7], and **Yoshua Bengio**[1,2]

[1]Mila - Quebec AI Institute
[2]Université de Montréal
[3]LIA - Avignon Université
[4]Ohio State University
[5]Aalto University
[6]Università Politecnica delle Marche
[7]McGill University
[8]Indian Institute of Technology Madras
[9]IRIT - Université Paul Sabatier
[10]Toyota Technological Institute at Chicago
[11]University of Edinburgh
[12]Academia Sinica, Taiwan
[13]NVIDIA
[14]Université de Sherbrooke
[15]Samsung-SAIT
[16]CaMLSys - University of Cambridge

Abstract

SpeechBrain is an open-source and all-in-one speech toolkit. It is designed to facilitate the research and development of neural speech processing technologies by being simple, flexible, user-friendly, and well-documented. This paper describes the core architecture designed to support several tasks of common interest, allowing users to naturally conceive, compare and share novel speech processing pipelines. SpeechBrain achieves competitive or state-of-the-art performance in a wide range of speech benchmarks. It also provides training recipes, pretrained models, and inference scripts for popular speech datasets, as well as tutorials which allow anyone with basic Python proficiency to familiarize themselves with speech technologies.

1 Introduction

Open-source toolkits have played a critical role in the development of speech processing technology [1–5]. Kaldi [5], for instance, is an established speech recognition framework, which is implemented in C++ with recipes built on top of Bash, Perl, and Python scripts. Despite being efficient, its use of C++ can make prototyping of new deep learning methods difficult. With the advent of general-purpose deep learning libraries like TensorFlow [6] and PyTorch [7], more flexible speech recognition frameworks have quickly appeared, e.g., DeepSpeech [8], RETURNN [9], PyTorch-Kaldi [10], Espresso [11], Lingvo [12], Fairseq [13], ESPnet [14], and NeMo [15].

*Work conducted while at National Taiwan University.
†Work conducted while on an internship at Mila - Quebec AI Institute.

Preprint. Under review.

Table 1: List of speech tasks and corpora that are currently supported by SpeechBrain.

Task	Description	Techniques	Datasets
Speech recognition	*Speech-to-text.*	CTC [24] Transducers [25] CTC+Attention [26] Shallow fusion [27]	LibriSpeech [28] Common Voice [29] AISHELL [30] TIMIT [31]
Speaker recognition	*Speaker verification/ID.*	X-vectors [32] ECAPA-TDNN [33]	VoxCeleb1 [34] VoxCeleb2 [35]
Speaker diarization	*Detect who spoke when.*	Spectral Clustering [36] Neural embeddings [37]	AMI corpus [38]
Speech enhancement	*Noisy to clean speech.*	MetricGAN+ [39] Mimic Loss [40]	VoiceBank [41] DNS [42]
Speech separation	*Separate overlapped speech.*	ConvTasNet[43] DualPath RNNs [44] SepFormer [45]	WSJ-mix [46] WHAM [47] WHAMR [48] LibriMix [49]
Spoken language understanding	*Speech to intent/slots.*	Decoupled [50] Multistage [51] Direct [52]	TAS [50] SLURP [53] FSC [54]
Multi-microphone processing	*Combining input signals.*	Delay-and-sum MVDR [55] GEV [56] GCC-PHAT [57] SRP-PHAT [58] MUSIC [59]	Dataset-Independent

Recently, task-specific libraries have also been released. Examples are Asteroid [16] for speech separation, pyannote [17] and sidekit [18] for speaker diarization, and s3prl [19] for self-supervised speech representations. While excelling at specific tasks, these frameworks have different coding styles, standards, and programming languages, making it challenging and time-consuming to migrate from one codebase to another. Moreover, their combination in complex speech processing pipelines poses a challenge for interoperability, as connecting different frameworks might be unnatural and their codebases can interact in unpredictable ways.

Our experience suggests that having a *single*, *flexible*, *multi-task* toolkit can significantly speed up the development of speech technologies. Due to growing interest in end-to-end spoken dialog systems (e.g., virtual assistants), implementing composite pipelines within an integrated toolkit offers many advantages. A single toolkit, for instance, encourages the exploration of transfer learning and joint training techniques across different tasks [20–23] and enables the creation of fully differentiable graphs where multiple technologies are trained jointly and learn to interact.

Inspired by this vision, we have developed SpeechBrain[3], an all-in-one PyTorch-based toolkit designed to facilitate the development, portability, and ease of use of speech processing technologies. The name *SpeechBrain* highlights the need for a holistic system capable of performing multiple tasks at once, for example, recognize speech, understanding its content, language, emotions, and speakers. Our toolkit is not only intended for speech researchers, but also for the broader machine learning community, enabling users to easily integrate their models into different speech pipelines and compare them with state-of-the-art (SotA) baselines. Our main contributions in this paper are:

- The presentation of *SpeechBrain*, with an emphasis on how we designed it to support multiple tasks without sacrificing simplicity, modularity, or flexibility.
- The implementation and experimental validation of both recent and long-established speech processing models with SotA or competitive performance on a variety of tasks (cf. Table 1).

More broadly, we believe the SpeechBrain toolkit has the potential to significantly accelerate research and innovation in the field of speech processing and deep learning.

[3]The toolkit website can be found at `speechbrain.github.io/`.

2 Related Work

A few other toolkits support multiple speech tasks. Of these, the ones we consider most related to SpeechBrain are Fairseq [13], NeMo [15], and ESPnet [14]. Fairseq is developed by Facebook to support sequence-to-sequence processing. It includes models such as ConvS2S [60], transformers [61], and wav2vec [62]. However, speech processing encompasses several paradigms outside of sequence-to-sequence modeling. SpeechBrain also supports regression tasks (e.g., speech enhancement, separation), classification tasks (e.g., speaker recognition), clustering (e.g., diarization), and even signal processing techniques (e.g., multi-microphone combination).

NeMo is a toolkit for conversational AI developed by NVIDIA, which provides useful neural modules for many speech processing tasks, including speech recognition, speaker diarization, voice-activity detection and text-to-speech. Due to its industrial orientation, NeMo offers efficient ready-to-use models, such as Jasper [63], QuartzNet [64], and Citrinet [65]. SpeechBrain also provides several ready-to-use models, but focuses more heavily on research and education by providing a wide variety of baselines, models, and recipes that users can easily inspect and modify in the experiments.

ESPnet, in its current form, is the closest toolkit to SpeechBrain. Both are academically driven and support numerous speech tasks. ESPnet started as an end-to-end speech recognition library and progressively grew to support different tasks. By contrast, we designed SpeechBrain to address a wide variety of tasks from the outset. This means that combining technologies and developing recipes for new tasks is extremely simple.

3 Design Principles

Beyond the multi-task vision highlighted in the introduction, we developed SpeechBrain with the following design principles in mind:

Accessibility: SpeechBrain is designed to be easily understandable by a large user base, including early students and practitioners. Therefore, we devoted considerable effort to develop intuitive modules that are easy to interconnect with each other. One remarkable peculiarity of SpeechBrain is that it serves educational purposes as well. We thus have written extensive documentation and tutorials with Google Colab to help newcomers become more familiar with speech technologies. Prior work has shown code snippets aid in adopting a codebase [66]. Motivated by this, SpeechBrain provides runnable code snippets in docstrings (documenting interaction at the granular level), tutorial notebooks (explaining single topics), and template files (describing full experiments on different tasks). To make our toolkit as accessible as possible, we have released it under a very permissive license (Apache 2.0).

Ease of use: SpeechBrain employs a simple software stack (i.e., Python $\rightarrow$ PyTorch $\rightarrow$ SpeechBrain) to avoid dealing with too many levels of abstractions. It is developed on top of PyTorch directly, without an external API. PyTorch-compatible code works in our toolkit without any further modification. SpeechBrain has a minimal list of external dependencies that are all installable via PyPI. The installation process simply requires running the command `pip install speechbrain` and is done within a few minutes. The code is Pythonic and maximizes the use of PyTorch routines.

Replicability: SpeechBrain promotes open and transparent science. We trained most of our models with publicly available data. This way, our results can be easily replicated by the community. Several pre-trained models, which only require a few lines of code to use, are distributed via Hugging Face [67]. Besides sharing the code and the trained models, we also share the whole experiment folder, which contains all the needed details (e.g., logs) to reproduce our results.

4 Architecture

From an architectural standpoint, SpeechBrain sits in between a library and a framework. Where libraries require users to manage dataflow by calling library-defined functionality, frameworks primarily define a custom lifecycle in which user-defined functionalities are invoked in specific places (*inversion of control*). Most code in SpeechBrain follows a library-style collection of modular and standalone building blocks, including practical routines for data loading, decoding, signal processing, and other convenient utilities. However the central `Brain` class (see § 4.4), uses inversion of control

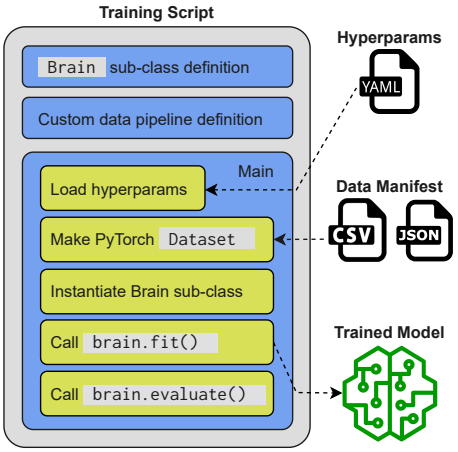

Figure 1: An overview of a basic training script.

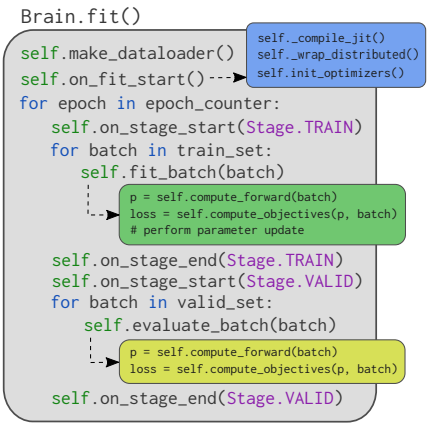

Figure 2: Illustration of `Brain.fit()`.

to define a general training loop. Therefore, SpeechBrain is most accurately described as a *toolkit*. As shown in Figure 1, the code for training a model is contained within a single Python script. Training begins by calling the script with a set of hyperparameters: `python train.py hparams.yaml`. These hyperparameters, declared in human-readable YAML format, contain the location of one or more data manifest files using either CSV or JSON formats (see Appendix A.4). Unlike many other toolkits, SpeechBrain orchestrates experiments in Python directly, without relying on external Bash scripts. This allows code for data loading, modeling, optimization, and evaluation to interact naturally. Moreover, the training script exposes the computations likely to be changed most frequently (e.g., forward computations, data transformations, etc.), making them easy to access and modify. SpeechBrain treats the user's code as a first-class citizen: all PyTorch-compatible code written by the user is treated the same as SpeechBrain code. In the following sub-sections, we explore the anatomy of a training script in more detail.

4.1 Hyperparameters

The model hyperparameters, in conjunction with the training script, regulate various properties of the pipeline such as model architecture, training, and decoding. SpeechBrain relies on an extended version of YAML called *HyperPyYAML*, as shown in the following excerpt:

```
dropout: 0.2
features: !new:speechbrain.lobes.features.MFCC
    n_mels: 40
    left_frames: 5
    right_frames: 5

model: !new:torch.nn.LSTM
   input_size: 440
   hidden_size: 256
   num_layers: 4
   dropout:  !ref <dropout>
   bidirectional: True
```

Listing 1: An excerpt of a YAML file for hyperparameter specification.

HyperPyYAML is not just an ordinary list of hyperparameters, but allows a complex hyperparameter specification that defines objects along with their corresponding arguments. There is always an explicit reference between the hyperparameter declarations and any object using them, making the code more interpretable and simpler to debug. Overriding the contents of the YAML file (e.g., for hyperparameter search) can also be done easily by passing command-line arguments:

```
$ python train.py hparams.yaml --learning_rate=0.1 --dropout=0.5
```

SpeechBrain initializes the classes automatically when reading the YAML file, thus eliminating boilerplate initialization code from the training script. HyperPyYAML is a general tool for specifying hyperparameters. To enable modular reusuability, we have released it as a separate repository on PyPI[4].

