Behavior Research Methods (2023) 55:4489–4501
https://doi.org/10.3758/s13428-022-02029-6

Shennong: A Python toolbox for audio speech features extraction

Mathieu Bernard[1,2] · **Maxime Poli**[1] · **Julien Karadayi**[1] · **Emmanuel Dupoux**[1,3]

Accepted: 17 November 2022 / Published online: 7 February 2023

Abstract

We introduce Shennong, a Python toolbox and command-line utility for audio speech features extraction. It implements a wide range of well-established state-of-the-art algorithms: spectro-temporal filters such as Mel-Frequency Cepstral Filterbank or Predictive Linear Filters, pre-trained neural networks, pitch estimators, speaker normalization methods, and post-processing algorithms. Shennong is an open source, reliable and extensible framework built on top of the popular Kaldi speech processing library. The Python implementation makes it easy to use by non-technical users and integrates with third-party speech modeling and machine learning tools from the Python ecosystem. This paper describes the Shennong software architecture, its core components, and implemented algorithms. Then, three applications illustrate its use. We first present a benchmark of speech features extraction algorithms available in Shennong on a phone discrimination task. We then analyze the performances of a speaker normalization model as a function of the speech duration used for training. We finally compare pitch estimation algorithms on speech under various noise conditions.

Keywords Speech processing · Features extraction · Pitch estimation · Software · Python

Introduction

Automatic processing of speech is at the heart of a wide range of applications: speech to text (Benzeghiba et al., 2007), speaker identification (Tirumala, Shahamiri, Garhwal, & Wang, 2017), emotion recognition (Koolagudi & Rao, 2012) or speaker diarization (Ryant et al., 2019; 2020). It is also applied to a variety of contexts such as multilingual models (Fer et al., 2017; Silnova et al., 2018), low-resource languages processing (Dunbar et al., 2017; 2020), pathological speech analysis (Orozco-Arroyave et al., 2016; Riad et al., 2020) or, more recently, end-to-end deep learning models (Saeed, Grangier, & Zeghidour, 2021; Zeghidour, Usunier, Synnaeve, Collobert, & Dupoux, 2018). All of those applications rely on some representation or *features* of the speech signal, i.e., a transformation of the raw audio signal which carries informative or discriminative information, usually in the time-frequency domain, that can further be processed and analyzed. Features extraction is thus the first step of most speech processing pipelines. For instance, the starting point of speaker identification systems is to extract some spectral information from the raw speech, then used for speaker modeling and discrimination (Tirumala et al., 2017). Another example is the classification of spoken sentences as statements or questions. This point can be addressed by extracting pitch – fundamental frequency – from the raw speech signal and analyzing its variations at the end of the sentences, a rise towards high frequencies at the end being an insight into whether a given sentence is a question or not (Liu, Surendran, & Xu, 2006).

Many speech features extraction software packages have been authored over time, with various implementations in different programming languages. Among them, some tools gained a wide audience. Kaldi (Povey et al., 2011) is an Automatic Speech Recognition toolkit that covers every aspect of this topic, from language modeling to decoding and features extraction. It is written in C++ and supports a collection of state-of-the-art recipes as Bash scripts. Although it is very reliable and efficient, it is hard to use and embed in third-party tools for non-technical users. Praat (Boersma, 2001) is another popular software used for speech analysis in phonetics, particularly for speech

Mathieu Bernard and Maxime Poli contributed equally to this work.

✉ Mathieu Bernard
mathieu.bernard.2@cnrs.fr

[1] Cognitive Machine Learning, PSL Research University, CNRS, EHESS, ENS, Inria, Paris, France

[2] EconomiX (UMR 7235), Université Paris Nanterre, CNRS, Nanterre, France

[3] Meta AI Research, Paris, France

annotation. Praat can be used from a graphical user interface or a custom scripting language. It includes basic spectro-temporal analysis, such as spectrogram, cochleogram, and pitch analysis. OpenSMILE (Eyben, Wöllmer, & Schuller, 2010) is another features extraction package designed for real-time processing. It focuses on audio signals but is also generic enough to be used for visual or physiological signals. Usable from command-line and wrappers in various programming languages, its generic approach makes it hard to use and configure. Finally, Surfboard (Lenain, Weston, Shivkumar, & Fristed, 2020) is a Python toolbox dedicated to speech features extraction. It is oriented toward medical applications and implements many specialized markers. OpenSMILE and Surfboard are suitable tools, but they lack general-purpose features such as speaker normalization and do not propose the fine-grained parameters Kaldi offers.

The main objective of Shennong[1] is to provide reference implementations of speech features extraction algorithms within an easy-to-use and reliable framework. By distributing such a tool to the community, our objective is to reduce the use of heterogeneous features extraction implementations in the literature and improve the replicability and comparability of studies in this domain. The Shennong toolbox relies on Kaldi (Povey et al., 2011) for most of the algorithms, thus providing the user with an accurate and efficient implementation while hiding technical details (code-source compilation, data format, pipeline scripting). On the other hand, it exposes a high-level easy-to-use Python library and command line interface. The use of Python makes it easy to integrate Shennong with machine learning tools from the Python ecosystem, such as scikit-learn (Pedregosa et al., 2011), PyTorch (Paszke et al., 2019), and Tensorflow (Abadi et al., 2016). Another design feature of Shennong is that it can be used by casual users, with provided pre-configured pipelines, and power users, being entirely customizable and easily extensible.

This paper is structured as follows. "The Shennong toolbox" section describes the speech processing algorithms available in Shennong and the architecture of the toolbox, from low-level components to high-level user interfaces. It also introduces simple usage examples using Python and the command line interface. "Applications" section exposes three applications of Shennong for speech processing. First, we benchmark the features extraction algorithms implemented in Shennong on a phoneme discrimination task. Then we analyze a speaker normalization model performance as a function of speech duration used for training. The final experiment compares three pitch estimation algorithms under different noise conditions.

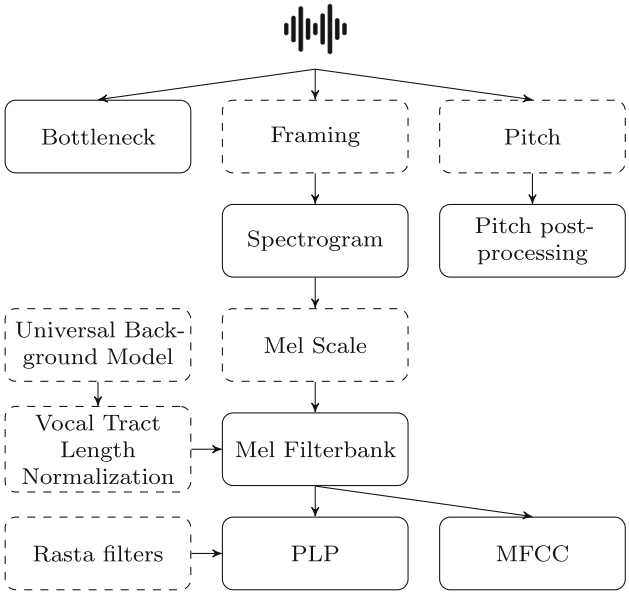

Fig. 1 Features extraction algorithms hierarchy, from speech signal to usable features (blocks in full lines) with intermediate or optional steps (blocks in dashed lines). The blocks are implemented after Kaldi (Povey et al., 2011), excepted the *Bottlneck* block after (Silnova et al., 2018), the *Pitch* block after (Kim et al., 2018; Povey et al., 2011) and the *Rasta filters* block after (Ellis, 2005). Table 1 shows the available parameters for each block. This diagram does not represent post-processing algorithms, such as Delta and Cepstral Mean Variance Normalization (see text for details)

