INSA Lyon - Département Informatique

Report

of the

Graduation Project

Graduation project to design an intelligent speech transcription system for unconventional data [R&D]

Projet de fin d'études de conception d'un système intelligent de transcription de la parole sur des données non conventionnelles [R&D]

Malo OLIVIER

Defended January 31st, 2024

Project conducted from **September 11th, 2023** to **January 12th, 2024**

within the reception structure

Laboratoire de Mécanique des Structures et des Systèmes Couplés - Conservatoire National des Arts et Métiers (Paris - 75003)

Referent	:	Sylvie CALABRETTO, Associate Professor	INSA Lyon
Tutor	:	Éric BAVU, Professor	CNAM (Paris)

Thanks

I would like to express my sincere gratitude to my supervisor, Pr. Éric BAVU, for his invaluable assistance and guidance throughout this project. I warmly thank Mr. Julien HAURET, who sponsored this project, for his trust and support. I would also like to thank Pr. Sylvie CALABRETTO for her support and advice. Lastly, I express my appreciation to Mrs. Sarah POIRÉE and Dr. Christophe LANGRENNE for their technical expertise.

Je remercie mon père, mon frère, ma mère ainsi que ma cousine d'avoir relu ce rapport et de m'avoir aidé à le peaufiner. Je suis heureux de transmettre ce rapport de fin de master aux membres de ma famille, qui j'espère sauront l'apprécier et remarquer le temps et les efforts fournis dans ce stage. Ce Projet de Fin d'Études marque la fin d'une période très soutenue à Paris et je suis content de pouvoir enfin partir à Stockholm en Érasmus. Je remercie également mes amis de Massy-Palaiseau et d'Antony et mes amis de l'INSA Lyon qui ont su me supporter depuis que je les connais.

Il faut être enthousiaste de son métier pour y exceller. - Denis Diderot

Contents

1 Introduction

1.1 Context

Julien HAURET is currently conducting research for his doctoral degree on Deep Learning methods for audio bandwidth extension applied to speech signals captured by noise-resistant bone conduction microphones [2]. He developped a Deep Neural Network, EBEN, that aims to enhance the clarity and authenticity of speech transmitted through body-conduction microphones by utilizing specialized Deep Learning architectures.

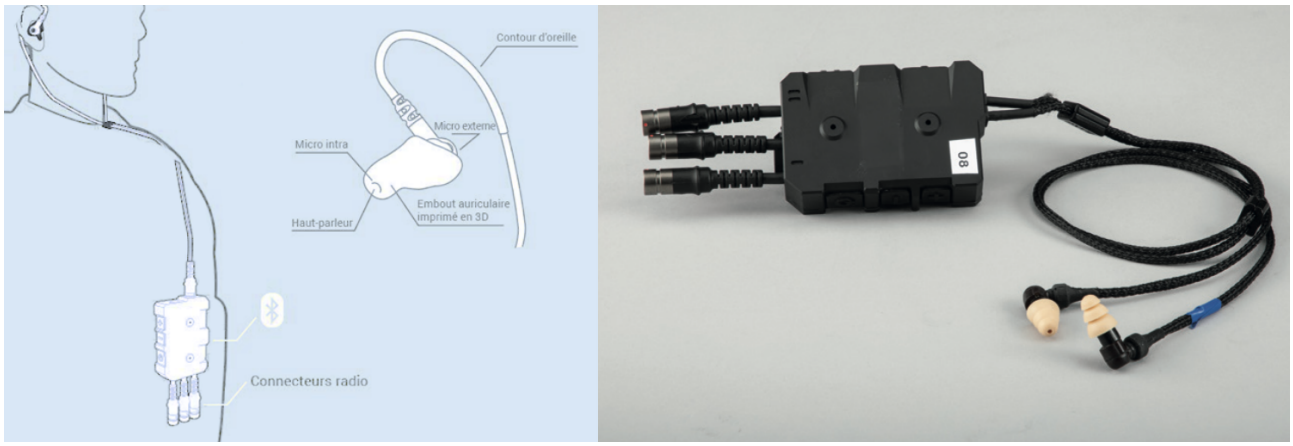


Figure 1: Example of an unconventional speech pickup device : in-ear pickup behind active/-passive hearing protectors [2]

The use of body-conducted microphones for audio data capture has the advantage of eliminating background noise, at the expense of speech bandwidth, thus requiring signal enhancement techniques to recover all frequencies of the vocal signal. The aim of this graduation project's experiments is to analyze the performance differences among various microphones and quantify the contribution of a speech enhancement system in pre-processing. To achieve this, the VibraVox dataset[1], which will consist in 50 hours of French speech recordings, is being created by recording 200 participants in various acoustical conditions imposed by a fifth-order ambisonic spatialization sphere [3]. Annex C shows the ambisonic sphere used during the recordings. The participants voices are captured using six kinds of microphones, five of which are body-conducted. The recordings from the non-conventional microphones will be used to finetune the EBEN network [2]. See Annex B for more details on the microphones used. The microphones are :

- A mouth headworn reference microphone (Shure WH20XLR) ;

[1]Éric BAVU, CNAM, LMSSC, VibraVox dataset [online]. Already available on the HuggingFace platform : `https://huggingface.co/datasets/Cnam-LMSSC/vibravox` (Consulted on january the 2^{nd} 2024)

- An in-ear comply foam microphone (Knowles SPH1642HT5H-1 top-port) ;
- An in-ear rigid earpiece microphone (STMicroelectronics MP34DT01) ;
- A temple contact condenser (AKG - C411) ;
- A forehead miniature accelerometer (Knowles BU23173-000) ;
- A throat piezoelectric sensor (Ixradio XVTM822D-D35) ;

The VibraVox dataset aims at being freely used for various tasks, including Automatic Speech Recognition (ASR) in Speech-to-Text (S2T) or Speech-to-Phoneme (S2P), bandwidth extension, speaker identification, and more.

1.2 Objectives

The creation of the VibraVox dataset needs to go through the following steps :

- Designing an experimental protocol that uses the six microphones and the ambisonic spatialization sphere ;
- Collecting the data ;
- Post-processing the data ;
- Making the dataset available on the HuggingFace platform ;

In order to measure the contribution of a speech enhancement system in the preprocessing of unconventional data, it is necessary to develop a training program[2] for pre-existing architectures on an Automatic Speech Recognition (ASR) task. The impact of a bandwidth extension technique could be quantified by comparing the performance of a speech recognition model trained on degraded recordings with the performance of a model trained on recordings where the bandwidth has been restored. This will give insight of the effectiveness of the bandwidth extension technique. The main task of the training program is Automatic Speech Recognition (ASR), with two variants : Speech-to-Text (S2T) or Speech-to-Phoneme (S2P). The aim is to implement a training program that can be used easily on any audio dataset. It is implemented in Python using the PyTorch, PyTorch Lightning, Hydra, and Tensorboard libraries. The program is compatible with wav2vec 2.0 [1] architectures with a Connectionist Temporal Classification (CTC) [4] loss function and Transformer [5] architectures in general. A strategy of freezing Transformer layers, a trapezoidal scheduler, and the hyperparameters described in [1] are adopted.

[2]Soon available on GitHub : `https://github.com/jhauret/vibravox`

Throughout the finetuning process, we used the supercomputer Jean ZAY from the IDRIS computing center. Thanks to its multiple powerful GPUs, we were able to train our models in a reasonable amount of time.

A scientific article will be written to present the potential applications of the VibraVox dataset.

1.3 Contributions

Julien HAURET, Pr. Éric BAVU, and I collaborated on the implementation of the training program using the AGILE methodology. The needs and functionalities of the training program were defined by Julien HAURET and Pr. Éric BAVU, taking into consideration the requirements of the scientific article writing and the continuity with Julien's thesis work. Throughout the project's progression, we were able to discuss technical choices and implementation strategies. The development goals have regularly changed to adapt to the priorities of execution. Thanks to this methodology and regular communication, we were able to make efficient progress on the project : objectives were achieved ahead of schedule, and we were able to accomplish additional tasks. The contribution of each one of us to the development of the training program is as follows :

- I have fully developed the training program ;
- Julien HAURET, as the project owner, provided the main guidelines for the training program and the functionalities to be implemented ;
- Pr. Éric BAVU, as the project manager, was able to guide me on the technical choices and implementation strategies ;

Our research team's contribution to the development of the VibraVox dataset is as follows :

- I was able to record around 40 participants during my intership ;
- Sarah POIRÉE and Christophe LANGRENNE also each involved around thirty participants in the data recording ;
- Pr. Éric BAVU oversaw the proper development of the VibraVox dataset, took charge of the involvement of several participants, contributed to the design of the experimental protocol with Julien HAURET, and was present during technical incidents ;

In the following, a state-of-the-art study provides an overview of the current knowledge on Deep Neural Networks and the various concepts used to optimize their performance for our specific use case. It identifies the most relevant works, the most promising methods and approaches, as well as the limitations and challenges that still need to be addressed. Subsequently, scientific and technical proposals formulated within the scope of the research are presented. In

addition, a personal assessment is made to evaluate the contributions of this end-of-studies project to my education and professional project. Finally, we conclude on the impact and prospects that my end-of-studies project will bring to research and industry.

2 Bibliographic study

While certain tasks or treatments remain more easily and skillfully executed by humans, they consume a lot of time and energy that could be of better use for human tasks. Recognizing this inherent trade-off, Deep Neural Networks are replacing these repetitive and complex tasks.

In the context of my final year project, Julien HAURET needed to train speech recognition models on data collected from unconventional microphones. Hence, I had to design a training program that would allow him to train speech recognition models on any dataset, but specifically the VibraVox dataset.

The state of the art in neural networks is rapidly evolving. Every month, new methods, architectures, and pre-trained models become available. In this study, we focus on the most relevant works, the most promising methods and approaches, particularly for our use case, as well as the limitations and challenges that still need to be addressed.

2.1 Deep Neural Networks

Numerous research papers have been published on Deep Neural Networks. These papers have contributed to the development of Deep Neural Network architectures and the optimization of their performance.

Firstly, a rapid overview of the functioning of Deep Neural Networks follows.

2.1.1 Architecture

The perceptron is the simplest neural network architecture. Its architecture consists of just a single neuron with one or more input nodes and one output node. The perceptron is defined by the following equation :

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i \times x_i + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where x_i is the input of the neuron i, w_i is the weight of the connection between the input i and the neuron and b is the bias, or threshold, of the neuron. See Figure 2.

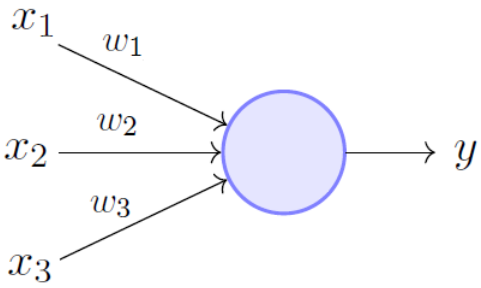


Figure 2: The perceptron architecture[3].

The sigmoid neuron is a perceptron with a sigmoid activation function. Hence the name sigmoid neuron. A rapid overview of the curves of activation functions can be found in Chapter 2.1.2. The sigmoid neuron is defined by the following equation :

$$y = \sigma\left(\sum_{i=1}^{n} w_i \times x_i + b\right) \tag{2}$$

It is indeed a perceptron that does not output a binary value but a value in the range $[0, 1]$.

The multilayer feedforward neural network is a neural network architecture that consists of at least three layers of nodes : an input layer, a hidden layer and an output layer [6] as the Figure 3 shows.

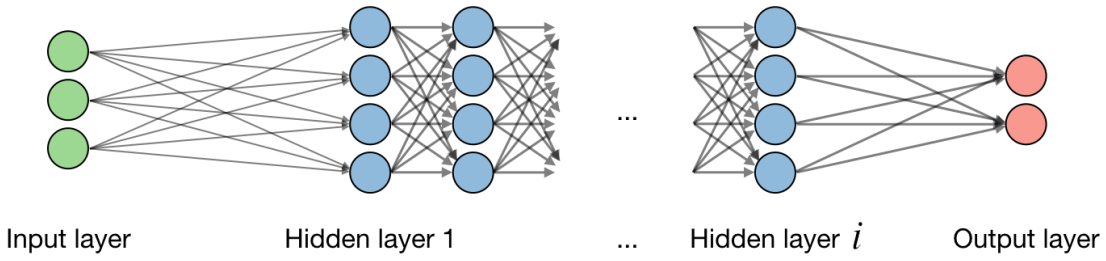


Figure 3: A typical feedforward neural network architecture[4].

By noting i the i^{th} hidden layer of the network and j the j^{th} neuron of the hidden layer i, the output of the neuron j of the hidden layer i is given by the following equation :

$$y_{i,j} = \sigma\left(\sum_{k=1}^{n} w_{i,j,k} \times x_{i-1,k} + b_{i,j}\right) \tag{3}$$

[3]Gerry SAPORITO, What is a perceptron [online]. Available on : `https://towardsdatascience.com/what-is-a-perceptron-210a50190c3b` (Consulted on january the 2^{nd} 2024)

[4]Afshine AMIDI, Shervine AMIDI, Deep Learning cheatsheet [online]. Available on : `https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning` (Consulted on december the 20^{th} 2023)

where $x_{i-1,k}$ is the output of the neuron k of the layer $i-1$, $w_{i,j,k}$ is the weight of the connection between the neuron k of the layer $i-1$ and the neuron j of the layer i, $b_{i,j}$ is the bias of the neuron j of the layer i and σ is the activation function of the neuron j of the layer i.[5]

2.1.2 Activation functions

For this basic neural architecture, each neuron of a layer is connected to all the neurons of the previous layer.[6] The activation function is a non-linear function that allows the network to learn. The most used activation functions are the *sigmoid* function, the *tanh* function and the *ReLU* function [7]. An overview of activation functions can be found in Annex A. The *sigmoid* function is defined by the following equation :

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4}$$

The *tanh* function :

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5}$$

The *ReLU* function :

$$ReLU(x) = \max(0, x) \tag{6}$$

2.1.3 Loss functions

A neural network is trained by minimizing a loss function. The loss function is a function that computes the difference between the predicted value and the true value. The most used loss functions are the *MeanSquaredError* (*MSE*) function, the *MeanAbsoluteError* (*MAE*) function and the *CrossEntropy* function. The *MSE* function is defined by the following equation :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{7}$$

The *MAE* function :

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{8}$$

The *CrossEntropy* function :

$$CrossEntropy = -\sum_{i=1}^{n} y_i \times \log(\hat{y}_i) \tag{9}$$

where y_i is the true value and $\hat{y}_i$ is the predicted value.

