# Development of an Albanian Language Transcriber using Artificial Intelligence

by

Florijan Qosja
001079978

Dissertation submitted in partial fulfillment of
the requirements for the
BSc Hons COMPUTER SCIENCE (EXT-UGIC)
(Computing and Mathematical Science)

April 2023

University of Greenwich
Old Royal Naval College, Park Row
London SE10 9LS

CERTIFICATION OF APPROVAL

**Development of an Albanian Language Transcriber using Artificial Intelligence**

by

Florijan Qosja
001079978

A project dissertation submitted to the
Computing and Mathematical Science Programme
University of Greenwich
In partial fulfillment of the requirements for the
BSC HONS COMPUTER SCIENCE (EXT-UGIC)
(COMPUTING AND MATHEMATICAL SCIENCE)

Approved by,

_____
Dr. Christopher Beckwith

University of Greenwich
Old Royal Naval College, Park Row, London SE10 9LS
April 2023

## CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

Florijan Qosja

# ABSTRACT

This report presents a specialized Albanian speech-to-text transcriber with an accuracy target of 45% or higher. The implementation uses a deep learning model, based on the popular Deepspeech architecture, that is optimized for the specific characteristics of the Albanian language. A user-friendly web interface has been developed to enable transcribing, labeling, and validating Albanian speech data, as well as an API that can be easily integrated with other systems.

To train and evaluate the model, a large dataset of over 40 hours of Albanian speech data, including various accents, dialects, and speaking styles, is collected and labeled. The dataset enables fine-tuning the model and exceeding the initial accuracy target, achieving an accuracy of 46.3

Moreover, a platform is established that can enhance the development of Albanian Automatic Speech Recognition (ASR) models, including creating ASR datasets and language corpora. The latest technologies and popular, well-documented frameworks are used, making it easy for other contributors to engage with the project.

Overall, this project represents a significant contribution to the development of the Albanian language processing technologies and will have a positive impact on various applications, such as voice recognition, speech analysis, and natural language processing.

# PREFACE

The ability to transcribe speech into text automatically has become an essential tool in many fields, from improving accessibility for people with hearing impairments to enhancing natural language processing in human-computer interaction. However, most existing Automatic Speech Recognition (ASR) systems are designed for major languages and struggle with less commonly spoken ones. Albanian, for example, is a language spoken by over 5 million people worldwide, but has received little attention in ASR research.

In this project, we aimed to fill this gap by developing a specialised speech-to-text Albanian transcriber. Our approach involved adapting the architecture of the popular DeepSpeech model architecture to the unique characteristics of Albanian phonology and orthography, and training it on a newly created Albanian speech dataset. We also built a web interface and API to enable easy access to our model and facilitate the collection and annotation of more Albanian speech data.

This report documents our implementation process and evaluates the accuracy of our transcriber on various test sets. We also discuss the potential applications and impact of our work, as well as the limitations and future directions for improvement. We hope that our project will contribute to the advancement of ASR research for the Albanian language and inspire further developments for other underrepresented languages.

# ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Dr Christopher Beckwith, for his unwavering support and mentorship throughout this project. His trust in me and willingness to supervise me in a challenging dissertation topic have been invaluable. His belief in my ability to achieve this task has given me the mental and emotional strength to accomplish it.

I would also like to extend my heartfelt thanks to my fiancee, who has provided unwavering support and encouragement throughout my academic journey. Despite the long hours of study and research, she has stood by me and provided a source of inspiration and motivation.

I am grateful to my parents for their financial support and continuous encouragement, especially during the difficult times of my academic journey. Their unwavering support and belief in me have been a source of strength, and I cannot thank them enough.

Lastly, I would like to express my gratitude to my only sister, who has been a constant reminder of my end goal and has provided unconditional support when needed. Her unwavering belief in me has been a great motivation throughout my academic journey.