The Shennong toolbox

We distribute the Shennong package as an open-source software[2] under a GPL3 license. It is available for Linux and macOS systems. Windows users can deploy it as a Docker image (Hung, Kristiyanto, Lee, & Yeung, 2016). It can be used as a Python library and can be integrated into third-party applications. It can also be used directly from the command line and called from bash scripts. The code follows high-quality standards regarding software development, testing, and documentation. Its modular design, inspired by the scikit-learn toolbox (Pedregosa et al., 2011), makes it easily extensible to new extraction algorithms (see "Low-level software architecture" section). We planned to extend the toolbox in the future, with new algorithms such as Contrastive Predictive Coding (Oord, Li, & Vinyals, 2018) and Voice Activity Detection (Ramirez, Górriz, & Segura, 2007). Because it is an open-source project, the code is also opened to users contributions. This paper is based on version 1.0.

[1] We named Shennong after the so-called Chinese Emperor that popularized tea according to Chinese Mythology. It is a reference to Kaldi, a speech recognition toolkit on which Shennong is built, and a legendary Ethiopian goatherd who discovered the coffee plant.

[2] https://github.com/bootphon/shennong

Implemented algorithms

Figure 1 presents the algorithms available in version 1.0 of Shennong. Most of them are implemented after Kaldi (Povey et al., 2011), using the pykaldi Python wrapper (Can, Martinez, Papadopoulos, & Narayanan, 2018). The complete set of parameters from the original implementations are provided with the algorithms, along with the default values given by their original authors, as detailed in Table 1. All the implemented algorithms have been extensively tested. The Shennong tests suite thus covers 99% of the source code and includes tests replicating the results of original implementations on a sample speech signal. The remaining section introduces those algorithms, for which "Phone discrimination task" section provides a benchmark.

Short-term spectro-temporal methods are commonly used for the extraction of speech features. Shennong includes Spectrogram, Mel-Filterbank, Mel-Frequency Cepstrum Coefficients (MFCC), and Perceptual Linear Predictive filters (PLP). The common point of those methods is estimating the power spectrum from overlapping frames extracted from the raw speech signal. This power spectrum, along with the signal energy and optionally expressed in the log domain, is used to generate the Spectrogram features. The Mel Filterbank is then obtained by applying a Mel scale to the power spectrum. Finally, MFCC and PLP are obtained with further processing in the cepstral domain. Rasta filters are optional bandpass filters that can be applied to PLP features (Hermansky, 1990; Hermansky & Morgan, 1994; Hermansky, Morgan, Bayya, & Kohn, 1991), to make them more robust to linear spectral distortions due to the communication channel.

Vocal Track Length Normalization (VTLN) (Kim, Umesh, Gales, Hain, & Woodland, 2004; Povey, 2010)is a normalization technique used to reduce inter-speaker variability. It can be applied to Mel-based representations, namely Mel Filterbank, MFCC, and PLP features. It consists of a model-based estimation of speaker-specific linear transforms of the power spectrum that scale the Mel filters center frequencies and bandwidths. A Universal Background Model (UBM) must be trained to estimate a VTLN warp coefficient per speaker, which is then applied to normalize the features. The training is unsupervised and does not require any annotation or phonetic transcription. "Phone discrimination task" section demonstrates the effectiveness of VTLN for inter-speaker phone discrimination, and "VTLN model training" section provides a study on the amount of data required to train a VTLN model.

The Bottleneck features (Fer et al., 2017; Silnova et al., 2018) rely on convolutional neural networks pre-trained for phone recognition. Three networks are available: monophone and triphone states, trained on US English from the Fisher dataset (Cieri, Miller, & Walker, 2004), and a multilingual triphone states network trained on 17 languages from the Babel dataset (Harper, 2013).

Shennong also implements two algorithms for pitch estimation. The first one from Kaldi (Ghahremani et al., 2014) is based on the normalized cross-correlation of the input signal. It outputs a pitch estimate and a probability of voicing for each frame. The second algorithm is Convolutional REpresentation for Pitch Estimation (CREPE) (Kim, Salamon, Li, & Bello, 2018) and is based on a convolutional neural network pre-trained on music datasets (Kim et al., 2018; Mauch & Dixon, 2014). We made the CREPE algorithm fully compatible with the Kaldi one by turning the maximum of the network activation matrix into a probability of voicing and interpolating pitch for frames with low confidence. Finally, a post-processing step, common to both algorithms, normalizes the pitch estimates, converts them to log domain, and extracts their first-order derivative. "Pitch estimation" section compares those algorithms under various noise conditions.

Finally, Shennong also provides post-processors that normalize or add information on extracted features. Delta computes the n^{th} order derivative of any features. Voice Activity Detection (VAD) is a simple energy-based method that makes binary decisions, mainly used to filter out silences. Cepstral Mean Variance Normalization (CMVN) normalizes features to a zero mean and unitary variance, and can be applied on a per-frame, per-utterance, or per-speaker basis.

Low-level software architecture

Shennong is built on a few low-level components, namely Python classes, that users can use to configure and run a features extraction pipeline.

The `Audio` class is the interface with raw audio data and is the input of all pipelines implemented in Shennong. It is used to load audio files as NumPy arrays, resample and manipulate them. It supports multiple audio file formats such as WAV or FLAC. The `Utterances` class provides a high-level view of speech fragments as it handles a collection of `Audio` instances, each one with an attached identifier, speaker information, and optional onset and offset times.

The `Features` class is the output returned by processing algorithms. It stores three attributes: a data array, a time array, and some properties. Data is a NumPy array of shape $[m, n]$ with m being the number of frames on the temporal axis and n being the dimension of the features, usually along the frequency axis. The time array stores the timestamps of each frame either as a single value corresponding to the central time of each frame, with a

