UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Ivan Lastovko

Joint Embeddings for Voices and Their Textual Descriptions

Master's Thesis (30 ECTS)

Supervisor(s): Mark Fishel, PhD

Tartu 2023

Joint Embeddings for Voices and Their Textual Descriptions

Abstract:
Embeddings are vector representations which is a highly effective method employed in machine learning to represent data in a more meaningful and efficient manner. In this study, we aim to implement vector representation for speakers' voices and corresponding textual descriptions, maximizing their cosine similarity. In other words, we want to build a system capable of representing both voices and descriptions of those voices as closely as possible in the multidimensional space. In our work, the data collection process involves using public datasets as well as manually annotated data. In order to conduct our research, we have utilized different training modes, such as standalone, where encoders are trained individually, and joint training techniques, where encoders are trained together to learn to adapt their outputs accordingly. We then evaluated the models on our control sample extracted from the manually collected dataset and assessed the quality of our annotations. We have also investigated the changes in cosine similarity between the speakers' and voice descriptions' vector representation with the decline in annotation quality.

Keywords: text-to-speech, embeddings, cosine similarity, Wav2Vec2, sentence encoders

CERCS: P176 Artificial intelligence

Häälte ja nende kirjelduste ühised vektoresitused

Lühikokkuvõte: Vektoresitus on väga tõhus meetod, mida kasutatakse masinõppes, et esitada andmeid sisukamal ja tõhusamal viisil. Selles uuringus on meie eesmärk rakendada kõnelejate häälte ja vastavate tekstiliste kirjelduste vektoresitus, maksimeerides nende koosinussarnasust. Teisisõnu, me tahame ehitada süsteemi, mis suudab esitada nii hääli kui ka nende häälte kirjeldusi võimalikult täpselt mitmemõõtmelises ruumis. Meie töös hõlmab andmete kogumise protsess nii avalike andmekogumite kui ka käsitsi annoteeritud andmete kasutamist. Uurimistöö läbiviimiseks oleme kasutanud erinevaid treeningrežiime, näiteks eraldiseisvaid, kus kodeerijaid koolitatakse individuaalselt, ja ühiseid treeningtehnikaid, kus kodeerijaid koolitatakse koos, et õppida oma väljundeid vastavalt kohandama. Seejärel hindasime käsitsi kogutud andmekogumist eraldatud kontrollproovi mudeleid ja hindasime meie annotatsioonide kvaliteeti. Samuti oleme uurinud muutusi koosinuse sarnasuses kõlarite ja häälkirjelduste vektoresituse vahel kui annotatsiooni kvaliteet langeb.

Võtmesõnad: kõnesüntees, vektorid, koosinussarnasus, Wav2Vec2, lause kodeerija

CERCS: P176 Tehisintellekt

Contents

1 Introduction

Natural Language Processing (NLP) is a field of computer science and artificial intelligence (AI) that focuses on the interaction between computers and human language. Its main goal is to enable machines to understand, interpret, and generate natural language in order to perform tasks such as text classification, sentiment analysis, language translation, question answering, and more.

In recent years, NLP has made significant progress thanks to the advancement of deep learning techniques, particularly with the development of transformers such as GPT-3, BERT, and T5. These models are trained on massive amounts of text data and can perform tasks with high accuracy and even generate coherent human-like language.

One of the key techniques used in NLP is the use of vector representation or embeddings, which is a way of representing words, phrases, and documents as numerical vectors. In this approach, each word or document is represented as a point in a high-dimensional space, where words that are semantically or syntactically similar are located closer to each other.

Embeddings and vector representations are powerful tools for data representation and analysis that can be applied to a wide range of data types. In essence, embeddings are a way of representing data points in a high-dimensional vector space, where each dimension corresponds to a feature or attribute of the data. The goal of embedding is to capture the essence of the data in a way that makes it easy to perform various computational tasks such as similarity comparison, clustering, and classification.

In the context of natural language processing, word embeddings are commonly used to represent words as dense, fixed-length vectors. These embeddings are trained on large text corpora using unsupervised learning algorithms such as Word2Vec or GloVe. The resulting vectors capture semantic and syntactic information about the words, allowing for more efficient and accurate processing of natural language data.

However, vector representations are not limited to NLP. They can be applied to any type of data, such as images, audio, or numerical data. For example, in computer vision, convolutional neural networks (CNNs) use learned filters to generate embeddings that capture relevant features of images. Similarly, in audio processing, recurrent neural networks (RNNs) can be used to generate embeddings that capture patterns in sound waves. The benefits of embeddings are numerous. They can reduce the dimensionality of data, making it easier to process and analyze. They can also capture important relationships between data points, allowing for more accurate predictions and classifications.

The objective of this project is to develop encoders that can represent voice recordings and textual descriptions of these voices in vector space. This encoding system must be capable of representing voices and corresponding textual descriptions very closely in a multidimensional vector space. If voice recording and description do not correspond, their embeddings should be located far away from each other. The process of building this project involves obtaining voice recordings of various individuals along with their

corresponding textual voice descriptions and then combining them into the same vector space. We aim to minimize the angle (increase cosine similarity) between the voice-description pairs that match while increasing the angle for those that do not.

The system we are building may have various practical applications. The most straightforward applications include:

1. **Text-to-speech.** The vector representations can be utilized to develop multi-speaker text-to-speech systems. Through this system, a user can specify the desired voice to the machine, which will then read the prompt in that particular voice. This functionality enables the generation of voices that were not previously included in the training data of the text-to-speech system, simply by describing them to the system.

2. **Voice description generation.** It is also possible to utilize this process in the opposite direction by generating a voice description from an audio recording. To achieve this, further training of the text decoder, such as GPT-3 [6], would be necessary. The decoder could take the vector of the voice as an input and produce an accurate voice description.

3. **Voice search.** Similarly to image search, it is also possible to implement a voice search with such technology. Image search works by representing images as high-dimensional vectors in a vector space. Each dimension of this space corresponds to a specific visual feature of the image, such as color, texture, or shape. The system searches for images whose vectors are closest to the vector representation of the search query. This is done using distance metrics such as Euclidean distance or cosine similarity. The images that are closest to the search query vector are returned as search results. Instead of images, it is possible to represent voices in vector space and then extract the vector that is closest to the search query.

During this study, we determined and tried to answer the following research questions:

RQ1: How well can this system perform? How accurate would be vector representations?

RQ2: Can prior fine-tuning of the encoders in standalone mode increase the overall accuracy of the predictions?

RQ3: How can freezing either encoder during joint training affect the accuracy of vector representations?

RQ4: How much of an impact can the quality of textual annotations of the voice characteristics have on the prediction accuracy?

Section 2 of this study is dedicated to providing a comprehensive explanation of the background, along with a description of the related work. We elaborate on various

technologies such as text and audio encoders, CLIP, and similarity measures. In Section 3, we explain the process of data collection, which involves providing details about the public datasets we used, as well as the manual data annotation workflow. Moving on to Chapter 4, we discuss the methodology of this study, in which we offer a detailed overview of the encoders' implementations. Section 5 describes the training process, which includes both standalone and joint training of the encoders. Section 6 provides an overview of the results achieved in this study, where we compare the results on the control sample and also examine how the accuracy of vector representation is impacted by annotation quality. In Section 7 we conclude the work we have done and answer the research questions.

2 Background & Related Work

2.1 Text Encoders

At the core of text encoders lies the concept of vectorization. This process involves transforming text into mathematical objects known as vectors, which represent words, phrases, or entire sentences in a high-dimensional space. These vectors capture semantic and syntactic information, enabling the algorithm to identify relationships, similarities, and patterns within the text. Once encoded, this numerical data serves as the input for various NLP tasks, such as sentiment analysis, machine translation, and summarization.

Over the years, various encoding techniques have emerged, each with its unique approach to mapping textual data into numerical representations. The earliest methods, such as the Bag-of-Words (BoW) [14] and Term Frequency-Inverse Document Frequency (TF-IDF) [1], focused on the frequency and distribution of words within a text corpus. Although these techniques were effective for simple tasks, they failed to capture the semantic relationships and contextual information crucial for more advanced NLP applications.

This shortcoming led to the development of more sophisticated encoding techniques, such as Word2Vec [17], GloVe [13], and FastText [4], which harnessed the power of neural networks to generate vector representations capable of capturing semantic relationships between words. By analyzing the context in which words appear, these models could generate word embeddings that effectively preserved the meaning and structure of the text.

However, the true revolution in text encoding came with the advent of transformer-based models like BERT [7] and RoBERTa [11]. These models employ self-attention mechanisms and multi-layer architectures to generate context-aware embeddings that can adapt to various NLP tasks. Capable of understanding complex language structures, these encoders have significantly advanced the state of the art in NLP, enabling machines to perform tasks that were once considered impossible, such as generating human-like text, answering questions with high accuracy, and summarizing long documents.

Transformers. Transformers are a type of neural network architecture that was first introduced by Vaswani et al. in 2017 in the paper *"Attention Is All You Need"* [19]. They have since become one of the most popular and widely used models in the field of natural language processing and have had a meaningful impact on the field.

At their core, transformers are designed to process sequences of data, such as text, speech, or even images, and to capture the relationships between the different elements in the sequence. The architecture of a transformer is composed of an encoder and a decoder. The encoder is responsible for processing the input sequence, while the decoder generates the output sequence. Both the encoder and the decoder are composed of multiple layers of attention and feedforward neural networks (see Figure 1).

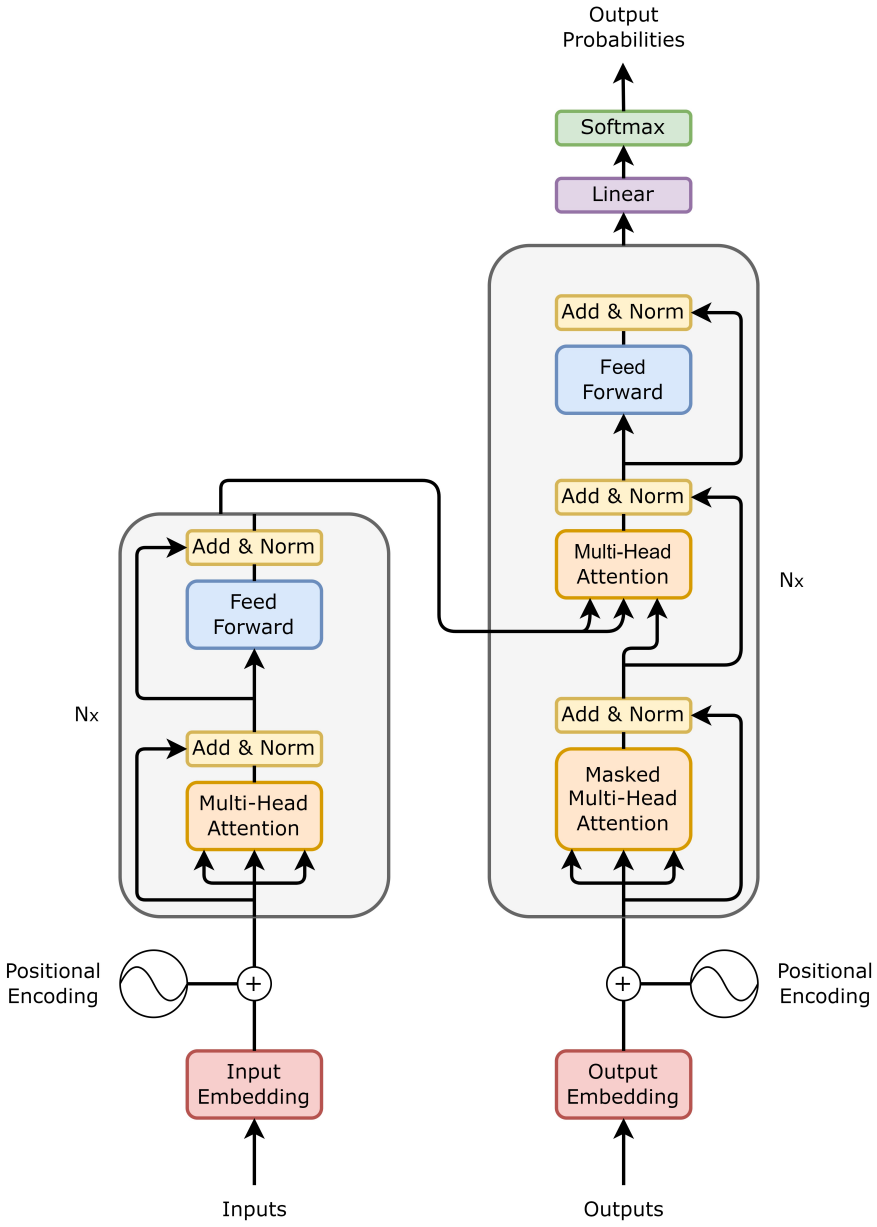

Figure 1. Transformers Architecture described in Vaswani et al. 2017 [19]. It consists of two parts: Encoder and Decoder, both of which include newly proposed Multi-Head Attention layers.