TABLE OF CONTENTS

LIST OF FIGURES

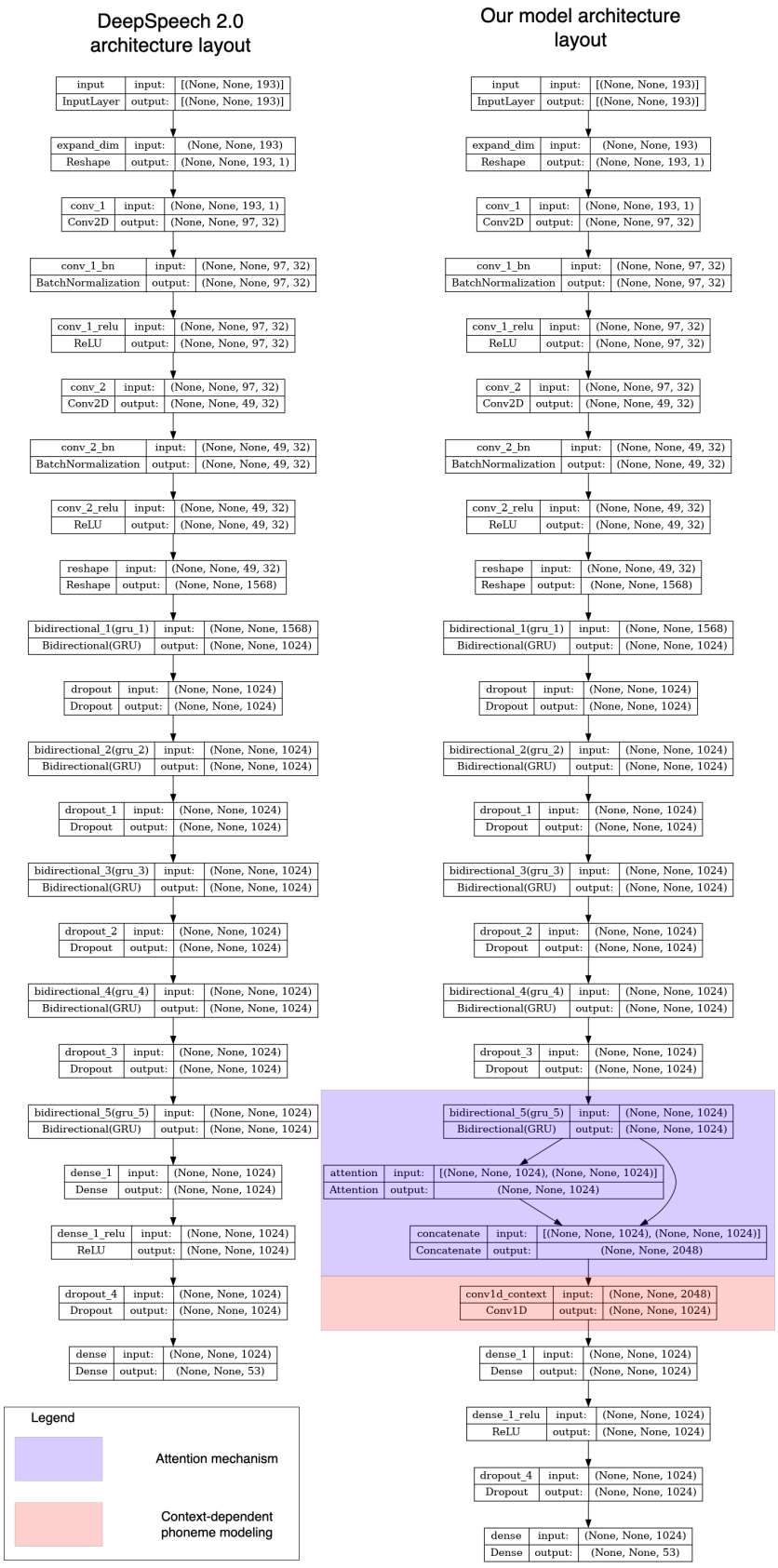


Figure 1: Models Architecture

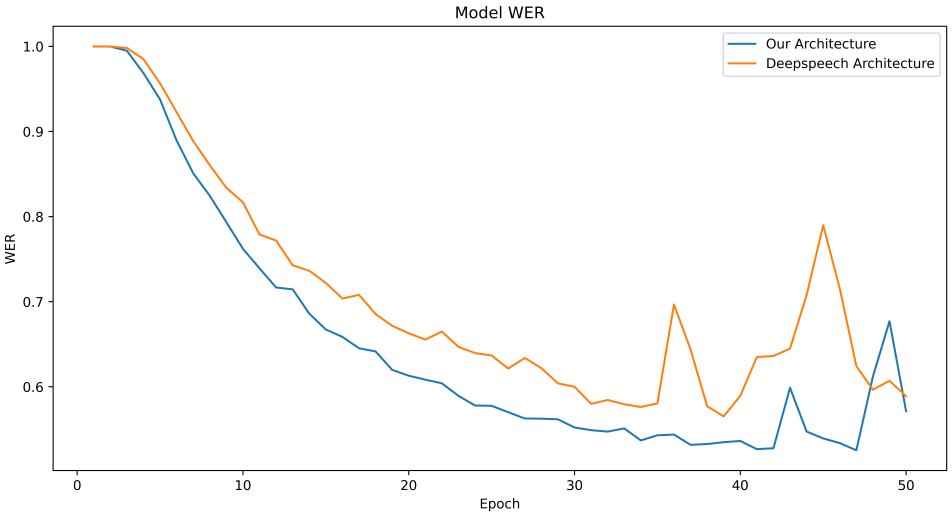


Figure 2: WER

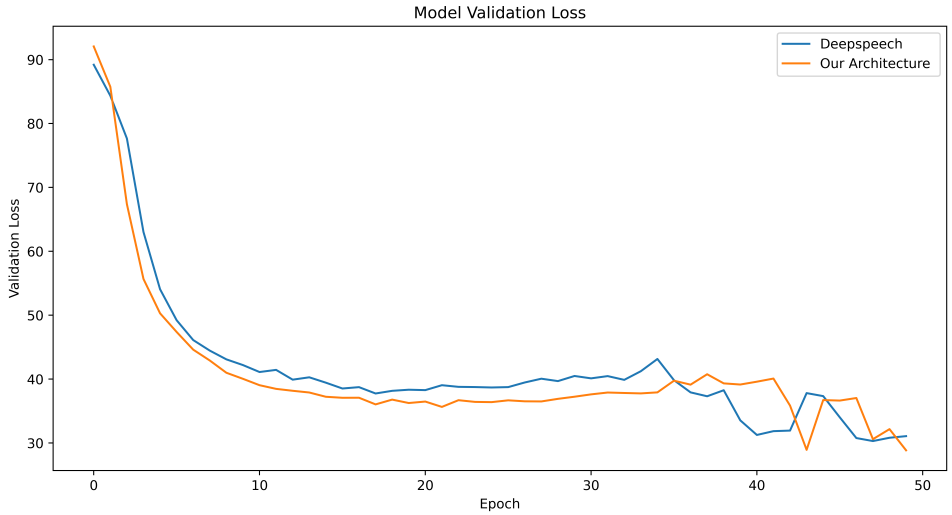


Figure 3: Validation Loss

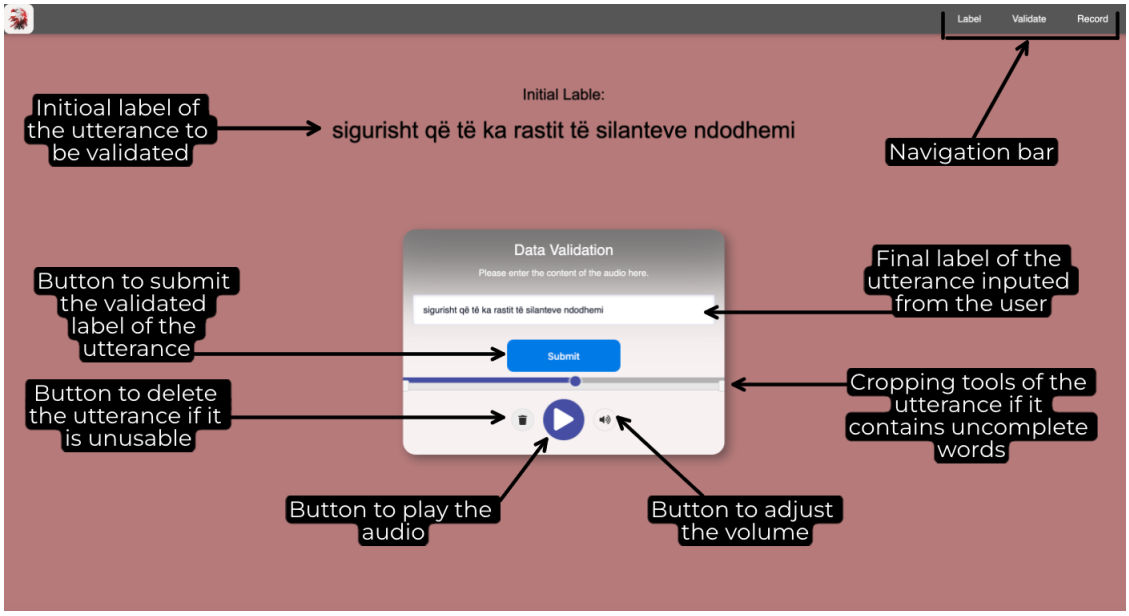


Figure 4: Web interface label validation page

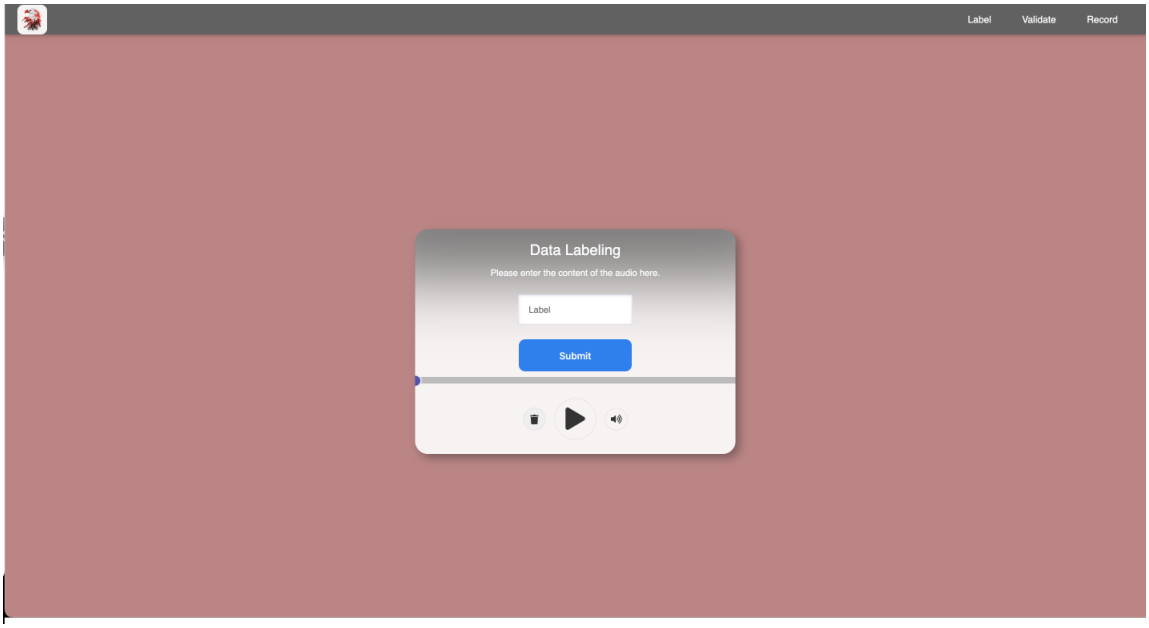


Figure 5: Web interface initial labelling page

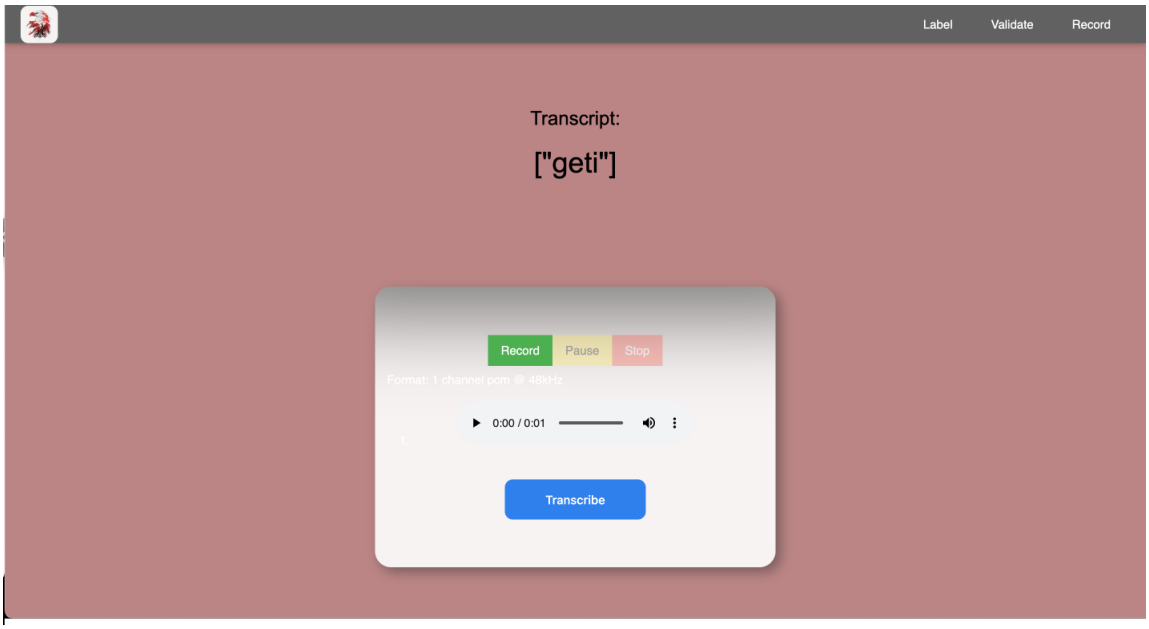


Figure 6: Web interface transcribe page

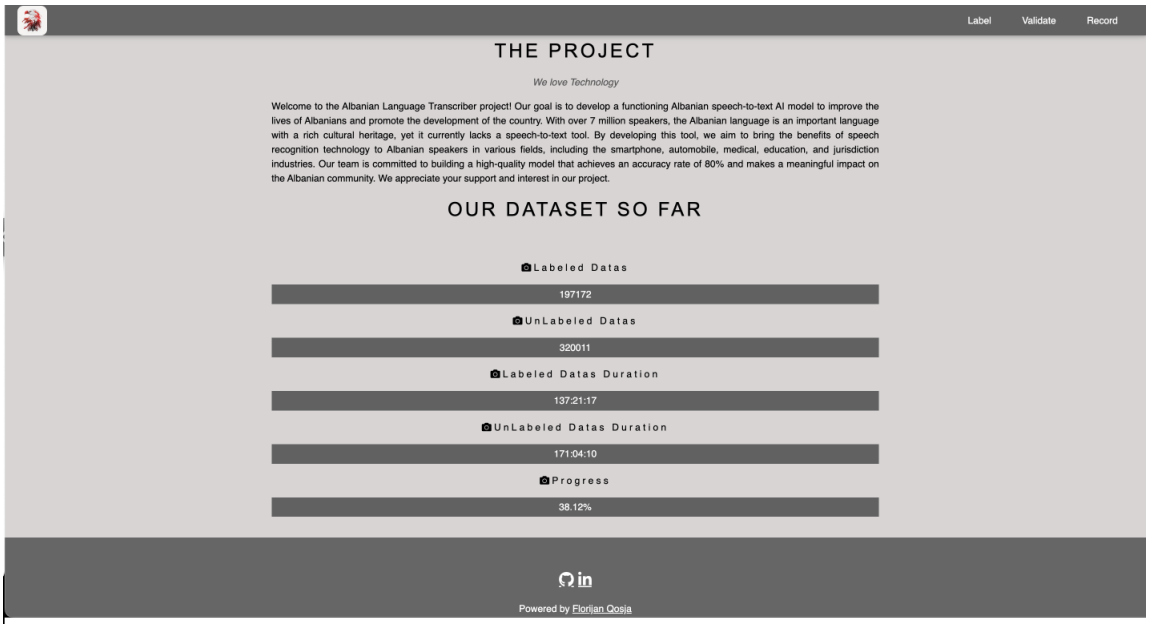


Figure 7: Web interface information section

# LIST OF ABBREVIATIONS

| | |
|---|---|
| $ASR$ | Automatic Speech Recognition |
| $API$ | Application programming interface |
| $RNN$ | Recurrent neural network |
| $CTC$ | Connectionist temporal classification |
| $ReLu$ | Rectified linear unit |
| $WER$ | Word Error Rate |
| $CER$ | Character Error Rate |
| $CNN$ | Convolutional neural network |
| $GPU$ | Graphics processing unit |

# CHAPTER 1

# INTRODUCTION

Albania is located in Southeastern Europe and is bordered by Montenegro, Kosova, North Macedonia and Greece to the south. The Albanian language is considered to be a unique branch within the Indo-European language family, with an estimated 7-8 million speakers worldwide. There are two primary dialects: Gejg, spoken in the north, and Tosk, spoken in the south. The standard Albanian language is based on the Tosk dialect.

Automatic Speech Recognition (ASR) technology has made significant strides in recent years, enabling the transformation of spoken language into written text. This has applications in transcription services, voice assistants, and various other domains. However, the development of ASR systems for different languages is influenced by each language's unique characteristics.

Despite being located in Europe, a region with advanced technology, Albania's technology advancement is not as close to that of other nations in the region. This project focuses on the technology advancement in Albania, specifically in the area of language applications. The Albanian language lacks such a technology, which is a key feature in multiple industries such as phone and computer, medical, education, automotive, media and many more.

# CHAPTER 2

# LITERATURE REVIEW

Automatic speech recognition (ASR) is an intelligent system that is able to generate sequence of letters from an audio speech input. AI-powered speech-to-text technology has come a long way since its inception. In the early days, the technology was limited to basic dictation and simple transcription, but it has since evolved to be more sophisticated and accurate. With the advancements in deep learning and neural networks, the technology has become more capable of transcribing speech with high accuracy and in multiple languages.

## 2.1 Development and use of ASR's

In the development of ASR systems, two primary components play a crucial role in determining the system's overall performance: the model architecture and the dataset. The model architecture refers to the structure and organization of the algorithms used to process the input audio signal and generate the corresponding textual transcription. This includes choices between end-to-end deep learning models or traditional, modular ASR systems. The dataset, on the other hand, represents the collection of labeled speech data used to train and validate the ASR model. The quality, diversity, and size of the dataset directly impact the model's ability to accurately recognize and transcribe various speech patterns, accents, and languages. A well-designed model architecture, paired with an appropriate dataset, contributes significantly to the success of an ASR system in delivering accurate and reliable speech recognition performance.

Speech-to-text technology has been developed for a variety of languages. Some

of the systems with the best accuracy developed to dates are shown in table 1. they are classified in two categories based on their architecture, end-to-end and traditional systems.

In an end-to-end ASR system, the entire process of converting spoken language into written text is managed by a single, comprehensive model. This type of system typically utilizes deep learning techniques, such as RNNs or transformer models, to process the audio input and generate text output. The model is trained on vast amounts of data, which enables it to learn the intricate patterns and nuances of human speech. This streamlined architecture eliminates the need for multiple components or complex pipelines that are often found in traditional ASR systems. Some popular examples are:

- DeepSpeech by Mozilla: DeepSpeech is an open-source ASR system developed by Mozilla based on the Baidu Research's Deep Speech research paper. It utilizes deep learning techniques and RNNs for speech recognition.

- Wav2Letter by Facebook AI Research (FAIR): Wav2Letter is another open-source ASR system that uses a convolutional neural network (CNN) architecture to perform speech recognition tasks. It is designed to be simple and efficient while still delivering competitive performance.

- Listen, Attend, and Spell (LAS): LAS is an end-to-end ASR model that uses an attention-based mechanism to generate transcriptions directly from the audio signal. It incorporates an encoder-decoder architecture with an attention mechanism that allows the model to learn alignments between the input audio and the output text.

On the other hand, traditional ASR systems consist of multiple components working together in a sequential manner. The process typically begins with feature extraction, wherein the audio signal is converted into a set of features that represent the acoustic properties of the speech. This is followed by an acoustic model that maps these features to phonetic units, such as phonemes or sub-phonemes. The next stage involves a language model, which uses the phonetic units to generate a sequence of words based on

linguistic probabilities. Finally, a pronunciation model is employed to map the generated words to their corresponding phonetic units, which are then combined to produce the final transcription. Some popular examples are:

- Kaldi: Kaldi is a widely-used, open-source toolkit for developing ASR systems. It provides a modular framework that includes tools for feature extraction, acoustic modeling, language modeling, and pronunciation modeling. Kaldi can be used to build traditional ASR systems with custom components tailored to specific applications.

- HTK (Hidden Markov Model Toolkit): HTK is another popular toolkit for developing traditional ASR systems. It is primarily focused on using hidden Markov models (HMMs) for acoustic modeling and provides a range of tools for building ASR systems, including feature extraction, training, and decoding components.

- CMU Sphinx: CMU Sphinx is an open-source ASR system developed by Carnegie Mellon University. It includes various tools for building traditional ASR systems, such as acoustic models, language models, and pronunciation dictionaries. CMU Sphinx is designed to be highly adaptable and can be used in various applications, including transcription services, voice assistants, and more.

