VELS
INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)
(Deemed to be University Estd. u/s 3 of the UGC Act, 1956)
PALLAVARAM - CHENNAI
ACCREDITED BY NAAC WITH 'A' GRADE
Marching Beyond 30 Years Successfully
INSTITUTION WITH UGC 12B STATUS

PROFANITY FILTERING IN AUDIO

A Project Work Report

Submitted for the Partial fulfillment for the award of degree of

MASTER OF SCIENCE

IN

DATA SCIENCE AND BUSINESS ANALITYCS

By

NAME : **DEEPAN CHAKARAVARTH J**

REGISTER No. : **22235106**

Under the guidance of

Dr. DEVI

Assistant Professor

School of Computing Sciences

Department of Information Technology (BCA & IT)

INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)
(Deemed to be University Estd. u/s 3 of the UGC Act, 1956)
PALLAVARAM - CHENNAI
ACCREDITED BY NAAC WITH 'A' GRADE
Marching Beyond 30 Years Successfully
INSTITUTION WITH UGC 12B STATUS

CERTIFICATE

This is to certify that the project titled "**PROFANITY FILTERING IN AUDIO"** is the original record work done by **DEEPAN CHAKARAVARTHI J (22235106)**, under my guidance and supervision for the partial fulfillment of award of degree in Master of Science in Data science and business analytics (M.sc ds&ba), as per the syllabus prescribed by the Vels Institute of Science , Technology and Advanced Studies(VISTAS).

GUIDE HEAD OF THE DEPARTMENT

Submitted for the Viva-Voce examination held on _____ at Vels Institute of Science, Technology and Advanced Studies(VISTAS).

Place: Chennai EXAMINERS

Date: 1.

INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)
(Deemed to be University Estd. u/s 3 of the UGC Act, 1956)
PALLAVARAM - CHENNAI
ACCREDITED BY NAAC WITH 'A' GRADE
Marching Beyond 30 Years Successfully
INSTITUTION WITH UGC 12B STATUS

DECLARATION

I, **DEEPAN CHAKARAVARTHI J (REGISTER No. 22235106)**, declares that the project entitled "**PROFANITY FILTERING IN AUDIO**" submitted by me during the period from **2022-2024** under the guidance **Dr. DEVI** , and has not formed the basis for the award of any degree diploma, associate-ship, fellowship, titles in this or any other University or other similar institutions of higher learning.

Place: Chennai

Signature of the Student

Date:

DEEPAN CHAKARAVARTHI J

INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)
(Deemed to be University Estd. u/s 3 of the UGC Act, 1956)
PALLAVARAM - CHENNAI
ACCREDITED BY NAAC WITH 'A' GRADE
Marching Beyond 30 Years Successfully
INSTITUTION WITH UGC 12B STATUS

BONAFIDE CERTIFICATE

I certify that the project titled "**PROFANITY FILTERING IN AUDIO"** for the Master of Science in Data science and business analytics(M.sc ds&ba) done by **DEEPAN CHAKARAVARTHIJ (REGISTER No. 22235106)** is the project work carried out by her during the period from **2022-2024** under my guidance and supervision and that this work has not formed the basis for the award of any degree, diploma, associate-ship, fellowship, titles in this or any other University or other similar institutions of higher learning. He/She fulfills the eligibility criteria for submission of this project as per rules and regulations of the University.

Place: Chennai

Date: Supervisor

ACKNOWLEDGEMENT

I thank the lord almighty for his boundless blessings showered on me. If awards are considered as symbols of approval and token acknowledgement, then the words play a heralding role of expressing my gratitude to all who have helped me directly and indirectly during my project work.

I express my sincere gratitude to **Dr. Ishari K Ganesh**, Founder Chancellor, Pro Chancellor - Planning & Development, **Dr. A. Jothi Murugan**, Pro Chancellor – Academics **Dr. Arthi K. Ganesh**, Vice President - Vels Group Of Institutions **Dr. Preethaa Ganesh**, Vice Chancellor **Dr. S. Sriman Narayanan** And Pro- Vice Chancellor Dr. **M. Bhaskaran** And Registrar **Dr. P. Saravanan**, Vels Institute Of Science And Technology (Vistas) for providing me an opportunity and motivation to do this research.

I sincerely thank **Dr. P. Sujatha** , Professor and **Head of the Department of Information Technology**, VISTAS for her continued support and motivation.

I profoundly thank my project supervisor **Dr. DEVI** for her total encouragement, inspiration and advice from the beginning till the end of this project / dissertation. Her wise academic advice and ideas have played an extremely vital role in the work presented in this project work. Without her support, this research work would not have been possible.

I am grateful to the faculties of the department of Information Technology, VISTAS for their assistance for the successful completion of this dissertation.

I thank VISTAS librarians for providing me access to the useful textual materials available in VISTAS which supported my research genre.

I am grateful to my friends and family for rendering me their recommendations and moral support.

Name of the student / scholar

DEEPAN CHAKARAVARTHI J

ABSTRACT

The work consists in the manual designing of an automated system for removing offensive language in audio recordings. The system is a use of the speech-to-text to annotate offensive words from audio. When profanity is noticed, the system deletes the segments of audio where the swearing happens and replaces them with one second of silence before and after the interval during which a person has pronounced the offensive word. As well as that, a short sound of a beep is inserted in the intervals where profanities are censored to bring the listener's attention to the censored content. The system implementation is based on the utilization of the following: machine learning models for voice recognition and profanity detection along with audio signal processing to handle audio data. The system focuses on profanity censorship in audio recordings by creating a dependable and automatic method which is essential for a number of tasks, such as content moderation in media platforms, educational resources, and communication systems.

Contents

1 Introduction

1.1 Objectives

Throughout the ages, media has been used by human societies to report on environmental hazards, express views and facts, transmit knowledge and cultural heritage as well as for entertainment. These various forms of media are audio, video, text or all of them combined play a key role in shaping human destiny. Recent developments in deep learning algorithms especially in computer vision, audio analysis and natural language processing have attracted significant attention due to the growing application of multimedia technologies.

Within audio analysis as a field there have emerged various sub-domains such as automatic speech recognition (ASR), digital signal processing (DSP), music classification systems and speech emotion detection among others which are still developing. This is an indication of how important media has become for humanity.

An article by bigdataanalyticsnews.com shows that the expected volume of data created worldwide will be 147 Zettabytes (ZB) by 2024 compared to 9ZB in 2013. By the end of 2025 this data is expected to reach around 181 ZBs.. These huge datasets encompass different types such as text, audio and video just to mention but a few. The volume of data produced is mind-boggling with about five hundred million tweets every day and YouTube uploading approximately seven hundred twenty thousand hours' worth of videos daily.

Big data, often referred to as an explosive amount of information, has turned out to be a driving influence behind the global economy . However, a considerable part of this constitutes problematic materials such as fake news, hate speech, expletives, inappropriate words, adult content and racial slurs among others. Tackling these problems calls for innovative strategies and techniques which may include deep learning algorithms in creating effective content moderation systems that keep online platforms respectful, safe and favorable for positive interactions.

Key points

- Build an intentional speech-to-text transcription application which captures the acoustics of any audio to text.

- Design the detection installation which can pick out and highlight the profanities within the transcribed text.

- Develop machine learning programs for unobscuring insulting words from the audio file, yet maintaining the general context.

- Use a way to put a pair of 1-second silences respectively ahead of and at the end of any portion of profanity before it is removed so that its context is given the proper placement.

- Incorporate a brief tone, model it on a short beep sound, as it will be used as interval warnings of censored words.

- Efficiently using the system to process audio of large volumes and in real-time is a necessity.

- Extensively testing the system before deployment and ensuring that there are not any backend spillovers is necessary.

- Prepare the implementation and usage instructions for the system. Implement the documents on profanity censorship system.

- Investigate the quality of the system employing user feedback and testing with different audio sources , and profanity forms.

- To pick the further add-on to system, for example, implementing only certain words' block only or to team the audio technical processing machines up.

However, our long-term priority is to expand the application of the aforementioned research results in that direction, creating an executable program or a browser extension that would be able to filter out swearing words in any audio speech file of any size or quality. This gives users a powerful tool with which to achieve effective content moderation in the digital world and make it more reliable and decent.

1.2 Motivation for the work

The dangerous trend is the widespread use of raw language online, which includes a revolution of profanity and vulgarity, leading to specific risks for immature and inexperienced users. Language and speech like swearing, hate, and a foul one is something for those who create content, the media where they share it, and those who broadcast it to have issues with censoring tools for their content to be accurate. However, the majority of inaccurate content is still widely available online, so these measures are being taken.The human block of text programs for audio contents is a laborious, time-consuming, and exorbitant process that requires such a hefty investment in terms of manpower. Manual censorship is also susceptible to mistakes coming from causes that prevent a person from making a clear judgment.

Consequently, the requirement for machine-learning-based automated models for audio or speech content suppression has increased. The purpose of this project is to solve the difficulties of accessing content, and with this, we will build a deep learning model capable of recognizing and suppressing speech content that is negative from audio files. Through sophisticated technologies applied, this initiative plans to perfect censorship processes and let no unworthy words reach users. Online space will be like a zone without a hostile atmosphere for all users.

The motivation for undertaking this project stems from several key factors:

Growing Concerns Over Online Safety:

Nowadays, confronted by the fact that online platforms and social media are spreading so rapidly, is inevitable the emergence of aprehension in terms of the user' safety and well-being as specific offensive language or inappropriate content may be involved. This effort seeks to tackle the issues of fear and censorship by building a top-notch profanity detection mechanism that, in turn, serves to produce a safer digital space to engage in.

Impact on User Experience:

Offensive words, along with inappropriate content negatively affect user experience on the online platforms, as they lead to distress and aversion. Due to this, users have the risk of feeling uncomfortable and de-motivated. A filer of profanities is put in place by the enactment of the project

as it strives to mature the user experience by providing a positive and respectful online environment which promotes interactive communication.

Support for Content Moderation:

Content moderate would appear as a formidable responsibilities to online platforms as number of the user-generated content is tremendous. Automation of profanity detection in speech content could mark a huge success for content moderation because the platform would be able to prune it more ambitiously and make sure that the quality of the content has been aggravated adequately.

Relevance of Deep Learning Techniques:

Deep neural network algorithms have presented phenomenal achievements, resulting in breakthroughs in domains as natural language processing and sound analysis. Using the given approaches, the profanity filtering on speech content serves as an impetus for delving deep learning into the reality of addressing the major issues and widespread moderation in the content world.

Alignment with Ethical and Social Responsibility:

With the inevitability of tech becoming ever more influential in the human relations and the way we communicate, there is an increased awareness among those who develop the technologies and those who own the platforms of the ethical and social responsibilities to keep in mind. This project works to accomplish its goals through building tools and systems to enhance respectful communication and prevent any harm online; therefore, it complies with those moral considerations by eventually creating a healthier and safer virtual environment.

Potential for Broad Impact:

The project's influence to go beyond the individual online forums play a grander role of instigating social acceptance while creating safer online environment for everyone. Through emphasizing the rationality aspects and contributing to the eradication of unacceptable language transmission, the project can therefore have a positive impact on the virtual community.

As a whole, motivating reasons for this project are found in its capabilities of tackling serious issues about online safety, facilitating user experience, supporting content monitoring, utilizing up-to-date technologies, ensuring ethical concerns, and achieving a positive online culture.

1.3 Significance of the work

This project is of importance since it contributes to a safer, more respectful online environment. In today's digital era where media consumption is pervasive, the prevalence of offensive language, hate speech and inappropriate content are major challenges to online communities. The objective of this project is thus to address these challenges through developing a robust profanity detection system using deep learning algorithms that would enhance positive online interactions.

Improving User Experience:

By filtering out abusive or ill-suited language from voice input, the user experience can be optimized on different platforms. This creates an enabling environment for audiences to access content without being exposed to cruel words; hence promoting a more encouraging and egalitarian virtual community.

Promoting Respectful Communication:

Users are encouraged to communicate respectfully and responsibly by implementing a profanity filter. By discouraging people from using vulgar language, this initiative promotes polite conversation among users in social networks.

Supporting Content Moderation: Online media platforms face the difficulty of moderating the huge amount of user generated content that goes online. The project aides in content moderation efforts as it automatically detects swearing in speech content that allows the platform to keep community guidelines and standards more effectively.

Empowering Platform Owners:

The use of profanity detection systems by platforms gives platform owners and administrators a better mechanism to ensure that the policies which they have set up are being followed. They are capable of detecting and removing the offensive content in time, thus, they maintain the reputation of their platform and build up users' trust.

Adapting to Evolving Language Trends:

Language goes through constant changes, and what is viewed as inoffensive today may have a chance of being seen as offensive tomorrow. The project's deep learning enables learning for the new forms of profanity and inappropriate language, which makes the project always up-to-date and brings in best outcomes.

Contributing to Technological Advancements:

The project how to unearth the hidden potential the deep learning algorithms for profanity detection in speech content contributes to progress the natural language processing and audio analysis areas. Despite the broader implications of these advancements beyond profanity blocking, the speech recognition systems get better and multimedia content analysis capabilities become enhanced.

Primarily, the project is important because it can generate a virtual space that is more secure, respectful and hospitable online for the users from all over the world and at the same time, it is an attempt to improve deep learning and content moderation technologies.

1.4 Background of the study

There is an important issue of profanity and offensive language being present in digital material now, i.e., speech audio files, which is one of the most vulnerable topics in the online environment. People, in particular those who are below the age of or have a low tolerance, might even develop mental distress because of the same. Although internet content providers make an effort to ensure the observance of community rules and research the tools to censor, there is still a substantial amount of indecent information circulating freely on the internet.

The conventional objectives of self-censorship of audio content are inefficient, slow, and expensive due to the use of human moderators, who do the work of reviewing and editing what is sent in. In addition to that, human censorship may be due to mistakes that will generate inconsistencies in content moderation activities. On the other hand, people are increasingly asking for automated tools that can successfully identify and filter the profane language in audio files' speech.

The existing research in this field manifests major investigations involving the adoption of speech-to-text transformation methods to detect profanity. On the other hand, the two

approaches may fail to recognize the manner in which swearing is sometimes used in audio content, resulting in inaccuracies in detection.

Through the focus on audio characteristics, we developed an approach that can handle the current methods limitations and improve the profanity detection accuracy and average time of the speech audio files. Besides, we also want to move past just recognition and investigate censoring techniques that involve replacing cursing words throughout audio with no or only slight audio quality loss . As a result, we suggest that additional and improved approaches for moderation, which are more efficient and effective in controlling audio content in digital media, should be implemented. Through our efforts to create cutting-edge deep learning models and algorithms, we strive to supply the entire online world with a safer and more respectful digital space where users can freely express themselves.

1.5 Scope of the study

- Study will only focus on speech audio files. (Video by extracting the audio)
- Focusing on only english language
- his study focus on 835 profanity words
- Classification only focused on positive words(profine)
- Censoring techniques are muting and beeping
- The end product will be censored and quality will not be that high but not much is lost. original file

1.6 Feasibility study

Technical Feasibility: Audio Processing:

The proposed technique, in its implementation, majorly depends on the availability of fast audio processing mechanisms for transcription, detection, and edition of audio segments. For speech-to-text transcription, profanity detection, and audio manipulation purposes, there are currently existing libraries and APIs that are available for usage; therefore, the technical part does not pose much of a problem.

Scalability: The system should be fully able to manage large audio files without any incapacity, which is a must since there is a need to consider processor power and memory usage. Distributed computing and/or cloud-based alternatives would be required for scalability.

Achieving the ultimate goal.

Integration: Correspondingly, integrating approaches like text-to-speech, profanity detection, and audio editing can be difficult, and the success of this integration can depend on the implementation and testing of those tools to produce the desired user experience.

Economic Feasibility:

Cost of Development: The PraudioCensor system creation cost budget includes software development, system integration, testing, and other possible expenses. The presence of open-source libraries and pre-existing models can, thus, minimize the expense involved in the process.

Operational Costs: Total operating expenses include maintenance and updates, as well as the possibility of using a fee-based service outside of the company or cloud computing resources. There could be some costs associated with this system. In this case, it should be evaluated in light of the benefits it provides.

Market Feasibility:

Demand: People are recently looking for content moderation tools for use in different sectors such as media, entertainment, and education.

Audio Censorship: PraudioCensor is one of the best automated solutions for sweeping out "shock words" in audio content.

Competition: Through the creation of these features, customers might perceive that PraudioCensor is distinct from its competitors. Context-aware silent insertion and beep sound integration may contribute to the customer's differentiation.

Legal and Ethical Feasibility:

Compliance: The system should be able to present information in accordance with the legal requirements on data privacy, data protection, and content censorship in relevant country or

state rules. Ethics compliance in the workplace is indispensable since ethical norms and principles of equity and transparency play a major role.

User Consent: It is advisable to inform the users that the organization has enabled the function of censoring the swear words, and they can toggle the feature on or off. Providing consumers with choice and respecting their privacy is of the essence.

2 Proposed methodology

2.1 Challenges and Limitations in Existing Systems

The challenges in profanity detection systems encompass inaccuracies due to diverse languages and contexts, degraded performance with low-quality audio and accents, and issues with scalability and integration complexity. Additionally, user preferences and privacy concerns pose challenges regarding customization options and data security. Addressing these challenges requires advancements in contextual understanding, scalability, integration, and user control while ensuring resource efficiency and data privacy.