One of the key innovations of the transformer architecture is the attention mechanism, which allows the model to selectively focus on different parts of the input sequence as it

processes it. This attention mechanism makes transformers more powerful and flexible for sequence processing than previous neural network architectures, such as recurrent neural networks and convolutional neural networks.

The attention mechanism works by computing a set of attention weights, which indicate how much each element in the input sequence should contribute to the output at each step of the model. The attention weights are learned during training. By using attention, transformers are able to model long-range dependencies more effectively than RNNs, which can suffer from vanishing gradients when processing long sequences [12].

Sentence Transformers. Sentence Transformers, also known as SBERT, are a type of neural network model that is designed to encode natural language sentences into fixed-length vectors. Unlike traditional word embeddings that represent words in fixed-size vectors, sentence transformers are able to capture the contextual meaning of a sentence by considering not only the individual words but also the relationships between them.

The input sentence is first tokenized and passed through a pre-trained transformer model, such as BERT, RoBERTa, or DistilBERT. The transformer model uses self-attention to encode the input sentence into a sequence of word embeddings. Next, SBERTs use a pooling layer to aggregate the sequence of word embeddings into a single vector representation. The pooling layer can be of different types, such as mean pooling or max pooling. Mean pooling computes the mean of all the word embeddings in the sentence, while max pooling computes the maximum value of each dimension of the embeddings. Finally, the encoded sentence vector is passed through a few fully connected layers to further compress the information and output the final sentence embedding [16].

2.2 Audio Encoders

Audio encoders play a pivotal role in the field of machine learning, as they enable the transformation of audio data into a more manageable and structured format for processing and analysis. Machine learning algorithms can then utilize this transformed data to perform tasks such as speech recognition, audio classification, and music generation, among others. As the audio processing technology continues to advance, various audio encoding techniques have emerged to address the unique challenges of handling audio data.

Speaker Encoders. A speaker encoder is a type of deep neural network that can learn to represent a person's voice or speech patterns in a high-dimensional vector space. The resulting vector representation can be used for various speaker-related tasks, such as speaker verification, speaker diarization, or speaker recognition. Speaker encoders are typically trained on large datasets of speech recordings from many different speakers. During training, the encoder learns to extract high-level features from the speech signals,

such as pitch, timbre, and intonation, that are characteristic of each speaker's voice. The encoder then maps these features onto a vector space, where each speaker is represented by a unique vector.

In this particular section, we will be examining three commonly used approaches for constructing speaker encoders. The techniques that we will be reviewing are the following: recurrent neural networks, *Pyannote* Python library, and Wav2Vec2.

RNN. Recurrent Neural Networks (RNNs) are a type of neural network architecture that can process sequential data such as time series, speech, and natural language. They are designed to handle inputs of varying lengths and extract relevant features from the input sequence by maintaining a hidden state that can store past information.

Mel spectrograms can be used as input to an RNN for speaker encoding. A mel spectrogram is a type of spectrogram that is commonly used in speech and audio processing. It is a visualization of how the frequency content of an audio signal changes over time.

A regular spectrogram is a 2D representation of a signal that shows the frequency content of the signal over time. It is obtained by applying a Fourier transform to the signal and plotting the magnitudes of the resulting frequency components as a function of time.

A mel spectrogram is similar to a regular spectrogram, but it uses a different frequency scale that is based on the human auditory system. The human ear is more sensitive to certain frequencies than others, and the mel scale takes this into account. In the mel scale, lower frequencies are spaced farther apart, while higher frequencies are spaced closer together (see Figure 2).

The RNN can be trained to take in a sequence of mel spectrograms and output a fixed-size embedding vector that represents the speaker's identity. One way to do this is to use a sequence-to-sequence RNN architecture, such as a Long Short-Term Memory (LSTM) network or a Gated Recurrent Unit (GRU) network. The mel spectrograms can be fed into the input layer of the RNN one at a time, and the hidden state of the RNN can be used as the output embedding vector.

Pyannote. Pyannote is an open-source toolkit for analyzing and processing audio signals, specifically designed for speech and music processing applications. It provides a wide range of functionalities for speaker diarization, speech activity detection, speaker embedding, speech recognition, and other related tasks. Pyannote also comes with a set of trainable end-to-end neural networks that can be combined and jointly fine-tuned in order to build audio processing pipelines [5].

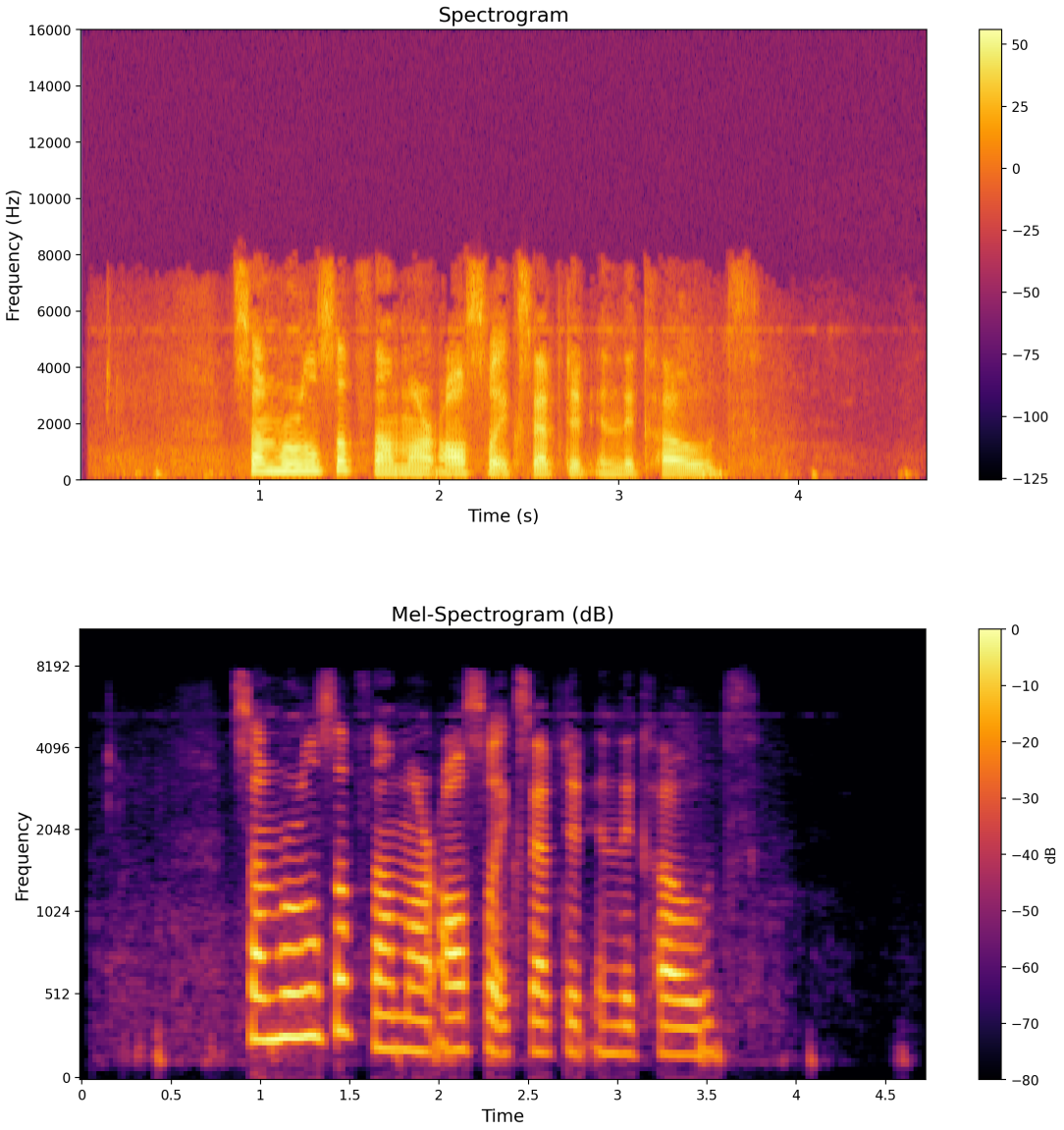

Figure 2. Difference between Spectrogram and Mel Spectrogram created from the same audio sample.

Wav2Vec2. Wav2Vec2 is a state-of-the-art speech recognition model developed by Facebook AI Research (FAIR) that uses self-supervised learning to improve the accuracy of automatic speech recognition (ASR) systems. The model was introduced in a paper titled "Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech

Representations" by Alexei Baevski et al. in 2020 [3].

Traditional ASR systems require large amounts of labeled training data, which can be expensive and time-consuming to collect. In contrast, self-supervised learning aims to learn useful representations of data by leveraging the inherent structure in the data itself, without the need for explicit labels.

The architecture of Wav2Vec2 can be divided into two main components: the feature encoder and the context network (see Figure 3):

1. **Feature Encoder.** The feature encoder is a convolutional neural network responsible for processing the raw audio waveform and extracting high-level features from it. It takes as input the raw audio waveform and applies a series of 1D convolutional layers. This results in a hierarchical representation of the input waveform, capturing both low-level acoustic information and high-level temporal structure. The output of the feature encoder is a sequence of hidden representations, which serve as the input for the context network.

2. **Context Network.** The context network is responsible for modeling the temporal dependencies in the hidden speech representations obtained from the feature encoder. It helps capture the long-range dependencies and relationships between different parts of the input audio, which is crucial for understanding and transcribing speech effectively. The context network is based on the transformer architecture. It consists of a stack of self-attention layers that enable the model to focus on different parts of the input sequence depending on the context.

2.3 Joint Encoders

Joint encoders are a type of neural network architecture used for encoding and processing input data. Unlike traditional encoders, which process input data independently, joint encoders are designed to process multiple types of input data in a coordinated and integrated manner.