Table 1 Parameters of the features extraction algorithms implemented in Shennong

Algorithm	Parameter	Default	Comment
Bottleneck	`weights`	`BabelMulti`	Pretrained network to use, in `FisherMono`, `FisherMulti` or `BabelMulti`
	`dither`	0.1	Amount of dithering to add
Framing	`sample_rate`	16000	Sampling frequency in Hz
	`frame_shift`	0.01	Frame shift in second
	`frame_length`	0.025	Frame length in second
	`dither`	0.1	Amount of dithering to add
	`preemph_coeff`	0.97	Signal preemphasis coefficient
	`remove_dc_offset`	`True`	Whether to subtract mean on each frame
	`window_type`	`povey`	Window to use in `hamming`, `hanning`, `povey`, `rectangular` or `blackman`
	`snip_edges`	`True`	If true, output only frames that completely fit in the input signal
Spectrogram	*all from Framing plus...*		
	`energy_floor`	0.0	Absolute floor on energy
	`raw_energy`	`True`	When true, compute energy before preemphasis and windowing
Mel Scale	*all from Framing plus...*		
	`num_bins`	23	Number of triangular mel-frequency bins
	`low_freq`	20	Low cutoff frequency for mel bins in Hz
	`high_freq`	0	High cutoff frequency for mel bins in Hz
	`vtln_low`	100	Low inflection point in VTLN in Hz
	`vtln_high`	-500	High inflection point in VTLN in Hz
Filterbank	*all from Mel Scale plus...*		
	`use_energy`	False	Add an extra dimension with energy to the filterbank output
	`energy_floor`	0.0	Absolute floor on energy
	`raw_energy`	`True`	When true, compute energy before preemphasis and windowing
	`use_log_fbank`	`True`	Whether to produce log or linear filterbank
	`use_power`	`True`	Whether to use power or magnitude
MFCC	*all from Mel Scale plus...*		
	`num_ceps`	13	Number of cepstra, including C0
	`use_energy`	False	Add an extra dimension with energy to the filterbank output
	`energy_floor`	0.0	Absolute floor on energy
	`raw_energy`	`True`	When true, compute energy before preemphasis and windowing
	`cepstral_lifter`	22.0	Constant that controls scaling of MFCCs
PLP	*all from Mel Scale plus...*		
	`rasta`	`False`	Whether to do RASTA filtering
	`lpc_order`	12	Order of LPC analysis
	`num_ceps`	13	Number of cepstra, including C0
	`use_energy`	False	Add an extra dimension with energy to the filterbank output
	`energy_floor`	0.0	Absolute floor on energy
	`raw_energy`	`True`	When true, compute energy before preemphasis and windowing
	`compress_factor`	1/3	Compression factor
	`cepstral_lifter`	22.0	Constant that controls scaling of PLPs
	`cepstral_scale`	1.0	Cepstral constant in PLP computation

Table 1 (continued)

Algorithm	Parameter	Default	Comment
Pitch	`sample_rate`	16000	Sampling frequency in Hz
(Kaldi algorithm)	`frame_shift`	0.01	Frame shift in second
	`frame_length`	0.025	Frame length in second
	`min_f0`	50	Minimum F0 to search for in Hz
	`max_f0`	400	Maximum F0 to search for in Hz
	`soft_min_f0`	10	Minimum F0 to search for in Hz, applied in soft way
	`penalty_factor`	0.1	Cost factor for F0 change
	`lowpass_cutoff`	1000	Cutoff frequency for low-pass filter in Hz
	`resample_freq`	4000	Downsampling frequency in Hz
	`delta_pitch`	0.005	Smallest relative change in pitch that the algorithm measures
	`nccf_ballast`	7000	Increasing this factor ensure pitch continuity in unvoiced regions
Pitch	`model_capacity`	`full`	Pretrained model to use, in `tiny`, `small`, `medium`, `large` or `full`
(CREPE algorithm)	`frame_shift`	0.01	Frame shift in second
	`frame_length`	0.025	Frame length in second
	`viterbi`	`True`	Whether to apply Viterbi smoothing to the estimated pitch curve
	`center`	`True`	Whether to center the window on the current frame
Pitch	`pitch_scale`	2.0	Scaling factor for the final normalized log-pitch value
(post-processing)	`pov_scale`	2.0	Scaling factor for final probability of voicing feature
	`delta_pitch_scale`	10.0	Term to scale the final delta log-pitch feature
	`delta_pitch_noise_stddev`	0.005	Standard deviation for noise we add to the delta log-pitch
	`delta_window`	2	Number of frames on each side of central frame
	`delay`	0	Number of frames by which the pitch information is delayed
Universal	`num_gauss`	64	Number of Gaussians in the model
Background Model	`num_iters`	4	Number of training iterations
	`initial_gauss_proportion`	0.5	Proportion of Gaussians to start with in initialization phase
	`num_iters_init`	20	Number of E-M iterations for model initialization
	`num_frames`	5.10^5	Maximum num-frames to keep in memory for model initialization
	`min_gaussian_weight`	10^{-4}	Minimum weight below which a Gaussian is not updated
	`remove_low_count_gaussians`	`False`	Remove Gaussians with a weight below `min_gaussian_weight`
Vocal Tract Length	*all from UBM plus...*		
Normalization	`num_iters`	15	Number of training iterations
	`min_warp`	0.85	Minimum warp considered
	`max_warp`	1.15	Maximum warp considered
	`warp_step`	0.01	Warp step
	`logdet_scale`	0.0	Scale on log-determinant term in auxiliary function
	`norm_type`	`offset`	Type of fMMLR applied, in `offset`, `none` or `diag`

Zero or negative frequencies are relative to the Nyquist frequency

shape $[m, 1]$ or as a pair of onset/offset times with a shape $[m, 2]$. Several `Features` instances sharing the same time values can be concatenated over the frequency axis to obtain composite data within the same array, e.g. MFCC and pitch. Finally, the properties record details of the extraction pipeline, such as the name of the input audio file and processing parameters values.

The `Features` class is designed to store a single matrix corresponding to a single `Audio` object. Several `Features` are usually grouped into a `FeaturesCollection`, for instance, to manage a whole dataset represented as an `Utterances` easily. This class indexes `Features` by name and allows saving and loading features to/from various file formats (see Table 2). The *pickle* format is the native Python one. It is very fast in writing and reading times and should be the preferred format for small to medium datasets. The *h5features* format (Schatz, Bernard, & Thiollière, 2020) is specifically

Table 2 File formats supported by Shennong for reading and writing a `FeaturesCollection`

Format	File size	Write time	Read time
pickle	883 MB	0:00:07	0:00:05
h5features	873 MB	0:00:21	0:00:07
numpy	869 MB	0:02:30	0:00:22
matlab	721 MB	0:00:59	0:00:11
kaldi	1.3 GB	0:00:06	0:00:07
csv	4.8 GB	0:03:02	0:03:11

The read/write times and file size have been obtained on MFCC features computed on the Buckeye English Corpus (Pitt et al., 2007) (40 speakers, about 38 hours of speech in 254 files) using a Linux machine with an Intel Xeon CPU, 16 GB RAM, and an SSD hard drive

designed to handle extensive datasets, as it allows partial writing and reading of data larger than RAM. The formats *numpy*, *matlab* and *kaldi* propose compatibility layers to those respective tools. Finally, the *csv* format stores features into plain text CSV files, one file per `Features` in the collection, along with the features properties in JSON format.

The `Processor` class abstracts the features extraction algorithms (see Fig. 1). Therefore, all algorithms implemented in Shennong expose a homogeneous interface to the user: the parameters are specified in the constructor, and a `process()` method takes `Audio` or `Features` as input and returns `Features`. A generic method `process_all()` is also provided to compute features from a whole `Utterances` in a single call, using parallel jobs and returning a `FeaturesCollection`.