Comparatively, end-to-end ASR systems provide a more simplified and efficient approach, as they consolidate the entire speech recognition process into a single model. This makes the system easier to train, maintain, and update. However, traditional ASR systems, with their modular architecture, offer greater flexibility and customizability, allowing for fine-tuning and adaptation to specific domains or applications. While end-to-end ASR systems have demonstrated remarkable performance in recent years, end-to-end models, such as those based on deep learning techniques, often require massive amounts of labeled data to learn the complex relationships between the input audio signals and the output text transcriptions. The absence of sufficient training data can result in over-fitting, where the model becomes too specialized in recognizing the training data and fails to generalize well to unseen examples. Data efficiency is one of the biggest downsides of end to and models. Traditional ASR systems, conversely, can

be more data-efficient, particularly when leveraging expert knowledge or incorporating pre-existing components.

The accuracy of AI-powered speech-to-text technology varies based on several factors such as the quality of the audio, the language being transcribed, the environment where the audio was recorded, and the technology used as well as the size of the training dataset.

In general, the accuracy of these technologies has been steadily improving and the best ones currently have an accuracy of over 95% for the English language. However, some specific technologies have a higher accuracy rate than others. For example, Google's DeepMind's WaveNet has an accuracy of up to 98%. Similarly, Microsoft Azure Speech Services has an accuracy rate of around 96% for American English. On the other hand, some of the open-source implementations of speech-to-text technology like Kaldi Speech Recognition Toolkit have an accuracy of around 90%. It's important to note that the accuracy of these technologies is constantly improving, and it's expected that the accuracy will continue to increase in the future.

The data collection methods used vary depending on the language being studied. In some cases, the data is collected through crowd sourcing, where individuals are paid to transcribe audio recordings. In other cases, the data is collected through partnerships with companies and organizations that have access to large amounts of audio data.

The impact of speech-to-text technology on various industries has been significant. In the legal industry, for example, speech-to-text technology has made it easier to transcribe depositions, hearings, and other legal proceedings. In the medical industry, ASR technologies has made it easier to transcribe doctor-patient conversations and other medical records. In the education industry, speech-to-text technology has made it easier to transcribe lectures and other educational content. ASR has been used in the smartphone and computer industry to provide users with a hands-free experience, allowing them to dictate text messages and emails, control their devices through voice commands, and access voice assistants like Siri and Alexa [1]. In the automotive industry

they are used to provide hands-free voice assistants for drivers, allowing them to control various functions on their cars without having to take their hands off the wheel or their eyes off the road. In the media industry ASR technology has played a significant role , allowing for the automatic transcription of interviews, speeches, and other forms of audio content, making it easier to create transcripts for closed captioning, subtitling, and other purposes.

## 2.2 Development of ASR's for the Albanian language

Despite all of the advancement to date, the development of AI-powered speech-to-text technology in general faces several challenges, some of which are: Accurate transcription, languages and accents, handling background noise and interference, processing speed, privacy and security, post-effectiveness, integration with existing systems and many more.

The challenges faced in developing speech-to-text technology for different languages vary depending on the language being studied. For languages with complex grammar and syntax, such as Japanese, the technology needs to be able to handle the nuances of the language. For languages with a large number of speakers across different regions, such as Spanish, the technology needs to be able to handle different accents and dialects.

The development of AI-powered speech-to-text technology for low resource languages, specifically the Albanian language, faces several challenges, some of which are:

- Lack of data: The primary challenge lies in the scarcity of data and resources for low-resource languages like Albanian. Large datasets are crucial for training AI models, as they allow the models to learn the nuances of a language and improve their performance. However, the limited availability of such datasets for Albanian hampers the training of accurate and effective AI models.

- Complex linguistic structure: Albanian has a unique and complex linguistic structure, characterized by its distinct grammar and diverse accents. Pronunciation can vary significantly based on factors such as geographical location, social and cultural background, and age of the speaker. The Albanian language is known for its wide variety of dialects and accents. For instance, a speaker from Tirana, the capital of Albania, may exhibit different pronunciation and accent compared to a speaker from a rural area. This diversity presents a challenge for speech-to-text technology, as it struggles to accommodate and accurately transcribe the various accents and pronunciations.

- Lack of research: Research into AI-powered speech-to-text technology for the Albanian language has been limited. This scarcity of research hampers the growth of knowledge and understanding of the specific challenges and requirements associated with the Albanian language. Consequently, it becomes difficult to develop targeted strategies and solutions to improve the performance of speech-to-text systems for Albanian speakers.

Addressing these challenges effectively requires the creation of customized model architectures tailored to each language characteristics. This approach ensures that the resulting models deliver high performance and accuracy in their respective language domains, enabling a more seamless and accurate speech-to-text experience.

## 2.3 Characteristics of the Albanian language

A review of the linguistic characteristics of the Albanian language, including its phonology, morphosyntax, and lexicon, and how these characteristics may impact the development of an Albanian speech-to-text model.

Albanian is an Indo-European language spoken primarily in Albania, Kosova, and parts of North Macedonia, Montenegro, and Greece and italy. It is considered a unique branch within the Indo-European language family, with an estimated 5-7 million speakers worldwide. There are two primary dialects: Gheg, spoken in the north, and Tosk, spoken in the south. The standard Albanian language is based on the Tosk dialect.

### 2.3.1 Rich vowel system

Albanian has a relatively large vowel inventory, with seven oral vowels and two nasal vowels. Additionally, each oral vowel has two forms: long and short. ASR systems must be capable of accurately distinguishing between these sounds, as misidentifying them can lead to errors in transcriptions.

Example: *The words "verë" (summer) and "verë" (wine) differ only in vowel length.*

Drawing on the experience of ASR systems for other languages with rich vowel systems, like Chinese, it is possible to implement more specialized layers in the model to focus on vowel distinctions. [2] These layers can be pretrained on a large dataset to fine-tune the recognition of Albanian vowels.

### 2.3.2 Consonant clusters

Albanian permits complex consonant clusters, which can sometimes be difficult for ASR systems to parse accurately. The development of an ASR system for Albanian requires careful consideration of these clusters to avoid misunderstandings in transcribed text.

Example: *In the word "ndihmë" (help), the consonant cluster "nd" is pronounced as a single sound.*

ASR systems for languages with complex consonant clusters can benefit from implementing context-dependent phoneme models. These models consider the surrounding phonemes to better estimate the pronunciation of a given consonant cluster. [3]

### 2.3.3 Tonal stress

Albanian exhibits lexical stress, which can fall on any syllable in a word. Accurate stress identification is crucial for ASR systems to differentiate between homographs – words with the same spelling but different meanings and stress patterns.

Example: *"drejtori" (head of an institution) and "drejtori" (main office of an*

*institution) have different stress patterns, where the first syllable is stressed in the former and the second vawl in the latter.*

A recent study conducted on an ASR model trained on a dataset comprising children's speech in the English language has demonstrated that the inclusion of prosodic features significantly improves the accuracy of the system. [4] This finding highlights the potential of prosodic features in enhancing the performance of ASR models for the Albanian language.(Role of Prosodic Features on Children's Speech Recognition)

### 2.3.4 Similar words with similar stress but with different meaning

Example: *"bar" (grass) or (medication) have the exact stress patterns but 2 totally different meaning.*

### 2.3.5 Rich inflectional system

Albanian is an inflectional language with a complex system of cases, tenses, and moods. The development of an ASR system for Albanian requires a deep understanding of these morphological rules to accurately recognize and transcribe words in various inflected forms.

Example: *The verb "të shkruaj" (to write) has numerous inflected forms, such as "shkruaj" (I write), "shkruan" (he/she writes), "shkruam" (we wrote), "shkrova" (I wrote), "shkruaja" (I used to write)*

An approach used for languages with rich inflectional systems, is to employ subword units, such as morphemes or phoneme clusters, in the ASR models [5]. This enables the system to capture the underlying structure of words and generalize across different inflections.

### 2.3.6 Word formation

Albanian exhibits agglutination, where words are formed by combining multiple morphemes. This characteristic can be challenging for ASR systems as they need to identify the boundaries between morphemes and recognize the morphological structure of words to produce accurate transcriptions.

Example: *The word "pamundësi" (impossibility) is formed by combining "pa-" (without), "mund" (possibility), and "-ësi" (suffix for abstract nouns).*

Techniques such as Byte Pair Encoding (BPE) or other subword tokenization methods, have shown improvements in performance of ASR's [6]. Hence, implementation of such techniques can help ASR systems better segment and recognize morpheme boundaries in Albanian.

### 2.3.7 Free word order

Albanian has relatively flexible word order, especially in comparison to English. While the subject-verb-object (SVO) order is most common, other orders are also possible. ASR systems must be designed to accommodate this variability, identifying the grammatical roles of words regardless of their position in a sentence.

Example: *The sentence "Unë e kam lexuar librin" (I have read the book) can also be expressed as "Librin e kam lexuar unë" without altering the meaning.*

Incorporating syntactic parsers into ASR systems, as done in languages like Japanese and Mandarin Chinese, can help the system identify and process different word orders [7]. This approach enables the ASR to adapt to the variability in Albanian sentence structures.

### 2.3.8 Use of particles

Particles are small function words that often convey subtle grammatical or pragmatic information in a sentence. In Albanian, particles like "po," "do," and "mos" can

indicate tense, mood, or negation. ASR systems need to accurately identify and process these particles to ensure correct transcription and interpretation of speech.

Example: *The particle "do" can be used to indicate future tense, as in "do të shkoj" (I will go).*

Languages like Chinese, which also rely heavily on particles, have benefited from incorporating attention mechanisms in their ASR systems. Attention mechanisms can help the model focus on relevant particles and improve the overall understanding of the sentence structure [8].

### 2.3.9 Borrowings and loanwords

Albanian has a substantial number of loanwords from various languages, including Latin, Turkish, Greek, Italian, and Slavic languages. ASR systems must be able to recognize these loanwords and transcribe them accurately, despite potential differences in pronunciation or phonetic patterns.

Example: *The word "perde" (curtains) is derived from the Turkish word "perdeler"*

By leveraging the proposed MetaAdapter and SimAdapter algorithms, ASR systems for Albanian can potentially improve their accuracy in recognizing loanwords from various languages [9]. The attention mechanism used in SimAdapter can help learn the similarities between Albanian and the source languages, while MetaAdapter can accelerate the training process and improve the overall performance.

### 2.3.10 Idiomatic expressions

Like all languages, Albanian contains idiomatic expressions that can be challenging for ASR systems. These expressions are often non-literal and can be difficult for an ASR system to interpret without a proper understanding of the language's idioms.

Example: *The expression "nuk e vë ujin ne zjarr" (lit. does not put the water in fire) means "He does not care"*

By incorporating a large-scale idiom database and integrating semantic understanding techniques, ASR systems can improve their ability to recognize and process idiomatic expressions in Albanian. This approach has been successful in languages like English, where ASR systems are trained on vast amounts of idiomatic data to better comprehend non-literal expressions [10].

## 2.4 Deepspeech

DeepSpeech is a neural network-based speech recognition system that uses a deep learning approach to convert speech signals into written text. The core architecture of DeepSpeech is based on a combination of CNNs and RNNs.[11]

CNNs are commonly used in image recognition tasks, but they can also be applied to speech recognition. In the case of DeepSpeech, the CNNs are used to extract features from the raw audio signal. The input to the CNNs is a sequence of audio frames, and the output is a sequence of feature vectors. The CNNs are trained to extract features that are relevant for speech recognition, such as spectral information and temporal patterns. [11]

The feature vectors extracted by the CNNs are then fed into the RNNs, which are used to model the temporal dependencies in the speech signal. RNNs are a type of neural network that can handle sequential data, such as speech signals, by maintaining an internal state that captures the context of the input. The output of the RNNs is a sequence of probability distributions over the possible outputs, which in the case of speech recognition are the different phonemes or words in the language.[11]

The training of the DeepSpeech model involves minimizing a loss function that measures the difference between the predicted output and the true output. The loss function used in DeepSpeech is the Connectionist Temporal Classification (CTC) loss, which allows the model to handle variable-length input and output sequences. The CTC loss is defined as follows:

$$L_{CTC} = -\log \sum_{\pi \in B(x)} p(\pi|x)$$

where $B(x)$ is the set of all possible alignments between the input sequence $x$ and the output sequence, and $p(\pi|x)$ is the probability of alignment $\pi$ given the input sequence $x$. The goal of the training process is to find the parameters of the model that minimize the CTC loss over a large dataset of labeled speech samples.