Accuracy of Profanity Detection: An existing automatic profanity classification may have inaccuracy, thus leading to false positives or false negatives. Profanity detection is essentially complex thanks to the multiple varieties of languages, dialects, and contextual phrasing.

Contextual Understanding: Profanity is more subject to context, manner, and purpose in many cases. To date, current classification systems often fail to accurately understand the context of speech; therefore, words may be mistakenly classified as offensive.

Audio Quality: Low-quality audio, train noise, and different accents or dialects all degrade the effectiveness of speech-to-text and profanity detection systems. Existing systems may not perform well in dealing with those deviations.

Scalability and Speed: Handling a large amount of sound information in real time or near real time brings the issues of scalability and speed. While current techniques may be stretched to

their limits when processing large amounts of audio data in a timely manner, the emergence of new tools and technologies is expected to solve this issue.

Integration Complexity: Integration of the text-to-speech transcription, foul language recognition, and audio customization functions into a uniform system can be very challenging. The current ones might not be so smoothly connected, which then may result in inefficiency and the possibility of operating the devices.

User Preferences and Control: The existing systems could be lacking in the number of customization options users need to control the company and to adjust parameters like sensitivity to detecting obscenity. Restriction on user control could lead to excessive or insufficient measures of censorship.

Resource Intensiveness: Some systems already in operation may have a substantially high demand for algorithms or need dedicated software for handling audio information quickly. It can therefore deter their accessibility to smaller companies and individuals alike.

Privacy Concerns: Privacy concerns focused on data security and confidentiality are likely to be raised by profanity detection systems that rely solely on cloud-based services. Database users may fear giving out their confidential data to non-protected storage and processing servers on the internet

2.2 Method

The method involves developing a comprehensive system for processing audio recordings, including transcription, profanity detection, segmentation of profane words, and audio manipulation techniques such as silent insertion, beep sound insertion, and audio replacement. The system aims to accurately transcribe audio while identifying and handling profanity appropriately, ensuring both context and rhythm are maintained. Integration and optimization are key to creating an efficient and scalable solution suitable for handling large volumes of audio data. Thorough testing and evaluation are essential to assess the precision of profanity detection, effectiveness of audio manipulation techniques, and overall system performance.

Audio Transcription: Apply a speech-to-text transcription system to change audio recordings into text. Thus, the model should be well-built to provide both correct transcription of the voice words as well as corresponding timestamps for each word.

Profanity Detection: Program a profanity-detection module that can identify profane words in the processed text. This process can be done with the help of natural language processing techniques or using a community model that is trained to detect such expressions.

Profane Word Segmentation: After the curse words are discovered, detach the audio record to specifically get the intervals where the curse words are contained. It involves generating the timestamps of the profane language that are matched with the respective audio clip sections.

Audio Manipulation:

Silent Insertion: Use no profane word during voiceover; insert one second of silence before and after each profaned word to provide context for the censorship. This implies that the listeners are reminded that the music has not started yet at the moment when censorship is used.

Beep Sound Insertion: Cue a short beep in place of profane words for the audience to perceive the explicit words' presence. This sound functions as a visual cue, but it still maintains the rhythm of the audio drama.

Audio Replacement: Accomplish this by extracting the audio portions concerning the explicit words while preserving the remainder of the audio material. Such operations include the output of silence instead of the obscene word and the discharge of the beep sound in the right locations.

Integration and Optimization: Synthesize all the activities to build a comprehensive process. Design the system to be optimal for efficiency and scalability, while it can be done on the fly to take into account huge audio recordings or batch processing scenarios.

Testing and Evaluation: The comprehensive audio test process should employ different recordings with various kinds of profanity. Calculate the precision of swearing detection, the audio forgery techniques, and the whole instrument, respectively.

2.3 Available (ASR) models

An automatic speech recognition (ASR) system is a computing model that receives an individual's voice input and converts it into text format. Its basic method of operation is audio input processing and transcribing to text input. The method is commonly known as neural networks (NN) or deep learning algorithms. ASR systems utilize a range of applications, from voice-manipulated virtual assistants to dictation software and transcription services. Most of the

time, these models have a preprocessing step to extract attributes from the audio signal, followed by an architecture containing a neural network trained on a huge amount of labeled audio-text pairs already existing in order to teach the model how to translate words heard and written down. ASR models, now seen as a key facilitator of real-time speech recognition technology, are vital enablers for humans to interact with computers through spoken language input.

ASR models that are offered today also diverge in architecture of their networks, source data for training, and what kind of features they provide.Some popular ones include:

Google Cloud Speech-to-Text: Started by Google, this model ensures that the verbatim is well done and also allows for different languages and dialects. Its ability to deal with the busy situations and it comes with real-time streaming is worth to mention.

Mozilla DeepSpeech: Mozilla advanced research foundation recently developed a deep learning technique-based open-source automatic speech recognition model. It is famous for its precision and incorporates many other languages. 1. It can be used for either research needs or added to a commercial product.

Microsoft Azure Speech Service: By means of Microsoft's ASR, the user everything needed to transcribe: speaker diarization and real-time translation. It is fully integrated with other Azure services including infrastructure as a service, virtualization, and multi-cloud services that runs in three different programming languages.

Kaldi: A wide utility of Kaldi - well-known open-source toolkit for speech recognition. It is largely used in research and industrial jobs. It builds great support for customization of different models and allows using various acoustic modeling methods.

Wit.ai: Acquired by Facebook, Wit.ai has ASR capabilities as characteristics of its new platform that serves as a platform for natural language understanding. It is considered as one of the tools for busy developers and helps developers quickly design conversational AI.

CMU Sphinx: An open-source toolkit is available at Carnegie Mellon University, called Sphinx, which gives an embedded ASR model for both online and offline purposes. It has sensitivity to various programming languages and can be configured to be tailored to particular needs.

OpenAI Whisper: OpenAI’s Whisper is a highly robust ASR model that fits it for recognizing accents and getting rid of background noise as well as for multilingual purposes. It involves a bigtruely diverse dataset and can be dubed upon for different speech processing tasks.

Every one of these models does have pros and cons. It is depending on accuracy demand, language support, deployment description, and integration facilities .

2.4 Used (ASR) MODEL

OpenAI is an artificial intelligence research organization focused on developing and promoting friendly AI for the benefit of humanity. They have created various powerful language models, including GPT (Generative Pre-trained Transformer), which can generate human-like text based on given prompts. OpenAI also conducts research in areas such as reinforcement learning, robotics, and AI safety.

Whisper is an automatic speech recognition (ASR) system developed by OpenAI, trained on a vast dataset of 680,000 hours of multilingual and multitask supervised data sourced from the web. This extensive and diverse dataset enhances the model's robustness to accents, background noise, and technical language. The architecture of Whisper is simple yet effective, utilizing an encoder-decoder Transformer. Input audio is segmented into 30-second chunks, converted into log-Mel spectrograms, and fed into an encoder, with a decoder trained to predict corresponding text captions. Special tokens are integrated to enable tasks such as language identification, phrase-level timestamps, multilingual speech transcription, and speech translation to English.

Unlike other approaches that utilize smaller, more closely paired audio-text datasets or unsupervised audio pretraining, Whisper's training on a large and diverse dataset without fine-tuning to any specific one contributes to its robustness. Although it may not outperform specialized models on benchmarks like LibriSpeech, Whisper demonstrates superior zero-shot performance across diverse datasets, making 50% fewer errors than comparable models.

A significant portion of Whisper's dataset comprises non-English audio, and the model is alternately tasked with transcribing in the original language or translating to English. This approach proves highly effective in learning speech-to-text translation, surpassing the supervised state-of-the-art (SOTA) on the CoVoST2 to English translation task in a zero-shot setting. Overall, Whisper represents a promising advancement in ASR technology, offering improved robustness and multilingual capabilities.

2.5 Why whisper model

OpenAI Whisper offers several advantages that make it a compelling choice for speech recognition tasks:OpenAI Whisper offers several advantages that make it a compelling choice for speech recognition tasks:

Robustness: Whisper was trained on a vast and varied data set, which makes it a robust device that spans a wide area from accents to background noise and technical language. This implies that the speech recognition system is totally reliable and can be used in any kind of speech environment.

Multilingual Capability: Whisper can be programmed to transcribe and translate talk from languages of all types into English. In this way, it can be used widely in any multilingual environment.

Zero-shot Performance: Whisper, as the best model working on a zero-shot basis, produces

fewer errors on varied data sets than other models after the model adapting process has thinned out.

End-to-End Architecture: Whisper is based upon a simple yet effective encoder-decoder transformer architecture that eliminated the confusing elements of ASR and made the process practically straightforward.

Open-source: OpenAI has made Whisper publicly available and shared the models and inference code, which can be utilized by developers and researchers to create apps and run research around fine speech processing.

Speech-to-Text Translation: In the case of Whisper, the model was able to trump other models at the tasks of speech-to-text conversion, being especially striking in the quality of producing English text even from zero-shot input.

In conclusion, OpenAI Whisper is a tempting solution for a variety of speech detection jobs as it is perfect for several skills such as resistance, multi-lingualism, and performance.

2.6 Limitation of whisper model

The Whisper ASR model, while advanced, has limitations. It may struggle with specialized domains, context understanding, and variability in accuracy. Resource intensiveness, language support issues, and privacy concerns are also notable. Maintenance and lack of fine-tuning pose additional challenges. Addressing these requires ongoing research and development efforts.

The Whisper ASR model, despite its advancements, also has certain limitations:The Whisper ASR model, despite its advancements, also has certain limitations:

Domain Specificity: While Whisper might not perform very well in specialized domains or industries specific to certain lingoes, technical terms, or ethnic accents, it would be safe to say that it will still be ideal in domains or industries with general languages.Its on-the-web domain diversity may not be rich enough to fit individual subject-based specialties.

Contextual Understanding: Whisper marginally faces the sterner test of detecting cursing or contextually sensitive language.Nevertheless, this technology could be improved even more by focusing more on transmitting complex context-dependent language use, which speech-to-text is not very good at.

Accuracy Variability: As Whisper's precision can depend on different factors such as audio quality, the speaker's accent, and background noise levels, the audio conditions and just the accent play a significant role.In uncertain environments, for instance, when using recordings of bad quality or when mispronunciations of words occur, the model might deteriorate its performance.

Resource Intensiveness: Training and launching the Whisper model can be very resource-consuming; hence, presenting COVID-19 symptoms may only be possible for people or businesses with sufficient computing resources.

Language Support: Even though Whisper can deal with a wide range of different languages, it is not always available since there are language-specific issues, such as handling some of the dialects, that can be a problem.

Privacy Concerns: This will be a crucial point since the Whisper web data set will be an issue of personal privacy, data use, and security because of the potential for sensitive or confidential information to be accessed when deploying the model in the environment.

Maintenance and Updates: Just like Whisper or any other language model, it won't be able to keep pace without regular updates and maintenance to remain current and efficient.A continuous effort to modernize the model by adhering to the changes in the language model, adding new accents, and using other resources on a regular basis looks like quite a big and challenging task.

3 Hardware and software requirements

3.1 Hardware specification

The Project is developed in the system having following configuration

Processor	Intel(R) Xeon(R) CPU @ 2.20GHz
Ram	12.7 GB
Graphics ram	15 GB
Monitor	15” COLOR
Hard Disk	107.7 GB

3.2 System requirements

Operation System	Ubuntu (Google Colaboratory)
Programming Language	Python
Environment	IPYNB (Python Notebook)

Python Environment

You should have a Python environment installed on your system. The code is written in Python, so you'll need Python to execute it. You can download Python from the official Python website (https://www.python.org/downloads/) and choose a version compatible with your operating system**.**

3.3 Python libraries

The code relies on several Python libraries. You can install these libraries using `pip`.

list for reference:

- pandas

- nltk

scikit-learn (for machine learning)

- wordcloud

- matplotlib

- pydub

-

You can install these libraries with the following commands:

pip install pandas re emoji contractions nltk scikit-learn transformers wordcloud matplotlib

NLTK Data: The code uses the Natural Language Toolkit (NLTK) for various natural language processing tasks. You will need to download some NLTK data using the following commands

import nltk

nltk.download('words')

Operating System

The code should work on most major operating systems, including Windows, macOS, and Linux.

Hardware

The code should run on standard desktop or laptop computers. There are no specific hardware requirements, but if you're working with very large datasets, you may benefit from a computer with more memory and processing power.

IDE or Text Editor

You can run the code in a Python IDE (e.g., PyCharm, VSCode, or Jupyter Notebook) or a simple text editor, depending on your preference. In this research, Google Colaboratory

HOW TO INSTALL PYTHON IDE

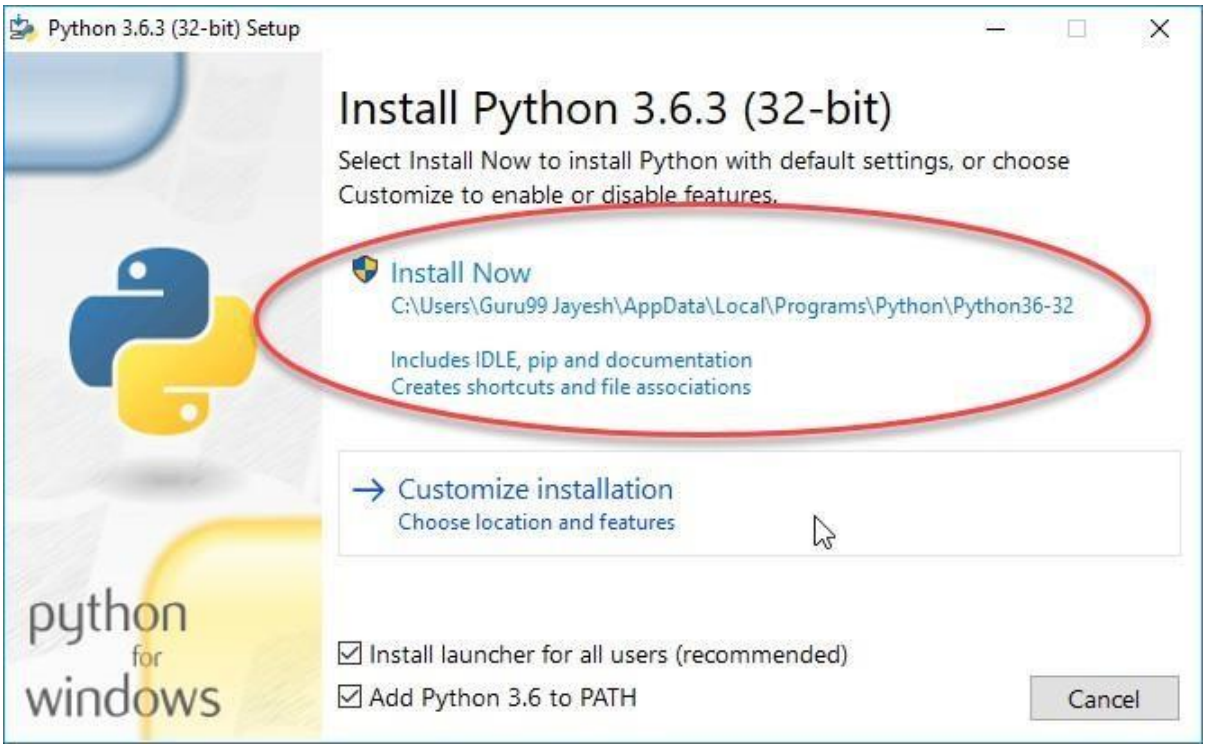
Below is a step-by-step process on how to download and install Python on Windows:

Step 1: To download and install

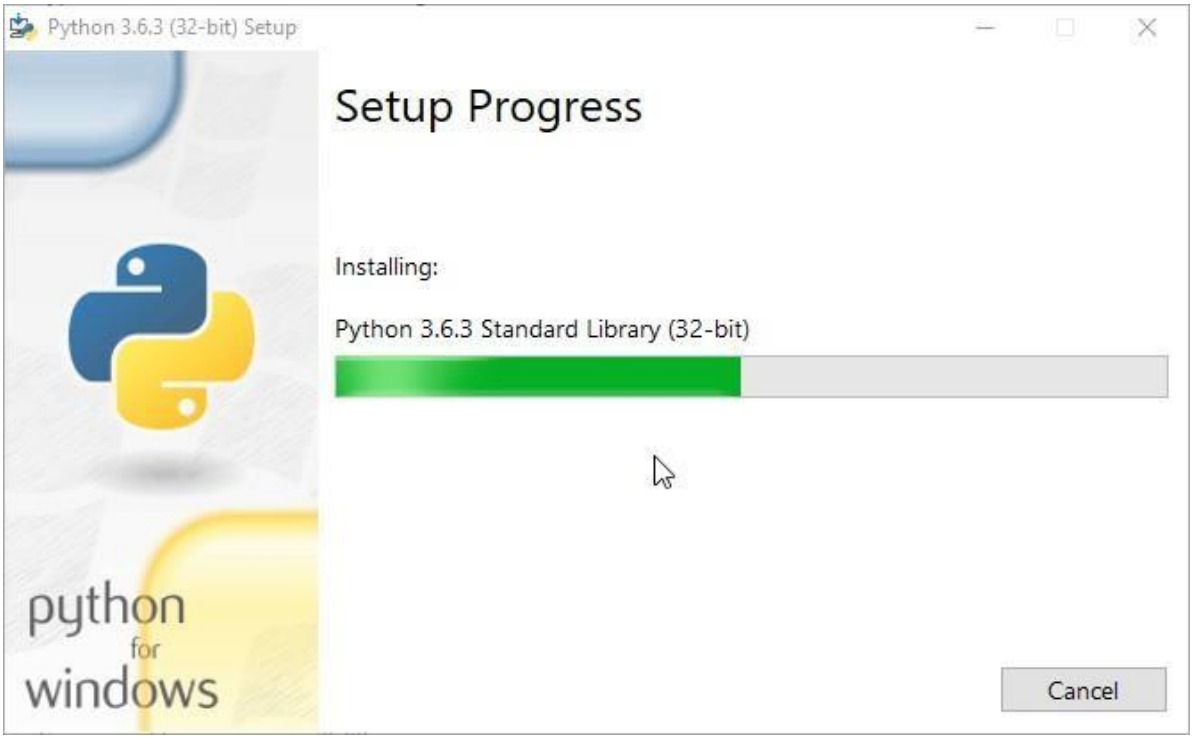
Data: You need to provide a CSV file with the data you want to process. The code expects the data to be in a specific format, so make sure your CSV file follows that format.