[5] Michael NIELSEN (2015), Neural Networks and Deep Learning [online]. Available on : `http://neuralnetworksanddeeplearning.com/chap2.html` (Consulted on december the 20^{th} 2023)

[6] Michael NIELSEN (2015), Neural Networks and Deep Learning [online]. Available on : `http://neuralnetworksanddeeplearning.com/chap2.html` (Consulted on december the 20^{th} 2023)

2.1.4 Learning rate

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated [8].

The learning rate plays a crucial role in training a Machine Learning model. Imagine it as the potentiometer that controls how much the model adjusts its predictions in response to its mistakes. The learning rate is a value in the range $[0, 1]$, with smaller values indicating cautious adjustments and larger values indicating bolder leaps. To find an optimal solution, the loss function must be minimized. Thus, it is the same as finding the minimum of a function. The learning rate is the step size of the gradient descent algorithm.

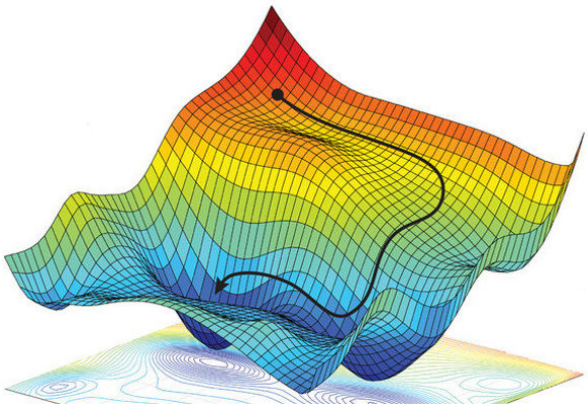


Figure 4: Gradient descent is an optimization algorithm that is used to find the lowest point of a function [9].

A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution. A learning rate that is too small can cause the process to get stuck.[7] The learning rate is a hyperparameter that you can tune either to be fixed or adapted during training.

One of the most used learning rate adaptation strategies is the *Adam* strategy, which is an adaptive learning rate optimization algorithm, designed specifically for training Deep Neural Networks.

2.1.5 Back propagation

Neural networks learn like a student learns with a teacher.

- The student takes a practice test (forward pass).

[7] Jason BROWNLEE, Machine Learning Mastery, Understand the Impact of Learning Rate on Neural Network Performance, september the 12^{th} 2020 [online]. Available on : `https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/` (Consulted on december the 20^{th} 2023)

- The teacher grades the test (calculating error).
- The teacher gives feedback (backpropagation).
- The student adjusts their knowledge (updating weights with gradient descent).
- The process is repeated until the student passes the test (convergence).

The backpropagation is the process of calculating the gradient of the loss function with respect to the weights of the network.[8] It is an algorithm that allows the network to learn by updating the weights of the network. Using chain rule, the backpropagation is defined by the following equations :

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial y_{i,j}} \times \frac{\partial y_{i,j}}{\partial w_{i,j}} \tag{10}$$

As a result, the weight is updated :

$$w_{i,j} = w_{i,j} - \alpha \times \frac{\partial L}{\partial w_{i,j}} \tag{11}$$

where L is the loss function, $w_{i,j}$ is the weight of the connection between the neuron k of the layer $i-1$ and the neuron j of the layer i, $y_{i,j}$ is the output of the neuron j of the layer i, α is the learning rate and $\frac{\partial L}{\partial y_{i,j}}$ is the gradient of the loss function with respect to the output of the neuron j of the layer i. The introduction of the variables is given in Equation (3).[9]

Different techniques and strategies have been developed to improve the performance of Deep Neural Networks. Such techniques are called regularization techniques. The most used regularization techniques are the $L1$ regularization, the $L2$ regularization, the dropout and the batch normalization.

To finetune a pretrained model for ASR task, dropout and batch normalization are used as well as the *Adam* strategy, a trapezoidal scheduler and a Transformer layers freeze strategy from [1].

[8] Venelin VALKOV, MLExpert, What is backpropagation, and how is it used in Machine Learning? [online]. Available on : `https://www.mlexpert.io/machine-learning/interview-questions/backpropagation` (Consulted on december the 20^{th} 2023)

[9] Michael NIELSEN (2015), Neural Networks and Deep Learning [online]. Available on : `http://neuralnetworksanddeeplearning.com/chap2.html` (Consulted on december the 20^{th} 2023)

2.2 Automatic Speech Recognition and state-of-the-art architectures

In this context, we focus on the most commonly used Deep Neural Network architectures in the field of speech recognition. These include :

- Recurrent Neural Networks (RNNs) ;
- Convolutional Neural Networks (CNNs) ;
- Transformer Neural Networks ;

We will first provide an overview of RNNs, CNNs, and Transformers, but our primary focus will be on the wav2vec 2.0 architecture that uses CNNs and Transformers.

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks have the ability to process sequential data. RNNs are comprised of recurrent cells, which enable the retention of memory for previously processed data. This unique characteristic allows for the utilization of previous outputs as inputs, while also maintaining hidden states.[10]

RNNs operate by sequentially processing data, one element at a time, through feedback loops. At each step, the network utilizes both the current data and the previous data it remembers to generate an output. For the task of speech recognition, Graves et al. [10] proposed the use of RNNs with a Connectionist Temporal Classification (CTC) [4] loss function for the speech recognition task. Unfortunately, RNNs suffer from major problems including the long-term memory problem. RNNs are unable to retain information for long periods of time.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks extract and leverage patterns within audio data. CNNs apply filters at different scales. When applied on 2D, they allow to extract features like edges, textures, and overall patterns within the spectrogram of the raw audio. More and more CNNs are also applied on 1D data such as raw audio data, thus acting as learnable audio filters. The Figure 5 shows the architecture of wav2vec 2.0 and its Convolutional Neural Networks.

[10] Afshine AMIDI, Shervine AMIDI, Deep Learning cheatsheet [online]. Available on : `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks` (Consulted on december the 20^{th} 2023)

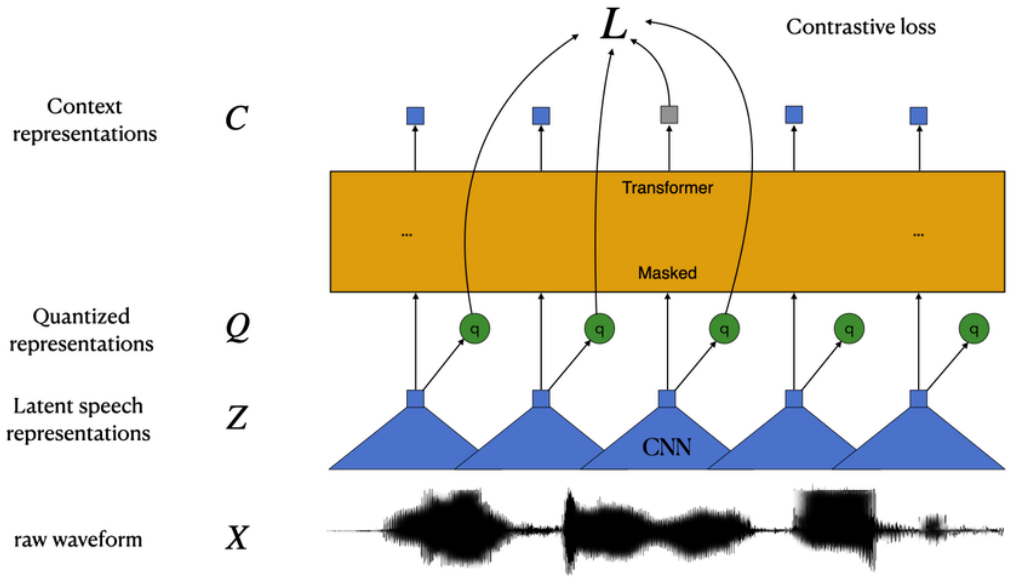


Figure 5: Framework of wav2vec 2.0 [1]

In the wav2vec architecture, CNNs extract short-term temporal features from the audio signal, effectively identifying local patterns within the waveform.

2.2.3 Transformer Neural Networks

The current state-of-the-art in the Automatic Speech Recognition (ASR) task relies on Transformer Neural Networks which were introduced in 2017 by Vaswani et al. in their article "Attention is all you need" [5]. The architecture of a Transformer network is depicted in the Figure 6.

They rely on self-attention, a technique that enables Machine Learning models to focus on specific parts of the data they are processing. Transformers have quickly surpassed Recurrent Neural Networks as the benchmark architecture for Natural Language Processing (NLP) tasks achieving breakthrough results in a variety of tasks, including automatic translation, natural language understanding, and text generation.

Next, I present wav2vec2 that emerged as a breakthrough architecture, integrating both CNNs and Transformers strengths to revolutionize speech recognition.

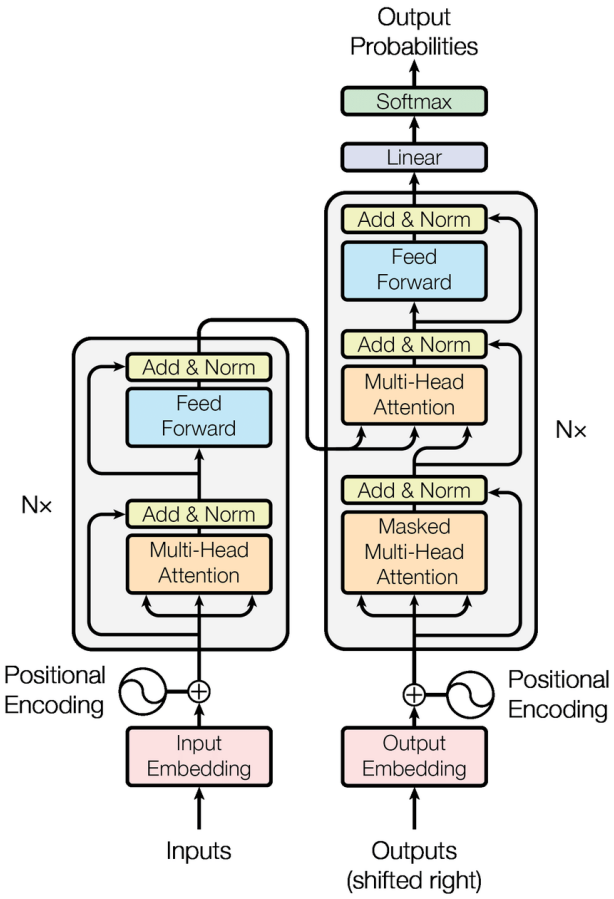


Figure 6: The Transformer architecture [5].

2.2.4 wav2vec 2.0

wav2vec 2.0 [1] is a well-structured architecture for processing audio data and identifying spoken words, composed of the following components :

- Feature extraction ;
- Encoder ;
- Quantization module ;
- Output layer ;

Refer to Figure 5 for a visual representation of the wav2vec 2.0 architecture.

Feature extraction : This initial stage acts as a bridge between the raw audio waveform and the higher-level representation used by the subsequent layers. The feature extraction module is

composed of a stack of convolutional layers (CNNs) that extract short-term temporal features from the audio signal, effectively identifying local patterns within the waveform. Residual connections skip multiple convolutional layers, directly feeding information to later stages. This helps preserve long-term information and prevents the model from drowning in local details.

Encoder : The core of wav2vec 2.0, the encoder leverages the powerful Transformer architecture to analyze the extracted features and capture long-range dependencies within the speech signal. The self-attention layers nested within the Transformer architecture play a crucial role in the model's ability to recognize spoken words by understanding the intricate relationships within the extracted audio features. In contrast to CNNs, self-attention layers analyze relationships between distant features, transcending the limitations of convolutional layers that focus on local patterns. They allow the model to consider the surrounding context while processing each feature, leading to a more nuanced understanding of the audio. By focusing on relevant information and downplaying irrelevant noise, self-attention layers improve the model's ability to recognize speech even in noisy environments.

Since audio signals lack inherent order like text, positional encoding injects information about the temporal dimension, enabling the model to understand the sequence of features.

Quantization module : To facilitate self-supervised learning, wav2vec2 employs a quantization module that discretizes the continuous outputs of the Transformer encoder into a set of discrete latent representations. This allows the model to learn by predicting the correct quantized representation for each time step of the audio.

Output layer : Based on the processed features and learned relationships, the output layer performs the final task of :

- Predicting the most likely sequence of words spoken in the audio input ;
- Generating synthetic speech based on the extracted features and learned patterns ;

Connectionist Temporal Classification (CTC) loss function [11] : wav2vec2 uses a Connectionist Temporal Classification (CTC) [4] decoder for speech recognition, which directly predicts the word sequence from the latent representations without explicitly aligning them to individual phonemes.

wav2vec2 analyzes the audio features and generates a sequence of probability distributions for each time step. Each distribution represents the likelihood of different phonemes being spoken at that specific moment.

The CTC creates a lattice, a directed acyclic graph, where each node represents a possible state in the word sequence. Edges between nodes represent transitions between phonemes. The probabilities from the neural network are used to assign weights to these edges.