DeepSpeech has achieved state-of-the-art performance on several speech recognition benchmarks, including the Wall Street Journal dataset and the LibriSpeech dataset. The model has also been used for speech recognition in multiple languages, including English, Mandarin, and Arabic.[11]

### 2.5 Ethical considerations

When building an ASR system, it is crucial to consider the ethical implications that may arise during its development and deployment. Some key ethical considerations include:

- Data Privacy: ASR systems are trained on large amounts of speech data. It's essential to ensure that user data is collected, stored, and processed securely and in accordance with privacy laws and regulations. Obtaining informed consent from users and anonymising data can help protect privacy [12] .

- Bias and Fairness: ASR systems can inadvertently learn biases present in the training data, leading to unfair treatment of certain groups or accents. It is important to use diverse and representative datasets and to continuously evaluate and mitigate biases in the system [12] .

- Accessibility and Inclusively: ASR systems should be designed to work effectively for users with diverse backgrounds, languages, accents, and speech patterns. Ensuring that the system is inclusive and accessible to all users is a significant ethical consideration).

- Transparency and Accountability: Clearly communicate how the ASR system works, its limitations, and potential biases. Establish mechanisms to address user concerns, accept feedback, and make necessary improvements.

- Employment and Labor Impact: ASR systems could potentially replace or disrupt jobs that rely on human transcription or translation services. It is important to consider the impact on labor markets and promote opportunities for workforce transition and reskilling.

# CHAPTER 3

# MAIN CHAPTERS

## 3.1 Analysis

### 3.1.1 Motivation and background

Automatic Speech Recognition (ASR) systems are crucial for various industries, enabling efficient transcription of spoken language into written text. However, ASR systems for the Albanian language are limited or non-existent, despite the significant role they can play in improving communication and access to information for Albanian speakers. To address this gap, we have developed an open-source project called Project DibraFlet (DibraSpeaks), which aims to accelerate the development of ASR technology for the Albanian language.

The motivation behind Project DibraSpeaks is to create an environment where specialized ASR models for the Albanian language can be developed, along with building the largest speech dataset for the language. To facilitate this, we have established a domain named ”www.uneduashqiperine.com” (translated as ”I love Albania”) to serve as the platform for our project. The website offers a range of functionalities, including hosting the best trained ASR models available for download, providing access to the dataset for users to download, offering free speech-to-text transcription services, and allowing users to label and validate speech segmented audio data. Additionally, the website publishes regular updates about the project. All of these functionalities are served through an API hosted at ”api.uneduashqiperine.com”, which users can access at any time to integrate the functionalities into their own applications.

### 3.1.2 Existing research and technologies

Various research papers related to building ASR systems for the Albanian language claim to have used datasets for training, but no direct links or traces of dataset availability online can be found [13]. Additionally, a Chinese company called Speechocean claims to have a 200-hour corpus of the Albanian language for sale, named King-ASR-779, but it is only available to corporations and not individuals.

The CMU Wilderness Multilingual Speech Dataset, which contains audio and aligned sentences for around 700 languages, includes Albanian language data with a duration of 08:27:54. However, this dataset has limitations, such as having only one speaker conducting the entire speech and background music being played during the recordings. The varying nature of the background music makes it challenging to remove, and having only one speaker for such a long duration may result in an unbalanced dataset, negatively affecting ASR performance. [14]

VoxForge, an open-source speech corpus and acoustic model repository for ASR research, provides free publicly available speech data for training ASR systems. However, the Albanian language data in VoxForge, which consists of 29.5 minutes of audio recorded and transcribed in 2014 using a webcam microphone, is of poor quality with high background noise, and thus was not included in our dataset.

Several companies, including Google and OpenAI, provide transcription services for the Albanian language using their ASR models, but limited information is disclosed about their models. For instance, Google offers a paid transcription service through its Speech-to-Text API platform, but the model evaluation metrics are not publicly available, and the model is not integrated with other Google services such as keyboard or voice assistant. The only known feature of Google's model for Albanian is the profanity filter, without automatic punctuation, diarization, model adaptation, word-level confidence, or spoken punctuation.

OpenAI, a prominent company in the AI field, has developed a multilingual ASR model called "Whisper" with high accuracies in many languages[15]. However, even with its extensive training on 680,000 hours of transcribed audio data, OpenAI has

managed to collect only 5.7 hours of audio data for the Albanian language, and there are no quantitative metrics available on Whisper's performance in Albanian. Recent feedback on Whisper's performance for Albanian indicates that the results are unusable, even when using source audio from news channels, which are expected to be easier to transcribe.

Despite the existence of some datasets and transcription services for the Albanian language, there are limitations such as data scarcity, poor audio quality, and lack of quantitative performance metrics, which pose challenges integrating these existing datasets in our project.

### 3.1.3 Challenges and limitations

The development of ASR systems for the Albanian language faces significant challenges and limitations. Despite the willingness of developers and academics to contribute voluntarily to the improvement of ASR for Albanian since 2014, there has been a lack of initiatives to build the fundamental components of such a project. One of the major challenges is the scarcity of available data for training ASR models, as Albanian is considered a low resource language in terms of speech data availability. This limited data availability makes it difficult to train accurate and robust ASR models, as well as experiment with different model architectures specifically designed for improving accuracy in the context of the Albanian language.

Furthermore, the uniqueness of the Albanian language, including its distinct phonetics, grammar rules, and variations in dialects and accents, further complicates the development of specialized ASR models. Additionally, there is a lack of research in the field of ASR development for the Albanian language, which adds to the complexity of achieving this task. Given the magnitude of this challenge, our approach in this project is to create an environment where ASR models for Albanian can be developed, with us taking the first step by creating the initial iteration of the model creation using the best available dataset. We then publish these models on the project's domain, This will allow the dataset to grow and the project to gain popularity, more developers will contribute

to the ASR system, leading to continuous improvement in its accuracy.

### 3.1.4 Ethical considerations

Ethical considerations are a critical aspect of our project's existence, and obtaining consent for the use of audio data is of utmost importance. Building a dataset of the size we have, which includes 120 hours of labeled data, requires extensive effort. During the experimentation process to improve accuracy for the Albanian language, we utilized Albanian speech extracted from YouTube to demonstrate the viability of the overall process. However, in the actual deployment of the project, we will only use data from users who have explicitly granted consent for access.

To address biases and ensure fairness in our ASR models, we take into account the diverse accents and dialects present in the Albanian language. Given the significant variation in accents and dialects, we store information about the speaker's dialect and accent in our database. This enables us to measure the distribution of data portions on which the models are trained, providing insights into potential biases that may arise due to different accents. Our test dataset currently consists of academic Albanian for simplicity, but as the dataset is enriched with contents from other dialects and accents, the inclusion of these audio data will help reduce model bias. We are committed to continually refining our models to minimize biases and ensure fairness in our ASR system.

## 3.2 Requirements Specification

### 3.2.1 Dataset construction

In order to ensure the successful development of an accurate ASR system, the quality of the training dataset plays a crucial role. The inclusion of detailed information about each audio utterance can significantly impact the accuracy of the trained ASR model. This information encompasses various aspects, such as the speaker's age, gender, accent, and dialect. Additionally, details about the audio origin, such as whether it is from a podcast, lesson, or other sources, are also recorded. Furthermore, parameters

like audio speed and length in seconds are documented. Another important aspect is the validation state of the audio, which is assigned a numerical value ranging from 0 to 1. For instance, an unlabelled utterance is assigned a validation level of 0, while an utterance labeled by only one user is assigned a validation level of 0.5. When the same utterance is validated by another user with the exact same text, the validation level is updated to 1, indicating that the audio is 100% correct. Lastly, the audios in the dataset are in WAV format with a sample rate of 16000 Hz and mono channel. We not only include metadata information about the utterances that other datasets include, but we further include other entities to improve the insight informations about the dataset. In addition to including standard metadata information typically found in other datasets, our dataset goes beyond by incorporating additional entities to enhance the depth and quality of insights available from the dataset.

The reason for keeping track of each of these entities is to provide a comprehensive and well-annotated dataset for training an ASR system. The speaker-related information, such as age, gender, accent, and dialect, can impact the system's performance as speech characteristics vary among different speakers. The audio origin information helps in understanding the context and domain of the audio data, which may be useful in developing a system tailored to specific applications. Parameters like audio speed and length can be important factors in modeling speech dynamics and processing audio in real-time. The validation state of the audio ensures the quality and accuracy of the labeled data, which is crucial for training a reliable ASR system. Lastly, the audio format details, such as WAV format, sample rate, and mono channel, provide technical specifications that are essential for processing the data effectively.

### 3.2.2 API endpoints

A critical aspect of the project is the availability of information related to the dataset, as well as the dataset itself, and other related services through an API (Application Programming Interface). The API comprises 23 endpoints that offer a wide range of functionalities. These functionalities include serving the audio utterances, uploading trained models, uploading log data during training, labeling data, validating data, transcribing

audio data, and providing information about the dataset.

The endpoints are designed to cater to various requirements of the project. For instance, the endpoint for serving audio utterances allows access to the audio data stored in the dataset. The endpoint for uploading trained models facilitates the integration of trained models into the project. The endpoint for uploading log data during training allows for the logging and monitoring of training progress and performance. The endpoint for labeling data enables the annotation of unlabelled utterances with relevant information, such as speaker details, transcriptions, and other metadata. The endpoint for validating data helps in verifying the accuracy and quality of labeled utterances, with a validation level assigned based on multiple users' agreement. The endpoint for transcribing audio data performs automatic speech recognition to convert audio utterances into text transcriptions. Lastly, the endpoint for providing information about the dataset offers insights such as the total number of unlabelled, labeled, and validated utterances, along with their cumulative length in hours.

The availability of these functionalities through the API ensures efficient and effective management of the dataset, training of models, and processing of audio data. It allows for seamless integration with other systems and services, enabling the project to leverage the dataset and its related functionalities to achieve its objectives accurately and reliably.

### 3.2.3 Model Architecture

### 3.2.4 Model performance measure

The performance evaluation of ASR systems is commonly assessed using two widely accepted metrics: Word Error Rate (WER) and Character Error Rate (CER). WER measures the accuracy of the ASR system by quantifying the percentage of incorrectly recognized words compared to the total number of words in the reference transcription. A lower WER indicates higher accuracy, with a WER of 0.05 or lower (equivalent to 95% accuracy) often considered as a threshold for a model to be deemed suitable for real-world applications.

Similarly, CER measures the accuracy of the ASR system at the character level, calculating the percentage of incorrectly recognized characters compared to the total number of characters in the reference transcription. Both WER and CER are commonly used in ASR evaluations as they provide quantitative measures of the accuracy and performance of the system. In our evaluation, we employ these same metrics to assess the performance of our ASR model, with the goal of achieving optimal accuracy levels for real-world deployment.

### 3.2.5 Model transcriptions

As our project focuses on developing specialized ASR models for the Albanian language, we strive to provide users with the ability to test and experience the performance of our models in real-time. To facilitate this, we have implemented a dedicated web page where users can conveniently record their speech using their web browser, and receive real-time transcriptions in Albanian.

This feature serves as a valuable tool for users to assess the accuracy and effectiveness of our ASR models in real-world scenarios. By providing a user-friendly interface that allows for speech recording and immediate transcription in Albanian, we aim to enable users to experience the capabilities of our ASR models firsthand. This real-time testing functionality serves as an essential element of our project, providing users with an interactive and informative experience, while also allowing us to gather valuable feedback and insights to further improve the performance and usability of our specialized ASR models for the Albanian language.

## 3.3 Design

### 3.3.1 ASR model design

#### *3.3.1.1 Main model Architecture*

We use as a main model the implementation of deepspeech 2 architecture from Keras. This model architecture scored a WER of 6.71, equivalent of an accuracy of

93.29% when trained on LibriSpeech dataset worth of 1000 hours of English speech. The model Architecture is explained as followed:

1. Input Layer: The input layer of the model takes in a spectrogram, which is a visual representation of the frequency content of an audio signal over time. The spectrogram is a 2D matrix where each column represents the spectral content of a small section of the audio signal and each row represents the frequencies that make up that section.

2. Reshape layer: The input spectrogram is reshaped to a 4-dimensional tensor with shape (-1, input_dim, 1, 1). The second and third dimensions represent the frequency and time dimensions, respectively. The last dimension represents the number of channels, which is 1 in this case.

3. Convolutional Layers: The spectrogram is first passed through two convolutional layers, each with a large filter size ([11, 41] and [11, 21], respectively) and a stride of [2, 2] and [1, 2], respectively. These layers are designed to learn high-level features from the input spectrogram by detecting local patterns in the frequency domain. The batch normalization layer is used to ensure that the distribution of activations across the batch remains stable, and the ReLU activation function helps to introduce nonlinearity into the model.

