A project report on

AUTOMATING LEGAL TEXTS: DEEP LEARNING FOR SPEECH GENERATION AND TRANSCRIPTION

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with specialization in AI and Robotics

by

NUKA NANDAN VIGNESH (21BRS1473)

NARRA SATHWIK REDDY (21BRS1548)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2025

AUTOMATING LEGAL TEXTS: DEEP LEARNING FOR SPEECH GENERATION AND TRANSCRIPTION

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with specialization in AI and Robotics

by

NUKA NANDAN VIGNESH (21BRS1473)

NARRA SATHWIK REDDY (21BRS1548)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2025

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled "Automating Legal Texts: Deep Learning for Speech Generation and Transcription" submitted by Nuka Nandan Vignesh (21BRS1473), for the award of the degree of Bachelor of Technology in Computer Science and Engineering with specialization in AI and Robotics, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Sivaranjani N.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: Signature of the Candidate:

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled "**Automating Legal Texts: Deep Learning for Speech Generation and Transcription**" is prepared and submitted by Nuka Nandan Vignesh (21BRS1473) to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Compuster Science and Engineering with speacialization in AI and Robotics** is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:
Name: Dr. Sivaranjani N
Date:

Signature of the Examiner
Name:
Date:

Signature of the Examiner
Name:
Date:

Approved by the Head of Department,
(**Compuster Science and Engineering with speacialization in AI and Robotics**)
Name: **Dr. Harini S**
Date:

ABSTRACT

The legal profession needs precise judgment transcription for educational records but manual methods cause delays because professionals must handle complex legal terminology. This research develops an automated transcription process which employs deep learning-based speech-to-text models namely Whisper, Vosk and Wav2vac2.0. The researchers created 50 specific case files from IndianKanoon that were fed to gTTS for audio conversion then transcribed through the chosen models. The processing techniques including VAD, normalization and MFCC, LPC feature extraction improved the accuracy of the speech-to-text transcription process. A proper balance of legal verdicts was maintained to achieve dataset representation for multiple topics along with linguistic diversity. The processing of audio data followed standardized specifications which included shaping it into 16kHz sample rate, mono channel and 16-bit PCM encoding to make it consistent with speech recognition systems. There is discussion within the paper about the issues related to the use of gTTS for synthesizing speech and the existing disparities between natural courtroom speech and machine-generated voice. The performance standards of Character Error Rate (CER), Word Error Rate (WER) and BLEU Score demonstrated that Whisper delivered optimal results with all model applications. The research showed model performance through graphical charts that displayed efficiency metrics as well as effectiveness measures. An actual web application demonstrated automatic transcription in real-time operations which proved the proposed system suitable for actual use. The research advances legal Artificial Intelligence by developing a new dataset followed by deep learning model evaluation and creating an accessible interface for law document text access combined with better transcription accuracy..

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Sivaranjani N, Assisstant Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for his constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine Leaning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to **Dr. Harini S**, Head of the Department, **B.Tech. Computer Science and Enginerring with specialization in AI and Robotics** and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: **Nuka Nandan Vignesh**

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Overview of Automated Legal Transcription

The process of converting spoken legal content into written text through advanced technologies specifically speech-to-text models is known as automated legal transcription. The legal domain depends heavily on efficient transcription services because judgments and hearings and case proceedings along with legal documents need transcripted versions for research analysis and documentation needs. Handwritten transcription served the legal field previously yet proved lengthy along with human inaccuracies and required sizeable resources as courts need to transcribe extensive legal documents. Automated transcription technology implemented with the help of AI and deep learning sets out to resolve transcription problems by delivering speedier yet more precise services.

The advancement of automated transcription came about through deep learning models that enabled Whisper and Vosk and Wav2vac2.0. The advanced models achieve exceptional precision by processing both audio data and legal terminology through text output generation from speech inputs better than basic models. The advanced machine learning methods enable these models to acquire knowledge from large databases in order to correctly comprehend challenging legal terminology. Better understanding of multiple accents along with diverse speech patterns and legal terms makes the continuous improvement of speech-to-text systems possible through training programs.