4.2 Data loading

SpeechBrain complements standard PyTorch data loading by addressing the typical challenges that occur when working with speech, such as handling variable-length sequences, large datasets, and complex data transformation pipelines. Our `DynamicItemDataset` inherits from `torch.utils.data.Dataset` and creates a dataset-interface based on a data-manifest file. The data-manifest file contains *static items*, such as filepaths or speaker labels. Then, *dynamic items* provide transformations based on the existing items (static or dynamic), as shown in the following example:

```
@speechbrain.utils.data_pipeline.takes("file_path")
@speechbrain.utils.data_pipeline.provides("signal")
def audio_pipeline(file_path):
  return speechbrain.dataio.read_audio(file_path)
```

Listing 2: An example of a custom data pipeline.

This function takes an audio file path (a static item) and reads it as a tensor called "signal" (a dynamic item). Any library for reading audio file can be used here, including torch.audio[5]. The evaluation order of the items is determined by a dependency graph. Users can define operations such as reading and augmenting an audio file, encoding a text label into an integer, basic text processing, etc. The dynamic items are defined in the training script and are thus directly customizable by the users. Moreover, by leveraging the PyTorch `DataLoader` class, these data pipelines are automatically applied in parallel across different workers.

4.3 Batching

Speech sequences for a given dataset typically vary in length and require zero-padding to create equal-length batches. This tends to add some complication during the training process. First, the length of each sentence within each batch must be tracked so we can later remove zero-padded elements from computations like normalization, statistical pooling, losses, etc. Another issue that arises is how to avoid wasting computational resources processing zero-padded elements.

One approach to mitigate this issue is to sort data by sequence length before batching, which minimizes zero-padding but sacrifices randomness in the batch creation process. A more sophisticated approach is to apply dynamic batching [68, 69], where sentences are clustered by length and sampled within the same cluster, a trade-off between random and sorted batching. This allows the batch size to be dynamically changed according to sentence length, leading to improved efficiency and better management of available GPU memory. All the aforementioned batching strategies are supported by SpeechBrain, allowing users to choose the approach that meets their specific needs.

4.4 The `Brain` class

SpeechBrain implements a general training loop in the `Brain` class. The `Brain.fit()` method is inspired by similar methods in libraries such as Scikit-learn [70], Scipy [71], Keras [72], fastai [73], and PyTorch Lightning [74]. Figure 2 illustrates the basic components of the `Brain.fit()` method. The following is a simple demonstration:

```
import torch, speechbrain

class SimpleBrain(speechbrain.Brain):
  def compute_forward(self, batch, stage):
    return self.modules.model(batch["input"])
```

[4] github.com/speechbrain/HyperPyYAML
[5] https://github.com/pytorch/audio

Table 2: Phoneme Error Rate (PER%) achieved with SpeechBrain on TIMIT using different speech recognizers.

Technique	# Params	Dev	Test
CTC	10 M	12.34	14.15
Transducer	10 M	12.66	14.12
CTC+Att	10 M	12.74	13.83
CTC+Att+SSL	318 M	**7.11**	**8.04**

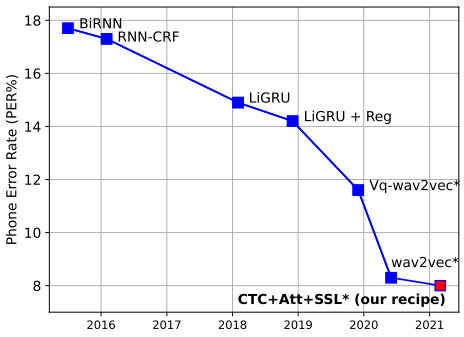

Figure 3: Evolution of the SotA performance for TIMIT. Entries marked with * use extra unlabelled data from the Libri-Light dataset. Source: `https://paperswithcode.com`.

```
  def compute_objectives(self, predictions, batch, stage):
    return torch.nn.functional.l1_loss(predictions, batch["target"])

modules = {"model": torch.nn.Linear(in_features=10, out_features=10)}
brain = SimpleBrain(modules, lambda x: torch.optim.SGD(x, 0.1))
data = [{"input": rand(10, 10), "target": rand(10, 10)}]
brain.fit(epoch_counter=range(15), train_set=data)
```

Listing 3: Training a simple model with SpeechBrain using the Brain class.

With only about ten lines of code, we can train a neural model. Repetitive boilerplate, such as setting `train()` and `eval()` flags, putting the models on the specified device, and computing gradients are handled by the `Brain` class. Users can override any step of the process, allowing the definition of more complicated (e.g., GAN [75]) training procedures. The `Brain` class also handles validation, learning rate scheduling, and fault-tolerant model checkpointing, so that training can resume where it left off if execution is interrupted (e.g., by preemption on a cluster). Further details about the `Brain` API are provided in § A.4.4.

4.5 Other features

Beyond the functionalities mentioned in the previous sections, additional features include:

Multi-GPU training: SpeechBrain supports both `DataParallel` and `DistributedDataParallel` modules, allowing the use of GPUs on the same and different machines. Automatic mixed-precision can be enabled by setting a single flag to reduce the memory footprint of the models. Moreover, the library supports PyTorch's Just-In-Time (JIT) compiler for native compilation.

Large-scale experiments: SpeechBrain extends WebDataset[6] with on-the-fly dynamic batching and bucketing. This enables efficient batching in sequential shard-based data reading, which is necessary for processing large corpora on network filesystems.

On-the-fly feature generation: Rather than serializing intermediate features to disk, SpeechBrain loads raw waveforms and supports a wide variety of efficient streaming operations for audio processing. Standard features like the Short-Term Fourier Transform (STFT) and Mel-filterbanks are computed at training time, allowing differentiation and waveform-level augmentation [76]. Many recipes include on-the-fly augmentations such as adding noise, time warping, or feature masking.

[6] `https://github.com/webdataset/webdataset`

5 Results

This section describes use cases of SpeechBrain, highlighting the techniques implemented and the corresponding performance. For more details on datasets, models, and experimental settings, please refer to the appendix (§ A.5).

5.1 Speech recognition

The toolkit supports common techniques for end-to-end speech recognition with different levels of complexity. The simplest system employs an encoder trained with Connectionist Temporal Classification (CTC) [77]. An alternative model is the Transducer [25], which augment CTC with an autoregressive component and a prediction network. The toolkit supports attention-based encoder-decoder architectures as well [26]. In particular, CTC+Att systems rely on an encoder-decoder architecture with an additional CTC loss applied on the top of the encoder. SpeechBrain is designed such that users can easily plug in any encoder and decoder modules into the speech recognition pipeline. For instance, we implemented an effective CRDNN encoder, which combines convolutional, recurrent (e.g., LSTM [78], GRU [79], Light GRU [80]), and fully connected neural networks. As an alternative, users can plug in one of the transformers that we have made available. Pre-training based on self-supervised learning (SSL) with wav2vec 2.0 [62] is supported.

We also implemented an efficient GPU-based beam search that combines the acoustic and the language information to retrieve the final sequence of words. The training scripts for language models and tokenizers (using SentencePiece [81]) are provided as well. In the following, we report the performance achieved with SpeechBrain recipes on some popular speech benchmarks.

5.1.1 TIMIT

TIMIT [31] is a small speech dataset with expert-labeled phone sequences. Table 2 reports the Phone Error Rate (PER) achieved with the aforementioned techniques. All systems use a CRDNN encoder, except for the CTC+Att+SSL one which uses a pre-trained wav2vec 2.0 encoder [62]. We report the average performance out of five runs with different random seeds. The standard deviation ranges between 0.15% and 0.2% in all the models.

CTC and Transducers provide similar results, while the combination of CTC and attention (CTC+Att) reaches the best performance. The results achieved by our best model (PER 13.8%) is SotA for TIMIT performance with no extra data. A considerable improvement in PER is observed when Light-GRUs [80] are used instead of GRUs [79] or LSTMs [78] in the CRDNN encoder. We also observe a performance boost when using self-supervised pre-training with the wav2vec model trained on unlabelled data from the Libri-Light dataset (CTC+Att+SSL) [82]. Our result with this Libri-Light self-supervised pre-training (PER of 8.04%) slightly outperforms the previous SotA performance with the same pre-training data (PER of 8.30%), as shown in Figure 3.

5.1.2 LibriSpeech

LibriSpeech [28] is a popular speech recognition benchmark derived from audiobooks. Table 3 reports the results achieved with different SpeechBrain recipes on this dataset.

Table 3: Word Error Rate (WER%) achieved on LibriSpeech with SpeechBrain.

Technique	Encoder	Decoder	# Params	`test-clean`	`test-other`
CTC+Att	CRDNN	GRU	230 M	2.91	8.07
CTC+Att	Transformer	GRU	161 M	2.46	5.77

Our best system is a transformer [61] combined with a convolutional front-end based on ContextNet [83]. The autoregressive decoder estimates 5k subword tokens derived from running byte-pair encoding on top of the training transcriptions [81]. A transformer-based LM is trained on the LibriSpeech text corpus and used within the beam search to rescore partial hypotheses. The best WER that we have achieved on the `test-clean` dataset is 2.46%. This performance is comparable with the results reached in the literature when using transformers without additional data [84]. As

Table 4: Equal Error Rate (EER %) achieved on VoxCeleb1 - Cleaned dataset.

Technique	EER(%)
VoxCeleb2 baseline [35]	3.95
Kaldi x-vector [32]	3.10
ResNET-50 [87]	1.19
ECAPA (original paper) [33]	0.87
SpeechBrain x-vector + PLDA	3.20
SpeechBrain ECAPA	0.81
SpeechBrain ECAPA (vox1+2)	**0.69**

Table 5: Diarization Error Rate (DER%) on the eval set of the AMI corpus.

Technique	Known # spks	Estim. # spks
MCGAN [88]	4.49	5.38
ClusterGAN [88]	3.91	8.16
xvector+MCGAN [88]	4.23	4.92
xvector+ClusterGAN [88]	3.60	**2.87**
VBx (ResNet101) [89]	—	4.58
SpeechBrain ECAPA	**2.82**	3.01

one can note, the LibriSpeech task is almost perfectly solved by modern speech recognizers. We thus focus on more realistic tasks as well, as suggested in some recent works [85, 86]. See the appendix (§ A.2) for a more detailed comparison with other toolkits on LibriSpeech and other tasks.

5.1.3 Common Voice

The Common Voice corpus [29] is a multilingual open-source collection of transcribed speech based on crowdsourcing data collection. CommonVoice is challenging due to significant accented speech, hesitations, presence of foreign words, noise, reverberation, and other recording artifacts.

Table 6 reports the results obtained on four different languages. No language models are trained for this task. The best results are obtained with a wav2vec 2.0 encoder pre-trained with 100k hours of multilingual data from the VoxPopuli dataset [90]. Except for English, the best systems use a GRU decoder on the top of the pre-trained transformer. CommonVoice is a newer dataset, and there have been relatively few systems evaluated on it. To the best of our knowledge, however, our results are SotA for these languages.

Table 6: Word Error Rate (WER%) achieved with Common Voice Corpus 6.1 using SpeechBrain on the English (En), French (Fr), Italian (It), and Kinyarwanda (Kw) subsets.

Technique	Encoder	Decoder	# Params	En	Fr	It	Kw
CTC+Att	CRDNN	GRU	148M	24.89	17.70	16.61	24.27
CTC+SSL	Transformer	-	320M	**15.58**	14.44	10.93	23.12
CTC+Att+SSL	Transformer	GRU	330M	15.69	**13.34**	**9.86**	**18.91**

5.2 Speaker recognition and diarization

SpeechBrain implements the functionalities needed to support speaker recognition and speaker diarization. It supports popular embeddings derived from Time Delay Neural Networks (TDNNs) [91, 92], such as x-vectors [32] and the recent ECAPA-TDNN embeddings [33]. Furthermore, SpeechBrain provides traditional Probabilistic Linear Discriminant Analysis (PLDA) for speaker discrimination [93, 94].