4. Reshaping Layer: After the convolutional layers, the output is reshaped to feed into the recurrent layers. Specifically, the output volume from the convolutional layers is flattened along the temporal dimension, while the frequency and channel dimensions are preserved. This results in a 3D tensor of shape **‘(batch_size, num_timesteps, num_features)‘**.

5. Recurrent Layers: The model has **‘rnn_layers‘** bidirectional GRU layers, each with **‘rnn_units‘** units. These layers are designed to capture the temporal dependencies in the input sequence. The bidirectional architecture allows the model to capture both forward and backward context in the input sequence, which is important for speech recognition. The dropout layer is used to prevent

overfitting by randomly dropping out some of the activations from the previous layer during training.

6. Dense Layers: After the RNN layers, the output is passed through a dense layer with **‘rnn_units * 2‘** units and ReLU activation, followed by a dropout layer. This layer serves to further transform the features learned by the RNN layers into a higher-level representation that can be more easily classified by the final output layer.

7. Output Layer: The final output layer is a dense layer with **‘output_dim + 1‘** units and softmax activation. The extra unit is for the blank symbol, which is used in the CTC loss function. The softmax activation function converts the outputs of the model into a probability distribution over the different output labels, which can be used to determine the most likely transcription of the input audio signal.

8. Optimizer and Loss Function: The model is compiled using the Adam optimizer with a learning rate of **‘1e-4‘** and CTC loss function. The CTC loss function is a popular loss function for speech recognition that accounts for the possibility of insertions and deletions in the predicted label sequence. The goal of training is to minimize this loss function by adjusting the weights and biases of the model during training.

#### *3.3.1.2 Specialised model Architecture for the Albanian language*

In SECTION 2.3.1 and 2.3.8 we saw that ASR systems for languages with rich vowel systems and relying heavily on particles like Finnish and Chinese respectively have benefited from the implementation of more specialized layers (in the form of attention based mechanism) in the model to focus on vowel distinctions as well as focus on relevant particles to improve the overall understanding of the sentence structure. Similarly, in the SECTION 2.3.2 we saw that languages like Georgian can benefit from implementing context-dependent phoneme systems in the their models architecture to be able to better estimate the pronunciation of a given consonant cluster. In order to improve the models architecture specifically for the Albanian language we implement

an additional attention mechanism and a context-dependent phoneme modeling through an addition of the 1D convolutional layer to the existing architecture of DeepSpeech 2.0 as follow:

1. Attention mechanism: An attention layer is applied to the output of the RNN layers to compute a context vector. The context vector is concatenated with the output of the RNN layers.

2. Convolutional layer: A 1D convolutional layer is applied to the concatenated output from the attention layer and RNN layers. The layer has rnn_units * 2 number of filters with a kernel size of 3 and a ReLU activation function.

The following blocks are inserted just after the Recurrent Layers of our main model architecture. These blocks increase the number of total parameters of the model to 32,942,486 compare to the main model 26,650,005. A clear understanding of both models architecture can be seen at Figure 1

### 3.3.2 API design

Our system is designed to provide a FastAPI API for accessing the dataset, models, and other services. The API is designed to be flexible and scalable, with currently 21 endpoints that provide various functionalities. They can be accessed from the API domain "https://api.uneduashqiperine.com/endpoint" The endpoints are described below:

- /video/add : Endpoint used to Upload a video, the audio content of witch can be processed and turned into unlabeled data. The consent form from the speaker for data usage need to be uploaded as well.

- /audio/add : Endpoint used to Upload an audio, the content of witch can be processed and turned into unlabeled data. The consent form from the speaker for data usage need to be uploaded as well.

- /audio/getsa : Endpoint used to get the link of the next unlabeled utterance.

- /audio/get_splice_length : Endpoint used to get the length of the next unlabeled

utterance.

- /audio/get_validation_audio_link_plus : Endpoint used to get the link of the next labeled utterance to be validated. It return the ID, Link, and the current label of that particular utterance.

- /transcribe : Endpoint used to tr ascribe audio using the latest trained model. Takes as input an audio file in WAV format and return the transcribed content for that audio.

- /upload_model : Endpoint used to upload trained models. We use this endpoint to upload the best trained models during a training process.

- /upload_logs : Endpoint used to upload log metadata of a training process of a model such WER score, CER score, Number of epoch, Validation loss and loss.

- /audio/get_validation_audio_link : Endpoint used to return just the link of the next audio utterance to be validated.

- /audio/label/validated/clip_id : Endpoint used to post the label of a utterance that needs to be validated. it takes in the id of the utterance to be validated.

- /audio/label/clip_id : Endpoint used to post the label of an utterance that is never labeled before. it takes in the clip id, label and the validation metric

- /audio/label/v2/clip_id/label_content : Endpoint used to do exactly what /audio/label/clip_id does but can be used only on automation.

- /video/delete/clip_id Delete Clip : Endpoint used to delete a utterance. example in a case when the utterance is very short or long in length, or have background noise.

- /audio/delete/validated/clip_id : Endpoint used to remove a utterance that was labeled initially but can not be validated.

- /clip_ID/ : Endpoint used to get the utterance id of the next utterance to be labeled.

- /sumOfLabeledDuration/ : Endpoint used to get the total duration of labeled not validated utterances in seconds.
- /sumOfLabeledDuration/validated : Endpoint used to get the total duration of validated utterances in seconds.
- /sumOfUnLabeledDuration/ : Endpoint used to get the total duration of unlabeled utterances in seconds.
- /sumOfLabeled/ : Endpoint used to get the total number of labeled not validated utterances in seconds.
- /sumOfUnLabeled/ : Endpoint used to get the total number of unlabeled utterances in seconds.
- /add_training_data/datas : Endpoint used to upload log files generated from Tensorboard.

### 3.3.3 Web application design

The web application is a crucial component of our project, serving as the primary interface for managing the dataset, accessing the API, and interacting with the ASR system. As such, it plays a critical role in facilitating the creation and deployment of specialized ASR models for the Albanian language. The application is designed with a user-friendly interface that provides several features such as labeling and validation of speech data, real-time metrics, and beta transcription capabilities.

### 3.3.4 Labeling

The labeling page of our web application is an essential component of our dataset creation process. A snapshot of the user interface can be seen in Figure 5. Here, users are presented with an unlabelled utterance that they can play and transcribe in Albanian using the keyboard. After submitting the transcription, the page will automatically reload with a new unlabelled utterance to be labeled.

At the bottom of the labeling page, the user can see real-time information about how many utterances have been labeled and their total duration in hours. A snapshot of the user interface can be seen in Figure 7. This information is crucial to keep track of the progress of the labeling process and make sure that all utterances are labeled.

### 3.3.5 Validation

The validation page is another essential component of the dataset creation process. A snapshot of the user interface can be seen in Figure 4. Here, users are presented with labeled but invalidated utterances that they can play and check against the current label to see if it is 100% correct. In addition, the user can trim the audio if it starts or ends with an incomplete word, and delete the utterance if it is too long, too short, or contains background noise that would not be a good fit as speech data for an ASR.

Similar to the labeling page, users can see real-time metrics about the total number of validated utterances and their total duration in hours. This information helps keep track of the validation process and gives users insight information about current dataset metrics.

### 3.3.6 Beta Transcription Page

The beta transcription page is the final component of the web application, and it provides an opportunity for users to test the latest trained ASR model in real-time. A snapshot of the user interface can be seen in Figure 6. Here, users are presented with an audio recorder in their browser that can record audio to be transcribed. After clicking the ”transcribe” button, the ASR model will transcribe the recorded audio into Albanian and display the result on the screen.

This feature is crucial for users who want to test the accuracy of the latest trained model and see how well it performs on their recorded audio. The ability to test the model in real-time also helps to ensure that the model is performing well in a real-world scenario.

### 3.3.7 Dataset design

All dataset information are stored in a database which is made of 5 tables.

#### *3.3.7.1 video_table*

- ”Vid_ID” : Unique id to each video or audio
- ”Vid_NAME” : Name of the video or audio (for example it can be the tittle if it was a YouTube video).
- ”Vid_PATH” : The path of the video where it is saved in the server.
- ”Vid_CATEGORY” : Video or audio category. (for example it can be a news, podcast, audio-book and more)
- ”Vid_UPLOAD_TIME” : Auto assigned time when the video of audio was uploaded.
- ”Vid_TO_MP3_STATUS” : if it was an video it would automatically be converted to an audio so that it can latter be processes further. we update this entity from false to true if the video was converted successfully. we keep track of this so it makes it easy to troubleshot in case of any issue.
- ”Vid_SPLICE_STATUS” : after the video in converted in audio if it was an audio that it would automatically get spliced in short utterances to be used as unlabeled data.
- ”mp3_path” : The path of the audio where it is saved in the server.

#### *3.3.7.2 splice_table*

- ”Sp_ID” : Unique id to each utterance.
- ”Sp_NAME” : The file name of the utterance.

- ”Sp_PATH” : The path of utterance.
- ”Sp_LABEL” : The label of utterance.
- ”Sp_ORIGIN” : The origin of utterance, usually the name if the mp3 file where it was spliced from.
- ”Sp_DURATION” : The duration in seconds of utterance.
- ”Sp_VALIDATION” : The validation metric of utterance, a number varying from 0 to 1.

#### *3.3.7.3 labeled_splice_table*

This table have the exact fields of the splice_table, but just contains utterances that have been labeled. The reason why we are keeping the splices in separate tables is for easy access of use.

#### *3.3.7.4 high_quality_splice_table*

This table have the exact fields of the splice_table as well, but just contains utterances that have been validated.

#### *3.3.7.5 deleted_splice_table*

This table have the exact fields of the splice_table as well, but just contains utterances that have been deleted from users using the web application. We keep track of all the deleted utterances in case of any issue to be able to go back and troubleshoot easel.

## 3.4 Implementation and Integration

### 3.4.1 Programming languages, frameworks, and tools

#### *3.4.1.1 Nginx*

In essence, Nginx is a high-performance web server and reverse proxy server used to handle heavy traffic loads and improve website performance. It is used to serve static and dynamic content and can be configured as a standalone web server or a reverse proxy server. Nginx is known for its speed, scalability, and flexible configuration options, making it a popular choice for serving web applications. In the context of hosting a Python application, Nginx can be used as a reverse proxy to forward requests to a Python web server, providing improved performance and scalability for Python web applications. In this project it is used to serve the API and the Web interface.

#### *3.4.1.2 FastAPI*

FastAPI is a modern, fast (hence the name), web framework for building APIs with Python 3.7+ based on the standard Python type hints. It is built on top of Starlette for the web parts and Pydantic for the data validation and serialization parts. FastAPI provides high performance due to the asynchronous (async) programming model and automatic generation of OpenAPI and JSON Schema documentation. It is designed to be easy to use, fast to develop with, and easy to maintain, making it a popular choice for building modern, high-performance web applications and APIs with Python. We use this framework to build the Api of this froject.

#### *3.4.1.3 Tensorboard*

TensorBoard is a web-based visualization tool for machine learning experiments in TensorFlow. It provides a suite of visualization tools to help users understand, debug, and optimize machine learning models. With TensorBoard, users can visualize training data and model performance metrics, such as loss and accuracy, in real-time. It also allows users to visualize the graph of a TensorFlow model, explore how the data flows through the model, and debug issues that may arise during training. TensorBoard also

supports profiling and debugging tools to help optimize the performance of a machine learning model. Overall, TensorBoard is a powerful tool for analyzing and improving the performance of TensorFlow models, making it an essential part of the machine learning development process.

#### *3.4.1.4 Tensorflow and Keras*

TensorFlow is an open-source machine learning framework that provides a set of tools for building and training machine learning models. It is designed to be flexible and scalable, allowing developers to build and deploy machine learning models for a wide range of applications. TensorFlow is widely used for tasks such as image classification, natural language processing, and time-series analysis, among others.

Keras is a high-level API for building and training machine learning models that runs on top of TensorFlow. It provides a simplified interface for building complex models, allowing developers to focus on the high-level design of their models rather than the low-level implementation details. Keras is designed to be user-friendly, modular, and extensible, making it easy to build and train complex models quickly.

Together, TensorFlow and Keras provide a powerful machine learning framework for building and training complex models. TensorFlow provides the underlying computational framework, while Keras provides a user-friendly API for building and training models. With TensorFlow and Keras, developers can build and train machine learning models for a wide range of applications, from image and speech recognition to natural language processing and time-series analysis.

#### *3.4.1.5 Flask*

Flask is a popular open-source web framework for building web applications with Python. It is designed to be lightweight and flexible, allowing developers to build web applications quickly and easily. Flask provides a set of tools for handling common web development tasks, such as routing, template rendering, and request handling. It also supports extensions for adding additional functionality to the framework, such

as database integration, user authentication, and RESTful API development. Flask is known for its simplicity and ease of use, making it a popular choice for building small to medium-sized web applications. It is also highly customizable and can be adapted to handle more complex applications as well. Overall, Flask is a versatile and powerful web framework for building web applications with Python.