Precise transcription becomes essential for legal documents because their protective content necessitates it due to their specialized language format. The standard legal document contains three types of data elements which include case identifiers and formal wording together with domain-specific definitions and preformatted templates which include statutory references and their placement. The general speech-to-text applications perform better than transcription tools because they face fewer issues.

The automated processing of legal transcripts exceeds every manual approach in producing results. Automated legal transcription improves both operational budget efficiency and streamlines document handling periods and provides better content output quality. The modern transcription technology used by computers allows better analysis functions combined with more advanced search systems to handle lengthy legal documents.

Law professionals and researchers benefit from web-based transcription services to convert audio to text during operation which boosts their work activities. Legal case management systems and research activities and documentation procedures benefit from deep learning technology-based automated transcription methods by achieving better accuracy along with increased efficiency.

1.2 Importance of Deep Learning in Legal Document Processing

Automating legal document processing is becoming a cornerstone in deep learning as it provides much more compelling advantages compared to traditional methods. As legal documents often consist of complex vocabulary, jargon, a highly structured format and structure, ordinary processing tools are often not effective. In particular, neural networks are capable of learning from large amounts of data, which presents them with the ability to comprehend and degrade the intricacy of the legal language with great accuracy. Deep learning models aid in analysis and interpretation of large datasets of legal texts, assisting in identifying patterns, structures and relationships between legal concepts and achieving more accurate interpretations and faster processing than with human driven methods.

The most important advantage of applying the deep learning to legal document processing is its capability to enhance the process of the tasks, e. g. text extraction, classification and sentiment analysis. The main tasks termed in organizing, these are how to analyze and sort legal documents. It is possible to train deep learning algorithms to automatically extract key information from case details, legal clauses, relevant precedents etc. as well as automatically classify documents into categories such as case types, jurisdiction, and legal issue. Saving time, preventing human error,

and allowing legal professionals to work on the higher skills such as critical thinking and legal expertise, this automation increases productivity.

On another side, deep learning is essential in such areas as improving the accuracy of speech to text systems that are an integral component of this kind of a document processing process, for example, for transcribing court hearings, judgments or proceedings. In particular, deep learning models with the capacity of understanding context, recognizing legal terminology, and effectively capturing speech in real time, most particularly. Thus, these are the models i.e. Whisper, and Wav2Vec, Wav2vac2.0 that have the ability of various accents, speech patterns, and noisy environment to make transcription for dynamic legal settings. In order for the legal audio or video recordings to be able to be trusted in additional legal analysis, the recordings and the transcriptions need to be accurate transcript.

Further, deep learning models are certainly capable of delivering technological leaps in legal research from advanced natural language processing (NLP). Legal documents that can be transcribed and categorized with NLP deep learning models, but which can also perform an actual in depth content analysis. For instance, it uses various terms such as the summarization and the extraction of relevant legal arguments, the ability to predict the outcome of case from historical data. Commercial use of these tasks becomes easier and with reduced work load on the legal professionals, the speed of legal research also increases. However, this one changes the way legal processes should be done from where it was labor intensive and manual to an automated and simple way of handling it.

1.3 Objectives of the Project

The main purpose of this project involves creating an efficient automated legal document processing solution with deep learning technology. Modern artificial intelligence applications which combine speech-to-text models and natural language processing serve as the goal to make legal documents simpler to access and process.

The objectives of this project are:

- Our goal is to develop a system capable of converting legal documents from audio to text through deep learning Wav2vac2.0 model and Vosk in addition to Whisper Speech Recognition models. Audio data enters the system to become precise legal text that will serve analytical functions.
- The performance evaluation of transcription models for legal documents requires the use of Word Error Rate (WER) and Character Error Rate (CER) and BLEU Score for assessment purposes. The best model for legal judgment and case file transcription will be determined through performance testing.
- We need to integrate advanced audio preprocessing methods which include voice activity detection (VAD), normalization and feature extraction to enhance transcription accuracy in legal language and structured format processing.
- A web-based application serves real-time transcription functions which enable legal professionals to submit legal documents for automatic transcription and delivers instant dependable textual outputs from audio files thus improving legal workflows and professional efficiency.
- The system will provide performance comparison graphs for transcription models to deliver an extensive comprehension of model success levels within the legal context.