The basic idea behind joint encoders is to combine the encoding processes for different types of input data into a single model. This allows the model to capture the relationships and interactions between the different types of data, which can be useful for a variety of applications.

For example, in natural language processing, joint encoders can be used to process both text and audio data, allowing the model to better understand the nuances of spoken language. Similarly, in computer vision, joint encoders can be used to process both image and text data, allowing the model to better understand the context of the images. The key benefit of joint encoders is that they can lead to better performance on tasks that require processing multiple types of input data.

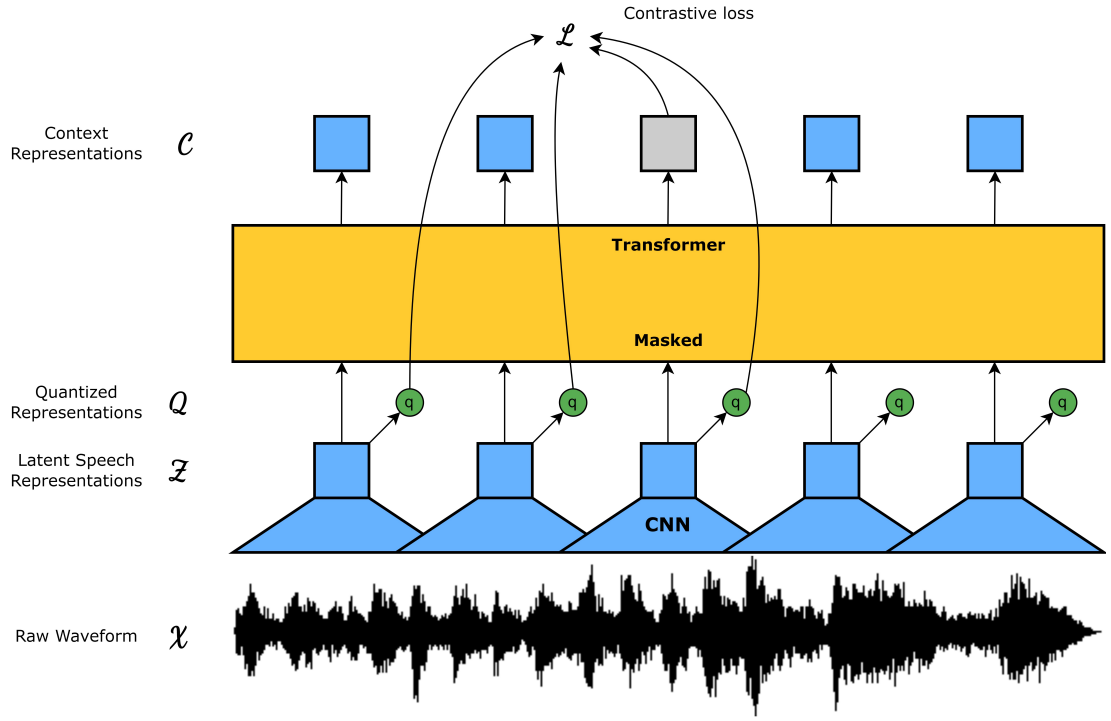

Figure 3. Architecture of the Wav2Vec2. It takes raw waveform and converts it into an array of numbers. Then 1D convolution is applied. Outputs from the CNN are passed into the context network, which is based on the transformer's architecture.

CLIP (Contrastive Language-Image Pre-Training) [15] and AudioCLIP [8] are two examples of joint encoder models that have gained significant attention in recent years. One of the key advantages of joint encoder models like CLIP and AudioCLIP is that they can be used for a wide range of applications without the need for extensive retraining. This is because the models are trained on large amounts of data in an unsupervised manner, allowing them to learn general representations that can be applied to a variety of tasks.

CLIP. CLIP is a joint encoder model developed by OpenAI that has gained significant attention in the fields of computer vision and natural language processing. It is designed to process both images and natural language text and has shown impressive results in a variety of tasks, including image classification and image captioning. During training, CLIP is exposed to a large corpus of images and their corresponding textual descriptions. The model learns to encode the images and text in a shared latent space, where similar images and text are placed close together and dissimilar images and text are placed far apart (see Figure 4). This allows the model to associate images and text that are

semantically related, even if they are not directly matched during training [15].

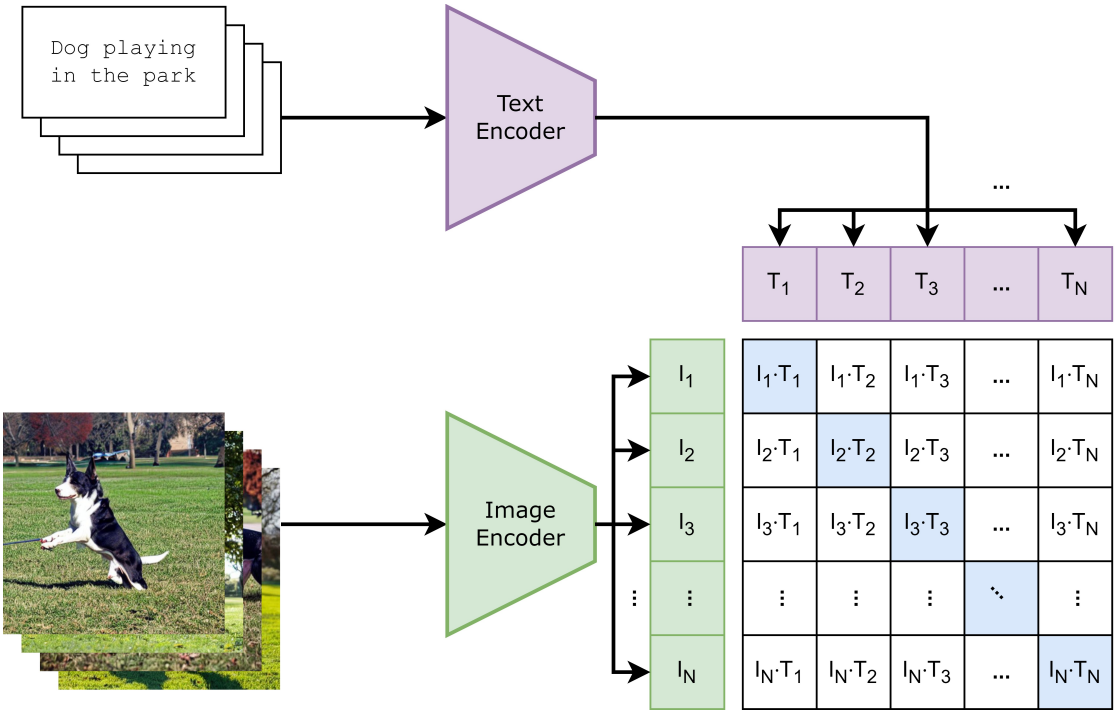

Figure 4. Illustrated approach to CLIP pre-training. Batches of images and text are passed into corresponding encoders. For each of the items in the batch, encoders output a generated vector. CLIP trains an image encoder and a text encoder jointly in order to predict the correct pairings of a batch of training examples.[1]

Our work was greatly inspired by CLIP technology, and despite utilizing different modalities, it adopted a similar approach.

AudioCLIP. AudioCLIP is a combination of a contrastive text-image model (CLIP) with a high-performance audio model - ESResNeXt [9]. The AudioCLIP model comprises of three subnetworks, namely text, image, and audio heads. Apart from the current loss term of text-to-image similarity, two new loss terms of text-to-audio and image-to-audio have been introduced. The proposed model can handle all three modalities simultaneously, as well as any combination of two modalities [8].

[1]The dog images were generated from Stable Diffusion (18.04.2023), i.e. a text-to-image diffusion model. Stable Diffusion is developed by Stability AI. For more information about Stable Diffusion and Stability AI: `https://stability.ai`

Prompt: "Dog playing fetch in the park"

2.4 Similarity Measures

Cosine Similarity. Cosine similarity is a way to measure the similarity between two embeddings. The cosine similarity ranges between -1 and 1, with 1 indicating that the two vectors are identical, 0 indicating that the vectors are orthogonal, and -1 indicating that the two vectors are entirely opposite. The cosine similarity between two vectors x and y is defined as the cosine of the angle between them, and can be calculated using the dot product of the vectors:

$$cosine\_similarity(x, y) = \frac{x \cdot y}{\|x\| \, \|y\|} \tag{1}$$

where $||x||$ and $||y||$ are the Euclidean norms of the vectors x and y, respectively.

MAE Loss. The mean absolute error (MAE) loss is a commonly used measure of the difference between two sets of values. It calculates the average difference between the predicted and actual values. MAE is calculated by taking the absolute difference between the predicted and actual values and then averaging those differences:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2}$$

Where:

n is the number of data points;

y_i is the actual value for the i-th data point;

$\hat{y}_i$ is the predicted value for the i-th data point;

MAE loss is commonly used in machine learning, particularly in regression problems. It is a popular alternative to other loss functions like mean squared error (MSE) because it is less sensitive to outliers. That means that MAE is a better choice when we have data with extreme values or when we want to avoid penalizing large errors too heavily.

3 Data Collection

There is no dataset in an open-source community that could suit our goals. Because of that, we decided to collect our own dataset with the help of some participants we employed for this task.

The aim of this section is to provide a detailed overview of the process of data collection in this study. Firstly, we elaborate on two publicly available datasets which were downloaded and utilized for this study. Next, we write about the process of data annotation. This process involves the recruitment of additional participants for the manual annotation of voice descriptions. We then proceed to set up Label Studio in a virtual machine and upload the audio samples. Lastly, we describe the procedure for extracting control samples from the primary dataset that was collected and assess their quality.

3.1 Public Audio Datasets

The initial step in the data collection process involved determining the required format for the datasets necessary for the project. We decided that the dataset should contain a significant number of audio samples that feature diverse speakers and corresponding text prompts. The choice was given to Common Voice and VCTK datasets.

Common Voice. Common Voice is an open-source initiative launched by Mozilla. It is a multilingual dataset of human voice recordings and transcriptions. The dataset consists of three main components: the voice recordings, their transcriptions, and metadata about the recordings. The voice recordings are provided in WAV format and are approximately five seconds in length. The transcriptions are provided in plain text format and correspond to the spoken words in the recordings. The metadata includes information about the speaker's gender, age, and accent. Common Voice is often used for training machine learning models for building audio applications. As of the day of writing this thesis, the Common Voice dataset features 3216 hours of English-language audio recordings with more than 87500 distinct speakers. This vast collection of speech data and the diversity of speakers were the main factors behind the selection of this dataset. The recordings come from a wide range of speakers, including people of different ages, genders, and accents. This diversity helps to ensure that speech recognition systems trained on the dataset are robust and can handle a variety of accents and speaking styles. Another unique aspect of the Common Voice dataset is that it is community-driven. Anyone can contribute to the dataset by recording and submitting their own voice recordings. This crowdsourced approach to data collection helps to ensure that the dataset remains up-to-date and reflects the diversity of the global population. Nonetheless, the drawback is that the audio quality is frequently distorted, and sometimes, background noise may be present [2].

VCTK. The VCTK is another publicly available dataset consisting of speech recordings from 109 English speakers with diverse accents, including British, Irish, Scottish, and American. Initially, it consisted of 110 speakers, but the speaker 'p315' text was lost due to a hard disk error. The VCTK dataset is also notable for its quality. The recordings are of high fidelity and have low noise and distortion levels, making them ideal for research on speech recognition and machine learning. The speakers were recorded in a soundproof booth using high-quality microphones and digital audio recorders. Each speaker reads about 400 sentences selected from newspapers and other publications. Although the speakers' variation is much lower than the Common Voice English-language segment, the audio samples have a much higher quality and were recorded using an identical setup [20].