For Python, visit the official website of Python at https://www.python.org/downloads/ and choose your version. We have chosen Python version 3.6.3.

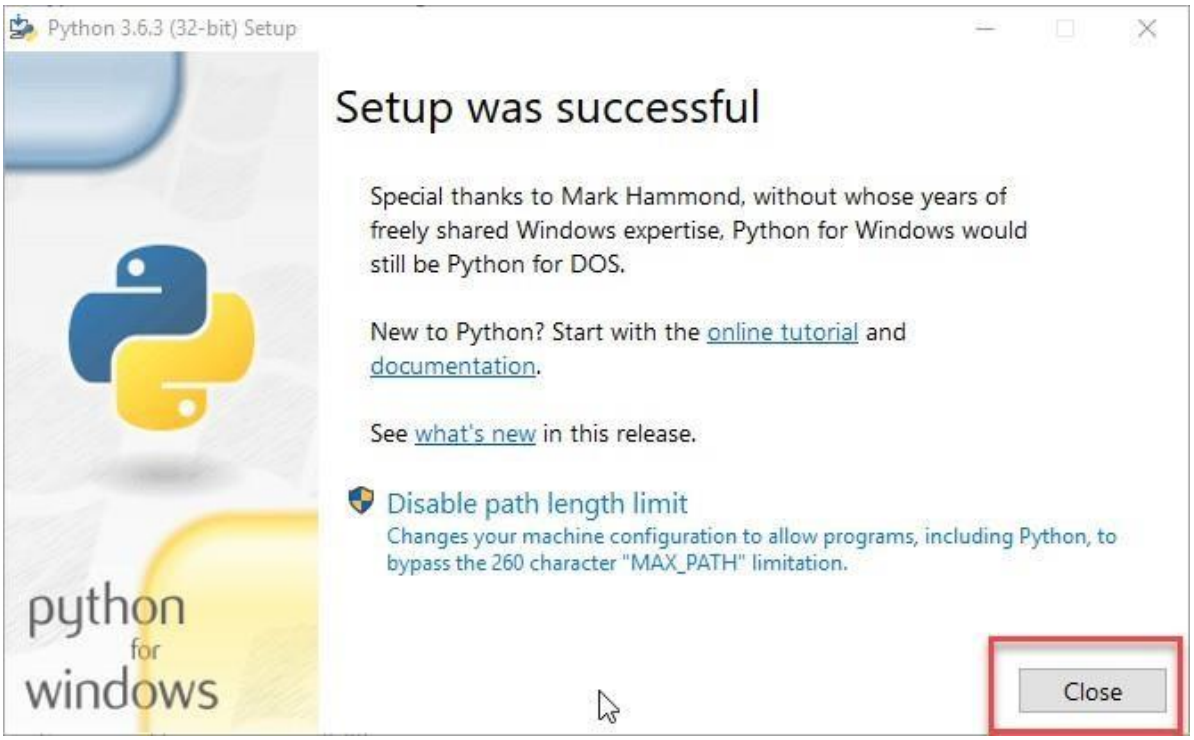
Step 2: Once the download is completed, run the .exe file to install Python. Now click on Install Now.



Step 3: You can see Python installing at this point.



Step 4: When it finishes, you can see a screen that says the Setup was successful. Now click on “Close”.



Google Colaboratory

Google Colaboratory, or Colab, is an as-a-service version of Jupyter Notebook that enables you to write and execute Python code through your browser.

Jupyter Notebook is a free, open source creation from the Jupyter Project. A Jupyter notebook is like an interactive laboratory notebook that includes not just notes and data, but also code that can manipulate the data. The code can be executed within the notebook, which, in turn, can capture the code output. Applications such as Matlab and Mathematica pioneered this model, but unlike those applications, Jupyter is a browser-based web application.

Google Colab is built around Project Jupyter code and hosts Jupyter notebooks without requiring any local software installation. But while Jupyter notebooks support multiple languages, including Python, Julia and R, Colab currently only supports Python.

Colab notebooks are stored in a Google Drive account and can be shared with other users, similar to other Google Drive files. The notebooks also include an autosave feature, but they do not support simultaneous editing, so collaboration must be serial rather than parallel.

Colab is free, but has limitations. There are some code types that are forbidden, such as media serving and crypto mining. Available resources are also limited and vary depending on demand, though Google Colab offers a pro version with more reliable resourcing. There are other cloud services based on Jupyter Notebook, including Azure Notebooks from Microsoft and SageMaker Notebooks from Amazon.

The Benefits of Google Colab

Enterprise data analysts and analytics developers can use Colab to work through data analytics and manipulation problems in collaboration. They can write, execute and revise core code in a tight loop, developing the documentation in Markdown format, LaTeX or HTML as they go.

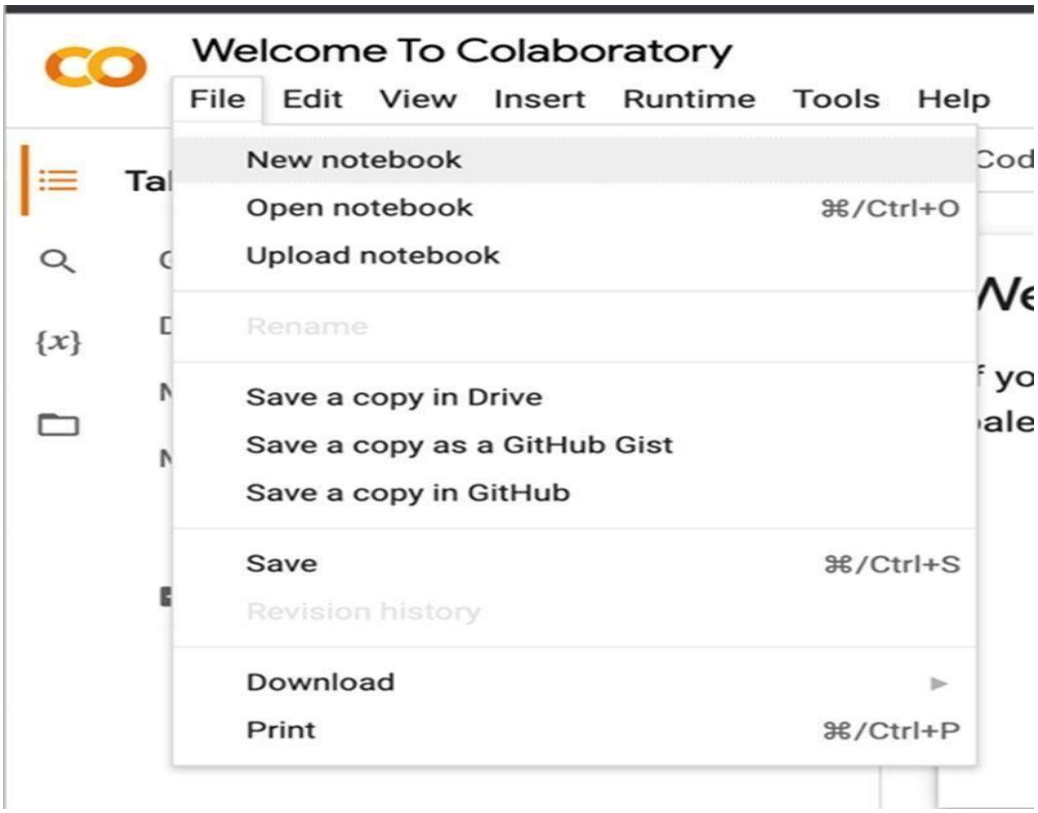
Notebooks can include embedded images as part of the documentation or as generated output. In addition, you can copy finished analytics code, with documentation, into other platforms for production use once sufficiently tested and debugged.

Google Colab eliminates the need for complex configuration setup and installation, as it runs right in the browser. It also includes pre-installed Python libraries that require no setup to use.

How to Use Colaboratory

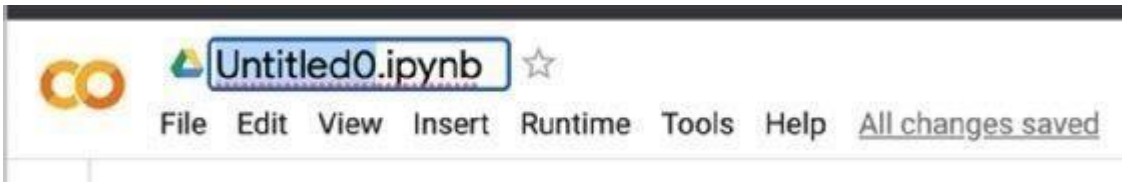
To use Colaboratory, you must have a Google account.

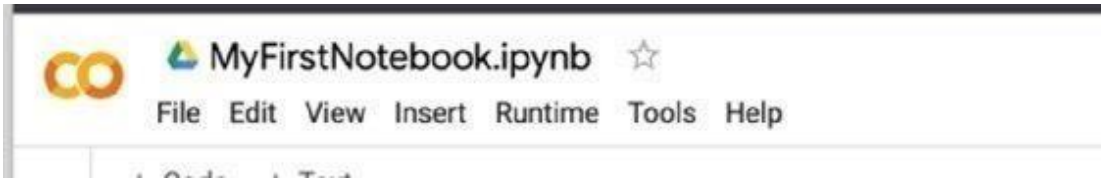
On your first visit, you will see a Welcome To Colaboratory notebook with links to video introductions and basic information on how to use Colab.



If you are not yet logged in to a Google account, the system will prompt you to log in.

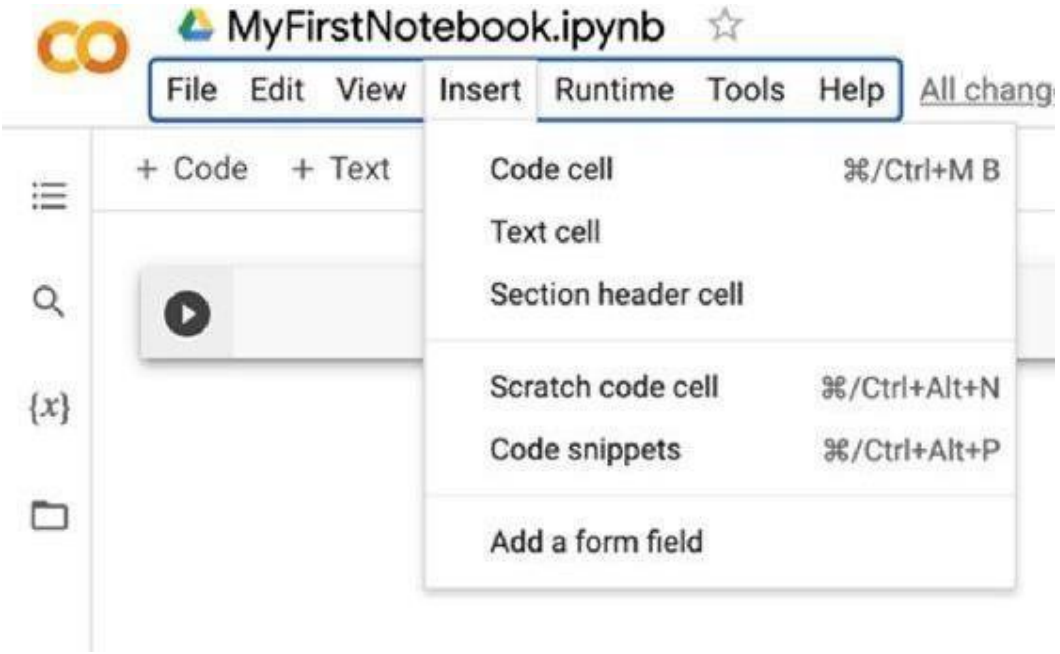
The notebook will by default have a generic name; click on the filename field to rename it.





The file type, IPYNB, is short for "IPython notebook" because IPython was the forerunner of Jupyter Notebook.

The interface allows you to insert various kinds of cells, mainly text and code, which have their own shortcut buttons under the menu bar via the Insert menu.

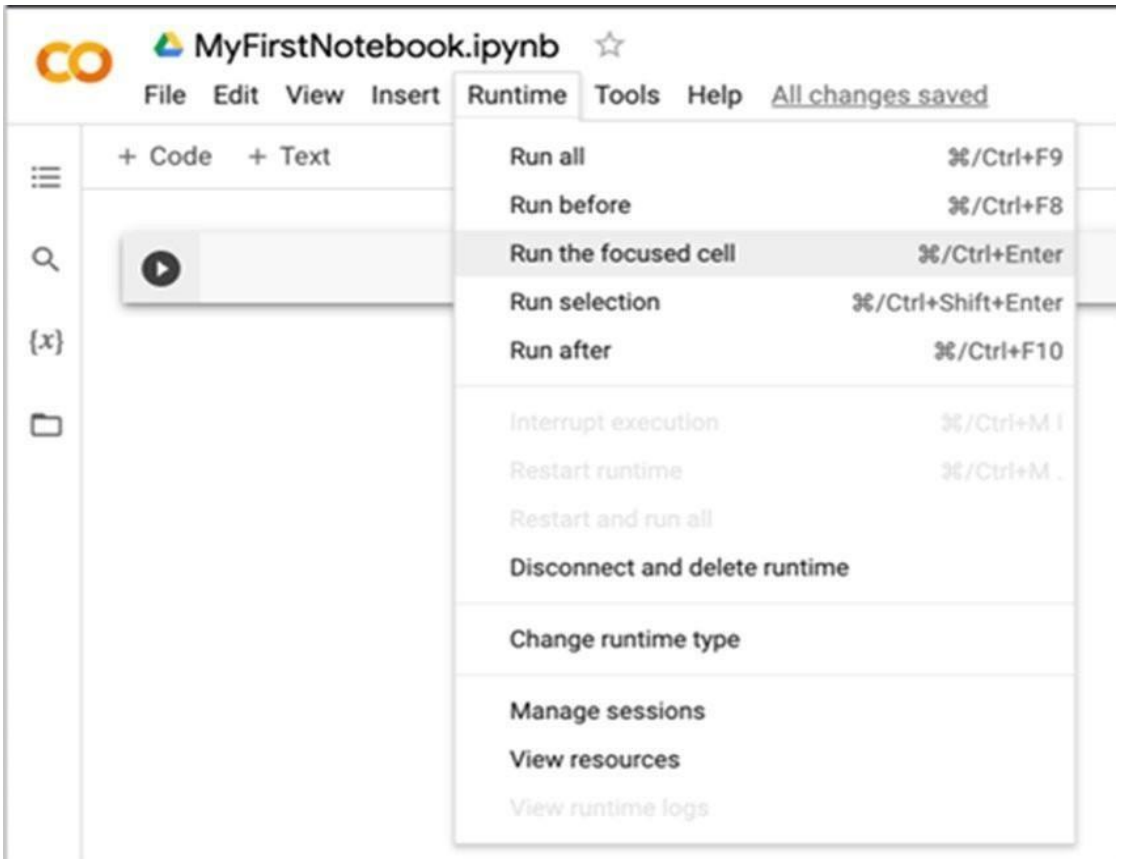


Because notebooks are meant for sharing, there are accommodations throughout for structured documentation.

Code, Debug, Repeat

You can insert Python code to execute in a code cell. The code can be entirely standalone or imported from various Python libraries.

A notebook can be treated as a rolling log of work, with earlier code snippets being no longer executed in favor of later ones, or treated as an evolving set of code blocks intended for ongoing execution. The Runtime menu offers execution options, such as Run all, Run before or Run the focused cell, to match either approach.