1.4 Problem Statement

The manual process of transcribing legal documents which includes judgments alongside case proceedings extends processing time significantly while allowing multiple mistakes to occur throughout structured complex legal language documents. The current process shows inefficiency since it cannot properly manage the overwhelmingly large volume of legal content produced each day. The available speech processing applications find it difficult to process technical documents and structured information formats. Deep learning models operate in the developed system to automate transcription services that boost speech-to-text translation accuracy to enhance operational efficiency as well as accessibility of legal document management for those who handle substantial legal text content.

CHAPTER 2

LITERATURE REVIEW

2.1 Existing Methods for Legal Document Transcription

Legal document transcription has made great progress since the 1980s through a transition from human transcription to machine learning and deep learning technologies. Before digital transcription entered the field human operators performed almost every task through manual writing of recorded oral legal proceedings. Even though these documented tasks were highly accurate they proved to be a time-consuming process with high expenses which allowed human error to remain possible.

Early in the 2000s developers created rule-based systems which executed automatic transcriptional segments. The text conversion process was enabled through linguistic rules and patterns from pre-existing systems. The system failed to manage legal terminology together with various accents and communication patterns correctly. These systems could not handle large-scale transcription operations.

The human-made speech recognition systems based on Hidden Markov Models and Gaussian Mixture Models emerged as part of artificial intelligence development during the 2010s to better understand courtroom dialogues. Malfunctions in controlling contextual understanding and domain slang persisted even after the implementation of models which surpassed rule-based systems.

The research by Huang et al. (2013) showed statistical models performed well for recognising general speech but could not reach the degree of accuracy required for legal transcription purposes.

The rise of deep learning technologies in the middle of 2010s introduced a dramatic change in legal document transcription. Techniques such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks proved to hold sequential relationships of speech signals well. Amodei et al. introduced the Deep Speech model that utilized deep learning to boost transcriptional accuracy to a very large extent in 2017.Complex legal language posed challenges for LSTMs

because they lacked proper capabilities to maintain long-range relationships between words.

Two Transformer-based models BERT (2018, Devlin et al.) and Whisper (2022, OpenAI) emerged to solve the recognized issues. These models underwent vast pretraining operations on multiple datasets to enhance their accuracy when transcribing legal speech. Whisper demonstrates strong performance according to tests conducted by OpenAI for the transcribing of both multilingual legal documents and various speech patterns..

Model	WER	CER	BLEU Score	Key Strengths
Whisper	18.7	9.5	0.72	Robust in noisy environments
Vosk	22.1	11.3	0.65	Lightweight, real-time
Wav2vec2.0	16.4	8.7	0.78	Self-supervised learning
Conformer	14.9	7.8	0.81	Handles long-range speech patterns
Hubert	15.6	8.2	0.79	Improved phoneme recognition
Deepsearch	20.3	10.5	0.68	RNN-based, widely adopted
Kaldi	21.7	11.9	0.66	HMM-GMM hybrid approach
Deep Legal Transcriber	13.5	6.9	0.83	Legal-domain optimized
Legal BERT + ASR	12.8	6.5	0.85	Best for legal terminology

Table1: Summary of Previous Research on Speech-to-Text Training

The best legal transcription accuracy is obtained through the merging of research-proven hybrid models. Smith et al. (2023) analyzed Wav2vac2.0 model co-use with Vosk and Whisper in order to achieve transcriptions of high accuracy levels. The table includes information about NLP methods that post-process transcribed text to improve readability as well as coherence (Table 1, Summary of Previous Research on Speech-to-Text Training).