The CTC employs a beam search algorithm to navigate this lattice. Starting from the initial node ("silence"), the algorithm explores paths with the highest accumulated probability, effectively prioritizing the most likely word sequences.

The best path through the lattice, determined by the beam search, is then converted into a sequence of words, representing the CTC's final prediction of the spoken text.

Overall, wav2vec2 achieves near-human performance in clean speech recognition tasks, surpassing previous state-of-the-art models by a significant margin, even for its lightest variant. It also outperforms previous models in noisy environments and diverse pronunciations, demonstrating its robustness to background noise, it thus makes it practical for real-world applications. It efficiently handles large or low amounts of data, achieving state-of-the-art results with as little as 10 minutes of labeled data. It accurately processes long audio inputs without requiring explicit segmentation or alignment, making it suitable for large datasets lacking such pre-processing steps. Also it does not need extensive manually annotated data. It adapts to various tasks beyond speech recognition, including speaker identification and speech summarization.

wav2vec2 is a good candidate for the ASR task on unconventional microphones, making it efficient and robust to background noise while being lightweight. The training program is implemented and thought to be compatible and most efficient with wav2vec2 while still being flexible enough to finetune other architectures.

3 Proposal

Two objectives were assigned to me for this graduation project.

1. Creating the VibraVox dataset ;
2. Implementing the training program ;

Additional missions were added to the project such as :

- Controlling the quality of the data ;
- Post-processing the data ;
- Uploading the dataset to the HuggingFace platform ;
- Making the training program user-friendly and well-documented for a GitHub release ;
- Finetuning the wav2vec 2.0 architecture on the VibraVox dataset ;
- Participate to the writing of a scientific article on the VibraVox dataset and on the ASR task achieved on this dataset ;

In this section is presented the directions, the choices taken and strategies that were adopted to achieve these two main objectives.

3.1 VibraVox dataset

3.1.1 Data collection

Building the VibraVox dataset is a high-priority task for Julien HAURET's work. Indeed, benchmarking the performance of the EBEN network on the VibraVox dataset is essential to evaluate the effectiveness of the bandwidth extension technique, and for this purpose he needed to collect enough degraded recordings.

Unlike traditional microphones, which rely on airborne sound waves, body-conduction microphones capture speech signals directly from the body, offering advantages in noisy environments by eliminating the capture of ambient noise.[11] However, speech recorded with these unconventional microphones is degraded by the acoustic path between the mouth and the transducers : with in-ear microphones, low frequencies are amplified and almost no acoustic signal is recorded above 2 kHz, motivating the use of Deep Learning signal enhancement methods to extrapolate the missing high-frequency content [2].

Thus, the VibraVox dataset is composed of 50 hours of French speech recordings from six microphones, five of which are body-conducted. There are 4 different types of recordings :

[11] CNAM, LMSSC, HuggingFace, VibraVox dataset [online]. Available on : `https://huggingface.co/datasets/Cnam-LMSSC/vibravox` (Consulted on january the 3^{rd} 2024)

- 15 minutes per participant of recordings in which they read a French text in a quiet environment (speech without noise) ;

- 2 min 24 seconds per participant of recordings in which they are silent in a noisy environment (noise without speech) ;

- 54 seconds per participant of recordings in which they are silent in a quiet environment (no speech & no noise) ;

- 54 seconds per participant of recordings in which they read a French text in a noisy environment (speech with noise) ;

Noise without speech audios contain samples that have been selected from relevant classes of the Google Audioset dataset, normalized in loudness, pseudo-spatialized and played from random directions using the spatialization sphere equipped with 56 loudspeakers. The direction of sound is sampled uniformly on the unit sphere using the inverse cumulative distribution function. The objective of this measurement phase is to gather realistic background noises that will be combined with the speech without noise recordings to maintain a clean reference. No speech & no noise parts record solely physiological and microphone noises. These samples can be valuable for tasks such as heart rate tracking or simply analyzing the noise properties of the various microphones.

The speech with noise phase serves to test the different systems (Speech Enhancement, Automatic Speech Recognition, Speaker Identification) that will be developed based on the recordings from the other three phases. This real-world testing will provide valuable insights into the performance and effectiveness of these systems in practical scenarios. Opera pieces, applause and demonstrations, were recorded using the ZYLIA ZR-1 Portable Recorder from spatialized scenes and replayed in the spatialization sphere with ambisonic processing. For a comprehensive overview of the ZYLIA ZR-1 Portable Recorder, refer to Annex D.

Each participant reads French sentences from Wikipedia. The sentences are chosen to be phonetically rich and to cover all the phonemes of the French language. The participants are recorded in a fifth-order ambisonic spatialization sphere [3] to simulate different acoustical environments. The six microphones record simultaneously with the Zoom F8n multitrack field recorder : the six-sensor recorded audios are aligned and synchronous to insure the same temporal position for each microphone. Annex E presents the Zoom F8n multitrack field recorder. The audios are recorded with a sampling rate of 48 kHz and encoded with a bit depth of 32 bits. Here follows the parameters table for each microphone :

Microphone	Sampling rate	Bit depth	Channels	Trim (dB)	High-pass filter cutoff frequency (Hz)
Mouth air-borne	48 kHz	32 bits	1	52	20
Rigid In-ear	48 kHz	32 bits	1	20	20
Soft In-ear	48 kHz	32 bits	1	30	20
Temple	48 kHz	32 bits	1	65	20
Forehead	48 kHz	32 bits	1	56	20
Throat	48 kHz	32 bits	1	24	20

Table 1: Parameters table for each microphone[12].

3.1.2 Post-processing

The whisper-timestamped[13] Deep Learning model has been used for the post-processing of the dataset. Whisper-timestamped is an extension of the OpenAI-whisper [12]. We used its Voice Activity Detection (VAD) feature to accurately identify sequences with vocals. Using a Dynamic Time Warping approach applied to cross-attention weights, and an additionnal VAD method (silero[14]), beginning and end timestamps estimation is more accurate than other VADs. The silero VAD plays a crucial role in segmenting the audio into speech and non-speech parts, acting as a filter and effectively reducing the occurrence of false positives. Moreover, confidence scores are provided for each timestamp, further enhancing the accuracy of the results. Whisper-timestamped is specifically fine-tuned for VAD purposes, ensuring that it avoids generating false detections while delivering more reliable word timestamps.

To evaluate its quality and coherence, whisper-timestamped VAD was applied to the entire VibraVox dataset. This enabled the calculation of the Word Error Rate (WER) for each individual audio and its corresponding timestamps. To maximize the likelihood of accurately capturing vocal segments and minimize the risk of unintentionally excluding valid speech portions, the VAD-generated timestamps were extended by 0.3 seconds on both ends.

In addition to the whisper-timestamped VAD, we used an additional energy-based VAD, computed using the standard deviation of energy across 4096-sample frames for both vocal and non-vocal segments of each recording. This quantity corresponds to an equivalent level

[12]CNAM, LMSSC, VibraVox dataset [online]. Available on : `https://vibravox.cnam.fr` (Consulted on january the 2^{nd} 2024)

[13]LinTO.ai, whisper-timestamped [online]. Available on : `https://github.com/linto-ai/whisper-timestamped` (Consulted on january the 3^{rd} 2024)

[14]silero.ai, silero-models [online]. Available on : `https://github.com/snakers4/silero-models` (Consulted on january the 3^{rd} 2024)

L_{eq} of the audio recordings in each kinds of segments. This offers the possibility to extract a quantitative measure of the energy distribution within the audio signal and facilitated the identification of potentially unreliable recordings with low vocal energy levels. To establish a reference point for evaluating the energy levels of individual recordings, the noise level for each sensor was determined by concatenating the silence portions of all recordings per sensor and calculating the median energy level. This established a sensor-specific reference point L_0 for evaluating the energy levels of individual recordings and enabled the implementation of sensor-specific removal criteria.

Based on these energy-based VAD results, sensor-specific removal criteria were employed to ensure the quality and consistency of the dataset. The temple sensor being very noisy and sensitive to positioning, this particular sensor has a special processing which will be detailed in the following. For all sensors except the temple contact microphone, recordings with an equivalent sound level $L_{eq}(speech)$ for speech sequences lower than -3 dB below the sensor equivalent noise level L_0 are discarded. This condition aims at automatically discarding recordings with exceptionally low vocal energy levels, potentially indicating sensor malfunction or poor recording conditions. Secondly, we also used a second criterion for all sensors : recordings where $L_{eq}(speech) - L_{eq}(silence)$ is less than 3 dB are also discarded when $L_{eq}(speech)$ are lower than $L_0 + 2$ dB. This ensures to discard the noisy recordings. The temple sensor being very noisy ($L_{eq}(speech) - L_{eq}(silence)$ is very close to 0 dB), a simpler strategy has been used, by only discarding the recordings where $L_{eq}(speech)$ is lower than $L_0 + 2$ dB.

Lastly, hand-picked reference recordings with a WER exceeding 45% identified empty or incoherent 6-sensor audios. This ensured that empty recordings or audios with misaligned transcripts were removed from the dataset. These post-processing steps played a critical role in guaranteeing the reliability and homogeneity of the VibraVox dataset. By incorporating whisper-timestamped Voice Activity Detection (VAD), energy-based VAD, and sensor-specific removal criteria, researchers can be confident that the dataset contains high-quality audio recordings suitable for further body-conducted speech analysis. [13]

Additional, and up-to-date, data statistics are available on the HuggingFace platform[15].

3.1.3 Data upload

Following the standard HuggingFace community format, the VibraVox dataset is organized into separate sections for different data types and splits :

1. ASR data : this section contains audio recordings and transcripts for Automatic Speech Recognition (ASR) tasks. Transcripts are provided in both phonemes and normalized text formats for each microphone. Data is then divided into train, validation, and test splits.

[15]CNAM, LMSSC, HuggingFace, VibraVox dataset [online]. Available on : `https://huggingface.co/datasets/Cnam-LMSSC/vibravox` (Consulted on january the 3^{rd} 2024)

2. BWE data : this section contains audio recordings and transcripts for Bandwidth Extension (BWE) tasks. Transcripts are only provided in normalized text format and focused on stereo audio configurations. Data is also divided into train, validation, and test splits.

Each BWE recording is a stereo audio file, with the first channel corresponding to the body-conduction sensor, the second channel to the reference aligned audio, captured with the reference headworn microphone[16] :

- In-ear comply foam microphone ;
- In-ear rigid earpiece microphone ;
- Forehead miniature accelerometer ;
- Temple contact microphone ;
- Larynx piezoelectric sensor ;
- All sensors ;

3.2 The training program

The training program[17] is a Python program that allows the user to train a speech recognition model on any dataset on Speech-to-Text (S2T) or Speech-to-Phoneme (S2P) tasks. It is designed to be user-friendly, well-documented, flexible, efficient, compatible with any dataset available on the HuggingFace platform, and compatible with at least any wav2vec2 variant model available on the HuggingFace platform. The wav2vec2 model was chosen for its numerous advantages and its state-of-the-art performance in the Automatic Speech Recognition (ASR) task, see Chapter 2.2.4 for more information.

The training program, implemented with PyTorch, PyTorch Lightning, Hydra and Tensorboard libraries, is composed of the following components :

- YAML Hydra configuration files for setting hyperparameters, a specific model, a specific dataset, trainer parameters, schedulers, callbacks and optimizers ;
- A LightningDataModule class for loading the dataset ;
- A LightningModule class for defining the model, the loss function and the metrics ;
- A custom ModelCheckpoint callback class for saving the model ;
- A custom trapezoidal scheduler class for scheduling the learning rate ;

[16] CNAM, LMSSC, HuggingFace, VibraVox dataset [online]. Available on : `https://huggingface.co/datasets/Cnam-LMSSC/vibravox` (Consulted on january the 3^{rd} 2024)

[17] Soon available on GitHub : `https://github.com/jhauret/vibravox`

- A training script for training the model, saving the model and logging the training, validation and testing results ;
- A testing script for testing the model, loading the model and logging the testing results ;
- A README file ;

Given the program's architectural components, we'll begin by dissecting its functionality for S2T and S2P tasks in the next section. Subsequently, we'll expose the technology used to implement it.

3.2.1 Speech-to-Text (S2T) and Speech-to-Phoneme (S2P) tasks

Speech recognition involves transcribing spoken language into text, be it words or phonemes. While the Vibravox dataset focuses on the French language, the program's flexibility allows for training on text and phonemes across diverse languages. Both tasks serve different purposes : Speech-to-Text (S2T) converts spoken words into written text with the Latin alphabet whereas Speech-to-Phoneme (S2P) converts spoken words into phonemes i.e. with the French IPA alphabet. Besides this difference, training a model for one or the other task does not involve any significant changes to the program's architecture. It all comes down to the tokenizer's vocabulary : the vocabulary may contain Latin characters for S2T task or IPA characters for S2P task.

Deep Learning powers both S2T and S2P, but their applications are not the same. S2T is used for dictation, automatic captioning and voice assistants while S2P is used for speech synthesis, understanding linguistic details of the language structures and improving S2T accuracy by segmenting sounds.

If the model hears someone say *Bonjour* :

- S2T would transcribe it as *Bonjour* ;
- S2P would analyze the sounds and output it as *bɔ̃ʒuʁ* ;

The choice between S2T and S2P depends on the desired outcome. S2T focuses on meaning and natural language understanding which drive user-oriented applications while S2P in-depth phonemic analysis offers invaluable insights for research and development. Ultimately, these are complex tasks only possible thanks to the power of Deep Learning.