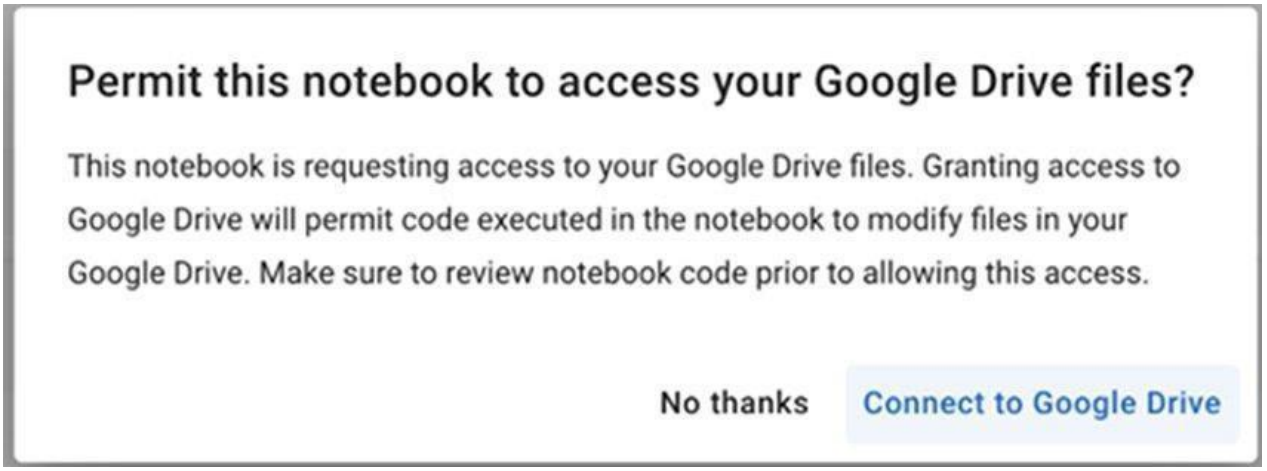


Incorporating Data into the Notebook

After getting comfortable with the interface and using it for initial test coding, you must eventually provide the code with data to analyse or otherwise manipulate.

Colab can mount a user's Google Drive to the VM hosting their notebook using a code cell.

Once you hit run, Google will ask for permission to mount the drive.



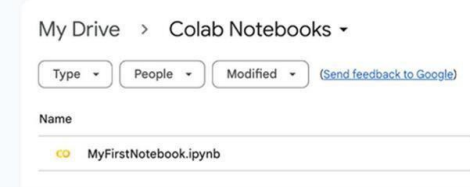
If you allow it to connect, you will then have access to the files in your Google Drive via the /my_drive path.

If you prefer not to grant access to your Drive space, you can upload files or any network file space mounted as a drive from your local machine instead.

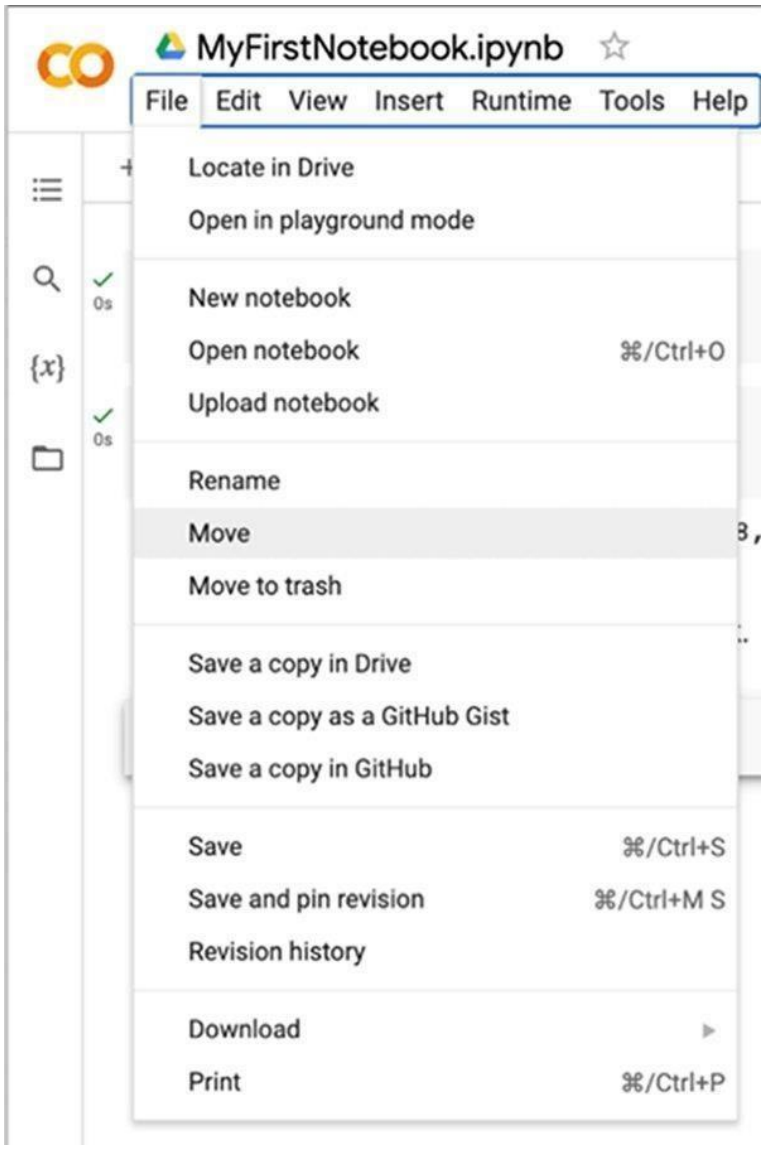
With file access, many functions are available to read data in various ways. For example, importing the Pandas library gives access to functions such as read_csv and read_json.

Save and Share

By default, Colab puts notebooks in a Colab Notebooks folder under My Drive in Google Drive.



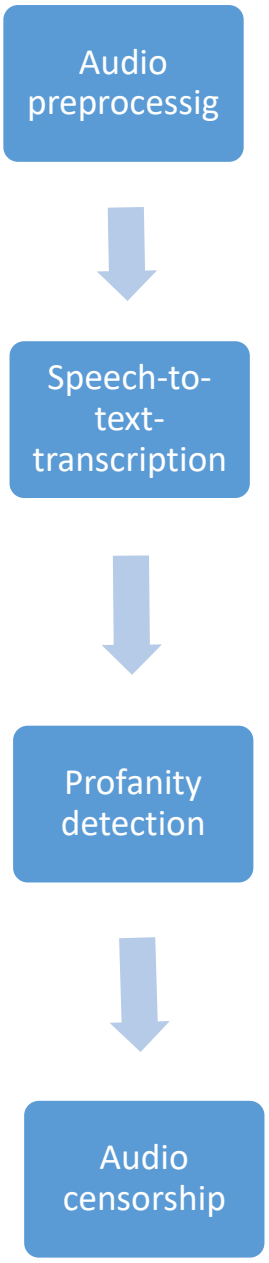
The File menu enables notebooks to be saved as named revisions in the version history, relocated using Move, or saved as a copy in Drive or GitHub. It also allows you to download and upload notebooks. Tools based on Jupyter provide broad compatibilities, so you can create notebooks in one place and then upload and use them in another.



You can use the Share button in the upper right to grant other Google users access to the notebook and to copy links.

Google also provides example notebooks illustrating available resources, such as pre-trained image classifiers and language transformers, as well as addressing common business problems, such as working with BigQuery or performing time series analytics. It also provides links to introductory Python coding notebooks.

4 Module design



4.1 Audio preprocessing

Data pre-processing in audio files are different from the general data pre-processing techniques used in other Machine Learning approaches. These can be mainly categorized into two sections.
1. Formatting the audio file
2. Cleaning the audio file
In this project, multiple approaches have been used for the above mentioned two sections.

Formatting the audio

In the present study, the MP3 files were subjected to a compression format with loss of information.

Thus, as a first step in formatting the audio clips, they were converted to the WAV format. The Pydub audio manipulation library, implemented in python, was employed for the MP3 to WAV conversion. The online data gathering method generated files in MP4 format as they were downloaded from YouTube. Hence, the youtube_dl python library was used for extracting the audio files directly from YouTube.

As regards the acoustic dimension, one can confidently say that a major part of the human language is uttered as a monosync. Consequently,Mono appeared to be more desirable than stereo. A stereo recording refers to a recording or playback process utilizing two or more speakers.two tracks plus another several tracks, while mono means one track only. For manipulating audio recordings, the Pydub Python library was used.Mono is processed. The result of these samplings was a new WAV file at the 16 000 Hz sampling rate.

Preprocessing Audio:

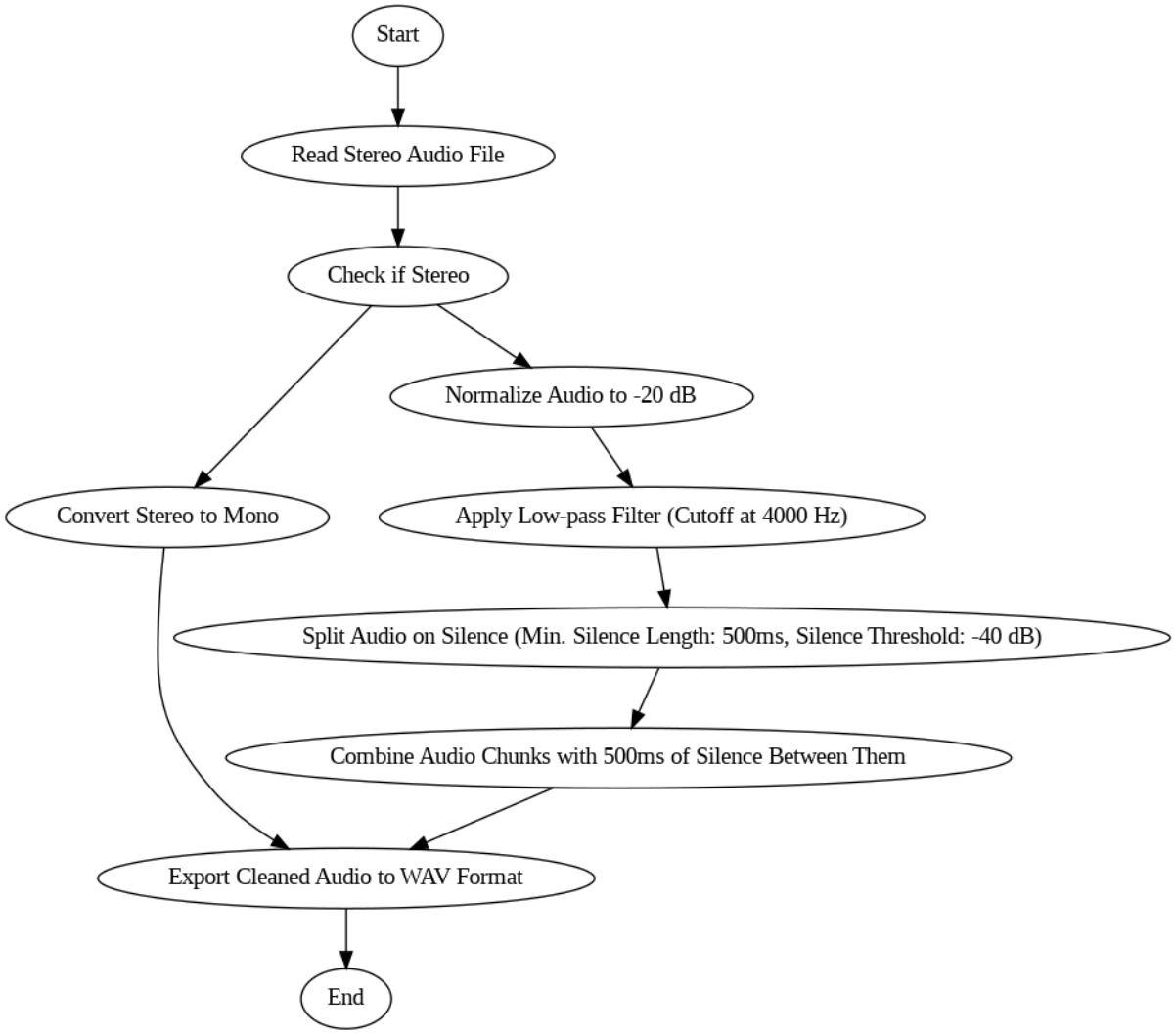
The preprocess_audio() function applies the preprocessing functions, namely, audio normalization to -20 dB and a low-pass filter to all the audio files for reduction of high-frequency noise. This phase is specific to the audio control and eliminating unwanted outside sounds.

Cleaning Audio:

The clean_audio function sets the input audio file and preprocesses it with the preprocess_audio function.

Following this, the audio is divided into chunks, which comprise phases of silence that have at least 500 ms of silence between each of them. This can be done with the help of pydub's silence module via its split_on_silence function.

Short chunks having a duration of less than 500 milliseconds are removed, and the objects remaining are grafted together to create a clean audio segment.



Stereo to Mono Conversion: The simple stereo format has two channels (left and right) in which each channel could record different parts of the sound. Switching stereo to mono, in fact, helps to unify both channels into a single so that only one channel is used, which is convenient for notification systems or applications that only support mono audio.

Audio Normalization: Equalization corrects the amount of audio to a pre-decided level. This is critical to ensuring that the levels of audio are uniform throughout all the tracks or even segments, which consequently prevents the tearing of speakers and makes the listener have a better experience, as well as no distortion or clipping.

Low-pass Filtering: Frequencies close together hinder the quality of the audio reduction. Background buzz or hiss, specifically, damage audio recordings. The retention of frequencies above a specified level using a low-pass filter will reduce background noise in the audio and restore clarity of the sound.

Splitting Audio on Silence: While many audio recordings may have machine gun fire between dialogue utterances or musical notes, others may use silence to emphasize certain moments. Use the silence detection ability that makes it possible for you to extract these exact parts, and that lets you work with them one at a time. This can be really helpful when you talk about call centers, where you want to spot a certain phrase someone said. At the same time, you don't hear the background noise at all.

Combining Audio Chunks: The next step, alternating overlaying the pieces and cutting out spaces of silence with a certain time duration between them, enables the stream to be smooth and whole as if it came from one source. It can therefore give a better voice to the audio and eliminate any random breaks or delays between two sentences, creating a more natural subject.

Each of these preprocessing approaches' roles makes the source audio data cleaner, consistent, and ready to be available for various applications, including speech recognition, audio transcription, and the production of music, among others. The significance of each step may be determined by other conditions, such as the purpose of the task and what properties of audio data are required.

4.2 Speech-to-text- transcription

The development of unsupervised pre-training techniques, represented mainly in Wav2Vec 2.0, was the biggest step forward and has substantially contributed to speech recognition advances. These approaches are data-driven, indicative of the fact that there is plenty of data that can be used for their training, including unlabeled large sets of speech that can be expanded up to millions of hours. This method has given rise to some developments in the state of the art, especially in data-driven environments.

On the other hand, though the pre-trained audio encoders get rather appropriately-formed representations of speech, they still do not have a correspondingly accurate decoder. Therefore, this requires domain adaptation for hard tasks like speech recognition, which could be thrown at amateurs. Also, we note that fine-tuning can lead to a specificity fit to particular datasets at the potential risk of impaired robustness.

Recent research found that pretrained models that were trained using different datasets are much

better and easier to generalize than models that have gone through only one specific set of data. Nevertheless, the scarcity of a large quantity of labeled speech data of outstanding quality is still an issue. However, efforts such as SpeechStew combine them more, and the volume of label data is still relatively small in regard to the huge amount of unlabeled audio that is used for unsupervised pre-training.

The speech-to-text transcription Material for that offensive language censorship application is a major part of this system and functions to transform audio recordings of talking into text transcripts. Here's a detailed overview of this module:Here's a detailed overview of this module:

Functionality:

Recognize the spoken words from audio files, then transcribe them into written text.

keeps timestamps of every word so that the hours are synced with the original recordings.

Key Features:

1. Speech Recognition:
Employs the concept of current voice recognition technology known as models and algorithms in the conversion of spoken words into text. They would utilize cutting-edge learning approaches like recurrent neural networks (RNNs), convolutional neural networks (CNNs), or models such as BERT or Wave2Vec.

2. Language Support: Supports multiple languages and dialects that provide a wide range of audio content, accommodating the diversity of audio materials. Language models derive their proficiency either from training or fine-tuning language data, depending on the BLEU (Bilingual Evaluation Under Study) scores.

3. Noise Handling: This feature reduces noise and enhances sound quality, thereby improving the auto-generated transcription sentences in unfavorable sound environments, e.g., in noisy and/or background interference sound recordings.

4. Real-Time Processing: Designed for the task of carrying out real-time audio feeds, such as in live broadcasts or interactive systems, which highly necessitate immediate transcription feedback.

5.Customization Options: Offers the possibilities to configure parameters for transcription to do individual adjustments like the settings that measure the recognition confidence thresholds, speaker diarization, or language-specific models to meet different needs.

Integration:

Such APIs include speech recognition libraries and interfaces provided by specialized companies such as Google Cloud Speech-to-Text, Amazon Transcribe, and Mozilla DeepSpeech, among others.

supports popular audio input and output formats like WAV, MP3, and other common formats. This will guarantee the application will connect with a wide range of recording sources.

Error Handling:

implements a strong error-handling mechanism to automatically re-read, correct, or add any missing or misunderstood words.

Included error correction strategies, among other spell-checking or context-based correction, are aimed at increasing the accuracy of transcription.

Performance Optimization:
Makes optimal use of processing speed and resource utilization so as to get good transcription performance, especially with large audio files or high-throughput processing cases.