Table 4 reports the performance achieved on a speaker verification task with models trained on VoxCeleb2 [35] and tested on VoxCeleb1-clean [34]. The best model for speaker embeddings available in SpeechBrain is the ECAPA-TDNN, which matches the performance achieved in the original paper [33]. This model outperforms both the x-vectors [32] and the ResNet-34 [87] by a large margin. To the best of our knowledge, the EER reached so far by SpeechBrain on VoxCeleb is the best so far reached by an open-source toolkit.

Table 5 reports the performance achieved on speaker diarization with the AMI meeting corpus [38] when using the embeddings available in SpeechBrain. In this case, the embeddings are clustered with spectral clustering to assign a relative speaker label to each segment of the recording [37]. The results shown are obtained on the official Full-ASR split of the AMI corpus while keeping 0.25 sec of forgiveness collar. The best diarization system available in SpeechBrain outperforms recent

Table 7: Speech enhancement performance on VoiceBank-DEMAND.

Technique	# Params	PESQ	COVL
Facebook DEMUCS [95]	60.8 M	3.07	3.63
SpeechBrain Mimic Loss	22.3 M	3.05	**3.74**
SpeechBrain MetricGAN+	1.9 M	**3.15**	3.62

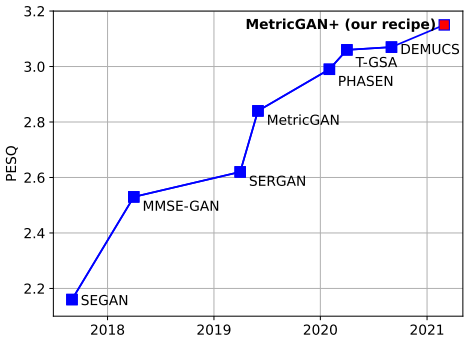

Figure 4: Evolution of the speech enhancement performance (PESQ) for Voicebank-DEMAND.

Table 8: Scale-invariant signal-to-noise ratio improvement (SI-SNRi) in dB achieved with SpeechBrain on WSJ2mix and WSJ3mix.

Technique	2-mix	3-mix
ConvTasnet	15.3	12.7
DualPath-RNN	18.8	14.7
SepFormer	20.4	17.6
SepFormer+DM	**22.3**	**19.5**

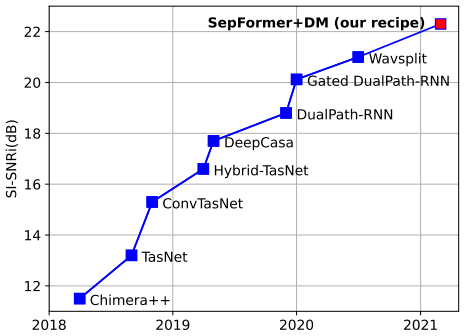

Figure 5: Evolution of the SotA performance (SI-SNRi) on the wsj2mix dataset. Source: `https://paperswithcode.com`.

approaches based on meta-learning (MCGAN/ClusterGAN) [88], and Variational Bayes (VBx) [89] when the number of speakers is known (e.g., in a meeting). We have also obtained competitive results when the number of speakers is unknown.

5.3 Speech enhancement and separation

SpeechBrain supports speech enhancement models with different input features (e.g., spectral and waveform domain) and training losses (e.g., L1, MSE, and STOI). In addition, it supports a variety of more sophisticated multi-model training techniques such as Mimic Loss [40] and MetricGAN+ [39].

In Table 7 we compare the best enhancement systems available in SpeechBrain against the SotA DEMUCS model [95] on the Voicebank-DEMAND corpus [96]. The mimic loss system uses a speech recognition model to provide a perceptual loss, achieving SotA performance on the COVL metric. Combining models for different tasks (as done here) is natural to implement in SpeechBrain. We also re-implemented the recently proposed MetricGAN+, which performs speech enhancement with an adversarially trained metric network [75]. Figure 4 shows the evolution of the PESQ performance on this corpus over the last few years. The SpeechBrain implementation of MetricGAN+ achieves the SotA PESQ performance when no extra data are used.

SpeechBrain implements popular models for speech separation as well, namely ConvTasnet [43] and Dual-path RNN [44]. Moreover, it supports the recently proposed SepFormer [45], which uses a pipeline of two transformers within a dual-path framework. Table 8 reports the results achieved on the standard WSJ0-2mix and WSJ0-3mix datasets [46], which contain mixtures composed of two or three overlapped speakers, respectively. The last row compares performance achieved with dynamic mixing, in which the training data are generated dynamically on-the-fly instead of using a frozen dataset. As shown in Figure 5, SpeechBrain's SepFormer implementation achieves SotA on both datasets.

6 Limitations and Future Work

The current version of SpeechBrain supports many other tasks, including spoken language understanding, keyword spotting, multi-microphone signal processing, and language modeling. The toolkit also supports complex [97] and quaternion neural networks [98]. Please refer to A.3 for further details. It does not currently support text-to-speech, which will be added shortly (pending pull-requests under review). In the future, we plan to support decoding with Finite State Transducers (FSTs) [99] and are considering to adopt the FST implementation of the ongoing k2 project [100] once stable. We plan to devote further effort to real-time speech processing, which was not the main focus of the first release. Finally, our goal is to add support for additional languages and further expand the set of recipes to open-source datasets not yet available in the toolkit (e.g., TED-LIUM [101]).

7 Conclusion

This paper described SpeechBrain, a novel, open-source, all-in-one speech processing toolkit. Our work illustrated the main design principles behind this toolkit and remarked on the design principles that led us to support multiple tasks without sacrificing simplicity, modularity, or flexibility. Finally, we showed several use cases where the technology developed in SpeechBrain reaches SotA or competitive performance. The main contribution to the scientific community is the development of a novel toolkit that can significantly accelerate future research in the fields of speech processing and deep learning. SpeechBrain is a coordinated effort towards making speech processing technology accessible, and are eager to see where its rapidly growing community of users takes the project in the future.

Acknowledgments and Disclosure of Funding

We would like to sincerely thank our generous sponsors: Samsung, Dolby, Nvidia, Nuance, ViaDialog. Special thanks to our institutional partners: Mila, LIA (Avignon University), CaMLSys (University of Cambridge), Sherbrooke University, and Bio-ASP (Academia Sinica). We also would like to acknowledge Breandan Considine, Olexa Bilaniuk, Frederic Osterrath, Mirko Bronzi, Anthony Larcher, Ralf Leibold, Salima Mdhaffar, Yannick Estève, Yu Tsao, Abdelmoumene Boumadane for helpful comments and discussions. We would like to express our gratitude to all the pre-release beta-testers and to the whole community that we are building around this project. Thanks to Compute Canada for providing computational resources and support. SpeechBrain was also granted access to the HPC resources of IDRIS under the allocation 2021-AD011012633 made by GENCI.

References

[1] S. Young, G. Evermann, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland. *The HTK Book*. Entropic Cambridge Research Laboratory, Cambridge, United Kingdom, 2002.

[2] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *in Proc. of ICASSP*, 2006.

[3] A. Lee, T. Kawahara, and K. Shikano. Julius: An open source realtime large vocabulary recognition engine. In *Proc. of EUROSPEECH*, 2001.

[4] D. Rybach, S. Hahn, P. Lehnen, D. Nolden, M. Sundermeyer, Z. Tüske, S. Wiesler, R. Schlüter, and H. Ney. RASR - The RWTH Aachen University Open Source Speech Recognition Toolkit. In *Proc. of ASRU*, 2011.

[5] D. Povey et al. The Kaldi Speech Recognition Toolkit. In *Proc. of ASRU*, 2011.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *Proc. of. USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proc. of NeurIPS*, 2019.

[8] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep speech: Scaling up end-to-end speech recognition, 2014. arXiv:1412.5567.

[9] A. Zeyer, T. Alkhouli, and H. Ney. RETURNN as a generic flexible neural toolkit with application to translation and speech recognition. In *Proc. of ACML*, 2018.

[10] M. Ravanelli, T. Parcollet, and Y. Bengio. The PyTorch-Kaldi Speech Recognition Toolkit. In *Proc. of ICASSP*, 2019.

[11] Y. Wang, T. Chen, H. Xu, S. Ding, H. Lv, Y. Shao, N. Peng, L. Xie, S. Watanabe, and S. Khudanpur. Espresso: A fast end-to-end neural speech recognition toolkit. In *Proc. of ASRU*, 2019.

[12] J. Shen, P. Nguyen, Y. Wu, Z. Chen, M. X. Chen, Y. Jia, A. Kannan, T. Sainath, Y. Cao, C. Chiu, et al. Lingvo: A modular and scalable framework for sequence-to-sequence modeling. 2019. arXiv:1902.08295.

[13] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. Fairseq: A fast, extensible toolkit for sequence modeling, 2019. arXiv:1904.01038.

[14] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala, and T. Ochiai. ESPnet: End-to-end speech processing toolkit. In *Proc. of Interspeech*, 2018.

[15] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook, P. Castonguay, M. Popova, J. Huang, and J. M. Cohen. NeMo: a toolkit for building AI applications using Neural Modules, 2019. arXiv:1909.09577.

[16] M. Pariente, S. Cornell, J. Cosentino, S. Sivasankaran, E. Tzinis, J. Heitkaemper, M. Olvera, F. Stöter, M. Hu, J. M. Martín-Doñas, D. Ditter, A. Frank, A. Deleforge, and E. Vincent. Asteroid: the PyTorch-based audio source separation toolkit for researchers. In *Proc. of Interspeech*, 2020.

[17] H. Bredin, R. Yin, J. M. Coria, G. Gelly, P. Korshunov, M. Lavechin, D. Fustes, H. Titeux, W. Bouaziz, and M. Gill. pyannote.audio: neural building blocks for speaker diarization. In *Proc. of ICASSP*, 2020.

[18] A. Larcher, K. A. Lee, and S. Meignier. An extensible speaker identification sidekit in python. In *Proc. of ICASSP*, 2016.

[19] S. Yang, P. Chi, Y. Chuang, C. Lai, K. Lakhotia, Y. Y. Lin, A. T. Liu, J. Shi, X. Chang, G. Lin, T. Huang, W. Tseng, K. Lee, D. Liu, Z. Huang, S. Dong, S. Li, S. Watanabe, A. Mohamed, and H. Lee. Superb: Speech processing universal performance benchmark, 2021. arXiv:2105.01051.

[20] Z. Wang and D. Wang. A joint training framework for robust automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(4):796–806, 2016.

[21] Z. Chen, S. Watanabe, H. Erdogan, and J.R. Hershey. Speech enhancement and recognition using multi-task learning of long short-term memory recurrent neural networks. In *Proc. of Interspeech*, 2015.

[22] T. Gao, J. Du, L. Dai, and C. Lee. Joint training of front-end and back-end deep neural networks for robust speech recognition. In *Proc. of ICASSP*, 2015.

[23] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio. Batch-normalized joint training for DNN-based distant speech recognition. In *Proc. of SLT*, 2016.

[24] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proc. of ICML*, 2006.

[25] A. Graves. Sequence transduction with recurrent neural networks. *ICML — Workshop on Representation Learning*, 2012.

[26] S. Watanabe, T. Hori, S. Kim, J. R. Hershey, and T. Hayashi. Hybrid ctc/attention architecture for end-to-end speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, 11 (8):1240–1253, 2017.

[27] S. Toshniwal, A. Kannan, C. Chiu, Y. Wu, T. Sainath, and K. Livescu. A comparison of techniques for language model integration in encoder-decoder speech recognition. 2018.

[28] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. LibriSpeech: An ASR corpus based on public domain audio books. In *Proc. of ICASSP*, 2015.

[29] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber. Common Voice: a massively-multilingual speech corpus. In *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, 2020.

[30] H. Bu, J. Du, X. Na, B. Wu, and H. Zheng. Aishell-1: An open-source mandarin speech corpus and a speech recognition baseline. In *Oriental COCOSDA*, 2017.

[31] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren. DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM, 1993.

[32] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur. X-vectors: Robust DNN embeddings for speaker recognition. In *Proc. of ICASSP*, 2018.

[33] B. Desplanques, J. Thienpondt, and K. Demuynck. ECAPA-TDNN: emphasized channel attention, propagation and aggregation in TDNN based speaker verification. In *Proc. of Interspeech*, 2020.