3.2.2 Hydra configuration files

Hydra is a framework for elegantly configuring complex applications. It provides a hierarchical configuration scheme that allows parameters to be redefined on a per-directory basis. Hydra is designed to simplify the development of versatile applications, particularly in the field of Machine Learning, by providing the following features :

- A powerful composition system that allows the user to override parameters from the command line, environment variables, or by using a rich API ;
- A flexible plugin system that allows the user to extend the functionality of the application by adding new components ;
- A simple API that allows the user to access the configuration from anywhere in the application ;
- A powerful templating system that allows the user to define reusable configuration templates ;

Hydra is used to define hyperparameters and parameters the most user-friendly way possible in the YAML configuration files. The training program was designed to be component-based, easily configurable and flexible. Every aspect of the finetuning process is configurable as :

- The model, its parameters, its architecture, its pretrained weights, whether it be the lightest wav2vec2 model or a pretrained French specialized wav2vec2 model, etc. ;
- The dataset, its configuration, its splits, the number of hours to be trained, the sampling rate, the processor to be used, etc. ;
- The trainer, the number of epochs, the batch size, the number of workers, the number of gpus, etc. ;
- The scheduler, whether it be trapezoidal, cyclic, linear, etc. ;
- The callbacks, whether it be the ModelCheckpoint callback, the EarlyStopping callback, the LearningRateMonitor callback, etc. ;
- The optimizer, whether it be Adam, AdamW, etc. ;

In this way, the user can easily configure the training program to finetune any model on any dataset. Moreover, the program is modular, easily extensible and human-readable thanks to the PyTorchLightning library.

3.2.3 PyTorch and PyTorch Lightning

One of the most fundamental aspects of the program is to be modular and easily extensible so new features can be added. The PyTorch library gives all tensor tools needed to build a neural network from scratch, to define loss functions and metrics whereas the PyTorch Lightning library encapsulates and eases the development of Machine Learning training code.

PyTorch Lightning is a lightweight framework built on top of PyTorch for high-performance AI research and AI engineering. It allows the user to decouple the code from the research, specifically designed for researchers and engineers who want to train and deploy Machine Learning

models efficiently. It abstracts away boilerplate code and repetitive tasks, allowing you to focus on the core logic of your model and achieve faster results.

PyTorch Lightning is composed of the main classes and methods used for common Deep Learning or Machine Learning tasks. Being a framework, data loading, training loop setup, logging, and checkpointing are automated, saving significant time and effort compared to writing everything from scratch as in earlier versions of PyTorch. It offers a concise API and a clear documentation on their website.[18] PyTorch Lightning also leverages techniques like early stopping, gradient clipping, and mixed precision training to optimize the training runs, leading to faster model convergence and improved performance.

For a fast and efficient training, we exploited the multi-GPU capabilities of PyTorch Lightning. The library simplifies the finetuning process on multiple GPUs by effortlessly scaling across multiple GPUs without any code changes. A myriad of possibilities are available as they are usable and adaptable at will :

- Data Parallel (DP) ;
- Distributed Data Parallel (DDP) ;
- Model Parallel (MP) ;
- Pipeline Parallel (PP) ;
- Tensor Parallel (TP) ;
- A mix of these parallelism strategies for custom and efficient runs like DP + PP or DP + PP + TP ;
- Other parallelism strategies ;

According to the HuggingFace blog on Parallelism[19] :

- Data Parallel (DP) replicates the model accross multiple GPUs and splits the batch of data between them. The training is done in parallel on each GPU and the gradients are finally aggregated ;
- Distributed Data Parallel (DDP) is to be prefered over Data Parallel (DP) as it is ideal for large models that exceed the memory capacity of a single GPU : it splits both the model and data across multiple GPUs. It is also required when working with multiple machines or multiple nodes ;

[18] The documentation of PyTorch Lightning is available on : `https://lightning.ai/docs/pytorch/stable/`
[19] HuggingFace, Model Parallelism [online]. Available on : `https://huggingface.co/docs/transformers/v4.15.0/parallelism` (Consulted on january the 3^{rd} 2024)

- Model Parallel (MP) or vertical parallel splits the different layers of the model across multiple GPUs. The data is however not split and goes through the model sequentially hence GPUs not being used are idle;

- Pipeline Parallel (PP) is an advanced version of Model Parallel (MP) : it addresses the issue of GPU underutilization by micro-batching the incoming batch of data and artificially creating a pipeline. This creates a collaborative workflow where multiple GPUs participate concurrently in the computation process ;

- Tensor Parallel (TP) splits the tensors in each GPU process ;

All these considerations are very interesting as they are a continuity for MLOps[20] and the deployment of models in production. Choosing the right combination of parallelism strategies needs to be taken into account for an efficient inference or training processes in production.

The following Figure 7 shows the combination of Data Parallel (DP) and Pipeline Parallel (PP) strategies.

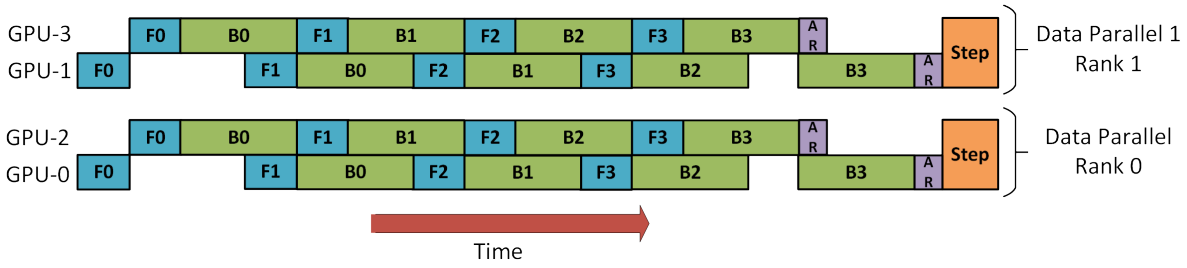


Figure 7: Combination of Data Parallel (DP) and Pipeline Parallel (PP) strategies.[21]

A mix of these parallelism strategies on three dimensions is possible as the Figure 8 shows on next page.

[20]Machine Learning Operations is a set of practices for operationalizing Machine Learning models. It automates and standardizes the process of building, deploying, and managing Machine Learning models. See also the definition of DevOps. Available on : `https://en.wikipedia.org/wiki/MLOps` (Consulted on january the 3^{rd} 2024)

[21]HuggingFace, Model Parallelism [online]. Available on : `https://huggingface.co/docs/transformers/v4.15.0/parallelism` (Consulted on january the 3^{rd} 2024)

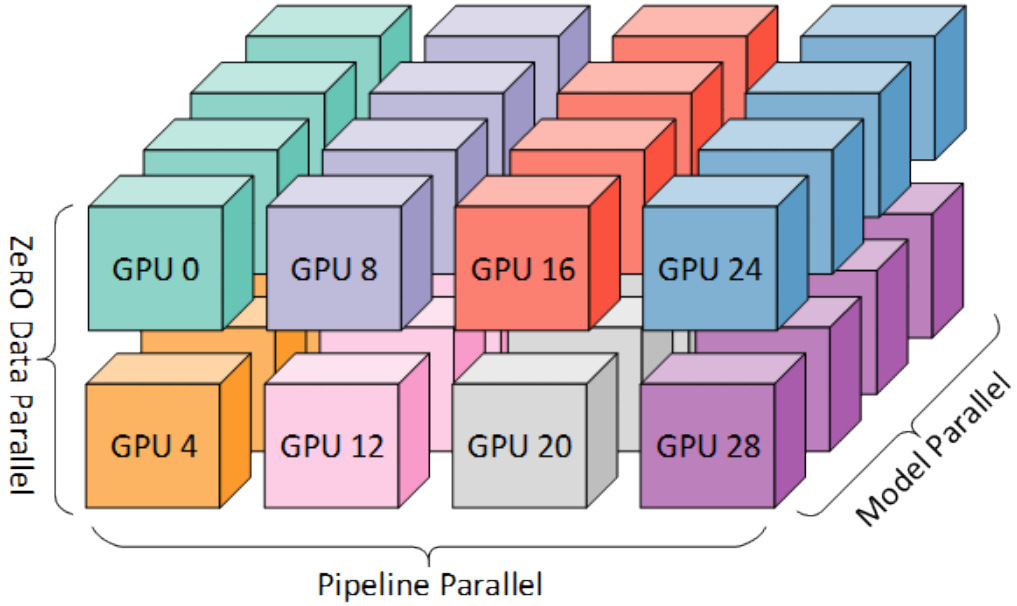


Figure 8: Combination of Data Parallel (DP), Pipeline Parallel (PP) and Tensor Parallel (TP) strategies[22].

All in all, PyTorch Lightning is a very powerful library that I did not have the time to fully explore and that I would be excited to delve into.

3.2.4 Tensorboard

Analyzing, displaying data in graphs and visualizing the processes is a crucial part of the training program. Tensorboard is a visualization toolkit for Machine Learning experimentation that enables tracking and sharing of Deep Learning experiments. It is a powerful tool that allows the user to visualize the training, validation and testing processes, the metrics and the losses continously updated in real-time. It is as fondamental as debugging any other program as it helps addressing potential issues regarding the performance of the model.

Here follows an example of a posteriogram used for understanding the functioning of a wav2vec2 model finetuned on a French dataset for the ASR Speech-to-Phoneme (S2P) task :

[22]HuggingFace, Model Parallelism [online]. Available on : `https://huggingface.co/docs/transformers/v4.15.0/parallelism` (Consulted on january the 3^{rd} 2024)

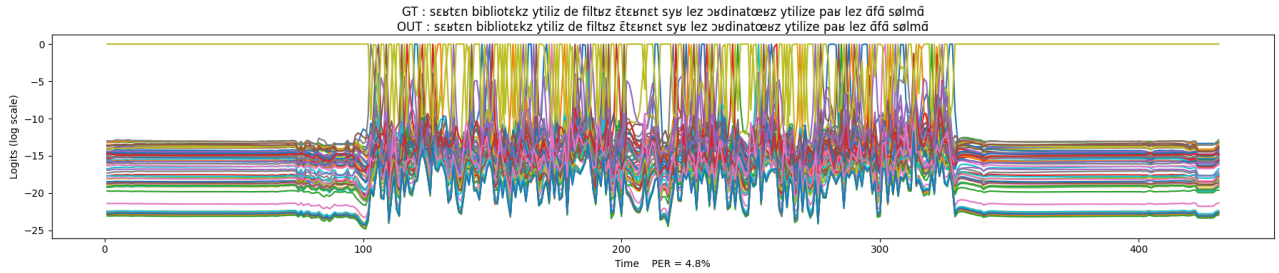


Figure 9: Posteriogram at epoch 6 in validation process.

It pictures the probability of each phoneme being spoken at each time step. The phonemes are represented by the letters of the French IPA alphabet[23] and the blank symbol is represented by the pipe | symbol - which cannot be seen here for readability purposes.

A posteriogram can be generated at each step of the training process or at every epoch of the validation process to insure the model is learning correctly. Indeed, here the model predicts with an error of 4.8% the phonemes of the sentence *certaines bibliothèques utilisent des filtres internet sur les ordinateurs utilisés par les enfants seulement* at epoch 6 in the validation process.

As it is common, Tensorboard also displays the learning rate graph, the training, validation and testing losses graphs, Phoneme Error Rate (PER) and Word Error Rate (WER) graphs in validation or testing, etc.

In light of the aforementioned observations, the training program is meant to be fully versatile, modulable and user-friendly, specifically tailored for the VibraVox dataset and its extensive S2P functionalities, facilitated by the pre-existing clean phonemized transcripts. This enables efficient model fine-tuning for S2P tasks. Additionally, it has been developed to use Distributed Data Parallel (DDP). Nevertheless, additional efforts are required to ensure its compatibility and optimization with models other than wav2vec2, as well as to fully harness the GPUs' maximum potential. Indeed, it could be highly valuable to generalize its use to other models such as whisper or a custom model or company-custom nodes other than Jean ZAY.

As I am not sure it currently works with whisper or a custom model, I will not claim it does. Nevertheless, it is a very interesting project that I would love to continue working on.

[23] Wikipedia, IPA/French [online]. Available on : `https://en.wikipedia.org/wiki/Help:IPA/French` (Consulted on january the 3^{rd} 2024)

4 Technical implementation

4.1 Training a model, how it works

See Annex H for the implementation of the trainer.

Before effectively training a model, the Trainer class *fits* the model to the training, validation and testing datasets. Fitting the model goes through the following phases :

1. Initialization :
   - Set up logging for training progress tracking ;
   - Prepare device (GPU or CPU) for model placement ;
   - Initialize dataloaders for training and validation data ;
   - Initialize callbacks for additional training tasks ;
2. Training loop :
   - At each epoch start, run callbacks for tasks like learning rate scheduling ;
   - For each training batch :
     (a) Fetch a batch of data using the dataloader ;
     (b) Zero gradients to avoid accumulation from previous steps ;
     (c) Forward pass the batch through the model ;
     (d) Calculate and backpropagate loss ;
     (e) Update model parameters ;
     (f) Log training metrics and loss ;
     (g) At batch end, execute callbacks for tasks like logging ;
   - At each epoch end, run callbacks for tasks like checkpointing ;
   - Validate the model on the validation data (no parameter updates) :
     (a) Calculate and log validation metrics and loss ;
     (b) Execute callbacks on validation epoch end like logging or early stopping ;
3. Clean up :
   - At fit end, run callbacks for tasks like saving final models or generating reports ;
   - Release resources and handle potential cleanup tasks ;

The `fit` method orchestrates the entire training process, managing data flow, model optimization, metric evaluation, and callback execution. Because it is a framework, the aforementioned methods can be overriden for custom behavior.

4.2 Strategies implementation

As it would not be necessary nor interesting to review the entire code for the training program, I will only present ones of the few additions made to optimize the finetuning of wav2vec2. Two strategies and hyperparameters from [1] - with a special ModelCheckpoint class - were implemented to optimize the finetuning process :

- A trapezoidal scheduler ;
- A custom ModelCheckpoint callback ;
- A freeze Transformer layers strategy ;

4.2.1 Trapezoidal scheduler

See Annex F for the implementation of the trapezoidal scheduler.