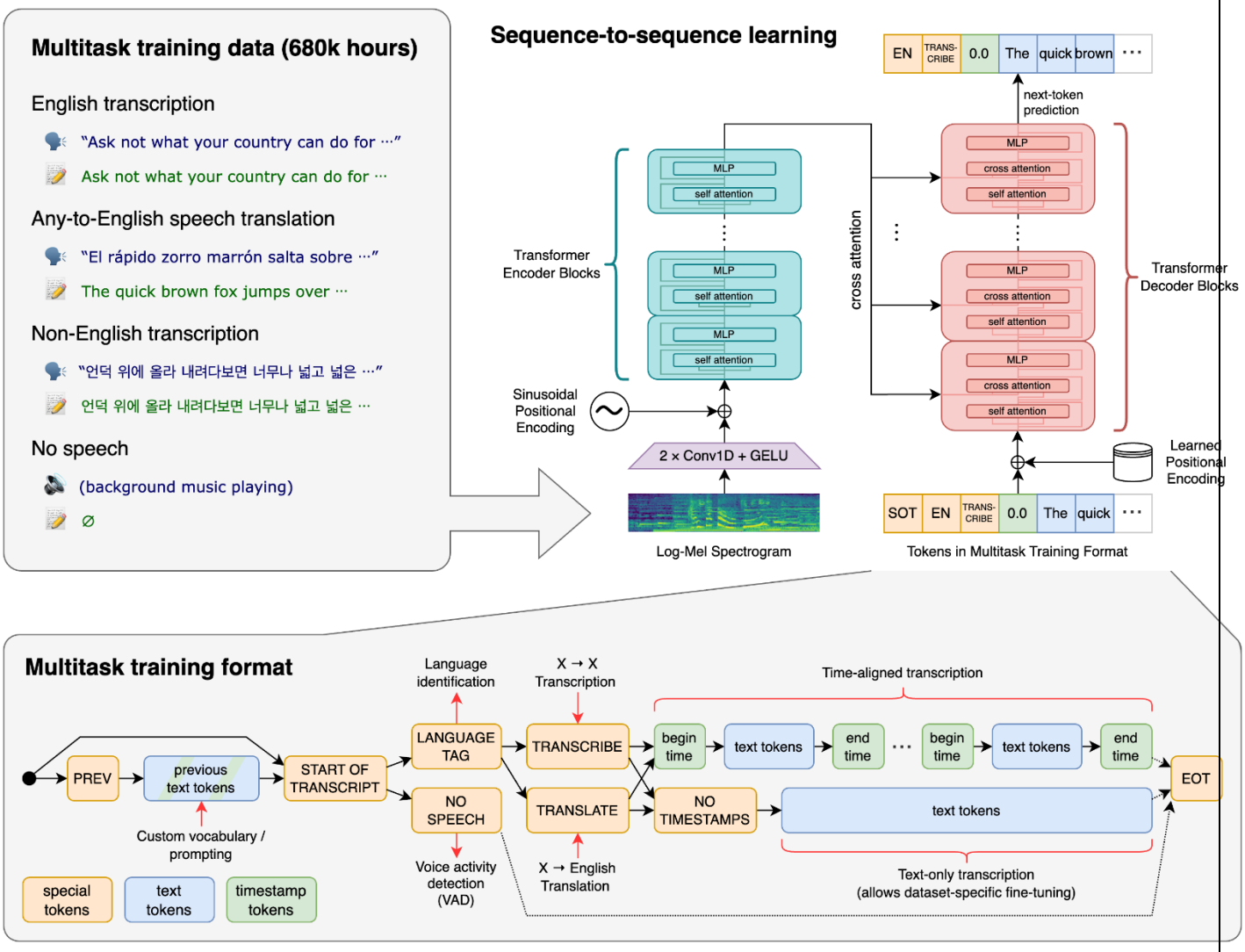
Utilizes paralleling methodologies and distributed computing structures for scalability and performance advancement.

Quality Assurance:

Includes quality assurance mechanisms such as a checklist for validation and transcription accuracy metrics to ensure the transcriptioned text is in absolute accordance with the original audio.The Speech-to-Text Transcription Module forms the basis for the profanity censorship mechanism, whereby audio documents are successfully converted to texts for further profanity detection and censorship procedures. Its reliability and accuracy are an important part of the overall good-working of the censorship system.

Whisper

Whisper is a general-purpose speech recognition model. It is trained on a large dataset of diverse audio and is also a multitasking model that can perform multilingual speech recognition, speech translation, and language identification.



Model Overview:

Whisper is one of the Transformer-based sequence-to-sequence models that has been trained on large datasets and it undertakes such tasks as multilingual speech recognition, speech translation, language identification and voice activity detection.

It approaches directly bypasses the stages of traditional speech-processing pipelines performing the tasks as tokens' sequence to be predicted by the decoder.

A Transformer sequence-to-sequence model is trained on various speech processing tasks, including multilingual speech recognition, speech translation, spoken language identification, and voice activity detection. These tasks are jointly represented as a sequence of tokens to be predicted by the decoder, allowing a single model to replace many stages of a traditional speech-processing pipeline. The multitask training format uses a set of special tokens that serve as task specifiers or classification targets.

Setup Instructions:

➢ As for training and testing the 3.8-3.11 version of Python and the 1.10.1 PyTorch edition are being used.

➢ Python codebase is using several packages, including one of OpenAI's – it's tiktoken for fast tokenizer implementation.

➢ Installation can be done via pip, either by installing the latest release (pip install -U openai-whisper) or directly from the GitHub repository (pip install git+https://github.com/openai/whisper.git>).

➢ The ffmpeg command-line tool is yet another dependency we need. Being distributed as packages through package managers such as apt, pacman, Homebrew, Chocolatey, and Scoop makes it easier to install.

4.3 WhisperX(Text time stamp retrieval)

In an era where the digital landscape is dominated by multimedia content, transcription tools have become essential for content creators, businesses, and educators alike. Whether it's repurposing podcasts, making videos accessible, or simply preserving valuable conversations, transcriptions serve a crucial role. OpenAI's Whisper API has been a game-changer in this domain, offering top-tier audio transcription capabilities. But what if you could take it a step further, customizing your transcription experience to fit your exact needs? Enter WhisperX, a groundbreaking extension to OpenAI's Whisper with multilingual support

All thanks to an amazing release by m-bain: https://github.com/m-bain/whisperX

WhisperX: Unleashing the Power of Customization

WhisperX is not just another transcription tool; it's a space where customization meets convenience. Created by Ashhad Ahsan, this space extends the capabilities of OpenAI's Whisper API to provide users with an unparalleled audio transcription experience. With WhisperX, you have the power to tailor your transcriptions to suit your specific requirements, ensuring that you get the most out of your audio content.

Key Features of WhisperX:

1. Audio-to-Text Transcription: WhisperX excels in its core functionality — converting audio files into accurate text transcriptions. Whether you're dealing with interviews, lectures, podcasts, or any other audio content, this tool delivers precise results.

2. Customizable Settings: What sets WhisperX apart is its ability to let you fine-tune your transcription settings. You can adjust parameters like transcription accuracy, language recognition, and more, ensuring that the output aligns perfectly with your needs.

3. Subtitle Generation: For content creators, adding subtitles to videos is crucial for accessibility and wider reach. WhisperX simplifies this process by generating subtitles directly from your transcriptions, saving you time and effort.

4. SRT and VTT Support: In addition to plain text, WhisperX supports popular subtitle formats like SubRip (SRT) and WebVTT (VTT). This versatility makes it a valuable tool for video producers and educators looking to create engaging and accessible content.

How WhisperX Benefits You

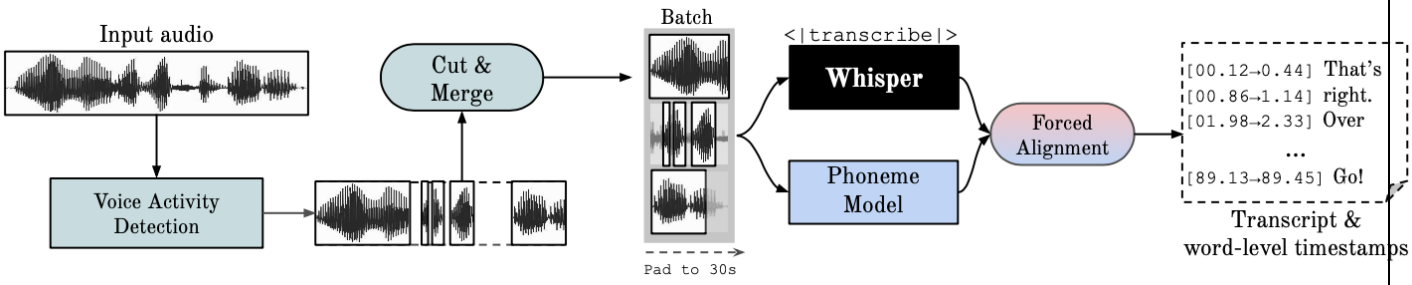
Now, let's delve into how WhisperX can benefit you, whether you're a content creator, a business owner, an educator, or simply someone who values efficient audio transcription.

1. **Content Creators**: WhisperX empowers content creators to enhance their videos by adding subtitles quickly and accurately. This not only improves accessibility but also makes your content more engaging for a broader audience.

2. **Businesses**: In the corporate world, time is money. WhisperX streamlines the process of transcribing important meetings, interviews, and presentations, helping businesses stay organized and efficient.

3. **Educators**: Educators can use WhisperX to create transcripts for their recorded lectures, making study materials more accessible for students. It also allows them to focus on teaching rather than manual transcription.

4. **Accessibility**: WhisperX's customizable settings and subtitle generation are a boon for making digital content accessible to people with disabilities. It ensures that everyone can enjoy and benefit from your multimedia content.



The WhisperX repository is the place where you can find ASR and AI-based automation using word-level timestamps and speaker diarization functionalities. Here's a summary of its key functionalities and components:Here's a summary of its key functionalities and components:

Batched Inference: Reduces transcription workload by allowing 70x real-time transcription in a Whisper large-v2 model, ultimately saving time and improving transcription speed.

Faster-Whisper Backend: Reaches with the backend optimized for speed, reducing the GPU memory usage up to 8GB for a large-v2 entity with a beam size of 5.

Word-Level Timestamps: Mention both quantified timestamps for each word and wav2vec2 alignment, so the synchronization with audio is precise and faultless.

Multispeaker ASR: Employed speaker diarization from pyannote-audio to provide multispeaker support for ASR, hence facilitation of transcribing of a conversation among multiple speakers.

Voice Activity Detection (VAD) Preprocessing: That incorporates dehallucinating VAD at all times, preserving batching and augmentation without any negative effect in WER.

Concurrently, WhisperX is aided by phoneme-based automatic speech recognition models and vocal segmentation techniques, resulting in improved transcript accuracy and speed.

Generally, WhisperX brings forth an all-in-one system that lets people rapidly and accurately understand spoken words and, therefore, can be applied for different purposes, including those necessitating automatic speech recognition of the finest kind.

Python usage

Transcription can also be performed within Python:

```
import whisper

model = whisper.load_model("base")
result = model.transcribe("audio.mp3")
print(result["text"])
```

Internally, the transcribe() method reads the entire file and processes the audio with a sliding 30-second window, performing autoregressive sequence-to-sequence predictions on each window.

Below is an example usage of whisper.detect_language() and whisper.decode() which provide lower-level access to the model.

```
import whisper



model = whisper.load_model("base")
```

```
# load audio and pad/trim it to fit 30 seconds

audio = whisper.load_audio("audio.mp3")

audio = whisper.pad_or_trim(audio)


# make log-Mel spectrogram and move to the same device as the model

mel = whisper.log_mel_spectrogram(audio).to(model.device)


# detect the spoken language

_, probs = model.detect_language(mel)

print(f"Detected language: {max(probs, key=probs.get)}")


# decode the audio

options = whisper.DecodingOptions()

result = whisper.decode(model, mel, options)


# print the recognized text

print(result.text)
```

4.4 Profanity detection

Content moderation is now an integral part of the operations of online platforms, guaranteeing that the massively contributed content remains in line with the published community guidelines and creates a friendly atmosphere for all users. A major problem during content moderation is recognizing and filtering profanity that might contain offensive speeches or be unsuitable for some audiences. In recent years, the field of natural language processing (NLP) has made massive progress, which has led to more proficiency in profanity detection techniques. However, one library among them stands out by the name "better profanity.". This post is about what "smart-swearing" is able to do and its advantages.

Understanding Profanity Detection:

The task of profane utterance comprises word detection to block outrageous language in text data. Customary methods are commonly based on attempts to find match-wording and may, thus, miss different changes, misspellings, or peculiarities of using obscene words in relation to a situation. Fundamental to the current profanity detection methods are the NLP algorithms employed to identify profanities with both machine learning models and the exhaustive profanity dictionaries that apply to accomplish this high accuracy and effectiveness.

"**Better-profanity**" is a Python package especially generated to perform profanity detection and deletion. Unlike basic profanity filters, "better-profanity" offers advanced features such as:Unlike basic profanity filters, "better-profanity" offers advanced features such as:

Contextual Profanity Detection: It takes into account the situation in which words are applied, making the work easier even in an elaborate language landscape.

Customization Options: Customized choices are available to the users, such as including or excluding a particular word based on the application or the needs of any given community.

Censorship Functionality: Moreover, an efficiently intelligent tool called "better-profanity" performs censorship reality, where you may put the approval or your own words meeting desired terms into the substring position.

Using "better profanity" in practice:

Writing an algorithm that detects profanity with "better profanity" is as simple as phenomenology. The library can be installed by developers either via pip or by sourcing it directly and added to their Python applications with minimum extra effort. Here's a basic example of how to use "better profanity" for profanity detection and censorship:Here's a basic example of how to use "better profanity" for profanity detection and censorship:

```
from better_profanity import profanity

# Check if a string contains profanity
text = "This is a sentence with some profanity like crap and damn."
if profanity.contains_profanity(text):
    print("Profanity detected!")
else:
    print("No profanity found.")

# Filter profanity from a string
filtered_text = profanity.censor(text)
print("Filtered text:", filtered_text)
```

Internet sites do care about content that is reported by users and leaves a proper environment for other users, so in principle, effective content moderation is the major concern. "Better-profanity"

library gives an extensive option on how the application can be upgraded to effectively deal with profanity detection and censorship activities. The main reason why "better-profanity" is so helpful is that it has a lot of unique features, provides options for customization, and is very easy to use, which enables developers to construct more advanced content moderation approaches, thus protecting user-created content from getting too rough in different online social environments.

Example profanity word

A	B	C	D	E	F	G	H	I	J	K
anal	ballsack	cup	damn	eatadick	facial	girls	hamflap	inbred	jackass	kawk
anus	bastard	call	deepthroa	eathairpie	fack	gae	handjob	incest	jackhole	kike
areole	bdsm	carpet	dick	ejaculate	fag	gai	hardcores	injun	jackoff	kikes
arian	beastial	cawk	dickhead	ejaculatec	fagg	gangbang	hardon		jap	kill
arrse	beastiality	chink	dildo	ejaculates	fagged	gangbang	hebe		japs	kinbaku
arse	bellend	cipa	dildos	ejaculatin	fagging	gangbang	heeb		jerk	kinky
arsehole	bestial	clit	dink	ejaculatin	faggit	ganja	hell		jerked	kinkyJesu
aryan	bestiality	clitoris	dinks	ejaculatio	faggitt	gassyass	hemp		jerkoff	kkk
asanchez	bimbo	clits	dlck	ejakulate	faggot	gay	hentai		jism	klan

4.5 Audio censorship

Method: Beeping or bleeping, depending on the speech that we will be listening to, happens to inject a concise beep sound through the objecting word or phrase, which will then make it difficult for us to hear it. This technique is a colling way used by various large media corporations to censor profanity or sensitive content while the flow of a conversation is not affected much.

Implementation :

The signature beep of the censoring software is displayed during the selected time segment where the objections occur.

A beep tone at 1000 Hz frequency with a specific duration is generated to make the signal clear but not distracting; the user's listening experience stays unobstructed.

The amplitude of the muting signal is correspondingly adjusted to be at the volume level of the first part of the sound, so that it would be unnoticeable to the audience during the transition.

After that, all of the bleeped audio that has been replaced by the beeps is then rendered or saved as a completely new audio file, where all of the censored audio is effectively masked by the inserted beep sounds.

Purpose: Chirping and other types of censorship are an art to comply with standards of broadcasting and community guidelines that prohibit language with explicit contexts or radio content that is too controversial. This technique helps content creators show their message without using objectionable words or phrases, sometimes in the form of beep sounds. This approach makes it easier to abide by censorship rules or to address the different sensitivities that audiences have towards specific topics.

Effectiveness and Considerations:

The practice of bleeping or beeping is a classic audio censoring technique that is widely used by radio, television, and social media in order to censor transmitted or published information.

The bleeping used as an instrument of censorship may ironically be noticed directly by the eavesdroppers, thus arousing their interest or contemplation towards the nearly-speaker words or phrases.

A specific choice of muffling time, frequency, and volume should be made to avoid disclosure and to eliminate background noise for the sake of the listener's contents.

The ethical issues of censorship consist of concerns about the suppression of freedom of expression to a large extent, artistic integrity, and the audience's unified interpretation of the content.

The audio manipulation for censorship, utilizing the bleeping technique, does censor profane words found in the transcript. The scripting solution purposely incorporates the beep signal, which replaces the tunes during intervals matching the censored words. Such a scheme proves the development of software-controlled audio editing.

5 Model evaluation

5.1 Evaluating ASR Model

Evaluating an Automatic Speech Recognition (ASR) model is a critical step in assessing its performance and ensuring its effectiveness in converting spoken language into text accurately. The evaluation process typically involves various metrics and techniques to measure the model's quality. Here are some key aspects and methods for evaluating ASR models:

- **Word Error Rate (WER):** WER measures the accuracy of the recognized words in the system's output compared to the reference or ground truth transcription. It quantifies the number of errors in terms of word substitutions, insertions, and deletions made by the ASR system.