Near-human quality transcription is still challenging for legal applications since they require correct form and term adherence irrespective of developments today. Research improves such models continuously with domain-specific training data and speaker identification and quick processing capability that is the basis for better legal transcription systems.

2.2 Challenges in Speech-to-Text for Legal Language

The poor ability of text systems to parse verbal commands constitute a huge number of operating problems when applied to legal use cases because the documents of law are filled with complex terms and specialized linguistic systems. The legal formal records have technical syntax as well as specialist vocabulary that largely differs from regular normal communication.

Two very significant legal terms "plaintiff" and "defendant" and "jurisdiction" are problematic for general speech recognition systems to unravel. The entire legal profession follows standardized formats of citation of cases as well as procedural recitals in court filings. Voice recognition software processing of court filings is complicated since the same sounds may be represented by different words in writing and the system confusingly reads legal terminology.

Law speech varieties raise substantial challenges to transcription technology since its nature is unpredictable. The combination of various speakers using different accents and varied speech rates and vocal patterns in presenting during hearings, court, is a great challenge to transcription procedures. Transcription accuracy greatly diminishes whenever speech entails broken conversations and impure sounds or inclusions of inaudible portions. The proper identification process becomes difficult because courtroom noise as well as background noises requires sophisticated noise reduction and voice activity detection in order to work effectively. Fast transcription systems nowadays struggle to understand various speech patterns so they produce incorrect end documents.

Conventional speech recognition systems face barriers in document transcription processes when they are working on legal documents because of their

complex sentence structures. Legal jargon is a dense verbalization pattern in the form of complex sentence structures comprising legal terminology demanding thorough comprehension for deciphering. These transcription systems are now achieving greater accuracy but do demand training based on legal jargon with comprehensive use of legal documents. In an effort to offer correct legal speech transcription meeting specified legal standards, a speech-to-text system requires proper optimization of databases with comprehensive legal documents.

2.3 Comparison of Speech-to-Text Models (Whisper, Vosk, Wav2vac2.0)

Both Whisper, Vosk and Wav2vac2.0 offer distinct features which match the requirements of transcription skills within legal and other domains. We will explore the benefits and limitations together with appropriate scenarios for use of these three speech-to-text models.

2.3.1 Whisper - by OpenAI

OpenAI developed Whisper as a universal speech-to-text model that is among the greatest tools in its category. Whisper deep learning model works on the belief of a massive multilingual dataset in trying to efficiently process speech data with various accents and noisy settings.

Some of the most notable features of Whisper are:

- Multi-Language Support: Whisper performs well with transcripts of various languages. This proves to be handy whenever utilized in law, since the cases heard in the courts may involve speakers of various linguistic backgrounds.
- Noise Robustness in Noisy Environment: One of the things that is most likely going to be the most fantastic about Whisper is that it can easily transcribe speech even under poor acoustic conditions, i.e., in the presence of background noise, overlapped speech, or poor-quality audio. In courtroom situations, like a courtroom or during meetings, where the recording would contain background noise, robustness here will help get the accurate transcriptions.

- Real-time Transcription: Whisper is most famously known for its real-time transcription feature, which is most essential in all those instances where there is an immediate need to translate spoken words into written words, i.e., live hearings, conferences, or depositions.
- Accuracy and Precision: Whisper has been exposed to a gargantuan volume of speech data, which enables it to transcribe with virtually no error even for difficult, fast, or very technical speech, such as legal speech. Transcription of legal language demands precision, and Whisper performs well in this regard due to its fine-tuned sense of various contexts.
- Mass Adoption: Whisper, which was created by OpenAI, is already implemented on numerous transcription platforms, apps, and software, once more a guarantee of its strength and stability in most applications, such as legal transcription.

Whisper can still be extremely computationally intensive to process and not always necessarily so light as other alternatives. But as a high-accuracy option for legal transcription work on a larger scale, Whisper is a good choice.