[34] A. Nagrani, J. S. Chung, and A. Zisserman. Voxceleb: a large-scale speaker identification dataset. In *Proc. of Interspech*, 2017.

[35] J. S. Chung, A. Nagrani, and A. Zisserman. Voxceleb2: Deep speaker recognition. In *Proc. of Interspeech*, 2018.

[36] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 2007.

[37] N. Dawalatabad, M. Ravanelli, F. Grondin, J. Thienpondt, B. Desplanques, and H. Na. ECAPA-TDNN embeddings for speaker diarization, 2021. arXiv:2104.01466.

[38] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal, G. Lathoud, M. Lincoln, A. Lisowska, I. McCowan, W. Post, D. Reidsma, and P. Wellner. The AMI meeting corpus: A pre-announcement. In *Proc. of the Second International Conference on Machine Learning for Multimodal Interaction*, 2006.

[39] S. Fu, C. Yu, T. Hsieh, P. Plantinga, M. Ravanelli, X. Lu, and Y. Tsao. MetricGAN+: An Improved Version of MetricGAN for Speech Enhancement. 2021. arXiv:2104.03538.

[40] D. Bagchi, P. Plantinga, A. Stiff, and E. Fosler-Lussier. Spectral feature mapping with mimic loss for robust speech recognition. In *Proc. of ICASSP*, 2018.

[41] C. Veaux, J. Yamagishi, and S. King. The voice bank corpus: Design, collection and data analysis of a large regional accent speech database. In *International Conference Oriental COCOSDA held jointly with Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*, 2013.

[42] C. Reddy, V. Gopal, R. Cutler, E. Beyrami, R. Cheng, H. Dubey, S. Matusevych, R. Aichner, A. Aazami, S. Braun, P. Rana, S. Srinivasan, and J. Gehrke. The interspeech 2020 deep noise suppression challenge: Datasets, subjective testing framework, and challenge results. In *Proc. of Interspeech*, 2020.

[43] Y. Luo and N. Mesgarani. Conv-tasnet: Surpassing ideal time-frequency magnitude masking for speech separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(8):1256–1266, 2019.

[44] Y. Luo, Z. Chen, and T. Yoshioka. Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation, 2020. arXiv:1910.06379.

[45] C. Subakan, M. Ravanelli, S. Cornell, M. Bronzi, and J. Zhong. Attention is all you need in speech separation. In *Proc. of ICASSP*, 2021.

[46] J. Hershey, Z. Chen, J. Le Roux, and S. Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. In *Proc. of ICASSP*, 2016.

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

[48] M. Maciejewski, G. Wichern, E. McQuinn, and J. Le Roux. Whamr!: Noisy and reverberant single-channel speech separation. In *Proc. of ICASSP*, 2020.

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

[50] L. Lugosch, P. Papreja, M. Ravanelli, A. Heba, and T. Parcollet. Timers and Such: A practical benchmark for spoken language understanding with numbers. *CoRR*, abs/2104.01604, 2021.

[51] P. Haghani, A. Narayanan, M. Bacchiani, G. Chuang, N. Gaur, P. Moreno, R. Prabhavalkar, Z. Qu, and A. Waters. From Audio to Semantics: Approaches to end-to-end spoken language understanding. *IEEE Spoken Language Technology Workshop (SLT)*, 2018.

[52] D. Serdyuk, Y. Wang, C. Fuegen, A. Kumar, B. Liu, and Y. Bengio. Towards end-to-end spoken language understanding. In *Proc. of ICASSP*, 2018.

[53] E. Bastianelli, A. Vanzo, P. Swietojanski, and V. Rieser. SLURP: A Spoken Language Understanding Resource Package. In *Proc. of EMNLP*, 2020.

[54] L. Lugosch, M. Ravanelli, P. Ignoto, V. Tomar, and Y. Bengio. Speech model pre-training for end-to-end spoken language understanding. In *Proc. of Interspeech*, 2019.

[55] E. Habets, J. Benesty, I. Cohen, S. Gannot, and J. Dmochowski. New insights into the MVDR beamformer in room acoustics. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(1):158–170, 2009.

[56] J. Heymann, L. Drude, and R. Haeb-Umbach. Neural network based spectral mask estimation for acoustic beamforming. In *Proc. of ICASSP*, 2016.

[57] C. H. Knapp and G. C. Carter. The generalized correlation method for estimation of time delay. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(4):320–327, 1976.

[58] M. Cobos, A. Marti, and J. Lopez. A modified SRP-PHAT functional for robust real-time sound source localization with scalable spatial sampling. *IEEE Signal Processing Letters*, 18 (1):71–74, 2010.

[59] R. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE transactions on antennas and propagation*, 34(3):276–280, 1986.

[60] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. Dauphin. Convolutional sequence to sequence learning. In *Proc. of ICML*. PMLR, 2017.

[61] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. of NeurIPS*, 2017.

[62] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Proc. of NeurIPS*, 2020.

[63] J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. Teja Gadde. Jasper: An end-to-end convolutional neural acoustic model. In *Proc. of Interspeech*, 2019.

[64] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In *Proc. of ICASSP*, 2020.

[65] S. Majumdar, J. Balam, O. Hrinchuk, V. Lavrukhin, V. Noroozi, and B. Ginsburg. Citrinet: Closing the gap between non-autoregressive and autoregressive end-to-end models for automatic speech recognition, 2021. arXiv:2104.01721.

[66] G. Fairbanks, D. Garlan, and W. Scherlis. Design fragments make using frameworks easier. *SIGPLAN Not.*, 41(10), 2006.

[67] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In *Proc of EMNLP*, 2020.

[68] E. Variani, T. Bagby, E. McDermott, and M. Bacchiani. End-to-end training of acoustic models for large vocabulary continuous speech recognition with tensorflow. In *Proc. of Interspeech*, 2017.

[69] M. Morishita, Y. Oda, G. Neubig, K. Yoshino, K. Sudoh, and S. Nakamura. An empirical study of mini-batch creation strategies for neural machine translation. In *Proc. of the WNMT*, 2017.

[70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[71] P. Virtanen, R. Gommers, T. E Oliphant, M. Haberland, T. Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, 17(3):261–272, 2020.

[72] A. Gulli and S. Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[73] J. Howard and S. Gugger. Fastai: A layered api for deep learning. *Information*, 11(2):108, 2020.

[74] W. Falcon et al. Pytorch Lightning. *GitHub*, 2019. `https://github.com/PyTorchLightning/pytorch-lightning`.

[75] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Proc. of NIPS*, 2014.

[76] D. Park, W. Chan, Y. Zhang, C. Chiu, B. Zoph, E. Cubuk, and Q. Le. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Proc. of Interspeech*, 2019.

[77] A. Graves and N. Jaitly. End-to-end speech recognition with recurrent neural networks. In *Proc. of ICML*, 2014.

[78] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

[79] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Proc. of NIPS*, 2014.

[80] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2): 92–102, 2018.

[81] T. Kudo and J. Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proc. of EMNLP*, 2018.

[82] J. Kahn, M. Rivière, W. Zheng, E. Kharitonov, Q. Xu, P. Mazaré, J. Karadayi, V. Liptchinsky, R. Collobert, C. Fuegen, T. Likhomanenko, G. Synnaeve, A. Joulin, A. Mohamed, and E. Dupoux. Libri-light: A benchmark for ASR with limited or no supervision. *CoRR*, abs/1912.07875, 2019.

[83] W. Han, Z. Zhang, Y. Zhang, J. Yu, C. Chiu, J. Qin, A. Gulati, R. Pang, and Y. Wu. Contextnet: Improving convolutional neural networks for automatic speech recognition with global context. 2020. arXiv:2005.03191.

[84] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. Soplin, R. Yamamoto, X. Wang, S. Watanabe, T. Yoshimura, and W. Zhang. A comparative study on transformer vs rnn in speech applications. In *Proc. of ASRU*, 2019.

[85] P. Szymański, P. Żelasko, M. Morzy, A. Szymczak, M. Żyła-Hoppe, J. Banaszczak, L. Augustyniak, J. Mizgajski, and Y. Carmiel. WER we are and WER we think we are. In *Proc. of EMNLP*, 2020.

[86] T. Likhomanenko, Q. Xu, V. Pratap, P. Tomasello, J. Kahn, G. Avidov, R. Collobert, and G. Synnaeve. Rethinking evaluation in ASR: are our models robust enough? *CoRR*, abs/2010.11745, 2020.

[87] H. Zeinali, S. Wang, A. Silnova, P. Matějka, and O. Plchot. But system description to voxceleb speaker recognition challenge 2019. In *Proc. of The VoxCeleb Challange Workshop*, 2019.

[88] M. Pal, M. Kumar, R. Peri, T. Park, S. Kim, C. Lord, S. Bishop, and S. Narayanan. Meta-learning with latent space clustering in generative adversarial network for speaker diarization, 2020. arXiv:2007.09635.

[89] F. Landini, J. Profant, M. Diez, and L. Burget. Bayesian HMM clustering of x-vector sequences (VBx) in speaker diarization: theory, implementation and analysis on standard tasks, 2020. arXiv:2012.14952.

[90] C. Wang, M. Rivière, A. Lee, A. Wu, C. Talnikar, D. Haziza, M. Williamson, J. Pino, and E. Dupoux. Voxpopuli: A large-scale multilingual speech corpus for representation learning, semi-supervised learning and interpretation. *arXiv:2101.00390*, 2021.

[91] K. J. Lang and G. E. Hinton. The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University, 1988.

[92] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:328–339, 1989.

[93] P. Kenny, T. Stafylakis, P. Ouellet, Md. J. Alam, and P. Dumouchel. PLDA for speaker verification with utterances of arbitrary duration. In *Proc. of ICASSP*, 2013.

[94] D. Garcia-Romero and C. Espy-Wilson. Analysis of i-vector length normalization in speaker recognition systems. In *Proc. of Interspeech*, 2011.

[95] A. Défossez, G. Synnaeve, and Y. Adi. Real time speech enhancement in the waveform domain. *Proc. of Interspeech*, 2020.

[96] C. Valentini-Botinhao. Noisy speech database for training speech enhancement algorithms and TTS models. *Edinburgh DataShare*, 2017.

[97] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. Pal. Deep complex networks. In *Proc. of ICLR*, 2018.

[98] T. Parcollet, M. Ravanelli, M. Morchid, G. Linarès, C. Trabelsi, R. De Mori, and Y. Bengio. Quaternion recurrent neural networks. In *Proc. of ICLR*, 2019.

[99] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[100] D. Povey et al. k2. `https://github.com/k2-fsa/k2`, 2020.

[101] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Estève. TED-LIUM 3: Twice as much data and corpus repartition for experiments on speaker adaptation. In *Proc. of SPECOM*, 2018.

[102] C. Li, J. Shi, W. Zhang, A. Subramanian, X. Chang, N. Kamo, M. Hira, T. Hayashi, C. Böddeker, Z. Chen, and Shinji Watanabe. Espnet-se: End-to-end speech enhancement and separation toolkit designed for ASR integration. In *Proc. of the IEEE Spoken Language Technology Workshop*, 2021.

[103] H. Erdogan, J. Hershey, S. Watanabe, M. Mandel, and J. Le Roux. Improved MVDR beamforming using single-channel mask prediction networks. In *Proc. of Interspeech*, 2016.

[104] F. Grondin, J. Lauzon, J. Vincent, and F. Michaud. GEV beamforming supported by DOA-based masks generated on pairs of microphones. In *Proc. of Interspeech*, 2020.

[105] F. Grondin and F. Michaud. Lightweight and optimized sound source localization and tracking methods for open and closed microphone array configurations. *Robotics and Autonomous Systems*, 2019.

[106] M. Omologo and P. Svaizer. Acoustic event localization using a crosspower-spectrum phase based technique. In *Proc. of ICASSP*, 1994.

[107] J. Du, Q. Wang, T. Gao, Y. Xu, L. Dai, and C. Lee. Robust speech recognition with speech enhanced deep neural networks. In *Proc. of Interspeech*, 2014.

[108] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *Proc. of ICLR*, 2015.

[109] S. Seo, D. Kwak, and B. Lee. Integration of pre-trained networks with continuous token interface for end-to-end spoken language understanding. *arXiv:2104.07253*, 2021.