  Here's how WER is calculated:
  Substitutions (S): This represents the number of words in the reference transcription that are incorrectly replaced by words in the ASR output.

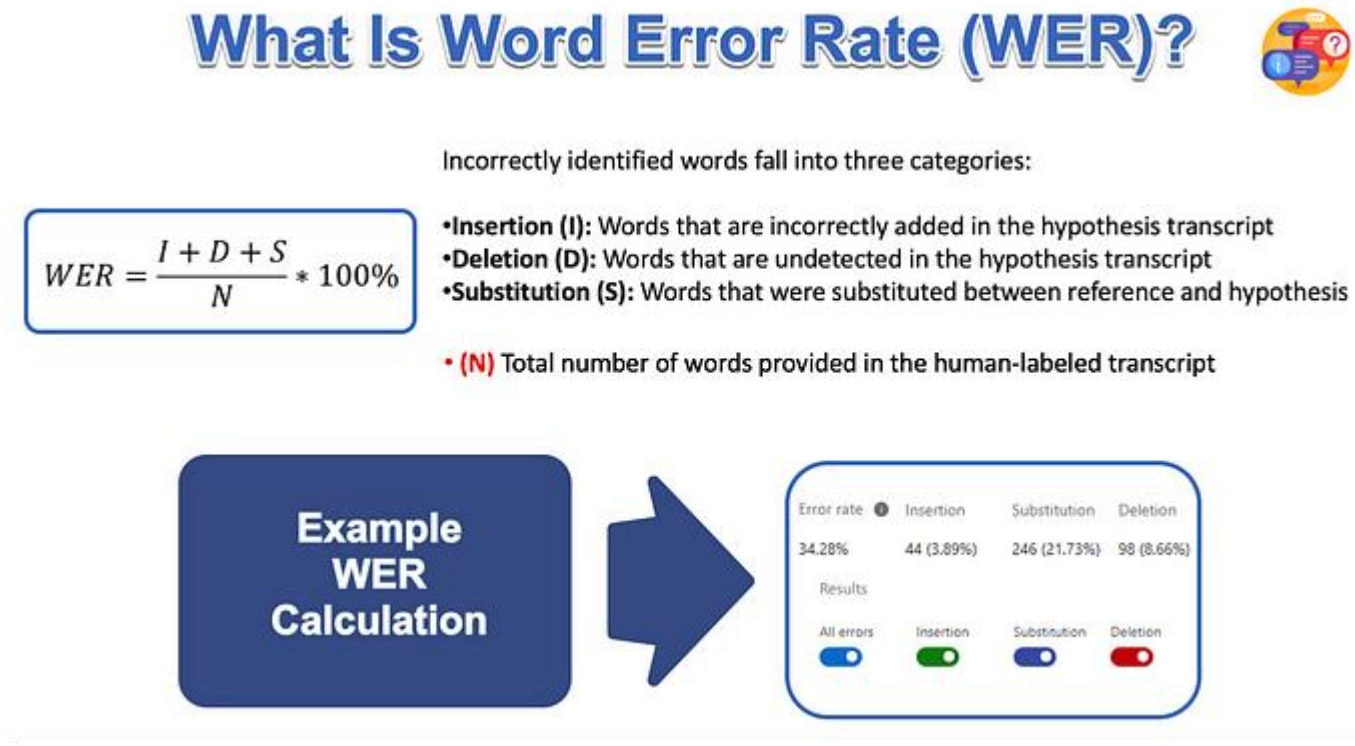
Insertions (I): Insertions count the number of extra words present in the ASR output that are not in the reference transcription.
Deletions (D): Deletions indicate the number of words in the reference transcription that are missing in the ASR output.

The formula for calculating WER is as follows:
Word Error Rate = (inserts + deletions + substitutions) / number of words in reference transcript
Simply put, this formula gives us the percentage of words that the ASR messed up. A lower WER, therefore, means a higher accuracy.

- **Character Error Rate (CER):** Similar to WER, CER measures the number of character-level errors in the recognized text compared to the reference text. It provides a finer-grained evaluation, especially useful for languages with complex scripts.

- **Accuracy:** This metric calculates the percentage of correctly recognized words or characters in the transcription. It is a straightforward measure of ASR model accuracy.

5.2 Word error rate



While the NLP field develops, profanity detection is one of the most important factors that improves speech recognition accuracy and maintains consistency in applications including voicebots, transcription services, and content moderation platforms. This content provides the readers with a

basic overview of the word error rate (WER) as one of the key factors for the quality assessment of profanity recognition units. We focus on WER as one of the criteria for evaluating the profanity detection accuracy level. We show how it affects the user experience norms, which are important in any business, especially ranging from marketing to social media. Also, we discuss the implications of these issues for the optimization of profanity detection algorithms.

Understanding Word Error Rate (WER) in Profanity Detection:Understanding Word Error Rate (WER) in Profanity Detection:

The word error rate (WER) estimates how much of a profanity detection system discrepancy there is in comparison to the detected transcript and the ground truth transcript. While WER calculates the differences in terms of insertions, deletions, and substitutions between the automatically generated transcript and the reference transcript, which need to be applied in order to get the aligned transcript, In speech recognition, when a WER rate is less than 1%, the probability of the output containing profanity is very low.

In the context of Word Error Rate (WER) evaluation for profanity detection or speech recognition systems, understanding the concepts of insertion, deletion, and substitution is crucial. These terms refer to different types of errors that occur when comparing the recognized or detected transcript with the reference (ground truth) transcript.

Insertion:

- An insertion occurs when the system erroneously adds an extra word or token that is not present in the reference transcript.
- For example, if the reference transcript contains the phrase "I don't like this," but the system outputs "I don't like this at all," the addition of "at all" represents an insertion error.

Deletion:

- A deletion occurs when the system fails to recognize or omit a word or token that appears in the reference transcript.
- For example, if the reference transcript contains the phrase "I love this song," but the system outputs "I love song," omitting the word "this," it represents a deletion error.

Substitution:

- A substitution occurs when the system incorrectly replaces one word or token with another word or token that differs from the reference transcript.
- For example, if the reference transcript contains the phrase "That's amazing," but the system outputs "That's astonishing," replacing "amazing" with "astonishing," it represents a substitution error.

Illustrative Example:

Consider the following reference transcript and the system's output:

Reference: "I can't believe you said that."

System Output: "I can't believe you said what."

In this example:

- There is one insertion: "what" is added in the system output but not present in the reference.
- There is one deletion: The word "that" is omitted in the system output compared to the reference.
- There are no substitutions because the words in the system output match those in the reference, albeit with an insertion and a deletion.

Calculating WER:

To calculate the Word Error Rate (WER), the total number of errors (insertions, deletions, and substitutions) is divided by the total number of words in the reference transcript. The resulting ratio is typically expressed as a percentage.

$$WER = (I + D + S) / N * 100\%$$

Where:

- I = Number of insertions
- D = Number of deletions
- S = Number of substitutions
- N = Total number of words in the reference transcript

By quantifying these types of errors, WER provides a comprehensive measure of the disparity between the recognized transcript and the reference transcript, allowing for objective evaluation and comparison of speech recognition or profanity detection systems.

Importance of WER in Profanity Detection:Importance of WER in Profanity Detection:

Performance Evaluation: This is the way for the metric system that is used to compare the effectiveness of systems to filter profane words objectively. Additionally, it offers insights about the system's capacity to correctly recognize and filter out offensive words, in addition to mitigating the number of false positive and false negative unlocks.

Quality Assurance: Human speech recognition (ASR) is a key method for profanity detection applications and quality monitoring processes. Monitoring WER on a periodic basis helps developers assess the system’s deterministic performance in terms of the stability, consistency, and robustness of profanity detection, whether this is across different datasets, languages, or user interactions.

User Experience: The efficacy of foul language inclusions influences the product or service under the influence of voice dominance applications and media content platforms. The more precise the

IRF is, the lower the WER is. It means that lower WER increases the satisfaction of users when using the system with content moderation because of the easy-to-reach leading rating.

5.3 Whisper's performance

Available models and languages

There are five model sizes, four with English-only versions, offering speed and accuracy tradeoffs. Below are the names of the available models and their approximate memory requirements and inference speed relative to the large model; actual speed may vary depending on many factors including the available hardware.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	tiny.en	tiny	~1 GB	~32x
base	74 M	base.en	base	~1 GB	~16x
small	244 M	small.en	small	~2 GB	~6x
medium	769 M	medium.en	medium	~5 GB	~2x
large	1550 M	N/A	large	~10 GB	1x

The .en models for English-only applications tend to perform better, especially for the tiny.en and base.en models. We observed that the difference becomes less significant for the small.en and medium.en models.

Whisper's performance varies widely depending on the language. The figure below shows a performance breakdown of large-v3 and large-v2 models by language, using WERs (word error rates) or CER (character error rates, shown in *Italic*) evaluated on the Common Voice 15 and Fleurs datasets.

Common Voice is an audio dataset that consists of a unique MP3 and corresponding text file. There are 9,283 recorded hours in the dataset. The dataset also includes demographic metadata like age, sex, and accent. The dataset consists of 7,335 validated hours in 60 languages.

Fleurs is the speech version of the FLoRes machine translation benchmark. We use 2009 n-way parallel sentences from the FLoRes dev and devtest publicly available sets, in 102 languages.

Training sets have around 10 hours of supervision. Speakers of the train sets are different than speakers from the dev/test sets. Multilingual fine-tuning is used and "unit error rate" (characters,

signs) of all languages is averaged. Languages and results are also grouped into seven geographical areas:

Western Europe: Asturian, Bosnian, Catalan, Croatian, Danish, Dutch, English, Finnish, French, Galician, German, Greek, Hungarian, Icelandic, Irish, Italian, Kabuverdianu, Luxembourgish, Maltese, Norwegian, Occitan, Portuguese, Spanish, Swedish, Welsh

Eastern Europe: Armenian, Belarusian, Bulgarian, Czech, Estonian, Georgian, Latvian, Lithuanian, Macedonian, Polish, Romanian, Russian, Serbian, Slovak, Slovenian, Ukrainian

Central-Asia/Middle-East/North-Africa: Arabic, Azerbaijani, Hebrew, Kazakh, Kyrgyz, Mongolian, Pashto, Persian, Sorani-Kurdish, Tajik, Turkish, Uzbek

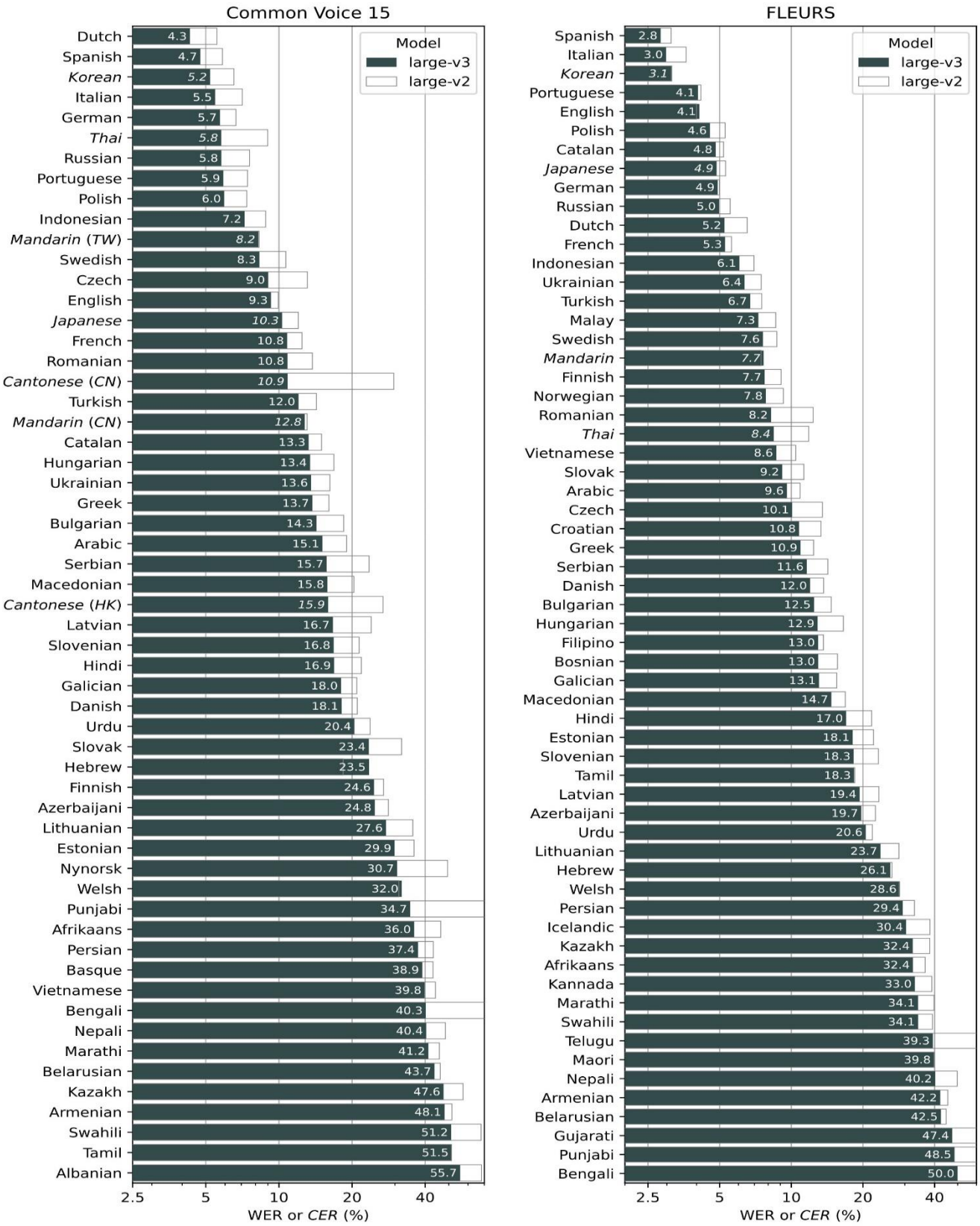
Sub-Saharan Africa: Afrikaans, Amharic, Fula, Ganda, Hausa, Igbo, Kamba, Lingala, Luo, Northern-Sotho, Nyanja, Oromo, Shona, Somali, Swahili, Umbundu, Wolof, Xhosa, Yoruba, Zulu

South-Asia: Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Nepali, Oriya, Punjabi, Sindhi, Tamil, Telugu, Urdu

South-East Asia: Burmese, Cebuano, Filipino, Indonesian, Javanese, Khmer, Lao, Malay, Maori, Thai, Vietnamese

CJK languages: Cantonese and Mandarin Chinese, Japanese, Korean

The evaluation conducted by OpenAI


Common Voice 15
Model
large-v3
large-v2
Dutch 4.3
Spanish 4.7
Korean 5.2
Italian 5.5
German 5.7
Thai 5.8
Russian 5.8
Portuguese 5.9
Polish 6.0
Indonesian 7.2
Mandarin (TW) 8.2
Swedish 8.3
Czech 9.0
English 9.3
Japanese 10.3
French 10.8
Romanian 10.8
Cantonese (CN) 10.9
Turkish 12.0
Mandarin (CN) 12.8
Catalan 13.3
Hungarian 13.4
Ukrainian 13.6
Greek 13.7
Bulgarian 14.3
Arabic 15.1
Serbian 15.7
Macedonian 15.8
Cantonese (HK) 15.9
Latvian 16.7
Slovenian 16.8
Hindi 16.9
Galician 18.0
Danish 18.1
Urdu 20.4
Slovak 23.4
Hebrew 23.5
Finnish 24.6
Azerbaijani 24.8
Lithuanian 27.6
Estonian 29.9
Nynorsk 30.7
Welsh 32.0
Punjabi 34.7
Afrikaans 36.0
Persian 37.4
Basque 38.9
Vietnamese 39.8
Bengali 40.3
Nepali 40.4
Marathi 41.2
Belarusian 43.7
Kazakh 47.6
Armenian 48.1
Swahili 51.2
Tamil 51.5
Albanian 55.7
2.5
5
10
20
40
WER or CER (%)
FLEURS
Model
large-v3
large-v2
Spanish 2.8
Italian 3.0
Korean 3.1
Portuguese 4.1
English 4.1
Polish 4.6
Catalan 4.8
Japanese 4.9
German 4.9
Russian 5.0
Dutch 5.2
French 5.3
Indonesian 6.1
Ukrainian 6.4
Turkish 6.7
Malay 7.3
Swedish 7.6
Mandarin 7.7
Finnish 7.7
Norwegian 7.8
Romanian 8.2
Thai 8.4
Vietnamese 8.6
Slovak 9.2
Arabic 9.6
Czech 10.1
Croatian 10.8
Greek 10.9
Serbian 11.6
Danish 12.0
Bulgarian 12.5
Hungarian 12.9
Filipino 13.0
Bosnian 13.0
Galician 13.1
Macedonian 14.7
Hindi 17.0
Estonian 18.1
Slovenian 18.3
Tamil 18.3
Latvian 19.4
Azerbaijani 19.7
Urdu 20.6
Lithuanian 23.7
Hebrew 26.1
Welsh 28.6
Persian 29.4
Icelandic 30.4
Kazakh 32.4
Afrikaans 32.4
Kannada 33.0
Marathi 34.1
Swahili 34.1
Telugu 39.3
Maori 39.8
Nepali 40.2
Armenian 42.2
Belarusian 42.5
Gujarati 47.4
Punjabi 48.5
Bengali 50.0
2.5
5
10
20
40
WER or CER (%)

6 SOURCE CODE

6.1 CODING

The coding phase brings the actual system into action by converting the design of the system into code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required and easily screwed into the system.