High-level extraction pipeline

The modular design described above allows the creation of arbitrary pipelines involving multiple steps, such as features extraction, pitch estimation, and normalization. To simplify the use of such complex pipelines, Shennong exposes a high-level interface made of three steps, which can be used from Python using the `pipeline` module or from the command-line using the `speech-features` program.

The first step is to define a list of utterances on which to apply the pipeline, as a list of audio files, with optional utterances name, speaker identification, and onset/offset times. The second step is configuring the extraction pipeline by selecting the extraction algorithms. This step generates a configuration with default parameters, which the user can further edit. The third and final step is to apply the configured pipeline to the defined utterances. Figure 2 illustrates two use-cases: the use of the low-level API to extract MFCCs on a single file (Fig. 2a) and the use of a high-level pipeline to extract both MFCCs and pitch on three utterances from two speakers, from the Python API (Fig. 2b) and command line (Fig. 2c).

Applications

This section illustrates the use of Shennong on three experimental setups: a benchmark of features extraction algorithms available in Shennong on a phone discrimination task, an analysis of the VTLN model performance as a function of speech duration used for training, and a comparison of pitch estimation algorithms on various noise conditions. After having detailed the background and motivation for each experimental setup, the remainder of this section presents their methods and discusses the obtained results. The code to replicate those experiments is distributed with Shennong[3] and can be used as introductory material by new users, along with the toolbox documentation.

Phone discrimination task

The goal of speech recognition systems is to decode words from raw speech. They must rely on a representation of speech sounds that is robust to within- and across-talker variations, thus supporting the identification of phones, syllables and words. For such applications, it is critical for the extracted features to allow for the classification of speech frames into phonetic categories. This experiment compares the discriminative power of the features extraction algorithms available in Shennong on a phone discrimination task, within- and across-talkers, in two languages.

Methods

This experiment replicates the sub-word modeling task of the Zero Speech Challenge 2015 (Versteegh, Anguera, Jansen, & Dupoux, 2016; Versteegh et al., 2015), using the same dataset and evaluation pipeline. The dataset is composed of curated segments from two free, open access, and

[3] https://github.com/bootphon/shennong/tree/v1.0/examples

```python
from shennong import Audio, FeaturesCollection
from shennong.processor import MfccProcessor

# load the input WAV file
audio = Audio.load('test.wav')

# extract MFCCs with default parameters
mfcc = MfccProcessor().process(audio)

# save the features as a numpy .npz file
FeaturesCollection({'mfcc': mfcc}).save('mfcc.npz')
```

(a) MFCC extraction in Python, using the low-level API.

```python
from shennong import pipeline

# generate a pipeline configuration with MFCC and pitch from Kaldi
# (user can then edit parameters in config)
config = pipeline.get_default_config('mfcc', with_pitch='kaldi')

# defines three utterances from two speakers
utterances = [
    ('utterance1', '/path/to/wav1.wav', 'speaker1'),
    ('utterance2', '/path/to/wav2.wav', 'speaker1'),
    ('utterance3', '/path/to/wav3.wav', 'speaker2')]

# apply the configured pipeline on the utterances, run on 3 CPU cores
# and save the extracted features to a numpy format
pipeline.extract_features(config, utterances, njobs=3).save('features.npz')
```

(b) MFCC and pitch extraction in Python, using an extraction pipeline.

```bash
# generate a pipeline configuration with MFCC and pitch from Kaldi
# (user can then edit parameters in config.yaml)
speech-features config mfcc --pitch kaldi -o config.yaml

# defines three utterances from two speakers
echo "utterance1 /path/to/wav1.wav speaker1" > utterances.txt
echo "utterance2 /path/to/wav2.wav speaker1" >> utterances.txt
echo "utterance3 /path/to/wav3.wav speaker2" >> utterances.txt

# apply the configured pipeline on the utterances, run on 3 CPU cores
# and save the extracted features to a numpy format
speech-features extract --njobs 3 config.yaml utterances.txt features.npz
```

(c) MFCC and pitch extraction from command line, using an extraction pipeline.

Fig. 2 Examples of use of Shennong. In (a) MFCC are extracted and saved from an input audio file. The features have 13 dimensions, the default number of Mel coefficients. In (b) and (c), a pipeline is used for MFCC and pitch extraction on three utterances from two speakers, the two scripts in Python and bash being strictly equivalent and giving the same result. For each utterance, the extracted features have 16 dimensions: 13 for MFCC and 3 for pitch estimates

annotated speech corpora: the Buckeye Corpus of American English (Pitt et al., 2007) (12 speakers, 10h34m44s) and the NCHLT Speech Corpus of Xitsonga (De Vries et al., 2014), a low resource Bantu language spoken in southern Africa (24 speakers, 4h24h37s). The English corpus contains spontaneous, casual speech, whereas the Xitsonga corpus contains read speech constructed out of a small vocabulary, tailored for producing speech recognition applications. The original recordings were segmented into short files that contain only clean speech, i.e. no overlap, pauses, or nonspeech noises, and contain only the speech of a single speaker. The gold phone-level transcriptions have been obtained from a forced alignment using Kaldi (Versteegh et al., 2015).

The evaluation of phone discriminability uses a minimal pair ABX task, a psychophysically inspired algorithm that only requires a notion of distance between the representations of speech segments (Schatz, Feldman, Goldwater, Cao, & Dupoux, 2021; Schatz et al., 2013; 2014). The ABX discriminability, for example, between

[*apa*] and [*aba*], is defined as the probability that the representations of A and X are more similar those of B and X, overall triplets of tokens such that A and X are tokens of [*aba*] and B a token of [*apa*]. The discriminability is evaluated *within* speakers, where A, B, and X are uttered by the same speaker, and *across* speakers, such that X is emitted by a different speaker than A and B. The global ABX phone discriminability score aggregates over the entire set of minimal triphone pairs such as ([*aba*], [*apa*]) to be found in the dataset. The metric used for ABX evaluation is the Dynamic Time Warping divergence using the cosine distance as the underlying frame-level metric.

We consider the following algorithms: spectrogram, filterbank, MFCC, PLP, RASTA-PLP, and multilingual bottleneck network. All the algorithms are used with default parameters. Each algorithm is declined in three pipeline configurations. The raw features alone are first considered, noted as *raw* in Table 3, and of dimension n. Then the concatenation of the raw features with their first and second-order derivatives, along with pitch estimates, are used and noted $+\Delta/F0$, giving a dimension $3n + 3$. The cross-correlation pitch estimation algorithm from Kaldi is used. It outputs three channels: the probability of voicing, the normalized log pitch, and the raw pitch derivative. Finally, CMVN is applied on a per-speaker basis on the $+\Delta/F0$ configuration, giving a zero mean and unitary variance on each channel independently, and is noted as $+CMVN$. Furthermore, a VTLN model is trained on 10 minutes of speech per speaker for each of the two corpora and is applied to the spectrogram, filterbank, MFCC, PLP, and RASTA-PLP, for each of the three pipeline configurations.

Results

Experimental results are presented in Table 3. First, considering the overall scores, the bottleneck deep neural network largely outperforms the spectro-temporal algorithms in every configuration. We expected those results as the bottleneck model is trained for phone discrimination. Among the spectro-temporal algorithms, the filterbank model performs very well and reaches the best score on seven over eight configurations, except on English across speakers with VTLN. This result is unexpected and has to be underlined, as it beats MFCC, which is by far the most used algorithm in the literature.