3.2 Voice Annotation

In addition to audio samples, our project needed to obtain textual descriptions of the speakers' voices. However, the datasets that were selected for this project did not offer adequate information on the speakers and their vocal characteristics. Therefore we had to conduct data collection on our own. While it is true that the Common Voice dataset provides some information on the speakers, such as their gender, approximate age, and nationality, this data is quite limited and does not meet the specific requirements of our project, as we require a comprehensive textual description rather than categorical information.

In order to manually gather as much data for the project as possible, we decided to involve other students. After finding 53 participants, we created a survey in which we asked each individual how much workload they were willing to take, and we split the audio datasets according to their answers. In total, we selected 55000 audio samples from both VCTK and Common Voice datasets with the ratio 1:4, respectively. The selection of a greater number of samples from the Common Voice dataset was based on the fact that it provides a broader range of vocal diversity. However, we also did not want to compromise on audio quality, so 20% of the selected samples were from VCTK.

Label Studio. The next step was to set up a Label Studio project in the virtual machine. Label Studio serves as an annotation tool that can be utilized in various domains, ranging from image classification to object detection or text summarization. One of its main advantages is the ability to construct a user-friendly interface through simple XML tags, as well as an API which made the creation of individual projects for each of the 53 participants much quicker. Through the utilization of this API, we have successfully

automated the process of project creation. To achieve this, we developed a Python script which performed the following tasks:

1. The script allocated specific audio samples to participants based on their workload preferences as indicated in the previously mentioned survey.

2. It sent HTTP requests to the Label Studio API located on our virtual machine in order to initiate project creation and upload the audio samples.

Every participant could connect to the Label Studio environment remotely by following the link in the browser and then start the annotation process. The participant's task was to listen to the audio sample and write a suitable description of the speaker's voice (see Figure 5).

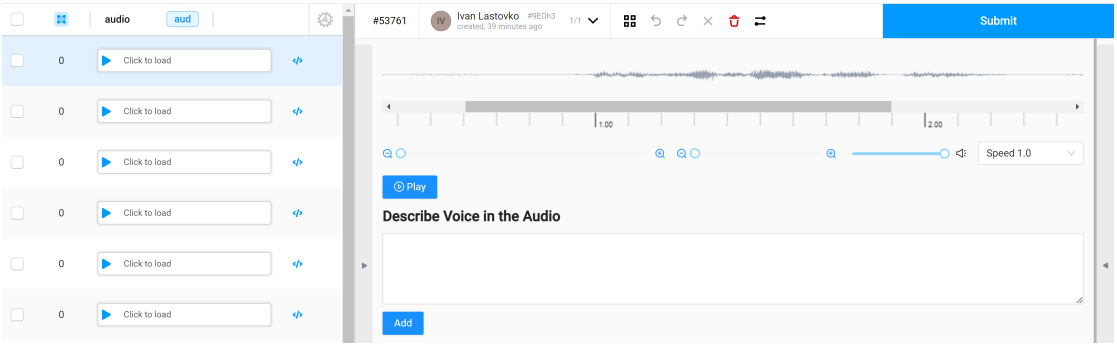

Figure 5. Label Studio user-interface screenshot. Users can play the audio recording on the top. Next, there is a text field for voice annotation. On the left, there is a list of audio samples enabling seamless transitions between them for the users.

The participants were given detailed instructions on what they needed to annotate and a few examples. We requested that they give additional consideration to the speaker's gender, accent, approximate age, voice specification and perception, speech style, and background noises (see Figure 6). The annotations were expected to be written in one sentence, although some participants faced difficulties with this and had to divide their voice descriptions into multiple sentences.

Once the annotation task was finished, it was necessary for us to obtain the voice descriptions written by the participants. To achieve this, we created an additional Python script that utilized the previously mentioned Label Studio API to send HTTP requests. Once the download was complete, the script collated these annotations into a single CSV file. In the end, a portion of the participants were unable to fulfill the assigned task completely. As a result, out of the initial selection of 55,000 audio samples, we were left with only 40,444.

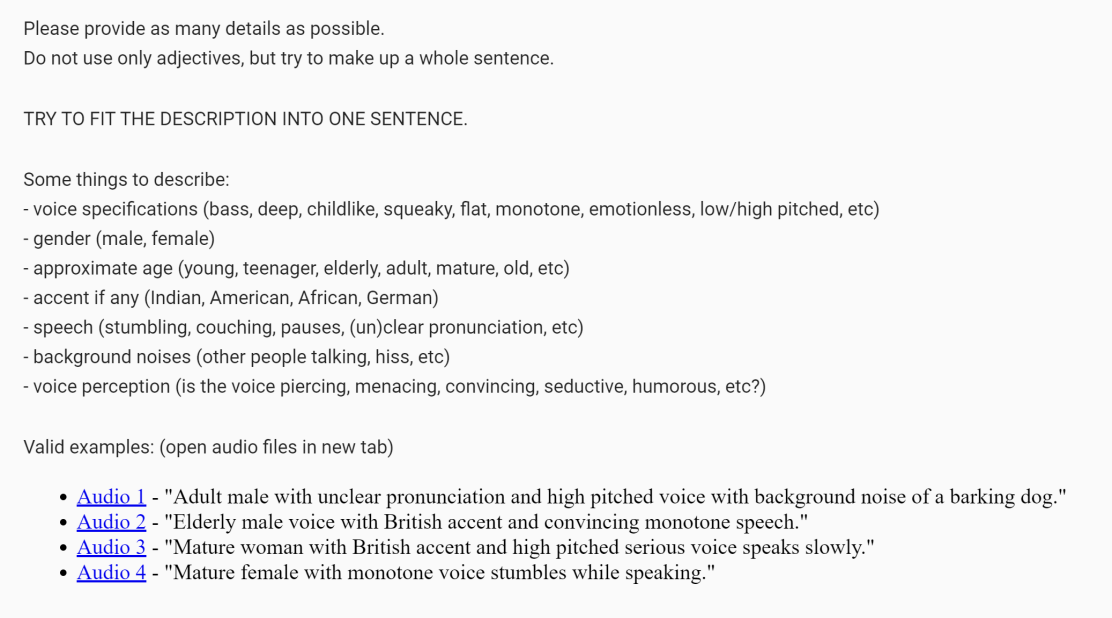

Figure 6. The screenshot of the task instruction provided in Label Studio for the annotation participants.

3.3 Control Samples

It was crucial to assess the quality of the annotations produced by the participants. To accomplish this, we randomly extracted 900 samples out of 40444 and requested another participant to evaluate the quality of the annotations. This participant was required to listen to each of the 900 voice recordings, review the annotations that were previously assigned to them, and then assess the relevancy and accuracy of the annotations. The assessment was conducted using a prepared in advance Excel spreadsheet. The first and second columns contained the audio sample filenames and annotations respectively, written by any of the participants. The third column provided a drop-down menu with five options that the person could choose from to rate the annotations. The result of the assessment can be found in Table 1.

As shown in the table, the majority of the annotations, 707 in total, received the highest grade, which means they were great, correct, and exhaustive. Other 194 annotations were either partial, minimal, or wrong.

Table 1. Annotation quality assessment.

Grade	Count
Great, correct, and exhaustive	707
partial	189
wrong or misleading	2
minimal or none	1
other - specify in a comment	1

3.4 Similarity Datasets

Once the control samples were extracted, it was necessary to generate similarity datasets for the audio and text encoders' training. These datasets included three columns: the first sample, the second sample, and the cosine similarity score between them, which could be either 0 or 1. We could not use the annotated dataset as it is, because it only included matching pairs of voice recordings and descriptions. In order to properly train the encoders, we needed to also add negative examples to the dataset and assign the similarity values.

Audio similarity dataset. In order to generate the audio similarity dataset, we did not require any annotations created by our participants since we used only VCTK and Common Voice datasets. Our process involved extracting approximately 40,000 audio samples from the VCTK dataset and an equal number of samples from Common Voice. We then created a Python script to aggregate the samples in pairs randomly. Specifically, we ensured that 50% of audio samples contained voices of the same speaker, while the remaining 50% had voices of different speakers. For each pair, we assigned a cosine similarity score of 1 if the audio samples belonged to the same speaker and 0 if they belonged to different speakers.

It is essential to mention that the similarities we have assigned were only partially accurate. There were instances where two audio samples appeared to contain nearly identical voices but were actually created by distinct individuals. As a consequence, these samples were given a similarity score of 0 instead of a score closer to 1.

Text similarity dataset. For producing a text similarity dataset, this time, we utilized our manually gathered annotations. These 40,444 text annotations were paired in a similar manner, the same as in the audio dataset. Voice descriptions that belonged to the same voice had a cosine similarity of 1, and voice descriptions belonging to different speakers were assigned a similarity of 0.

Similarly to the audio dataset, this pairing technique had a slight drawback: many textual annotations may have been describing voices in a similar way using the exact

words but still were assigned a similarity of 0 due to them belonging to different speakers. As an example, there are annotations such as *"Flat young male voice with drawn out pronounciation and slower speech, formal setting."* and *"Adult male speaks calmly with good pronunciation"*, which were assigned similarity of 0. In reality, they describe voice very similarly and the cosine value between them should be closer to 1. We acknowledge the fact that this drawback could have affected accuracy during training. However, we believed that our encoders would be able to generalize enough so that this problem would not have a significant impact.

Joint similarity dataset. Our last similarity dataset was created for joint training when both audio and text encoders were training at the same time. The dataset consisted of audio samples in the first column and corresponding voice descriptions in the second column. Using a script written in Python, we aggregated these samples in pairs and assigned cosine similarities to them. As a result, 50% of the pairings had accurate text annotations that precisely described the corresponding voices and were assigned a cosine value of 1. In the remaining 50% of cases, voice descriptions and voice recordings belonged to different speakers and thus were marked with a similarity of 0. In some instances, even if a text annotation accurately described the voice, the script had to assign a similarity of 0 because the voice belonged to a different speaker.

4 Methodology

Within this section, we provide a detailed overview of the selected models' structures and their implementation. Additionally, we explain the motivation behind our decision to choose these specific encoders for our study. Additionally, we briefly discuss the utilization of language models in the writing of this thesis.

The purpose of this project is to develop a solution that can represent speaker embeddings in the vector space based on their textual description. To achieve this, we developed two encoders for both text and audio. An encoder is a type of neural network that takes some input (audio or text in this case) and converts it into dense vector representations (also called embeddings). Embeddings represent data by mapping it into a high-dimensional vector space, where each dimension represents a feature or attribute of the data. These features are learned automatically from the data using machine learning. The encoder itself typically consists of several layers of neural networks that process the input and extract relevant features into vectors.

We trained the encoders to function in such a way that, if the textual description of a voice accurately corresponded with its actual voice, the cosine similarity between their vectors would be 1. Conversely, if the description was entirely opposite to reality, then the cosine similarity between their vectors would be closer to 0.