The trapezoidal scheduler is a learning rate scheduler that was introduced by Loshchilov et al. in [14] and used by Baevski et al. in [1]. From Soohwan Kim's _LRScheduler class[24], I implemented :

> *A tri-state schedule where the learning rate is warmed up for the first 10% of updates, held constant for the next 40% and then linearly decayed for the remainder.* [1]

I used learning rates starting at 10^{-5}, with a peak at 10^{-4} and ending at 10^{-7} as the figure 10 shows, also recommended by Baevski et al. [1]

Choosing an appropriate learning rate or scheduler is an important step in the training process. Indeed, the learning rate is one of the most crucial hyperparameters to tune as it controls the rate at which the model learns.[25] Therefore, it is very wise to employ a scheduler that has been proven to efficiently pre-train a specified model, similar to Baevski et al.' [1] approach to pre-train wav2vec2, in order to finetune the same model on a chosen dataset.

4.2.2 Custom ModelCheckpoint callback

See Annex G for the implementation of the custom ModelCheckpoint callback.

Overriding the standard ModelCheckpoint callback from PyTorch Lightning here is useful to save the following :

1. The last trained model, including weights, the vocabulary used for the tokenizer, tokenizer, and processor ;

[24] Soohwan Kim, pytorch-lr-scheduler [online]. Available on : `https://github.com/sooftware/pytorch-lr-scheduler` (Consulted on january the 3^{rd} 2024)

[25] Jason BROWNLEE, Machine Learning Mastery, Understand the Impact of Learning Rate on Neural Network Performance, september the 12^{th} 2020 [online]. Available on : `https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/` (Consulted on december the 20^{th} 2023)

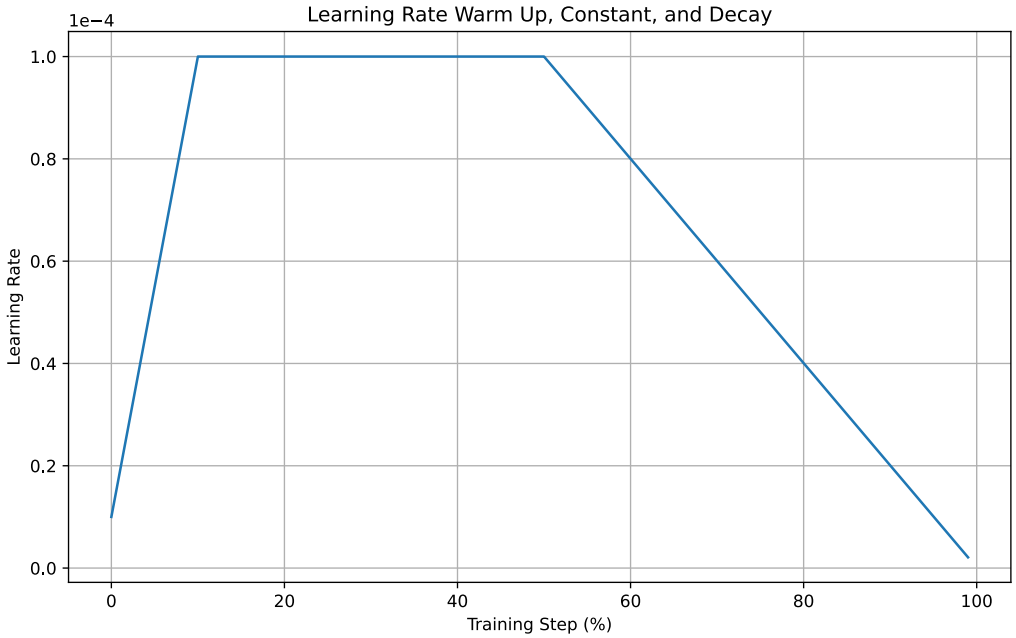


Figure 10: Trapezoidal scheduler.

2. The top k best models by PER or WER, including the same components ;

Which is not possible with the current implementation of the ModelCheckpoint class that solely saves *.ckpt* files i.e. the model state dict, the optimizer state including the exact point when it was saved, the scheduler state and other hyperparameters.

4.2.3 Freeze Transformer layers strategy

Introduced in [1], the freeze Transformer layers strategy consists in freezing the Transformer layers of the wav2vec2 model for a certain number of steps from the start. The model's Transformer encoder seen in Chapter 2.2.4 consists of multiple layers stacked sequentially. In the freeze strategy, instead of training all layers equally, some initial layers are frozen - meaning that their weights remain unchanged, while later layers are unfrozen and allowed to adapt during training.

Since early layers extract general acoustic features, freezing them leverages this generalization without introducing unnecessary complexity to the training process. While later layers capture language-specific information, unfreezing them allows them to specialize in the target language and consequently improve recognition accuracy. Freezing layers reduces the number of trainable parameters, thereby leading to faster training and a smaller memory footprint, thus further reducing training complexity.

Code below shows the implementation of the freeze Transformer layers strategy in the training program. The `on_train_start` method is overridden to freeze the Transformer layers

at the start of the training process. The `training_step` method is overridden to unfreeze the Transformer layers at the specified update `unfreeze_at_step`.

Implementation of the freeze Transformer layers strategy.

```
def on_train_start(self) -> None:
    self.once = False
    if self.unfreeze_at_step > 0:
        logger.info("Entering freeze Transformer layers learning strategy")
        for param in self.model.wav2vec2.encoder.layers.parameters():
            param.requires_grad = False # Freeze Transformer layers weights
        self.once = True

def training_step(self, batch: Dataset, batch_idx: int) -> float:
    if self.once and 0 < self.unfreeze_at_step < self.global_step:
        logger.info("Entering unfreeze Transformer layers learning strategy")
        for param in self.model.wav2vec2.encoder.layers.parameters():
            if not param.requires_grad:
                param.requires_grad = True # Unfreeze Transformer layers
                ↪  weights
        self.once = False # Do not do it again

    #The remainder of the code runs normally

    loss = self.forward(batch).loss

    logits, predicted_ids, target = self.compute_metrics(batch)

    [...]

    return loss
```

We have seen that the freeze strategy leads to a faster convergence thus reducing the training time but only for a good choice of the freezing point. Baevski et al. recommends freezing the Transformer layers for the first 10% of the training process :

> *For the first 10k updates only* [meaning here the 10% of updates] *the output classifier is trained, after which the Transformer is also updated.* [1]

With these strategies in mind, it is now crucial to note that :

- Alternative approaches exist for optimizing finetuning process ;

- These strategies accelerate wav2vec2 convergence but may not generalize to other models ;
- They are empirical heuristics and not theoretical-proven effective strategies ;

Although Baevski et al. spent a lot of time and effort to find the best finetuning techniques, concerns linger about their performance guarantees. Their empirical nature makes it difficult to predict their behavior in different situations and can lead to unexpected failure.

Despite potential limitations in predictability, their simplicity in implementation and potentially lower computational footprint make these strategies attractive options.

5 Experimental evaluation and validation

5.1 Experimental protocol

I rapidly present here the experimental protocol that was followed to preprocess and collect data for the VibraVox dataset [13]. The VibraVox dataset is a general purpose dataset of French speech captured with 5 body-conduction transducers and one airborne microphone. This dataset can be used for various audio Machine Learning tasks such as Automatic Speech Recognition (ASR) in Speech-to-Text (S2T) or Speech-to-Phoneme (S2P), Audio Bandwidth Extension (BWE), Speaker identification and others.[26][27]

5.1.1 Ethical considerations

The participants were informed of the objective of the study and the use of the data collected for research purposes thus they give their consent for data collection. Their voice recordings are anonymized and encrypted during storage and analysis meaning their identities remain completely confidential.

In accordance with the General Data Protection Regulation (GDPR) of the European Union (EU) 2016/679 and the French laws n° 2018-493 of June 20, 2018 on the protection of personal data and n° 2004-801 of August 6, 2004 on the protection of individuals with regard to the processing of personal data and amending the law n° 78-17 of January 6, 1978 on data processing, files and freedoms, the participants have the right to access, rectify, object, erase, restrict and port their personal data, which are their voice recordings.

In addition, the use of Wikipedia text ensures compliance with copyright and licensing issues. Finally, to increase the representativeness and inclusivity of the dataset, a deliberate effort was made to recruit a diverse and gender-balanced group of participants. [13]

5.1.2 Experimental setup

The experimental setup consists of a Zoom F8n multitrack field recorder, a 5^{th}-order ambisonic spatialization sphere and six microphones :

- A mouth headworn reference microphone (Shure WH20XLR) ;
- An in-ear comply foam microphone (Knowles SPH1642HT5H-1 top-port) ;
- An in-ear rigid earpiece microphone (STMicroelectronics MP34DT01) ;
- A temple contact condenser (AKG - C411) ;

[26] CNAM, LMSSC, VibraVox dataset [online]. Available on : `https://vibravox.cnam.fr` (Consulted on january the 2^{nd} 2024)

[27] CNAM, LMSSC, HuggingFace, VibraVox dataset [online]. Available on : `https://huggingface.co/datasets/Cnam-LMSSC/vibravox` (Consulted on january the 2^{nd} 2024)

- A forehead miniature accelerometer (Knowles BU23173-000) ;
- A throat piezoelectric sensor (Ixradio XVTM822D-D35) ;

We employed a controlled acoustic environment for data collection. To minimize background noise, the participant is recorded in a low-reverberation, low-reflection studio. Simultaneous recording with six independent sensors was able thanks to a Zoom F8n multitrack recorder that guarantees precise audio alignment and synchronization. The participant sits within a 5^{th}-order ambisonic sphere outfitted with 56 speakers, to simulate different acoustical environments.

5.1.3 Steps for data collection

The experiment follows four different stages, carefully explained to the participant :

1. Speech without noise (15 minutes) :
   - The participant must read a given French text at a normal pace, without rushing or slowing down, and without making any mistakes, in order to maintain natural pronunciation and clarity throughout the reading ;
   - Each time a mistake is made, the participant must start over from the beginning of the sentence ;
2. Noise without speech (2 minutes 24 seconds) :
   - The participant stays quiet, without speaking, while a pre-recorded noise is played in all directions by the sphere. The participant hears pre-selected noise samples from different categories, each with normalized loudness and dynamically spatialized via the 56-speaker ambisonic sphere. ;
   - Breathing, swallowing, and minor movements are permitted, but speaking is strictly prohibited and moving the head should be avoided ;
   - In-ear foam and rigid earpiece microphones protect participants' hearing from the noise ;
3. No speech & no noise (54 seconds) :
   - The participant stays quiet, without speaking, in a completely quiet environment ;
   - Similar to Phase 2, breathing, swallowing, and slight movements are permissible, while speaking is not and head movements are to be minimized ;
4. Speech with noise (54 seconds) :
   - As in the first Phase, the participant reads the French text while opera pieces are played by the ambisonic sphere ;

- The participant must speak louder and still has to accurately pronounce the words ;

A custom Python frontend application interface was designed to facilitate the recording process : it displays the text to be read, the elapsed time and the remaining time in the form of a progress bar. Four labelled buttons are displayed : *Phrase suivante* to start recording the next sentence, one yellow *Pause* button to pause the recording with a green *Refaire* button overlay on top of the yellow one to retry the current sentence, and a red *Poubelle* button to discard the recording.

See Annex I for an overview of the frontend application interface.

In the writing of this report, the data collection process is still ongoing and the dataset is not fully available yet. Thus, only 121 participants have been recorded so far, providing up to 18 hours of postprocessed audio data to this day.[28]

Data is postprocessed and cleaned using the following techniques :

- Whisper-timestamped Voice Activity Detection (VAD)[29] : employing precise and temporally extended whisper timestamps with added security, this VAD method pinpoints speech segments within recordings with great accuracy ;

- Energy-based VAD : this method analyzes audio energy levels to distinguish speech from silence or background noise, targeting relevant content ;

- Sensor-specific removal criteria : each sensor type undergoes tailored data removal processes to discard very low quality or incoherent audio and optimize signal quality ;

The refined data is divided into two main sets :

- ASR dataset focusing on speech recognition tasks ;

- BWE dataset prioritizing bandwidth extension tasks ;

Each dataset is available in three different splits :

- Train split (80%) ;

- Validation split (10%) ;

- Test split (10%) ;

[28] CNAM, LMSSC, HuggingFace, VibraVox dataset [online]. Available on : `https://huggingface.co/datasets/Cnam-LMSSC/vibravox` (Consulted on january the 2^{nd} 2024)

[29] LinTO.ai, whisper-timestamped [online]. Available on : `https://github.com/linto-ai/whisper-timestamped` (Consulted on january the 3^{rd} 2024)

5.2 Performance metrics

To evaluate the model performance both in validation and testing processes, we use the following metrics :

- For Speech-to-Text (S2T) task, the Word Error Rate (WER) metric computes the percentage of words incorrectly predicted.
- For Speech-to-Phoneme (S2P) task, the Phoneme Error Rate (PER) metric calculates the percentage of incorrectly predicted phonemes.

According to TorchMetrics documentation[30], the WER is defined as follows :

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \tag{12}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, C is the number of correct words and $N = S + D + C$ is the number of words in the reference (truth) transcription.

The code below illustrates a use of the WER metric.

Example of the WER metric.

```
from torchmetrics.text import WordErrorRate
preds = "This is a prediction"
target = "This is a target"
wer = WordErrorRate()

wer.update(preds=preds, target=target)

print(wer.compute())

>>> tensor(0.2500)
```

The PER used for the S2P task is simply a Character Error Rate (CER) metric adapted to phonemes, the smallest units of speech equivalent to characters in text. This is justified because the model's encoder and decoder operate on the French IPA alphabet rather than the Latin alphabet. However the CER is defined as follows :

$$CER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \tag{13}$$

[30] Lightning AI, PorchMetrics [online]. Available on : `https://lightning.ai/docs/torchmetrics/` (Consulted on january the 5^{th} 2024)

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, C is the number of correct characters and $N = S+D+C$ is the number of characters in the reference (truth) transcription.[31]

Considering two variables `preds` and `target` :
`preds` = ãʃãte ʒə mapɛl ʒyljɛ̃n i.e. *Enchanté, je m'appelle Julien*
`target` = ãʃãte ʒə mapɛl malo i.e. *Enchanté, je m'appelle Malo*

The code below illustrates a use of the PER metric.