[110] J. Thiemann, N. Ito, and E. Vincent. The Diverse Environments Multi-channel Acoustic Noise Database (DEMAND): A database of multichannel environmental noise recordings. In *21st International Congress on Acoustics*, 2013.

[111] 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 *Proc. of ICASSP*, 2001.

[112] Y. Hu and P. Loizou. Evaluation of objective quality measures for speech enhancement. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(1):229–238, 2007.

[113] E. Vincent, R. Gribonval, and C. Fevotte. Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1462–1469, 2006.

A Appendix

A.1 Statement on social impact

Speech technologies can support humans in a variety of positive ways (e.g., helping hearing-impaired individuals, detecting speech pathologies, helping people learning new languages, allowing people with physical disabilities to control their home appliances, etc.). They can make our life safer (e.g., in-car speech recognition) or just more comfortable (e.g., with voice assistants, etc.). The growing demand for speech technology observed in the last few years confirms the importance of this technology in everyday lives. However, non-ethical misuses of these technologies are possible as well. Most of them are related to well-known privacy concerns, which can be mitigated with more rigid regulations such as the General Data Protection Regulation[7] (GDPR) adopted in Europe.

As with all other open-source toolkits, we cannot have full control over the actual use of the developed technologies. However, we strongly encourage ethical use of our toolkit, and we ask all SpeechBrain users to fully respect the *Montreal Declaration for a Responsible Development of Artificial Intelligence*[8]. Moreover, we think that having open-source technology available to everyone is better than leaving it in the hands of a few players only. This can potentially mitigate the negative consequences of this ongoing societal change towards an AI-aided society.

A.2 Performance comparison with other toolkits

Comparing the performance across speech processing toolkits is often problematic for several reasons and can be deceptive if not framed in a much larger context. First, each toolkit focuses more on some tasks or models and provides recipes only for specific datasets. Secondly, there are intrinsic differences in how these toolkits implement recipes for the same task on the same dataset. For example, Kaldi relies only on hybrid speech recognition, while others such as SpeechBrain, ESPNet, and NeMo do not currently support hybrid speech recognition but instead provide more modern E2E speech recognition models. Thirdly, even across recipes concerning the same model on the same dataset, some differences arise due to different feature extraction, data loading pipelines, batching mechanisms, and other implementation details. Finally, most of the aforementioned toolkits are active projects, and the performance of a given task might change over time. We think that the comparison proposed in the section can only be used to probe whether a toolkit can provide reasonable performance compared to other open-source implementations. Table 9 compares the best results reported in the official repository of each toolkit on tasks and datasets we have found in common (as of May 2021).

We can see that SpeechBrain achieves competitive performance with other pre-existing toolkits across different tasks and datasets. It is worth mentioning that all the toolkits considered here can support all the tasks and datasets reported in Table 9. Each toolkit can potentially fill the performance gap with the best-performing one just by implementing a better model with properly fine-tuned hyperparameters for the specific task. We thus think that the actual value of a toolkit mainly lies in its usability and flexibility, which are the main principles that guided the design of SpeechBrain.

A.3 Additional tasks

In the following, we describe some of the supported applications not discussed in the main paper.

A.3.1 Multi-microphone signal processing

Multi-microphone signal processing techniques are useful in different ways within a speech processing pipeline. The information captured by different microphones can be used to estimate the direction of arrival (DOA) of a sound source. We can then use beamforming to enhance the target source. SpeechBrain performs multi-channel processing in the frequency domain. For both DOA estimation and beamforming, it is assumed that the spatial covariance matrices (SCMs) are computed for each frequency bin k using the Short-Time Fourier Transform (STFT). We denote the SCMs for the target speech, the interfering noise and the resulting mixture as $\mathbf{R}_{SS}[k] \in \mathbb{C}^{M \times M}$, $\mathbf{R}_{NN}[k] \in \mathbb{C}^{M \times M}$ and

[7] `https://gdpr-info.eu`

[8] `https://www.montrealdeclaration-responsibleai.com/`

Table 9: Performance comparison across speech toolkits on common tasks. For each toolkit, dataset, and task we report the best performance on the test set (as of May 2021). The arrow ↓ indicates the lower the better, while ↑ indicates the higher the better.

Task type	Metric	Dataset	SpeechBrain	ESPNet	NeMo	Kaldi
Speech Rec.	WER(%) ↓	Common Voice Fr	**13.34***	13.9[a] †	14.01[b]	n.a
Speech Rec.	WER(%) ↓	Common Voice It	**9.86***	16.1[c] †	15.22[d]	n.a
Speech Rec.	WER(%) ↓	LibriSpeech `test-clean`	2.46	2.1 [e]	**2.00**[f]	4.17[g]
Speech Rec.	CER(%) ↓	AISHELL-1	5.58	**4.7**[h]†	5.55[i]	7.43[j]
Speech Rec.	PER(%) ↓	TIMIT	**8.04***	19.5[k]	n.a.	18.4[l]
Speaker Ver.	EER(%) ↓	Voxceleb1+2	**0.69**	n.a	2.05[m]	3.10 [n]
Speech Sep.	SNRi(dB) ↑	WSJ2-mix	**22.3**	17.9[o]	n.a.	n.a.

*uses self-supervised pre-training with wav2vec 2.0.
† ESPNet uses transformer language models (SpeechBrain does not for these tasks).

[a] https://github.com/espnet/espnet/blob/master/egs2/commonvoice/asr1/README.md
[b] https://ngc.nvidia.com/catalog/models/nvidia:nemo:stt_fr_quartznet15x5
[c] https://github.com/espnet/espnet/blob/master/egs2/commonvoice/asr1/README.md
[d] https://ngc.nvidia.com/catalog/models/nvidia:nemo:stt_it_quartznet15x5
[e] https://github.com/espnet/espnet/tree/master/egs2/librispeech/asr1
[f] Result taken from [65].
[g] https://github.com/kaldi-asr/kaldi/blob/master/egs/librispeech/s5/RESULTS
[h] https://github.com/espnet/espnet/tree/master/egs2/aishell/asr1
[i] Results taken from [65]. It uses extra data from Multilingual LibriSpeech.
[j] https://github.com/kaldi-asr/kaldi/blob/master/egs/aishell/s5/RESULTS
[k] https://github.com/espnet/espnet/blob/master/egs2/timit/asr1/README.md
[l] https://github.com/kaldi-asr/kaldi/blob/master/egs/timit/s5/RESULTS
[m] https://ngc.nvidia.com/catalog/models/nvidia:nemo:speakerverification_speakernet/
[n] https://kaldi-asr.org/models/m7
[o] Results taken from [102].

$\mathbf{R}_{XX}[k] \in \mathbb{C}^{M \times M}$, respectively, where M stands for the number of microphones. The SCMs for speech and noise can be obtained using time-frequency masks [56, 103, 104].

The DOA of the sound can be computed using the Generalized Cross-Correlation Phase Transform (GCC-PHAT) [57], the Steered-Response Power with Phase Transform (SRP-PHAT) [58], or the the Multiple Signal Classification (MUSIC) algorithm. All of these techniques are implemented in SpeechBrain using GPU-friendly functions. The GCC-PHAT computes the DOA on a pair of microphones and returns the time difference of arrival (TDOA), which can be mapped to a DOA on an arc from 0° to 180°. SRP-PHAT scans each potential DOA on a virtual unit sphere around the array and computes the corresponding power [105]. For each DOA (denoted by the unit vector $\mathbf{u}$), there is a steering vector $\mathbf{A}(k, \mathbf{u}) \in \mathbb{C}^{M \times 1}$ in the direction of $\mathbf{u}$:

$$E(\mathbf{u}) = \sum_{p=1}^{M} \sum_{q=p+1}^{M} \sum_{k} \frac{R_{p,q}[k]}{|R_{p,q}[k]|} A_p(k, \mathbf{u}) A_q(k, \mathbf{u})^* \tag{1}$$

where $R_{p,q}[k]$ stands for the element at the p-th row and q-th column in the SCM, and $A_p(k, \mathbf{u})$ and $A_q(k, \mathbf{u})$ stands for the p-th and q-th elements of the steering vector. The DOA $\mathbf{u}$ with the maximum power $E(\mathbf{u})$ is selected as the DOA of sound. It is worth mentioning that SRP-PHAT [58] is conceptually the same as another popular localization technique called Global Coherence Field (GCF) [106], which projects the DOA information into 2D or 3D plans. That will be possibly the object of future implementation in SpeechBrain.

It is also possible to estimate the DOA using the Multiple Signal Classification (MUSIC) algorithms [59]. MUSIC scans each potential direction of arrival on a virtual unit sphere around the array and computes the corresponding power. The matrix $\mathbf{U}(k) \in \mathbb{C}^{M \times S}$ contains the S eigenvectors that correspond to the S smallest eigenvalues obtained while performing eigenvalue decomposition on the

SCM. The power corresponds to:

$$E(\mathbf{u}) = \sum_k \frac{\mathbf{A}(k,\mathbf{u})^H \mathbf{A}(k,\mathbf{u})}{\sqrt{\mathbf{A}(k,\mathbf{u})^H \mathbf{U}(k)\mathbf{U}(k)^H \mathbf{A}(k,\mathbf{u})}} \tag{2}$$

where $\{\dots\}^H$ stands for the Hermitian operator, and the DOA corresponds to the unit vector $\mathbf{u}$ associated with the maximum value of $E(\mathbf{u})$.

Speech can be enhanced with beamforming methods. The most straightforward approach consists of using a delay-and-sum beamformer to produce constructive interference in the DOA of the target sound source. Beamforming generates frequency-wise coefficients $\mathbf{W}(k) \in \mathbb{C}^{M\times 1}$ that multiply the STFT of each microphone ($\mathbf{X}(t,k) \in \mathbb{C}^{M\times T}$) and adds the products to produce the enhanced speech STFT ($Y(t,k) \in \mathbb{C}^{1\times T}$):

$$Y(t,k) = \mathbf{W}^H(k)\mathbf{X}(t,k) \tag{3}$$

With delay-and-sum, the coefficients are obtained using the steering vector as follows:

$$\mathbf{W}(k) = \frac{1}{M}\mathbf{A}(k) \tag{4}$$

Alternatively, the Minimum Variance Distortionless Response (MVDR) beamformer [55] exploits the DOA but also the SCMs, and generates the following coefficients:

$$\mathbf{W}(k) = \frac{\mathbf{R}_{XX}^{-1}(k)\mathbf{A}(k)}{\mathbf{A}^H(k)\mathbf{R}_{XX}^{-1}(k)\mathbf{A}(k)} \tag{5}$$

Finally, the Generalized Eigenvalue Decomposition (GEV) beamformer [56] extracts the principal component using generalized eigenvalue decomposition using the speech and noise SCMs to generate the coefficients:

$$\mathbf{R}_{SS}(k)\mathbf{W}(k) = \lambda \mathbf{R}_{NN}(k)\mathbf{W}(k) \tag{6}$$

Speech enhancement using beamforming methods is appealing for speech recognition as it improves the signal-to-distortion ratio (SDR) without introducing nonlinearities that might hurt the speech recognition performance [107].

A.3.2 Spoken language understanding

SpeechBrain has several recipes for spoken language understanding (SLU). The SLU recipes demonstrate many useful capabilities of the toolkit, like combining pre-trained tokenizers, language models, and ASR models from other recipes, and using different input sources (audio or text) depending on whether the model is training or testing.

There are currently recipes for three open-source SLU datasets with different levels of complexity: Fluent Speech Commands (FSC) [54], Timers and Such [50], and SLURP [53]. The recipes all use attention-based RNN sequence-to-sequence models [108] to map the input (either the speech signal or a transcript) to the output (a semantic dictionary containing the intent/slots/slot values for the utterance, as a sequence of characters).

The recipes implement both "conventional" SLU (training on ground-truth transcripts) and "end-to-end" SLU (training on audio). The conventional "decoupled" recipe uses the LibriSpeech ASR model described in § 5.1.2 to transcribe the input signal at test time, instead of using the true transcript. The "multistage" [51] end-to-end recipe uses the same ASR model but during both training and testing. The "direct" [52] recipe uses a single model to map audio directly to semantics without an intermediate search step. For the ASR-based models, either the default LibriSpeech language model or a language model trained on the SLU dataset transcripts can be used.