#### *3.4.1.6 SLURM*

SLURM is an open-source job scheduler and resource manager used in high-performance computing environments. It manages the allocation of computing resources and ensures that jobs are executed efficiently. It is commonly used for training machine learning models on large clusters of computers, enabling efficient use of resources and faster training times.

### 3.4.2 Hosting

#### *3.4.2.1 Domain*

The domain we use for this project, ”www.uneduashqiperine.com” is purchased though the hosting provider ”www.hosting24” for 10.99 pounds per year. This hosting provider allows users that own a domain to manage their DNS and Nameservers from their online platform. We have directed the domain to our server IP Address as well as have created two subdomains including ”www.api.uneduashqiperine.com” and ”www.tensorboard.uneduashqiperine.com”. These subdomains are used to serve the api andpoints and to serve the tensorboard interface respectively.

#### *3.4.2.2 Server*

The server used for this project is a 8 GB / 4 CPUs / 160 GB SSD Disk / 5 TB transfer, virtual machine running Ubuntu 22.04 (LTS) X86. the server is provided from Digita Ocean and it costs 48$ per month. This company provides a 300$ free credit for university students, so we have utilised this free trial for students so far.

### 3.4.3 Tensorboard

The implementation of the Tensorboard is pretty basic. The Tensorboard library visualises log files generated during a training process of a model when used Tensorflow. These log data are stored at a specific folder called logs. We create a specialised function that creates a zip archive with this log files generated during training (it deletes and saves the new archive if one exists with the same name) and post this zip archive to the our /upload_logs endpoint. When we train a model in any remote environment, we call this function at the end of each epoch. This posts the data to the endpoint at the end of each training epoch. On the other end-point end, we get the uploaded zip file, unzip it and save the files (if the files exist with the same name we replace them with new ones as they would carry the old logs as well) to the folder that the hosted Tensorboard is listening. In this way we integrate any training happening in any environment remote or not be integrated real time with our hosted Tensorboard. In this way we can monitor training in real time from our Tensorboard subdomain and on the same time in this way we store the log data of any model safely so we can go back and study them at any time.

### 3.4.4 API

The API of this project was implements using the Fastapi python framework. All the information of the database are stored in a db file. We use SQLite to fetch, delete, insert or create record in the database.

#### *3.4.4.1 /video/add Endpoint*

At the /video/add enpoint we expect a video file as an input together with the name and category. Than we perform the following operation in order: → Using the os python library (used for interacting with the operating system)we create 2 folder with the video name in our main mp3, mp4 and splices directories.
→ Than we save the mp4 video in the folder with the video name we created previously.
→ Than we populate the database table with the video information.
→ Than we convert the mp4 file to mp3 file and save it to the folder with the video name in the mp3 main directory we create previously.

→ Than we update the video_tale Vid_TO_MP3_STATUS entity to True.
→ Than we run the script of pyAudioAnalysis (A Python library for audio feature extraction, classification, segmentation and applications) with the argument as our just coveted mp3 file. This would splice the mp3 file in short clips based in quite areas of the audio varying from 1 to 15 seconds and save them in the the folder with the video name in the splices main directory we create previously.
→ Than we then we populate the splice table with the new information of each splice and update the video table

#### *3.4.4.2 /audio/add/*

Performs exactly what /video/add Endpoint performs, just skips audio conversion of the video to audio as the file is already an audio.

#### *3.4.4.3 /audio/getsa/ Endpoint*

When this endpoint is called, the splice_table in the database is query using SQLite to get the first record in the table. then from the results we strip all the row information and keep only the Sp_PATH that next we return.

#### *3.4.4.4 /audio/get_splice_length/ Endpoint*

We perform exactly what we performed in the /audio/getsa/ endpoint but instead of returning the path this time we return Sp_DURATION

#### *3.4.4.5 /audio/get_validation_audio_link_plus/ Endpoint*

We perform exactly what we performed in the /audio/getsa/ endpoint but instead of query splice_table, we query the labeled_splice_table and return a json object containing the ”Path”: Sp_PATH, ”Label”: Sp_LABEL, ”Id”: Sp_ID

#### *3.4.4.6 /transcribe/ Endpoint*

#### *3.4.4.7 /upload_model/ Endpoint*

This endpoint allows a .h5 model file to be uploaded. the upload model than is saved to the main directory folder called models, where we store all the trained models.

#### *3.4.4.8 /upload_logs/ Endpoint*

This endpoint have been explained in details at SECTION **??**

#### *3.4.4.9 /audio/label/validated/ Endpoint*

This endpoint takes in the id of the utterance to be validated and the label. It performs the following actions: → Updates the labeled_splice_table by setting the validation metrics to 1 for the utterance corresponding with the input id. → Insert in the high_quality_labeled_splice_table the record we updated previously. → Delete the record from the labeled_splice_table In this way we have validated with the new label the utterance and we have transport it from labeled_splice_table to high_quality_labeled_splice_table.

#### *3.4.4.10 /audio/label/ Endpoint*

This endpoint takes in the id of the utterance to be labeled, the label and the validation metric. It performs the following actions: → Updates the splice_table by setting the validation metrics from the input as well as the label for the utterance corresponding with the input id. → Insert in the labeled_splice_table the record we updated previously. → Delete the record from the splice_table In this way we have validated with the new label the utterance and we have transport it from splice_table to labeled_splice_table.

#### *3.4.4.11 /audio/label/v2/ Endpoint*

Performs exactly what /audio/label/ Endpoint performs. We created this endpoint to be used only for automation use only.

#### *3.4.4.12 /video/delete/ Endpoint*

This endpoint takes in the id of the utterance to be deleted. It performs the following actions: → Insert in the deleted_splice_table the record with the id as an input from splice_table. → Delete the record from the splice_table In this way we have validated with the new label the utterance and we have transport it from splice_table to deleted_splice_table.

#### *3.4.4.13 /audio/delete/validated/ Endpoint*

This endpoint takes in the id of the utterance to be deleted. It performs the following actions: → Insert in the deleted_splice_table the record with the id as an input from labeled_splice_table. → Delete the record from the splice_table In this way we have validated with the new label the utterance and we have transport it from labeled_splice_table to deleted_splice_table.

#### *3.4.4.14 /clip_ID/ Endpoint*

This endpoint queries the splice_table to get the first row in the table, and then it return only the id of that record. This endpoint is used to get the id of the utterance to be labeled for the first time.

#### *3.4.4.15 /sumOfUnLabeledDuration/ Endpoint*

#### *3.4.4.16 /sumOfLabeled/ Endpoint*

In this endpoint we query the labeled_splice_table to get the sum of all Sp_DURATION. The returned value is a number in seconds meaning the total number of all labeled utterances in the database, and that is what this endpoint return.

#### *3.4.4.17 /sumOfUnLabeled/ Endpoint*

In this endpoint we query the splice_table to get the sum of all Sp_DURATION. The returned value is a number in seconds meaning the total number of all unlabeled

utterances in the database, and that is the sum o what this endpoint return.

#### *3.4.4.18 /sumOfLabeledDuration/ Endpoint*

This endpoint returns the value of /sumOfLabeled/ endpoint converted from seconds to hour:minutes:seconds.

#### *3.4.4.19 /sumOfLabeledDuration/validated/ Endpoint*

In this endpoint we query the high_quality_labeled_splice_table to get the sum of all Sp_DURATION. The returned value is a number in seconds meaning the total number of all validated utterances in the database. This value is converted from seconds to hour:minutes:seconds and then returned.

### 3.4.5 Web interface

The Web interface for this project was built using HTML, CSS and JavaScript for the front-end, and Flask in Python for the back-end. The interface is divided into two main sections on each page: the functionality section, which allows users to label, validate, or transcribe audio data, and the bottom section, which displays real-time metrics of the dataset. The metrics are rendered as soon as the page is loaded, using Flask to make API requests to retrieve the current metrics and then rendering them in the HTML page.

In the functionality SECTION of the transcribe page, we use a pre-built library for audio recording in JavaScript and customized it to meet our needs. After the recording of the audio by the user is done and the button transcribe is clicked, we wrap the audio in a request sent to our /transcribe endpoint and activate a loading animation to indicate that the audio is being transcribed. As soon as the server responded with the transcription response, we show it at the top of the page. If there was already a transcription from a previous audio, we update it with the new one.

For the label and validate pages, we use a different pre-built audio player that was again customized to fit our needs. We include the feature to trim utterances by adding a

double-range slider. At this stage, we use Flask to make API requests to fetch the link of the audio to be played, as well as to post the labels or delete an utterance. After any of these actions is performed, the page is automatically refreshed to re-render the new audio data in the player.

At the validate page users are presented with already labeled but unvalidated audio files that they listen to and compare with the current label to ensure 100% accuracy. If the label of the utterance is incorrect, the user can edit the label and submit the new label. At the end of the validation process, users can see real-time updates on the number of utterances validated and the total duration of the validated audio data.

Than we serve the Flask application using Nginx to our domain.

### 3.4.6 Automation

#### *3.4.6.1 Auto-transcription using Google's API*

In order to generate a dataset that we could use latter to train our models we created a script that would automatically label data we had already spliced. we did this by using google's Speech to text API which is a paid transcription. Google offer a 300$ free credit to be spent on their API for new users. so we created new profile just for the transcription purpose and labeled 133 hours of speech data. The API returns a confidence level together with the audio transcript so we stored this confidence level as our validation level in our database. We created a loop that runs forever where we call our API to get the audio to be validated, we get that audio and send it to google's API for transcription, and the returned label and confidence we post it to our back end, perform just what a user is supposed to perform on our we interface, just this time the labels would not be accurate always as google speech to text model performance for the Albanian language is not great.

#### *3.4.6.2 YouTube playlists auto splicing*

Inserting videos one by one for splicing in our API would take an extensive amount of time. Instead we created a script that reads a txt file line by line and expects a

YouTube playlist or video to automatically download it, and upload it to the endpoint /video/add. wen the responds with code 200 meaning the process was successfully (the video got processed, spliced, and inserted in our dataset), we delete the row with the link in the txt file we just processed and read the next one. so what we did is compiled a txt file containing hundreds of mostly Albanian news, podcasts, online lectures playlists. this generated us a total of 388 hours worth of spliced audio speech in Albanian.

#### *3.4.6.3 Dataset Script*

We created a script that takes an input a csv file exported from our database, and generates dataset zip file as it is required on the training process. we created a bash script that firstly creates the metadat.csv wich is the final csv file we use in the dataset, containing the filenames and the labels. the filename and labels are generated from the input csv file. then we read the filelocation recursivly for each file, copy the file, re-sample it to 16000 Hz and place it in a folder called clips. After the re-sampling we zip the metadata.csv file and the clips folder.

### 3.4.7 Dataset generation

After performing the automation mentioned in SECTION 3.4.6.2 and 3.4.6.1 we generated a total of 133 hours of labeled speech audios to date. However the labeling quality was not of a good quality. in order to generate the dataset we need filter out the bad quality labeled splices. We did thet by performing a sql statement on the dataset that would create a copy of the labeled_splice_table (containing the labeled utterances) with records that had a validation accuracy of higher that 0.9, meaning that the confidence that the label is 100% correct it 90%. with this csv file generated we run the bash script we explained in SECTION 3.4.6.3, leaving with a dataset with utterances of a quality of 90% or higher, worth of 42 hours and 15 minutes ready to train models.

### 3.4.8 Training

The ASR models are trained using the widely adopted TensorFlow and Keras frameworks. The numpy and pandas libraries are utilized for data manipulation, while the matplotlib library is used to visually represent the results of the model. To integrate the training process with our API, the requests library is used to post logs and fetch datasets. The Jiwer Python library is used to calculate the Word Error Rate (WER) during testing and validation, enabling us to measure the performance of the model accurately.

To optimize our computational resources, we leverage Kaggle's environment to train our models, which provides free 30-hour-per-week GPU usage to users. Kaggle offers users the option of using one P100 or two T4 GPUs, thereby enabling us to train our models with high computational efficiency. Additionally, we use HPC's running on SLURM provided by the University of Greenwich for our training needs.

The implementation of the training process begins with the installation and importation of the necessary Python libraries mentioned above. Following this, we define a set of functions that we use during the training process, including:

#### *3.4.8.1 upload_data*

This function takes in the path of the model file to be uploaded and uploads the model found in the path by performing a post request using requests library to our /upload_model endpoint.