Hugging Face. Both encoders that are used in this work were downloaded from Hugging Face and then later fine-tuned on collected datasets[2]. Hugging Face is a popular platform, which hosts pre-trained models, datasets, and other resources for the NLP community. The Hugging Face provides a wide range of pre-trained models that can be used for tasks such as text classification, question answering, text generation, and more. These models are available in different languages and are trained on a variety of datasets. The platform also allows users to fine-tune these pre-trained models on their own datasets and customize them according to their specific needs. In addition to the pre-trained models, Hugging Face offers a variety of open-source tools and libraries for NLP tasks. For example, the company's Transformers library provides a simple interface for using and fine-tuning pre-trained models. At the same time, the Datasets library provides a way to easily access and work with a variety of datasets for NLP tasks.

4.1 Language model utilization

In the composition of this thesis, we used two well-known language models for text generation, namely ChatGPT and Grammarly. It should be emphasized that the contents of this work were not generated by directly posing questions to the language models and copying the responses into the thesis. Instead, they were used for rephrasing and

[2]Hugging Face homepage: `https://huggingface.co/`

correcting grammatical errors in the text. All of the sentences generated by language models were carefully reviewed before including them in this thesis.

ChatGPT. ChatGPT is a text-generating large language model developed by OpenAI, which is based on the GPT-3.5 (or GPT-4 with ChatGPT Plus subscription) architecture. This language model was utilized only for rephrasing original sentences. Firstly, it was given an initial prompt with the task to rephrase the text in a more formal format. Once the model generated a response, the text has been provided to it.

- Initial prompt: *"rephrase my text in a more formal and scientific format. Correct mistakes and add more words if necessary"*
- ChatGPT response: *"Please provide the text that you would like me to rephrase in a more formal and scientific format."*
- Second prompt: `[text to rephrase]`

Grammarly. Grammarly is an AI-powered writing assistant that helps users improve their writing skills by providing suggestions for grammar, spelling, punctuation, style, and tone. Grammarly uses machine learning algorithms to identify and correct errors and enhance the clarity and correctness of the writing. It can be used as a browser extension, desktop app, or mobile app and supports multiple platforms and applications such as Microsoft Word, Google Docs, and Overleaf.

In this work, we utilized Grammarly to identify grammatical errors, rearrange word order, enhance punctuation, and occasionally substitute words with synonyms in sentences. Additionally, it is worth mentioning that several sentences produced by ChatGPT were subsequently inserted into the Grammarly editor for the same purpose. Not all of the suggestions proposed by language models were accepted for this writing.

4.2 Text Encoder

The first step of encoder implementation was to decide on which architecture to use. The most suitable solution was to employ transformers [19]. As we already explained in Section 2, transformers are a type of neural network, that processes sequential data, such as text or speech, using attention mechanisms. Their ability to show state-of-the-art performance in various natural language processing tasks and the abundance of publicly available resources on how to fine-tune them was the primary motivation for choosing this architecture. Transformers consist of two parts: encoder and decoder (see Figure 1). The first part, the encoder, is what we needed for vector representation.

In this work, we utilize a sentence transformers library, also known as SBERT[3]. As it was stated in Section 2, sentence transformers are able to capture the contextual meaning

[3]For more information about SBERT: `https://www.sbert.net/`

of a whole sentence rather than the individual words. The process involves encoding individual tokens and then applying the pooling layer in order to aggregate the sequence of word embeddings into a single vector representation.

SBERT provides a long list of pre-trained sentence encoders, as well as their evaluation scores[4]. Also, additional information is provided about each of the models, such as output embedding dimensions, size of the model, training data length in tokens, pooling layer type (such as average pooling or mean pooling), and the base model on which it was extended.

In order to choose the suitable model for our project, we sorted the list of available models by evaluation scores. The highest ranked model was *"all-mpnet-base-v2"*, which is based on the model developed by Microsoft - *"mpnet-base"* [18][5]. This model had the following properties:

- Fine-tuned on 1B+ sentence pairs training dataset
- Max sequence length - 384
- Output embedding dimension - 768
- Model size 420 MB
- Mean pooling

4.3 Audio Encoder

In order to select the audio encoder for our study, we have tried several techniques. Each of them had their own advantages and downsides.

The first method we tried was implementing a recurrent neural network with several LSTM layers. RNN processes audio by taking their mel spectrograms as input. As we already explained, a mel spectrogram is a visual representation of the frequency content of an audio signal over time. The vertical axis shows the distribution of the signal's power across different frequency bands, and the horizontal axis represents time.

We converted all of the audio files from training and test datasets into mel spectrograms prior to the training. That way we managed to increase the performance by saving computational power during training. Unfortunately due to its relative simplicity, RNN was not able to learn distinguishable voice characteristics and did not achieve the acceptable accuracy that was required for the project.

[4] `https://www.sbert.net/docs/pretrained_models.html`

[5] *"all-mpnet-base-v2"* is publicly available at `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`

Next voice representations system was built based on *Pyannote* Python library [5]. Pyannote showed relatively good results out of the box without any fine-tuning, yet it had several limitations, such as returning NumPy arrays rather than PyTorch tensors and the absence of any publicly available information or examples of fine-tuning the model. After careful consideration, we decided to move on to our last and most promising option - Wav2Vec2.

As we discussed in Section 2, Wav2Vec2 is a state-of-the-art speech recognition model developed by Facebook AI Research. Although originally it was designed for speech recognition systems, we fine-tune wav2vec2 to perform speaker encoding tasks. Compared to usual RNNs, wav2vec2 does not require mel spectrograms to be passed as inputs. Instead, it takes in a raw audio file. But on the other hand, due to its complex structure, wav2vec2 requires big computational power and it took a long time to fine-tune the model.

In order to make Wav2Vec2 compatible with our text encoder, they both had to output embeddings of the same dimension. Vectors produced by the Wav2Vec2 had the dimensionality of 1024, while vectors produced by SBERT - 768. We have decided that a more reliable way for the models to learn correct representations was to downscale Wav2Vec2 embeddings by adding a new linear layer at the end of the encoder.

4.4 Joint Encoders

Implemented joint encoders training pipeline includes several steps (see Figure 7). To begin with, batches of both text and audio samples are fed into their corresponding encoders. These encoders generate embeddings, which are then used to calculate the cosine similarity between them. Next, the obtained similarity scores are compared to the actual similarities present in the dataset. The mean absolute error (L1Loss) is computed based on this comparison. The model's weights are then modified using backpropagation.

The idea behind joint encoders is inspired by the CLIP technology, which was developed by OpenAI [15]. CLIP uses a similar approach to joint encoders but with images instead of audio. CLIP learns a shared representation of images and text by training a neural network to predict which text captions are associated with which images, and vice versa. We followed a similar approach by presenting both corresponding and non-corresponding pairs of recordings and descriptions to the encoders.

Due to the vast number of possible approaches to training encoders jointly, we were unsure of the optimal method. However, we had some promising ideas that could potentially enhance training performance. We wanted to see whether encoders require prior standalone fine-tuning. Also, we had to understand whether freezing the gradients of either encoder would improve the accuracy of the predictions.

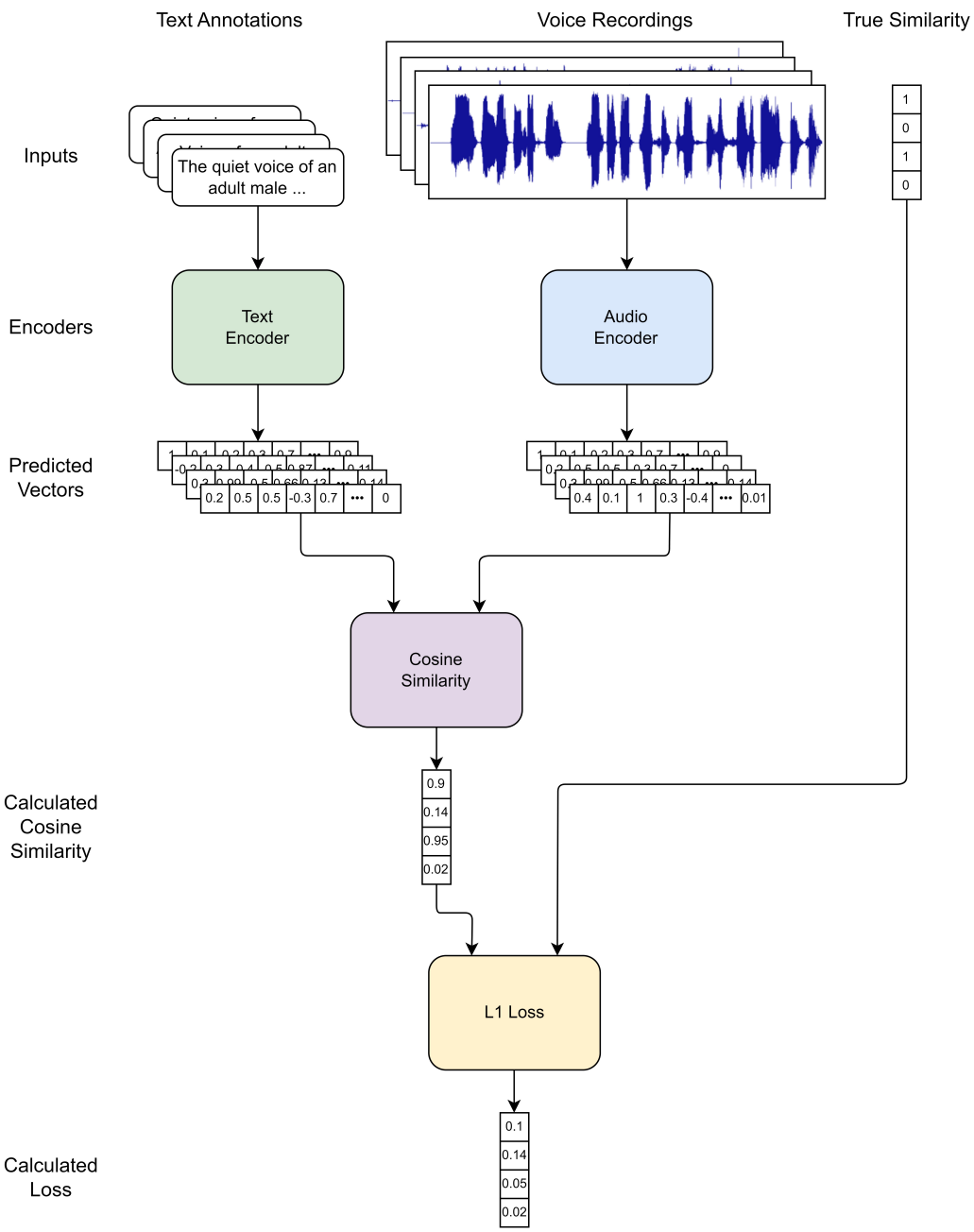

Figure 7. Joint encoders training pipeline. Encoders output predicted vectors for each item in the batch. Next, cosine similarity is calculated between the vectors of different encoders. Lastly, MAE Loss is calculated between true and produced cosine similarities.

5 Experiments

After making a decision on which encoders to use and implementing them, it was time to run the experiment. In this section, we are going to discuss in detail the training process of the encoders. Firstly, we start with describing the standalone training of the text and audio encoders, and then we move on to provide an overview of the joint training, where these encoders learn to adjust to each other in order to produce similar vector representations for corresponding voice-text pairs. All the joint training as well as standalone training of the encoders were conducted in the HPC center of the University of Tartu.

5.1 Standalone Training - Text Encoder

We implemented our sentence encoder using SBERT [16]. The training was done on the textual voice descriptions annotated by the participants that we employed. In total, 65468 sentence pairs were used for training and 1000 sentence pairs we utilized for evaluation during training.