Considering now the impact of raw, $+\Delta/F0$, and $+CMNV$ pipelines for the different algorithms, it is demonstrated that adding pitch, deltas, and CMVN to raw features are beneficial for both MFCC, PLP, and Rasta-PLP in all configurations, except for the bottleneck algorithm. Spectrogram and filterbank algorithms benefit from pitch and deltas as well, but the addition of CMVN degrades

Table 3 Comparison of features extraction algorithms available in Shennong on a phone discrimination task, within and across speakers, with and without VTLN, for English and Xitsonga datasets

Algorithm	Within speakers						Across speakers					
	without VTLN			with VTLN			without VTLN			with VTLN		
	raw	+Δ/F0	+CMNV	raw	+Δ/F0	+CMNV	raw	+Δ/F0	+CMNV	raw	+Δ/F0	+CMNV
(a) ABX scores for English												
Spectrogram	16.7	15.2	20.2	–	–	–	30.3	27.9	29.7	–	–	–
Filterbank	12.8	**11.6**	18.2	12.6	**11.4**	18.1	24.9	**22.1**	26.5	23.2	20.7	25.4
MFCC	13.0	12.5	12.4	12.8	12.3	12.0	27.2	26.4	24.0	23.4	22.7	20.0
PLP	12.5	12.4	11.9	12.5	12.4	11.9	28.0	26.6	23.8	24.7	23.5	**19.7**
Rasta-PLP	14.3	14.2	12.5	14.2	14.1	12.5	28.5	26.8	25.3	24.6	23.6	21.3
Bottleneck	**8.5**	8.5	8.6	–	–	–	**12.5**	12.5	12.5	–	–	–
(b) ABX scores for Xitsonga												
Spectrogram	19.2	16.8	19.2	–	–	–	34.6	32.0	26.5	–	–	–
Filterbank	13.8	**11.7**	15.2	13.6	**11.4**	15.2	28.1	25.1	**21.5**	26.9	24.0	**20.7**
MFCC	17.1	16.2	14.6	17.5	16.5	14.6	33.6	32.8	26.0	31.4	30.6	22.7
PLP	16.2	14.6	14.0	16.2	14.7	14.2	33.5	31.2	26.2	31.7	29.5	22.2
Rasta-PLP	13.7	12.5	12.3	13.5	12.2	12.0	27.9	25.2	23.9	25.0	22.8	21.7
Bottleneck	**6.9**	7.0	7.3	–	–	–	**9.5**	9.6	9.6	–	–	–

Scores are ABX error rates in % (random score is 50%). The *raw* configuration is based on raw features alone. The $+\Delta/F0$ adds first/second order derivatives and Kaldi pitch estimates. The *+CMVN* adds a CMVN normalization by speaker on top of $+\Delta/F0$. VTLN is not available for spectrogram and bottleneck features. The best scores for each configuration are in bold font

the ABX score, with the exception of the Xitsonga across speakers configuration. This is expected as CMVN is tailored towards the cepstral domain, but spectrogram and filterbank are in the spectral domain. Rasta filtering on PLP gives different results across languages: it degrades the score in English but improves them on Xitsonga. RASTA filtering is used to reduce distortions from the communication channel (Hermansky & Morgan, 1994), so this difference can be explained by the recording conditions of the two corpora.

The use of VTLN for speaker normalization improves both MFCC and PLP scores by about 4% on the across speakers context, whereas filterbank gains about 1%. No or slight improvement is attested within speakers for all the algorithms. Those results are expected because ABX scores are computed on a single speaker, but VTLN normalizes features across speakers.

Finally, our results are in line with those from the subword modeling task of the Zero Speech 2015 challenge.[4] Indeed, the challenge baseline used raw MFCCs from Kaldi and obtained an error rate 1.4% higher than ours (mean over the two languages and within/across conditions). Since the data and the features extraction algorithm are the same, this small difference is explained by improvements and fixes in the ABX evaluation code since the release of the challenge in 2015.

VTLN model training

Vocal Tract Length Normalization (VTLN) is used to reduce the variability of individual voices in the features space. "Phone discrimination task" section has shown that VTLN significantly improves the phone discriminability score in the across-speakers context. Nevertheless, we trained the VTLN model on 10 minutes of speech per speaker without further justification. This section thus explores the influence of the amount of speech duration used for VTLN training on the resulting VTLN coefficients and phone discriminability scores. To the best of our knowledge, there is no such experiment available in the literature. The same dataset and evaluation pipeline that in "Phone discrimination task" section are used here.

Methods

The same segment of the Buckeye English corpus as in "Phone discrimination task" section is used. It is composed of 10h34m44s of speech balanced across 12 speakers. In order to train several VTLN models on variable speech duration, this corpus is split into sub-corpora containing a given speech duration per speaker. The considered durations are 5s, 10s, 20s, 30s, 60s, and up to 600s by steps of 60s. The subcorpora are built without overlap: the first blocks of fixed duration for each speaker are joined together, then for the second blocks, etc. This gives a total of 1010 corpora, from 479 for 5s per speaker to 2 for 600s per speaker, following a power law. For each of those corpora, a VTLN model is trained using the default parameters, and VTLN coefficients are extracted.

Then MFCC features are extracted from those corpora, using default parameters, and normalized with their associated VTLN coefficients. MFCC features are declined over the 3 pipeline configurations raw, $+\Delta/F0$ and $+CMVN$, as detailed in "Phone discrimination task" section. The ABX discriminability score is then computed across speakers as before. To mitigate the computational cost, a maximum of 10 corpora per duration are randomly sampled and considered for MFCC extraction and ABX scoring. Moreover, MFCC and ABX are computed without VTLN and with a VTLN model trained on the entire corpus, giving 102 discriminability scores for all the considered durations.

Results

Figure 3 shows the evolution of the VTLN coefficients for different speakers as a function of the amount of speech per speaker used for training. With 300s per speaker, or 1h of speech in total, the coefficients have largely converged and remain overall stable when more data is added for training. This demonstrates that training a VTLN does not require a large amount of data, thus reducing the computational needs and training time. Moreover, the differentiation comes very

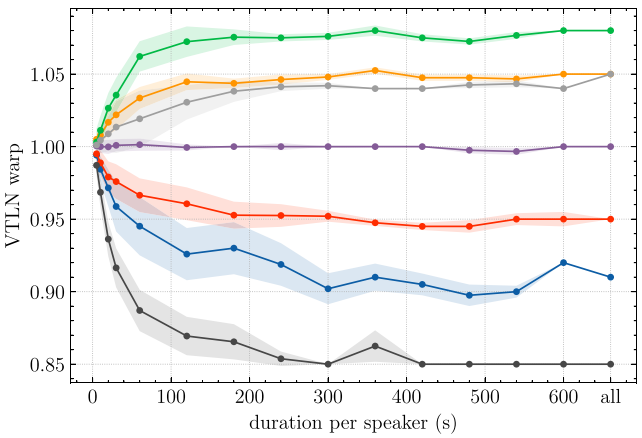

Fig. 3 Average VTLN coefficients for different speakers according to the speech duration per speaker used for VTLN training. The lines correspond to coefficients of 7 representative speakers out of 12, with VTLN warps spanning from 0.85 to 1.08 after convergence. The remaining five speakers, not displayed here for clarity, show similar and overlapping patterns to the lines shown. The shaded areas are standard deviations

[4] https://zerospeech.com/tracks/2015/results

early: with 30s of speech per speaker only, the VTLN coefficients are already clustered.