Example of the PER metric.

```
from torchmetrics.text import CharErrorRate

per = CharErrorRate()

per.update(preds=preds, target=target)

print(per.compute())

>>> tensor(0.2632)
```

For both these metrics, the lower the better.

5.3 Results and analysis

For reproducibility, I compare WER results from the wav2vec2 model Baevski et al. finetuned with ours on the same dataset, with the same number of hours. Thereafter, I delve into PER results of the facebook/wav2vec2-base-fr-voxpopuli-v2[32] [15] model Meta pretrained on French unlabeled audio that we finetuned on Librispeech French 1076h and Commonvoice French 1000h datasets. Finally, I analyze the performance of the same Meta-pretrained model we finetuned on the 121-participant partial VibraVox dataset.

[31] Lightning AI, PorchMetrics [online]. Available on : `https://lightning.ai/docs/torchmetrics/` (Consulted on january the 5^{th} 2024)

[32] Meta (Facebook), HuggingFace, wav2vec2-base-fr-voxpopuli-v2 [online]. Available on : `https://huggingface.co/facebook/wav2vec2-base-fr-voxpopuli-v2` (Consulted on january the 5^{th} 2024)

5.3.1 Reproducibility of the results

The table 2 shows the results of the Baevski-finetuned wav2vec2 model [1] compared with ours :

Use case	wav2vec2 pre-trained size model	Number of parameters	Dataset name	Number of hours finetuned on	Language model	Validation clean WER ([1])	Test clean WER ([1])
Speech-to-Text	Base	95M	Librispeech-960h	Librispeech-100h labeled	No LM	6.2% (6.1%)	6.2% (6.1%)
Speech-to-Text	Base	95M	Librispeech-960h	Librispeech-1h labeled	No LM	21.7% (24.1%)	22.5% (24.5%)

Table 2: Reproducibility of results from [1].

5.3.2 wav2vec2 performance on S2P task with our training program

For S2P evaluation, we finetuned the model on the Librispeech French 1076h dataset and the Commonvoice French 1000h dataset. However, it was necessary to fully phonemize the transcripts of these datasets using a phonemizer[33] [16] as they are only fully Latin-character textual.

The Table 3 shows the results of our finetuned, Meta-pretrained, facebook/wav2vec2-base-fr-voxpopuli-v2 model on Librispeech and Commonvoice datasets :

Use case	Dataset finetuned on	Validation clean PER	Test clean PER
Speech-to-Phoneme	Librispeech French 1076h	4.2%	3.4%
Speech-to-Phoneme	Commonvoice French 1000h	4.3%	5.5%

Table 3: facebook/wav2vec2-base-fr-voxpopuli-v2 95M pretrained on 22.8k hours of French unlabeled audio [15] finetuned on Librispeech and Commonvoice.

[33] bootphon, CoML, phonemizer [online]. Available on : `https://github.com/bootphon/phonemizer` (Consulted on january the 5^{th} 2024)

5.3.3 VibraVox performance

The Table 4 shows the results of our finetuned, Meta-pretrained, facebook/wav2vec2-base-fr-voxpopuli-v2 model on the 121-participant partial VibraVox dataset :

Use case	Subset	Number of hours finetuned on	Validation clean PER	Test clean PER
Speech-to-Phoneme	ASR mouth headworn reference microphone	14h	6.4%	7.0%
Speech-to-Phoneme	ASR forehead miniature accelerometer	13h	13%	14%
Speech-to-Phoneme	ASR in-ear rigid earpiece microphone	14h	21%	22%
Speech-to-Phoneme	ASR throat piezoelectric sensor	14h	22%	21%
Speech-to-Phoneme	ASR in-ear comply foam microphone	14h	23%	24%
Speech-to-Phoneme	ASR temple contact microphone	13h	44%	43%

Table 4: facebook/wav2vec2-base-fr-voxpopuli-v2 95M pretrained on 22.8k hours of French unlabeled audio [15] finetuned on VibraVox.

5.4 Limitations and future directions

As the Table 2 shows, we succesfully reproduced the results from [1] on the Librispeech French 960h dataset on 100h and 1h. For further experiments and to ensure the reproducibility of the training program performance, we shall finetune wav2vec2 on the Librispeech French 960h dataset on other subsets and on the *other* set.

The French Librispeech dataset exhibits substantial post-processing artifacts, jeopardizing our ability to achieve optimal validation and test PER performance. Indeed, we believe the training dataset was misbuilt : some transcriptions don't match the audio. The origin of this discrepancy is unknown and we were not able to fix it. Consequently, the model completely fails to transcribe certain words, such as *chapitre* and numbers, thereby contributing to the suboptimal PER results.

The Commonvoice dataset being very noisy, we were not able to obtain better validation or test PER results. While the dataset is open-source, large, containing a diverse range of speakers, including various accents, ages, and backgrounds and said to be verified, the data collected is not always of good quality. Indeed, it is a community-driven project :

- The collected data is of variable quality, sometimes very poor due to various recording environments and equipment ;

- With community contributions, there's a higher chance of duplicate recordings from the same speakers, potentially biasing the dataset towards specific demographics or accents ;

This explains why test PER results are not better than validation PER results.

However, as you can see on Table 4, the VibraVox dataset, for all subsets, outperforms both the Librispeech French 1076h and the Commonvoice French 1000h datasets in terms of both validation and test PER with a limited corpus of 13 hours. This is very promising as this achievement shows a great quality of the recordings captured with the following microphones :

- The mouth headworn reference microphone : this reference microphone is a pivot for comparison with the other microphones ;
- The forehead miniature accelerometer ;
- The in-ear rigid earpiece microphone ;
- The throat piezoelectric sensor ;
- The in-ear comply foam microphone ;

The temple contact microphone is the only one that provides poor performance due to its significantly lower signal-to-noise ratio.

The results suggest that the body conduction wave transmission introduces a low-pass filter, removing essential frequency components that the wav2vec model struggles to reconstruct. These preliminary results underscore the need for a bandwidth extension technique to enhance the audio signal and improve the performance of the model.

In summary, the VibraVox dataset has demonstrated remarkable performance even with a considerably smaller data volume than the Librispeech French 1076h and Commonvoice French 1000h datasets thanks to the time and efforts given by our recording team to collect clean and high-quality audio data. These findings highlight the potential of the VibraVox dataset for improving speech recognition in real-world applications, particularly those involving body-conducted microphone configurations.

6 Personal review

My internship at the Conservatoire National des Arts et Métiers (CNAM) focused on two primary objectives: creating the VibraVox dataset and developing a training program to finetune wav2vec2 with this dataset.

We meticulously followed the experimental protocol and recorded over a hundred participants, resolving and tackling any problems that arose during the process like the lack of participants by recruiting students from the Centre de Préparation au Diplôme d'État

d'Audioprothésiste (CPDA)[34]. Another challenge was addressing technical issues by replacing faulty microphones. It was also crucial to ensure that the participants were comfortable and relaxed during the recording process, by explaining them the purpose of the project and their valuable contribution. While post-processing, cleaning and uploading the dataset were tedious tasks, the dataset is now larger and cleaner than before my arrival at the CNAM and it is something I am proud of.[35]

Though new to research, experimental sciences and Artificial Intelligence, I successfully developed a training program that adhered to the specifications and constraints given by Éric BAVU and Julien HAURET. Thanks to my previous experience in software development, I was able to quickly learn and adapt to the research environment, a new experience for me. I embraced new concepts and techniques like PyTorch and tensor manipulation, tools I had never used before. Research field has other considerations and constraints than software development, such as the reproducibility of the results, which became a new challenge for me. Nevertheless, my work really helped Julien HAURET and I consider working with him in the future. These both efforts aimed to contribute to research on degraded speech, ultimately leading to advancements in hearing aids - a possible field of application of Julien's thesis 1.

The potential of this project to positively impact people's lives is a strong motivator for me as I would love to continue on a thesis with Éric BAVU on a similar subject. I am passionate about working in the field of science, and I am grateful for this chance, which has allowed me to explore my interest in AI. I enjoyed recording participants, mostly CPDA students, explaining them the purpose of the project, the importance of their contribution and the work I did.

I was first introduced to AI and Deep Learning, both theoretically and practically, two months prior to my arrival at the CNAM and I found myself astonished by the complexity of the field and the possibilities it offers. This opportunity accelerated my learning curve, not only in AI and Deep Learning but also in the specific domain of acoustics and degraded speech. I am proud of the significant knowledge and skills I gained during this time.

I enjoyed working with Éric BAVU and Julien HAURET as they were expert and dedicated for this project, they did help me a lot on this project and supported me and I am grateful for the opportunity they gave me. I had a lot of fun with the PhD students and the researchers at the Laboratoire de Mécanique des Structures et des Systèmes Couplés (LMSSC)[36], exchanging about our passions whether it be computer sciences, mechanics or simply music. I hope to have the chance to work with them in the future, to benefit the vast majority.

[34] `https://cpda.cnam.fr`

[35] CNAM, LMSSC, HuggingFace, VibraVox dataset [online]. Available on : `https://huggingface.co/datasets/Cnam-LMSSC/vibravox` (Consulted on january the 3^{rd} 2024)

[36] `https://www.lmssc.cnam.fr/`

7 Conclusion and perspectives

Taken together, the successful development of the VibraVox dataset and the design of an Automatic Speech Recognition (ASR) system for unconventional transducers represent significant strides in tackling low quality and low intelligibility degraded speech. The bandwidth extension technique Julien HAURET is working on gives a promising solution to enhance audio signals across the board. It could bring substantial improvements in the hearing aid industry. Other than hearing implants, the degraded speech field has potential applications and I believe it is not limited to speech. This technology could revolutionize how we restore any degraded signal.

The insights gained from the VibraVox project hold immense potential beyond just speech recognition. The principles of noise removal and signal enhancement can be applied to any audio signal, or even other forms of data like images and videos. Imagine maximizing the signal-to-noise ratio of any transmission device, from medical scans to environmental sensors. There indeed exists a few applications in the medical field like in Choi et al. [17] or in Koonjoo et al. [18] for terahertz imaging and MRI enhancement. The possibilities extend far beyond medicine. From astronomical spectroscopy to industrial quality control, the ability to use these principles to various data types is truly exciting. I imagine a future where weak signals become a thing of the past, and I hope the progress achieved in this project will help make that future a reality.

> *Low SNR is not a unique challenge ; it is a universal problem in data collection ; therefore, we might see more innovative ways to overcome it in the future.* - Aya Takase[37]

□

References

[1] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33:12449–12460, 2020.

[2] Julien Hauret, Thomas Joubaud, Véronique Zimpfer, and Éric Bavu. Eben: Extreme bandwidth extension network applied to speech signals captured with noise-resilient body-conduction microphones. *arXiv preprint arXiv:2210.14090*, pages 1–5, 2023.

[3] Pierre Lecomte. Three dimensional higher order ambisonics : Sound field capture, transformations and adaptive decoding. *tel-01544953*, 2016.

[37] Aya Takase, Rigaku, How to Improve the Signal-to-noise Ratio of X-ray CT Images [online]. Available on : `https://imaging.rigaku.com/blog/how-improve-signal-noise-ration-xray-ct-images` (Consulted on january the 5^{th} 2024)

[4] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. pages 369–376, 2006.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 33:5998–6008, 2023.

[6] Qingyao Qiao. Development of a holistic machine learning-based approach for building energy consumption prediction under limited data conditions. 2023.

[7] Junlai Wu Feifei Gou Xia Yin Yunqian Long Jun Pu, Wenfang Song. Pinn-based method for predicting flow field distribution of the tight reservoir after fracturing. *Geofluids, vol. 2022, Article ID 1781388, 10 pages*, 2022.

[8] Tom Bäckström, Okko Räsänen, Abraham Zewoudie, Pablo Pérez Zarazaga, Liisa Koivusalo, Sneha Das, Esteban Gómez Mellado, Marieum Bouafif Mansali, Daniel Ramos, Sudarsana Kadiri, and Paavo Alku. *Introduction to Speech Processing*. 2 edition, 2022.

[9] Alexander Amini, Ava Soleimany, and Sertac Karaman. Spatial uncertainty sampling for end to end control. *arXiv preprint arXiv:1805.04829*, 2019.

[10] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649, 2013.

[11] Awni Hannun. Sequence modeling with ctc. *10.23915/distill.00008*, 2017.

[12] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. 2022.

[13] Julien Hauret, Malo Olivier, Thomas Joubaud, Christophe Langrenne, Sarah Poirée, Véronique Zimpfer, and Éric Bavu. Vibravox: A general purpose database of speech captured with body-conduction microphones. *in writing*, 2024.

[14] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[15] Changhan Wang, Morgane Rivière, Ann Lee, Anne Wu, Chaitanya Talnikar, Daniel Haziza, Mary Williamson, Juan Pino, and Emmanuel Dupoux. Voxpopuli: A large-scale multilingual speech corpus for representation learning, semi-supervised learning and interpretation. 2021.