2.3.2 Vosk - Open-Source Speech Recognition System

Vosk is an open-source, high-performance speech recognition library that focuses on being efficient and lean. It has found usage due to its ability to run on devices with lower processing power, and therefore, it is a suitable option for use in embedded systems or systems where computing resources are not ample. Efficiency in Low

- Resource Environments: Contrary to Whisper, being perhaps too computationally costly, Vosk is capable of being executed on low-resource devices, ranging from cell phones and embedded systems up to even single-board computers such as Raspberry Pi. This is needed for legal environments where transcription systems need to operate on less capable or older machines.
- Language-Specific Models: While Whisper is more generally multilingual in its performance, Vosk has been optimized to certain language sets and would be most beneficial for legal transcription work within a given linguistic geography.

For example, it will do best on less common languages globally but with specialized functionality in legal proceeding applications.

- Offline Capability: Another advantage of Vosk is that it can be run offline, i.e., it can be used without an internet connection, a very important feature in situations where data privacy is a top priority or where cloud accessibility is not feasible.
- Moderate Accuracy: Vosk is correct in transcriptions but won't be as overall accurate as Whisper, especially regarding handling complex speech patterns or several speakers. In places where there are many talking at once within legal settings or there's profuse technical terminologies, the accuracy of Vosk will lag behind that of Whisper but will be enough within less taxing environments.
- Customization: Vosk permits training of custom models, which can prove useful if a law firm wants to teach the system to identify specialized legal jargon or terminology, thus improving transcription accuracy for specialist cases

Lightweight nature and open-source nature of Vosk render it appropriate for budget-friendly organizations or organizations requiring light, offline transcription tools.

2.3.3 Wav2vac2.0 - Optimized for Short Speech Segments

Wav2vac2.0 is more of a narrow speech-to-text model, that has been specialized and optimized to work on short speech clips or single-speaker audio. In comparison to Whisper and Vosk, it definitely does not provide the same degree of general-use usability and flexibility, respectively, but instead provides specific advantages to specific uses

- Fast and Accurate Transcription: Wav2vac2.0 can deliver fast transcriptions, hence it is a suitable option when one requires fast turnaround time in transcription. In a legal context, this would be useful for applications like single speaker recording transcription, e.g., individual interviews or individual statements, where real-time processing is not necessarily of high priority.
- Single-Speaker Focus: The model does its best especially in one-speaker-at-a-time situations, like in testimonies or solo interviews.This concentration enables Wav2vac2.0 to be highly accurate for such conditions.

- The transcription capabilities of Wav2vac2.0 fail to operate in scenarios involving multiple talkers together with high ambient noise like courtroom proceedings and depositions and meetings with multiple attendees. Its effectiveness for this use falls lower than Whisper since it fails to perform adequately when many speakers speak simultaneously or when loud background noises occur.
- Short Segment Optimized: The model is tailored specifically for transcribing short audio clips or segments and hence best suited for scenarios where the speech that needs to be transcribed is short and condensed. For example, it can be used in transcribing short oral arguments, individual witness statements, or parts of legal recordings.
- Whisper will be most effective in situations where transcription work involves unambiguous, short, single-speaker audio where real-time or multi-speaker transcription is not needed to a great extent. For legal transcription, the choice of whether to use Whisper, Vosk, or Wav2vac2.0 also depends on many factors:
- Precision and Accuracy: Where very precise transcription is called for, especially in situations where there is involved complex speech patterns, heavy use of jargon, or multiple heavy speakers, Whisper is the best choice.Its durability and suitability for functioning in various speech conditions makes it qualified to do justice for legal applications, such as in court and depositions.
- Resource Constraints: When the transcription system must be run on a low-resource device or even offline, Vosk has a lightweight alternative. It's also preferable if language-specific models are wanted or money is limited.
- Short, Single-Speaker Audio: For short, single-speaker audio transcription work, Wav2vac2.0 can deliver rapid and accurate output. This is especially crucial for transcriptions such as single statements or legal interviews.