After several experiments, we found the optimal hyper-parameters for training. They can be found in Table 2.

Table 2. SBERT hyper-parameters.

Hyper-parameter	**Value**
Learning rate	1e-6
Epochs	200
Batch size	128
Loss function	MAE loss
Optimizer	Adam

The training was conducted for a total of 200 epochs, during which the best score on the evaluation set was obtained on the 87th epoch. We decided to use MAE loss as our loss function. The mean absolute error was calculated between predicted and true cosine similarities of the voice descriptions. The best (lowest) MAE score that we achieved on the evaluation set was 0.266. In comparison, the zero-shot SBERT model achieved a loss score of more than 0.44 on the evaluation set. The model's checkpoint saved at this epoch was used later in the joint training of both encoders.

SBERT training and evaluation logs are represented in Figure 8. From there we can see how loss gradually decreases over time. In order to better understand patterns and identify simplified changes, a smoothing technique with a value of 0.7 was applied to the chart. Smoothing is a technique used to reduce noise and fluctuations in data by creating a smoother representation of a signal.

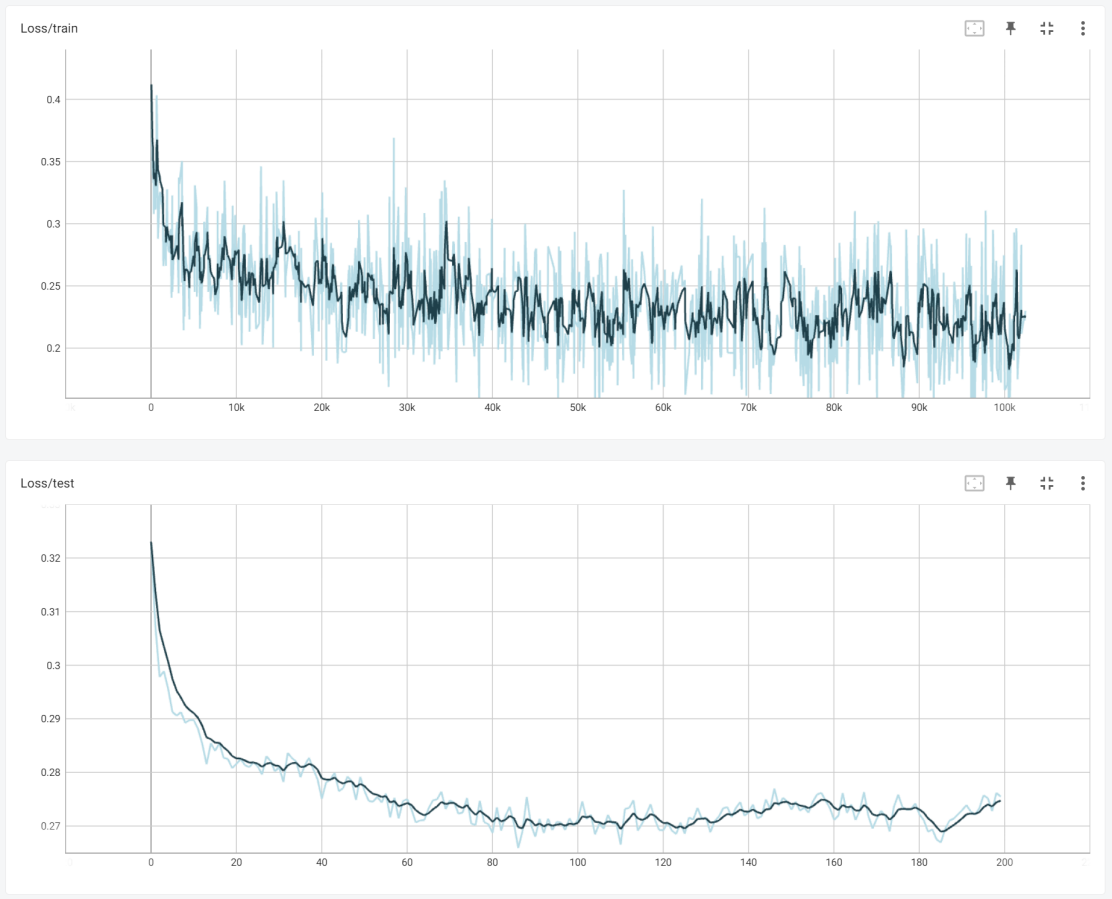

Figure 8. Training and evaluation log of SBERT made in Tensorboard. Smoothing applied: 0.7

Although the loss was lower compared to the zero-shot SBERT model, we wanted to have a better understanding of what the sentence encoder learned and how exactly data representation would change depending on the textual content.

In this work, our encoders represent data as 768-dimensional vectors. Such a high number of dimensions is hard to visualize, so we used a technique called dimensionality reduction. Dimensionality reduction is a process of reducing the number of features or variables in a dataset while retaining as much relevant information as possible. In other words, it involves transforming high-dimensional data into a lower-dimensional space while preserving the essential characteristics of the original data. The most common dimensionality reduction algorithm is PCA (Principal Component Analysis) [10]. PCA is commonly used in machine learning, statistics, and data analysis. It achieves dimensionality reduction by transforming the data into a new coordinate system, where

the principal components are aligned with the directions of maximum variance in the data. PCA can not only reduce the number of dimensions in the data, making it easier to visualize and analyze, but it also can remove noise from the data by filtering out the dimensions that contribute little to the overall variance.

In order to visualize vector representations learned by our sentence transform during training, we utilized our control samples dataset which contained 900 textual annotations and corresponding voice recordings. We encoded all 900 annotations and applied PCA (with 2 principal components) on these obtained vectors. The result can be seen in Figure 9.

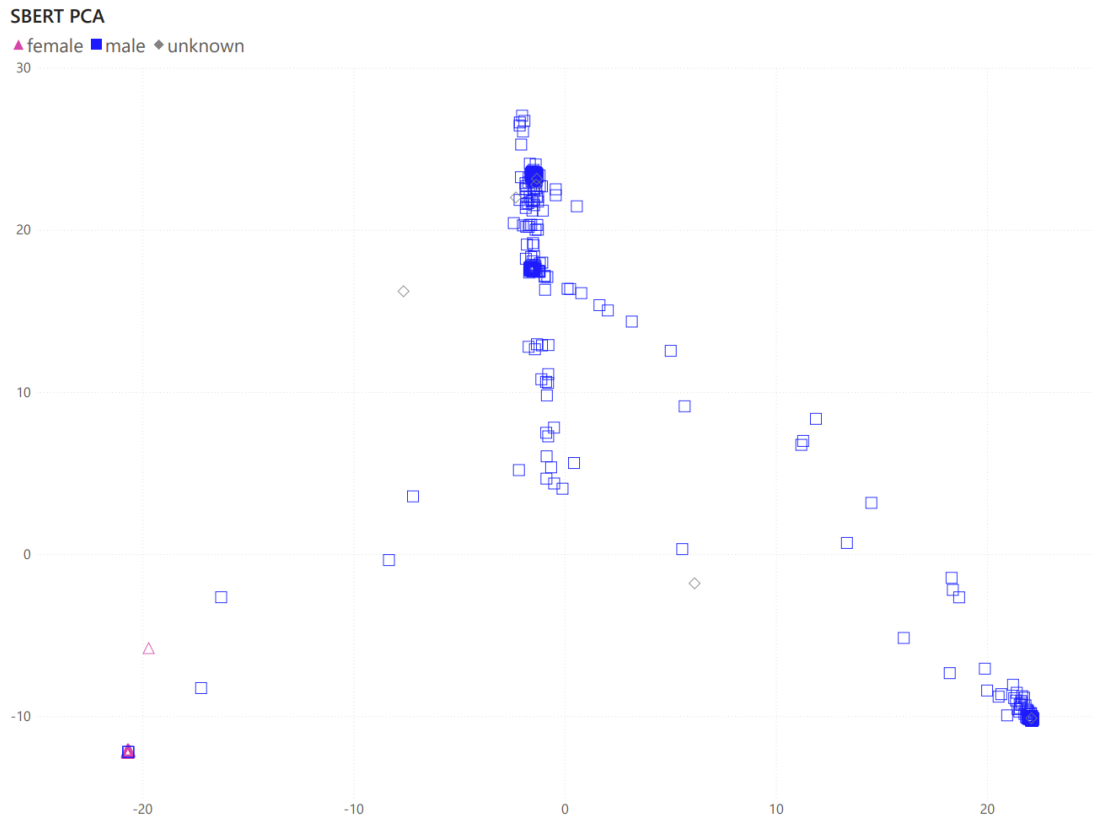

Figure 9. PCA visualization of SBERT embeddings after fine-tuning. Partitioned by gender. Blue squares - male voice descriptions; red triangles - female voices; gray rhombuses - unknown gender voice descriptions.

In total, there were 283 female voices, 611 male voices, and 6 unknown gender voices. We can see that the model learned to successfully identify female voice descriptions, and as a result, it always puts them in the bottom-left corner. At the same time, male voices are distributed over the chart forming two major groups on the top-center and bottom-

right sides of the chart. There is no obvious difference between these two groups since both of them contain male voices of various pitches, tones, accents, and pronunciations.

For comparison, we also included the same visualization of applied PCA on embeddings produced by zero-shot SBERT (see Figure 10).

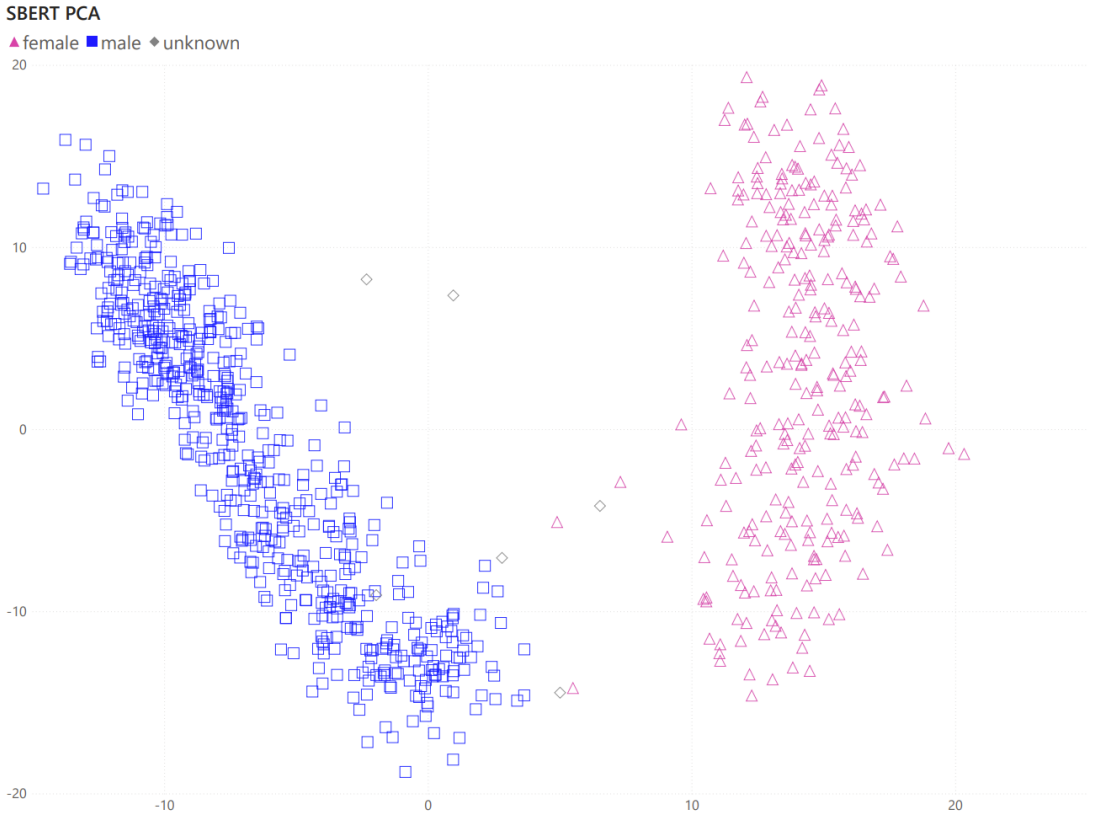

Figure 10. PCA visualization of SBERT embeddings without fine-tuning. Partitioned by gender. Blue squares - male voice descriptions; red triangles - female voices; gray rhombuses - unknown gender voice descriptions.