Figure 4 shows the ABX error rate obtained on MFCC features without VTLN normalization and with VTLN computed using different speech durations per speaker. First considering the scores obtained without VTLN and with VTLN trained on the whole dataset, results match those displayed in Table 3: in raw configuration the scores go from 27.2% to 23.4%, from 26.4% to 22.7% for $+\Delta/F0$ and from 24.0% to 20.0% for $+CMVN$. The three configurations follow the same tendency and rapidly converge to a nearly optimal score, starting with 60s of speech per speaker for VTLN training. Consolidating from results on Fig. 3, it is shown here that the VTLN coefficients do not need to have fully converged to yield a close to optimal normalization. This conclusion has to be underlined because researchers usually train a VTLN model on all the available data, as for instance in VTLN-based Kaldi recipes.

Pitch estimation

Shennong includes two pitch estimators. The Kaldi algorithm performs an auto-correlation of the speech signal and the CREPE one is a deep neural network trained on music datasets. In order to quantify the capacity of CREPE to generalize from music to speech, this section compares pitch estimation algorithms on speech, under various noise conditions. We also compare Kaldi and CREPE algorithms from Shennong to two popular alternatives for speech pitch estimation: PRAAT and YAAPT.

Methods

The KEELE Pitch Database (Plante, Meyer, & Ainsworth, 1995) is used for evaluation. It consists of approximately 6 minutes of clean speech with pitch estimates, separated into ten phonetically balanced sentences by five male and five female speakers. The pitch is estimated from the auto-correlation of a laryngograph signal using frames of 25.6 ms with a 10 ms overlap. As noise robustness is key to applications with real-world data, the KEELE dataset has been corrupted by additive noise at seven signal-to-noise ratios (SNR) ranging from -15 dB to 15 dB. We consider two kinds of noise: white Gaussian noise and babble noise, which consist of a recording of a restaurant ambiance.

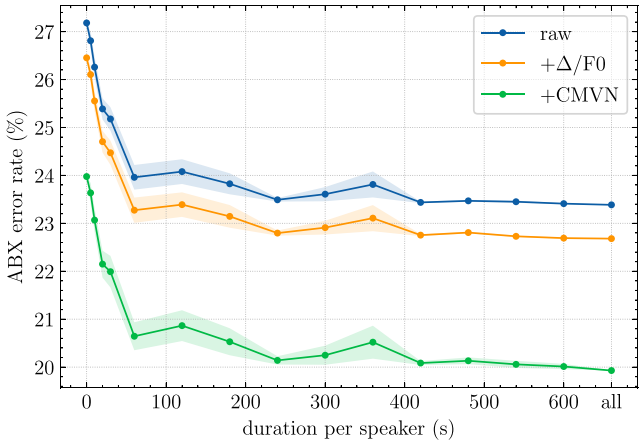

Fig. 4 ABX error rate obtained for MFCC features across speakers on English, for three pipeline configurations, and various speech duration per speaker used for VTLN training. The shaded areas are standard deviations

The Kaldi and CREPE pitch estimators from Shennong (see "Implemented algorithms" section) are compared with two other popular models: the Praat algorithm (Boersma, 1993; 2001), which uses an auto-correlation method, and the YAAPT algorithm (Zahorian & Hu, 2008) based on a combination of time and frequency domain processing using normalized cross-correlation. To match the gold pitch estimates from the KEELE dataset, the Kaldi, CREPE, and YAAPT algorithms are parameterized to use frames of 25.6 ms with a 10 ms overlap. The Praat algorithm does not support frame parametrization, so its estimates have been linearly interpolated to match the gold timestamps.

Finally, a significant amount of frames is estimated as unvoiced on clean speech: only 50.3% of the KEELE gold estimates are valid pitches. Other values indicate an absence of voiced speech or a corrupted laryngograph signal. The algorithms as well estimate some frames as unvoiced. This is detected by a zero pitch estimate for Praat and YAAPT models, or by low confidence for Kaldi and CREPE. To avoid estimation biases, the union of all the frames classified as unvoiced within the KEELE dataset and by the 4 algorithms on clean speech is removed from the evaluation. This leads to 36.3% of the total number of frames being conserved for the evaluations at different SNRs.

We consider two performance measures. The Mean Absolute Error (MAE) is the mean of the absolute error between the pitch estimates and the ground truth. The Gross Error Ratio (GER) is the proportion of pitch estimates that differ from more than 5% from the ground truth. Thus, given a speech signal with n frames, $x \in \mathbb{R}^n$ its ground truth and $\tilde{x} \in \mathbb{R}^n$ its pitch estimates for each frame, the MAE and GER metrics are expressed as follows:

$$\mathrm{MAE}(\tilde{x}, x) = \frac{1}{n}\sum_{i=1}^{n} |\tilde{x}_i - x_i|, \tag{1}$$

$$\mathrm{GER}(\tilde{x}, x) = \frac{100}{n}\sum_{i=1}^{n} \mathbb{1}_{|\tilde{x}_i - x_i| > 0.05 x_i}, \tag{2}$$

where $\mathbb{1}_{p(.)}$ is 1 when the predicate $p(.)$ is true and 0 otherwise.