CODING STANDARDS

Coding standards are guidelines for programming that focus on the physical structure and appearance of the program. They make the code easier to read, understand, and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standards needed to achieve the above-mentioned objectives are as follows:

The program should be simple, clear, and easy to understand.

- Naming conventions
- Value conventions
- Script and comment procedure
- Message box format
- Exception and error handle

6.2 Code

PROFANITY FILTERING IN AUDIO

INSTALING DEPENDENCY

Inspired from package profanity of Ben Friedland, this library is significantly faster than the original one, by using string comparison instead of regex.

```
[ ]: !pip install better-profanity
```

```
[ ]: !pip install openai-whisper
```

PyDub 's AudioSegment class makes it easy to import and manipulate audio files with Python.

```
[ ]: !pip install pydub
```

```
Requirement already satisfied: pydub in /usr/local/lib/python3.10/dist-packages (0.25.1)
```

Used to visualize the words

```
[ ]: !pip install wordcloud
```

IMPORTING LIBRARYS

```
[ ]: import pandas as pd
import numpy as np
import wave
from pydub import AudioSegment
import better_profanity
from math import ceil
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import spacy
from spacy import displacy
import networkx as nx
from pydub.effects import normalize
import os
```

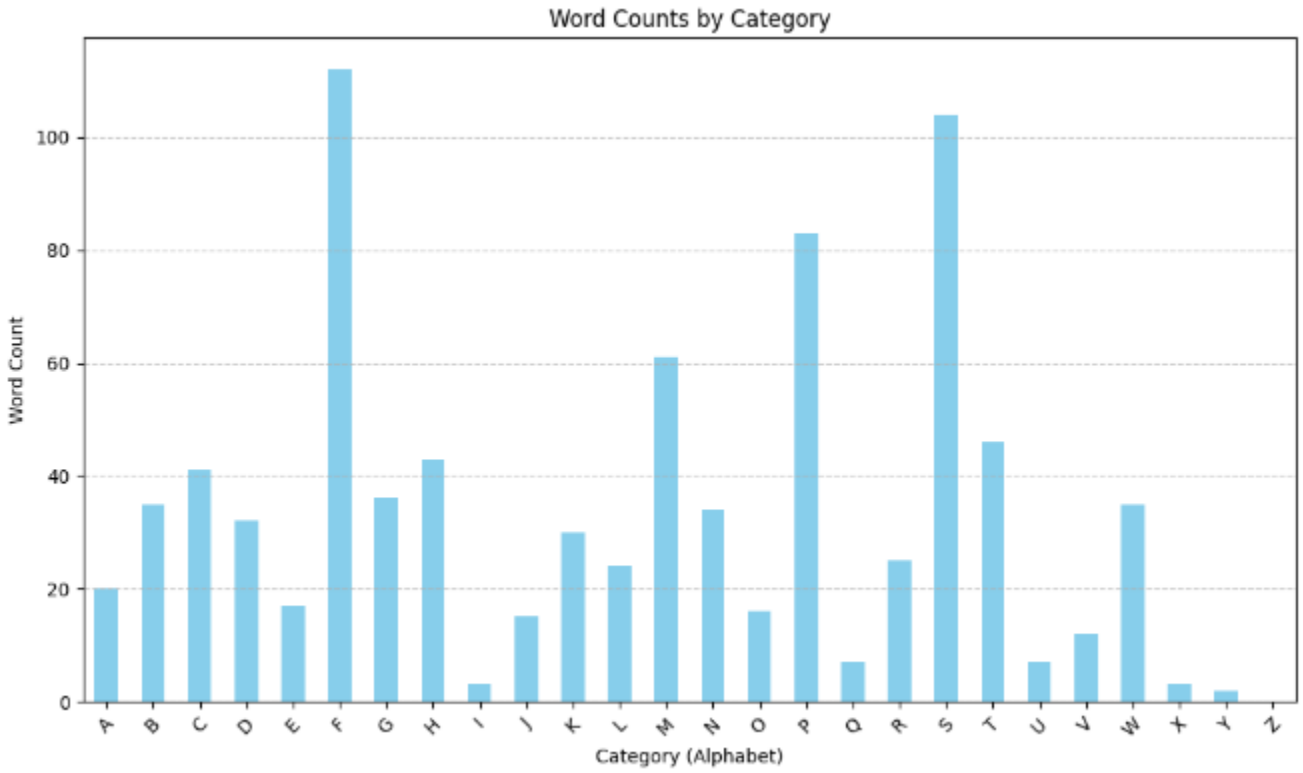
PROFANITY WORD ANALYSIS

```
[ ]: import pandas as pd

with open('/content/drive/MyDrive/profanity_wordlist.txt', 'r') as file:
    words = file.read().split()


alphabet_dict = {chr(i): [] for i in range(ord('A'), ord('Z')+1)}

# Classify words by alphabet
for word in words:
    first_letter = word[0].upper()
    if first_letter in alphabet_dict:
        alphabet_dict[first_letter].append(word)


max_length = max(len(words) for words in alphabet_dict.values())

for key, value in alphabet_dict.items():
    alphabet_dict[key] += [''] * (max_length - len(value))

# Create DataFrame
df = pd.DataFrame(alphabet_dict)

# Display the DataFrame
print(df)
```

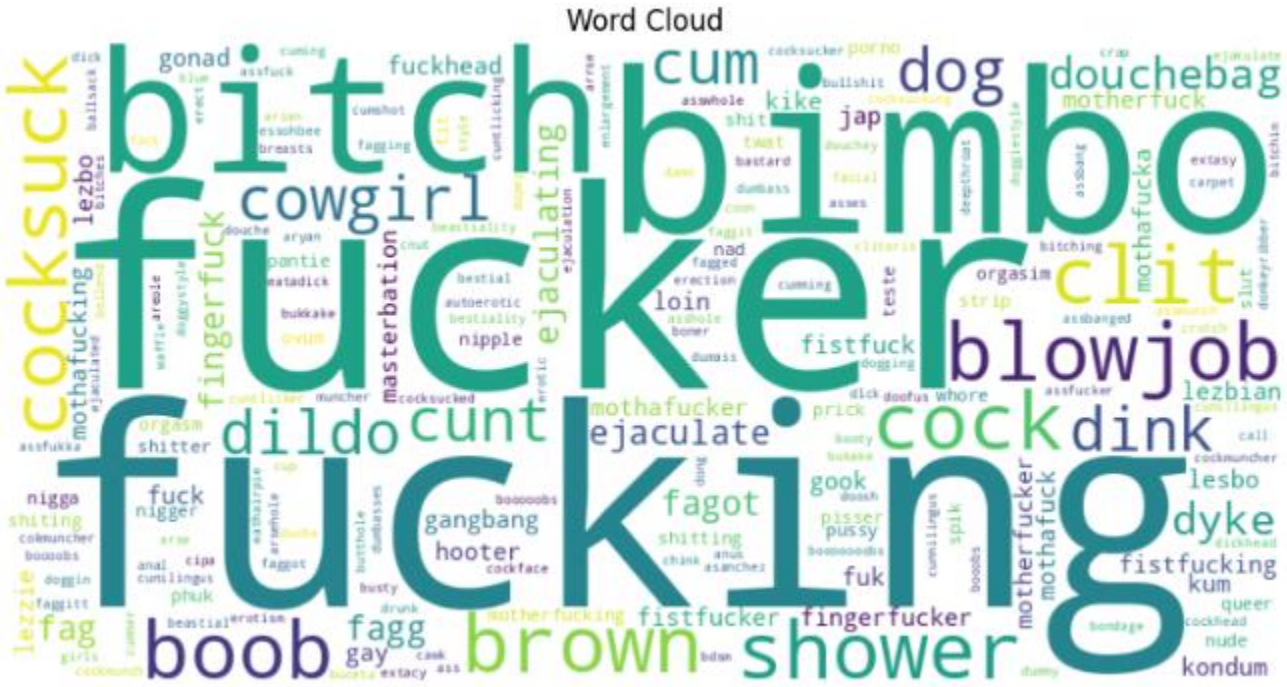
```
[ ]: category_counts = df.applymap(lambda x: 0 if x == '' else 1).sum()

# Display the counts
print(category_counts)
```

```
A     20
B     35
C     41
D     32
E     17
F    112
G     36
H     43
I      3
J     15
K     30
L     24
M     61
N     34
O     16
P     83
Q      7
R     25
S    104
T     46
U      7
V     12
W     35
X      3
Y      2
```

```
[ ]: # Plotting
plt.figure(figsize=(10, 6))
category_counts.plot(kind='bar', color='skyblue')
plt.title('Word Counts by Category')
plt.xlabel('Category (Alphabet)')
plt.ylabel('Word Count')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



[ ]:
```
# Join all words into a single string
all_words = ' '.join(words)

# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_words)

# Plot word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud')
plt.axis('off')
plt.show()
```



AUDIO ANALYSIS & PREPROCESSING

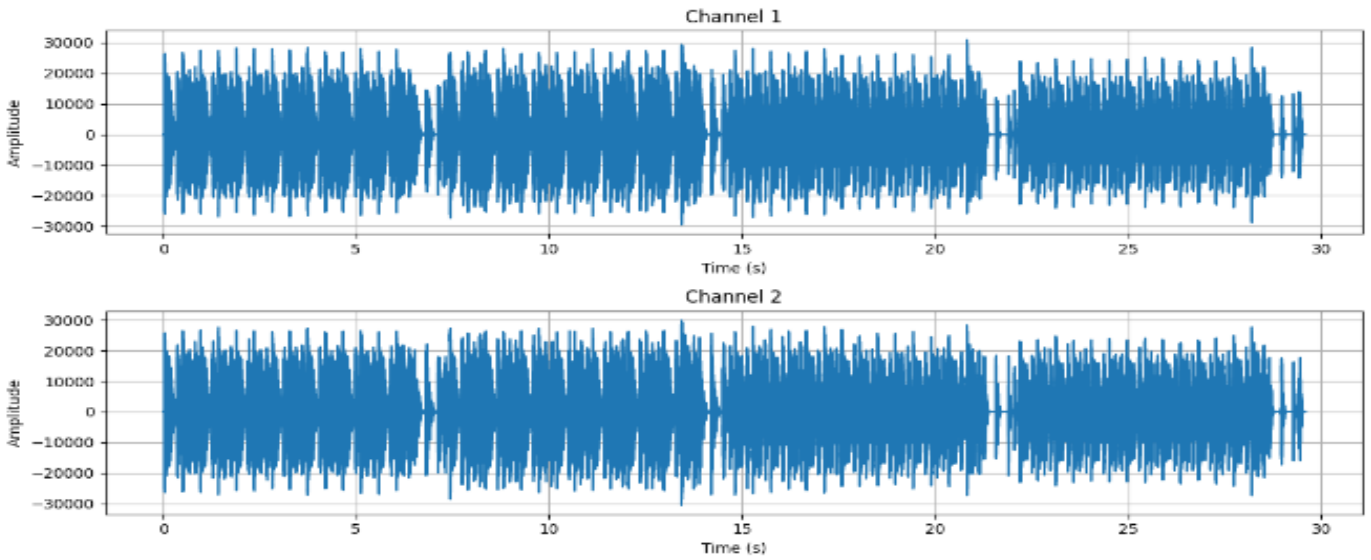
```
[ ]: def analyze_wav_file(file_path):
    try:
        # Open the WAV file
        with wave.open(file_path, 'rb') as wav_file:
            # Get basic information about the WAV file
            num_channels = wav_file.getnchannels()
            sample_width = wav_file.getsampwidth()
            frame_rate = wav_file.getframerate()
            num_frames = wav_file.getnframes()
            duration = num_frames / frame_rate

            print("WAV file information:")
            print(f"Number of channels: {num_channels}")
            print(f"Sample width (bytes): {sample_width}")
            print(f"Frame rate (frames per second): {frame_rate}")
            print(f"Number of frames: {num_frames}")
            print(f"Duration (seconds): {duration:.2f}")


            frames = wav_file.readframes(num_frames)
```

```
            frames = wav_file.readframes(num_frames)


    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
    except wave.Error as e:
        print(f"Error reading WAV file: {e}")


file_path = "//content/drive/MyDrive/grinder-drum-loop-6697.wav"
analyze_wav_file(file_path)
```

```
WAV file information:
Number of channels: 2
Sample width (bytes): 2
Frame rate (frames per second): 24000
Number of frames: 710208
Duration (seconds): 29.59
```

```
[ ]: # Load the WAV file
file_path = '/content/drive/MyDrive/grinder-drum-loop-6697.wav'
with wave.open(file_path, 'rb') as wav_file:
    num_channels = wav_file.getnchannels()
    frame_rate = wav_file.getframerate()
    num_frames = wav_file.getnframes()
    sample_width = wav_file.getsampwidth()

    # Read all frames from the WAV file
    frames = wav_file.readframes(num_frames)

# Convert frames to a NumPy array
samples = np.frombuffer(frames, dtype=np.int16)

# Reshape the samples into a 2D array for stereo
samples = samples.reshape(-1, num_channels)

# Calculate time array
time = np.arange(0, num_frames) / frame_rate

# Plot each channel separately in parallel
plt.figure(figsize=(12, 6))
for i in range(num_channels):
    plt.subplot(2, 1, i + 1)
    plt.plot(time, samples[:, i])
    plt.title(f'Channel {i + 1}')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```



CONVERT STEREO INTO MONO

```
[ ]: def stereo_to_mono(input_path, output_path):
    # Open the stereo WAV file
    with wave.open(input_path, 'rb') as wav_in:
        # Get parameters of the stereo WAV file
        num_channels = wav_in.getnchannels()
        num_frames = wav_in.getnframes()
        sample_width = wav_in.getsampwidth()
        frame_rate = wav_in.getframerate()

        # Check if the input file is stereo
        if num_channels != 2:
            print("Input file is not stereo.")
            return

        # Open a new mono WAV file for writing
        with wave.open(output_path, 'wb') as wav_out:
            # Set parameters for the mono file
            wav_out.setnchannels(1)  # Mono
            wav_out.setsampwidth(sample_width)
            wav_out.setframerate(frame_rate)
            wav_out.setnframes(num_frames)
```

```
            # Read and process frames
            for _ in range(num_frames):
                # Read one frame from the stereo file
                frame_stereo = wav_in.readframes(1)
                # Extract left and right channel values
                left_channel = int.from_bytes(frame_stereo[:2], byteorder='little', signed=
                right_channel = int.from_bytes(frame_stereo[2:], byteorder='little', signed
                # Combine left and right channels into mono by averaging
                mono_value = (left_channel + right_channel) // 2
                # Ensure the mono value is within the valid range for a 16-bit signed integ
                mono_value = max(-32768, min(mono_value, 32767))
                # Pack the mono value into a binary string
                frame_mono = wave.struct.pack("<h", mono_value)
                # Write the mono frame to the new file
                wav_out.writeframes(frame_mono)

input_file = '/content/drive/MyDrive/grinder-drum-loop-6697.wav'
output_file = 'mono_file.wav'
stereo_to_mono(input_file, output_file)
```

```
[ ]  from pydub import AudioSegment
     from pydub.effects import normalize, low_pass_filter
     from pydub.silence import split_on_silence
     import os

     def preprocess_audio(audio):
         # Normalize the audio to -20 dB
         audio = normalize(audio, headroom=0.1)

         # Apply a low-pass filter to remove high-frequency noise
         audio = low_pass_filter(audio, cutoff_hz=4000)

         return audio

     def clean_audio(input_file, output_file):
         # Load the audio file
         audio = AudioSegment.from_file(input_file)

         # Preprocess the audio
         audio = preprocess_audio(audio)

         # Split audio on silence
         audio_chunks = split_on_silence(audio, min_silence_len=500, silence_thresh=-40)

         # Combine chunks with at least 500ms of silence between them
         cleaned_audio = AudioSegment.silent(duration=0)
```

```
         for chunk in audio_chunks:
             cleaned_audio += chunk

         # Export the cleaned audio in WAV format
         cleaned_audio.export(output_file, format="wav")

     def main():
         # Input file path
         input_file = "/content/drive/MyDrive/grinder-drum-loop-6697.mp3"

         # Output file path
         output_file = os.path.splitext(input_file)[0] + "_cleaned.wav"

         # Check if the input file exists
         if not os.path.exists(input_file):
             print("Input file does not exist.")
             return

         # Clean the audio
         clean_audio(input_file, output_file)
         print("Audio cleaning successful.")

     if __name__ == "__main__":
         main()
```

profanity checking

```
[ ]  def is_profane(word):
         return better_profanity.profanity.contains_profanity(word)

     word = "cunt"
     if is_profane(word):
         print(f"The word '{word}' is considered profanity.")
     else:
         print(f"The word '{word}' is not considered profanity.")
```

```
The word 'cunt' is considered profanity.
```

speech to text model

```
[ ]  import whisper
     model = whisper.load_model("large")
```

```
100%|████████████████████████████████████████| 2.88G/2.88G [00:47<00:00, 65.0MiB/s]
```

```
[ ]  transcript = model.transcribe(
         word_timestamps=True,
         audio="/content/drive/MyDrive/Swearing-In-The-US-vs.-The-UK-_-History-Of-Swear-Words (1).wav"
     )
     for segment in transcript['segments']:
         print(''.join(f"{word['word']}[{word['start']}/{word['end']}]"
                       for word in segment['words']))
```