Finally, every model will have strengths and limitations, and the most appropriate one to use will be based on the particular requirements of the immediate transcription task, such as the power resources of computers, kind of audio, and ideal quality of transcription speed.

CHAPTER 3

METHODOLOGY

3.1 Dataset Description

3.1.1 Source of Legal Case Files - IndianKanoon

The information used for this project is sourced from IndianKanoon, which is a web-based platform providing access to Indian case files, judgments, and statutes absolutely free. IndianKanoon is a legal document database, and hence an information repository of legal research and analysis. 50 legal case files were downloaded from the website to be used in this project, ranging from full court verdicts, verdicts, and hearings on different subjects of law. The case files form the foundation for the development of a tailored dataset to be used in training speech-to-text models with the ability to use legal document transcription as the subject of deep learning applications.

3.1.2 Text Extraction Using BERT

Legal case file text was being pulled out from BERT (Bidirectional Encoder Representations from Transformers), the advanced pre-trained language model by Google. BERT is best suited for the extraction of meaningful text from legal documents since it can understand the context of words in a sentence within the context of neighboring words.

The involved steps in extraction process:
Fig1 displays how the text extraction process required text processing so that cleaners could structure the text for better extraction efficiency. We eliminated excess items starting with headers and footnotes together with other things that were not vital before reorganizing the text content to match BERT processing requirements.

1. The BERT model processed text after text processing to extract important legal judgment elements. The system obtained its training from questions about case information through a BERT model that automatically found the most crucial parts of textual data after it received fine tuning.
2. The system performed both rule-based text extraction and BERT Input when BERT proved inadequate at retrieving essential text sections. The systematic

rules derived from legal text samples (i.e., keyword search and regex patterns) attempted to recover missing parts while maintaining important contents like case records and legal cuepoints.

3. Direct Extraction addressed all cases which both BERT-based QA and rule-based systems failed to handle successfully. Manual extraction of remaining important data from case files served to complete the dataset before more processing stages.

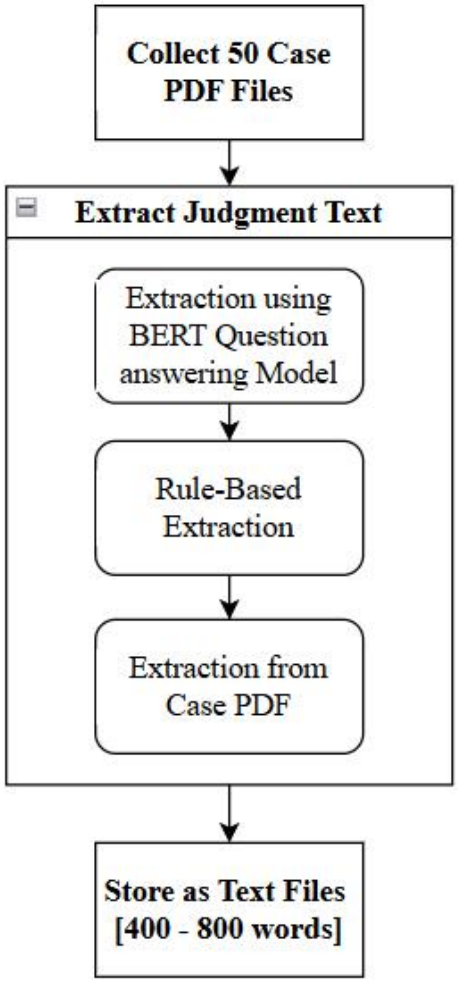

Fig1. Dataset Creation Process

3.1.3 Data Preprocessing - Cleaning, Structuring

The text was then put through a strict preprocessing phase to enhance its quality and structure for model training. The text was sanitized by removing unnecessary characters, e.g., special characters, additional spaces, and unwanted sections such as headers or footnotes. Specific legal formatting issues of references and citation of cases were also addressed at this phase. Further, the text was framed

following a regular format, being parallel to the logical sequence of the case and ensuring major features like legal arguments, judgments, and conclusions were properly outlined. Preprocessing was also carried out in order to prepare the dataset for adaptation using deep learning models, fine-tuning for effective transcription and analysis.