The test accuracy for our FSC recipe is 99.60%, which is close to the recent SotA CTI model based on wav2vec 2.0 (99.7% in [109]). No comparisons for Timers and Such with other papers are available

Table 10: Performance on SLURP (audio as input).

Model	**scenario** (accuracy)	**action** (accuracy)	**intent** (accuracy)	**Word-F1**	**Char-F1**	**SLU-F1**
HerMiT [53]	85.69	81.42	78.33	69.34	72.39	70.83
CTI [109]	—	—	**86.92**	—	—	**74.66**
SpeechBrain Direct (CRDNN)	82.15	77.79	75.64	62.35	66.45	64.34
SpeechBrain Direct (wav2vec 2.0)	**89.49**	**86.40**	85.34	**72.60**	**76.76**	**74.62**

yet, as the dataset was only released recently [50]. The performance metrics for SLURP with audio as input are given in Table 10. Our direct recipe using a wav2vec 2.0 encoder outperforms the HerMiT baseline provided in the original SLURP paper [53] across all metrics. The recipe is slightly worse than SotA performance achieved by CTI for the intent accuracy metric and closely matches the SLU-F1 metric reported in [109]. Note that unlike CTI, our recipe currently does not use NLU pre-training and does not take advantage of an application-specific CRF architecture or word-aligned slot and slot value labels; instead, the recipe uses a very simple seq2seq model to predict the semantic dictionary directly. When this seq2seq model is applied directly to the ground-truth transcripts instead of audio, we achieve state-of-the-art results (Table 11).

Table 11: Performance on SLURP (NLU / ground-truth transcripts as input).

Model	**scenario** (accuracy)	**action** (accuracy)	**intent** (accuracy)
HerMiT [53]	90.15	86.99	84.84
CTI [109]	—	—	87.73
SpeechBrain NLU	**91.45**	**89.46**	**88.68**

A.4 Architecture details

In this section, we provide more details on the SpeechBrain architecture outlined in § 4.

A.4.1 Data preparation

The goal of data preparation is to parse a dataset and create the data-manifest files, which contain meta-information about the input data (e.g., file path, annotation, etc.). SpeechBrain data-io supports the CSV and JSON file formats, or the user can simply provide a dict. Listing 4 reports an excerpt of a JSON data-manifest file for speech recognition:

```
{
  "sentence001": {
    "wav": "{data_root}/file_snt001.wav",
    "length": 2.10,
    "words": "SWITCH OFF THE LIGHT"
  },
}
```

Listing 4: An excerpt of a JSON data-manifest file for speech recognition.

We use a dict (key-value map) structure where each example or spoken utterance is identified and addressable by a unique key or *example ID*. The entries in each example vary by task and dataset. For example, in speech recognition, audio files and the corresponding text are needed, whereas, in source separation, we would expect each example to contain the entries for *mixture* and *sources* signals. The CSV format can also be used:

```
ID,length,wav,words
sentence001,2.10,{data_root}/file_snt001.wav,"SWITCH OFF THE LIGHT"
sentence002,2.70,{data_root}/file_snt002.wav,"SWITCH ON THE LIGHT"
sentence003,3.20,{data_root}/file_snt003.wav,"PLEASE, TURN OFF THE
    LIGHT"
```

Listing 5: An excerpt of a CSV data-manifest file for speech recognition.

Dataset parsing scripts, which create the data-manifest files, are provided for many commonly-used speech datasets. Since the data manifests can generally be made relative to the data directory root, data-manifest files can even be provided for download directly, skipping the dataset parsing. All datasets and tasks tend to have at least small subtle differences in formats, and thus SpeechBrain does not have any required entries besides the example ID.

A.4.2 HyperPyYAML details

Our primary additions to the YAML format are addition of the following special tags, which are are added before an item definition, and are prefixed with `!`:

- `!new`: instantiates python objects. The object is created with the arguments passed with a list for positional arguments or as a dictionary for keyword arguments.
- `!name`: creates function objects. Behind the scenes, it uses functools.partial to create a new function definition with the default arguments provided.
- `!ref`: used for referring to a previously-defined item. It can support simple arithmetic and string concatenation for basic hyperparameter combinations.
- `!copy`: used to perform a deep copy of a previosly define object.
- `!tuple`: creates tuples.
- `!include`: used to insert other YAML files.
- `!apply`: loads and executes a python function, storing the result.

A.4.3 Data-io details

SpeechBrain data-io is built to extend PyTorch `data.utils` and provides the user with several abstractions for reading, encoding, padding and batching data. It is designed with speech processing in mind specifically, but most of the problems it solves are general to variable-length sequence processing.

Most of the data-io is built around four abstractions: `DynamicItemDataset`, `DynamicItem`, `PaddedBatch` and `SaveableDataloader`. SpeechBrain also provides a `CategoricalEncoder` class which implements label encoding for classification tasks such as speaker recognition. The SpeechBrain data-io is illustrated in Figure 6.

Based on a data-manifest (file or dict), a `DynamicItemDataset` can be created.

```
from speechbrain.dataio.dataset import DynamicItemDataset
train_dataset = DynamicItemDataset.from_json("train.json")
val_dataset = DynamicItemDataset.from_json("val.json")
```

Listing 6: `DynamicItemDataset` instantiation.

Each entry in an example is an *item*, following the Python dict terminology. The items that the data-manifest provides statically are called *static items*: they are kept in memory and stay unchanged. *Dynamic items* are specified by a transformation (e.g., a function) of any number of existing items. These dynamic items are evaluated on-demand. A clear case is a dynamic item that takes a path to an audio file and provides the loaded audio signal. Dynamic items can take other dynamic items as inputs, and a dependency graph is used to determine an evaluation order. Thus, another dynamic item

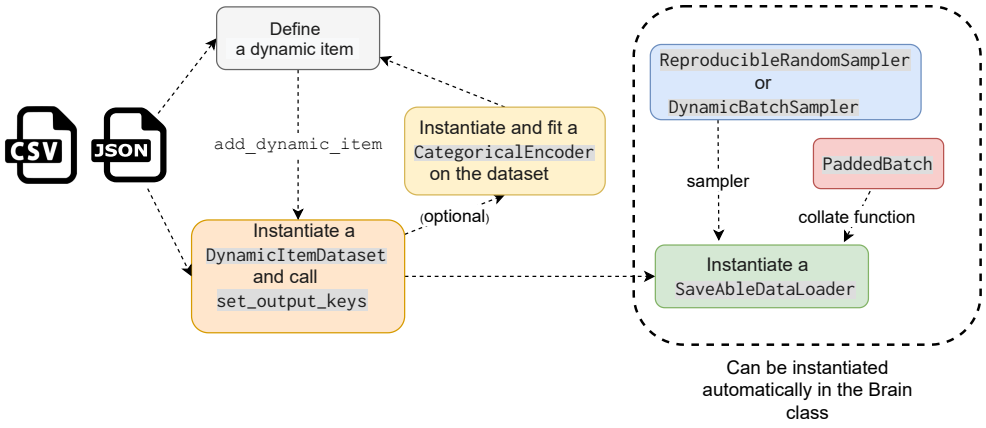

Figure 6: An overview of SpeechBrain data-io.

could take the loaded audio signal and provide an augmented version. A `GeneratorDynamicItem` takes any number of inputs and provides a chain of related dynamic items via the Python generator function syntax. Listings 7 and 8 show implementations of the aforementioned examples: first, a dynamic item loads an audio file, and then a chain of dynamic items augment the loaded signal.

```
@speechbrain.utils.data_pipeline.takes("file_path")
@speechbrain.utils.data_pipeline.provides("signal")
def audio_input(file_path):
    sig = speechbrain.dataio.dataio.read_audio(file_path)
    return sig
```

Listing 7: A dynamic item which loads an audio file.

```
import random
speechbrain.utils.data_pipeline.takes("sig")
@speechbrain.utils.data_pipeline.provides("rgain", "rgain_offset")
def augmentation(sig):
    random_gain_sig = sig*random.rand()
    yield random_gain_sig
    sig_with_offset = random_gain_sig + 1
    yield sig_with_offset
```

Listing 8: Example of a chain of dynamic items, which augments the output of another dynamic item.

The user specifies which items should be returned by the DynamicItemDataset. Items are evaluated lazily: only the strictly necessary operations for the user requested items are performed. This allows for significant computational savings and faster execution. For example in listing 9 only the example ID (*id*) and speaker ID (*spkid*) static items and the random gain augmentation dynamic item (*rgain*) are requested.

```
speechbrain.dataio.dataset.set_output_keys(
        [train_dataset], ["id, "spkid", "rgain"],
    )
```

Listing 9: Setting output items for the train_dataset `DynamicItemDataset`

Since the audio tensor with offset (*rgain_offset*) is not requested it is not computed at all in this example. On the contrary the audio tensor *sig* is needed for computing *rgain* and thus it is evaluated.

The `DynamicItemDataset` can thus provide multiple different views of the same dataset on demand. Iterating over the dataset can be extremely fast if the user only needs a particular item, e.g., to fit a `CategoricalEncoder`. Listing 10 shows fitting a `CategoricalEncoder` in a speaker identification task, continuing the above examples.

```
from speechbrain.dataio.encoder import CategoricalEncoder
spk_id_encoder = CategoricalEncoder()
spk_id_encoder.update_from_didataset(dataset, "spkid")
train_dataset.add_dynamic_item(spk_id_encoder.encode_label, takes="
    spkid", provides="spkid_enc")
```

Listing 10: Fitting a `CategoricalEncoder` for speaker recognition. This only evaluates the *spkid* item.

SpeechBrain also provides `CategoricalEncoder` sub-classes for encoding text and handle special tokens for the training of sequence-to-sequence models.

A `DynamicItemDataset` object can be wrapped by a standard PyTorch `DataLoader` or by SpeechBrain `SaveableDataloader`. The `Brain` class can handle this automatically for the user and uses the `SaveableDataloader` by default with `PaddedBatch` as the default *collate function*. and injecting `ReproducibleRandomSampler` as the *sampler* in case `shuffle=True` . More in general, custom PyTorch *samplers* and *collate functions* can be integrated seamlessly in SpeechBrain data-io.

`SaveableDataloader` allows for intra-epoch checkpointing, a feature that is useful when running extremely computationally demanding experiments where each epoch can take several hours.

`PaddedBatch` is both a *collate function* and a batch object. It handles for the user the rather annoying task of padding examples together. By default, it batches together only PyTorch tensors by adding zeros to the right on the last dimension. Other data types are not batched together but, instead, are returned in a python list. It also provides a semantically meaningful interface, as shown in listing 11.

```
from speechbrain.dataio.dataloader import make_dataloader
train_dataset.set_output_keys(["id", "rgain"])
dataloader = make_dataloader(train_dataset, batch_size=8)
for batch in dataloader:
    # Access a list of the example IDs in this batch
    batch.id
    # Access the speech data:
    batch.rgain.data
    # Access the relative lengths:
    batch.rgain.lengths
```

Listing 11: Accessing in `PaddedBatch` each requested item as well as the relative lengths of the padded data

A.4.4 `Brain` class details

The `Brain` class implements customizable methods for managing the different aspects of training and evaluation. Table 12 describes more in detail these useful methods.

The Brain class only takes the following arguments:

- **`modules`**: takes a dictionary of PyTorch modules and converts it to a PyTorch `ModuleDict`. provides a convenient way to move all parameters to the correct device, call `train()` and `eval()`, and wrap the modules in the appropriate distributed wrapper if necessary.
- **`opt_class`**: takes a function definition for a PyTorch optimizer. The reason for choosing this as input rather than a pre-constructed PyTorch optimizer is that the `Brain` class automatically handles wrapping the module parameters in distributed wrappers if requested. That needs to happen before the parameters get passed to the optimizer constructor.
- **`hparams`**: accepts a dictionary of hyperparameters that will be accessible to all the internal methods.

Table 12: Main methods implemented in the `Brain` class.