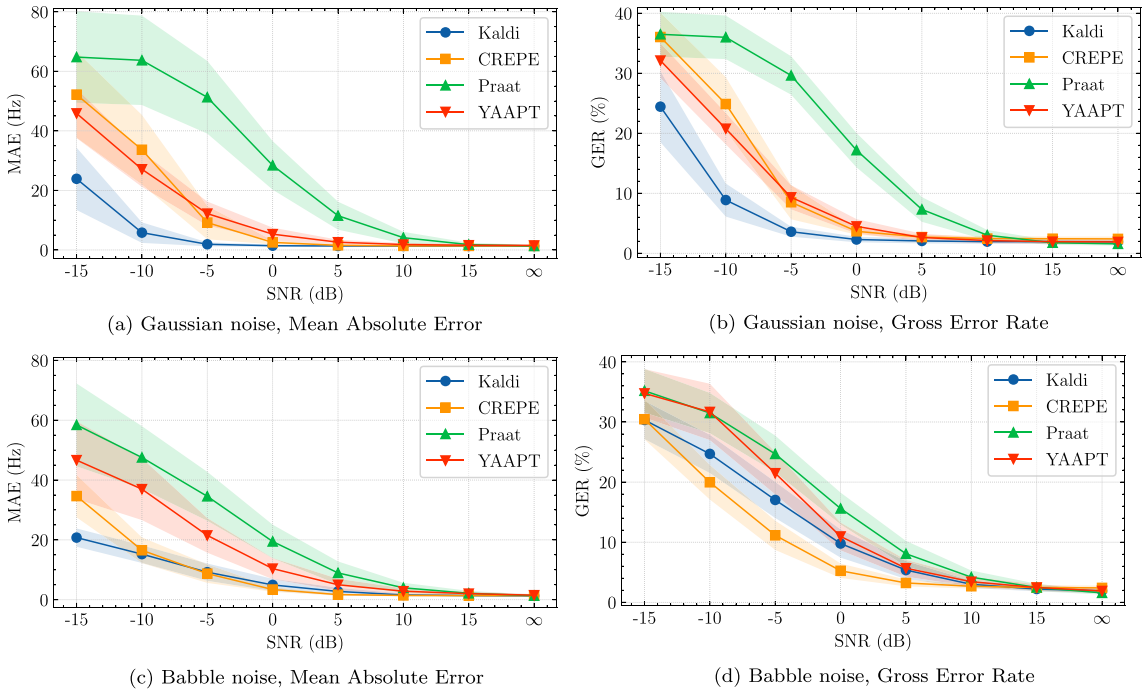

Fig. 5 Pitch estimation error for the Kaldi, CREPE, Praat, and YAAPT algorithms using Mean Absolute Error and Gross Error Rate on the KEELE dataset. White Gaussian noise or babble noise is added at various signal-to-noise ratios. Lines are mean errors over all the ten speakers, and shaded areas are standard deviations

Results

Figure 5 shows the evaluation error obtained for the four algorithms and the two kinds of noises at the considered SNR for both MAE and GER metrics. Considering first the errors obtained on Gaussian noise (Fig. 5a and b), MAE and GER follow similar patterns for all the algorithms. When there is little noise, all models obtain a low error that is stable across speakers. The Praat algorithm is the first to have degraded performances, starting at 10 dB, where CREPE and YAAPT start at 5 dB. The Kaldi algorithm is robust against this noise, with a stable error up to -5 dB. CREPE and YAAPT have similar errors across SNR, with CREPE being more robust to noise up to -5 dB and YAAPT below -5 dB. When considering the effect of additive babble noise (Fig. 5c and d), the algorithms performance starts to decrease at 10 dB. Across the SNR range, Kaldi and CREPE performances are very close and give the lowest errors. CREPE is more reliable with the lowest GER, excepted at -15 dB, where Kaldi performs better. YAAPT and Praat do not perform well on babble noise and have high error rates and standard deviations.

Overall, the four estimators have equivalent performances on clean speech signals. Kaldi, CREPE, and YAAPT perform better on Gaussian noise than babble noise, YAAPT being the most sensitive to the latter. Praat is the only algorithm with an increase in performance on babble noise. Finally, both CREPE and Kaldi appear to be more reliable estimators than Praat and YAAPT, with Kaldi being more robust to Gaussian noise and CREPE to babble noise. The PRAAT estimator appears to be the less reliable estimator under noise, both for Gaussian and babble noises. The good performances of CREPE have to be emphasized as it is trained initially on musical signals but demonstrates good generalization to speech.

Discussion

This paper introduced Shennong, an open-source Python package for audio speech features extraction. The toolbox covers many well-established state-of-the-art algorithms, primarily implemented after Kaldi. Shennong's software architecture and components focus on ease of use, reliability, and extensibility. The main benefit of Shennong is for non-technical users who need to extract speech features from an algorithm available in Kaldi. Shennong hides the complexity inherent to Kaldi and exposes all the features-related algorithms and parameters in a clear and consistent way. Compared to other available toolboxes, Shennong is specialized on speech processing and, as such, provides advanced functionalities such as Rasta filtering or VTLN out-of-the-box, as well as an extensive set of parameters for each algorithm. Finally, Shennong covers different use cases: few lines of code are sufficient to configure and apply a complex extraction pipeline, but

power-users can benefit from the Python API to hand-tune any part of the pipeline or integrate Shennong in their projects.

Three experiments on speech features extraction using the Shennong toolbox are detailed. They show that Shennong can be integrated into complex processing pipelines. The source code of those experiments is distributed with Shennong and can be used as introductory examples to new users. Moreover, those experiments draw some interesting insights. The first experiment demonstrated that Mel filterbank performs better than the popular MFCC on a phone discrimination task. It also showed that VTLN speaker normalization reduces the error rates by 5 %. The second experiment analyzed the amount of speech required to train a VTLN model and demonstrated that 5 to 10 minutes of signal per speaker are enough to reach near-optimal performances, whereas the common use is to use several hours of speech. Finally, the last experiment compared pitch estimation algorithms under various noise conditions and demonstrated that the CREPE algorithm provided by Shennong, although trained on music, shows a good generalization capacity to speech. It is also more robust to noise than YAAPT and Praat algorithms, popular alternatives commonly used in phonology.

The development of Shennong is not over. Indeed we plan to add more features extraction algorithms, such as Voice Activity Detection and Contrastive Predictive Coding (Oord et al., 2018). Furthermore, because Shennong is free and open-source software, the user's needs and requests will also impact its future. We hope Shennong's community of users and contributors will grow as its visibility increases.

Open Practices Statement

The authors declare that they have no conflict of interest. The Shennong software and materials for all experiments are available online at https://github.com/bootphon/shennong. None of the experiments was preregistered.

Acknowledgements This work is funded by Inria (Grant ADT-193), the Agence Nationale de la Recherche (ANR-17-EURE-0017 Frontcog, ANR-10-IDEX-0001-02 PSL, ANR-19-P3IA-0001 PRAIRIE 3IA Institute), CIFAR (Learning in Minds and Brains) and Facebook AI Research (Research Grant).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ..., et al (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI16)*, (pp. 265–283).

Benzeghiba, M., De Mori, R., Deroo, O., Dupont, S., Erbes, T., Jouvet, D., ..., et al. (2007). Automatic speech recognition and speech variability: A review. *Speech Communication*, *49*(10-11), 763–786.

Boersma, P. (1993). Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. In *Proceedings of the institute of Phonetic sciences*, (Vol. 17, pp. 97–110).

Boersma, P. (2001). Praat, a system for doing phonetics by computer. *Glot International*, *5*(9/10), 341–345.

Can, D., Martinez, V. R., Papadopoulos, P., & Narayanan, S. S. (2018). Pykaldi: A python wrapper for kaldi. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*: IEEE.

Cieri, C., Miller, D., & Walker, K. (2004). The fisher corpus: A resource for the next generations of speech-to-text. In *LREC*, (Vol. 4, pp. 69–71).

De Vries, N. J., Davel, M. H., Badenhorst, J., Basson, W. D., De Wet, F., Barnard, E., & De Waal, A. (2014). A smartphone-based ASR data collection tool for under-resourced languages. *Speech Communication*, *56*, 119–131.

Dunbar, E., Cao, X. N., Benjumea, J., Karadayi, J., Bernard, M., Besacier, L., ..., Dupoux, E. (2017). The zero resource speech challenge 2017. In *2017 IEEE automatic speech recognition and understanding workshop (ASRU)*, (pp. 323–330): IEEE.