Here, we can see an explicit division between male and female voice descriptions. This is not a surprise since the model we chose for the experiment has been showing state-of-the-art results in other sentence encoding tasks. But regardless of how data points are visualized after dimensionality reduction, the zero-shot model still performed poorly on our dataset. Apart from gender, it was not able to separate other voice characteristics, such as age, tone, pronunciation, or accent.

5.2 Standalone Training - Audio Encoder

For the implementation of the audio encoder, we utilized Wav2Vec2 [3]. We performed training on the dataset which consisted of the combination of the Common Voice and VCTK with around 40,000 audio samples from each of them. In total, this resulted in 162,773 pairs of voice recordings with similarities of either 1 or 0. 1000 of these pairs were defined as evaluation set.

Hyper-parameters were chosen after conducting several experiments and they can be found in Table 3.

Table 3. Wav2Vec2 hyper-parameters.

Hyper-parameter	**Value**
Learning rate	1e-5
Epochs	200
Batch size	8
Loss function	MAE loss
Optimizer	Adam

We had to reduce the batch size to the value of 8 due to the constant "`CUDA out of memory`" issue. Both the dataset and the model were so large, that they would take up all of the GPU memory. Same as with SBERT, the number of epochs was set to 200 and we used MAE loss as our loss function. We also slightly increased the learning rate up to a 1e-6.

Since the number of audio samples in the training set was so large, we were conducting evaluations several times in each epoch. Training and evaluation logs of Wav2Vec2 are represented in Figure 11. Smoothing with a value of 0.7 was applied to the chart. The lowest MAE score was achieved in the 9th epoch (iteration 87 on Figure 11). After this epoch, we stopped the training since the model started to overfit and the evaluation score did not change that much. The lowest MAE score we achieved was 0.200. The model's checkpoint saved at this epoch was used later in the joint training of both encoders.

Similarly to SBERT, we wanted to visualize the embeddings produced by fine-tuned Wav2Vec2 model and see what vector representations has it learned. To do that, we encoded all 900 audio recordings from our control sample. We then applied the same dimensionality reduction in order to visualize them in a 2D plot. The results of the visualization of the embeddings can be seen in Figure 12.

The model has grouped embeddings into 4 main clusters. Even though male and female voices are not located far from each other, we can see a clear partition between audio samples from different datasets. A cluster that is located in the center of the triangle formed by the other three clusters consists fully of audio recordings taken from the VCTK dataset. All of the other points in Figure 12 represent embeddings of Common

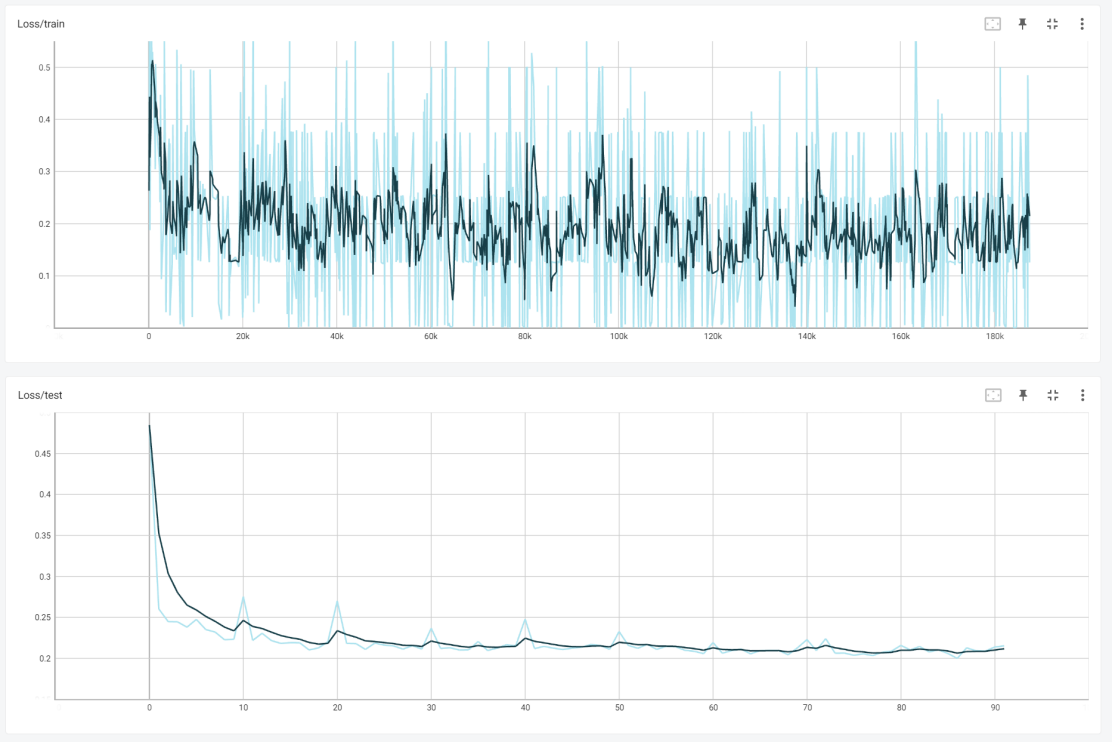

Figure 11. Training and evaluation log of Wav2Vec2 made in Tensorboard. Smoothing applied: 0.7

Voice audio samples. It seems that the model learned to distinguish voices by the quality of the recording rather than by the characteristics of the voice since all samples in the VCTK dataset were recorded using high-quality studio equipment.

5.3 Joint Training

After both encoders finished their training, it was time to start the experiments with joint encoders training. For this training, we utilized a dataset created from the annotations that were produced by our participants. The dataset consisted of 78,669 text-audio pairs with around 50% of them having similarity of 1, and the other half with a cosine similarity of 0. 1000 samples from this dataset were used during the evaluation.

We were going to try several different combinations of training since we did not know the best strategy to implement this. Some of the joint training implementations included freezing either of the model's parameters so that the other encoder would learn to adapt its embeddings to the outputs of this model. Training combinations were the following:

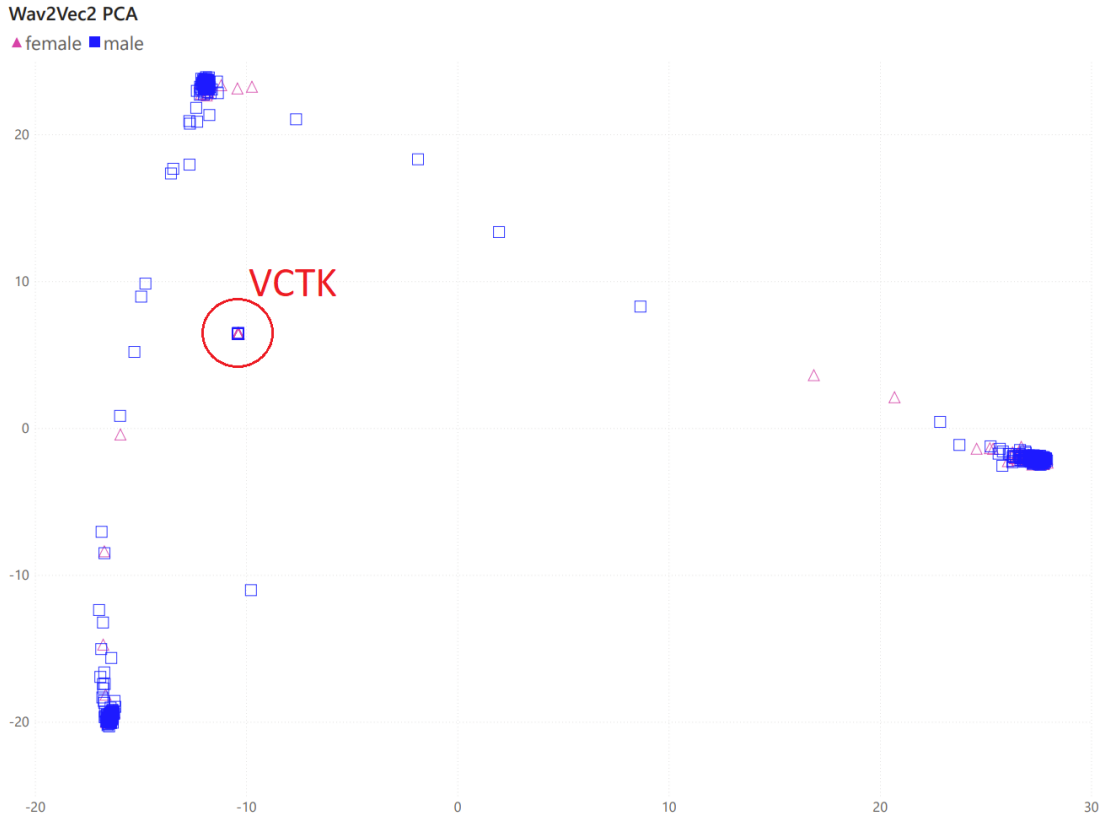

Figure 12. PCA visualization of Wav2Vec2 embeddings after fine-tuning. Partitioned by gender. Blue squares - male voice descriptions; red triangles - female voice descriptions. The red circle indicates that all audio samples from VCTK were grouped together.

- Joint training of both fine-tuned encoders
- Joint training of both encoders without fine-tuning
- Joint training of both fine-tuned encoders with SBERT freezing
- Joint training of both fine-tuned encoders with Wav2Vec2 freezing

The reason why we decided to try joint training of both encoders without fine-tuning is that we wanted to understand how important is the role of standalone fine-tuning of the encoders here.

Regardless of the training combinations, the hyper-parameters were always the following (see Table 4):

Table 4. Joint training hyper-parameters.

Hyper-parameter	**Value**
Wav2Vec2 learning rate	1e-5
SBERT learning rate	1e-6
Epochs	200
Batch size	8
Loss function	MAE loss
Wav2Vec2 Optimizer	Adam
SBERT Optimizer	Adam

Again, because of the Wav2Vec2 model, we couldn't use a higher batch size than 8 due to the constant "`CUDA out of memory`" issue. Freezing of the model's parameters was implemented by adding a condition and `model.eval()` to the PyTorch code.

We describe the results of the joint training in the next section.

6 Results

Within this section, we are going to describe the results of training text and audio encoders jointly. We compare their scores and decide which method for joint training is more efficient. We also compare the difference between encoding the text-audio pairs with high quality vs. low quality of the annotation.

6.1 Joint training results

In total, we conducted experiments with four different combinations of encoders in order to find the most effective way to build such a system.