Method	Description
`fit`	Main function for training. It iterates epochs and datasets to improve the objective.
`fit_batch`	Trains a batch. It calls `compute_forward`, `compute_objectives`, and optimizes the loss.
`compute_forward`	Defines computations from input to output predictions.
`compute_objective`	Defines computations from predictions to loss.
`on_stage_start`	Gets called at the beginning of a epoch. Useful for metric initialization.
`on_stage_end`	Gets called at the end of a epoch. Useful for statistics, checkpointing, learning rate annealing.

- `run_opts`: there are a large number of options for controlling the execution details for the `fit()` method, that can all be passed via this argument. Some examples include enabling debug mode, the execution device, and the distributed execution options.
- `checkpointer`: it is a pointer to the SpeechBrain checkpointer. This way, at the beginning of training, the most recent checkpoint is loaded and training is resumed from that point. If training is finished, this moves on to evaluation. During training, the checkpoints are saved every 15 minutes by default. At the beginning of the evaluation, the "best" checkpoint is loaded, as determined by the lowest or highest score on a metric recorded in the checkpoints.

A.4.5 Lobes

In neuroscience, the lobes are areas of the brain associated with some specific high-level functionality. Similarly, in SpeechBrain we collect common higher-level speech processing functionalities in the lobe folder. For instance, lobes contain popular models used for speech processing, as reported in Table 13. Moreover, we implement here data augmentation strategies, as discussed in Table 14.

A.4.6 Inference

To make inference with pre-trained models easier, we provide some inference classes able to support a variety of speech tasks. For instance, it is possible to transcribe an input sentence using a speech recognizer with just a few lines of code:

```
from speechbrain.pretrained import EncoderDecoderASR

asr_model = EncoderDecoderASR.from_hparams(
    source="speechbrain/asr-transformer-transformerlm-librispeech",
    savedir="pretrained_models/asr")
asr_model.transcribe_file("example.wav")
>>> ["THE BIRCH CANOE SLID ON THE SMOOTH PLANKS"]
```

Listing 12: Inference with a speech recognizer.

The inference API relies on a YAML similar to that used for training. Another example for speaker verification is reported in the following:

```
from speechbrain.pretrained import SpeakerRecognition
file1= "speechbrain/spkrec-ecapa-voxceleb/example1.wav"
file2= "speechbrain/spkrec-ecapa-voxceleb/example2.wav"
verification = SpeakerRecognition.from_hparams(
    source="speechbrain/spkrec-ecapa-voxcebeb",
    savedir="pretrained_models/spkrec-ecapa-voxceleb")
score, prediction = verification.verify_files(file1,file2)
```

Listing 13: Speaker verification inference.

Table 13: Main models implemented in lobes.

Method	Main use	Description
CRDNN	Speech recognition	A combination of convolutional, recurrent, and fully-connected networks. Layer and batch normalization are used for feedforward layers. Time-pooling can be optionally used for downsampling. Users can select the type of RNN to plug in (e.g, LSTM [78], GRU [79], LiGRU [80], vanilla RNN).
TransformerASR	Speech recognition	A basic sequence-to-sequence transformer [61] for speech recognition.
ECAPA-TDNN	Speaker recognition	The ECAPA-TDNN model [33] employs a channel- and context-dependent attention mechanism, Multilayer Feature Aggregation (MFA), as well as Squeeze-Excitation (SE) and residual blocks.
X-vector	Speaker recognition	Employs a TDNN [91, 92] followed by a statistical pooling layer. It is used to compute x-vector embeddings [32].
MetricGAN	Speech enhancement	Implements a LSTM-based generator followed by a discriminator that estimates the quality of speech using PESQ.
ConvTasNet	Speech separation	Uses a linear encoder to generate a representation of the speech waveform. Speaker separation is achieved by applying a mask to the encoded representation. The mask encoded representations are then converted back to the waveforms using a linear decoder [43].
Dual-Path	Speech separation	Splits long speech inputs into smaller chunks and applies intra- and inter-chunk operations over them [44].
SepFormer	Speech separation	Couples the Dual-Path framework with an efficient multi-scale transformers approach [45].

Table 14: Data augmentation techniques implemented in lobes.

Method	Description
SpecAugment	It applies time and frequency masking as well as time warping to the input spectrum (frequency-domain implementation) [76].
Time-Domain SpecAugment	It applies time/frequency masking and time warping to the input waveform (time-domain implementation). Each disturbance is randomly activated according to the specified activation probabilities.
Environmental Corruption	It adds noise, reverberation, and bubble (i.e., noisy from many speakers talking in the background). Each corruption technique is randomly activated according to the specified activation probabilities. The amount of noise added is controlled with proper settings. When not specified directly, we use the noise and impulse responses from the OpenRIR dataset[a].

[a] `http://www.openslr.org/28/`

In this case, we feed the verification system with two audio files, and the outcome is "0" if the files are from different speakers and "1" otherwise. We have shared our best-performing models on the Hugging Face hub[9].

A.5 Experiment details

In the following, we provide more details on the datasets, evaluation metrics, and experimental settings used in the experiments reported in the paper.

[9] `huggingface.co/speechbrain`

A.5.1 Datasets

As shown in Table 1, SpeechBrain already provides recipes for several common speech corpora[10]:

- **TIMIT** [31]: The TIMIT corpus contains about 5 hours of speech from 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences. It includes audio signals sampled at 16kHz (16-bit) resolution and the phonetic transcription of each sentence using the SAMPA phoneme set. TIMIT is licensed by the Linguistic Data Consortium (LDC).
- **LibriSpeech** [28]: LibriSpeech is a corpus of approximately 1000 hours of 16kHz read English speech. The data is derived from audiobooks from the LibriVox project[11]. The volunteers gave their consent to donated their recordings to the public domain. The training data is split into three partitions of 100hr, 360hr, and 500hr sets while the dev and test data are split into the 'clean' and 'other' categories, respectively. Each of the dev and test sets is around 5hr. The corpus is publicly available with the Creative Commons Attribution 4.0 License.
- **Common Voice** [29]: Common Voice is Mozilla's initiative to create a free database for speech recognition software. The project is supported by volunteers who record sample sentences with a microphone and review recordings of other users. The website clearly informs the volunteers of the purpose of the recordings. The text is derived from different open-source text sources, including Wikipedia. As of May 2021, the dataset contains 7.3k hours of transcribed and validated speech in 60 languages. Our paper uses the latest released version of the corpus (Common Voice 6.1). The dataset is publicly available with the Creative Commons Attribution 4.0 License.
- **VoxCeleb** [35]: VoxCeleb is an audio-visual dataset consisting of short clips of human speech, extracted from interview videos uploaded to YouTube. In this paper, we used both VoxCeleb1 [34] and voxceleb2 [35]. VoxCeleb1 contains over 100,000 utterances for 1,251 celebrities. VoxCeleb2 contains over a million utterances for 6,112 identities. The dataset is available to download under a Creative Commons Attribution 4.0 International License.
- **AMI** [38]: The AMI Meeting Corpus is a widely used multi-modal dataset consisting of 100 hours of meeting recordings. The meetings have been recorded with both close-talking and far-field microphones that are time-synchronized. The meetings are majorly divided into a scenario and non-scenario meetings. In a scenario meeting, four participants play a specific role assigned to them. The non-scenario ones, instead, include a general discussion between three to four participants. The AMI dataset also has fixed official splits for various tasks to foster replicability. The signals, transcription, and annotations, have been released publicly under the Creative Commons Attribution 4.0 International Licence (CC BY 4.0).
- **Voicebank-DEMAND** [96]: It contains speech of 30 clean speakers extracted from the Voice Bank corpus [41]: 28 are included in the training set, and two are in the validation set. The noisy speech is synthesized by contaminating the clean signals with noise from Diverse Environments Multichannel Acoustic Noise Database (DEMAND) [110]. Both speakers and noise conditions in the test set are unseen by the training set. The training and test set contains 11572 and 824 noisy-clean speech pairs, respectively. The dataset is available to the community with the Creative Commons Attribution 4.0 International Public License.
- **WSJ0-mix** [46]: It is a single-channel speech separation dataset derived from the Wall Street Journal corpus (licensed by LDC). It contains mixtures of two or three speakers. The training set consists of 30 hours of overlapped speech material that was generated by randomly selecting utterances by different speakers from the WSJ0 training set *si_tr_s*, and by mixing them at various signal-to-noise ratios (SNR).

[10]The datasets used for our research are anonymized and do not contain personally identifiable information or offensive content. Datasets available through LDC require that participants consented to share their data in a corpus. Unless explicitly mentioned, we were not able to find the consent information for the other datasets. However, we only use popular corpora, and we have reason to believe that creators explicitly asked for consent from the contributors.

[11]`https://librivox.org/`

A.5.2 Evaluation metrics

SpeechBrain supports all the standard evaluation metrics needed to assess the performance of the proposed tasks. In the following, we report a short description of the evaluation metrics used in this paper:

- **Word Error Rate (WER%)**: The WER(%) is derived from the Levenshtein distance and compares a reference and a hypothesized word-level transcription. It is computed by summing up the number of word insertions, deletions, substitutions and dividing it by the total number of words in the reference transcription. Listing 14 shows an example of the WER summary provided by SpeechBrain, where the alignment between the reference and the hypothesized transcription are provided as well.
- **Phone Error Rate (PER%)**: It is the same as the WER, but it is computed using phonemes as basic units rather than words.
- **Equal Error Rate (EER%)**: It corresponds to the error rate achieved when the false acceptance rate and the false rejection rate are equal. The lower the EER is, the higher is the accuracy of the system.
- **Diarization Error Rate (DER%)**: Diarization error rate (DER) is the standard metric for evaluating speaker diarization systems. It is defined as:

$$DER = \frac{false\ alarm + missed + confusion}{reference\ length} \tag{7}$$

  where *false alarm* is the length of non-speech incorrectly classified as speech, and *missed* detection is the length of segments that are considered as speech in reference, but not in hypothesis. *confusion* is the length of segments that are assigned to different speakers in hypothesis and reference, while *reference-length* is the total duration of speech in the reference. The lower DER is, the better the diarization system is.
- **Perceptual Evaluation of Speech Quality (PESQ)**: It is a complex metric designed to predict subjective opinion scores of a degraded audio sample [111]. PESQ (full reference modality) compares the clean and noisy signals and returns a score from 4.5 to -0.5, with higher scores indicating better quality.
- **MOS predictor of overall signal quality (COVL)**: The COVL metric is part of a set of three common metrics of enhancement quality, along with CSIG and CBAK. These metrics are a composite of other commonly used metrics, like PESQ and Itakura-Saito distance measure. The resulting metric showed a much higher correlation with human judgments than any contributing metric [112].
- **Scale-invariant signal-to-noise ratio improvement (SI-SNRi)**: Scale-invariant signal-to-noise ratio improvement (SI-SNRi) is a performance metric for source separation [43], proposed as an alternative to the Source-to-Distortion Ratio [113]. It is defined as follows:

$$\begin{aligned} s_{\textbf{target}} &:= \frac{(\widehat{s}^\top s)s}{\|s\|^2}, \\ e_{\textbf{noise}} &:= \widehat{s} - s_{\textbf{target}}, \\ \text{SI-SNR} &:= 10\log_{10}\frac{\|s_{\textbf{target}}\|^2}{\|e_{\textbf{noise}}\|^2}, \end{aligned}$$

  where $s \in \mathbb{R}^T$ is the ground truth source, and $\widehat{s} \in \mathbb{R}^T$ is the source estimated by the model, and $\|s\|^2 = s^\top s$, denotes the l_2 norm operation. The scale-invariance is ensured by removing the mean from s and $\widehat{s}$, and dividing them by their respective standard deviations before calculating the SNR. Finally, SI-SNR improvement, (SI-SNRi) is calculated as follows:

$$\text{SI-SNRi} := \text{SI-SNR}(s, \widehat{s}) - \text{SI-SNR}(s, x),$$

  where $x \in \mathbb{R}^T$ denotes the mixture signal corresponding to the source s.

```
%WER 2.46 [ 1291 / 52576, 169 ins, 124 del, 998 sub ]
%SER 28.55 [ 748 / 2620 ]
Scored 2620 sentences, 0 not present in hyp.
=========================================
ALIGNMENTS

Format:
<utterance-id>, WER DETAILS
<eps> ; reference  ; on ; the ; first ;  line
  I   ;     S      ; =  ;  =  ;   S   ;   D
 and  ; hypothesis ; on ; the ; third ; <eps>
=========================================
61-70968-0058, %WER 0.00 [ 0 / 5, 0 ins, 0 del, 0 sub ]
WILL ; YOU ; FORGIVE ; ME ; NOW
 =   ;  =  ;    =    ; =  ;  =
WILL ; YOU ; FORGIVE ; ME ; NOW
=========================================
5142-33396-0000, %WER 20.00 [ 1 / 5, 0 ins, 0 del, 1 sub ]
AT ; ANOTHER ; TIME ; HARALD ; ASKED
=  ;    =    ;  =   ;    S    ;   =
AT ; ANOTHER ; TIME ; HAROLD ; ASKED
=========================================
237-134500-0005, %WER 11.11 [ 1 / 9, 0 ins, 1 del, 0 sub ]
OH ; BUT ; I'M ; GLAD ; TO ; GET ; THIS ; PLACE ; MOWED
=  ;  =  ;  =  ;  =   ; =  ;  =  ;  =   ;   =   ;   D
OH ; BUT ; I'M ; GLAD ; TO ; GET ; THIS ; PLACE ; <eps>
=========================================
260-123288-0012, %WER 14.29 [ 1 / 7, 1 ins, 0 del, 0 sub ]
THAT ; WILL ; BE ; <eps> ; SAFEST ; NO ; NO ; NEVER
 =   ;  =   ; =  ;   I   ;   =    ; =  ; =  ;   =
THAT ; WILL ; BE ;  THE  ; SAFEST ; NO ; NO ; NEVER
```

Listing 14: Excerpt of the summary file generated by SpeechBrain for the WER metric described above.