#### *3.4.8.2 upload_logs*

add training data/ This function deletes the zip archive if one is found with the same name as the current logs and generates a new zip file using the current tensorboard logs. Then uploads the zip file generated by performing a post request using requests library to our /add_training_data/ endpoint.

#### *3.4.8.3 encode_single_sample*

This function takes a wav file and a label as input and returns the preprocessed spectrogram of the audio and the encoded label as a dictionary. The preprocessing steps include reading and decoding the wav file, getting the spectrogram, taking the magnitude, normalizing it, converting the label to lower case, splitting it into individual characters, and mapping each character to a number using a previously defined char_to_num mapping.

#### *3.4.8.4 Dataset processing*

The dataset is made of one folder where all the wav files are located and one csv file separated by " | " called metadata.csv containing the filenames and their corresponding labels. We use Pandas library to import the data from the csv file using the Pandas read_csv function. Than we split the dataset to train and validation, defined as df_train and df_val with a split of 80% and respectively 20%. Next we create a TensorFlow dataset objects for the training and validation datasets, which are used to feed the model with batches of data during training and evaluation.

For the training_dataset, we first create a Dataset object from the file names and labels using tf.data.Dataset.from_tensor_slices(). Then we apply the encode_single_sample function defined in SECTION 3.4.8.3 to each sample in the dataset using the map() function, which encodes the audio file into a spectrogram and computes the CTC loss target. The num_parallel_calls=tf.data.AUTOTUNE argument allows TensorFlow to automatically determine the number of CPU cores to use for paralleling the operations. Next, it calls padded_batch() to group the encoded samples into batches of size batch_size (set to 32 in our case). This function pads each spectrogram to the same length within each batch, which is necessary for efficient training with mini-batch gradient descent. Finally, prefetch() is called to prefetch the next batch of data asynchronously while the current batch is being processed by the GPU, which can speed up training. The same process is repeated for the validation_dataset, except that it is not shuffled as it is used only for evaluation.

#### *3.4.8.5 Alphabet definition*

In this section we start by defining list that includes all the characters of the albanian language, together with numbers from 0 to 9, as well as characters such as space, ”%”, ”-” and ”.”. Then we initialize a StringLookup layer called char_to_num that maps characters to integers. The vocabulary parameter specifies the set of characters to use for the mapping witch is the list we just created, and oov_token is a special token used for out-of-vocabulary characters. Secondly we initialize another StringLookup layer called num_to_char that maps integers back to their original characters. The vocabulary parameter is set to the same vocabulary as char_to_num so that the mapping is bijective, and oov_token is set to an empty string since there should be no out-of-vocabulary integers. Together, these two layers provide a convenient way to map characters to integers and vice versa tha we will use during training.

#### *3.4.8.6 encode_single_sample*

This function takes a wav file and a label as input and returns the preprocessed spectrogram of the audio and the encoded label as a dictionary. The preprocessing steps include reading and decoding the wav file, getting the spectrogram, taking the magnitude, normalizing it, converting the label to lower case, splitting it into individual characters, and mapping each character to a number using a previously defined char_to_num mapping.

#### *3.4.8.7 CTCLoss*

This function implements the CTC (Connectionist Temporal Classification) loss function we will use when training our model. The inputs to the function are the true labels (y_true) and the predicted labels (y_pred). The function first extracts the dimensions of these inputs using TensorFlow functions, and then constructs vectors for input_length and label_length based on these dimensions. These vectors are used as inputs to keras.backend.ctc_batch_cost, which computes the CTC loss between the true and predicted labels.

#### *3.4.8.8 build_model*

This function implements the DeepSpeech2 architecture using TensorFlow and Keras API's layers. The layers are stacked in a specific order to build the model:

- The input layer takes an input with shape (None, input_dim) where input_dim is the number of features in the input spectrogram.
- The input is reshaped to a 2D volume with dimensions (-1, input_dim, 1) using the Reshape layer.
- The reshaped input is passed through two convolutional layers (Conv2D), followed by batch normalization (BatchNormalization) and a rectified linear unit activation function (ReLU). The first convolutional layer has 32 filters with a kernel size of [11, 41] and strides of [2, 2], while the second convolutional layer has 32 filters with a kernel size of [11, 21] and strides of [1, 2].
- The resulting volume is reshaped again to have a shape of (-1, x.shape[-2] * x.shape[-1]) using Reshape layer, which will be fed to the recurrent layers.
- Recurrent layers (GRU) are stacked on top of the convolutional layers to learn temporal dependencies in the sequence of input spectrograms. Bidirectional wrapper is used to process the sequence both forward and backward direction for each layer.
- A Dense layer is added after the recurrent layers to map the learned features to a higher-dimensional space.
- The output layer is a Dense layer with softmax activation that produces a probability distribution over the vocabulary of possible characters.

After building the model architecture, the function compiles the model using an Adam optimizer with a learning rate of 1e-4 and the CTCLoss function we previously defined in SECTION 3.4.8.7. The reasoning of why layers have been stacked in this specific way has been explained in SECTION 3.3.1.1 In the case where we build the model specialised for the Albanian language, we insert the blocks of the Attention mechanism

and Convolutional layers as explained in SECTION 3.3.1.2

#### *3.4.8.9 build_model*

Here we define the function that will decode the predictions made by the model. It takes a batch of predictions generated by the model and decodes them into text. It uses a greedy search approach to find the most probable sequence of characters for each prediction. First, it creates an array input_len of the same length as the input batch, where each element is the number of timesteps (or frames) in the prediction. This is necessary for the CTC decode function to work correctly. Then, it uses the ctc_decode function from the Keras backend to decode the predictions. This function takes the predictions, the input_length array, and a boolean flag greedy, indicating whether to use greedy search or beam search. In this case, the function uses greedy search. The ctc_decode function returns a tuple containing a list of sparse tensors representing the decoded sequences and a list of sequence lengths. Since we only care about the decoded sequences, we access the first element of the first tuple element (i.e., ctc_decode(pred, input_length=input_len, greedy=True)[0][0]). Finally, the function iterates over each decoded sequence, converts it from a tensor to a string using the num_to_char defined at SECTION 3.4.8.5, and appends the resulting text to an array output_text. This array contains the decoded text for each prediction in the batch.

#### *3.4.8.10 CallbackEval*

This is actually a class that is used to define all the functions that will bu run during the training process at the end of each epoch. We calculate the WER and perform a prediction a on batch of the validation of the dataset. We insert the predictions, WER, Loss and Validation loss to the Tesorboard log files and we upload logs to our server by calling the function upload_logs defined in SECTION 3.4.8.2. We save and upload the model file with the current validation loss using the function upload_data defined in SECTION 3.4.8.1

#### *3.4.8.11 Final training*

Now we start training by calling the .fit function from TensorFlow using the train and validation dataset defined in SECTION 3.4.8.4, set the number of epoch (for our best dataset 50 its the best value as if the number is bigger model start to over-fit) and callbacks to the functions defined in SECTION 3.4.8.10

### 3.4.9 Implementation Challenges

Training machine learning models, especially ASR ones, requires an extremely high amount of processing power especially when the dataset is of the size of ours (over 42 hours). one of the biggest challenges of the implementation was the utilising and finding training resources. The university of Greenwich gave us access to their HPC, where we utilised over 8 GPUs on our testing process where we had to train multiple models with different architectures and parameters. Using the university's HPC was a challenge on its own as they run on SLURM system and not much documentation is available to train models using the methodology we had build. still even though very time consuming we managed to prepare script to train our models efficiently, and that really speed up the testing iteration process.

## 3.5 Testing

### 3.5.1 Api

The development of the API was a gradual process, implemented incrementally based on the feature requirements of the ASR system. While certain API operations involve simple tasks, such as sending or receiving text data, there are other operations that are more complex in nature. For example, when receiving large files like trained ASR models, the server configuration had to be adjusted to handle files of such size, as the default settings were inadequate. Through appropriate configuration changes, the server could now accept files of up to 2GB.

Furthermore, endpoints such as "/video/add" involve significant computations that begin as soon as the video file is received, including video conversion, audio splicing,

and database population. However, when processing very long videos, the server may return a timeout error, necessitating an adjustment of the timeout limit for that specific endpoint. Despite these challenges, the API has demonstrated strong performance overall, even during automation's where we made more than 110 requests per minute on different endpoint. this is because of the FastApi architecture design and their very well documentation support.

### 3.5.2 Web Interface

Similar to the API, the development and testing of the web interface were executed incrementally. The server hosting the interface does not solely serve the web content, but also hosts the API and Tensorboard subdomains, which sometimes results in a slight reduction in rendering speed when a large number of users access the interface simultaneously. However, the web interface functionalities have undergone multiple rounds of testing, and any errors identified have been promptly addressed and resolved. To improve the rendering speed, the option to purchase a server with better specifications or upgrade the current server could be explored. This would greatly enhance the performance of the web interface and ensure a seamless user experience. Overall, the web interface has been constructed and tested diligently to meet the requirements of the ASR system, and efforts to optimize its performance would further enhance its functionality.

### 3.5.3 ASR model

Testing the ASR model is a critical and final step in completing the project. Several datasets were created to test the architecture of the main Deepspeech model, as well as other architectures specifically designed to improve accuracy for the Albanian language. Due to word limit constraints, it is not possible to present all test scenarios conducted, way more informations about different models and architectures can be found on our web interface, "www.tensorboard.uneduashqiperine.com". Instead, this report will focus on the results of the main Deepspeech architecture trained on a dataset of 42 hours and 15 minutes. This dataset was generated by filtering labeled splices from the database with a validation score of over 0.9, indicating a label accuracy of 90% or

higher. While not perfect, this dataset provided enough data to measure the performance of the models effectively. Both the Deepspeech architecture and the model designed for the Albanian language were trained on the same dataset for 50 epochs, with a batch size of 32 and a dataset split of 80:20 for training and validation, respectively. The training of these models was conducted on a SLURM machine, utilizing four GPUs simultaneously. The results of the both models are presented clearly in the figures 3 and 2. Looking at the validation loss at figure 3 we can see that the level of the validation loss starts decreasing faster compare to the normal deep speech architecture, where after the 3rd epoch it always had a lower validation loss (our model architecture). furthermore, looking at the WER at figure 2 we can see that the model started to generalise to new examples much faster. for example at epocs 32, our model WER is 0.537 while the deepspeech model have a WER of 0.576. At this epoch our model beats the Deepspeech by 3.9%.

### 3.6 Product Evaluation

The primary objective of this project was to develop a bespoke Albanian speech-to-text transcriber with an accuracy rate of at least 45%. By refining the existing architecture of the Deepspeech model to suit the unique characteristics of the Albanian language, we achieved an accuracy rate of 46.3%, thereby accomplishing our goal. In addition to building the model, we created a platform that has the potential to facilitate the development of ASR datasets and language corpuses for Albanian. We employed state-of-the-art technologies and popular, well-documented frameworks, which will enable other contributors to participate with ease.

However, despite the considerable effort we have invested in this project, it is vast in scale and requires the involvement of multiple individuals. While we have undertaken much more work than we have documented in this report, there are still numerous areas that require improvement. These include the establishment of a terms and conditions policy and a privacy policy document that includes all the information we store about users visiting the web interface. These policies are critical for the project's longevity.

Moreover, we must devise a better system to approach speakers and acquire their consent to use their content.

Notably, a significant limitation of this project is that while we have assembled a vast dataset for the language and established an automated mechanism to expand it in the future, we cannot make it publicly accessible. This is because it contains information that we may not be permitted to publish. It is important to note that we have only used publicly available data from YouTube, and our dataset is not public.

Since there is not a single ASR model publicly available specifically for the Albanian language, the publication of our web interface, API and the design of our model with latter the firs established public Albanian speech dataset, we hope the impact would be huge in the development of ASR systems for the Albanian language.

The models we currently can train with the available data, do not achieve accuracy standards of a model that is ready for deployment, but with the effort, work and discoveries we have achieved in this project together with some contribution in the future, hopefully we will have a model that archives accuracy high enough to have a model deployed.

# CHAPTER 4

# CLOSING CHAPTER

## 4.1 Conclusion

In SECTION 3.4.8 we saw that by implementing additional mechanisms such as Attention mechanism and context-dependent phoneme modeling to a regular Deepspeech architecture, we would increase the accuracy rate for the Albanian language Automatic Speech Recognition (ASR) system as explained in SECTION 3.5.3. however we have shown proof that these mechanisms can further improve the accuracy of Deepspeech architecture, the accuracy of 46.3% is not of a level that can be of any use in real world. This is because the scarcity of the dataset availability for the language. We built a web interface that serves the purpose of speech data labeling and validation to create datasets for ASR systems. This web interface is integrated with an API we also created to automate and allow other systems to interact with the functionalities our systems provides, including real time transcription. This project is planned to be open-sourced so that other users can contribute to further improve the accuracy of the ASR. Future improvements have also been discussed in SECTION 3.6

## 4.2 Recommendation

In this project we have studied in depth the characteristics of the Albanian language including its phonology, morphosyntax, and lexicon, and we have discovered that there are other languages with well established ASR models that have some characteristics in common with the Albanian one. when researched how these well established ASR's have overcome the complication these characteristics manifest to the system, we find that some of these approaches used by them can be employed to systems that can than

be specialised to the Albanian language. This is an area of research that would discover new model architectures specialised to the Albanian Language.