Dunbar, E., Karadayi, J., Bernard, M., Cao, X. N., Algayres, R., Ondel, L., ..., Dupoux, E. (2020). The zero resource speech challenge 2020: Discovering discrete subword and word units. In *Interspeech 2020*.

Ellis, D. P. W. (2005). PLP and RASTA (and MFCC, and inversion) in Matlab. http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat. Online web resource.

Eyben, F., Wöllmer, M., & Schuller, B. (2010). Opensmile: The munich versatile and fast open-source audio feature extractor. In *Proceedings of the 18th ACM international conference on Multimedia*, (pp. 1459–1462).

Fer, R., Matějka, P., Grézl, F., Plchot, O., Veselỳ, K., & Černockỳ, J. H. (2017). Multilingually trained bottleneck features in spoken language recognition. *Computer Speech & Language*, *46*, 252–267.

Ghahremani, P., BabaAli, B., Povey, D., Riedhammer, K., Trmal, J., & Khudanpur, S. (2014). A pitch extraction algorithm tuned for automatic speech recognition. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, (pp. 2494–2498).

Harper, M. (2013). The babel program and low resource speech technology. *Proc. of ASRU 2013*.

Hermansky, H. (1990). Perceptual linear predictive (PLP) analysis of speech. *The Journal of the Acoustical Society of America*, *87*(4), 1738–1752.

Hermansky, H., & Morgan, N. (1994). Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing*, *2*(4), 578–589.

Hermansky, H., Morgan, N., Bayya, A., & Kohn, P. (1991). RASTA-PLP speech analysis. In *Proc. IEEE int'l conf. acoustics, speech and signal processing*, (Vol. 1, pp. 121–124).

Hung, L. H., Kristiyanto, D., Lee, S. B., & Yeung, K. Y. (2016). Guidock: Using docker containers with a common graphics user interface to address the reproducibility of research. *PloS ONE*, *11*(4), e0152686.

Kim, D., Umesh, S., Gales, M., Hain, T., & Woodland, P. (2004). Using VTLN for broadcast news transcription. In *8th international conference on spoken language processing*.

Kim, J. W., Salamon, J., Li, P., & Bello, J. P. (2018). Crepe: A convolutional representation for pitch estimation. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, (pp. 161–165).

Koolagudi, S. G., & Rao, K. S. (2012). Emotion recognition from speech: A review. *International Journal of Speech Technology*, *15*(2), 99–117.

Lenain, R., Weston, J., Shivkumar, A., & Fristed, E. (2020). Surfboard: Audio feature extraction for modern machine learning.

Liu, F., Surendran, D., & Xu, Y. (2006). Classification of statement and question intonations in Mandarin. *Proc. 3rd speech prosody,* (pp. 603–606).

Mauch, M., & Dixon, S. (2014). pYIN: A fundamental frequency estimator using probabilistic threshold distributions. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 659–663): IEEE.

Oord, A. v. d., Li, Y., & Vinyals, O. (2018). Representation learning with contrastive predictive coding. arXiv:1807.03748.

Orozco-Arroyave, J., Hönig, F., Arias-Londoño, J., Vargas-Bonilla, J., Daqrouq, K., Skodda, S., . . . , Nöth, E. (2016). Automatic detection of Parkinson's disease in running speech spoken in three different languages. *The Journal of the Acoustical Society of America*, *139*(1), 481–500.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., . . . , et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems, 32*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . , et al. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Pitt, M. A., Dilley, L., Johnson, K., Kiesling, S., Raymond, W., Hume, E., & Fosler-Lussier, E. (2007). Buckeye corpus of conversational speech (2nd release). Columbus, OH: Department of Psychology Ohio State University.

Plante, F., Meyer, G., & Ainsworth, W. (1995). A pitch extraction reference database. In *Eurospeech-1995*, (pp. 837–840).

Povey, D. (2010). Notes for affine transform-based VTLN. Microsoft Research.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., . . . , Vesely, K. (2011). The Kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*: IEEE Signal Processing Society.

Ramirez, J., Górriz, J. M., & Segura, J. C. (2007). Voice activity detection. Fundamentals and speech recognition system robustness. *Robust Speech Recognition and Understanding*, *6*(9), 1–22.

Riad, R., Titeux, H., Lemoine, L., Montillot, J., Bagnou, J. H., Cao, X. N., . . . , Bachoud-Lévi, A. C. (2020). Vocal markers from sustained phonation in huntington's disease. In *Interspeech 2020*.

Ryant, N., Church, K., Cieri, C., Cristia, A., Du, J., Ganapathy, S., & Liberman, M. (2019). The second dihard diarization challenge: Dataset, task, and baselines. *Interspeech, 2019*, (pp. 978–982).

Ryant, N., Singh, P., Krishnamohan, V., Varma, R., Church, K., Cieri, C., . . . , Liberman, M. (2020). The third dihard diarization challenge. arXiv:2012.01477.

Saeed, A., Grangier, D., & Zeghidour, N. (2021). Contrastive learning of general-purpose audio representations. In *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, (pp. 3875–3879).

Schatz, T., Bernard, M., & Thiollière, R. (2020). h5features: Efficient storage of large features data. https://github.com/bootphon/h5features/releases/tag/v1.3.3. Software. Version 1.3.3.

Schatz, T., Feldman, N. H., Goldwater, S., Cao, X. N., & Dupoux, E. (2021). Early phonetic learning without phonetic categories: Insights from large-scale simulations on realistic input. *Proceedings of the National Academy of Sciences, 118*(7).

Schatz, T., Peddinti, V., Bach, F., Jansen, A., Hermansky, H., & Dupoux, E. (2013). Evaluating speech features with the minimal-pair ABX task: Analysis of the classical MFC/PLP pipeline. In *Interspeech 2013*, (pp. 1–5).

Schatz, T., Peddinti, V., Cao, X. N., Bach, F., Hermansky, H., & Dupoux, E. (2014). Evaluating speech features with the minimal-pair ABX task (II): Resistance to noise. In *15th annual conference of the international speech communication association*.

Silnova, A., Matejka, P., Glembek, O., Plchot, O., Novotnỳ, O., Grezl, F., . . . , Cernockỳ, J. (2018). But/phonexia bottleneck feature extractor. In *Odyssey*, (pp. 283–287).

Tirumala, S. S., Shahamiri, S. R., Garhwal, A. S., & Wang, R. (2017). Speaker identification features extraction methods: A systematic review. *Expert Systems with Applications*, *90*, 250–271.

Versteegh, M., Anguera, X., Jansen, A., & Dupoux, E. (2016). The zero resource speech challenge 2015: Proposed approaches and results. *Procedia Computer Science*, *81*, 67–72.

Versteegh, M., Thiollière, R., Schatz, T., Cao, X. N., Anguera, X., Jansen, A., & Dupoux, E. (2015). The zero resource speech challenge 2015. In *Interspeech 2015*.

Zahorian, S. A., & Hu, H. (2008). A spectral/temporal method for robust fundamental frequency tracking. *The Journal of the Acoustical Society of America*, *123*(6), 4559–4571.

Zeghidour, N., Usunier, N., Synnaeve, G., Collobert, R., & Dupoux, E. (2018). End-to-end speech recognition from the raw waveform. In *Interspeech 2018*.