A.5.3 Experimental setups

In this section, we report more details for the experiments reported in the paper. For lower-level detail, please refer to the project repository directly[12]. The hyperparameters of the models were initially based on values reported in the literature for similar models. Then, several experiments were carried out to progressively derive better hyperparameters. We use 32GB NVIDIA V100 in our experiments. The best hyperparameters found so far are summarized in the following tables.

[12] github.com/speechbrain/speechbrain

Table 15: Main hyperparameters used in the reported LibriSpeech experiments.

Task	**Dataset**	**Technique**	**Experimental Setting**
Speech recognition	LibriSpeech	CTC+Att (RNN)	Encoder: CRDNN (2 CNNs, 4 LSTM, 1 DNN layers) Decoder: GRU (1 layer) + Beam search + LM Augmentation: yes Features: 40 fbanks Pretraining: no Dropout: 0.15 (for both encoder and decoder) Batchnorm: yes Number of epochs: 25 Batch size: 8 Learning rate: 1.0 LR scheduler: new bob Optimizer: Adam Loss: CTC+NLL CTC weight: 0.5 Number of tokens: 5000 Training Time: 5h 20m/epoch (on a NVIDIA V100)
Speech recognition	LibriSpeech	CTC+Att (Transf.)	Encoder: ContextNet (3 lay) + Transformer (12 lay) Decoder: Transformer (6 layers) + Beam search + LM Augmentation: yes Features: 80 fbanks Pretraining: no Dropout: no (for both encoder and decoder) Layernorm: yes Number of epochs: 110 Batch size: 16 Gradient accumulation: 4 Gradient clipping: 5.0 Learning rate: 1.0 Learning rate (fine tune with SGD): 0.000025 LR scheduler: new bob Optimizer: Adam Loss: CTC+NLL CTC weight: 0.4 Number of tokens: 5000 Training Time: 1h 50m/epoch (on 2 NVIDIA V100)

Table 16: Main hyperparameters used in the reported TIMIT experiments.

Task	Dataset	Technique	Experimental Setting
Speech recognition	TIMIT	CTC	Model: CRDNN (2 CNNs, 4 LiGRUs, 2 DNN layers) Augmentation: yes Features: 40 fbanks Pretraining: no Dropout: 0.15 (encoder), 0.5 (decoder) Batchnorm: yes Number of epochs: 50 Batch size: 8 Learning rate: 1.0 LR scheduler: new bob Optimizer: Adam Loss: CTC Training Time: 2m 25sec/epoch (on a NVIDIA V100)
Speech recognition	TIMIT	Transducer	Model: CRDNN (2 CNNs, 4 LiGRUs, 2 DNN layers) Augmentation: yes Features: 40 fbanks Pretraining: no Dropout: 0.15 (encoder), 0.5 (decoder) Batchnorm: yes Number of epochs: 50 Batch size: 8 Learning rate: 1.0 LR scheduler: new bob Optimizer: Adadelta Loss: Transducer Loss Training Time: 1m 10 sec/epoch (on a NVIDIA V100)
Speech recognition	TIMIT	CTC+Att	Encoder: CRDNN (2 CNNs, 4 LiGRUs, 2 DNN layers) Decoder: GRU (1 layer) + Beam search Augmentation: yes Features: 40 fbanks Pretraining: no Dropout: 0.15 (encoder), 0.5 (decoder) Batchnorm: yes Number of epochs: 20 Batch size: 8 Learning rate: 0.0003 LR scheduler: new bob Optimizer: Adam Loss: CTC+NLL Loss CTC weight: 0.2 Training Time: 2m 25 sec/epoch (on a NVIDIA V100)
Speech recognition	TIMIT	CTC+Att+ SSL	Encoder: wav2vec (Transformer) Decoder: GRU (1 layer) + Beam search Augmentation: yes Features: 40 fbanks Pretraining: wav2vec2-large-lv60 (Hugging Face) Dropout: 0.1 (encoder), 0.5 (decoder) Batchnorm: yes Number of epochs: 50 Batch size: 8 Learning rate: 0.0003 Learning rate : 0.0001 LR scheduler: new bob Optimizer: Adam Loss: CTC+NLL Loss CTC weight: 0.1 Training Time: 3m 14 sec/epoch (on a NVIDIA V100)

Table 17: Main hyperparameters used in the reported Common Voice experiments.

Task	**Dataset**	**Technique**	**Experimental Setting**
Speech recognition	Common Voice	CTC+Att	Encoder: CRDNN (3 CNNs, 5 LSTM, 2 DNN layers) Decoder: GRU (1 layer) + Beam search Augmentation: yes Features: 80 fbanks Pretraining: no Dropout: 0.15 (for both encoder and decoder) Batchnorm: yes Number of epochs: 50 Batch size: 12 Learning rate: 1.0 LR scheduler: new bob Optimizer: Adadelta Loss: CTC+NLL Loss Number of tokens: 500 CTC weight: 0.3 Training Time (En): 6h 40 min/epoch (NVIDIA V100) Training Time (Fr): 3h 20 min/epoch (NVIDIA V100) Training Time (It): 1h 00 min/epoch (NVIDIA V100) Training Time (Kw): 4h 30 min/epoch (NVIDIA V100)
Speech recognition	Common Voice	CTC+Att + SSL	Encoder: (Transformer) Decoder: GRU (1 layer) + Beam search Augmentation: yes Features: 80 fbanks Pretraining (En): 2-large-lv60 Pretraining (Fr): wav2vec2-large-100k-voxpopuli Pretraining (It): wav2vec2-large-100k-voxpopuli Pretraining (Kw): wav2vec2-large-100k-voxpopuli Dropout: 0.15 (for decoder) Batchnorm: yes Number of epochs: 30 Batch size: 12 Learning rate: 1.0 Learning rate wav2vec2: 0.0001 LR scheduler: new bob Optimizer: Adadelta Loss: CTC+NLL Loss Number of tokens: 500 CTC weight: 0.3 Training Time (En): 8h 20 min/epoch (2 NVIDIA V100) Training Time (Fr): 4h 05 min/epoch (2 NVIDIA V100) Training Time (It): 1h 30 min/epoch (NVIDIA V100) Training Time (Kw): 6h 00 min/epoch (NVIDIA V100)

Table 18: Main hyperparameters used in the Speaker Recognition and Diarization experiments.

Task	**Dataset**	**Technique**	**Experimental Setting**
Speaker recognition	Voxceleb2	x-vector + PLDA	Model: x-vector (5 TDNN layers) + statistical pool + MLP class Augmentation: yes Features: 80 fbanks Pretraining: no Dropout: no Batchnorm: yes Number of epochs: 20 Batch size: 256 Learning rate initial: 0.001 Learning rate final: 0.0001 LR scheduler: linear decay Optimizer: Adam Loss: NLL Loss Training Time (vox1+vox2): 4h 20 min/epoch (NVIDIA V100)
Speaker recognition	Voxceleb2	ECAPA-TDNN + cosine dist	Model: ECAPA-TDNN (5 tdnn layers) + att pooling + MLP class Augmentation: yes Features: 80 fbanks Pretraining: no Dropout: no Batchnorm: yes Number of epochs: 12 Batch size: 32 Learning base: 0.00000001 Learning rate max: 0.0001 LR scheduler: CyclicLRScheduler Optimizer: Adam Loss: NLL Loss Training Time (vox1+vox2): 12h 10 min/epoch (NVIDIA V100)
Speaker diarization	AMI	ECAPA-TDNN + spectral clustering	Embeddings: ECAPA-TDNN Clusteting: Spectral Clustering Split type: full_corpus_asr Skip_TNO: True Mic type: BeamformIt VAD type: oracle Max subseg dur: 3.0 Overlap: 1.5 Affinity: cos Max num spkrs: 10 Oracle # spkrs: True Ignore overlap: True Forgiveness collar: 0.25

Table 19: Main hyperparameters used for the speech enhancement experiments.

Task	**Dataset**	**Technique**	**Experimental Setting**
Speech enhancement	Voicebank-DEMAND	MimicLoss	Enhanced Model: CNN + Transformer ASR Model: CRDNN Features: Spectrogram Pretraining: no Dropout: 0.15 (CRDNN) Batchnorm: yes Number of epochs: 20 Batch size: 256 Learning rate: 0.0001 Optimizer: Adam Loss: NLL+MSE Loss Training Time (enhance): 26 min/epoch (NVIDIA V100) Training Time (perceptual): 11 min/epoch (NVIDIA V100) Training Time (ASR): 5 min/epoch (NVIDIA V100)
Speech enhancement	Voicebank-DEMAND	MetricGAN+	Enhanced Model: LSTM (2 layers) Discriminator Model: CNN (3 layers) + DNN (3 layers) Features: STFT Pretraining: no Batchnorm: yes Number of epochs: 600 Batch size: 256 Learning rate: 0.0005 Optimizer: Adam Loss: MSE + PESQ Loss Training Time: 11 min/epoch (NVIDIA V100)

Table 20: Main hyperparameters used in the speech separation experiments.

Task	Dataset	Technique	Experimental Setting
Speech separation	WSJ0-MIX	ConvTasNET	Model: ConvTasNET (Encoder, MaskNET, Decoder) Augmentation: yes Features: waveform Pretraining: no Dropout: no Normalization: GlobalLayerNorm Number of epochs: 200 Batch size: 1 Learning rate: 0.00015 LR scheduler: ReduceLROnPlateau Optimizer: Adam Loss: si-snr with pit-wrapper Training Time: 1h 00 min/epoch (NVIDIA V100)
Speech separation	WSJ0-MIX	DualPathRNN	Model: DualPathRNN (Encoder, MaskNET, inter-intra RNNs, Decoder) Augmentation: yes Features: waveform Pretraining: no Dropout: no Normalization: GlobalLayerNorm Number of epochs: 200 Batch size: 1 Learning rate: 0.00015 LR scheduler: ReduceLROnPlateau Optimizer: Adam Loss: si-snr with pit-wrapper Training Time: 3h 00 min/epoch (NVIDIA V100)
Speech separation	WSJ0-MIX	SepFormer	Model: SepFormer (Encoder, MaskNET, inter-intra Transformers, Decoder) Augmentation: yes Features: waveform Pretraining: no Dropout: no Normalization: LayerNorm Number of epochs: 200 Batch size: 1 Learning rate: 0.00015 LR scheduler: ReduceLROnPlateau Optimizer: Adam Loss: si-snr with pit-wrapper Training Time: 3h 00 min/epoch (NVIDIA V100)