Another area to be researched that we have tested and have seen a slight improvement is transfer learning. We have already explored this techniques by using the weights of pre-trained models with high accuracy on other languages on models that are build for the Albanian language alphabet and than latter fine-tuned using Albanian speech data.

The data availability is the number one factor affecting the performance of an ASR model. Hence the growth of a dataset for the Albanian language is crucial. this would be achieved through crowd-sourcing on our web interface but would be require a lot of labour. Another approach that we have already attempted is through private companies that have access to large amount of data, for example a company called audibooks.al have access to over 4000 hours of audio books. we already are in discussion presently to have access to these data. Such resources would fasten the dataset creation process.

# REFERENCES


[1] Jiahua Xu, Kaveen Matta, Shaiful Islam, and Andreas Nürnberger. German Speech Recognition System using DeepSpeech. In *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, pages 102–106. Association for Computing Machinery, New York, NY, USA, February 2021.

[2] Wenping Hu, Yao Qian, and Frank K. Soong. A DNN-based acoustic modeling of tonal language and its application to Mandarin pronunciation training. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3206–3210, May 2014. ISSN: 2379-190X.

[3] Tina Raissi, Eugen Beck, Ralf Schlüter, and Hermann Ney. Context-Dependent Acoustic Modeling without Explicit Phone Clustering. In *Interspeech 2020*, pages 4377–4381, October 2020. arXiv:2005.07578 [cs, eess].

[4] Hemant K. Kathania, S. Shahnawazuddin, Nagaraj Adiga, and Waquar Ahmad. Role of Prosodic Features on Children's Speech Recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5519–5523, April 2018. ISSN: 2379-190X.

[5] Hainan Xu, Shuoyang Ding, and Shinji Watanabe. Improving End-to-end Speech Recognition with Pronunciation-assisted Sub-word Modeling, February 2019. arXiv:1811.04284 [cs].

[6] Ronan Collobert, Awni Hannun, and Gabriel Synnaeve. Word-level Speech Recognition with a Letter to Word Encoder.

[7] Takaaki Hori, Jaejin Cho, and Shinji Watanabe. End-to-end Speech Recognition

with Word-based RNN Language Models, August 2018. arXiv:1808.02608 [cs].

[8] Jinchuan Tian, Jianwei Yu, Chao Weng, Yuexian Zou, and Dong Yu. Improving Mandarin End-to-End Speech Recognition with Word N-gram Language Model. *IEEE Signal Process. Lett.*, 29:812–816, 2022. arXiv:2201.01995 [cs, eess].

[9] Wenxin Hou, Han Zhu, Yidong Wang, Jindong Wang, Tao Qin, Renjun Xu, and Takahiro Shinozaki. Exploiting Adapters for Cross-lingual Low-resource Speech Recognition, December 2021. arXiv:2105.11905 [cs, eess].

[10] Nay San, Martijn Bartelds, Blaine Billings, Ella de Falco, Hendi Feriza, Johan Safri, Wawan Sahrozi, Ben Foley, Bradley McDonnell, and Dan Jurafsky. Leveraging supplementary text data to kick-start automatic speech recognition system development with limited transcriptions, February 2023. arXiv:2302.04975 [cs].

[11] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep Speech: Scaling up end-to-end speech recognition, December 2014. arXiv:1412.5567 [cs].

[12] Orestis Papakyriakopoulos and Alice Xiang. Considerations for ethical speech recognition datasets. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, WSDM '23, page 1287–1288, New York, NY, USA, 2023. Association for Computing Machinery.

[13] Amarildo Rista and Arbana Kadriu. CASR: A Corpus for Albanian Speech Recognition. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 438–441, September 2021. ISSN: 2623-8764.

[14] Alan W Black. CMU Wilderness Multilingual Speech Dataset. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5971–5975, May 2019. ISSN: 2379-190X.

[15] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey,

and Ilya Sutskever. Robust Speech Recognition via Large-Scale Weak Supervision.

[16] Ramiz Kastrati, Mentor Hamiti, and Lejla Abazi. The opportunity of using eSpeak as text-to-speech synthesizer for Albanian language. In *Proceedings of the 15th International Conference on Computer Systems and Technologies*, CompSysTech '14, pages 179–186, New York, NY, USA, June 2014. Association for Computing Machinery.

[17] Deepthi Karkada and Vikram A. Saletore. Training Speech Recognition Models on HPC Infrastructure. In *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*, pages 124–132, November 2018.

[18] Wav2vec: State-of-the-art speech recognition through self-supervision.

[19] Zhiyun Lu, Liangliang Cao, Yu Zhang, Chung-Cheng Chiu, and James Fan. Speech Sentiment Analysis via Pre-Trained Features from End-to-End ASR Models. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7149–7153, May 2020. ISSN: 2379-190X.

[20] Rahhal Errattahi, Salil Deena, Asmaa El Hannani, Hassan Ouahmane, and Thomas Hain. Improving ASR Error Detection with RNNLM Adaptation. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 190–196, December 2018.

[21] Isham Mohamed and Uthayasanker Thayasivam. Low Resource Multi-ASR Speech Command Recognition. In *2022 Moratuwa Engineering Research Conference (MERCon)*, pages 1–6, July 2022. ISSN: 2691-364X.

[22] Jing Zhao, Guixin Shi, Guan-Bo Wang, and Wei-Qiang Zhang. Automatic Speech Recognition for Low-Resource Languages: The Thuee Systems for the IARPA Openasr20 Evaluation. In *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 335–341, December 2021.

[23] Satwinder Singh, Ruili Wang, and Feng Hou. Improved Meta Learning for Low Resource Speech Recognition. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4798–4802, May 2022. ISSN: 2379-190X.

[24] Weizhe Wang, Xiaodong Yang, and Hongwu Yang. End-to-End Low-Resource Speech Recognition with a Deep CNN-LSTM Encoder. In *2020 IEEE 3rd International Conference on Information Communication and Signal Processing (ICICSP)*, pages 158–162, September 2020.

[25] Jennifer Drexler and James Glass. Combining End-to-End and Adversarial Training for Low-Resource Speech Recognition. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 361–368, December 2018.

[26] Reza Sahraeian and Dirk Van Compernolle. A study of rank-constrained multilingual DNNS for low-resource ASR. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5420–5424, March 2016. ISSN: 2379-190X.

[27] Bao Thai, Robert Jimerson, Dominic Arcoraci, Emily Prud'hommeaux, and Raymond Ptucha. Synthetic Data Augmentation for Improving Low-Resource ASR. In *2019 IEEE Western New York Image and Signal Processing Workshop (WNYISPW)*, pages 1–9, October 2019. ISSN: 2471-9242.

[28] Anoop C S, Prathosh A P, and A G Ramakrishnan. Unsupervised Domain Adaptation Schemes for Building ASR in Low-Resource Languages. In *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 342–349, December 2021.

[29] Alex Graves, Santiago Fernandez, Faustino Gomez, and Jurgen Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks.

[30] Julius Kunze, Louis Kirsch, Ilia Kurenkov, Andreas Krug, Jens Johannsmeier,

and Sebastian Stober. Transfer Learning for Speech Recognition on a Budget. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 168–177, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[31] Gundeep Singh, Sahil Sharma, Vijay Kumar, Manjit Kaur, Mohammed Baz, and Mehedi Masud. Spoken Language Identification Using Deep Learning. *Computational Intelligence and Neuroscience*, 2021:1–12, September 2021.

[32] Evis Trandafili, Alba Haveriku, and Antea Bendo. Employing a Seq2Seq Model for Spelling Correction in Albanian Language. In *2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, September 2022. ISSN: 1847-358X.

[33] Jörgen Valk and Tanel Alumäe. VoxLingua107: a Dataset for Spoken Language Recognition, November 2020. arXiv:2011.12998 [eess].

[34] Mirco Ravanelli, Titouan Parcollet, and Yoshua Bengio. The Pytorch-kaldi Speech Recognition Toolkit. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6465–6469, May 2019. ISSN: 2379-190X.

[35] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas˘ Burget, Ond˘rej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlıc˘ek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi Speech Recognition Toolkit.

[36] Oleksii Kuchaiev, Boris Ginsburg, Igor Gitman, Vitaly Lavrukhin, Jason Li, Huyen Nguyen, Carl Case, and Paulius Micikevicius. Mixed-Precision Training for NLP and Speech Recognition with OpenSeq2Seq, November 2018. arXiv:1805.10387 [cs].

[37] Theodoros Giannakopoulos. A Python library for audio feature extraction, classification, segmentation and applications, April 2023. original-date: 2014-08-27T12:43:13Z.

[38] Genc Struga. Why Native Albanian Speaker have an Easier Approach toward Acquisition of other Languages? *Journal of Neurology & Stroke*, 4, April 2016.

[39] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-Based Models for Speech Recognition. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[40] Aku Rouhe, Astrid Van Camp, Mittul Singh, Hugo Van Hamme, and Mikko Kurimo. An Equal Data Setting for Attention-Based Encoder-Decoder and HMM/DNN Models: A Case Study in Finnish ASR. In Alexey Karpov and Rodmonga Potapova, editors, *Speech and Computer*, Lecture Notes in Computer Science, pages 602–613, Cham, 2021. Springer International Publishing.

[41] Feature learning for efficient ASR-free keyword spotting in low-resource languages | Elsevier Enhanced Reader.

[42] Andrei Andrusenko, Aleksandr Laptev, and Ivan Medennikov. Exploration of End-to-End ASR for OpenSTT – Russian Open Speech-to-Text Dataset. In Alexey Karpov and Rodmonga Potapova, editors, *Speech and Computer*, Lecture Notes in Computer Science, pages 35–44, Cham, 2020. Springer International Publishing.

[43] Tiezheng Yu, Rita Frieske, Peng Xu, Samuel Cahyawijaya, Cheuk Tung Yiu, Holy Lovenia, Wenliang Dai, Elham J. Barezi, Qifeng Chen, Xiaojuan Ma, Bertram Shi, and Pascale Fung. Automatic Speech Recognition Datasets in Cantonese: A Survey and New Dataset. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6487–6494, Marseille, France, June 2022. European Language Resources Association.

[44] Zoey Liu, Justin Spence, and Emily Prud'Hommeaux. Studying the impact of

language model size for low-resource ASR. In *Proceedings of the Sixth Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 77–83, Remote, March 2023. Association for Computational Linguistics.

[45] Yogesh Kumar, Apeksha Koul, and Chamkaur Singh. A deep learning approaches in text-to-speech system: a systematic review and recent research perspective. *Multimed Tools Appl*, 82(10):15171–15197, April 2023.

[46] Andrew L. Maas, Peng Qi, Ziang Xie, Awni Y. Hannun, Christopher T. Lengerich, Daniel Jurafsky, and Andrew Y. Ng. Building DNN Acoustic Models for Large Vocabulary Speech Recognition, January 2015. arXiv:1406.7806 [cs, stat].

[47] Anja Virkkunen, Aku Rouhe, Nhan Phan, and Mikko Kurimo. Finnish parliament ASR corpus. *Lang Resources & Evaluation*, March 2023.

[48] Yanan Jia. A Deep Learning System for Domain-specific speech Recognition, March 2023. arXiv:2303.10510 [cs, eess].

[49] Yanan Jia. A Deep Learning System for Sentiment Analysis of Service Calls. In *Proceedings of the 3rd Workshop on e-Commerce and NLP*, pages 24–34, Seattle, WA, USA, July 2020. Association for Computational Linguistics.

[50] Hao Yang, Min Zhang, Shimin Tao, Miaomiao Ma, and Ying Qin. Chinese ASR and NER Improvement Based on Whisper Fine-Tuning. In *2023 25th International Conference on Advanced Communication Technology (ICACT)*, pages 213–217, February 2023. ISSN: 1738-9445.

[51] Apoorv Vyas, Pranay Dighe, Sibo Tong, and Hervé Bourlard. Analyzing Uncertainties in Speech Recognition Using Dropout. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6730–6734, May 2019. ISSN: 2379-190X.

[52] Iker Luengo, Eva Navas, Inmaculada Hernáez, and Jon Sánchez. Automatic emotion recognition using prosodic parameters. In *Interspeech 2005*, pages 493–496. ISCA, September 2005.