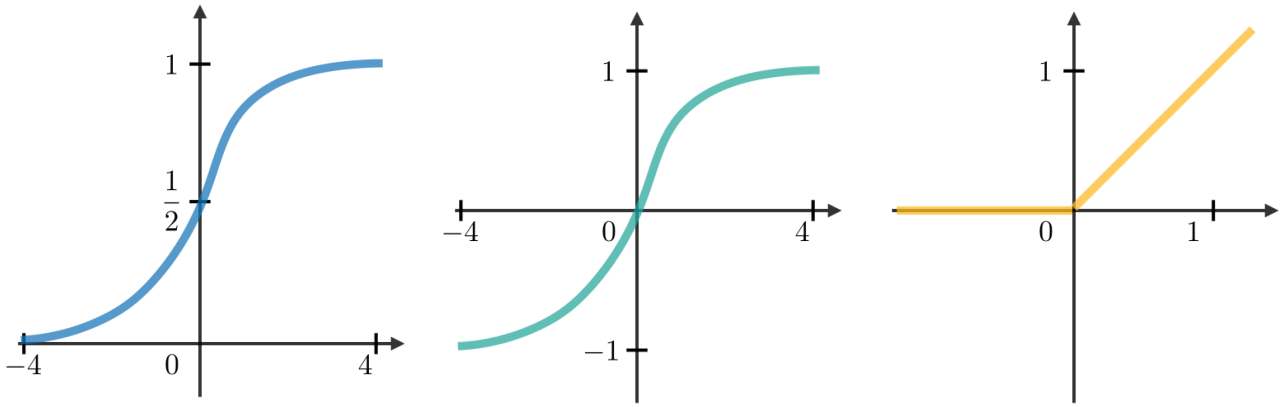
[16] Mathieu Bernard and Hadrien Titeux. Phonemizer: Text to phones transcription for multiple languages in python. *Journal of Open Source Software*, 6(68):3958, 2021.

[17] Hyunkook Choi, Sangmin Kim, Inhee Maeng, Joo-Hiuk Son, and Hochong Park. Improving signal-to-noise ratio of a terahertz signal using a wavenet-based neural network. *Opt. Express*, 30(4):5473–5485, Feb 2022.

[18] N. Koonjoo, B. Zhu, G. Cody Bagnall, D. Bhutto, and M. S. Rosen. Boosting the signal-to-noise of low-field mri with deep learning image reconstruction. *Scientific Reports*, 2021.

Annexes

Annex A: Commonly used activation functions



sigmoid σ 4 *tanh* 5 *ReLU* 6

Figure 11: Representative curves of functions *sigmoid*, *tanh* and *ReLU*[38]

[38] Afshine AMIDI, Shervine AMIDI, Deep Learning cheatsheet [online]. Available on : `https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning` (Consulted on december the 20^{th} 2023)

Annex B: Microphones

Figure 12: Mouth headworn reference microphone (Shure WH20XLR)

Figure 13: In-ear comply foam microphone (Knowles SPH1642HT5H-1 top-port)

Figure 14: Forehead miniature accelerometer (Knowles BU23173-000)

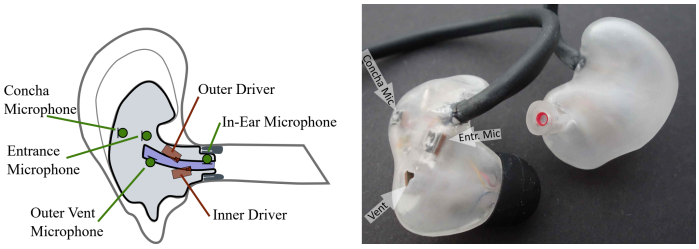


Figure 15: In-ear rigid earpiece microphone (STMicroelectronics MP34DT01)

Figure 16: Temple contact condenser (AKG - C411)

Figure 17: Throat piezoelectric sensor (Ixradio XVTM822D-D35)

Figure 18: The six microphones used to record VibraVox dataset[39].

Annex C: Ambisonic spatialization sphere

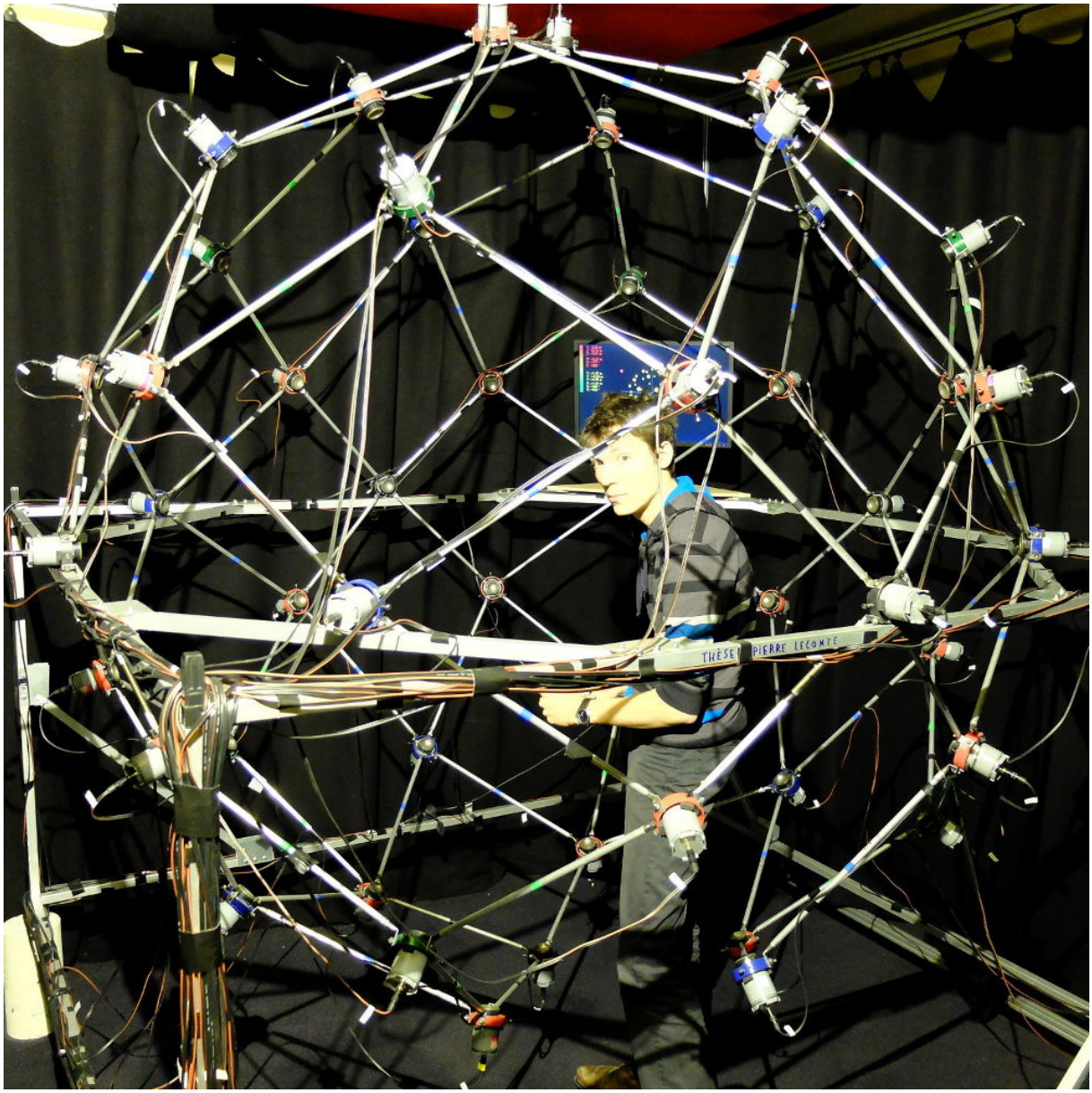


Figure 19: *Spherebedev* 5^{th}-order ambisonic spatialization sphere developed during Pierre Lecomte's PhD [3].

[39] CNAM, LMSSC, VibraVox dataset [online]. Available on : `https://vibravox.cnam.fr` (Consulted on january the 2^{nd} 2024)

Annex D: Noise recorder

Figure 20: ZYLIA ZR-1 Portable Recorder, opera audio recording microphone[40].

Annex E: Multitrack field recorder

Figure 21: Zoom F8n multitrack field recorder for synchronized recording[41].

[40]CNAM, LMSSC, VibraVox dataset [online]. Available on : `https://vibravox.cnam.fr` (Consulted on january the 2^{nd} 2024)

[40]ZYLIA, Zylia ZR-1 Portable Recorder [online]. Available on : `https://www.zylia.co/zylia-zr-1-portable-recorder.html` (Consulted on january the 2^{nd} 2024)

Annex F: Trapezoidal scheduler

Modification of Soohwan Kim's Tri-Stage Learning Rate Scheduler in PyTorch[42].

```
# MIT License
#
# Copyright (c) 2021 Soohwan Kim
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
↪  all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

import torch
from typing import Optional
from torch.optim import Optimizer
from torch.optim.lr_scheduler import _LRScheduler


class TriStageLRScheduler(_LRScheduler):
    r"""
    Tri-Stage Learning Rate Scheduler. Implement the learning rate scheduler in
    ↪  "SpecAugment"
```

[41] CNAM, LMSSC, VibraVox dataset [online]. Available on : `https://vibravox.cnam.fr` (Consulted on january the 2^{nd} 2024)

[41] Zoom Corp, The Zoom F8n [online, product discontinued]. Available on : `https://zoomcorp.com/en/us/field-recorders/field-recorders/f8n/` (Consulted on january the 2^{nd} 2024)

```
    Args:
        optimizer (Optimizer): Optimizer.
        init_lr (float): Initial learning rate.
        peak_lr (float): Maximum learning rate.
        final_lr (float): Final learning rate.
        init_lr_scale (float): Initial learning rate scale.
        final_lr_scale (float): Final learning rate scale.
        warmup_steps (int): Warmup the learning rate linearly for the first N
        ↪  updates.
        hold_steps (int): Hold the learning rate for the N updates.
        decay_steps (int): Decay the learning rate linearly for the first N
        ↪  updates.
        total_steps (int): Total steps in training.
    """

    def __init__(
        self,
        optimizer: Optimizer,
        init_lr: float,
        peak_lr: float,
        final_lr: float,
        init_lr_scale: float,
        final_lr_scale: float,
        warmup_steps: int,
        hold_steps: int,
        decay_steps: int,
        total_steps: int,
    ):
        assert isinstance(warmup_steps, int), "warmup_steps should be integer
        ↪  type"
        assert isinstance(total_steps, int), "total_steps should be integer
        ↪  type"
        assert isinstance(hold_steps, int), "hold_steps should be integer type"
        assert isinstance(decay_steps, int), "decay_steps should be integer
        ↪  type"

        super(_LRScheduler, self).__init__()
        self.optimizer = optimizer
        self.init_lr = init_lr
        self.init_lr *= init_lr_scale
        self.final_lr = final_lr
        self.final_lr *= final_lr_scale
```

```
        self.peak_lr = peak_lr
        self.warmup_steps = warmup_steps
        self.hold_steps = hold_steps
        self.decay_steps = decay_steps

        self.warmup_rate = (
            (self.peak_lr - self.init_lr) / self.warmup_steps
            if self.warmup_steps != 0
            else 0
        )
        self.decay_rate = (
            (self.peak_lr - self.final_lr) / self.decay_steps
            if self.decay_steps != 0
            else 0
        )

        self.lr = self.init_lr
        self.update_steps = 0

    def _decide_stage(self):
        if self.update_steps < self.warmup_steps:
            return 0, self.update_steps

        offset = self.warmup_steps

        if self.update_steps < offset + self.hold_steps:
            return 1, self.update_steps - offset

        offset += self.hold_steps

        if self.update_steps <= offset + self.decay_steps:
            # decay stage
            return 2, self.update_steps - offset

        offset += self.decay_steps

        return 3, self.update_steps - offset

    def step(self, val_loss: Optional[torch.FloatTensor] = None):
        stage, steps_in_stage = self._decide_stage()

        if stage == 0:
```

```
            self.lr = self.init_lr + self.warmup_rate * steps_in_stage
        elif stage == 1:
            self.lr = self.peak_lr
        elif stage == 2:
            self.lr = self.peak_lr - self.decay_rate * steps_in_stage
        elif stage == 3:
            self.lr = self.final_lr
        else:
            raise ValueError("Undefined stage")

        self.set_lr(self.optimizer, self.lr)
        self.update_steps += 1

        return self.lr

    @staticmethod
    def set_lr(optimizer, lr):
        for g in optimizer.param_groups:
            g["lr"] = lr

    def get_lr(self):
        for g in self.optimizer.param_groups:
            return g["lr"]
```

Annex G: Custom ModelCheckpoint callback

Override of the ModelCheckpoint callback class[43].

```
import logging
import shutil
from datetime import timedelta
from typing import Dict, Optional

import pytorch_lightning as pl
import torch
from pytorch_lightning.callbacks import ModelCheckpoint
```