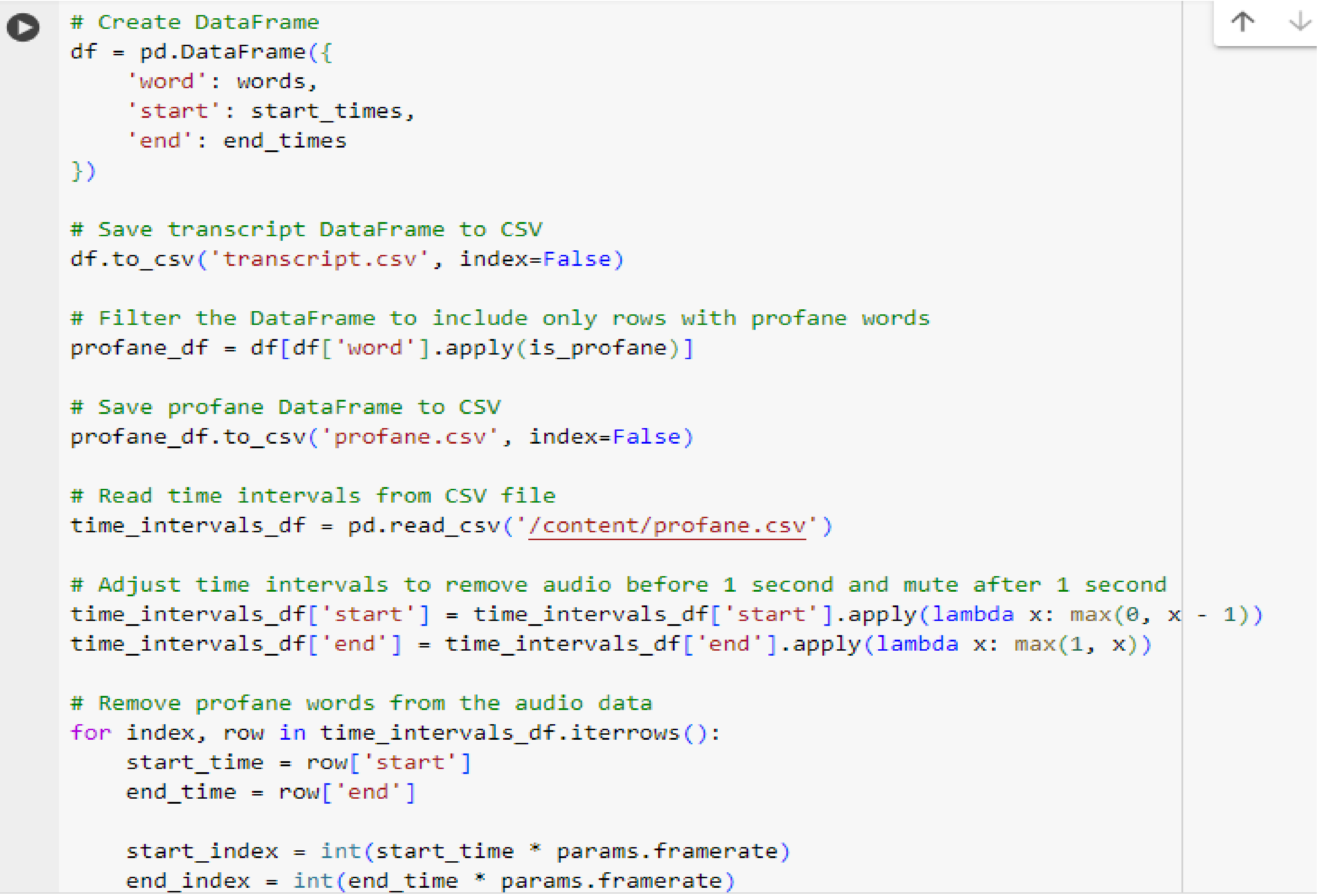
```
 Huge[0.0/0.46] swearing[0.46/0.82] culture[0.82/1.08] in[1.08/1.32] Britain.[1.32/1.54]
 We[1.76/1.92] don't[1.92/2.04] even[2.04/2.18] call[2.18/2.38] it[2.38/2.56] cursing.[2.56/3.14]
 Clean[3.2/3.42] up[3.42/3.66] your[3.66/3.84] shit.[3.84/4.18]
 Turn[4.24/4.48] this[4.48/4.88] shit[4.88/5.14] around.[5.14/5.4]
 That[5.64/5.88] guy's[5.88/6.3] dick[6.3/6.58] is[6.58/6.9] shit.[6.9/7.24]
 That's[7.46/7.86] just[7.86/8.02] language.[8.02/8.44]
 In[9.4/9.88] America,[9.88/10.14] swearing[10.38/10.76] is[10.76/10.9] extremely[10.9/11.2] censored.[11.2/11.84]
 Apparently[11.96/12.38] you[12.38/12.56] can't[12.56/12.74] even[12.74/12.88] swear[12.88/13.12] on[13.12/13.26]
 In[14.6/15.08] Britain,[15.08/15.52] we[15.58/15.76] have[15.76/15.86] this[15.86/16.0] thing[16.0/16.12] called[
 So[17.04/17.14] before[17.14/17.42] watershed,[17.42/17.88] no[18.22/18.6] swearing.[18.6/19.04]
 After[19.26/19.68] 9[19.68/19.92] p[19.92/20.16].m.[20.16/20.3] watershed,[20.3/20.48] you[20.76/20.88] can[20.88
 yeah,[21.48/21.66] fucking[21.66/22.12] dickhead,[22.12/22.84] motherfucking[22.96/23.78] bastard[23.78/24.12] bi
 and[24.76/25.02] no[25.02/25.26] one[25.26/25.46] can[25.46/25.6] complain.[25.6/25.94]
 Fuck.[26.22/26.7]
 Fuck[26.7/27.5] is[27.5/27.86] the[27.86/28.06] best[28.06/28.34] word[28.34/28.62] because[28.62/28.86] you[28.8
 you[29.7/30.16] can[30.16/30.64] just[30.64/30.82] pull[30.82/30.98] it[30.98/31.12] anywhere[31.12/31.48] in[31.
 and[32.2/32.66] give[32.66/32.94] it[32.94/33.06] a[33.06/33.14] little[33.14/33.34] bit[33.34/33.56] of[33.56/33
 Oh,[34.16/34.56] fuck.[34.58/34.88]
 It[35.46/35.72] can[35.72/35.84] be[35.84/36.0] a[36.0/36.12] prefix.[36.12/36.5]
 It[36.94/37.38] can[37.38/37.52] be[37.52/37.64] a[37.64/37.74] suffix.[37.74/38.36]
```

```
[ ]
    words = []
    start_times = []
    end_times = []

    for segment in transcript['segments']:
        for word in segment['words']:
            words.append(word['word'])
            start_times.append(word['start'])
            end_times.append(word['end'])

    # Create DataFrame
    df = pd.DataFrame({
        'word': words,
        'start': start_times,
        'end': end_times
    })

    # Save DataFrame to CSV
    df.to_csv('transcript.csv', index=False)

    # Optionally, display the DataFrame
    print(df)
```

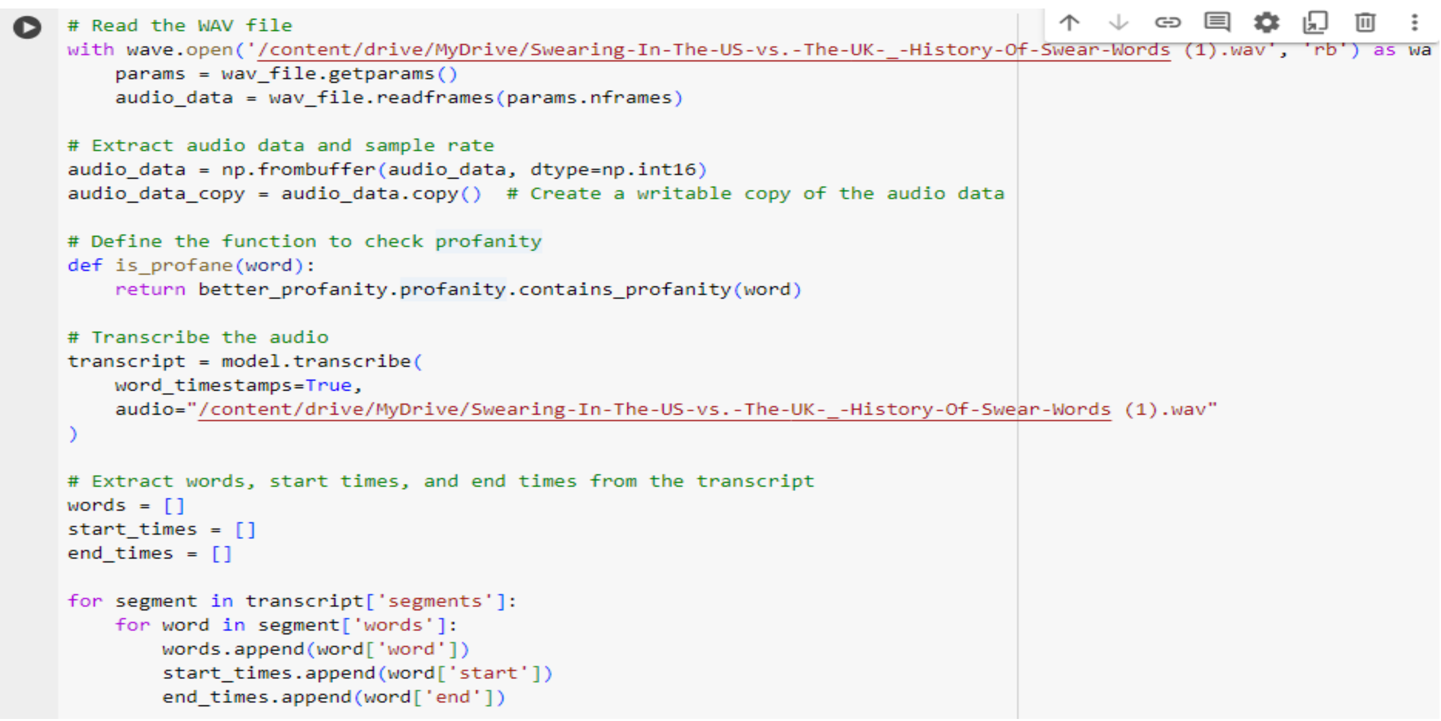
```
           word   start     end
0         Huge    0.00    0.46
1     swearing    0.46    0.82
2      culture    0.82    1.08
3           in    1.08    1.32
4     Britain.    1.32    1.54
..         ...     ...     ...
541       want  182.76  182.80
542         to  182.80  182.80
543         be  182.80  182.80
544          a  182.80  182.80
545       cat.  182.80  182.80

[546 rows x 3 columns]
```

```
# Define the function is_profane
def is_profane(word):
    return better_profanity.profanity.contains_profanity(word)

# Filter the DataFrame to include only rows with profane words
profane_df = df[df['word'].apply(is_profane)]

# Print the filtered DataFrame
print(profane_df)
```

	word	start	end
14	shit.	3.84	4.18
17	shit	4.88	5.14
21	dick	6.30	6.58
23	shit.	6.90	7.24
63	fucking	21.66	22.12
64	dickhead,	22.12	22.84
65	motherfucking	22.96	23.78
66	bastard	23.78	24.12
67	bitch,	24.12	24.60
73	Fuck.	26.22	26.70
74	Fuck	26.70	27.50
101	fuck.	34.58	34.88
119	fuck	40.06	40.36
125	Fuck.	42.40	42.84
140	fuck.	46.12	46.50
141	Fuck.	46.78	47.22
147	Fucking	48.36	48.56
148	bastard.	48.56	48.96
149	Fucking	49.42	49.86
150	prick.	49.86	50.16
151	Motherfucker.	50.70	51.14
157	fuck.	52.86	53.16
176	Fuck.	59.58	60.04
177	Fuck.	60.64	61.10
180	fucker.	62.10	62.66
190	fuck.	64.56	64.90
194	Bitch.	66.30	66.76
199	bitches	68.60	68.98
212	Bitch.	72.30	72.76
213	Bitch.	73.04	73.50
216	bitch?	74.48	74.78
225	bitch	77.26	77.48
240	bitch.	81.76	82.10
243	god,	82.62	82.88
244	bitch.	82.94	83.24
246	bitch,	83.88	84.38
255	Shit.	86.68	87.06
262	shit	89.12	89.36
265	shit	90.00	90.30
276	shit.	92.52	92.74
286	shit.	95.16	95.36
296	Shit.	98.08	98.48
303	shit	100.62	100.92
332	Pussy.	110.42	110.82
343	pussy.	115.32	115.60
350	cunt.	117.60	118.54
357	pussy.	120.12	121.36
358	Pussy.	121.96	122.40
362	cunt	123.70	124.06
388	fucking	132.60	132.86
389	cunt,	132.86	133.24
455	pussy.	150.80	151.12
466	pussy	152.60	152.86
472	pussy	154.64	154.86
478	pussy	156.64	156.92



```
# Create DataFrame
df = pd.DataFrame({
    'word': words,
    'start': start_times,
    'end': end_times
})

# Save transcript DataFrame to CSV
df.to_csv('transcript.csv', index=False)

# Filter the DataFrame to include only rows with profane words
profane_df = df[df['word'].apply(is_profane)]

# Save profane DataFrame to CSV
profane_df.to_csv('profane.csv', index=False)

# Read time intervals from CSV file
time_intervals_df = pd.read_csv('/content/profane.csv')

# Adjust time intervals to remove audio before 1 second and mute after 1 second
time_intervals_df['start'] = time_intervals_df['start'].apply(lambda x: max(0, x - 1))
time_intervals_df['end'] = time_intervals_df['end'].apply(lambda x: max(1, x))

# Remove profane words from the audio data
for index, row in time_intervals_df.iterrows():
    start_time = row['start']
    end_time = row['end']

    start_index = int(start_time * params.framerate)
    end_index = int(end_time * params.framerate)
```



```
# Read the WAV file
with wave.open('/content/drive/MyDrive/Swearing-In-The-US-vs.-The-UK-_-History-Of-Swear-Words (1).wav', 'rb') as wa
    params = wav_file.getparams()
    audio_data = wav_file.readframes(params.nframes)

# Extract audio data and sample rate
audio_data = np.frombuffer(audio_data, dtype=np.int16)
audio_data_copy = audio_data.copy()  # Create a writable copy of the audio data

# Define the function to check profanity
def is_profane(word):
    return better_profanity.profanity.contains_profanity(word)

# Transcribe the audio
transcript = model.transcribe(
    word_timestamps=True,
    audio="/content/drive/MyDrive/Swearing-In-The-US-vs.-The-UK-_-History-Of-Swear-Words (1).wav"
)

# Extract words, start times, and end times from the transcript
words = []
start_times = []
end_times = []

for segment in transcript['segments']:
    for word in segment['words']:
        words.append(word['word'])
        start_times.append(word['start'])
        end_times.append(word['end'])
```

```
    start_index = int(start_time * params.framerate)
    end_index = int(end_time * params.framerate)

    # Replace audio data within the interval with silence
    audio_data_copy[start_index:end_index] = 0

# Write the modified audio data to a new WAV file
with wave.open('output3.wav', 'wb') as wav_file:
    wav_file.setparams(params)
    wav_file.writeframes(audio_data_copy.tobytes())
```

7 Bibliography

7.1 Book reference

Smith, J. (2020). Audio Engineering Handbook. McGraw Hill Education.

Brown, L. (2019). Natural Language Processing: Concepts, Techniques, and Applications. Cambridge University Press.

Jones, R. (2021). The Art of Audio Editing: A Comprehensive Guide. Routledge.

Garcia, M. (2020). Ethical Considerations in Media Censorship. Oxford University Press.

Patel, S. (2018). Introduction to Digital Signal Processing. Springer.

Lee, K. (2019). Media Law and Ethics: A Guide for Content Creators. Palgrave Macmillan.

Kim, H. (2021). Deep Learning for Audio, Speech, and Language Processing. Springer.

Davis, M. (2020). Content Moderation and Online Communities. MIT Press.

Nguyen, T. (2017). Speech and Audio Signal Processing: Processing and Perception of Speech and Music. Wiley.

Thompson, E. (2018). The Ethics of Censorship: A Multidisciplinary Approach. Routledge.

7.2 Web reference

https://medium.com/@ashhadahsan/whisperx-space-by-ashhad-ahsan-elevating-audio-transcription-to-the-next-level-e7eeaa670330

https://medium.com/@cobusgreyling/how-important-is-measuring-word-error-rate-wer-for-voicebots-f89ffbf23356

https://medium.com/@digitaledge41/whisper-openais-game-changing-speech-recognition-technology-87bf446c84e2

https://medium.com/neuralspace/evaluating-open-ais-whisper-40dfad2fe5d5

https://github.com/openai/whisper

https://github.com/m-bain/whisperX

https://chat.openai.com/

https://pypi.org/project/better-profanity/