The most direct and straightforward approach was to make use of the fine-tuned encoders that we trained independently. As both of these encoders were able to generate accurate vector representations, we only needed to adjust their outputs to be compatible with each other.

This training was conducted for several epochs but was stopped after the 14th iteration because of overfitting. The best mean absolute error that we could achieve with this method on the evaluation set was 0.292, which is higher (worse) than each of the models achieved in standalone training. Although this was not a bad result, there were three more experiments to conduct.

Next, we wanted to look at how well the encoders are able to learn vector representations of each other's outputs without prior fine-tuning. For that, we took both models from Hugging Face and put them to train together jointly. This time, the results were much worse. The mean absolute error on the evaluation score could not go lower than 0.387. From that, we concluded that prior separate fine-tuning of the models was essential and significantly improved the performance.

The last two experiments included the freezing of either of the models. We started with freezing the Wav2Vec2 since it achieved a higher score in the standalone training. Also, it is a more complex model with more parameters than SBERT, and we assumed that it would be simpler to adapt the sentence encoder to Wav2Vec2. In reality, SBERT was not able to adapt well enough, and the models could not achieve the MAE score better than 0.347.

The final experiment that we tried involved freezing fine-tuned SBERT encoder. This method has achieved the best results over all three experiments. The training and evaluation logs can be seen in Figure 13. The lowest MAE score during the evaluation was set at 0.259 at the 17th iteration. After that, the model started to overfit slowly, and the training was stopped. What is impressive is that SBERT alone achieved a higher MAE score in a standalone training, but in combination with Wav2Vec2, the mean absolute error became even lower.

Once we had completed the training of all four distinct combinations, the next step was to evaluate the models using a control samples dataset that had been previously

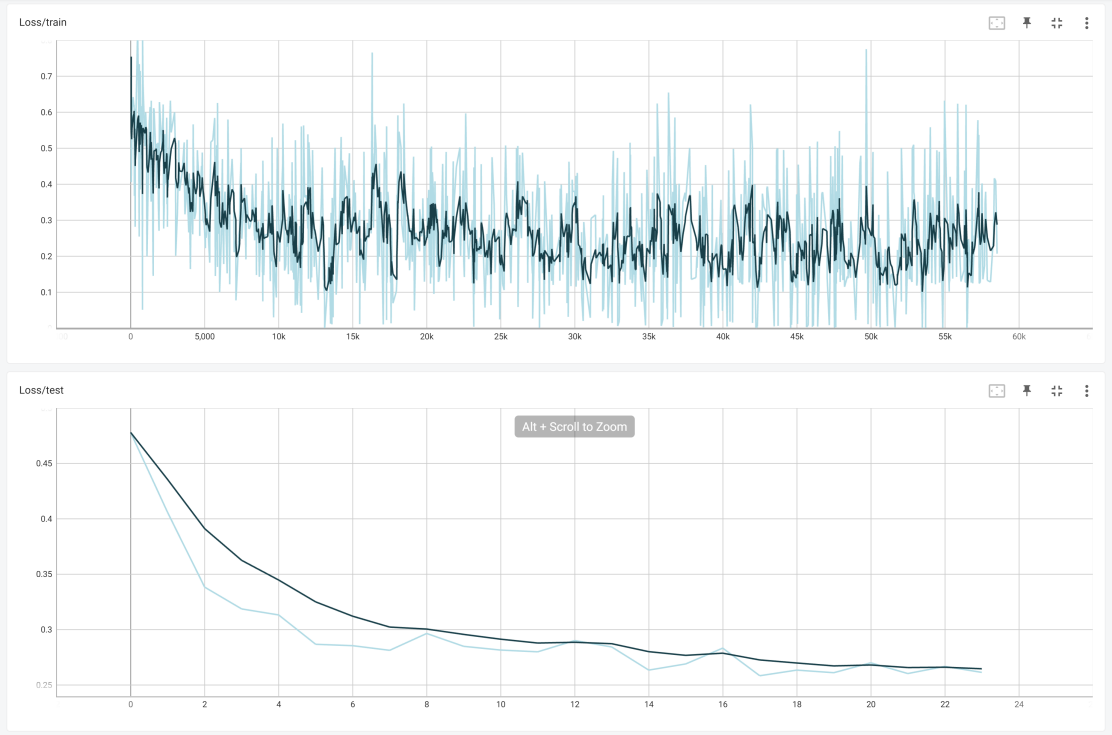

Figure 13. Training and evaluation log of joint training of both encoders with enabled SBERT freezing made in Tensorboard. Smoothing applied: 0.7

extracted. In order to establish a baseline for the results, we determined the scores obtained by evaluating Wav2Vec2 and SBERT models that had undergone standalone training. These models were capable of achieving favorable outcomes independently but had not been specifically trained to work in conjunction with one another. The results can be seen in Table 5.

Overall, all of the experiments performed better than the baseline. We found that the most effective approach was to freeze SBERT when training the models together, resulting in the lowest mean absolute error of 0.258. The second-best score was achieved by training both fine-tuned encoders without freezing either of the models. On the other hand, we have found that models without prior fine-tuning in standalone mode produced the poorest results when trained together. This highlights the significance of prior fine-tuning.

Table 5. Evaluation of the experiments on extracted control samples.

Experiment	**MAE Score**
Baseline (fine-tuned Wav2Vec2 & SBERT w/o joint training)	0.518
fine-tuned Wav2Vec2 & fine-tuned SBERT	0.289
zero-shot Wav2Vec2 & zero-shot SBERT	0.387
fine-tuned Wav2Vec2 freezing & fine-tuned SBERT	0.354
fine-tuned Wav2Vec2 & fine-tuned SBERT freezing	**0.258**

6.2 Annotation quality impact assessment

As a part of our experiment, we aimed to explore how strongly the annotation quality correlates with the accuracy of predictions. In the data collection section, we described the process of data collection and wrote about extracting control samples. We asked one of the participants to evaluate the quality of each of the 900 control samples. This participant was listening to audio samples, then she read the corresponding textual description and assessed how much the description matched the voice in the recording. We displayed the results of this assessment in Table 1.

For this experiment, we decided to use the best model combination which has shown the most satisfactory results in joint training - the one with freezing of the SBERT encoder's parameters. The results are displayed in Table 6.

Table 6. MAE loss dependency from textual annotation quality.

Annotation quality	**MAE loss**
Great, correct, and exhaustive	**0.230**
partial	0.253
wrong or misleading	0.001
minimal or none	0.998
other - specify in a comment	0.001

As it was expected, the utilization of correct and exhaustive textual voice descriptions results in a better performance. The cosine similarity between these descriptions and their corresponding voice recordings was predicted with a mean absolute error of 0.230. However, partially accurate descriptions resulted in a slightly worse score: 0.253. It is noteworthy that wrong and misleading annotations received a score of 0.001. However, this result should not be taken into account since there were only two samples in this category.

7 Conclusion

Vector representation is a fundamental concept in machine learning. This concept found use in numerous applications [8, 15, 13, 14] and a lot of research is being conducted in this field.

In this work, we presented our new system for vector representation. The system we developed is able to encode audio voice recordings and corresponding textual descriptions of the voice. For encoding both text and audio we utilized state-of-the-art transformer-based technologies: SBERT and Wav2Vec2 respectively.

We experimented with different training combinations of these encoders as well as researched the importance of having high-quality annotations in the dataset.

In the Introduction section, we have defined 4 research questions which we are going to answer:

RQ1: How well can this system perform? How accurate would be vector representations?

- Accuracy of the vector representation greatly depends on the implementation method. Training the encoders jointly allows them to adapt their output embeddings so that corresponding voice descriptions and recordings have the highest cosine similarity between them. With the correct implementation, we managed to reach the MAE loss of 0.258.

RQ2: Can prior fine-tuning of the encoders in standalone mode increase the overall accuracy of the predictions?

- Prior fine-tuning of the models can significantly improve the accuracy of the vector representations and decrease mean absolute error (by 0.1 in our case).

RQ3: How can freezing either encoder during joint training affect the accuracy of vector representations?

- Freezing the weight of SBERT during joint training increases the performance of the system while freezing Wav2Vec2 has the opposite effect. It may be caused by the fact that the Wav2Vec2 encoder's architecture is far more complex and it has more capabilities to adapt its outputs to the embeddings produced by another model, such as SBERT.

RQ4: How much of an impact can the quality of textual annotations of the voice characteristics have on the prediction accuracy?

- We found that exhaustive and detailed voice descriptions can result in slightly better accuracy of vector representations. At the same time, we found that mentioning

the speaker's gender has a more significant impact on embeddings than describing any other voice characteristics.

As we discussed previously, this system may be utilized in different applications. First of all, it can be used for the development of text-to-speech systems with ability to generate voice by their descriptions. Conversely, another application could be voice description generation from the recording. That would require connecting decoder part of the transformer such as GPT-3 [6] to the system. Lastly, with these models it is possible to implement a voice search by the search query by transforming it into the vector space.

There are also few things that can potentially improve the performance of our system:

1. The similarity datasets that were addressed in section 3.4 have potential for improvement. The current approach used in this study involved assigning a similarity score of either 1 or 0 to pairs of samples in the datasets. If the samples were from different speakers, they were automatically assigned a similarity score of 0. However, this approach has a limitation as it does not consider the possibility that samples from different individuals may be too similar to be assigned a score of 0.

2. It is possible that excluding the Common Voice samples from the audio dataset could enhance the accuracy of predictions. This is because a considerable number of recordings in the Common Voice dataset have low quality and contain significant background noise. However, it is worth noting that the Common Voice dataset offers a more diverse range of voices compared to the VCTK dataset. Therefore, including Common Voice samples in the dataset could potentially improve in the generalization ability of the encoders.

3. One last potential method for enhancing the vector representations is to experiment with different dimensionalities. In our present study, we utilized the default dimensionality of the SBERT encoder and reduced the dimensionality of the Wav2Vec2 outputs. However, modifying the dimensionality through further downsampling or upsampling could potentially impact the accuracy of our predictions.

References

[1] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[2] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M Tyers, and Gregor Weber. Common voice: A massively-multilingual speech corpus, 2019.

[3] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2017.

[5] Hervé Bredin, Ruiqing Yin, Juan Manuel Coria, Gregory Gelly, Pavel Korshunov, Marvin Lavechin, Diego Fustes, Hadrien Titeux, Wassim Bouaziz, and Marie-Philippe Gill. pyannote.audio: neural building blocks for speaker diarization, 2019.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[8] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel. Audioclip: Extending clip to image, text and audio, 2021.

[9] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel. Esresne(x)t-fbsp: Learning robust time-frequency transformation of audio, 04 2021.

[10] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[13] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[14] Wisam A Qader, Musa M Ameen, and Bilal I Ahmed. An overview of bag of words; importance, implementation, applications, and challenges. In *2019 international engineering conference (IEC)*, pages 200–204. IEEE, 2019.

[15] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[16] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[17] Xin Rong. word2vec parameter learning explained, 2016.

[18] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding, 2020.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[20] Christophe Veaux, Junichi Yamagishi, Kirsten MacDonald, et al. Cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit, 2017.

Appendix

I. Repository

The source code for the PyTorch implementation of this project can be found by following this link:
https://github.com/TartuNLP/speaker-CLAP

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Ivan Lastovko**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Joint Embeddings for Voices and Their Textual Descriptions,

   supervised by Mark Fishel.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Ivan Lastovko
09/05/2023