[42] Soohwan Kim, pytorch-lr-scheduler [online]. Available on : https://github.com/sooftware/pytorch-lr-scheduler (Consulted on january the 3rd 2024)

```
from pytorch_lightning.utilities import rank_zero_info, rank_zero_only
from pytorch_lightning.utilities.types import _METRIC, _PATH
from transformers.feature_extraction_sequence_utils import
↪  SequenceFeatureExtractor

logger: logging.Logger = logging.getLogger(__name__)


class ASRCheckpoint(ModelCheckpoint):
    """
    Inherits ModelCheckpoint class and its attributes.

    Args:
        feature_extractor (`SequenceFeatureExtractor`): Feature extractor to
        ↪  save.
    """

    def __init__(
        self,
        feature_extractor: SequenceFeatureExtractor,
        dirpath: Optional[_PATH] = None,
        filename: Optional[str] = None,
        monitor: Optional[str] = None,
        verbose: bool = False,
        save_last: Optional[bool] = None,
        save_top_k: int = 1,
        save_weights_only: bool = False,
        mode: str = "min",
        auto_insert_metric_name: bool = True,
        every_n_train_steps: Optional[int] = None,
        train_time_interval: Optional[timedelta] = None,
        every_n_epochs: Optional[int] = None,
        save_on_train_epoch_end: Optional[bool] = None,
        every_n_val_epochs: Optional[int] = None,
    ):
        super(ASRCheckpoint, self).__init__(
            dirpath,
            filename,
            monitor,
            verbose,
            save_last,
            save_top_k,
```

```
            save_weights_only,
            mode,
            auto_insert_metric_name,
            every_n_train_steps,
            train_time_interval,
            every_n_epochs,
            save_on_train_epoch_end,
            every_n_val_epochs,
        )
        self.feature_extractor = feature_extractor
        self.path = dirpath + filename
        self.last_path = ""

    def _save_last_checkpoint(
        self, trainer: "pl.Trainer", monitor_candidates: Dict[str, _METRIC]
    ) -> None:
        """
        Inherits _save_last_checkpoint.

        Args:
            trainer (`pl.Trainer`): pl.Trainer instance.
            monitor_candidates (Dict[str, _METRIC]): Contains valuable
            ↪  attributes of ModelCheckpoint.
        Returns:
            NoneType: None
        """
        if not self.save_last:
            return

        filepath = self.format_checkpoint_name(
            monitor_candidates, self.CHECKPOINT_NAME_LAST
        )
        trainer.save_checkpoint(filepath, self.save_weights_only)

        """
        OVERRIDDEN
        """
        path = self.format_checkpoint_name(
            metrics=monitor_candidates, filename=self.path
        )
        self._save_model(trainer=trainer, filepath=path,
        ↪  del_filepath=self.last_path)
```

```
        if self.last_model_path and self.last_model_path != filepath:
            ↪ trainer.training_type_plugin.remove_checkpoint(self.last_model_path)

        self.last_model_path = filepath
        self.last_path = path

    def _update_best_and_save(
        self,
        current: torch.Tensor,
        trainer: "pl.Trainer",
        monitor_candidates: Dict[str, _METRIC],
    ) -> None:
        """
        Inherits _update_best_and_save.

        Args:
            current (torch.Tensor):
            trainer (`pl.Trainer`): pl.Trainer instance.
            monitor_candidates (Dict[str, _METRIC]): Contains valuable
            ↪ attributes of ModelCheckpoint.
        Returns:
            NoneType: None
        """
        k = len(self.best_k_models) + 1 if self.save_top_k == -1 else
        ↪ self.save_top_k

        del_filepath = None
        if len(self.best_k_models) == k and k > 0:
            del_filepath = self.kth_best_model_path
            self.best_k_models.pop(del_filepath)

        # do not save nan, replace with +/- inf
        if isinstance(current, torch.Tensor) and torch.isnan(current):
            current = torch.tensor(
                float("inf" if self.mode == "min" else "-inf"),
                ↪ device=current.device
            )

        filepath = self._get_metric_interpolated_filepath_name(
            monitor_candidates, trainer, del_filepath
```

```
        )

        # save the current score
        self.current_score = current
        self.best_k_models[filepath] = current

        if len(self.best_k_models) == k:
            # monitor dict has reached k elements
            _op = max if self.mode == "min" else min
            self.kth_best_model_path = _op(
                self.best_k_models, key=self.best_k_models.get
            )
            self.kth_value = self.best_k_models[self.kth_best_model_path]

        _op = min if self.mode == "min" else max
        self.best_model_path = _op(self.best_k_models,
        ↪  key=self.best_k_models.get)
        self.best_model_score = self.best_k_models[self.best_model_path]

        if self.verbose:
            epoch = monitor_candidates.get("epoch")
            step = monitor_candidates.get("step")
            rank_zero_info(
                f"Epoch {epoch:d}, global step {step:d}: {self.monitor} reached
                ↪  {current:0.5f}"
                f' (best {self.best_model_score:0.5f}), saving model to
                ↪  "{filepath}" as top {k}'
            )
        trainer.save_checkpoint(filepath, self.save_weights_only)

        """
        OVERRIDDEN
        """
        self._save_model(trainer=trainer, filepath=filepath,
        ↪  del_filepath=del_filepath)

        if del_filepath is not None and filepath != del_filepath:
            trainer.training_type_plugin.remove_checkpoint(del_filepath)

    @rank_zero_only
    def _save_model(
        self, trainer: "pl.Trainer", filepath: _PATH, del_filepath: _PATH
```

```
    ) -> None:
        """

        Args:
            trainer (`pl.Trainer`): pl.Trainer instance.
            filepath (_PATH): Absolute path to checkpoint to save.
            del_filepath (_PATH): Absolute path to checkpoint to save that must
            ↪  be replaced by newest/best checkpoint.
        Returns:
            NoneType: None
        """
        logger.info("Saving best/last model")
        trainer.lightning_module.model.save_pretrained(
            save_directory=filepath.replace(".ckpt", "")
        )
        trainer.lightning_module.tokenizer.save_pretrained(
            save_directory=filepath.replace(".ckpt", "")
        )
        self.feature_extractor.save_pretrained(
            save_directory=filepath.replace(".ckpt", "")
        )
        if del_filepath is not None and filepath != del_filepath:
            try:
                logger.info("Deleting and replacing best/last model")
                shutil.rmtree(del_filepath.replace(".ckpt", ""))
            except:
                None
```

Annex H: Model trainer

Implementation of the model trainer program.

```
import datetime
import logging
import os
```

[43] Overriden code snippet from PyTorch Lightning, Model Checkpoint class [online]. Available on : `https://lightning.ai/docs/pytorch/stable/api/lightning.pytorch.callbacks.ModelCheckpoint.html` (Consulted on january the 3^{rd} 2024)

```python
import warnings
import hydra
from math import ceil
from omegaconf import DictConfig

# Disable annoying warnings from Huggingface transformers
warnings.filterwarnings("ignore", category=UserWarning, module="transformers")

# Set environment variables for Huggingface cache, for datasets and
↪  transformers models
# (should be defined before importing datasets and transformers modules)
dir_huggingface_cache_path: str = "/home/Donnees/Data/Huggingface_cache"
os.environ["HF_HOME"] = dir_huggingface_cache_path
os.environ["HF_DATASETS_CACHE"] = dir_huggingface_cache_path + "/datasets"
os.environ["TRANSFORMERS_CACHE"] = dir_huggingface_cache_path + "/models"

# Set environment variables for full trace of errors
os.environ["HYDRA_FULL_ERROR"] = "1"
dir_path: str =
↪  str(os.path.abspath(os.path.dirname(os.path.abspath(__file__))))
now: str = datetime.datetime.now().strftime("%Y-%m-%d/%H-%M-%S/")

logger: logging.Logger = logging.getLogger(__name__)


@hydra.main(config_path="../../configs", config_name="config")
def main(cfg: DictConfig):
    """
    Args:
        cfg (`DictConfig`): The config file containing parameters.
    """
    datamodule = hydra.utils.instantiate(
        cfg.data_module,
        _recursive_=False,
    )
    nn_module = hydra.utils.instantiate(
        cfg.neural_network,
        nb_steps_per_epoch=ceil(
            datamodule.get_nb_samples_train()
            / (cfg.trainer.gpus * cfg.batch_size *
            ↪  cfg.trainer.accumulate_grad_batches)
        ),
```

```
        tokenizer=datamodule.processor.tokenizer,
        _recursive_=False,
    )
    trainer = hydra.utils.instantiate(
        cfg.trainer,
        log_every_n_steps=ceil(
            datamodule.get_nb_samples_train()
            / (
                cfg.trainer.gpus
                * cfg.batch_size
                * cfg.trainer.accumulate_grad_batches
                * cfg.log_n_times_per_epoch
            )
        ),
        callbacks=[
            hydra.utils.instantiate(
                cfg.callbacks.checkpoint_callback_last,
                feature_extractor=datamodule.processor.feature_extractor,
            ),
            hydra.utils.instantiate(
                cfg.callbacks.checkpoint_callback_top_k,
                feature_extractor=datamodule.processor.feature_extractor,
            ),
            hydra.utils.instantiate(cfg.callbacks.lr_logger_callback),
            hydra.utils.instantiate(cfg.callbacks.early_stopping_callback),
            hydra.utils.instantiate(cfg.callbacks.rich_model_summary),
        ],
    )
    logger.info("Trainer trains with model")
    trainer.logger.experiment.add_hparams(
        hparam_dict=hydra.utils.instantiate(cfg.hparam_dict),
        metric_dict={"key": 0.0},
        run_name="./",
    )
    trainer.fit(model=nn_module, datamodule=datamodule)
    trainer.test(
        model=nn_module,
        datamodule=datamodule,
        verbose=True,
    )
```

```
if __name__ == "__main__":
    main()
    logger.info("/********************/")
```

Annex I: Frontend interface for data collection

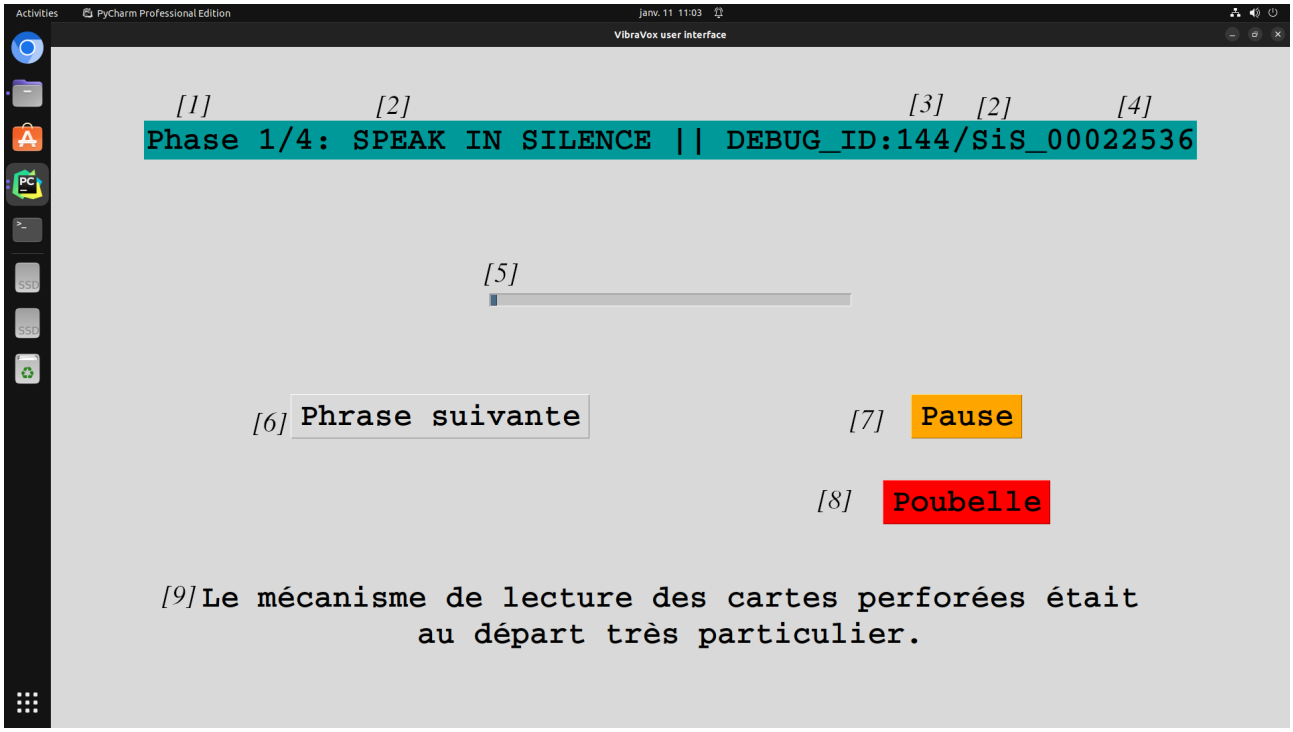


Figure 22: Frontend interface for data collection.

1 : Current recording phase ;
2 : Current phase label ;
3 : Participant id number ;
4 : Sentence id number ;
5 : Progress bar ;
6 : *Next sentence* button ;
7 : *Pause* button ;
8 : *Discard* button ;
9 : Current sentence to read ;
The participant reads on a graphic tablet and clicks the buttons with an electronic pen.

Graduation project to design an intelligent speech transcription system for unconventional data [R&D]

Malo OLIVIER

Abstract

In order to support the thesis work of Mr. Julien HAURET, the aim of this final project is to contribute to the development of the VibraVox dataset and the design of an Automatic Speech Recognition (ASR) system using unconventional data. A methodology based on the use of Deep Neural Networks and Transformer layers has been implemented. The finetuning of pre-trained models is carefully adjusted using implementation strategies and the hyperparameters described by Baevski et al. [1] are adopted. The experiments aim to analyze the performance differences between different microphones and quantify the impact of a speech enhancement system in pre-processing. A total of 200 participants (50 hours) will follow an experimental protocol to record audio data using different body-conducted microphones. A scientific article summarizes these findings and conclusions.

Keywords: Body conduction microphones ; Automatic Speech Recognition ; Deep Neural Networks ; Transformer layers ; Experimental sciences.

Résumé

Afin de soutenir les travaux de thèse de M. Julien HAURET, ce projet de fin d'études a pour but de contribuer à l'élaboration du dataset VibraVox et à la conception d'un système de reconnaissance automatique de la parole (ASR) à partir de données non conventionnelles. Une méthodologie basée sur l'utilisation de réseaux de neurones profonds et de couches Transformers a été mise en œuvre. Le réglage fin des modèles pré-entrainés est minutieusement ajusté en utilisant des stratégies d'implémentation et les hyperparamètres décrits par Baevski et al. [1] sont adoptés. Les expérimentations visent à analyser les différences de performance entre les différents microphones et à quantifier l'impact d'un système de réhaussement de la parole en pré-traitement. 200 participants au total (50 heures) devront suivre un protocole expérimental pour enregistrer des données audio à l'aide de différents microphones à conduction corporelle. Un article scientifique résume ces résultats et conclusions.

Mots-clés : Microphone à conduction corporelle ; Reconnaissance automatique de la parole ; Réseaux de neurones profonds ; Couches Transformers ; Sciences expérimentales.