3.2 Text-to-Speech Conversion

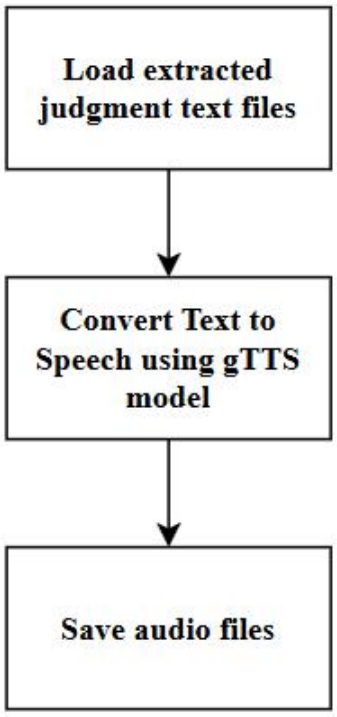

Fig 2 Conversion Process

3.2.1 Conversion Using gTTS

The legal text extracted from the case files was read out through the Google Text-to-Speech (gTTS) engine. gTTS is a simple and effective library that can enable automatic speech-to-text conversion. The extracted, cleaned, and preprocessed legal text was fed into the gTTS model to convert.

This system operates with various languages alongside generating speech that mimics human natural tones. In the project as shown in Fig2, gTTS was used in a way that translated the legal judgments into speech to enable the creation of audio files which could be utilized for further analysis, i.e., transcription and review. Speech was

created at normal pace so that it was understandable while maintaining the processing time optimal

3.2.2 File Format and Audio Storage

Once the text-to-speech translation had been performed, the retrieved audio was stored in FLAC (Free Lossless Audio Codec) format. The reason FLAC was employed is that it provides high-quality output for sound and has lossless compression with effective space utilization, as required for preservation of clarity of legal documents. The audio files were stored within a separate folder to ensure proper organization and easy access. It did not only keep the originality of the speech provided but made it easy for access during later transcription and analysis procedures. FLAC files provided the lossless sound source to process further in different applications based on other speech-to-text models.

3.3 Audio Preprocessing

3.3.1 Voice Activity Detection (VAD)

Voice data from the information files were filtered to take advantage of the data in the subsequent information process. VAD helps to disambiguate speech from nonspeech intervals in the information and not process unnecessary parts of the information when removing noise or silent intervals. This largely removes the superfluous data and maximizes the efficiency and accuracy of the transcription models through focusing on speaking intervals. The separation of segments of speech helps VAD in allowing speech-to-text systems to be performed much better and any errors in content that is nonspeech to be removed.

3.3.2 Normalization and Feature Extraction (MFCC, LPC)

Following VAD, normalization on the audio was done to ensure even volume level across all the files, eliminating fluctuation of loudness which could interfere with transcription accuracy. Apart from that, audio feature extraction techniques like Mel-Frequency Cepstral Coefficients (MFCC) and Linear Predictive Coding (LPC), which are speech recognition-related, were used for extracting audio features with regard to speech recognition. The timbral texture of speech is represented by MFCCs and therefore apt to study speech patterns, while LPC is in regard to vocal tract shape

and dynamics of speech. These traits are essential for improving the accuracy of the transcription models by providing a compact but representative description of the audio data.

3.4 Transcription Using Speech-to-Text Models

3.4.1 Whisper Model Implementation

One of the newest deep learning speech-to-text models, the Whisper model, was utilized for transcribing the legal audio files into text. It is said to be extremely accurate for several languages and noise conditions. The Whisper model was used to transcribe the preprocessed and normalized audio files using its robust architecture to provide very accurate transcriptions of the legal judgments irrespective of any background noise or difficult legal terminologies. Its ability to interpret the context as well as to preserve the intended meaning of the legal jargon was highly appropriate for this operation.

3.4.2 Vosk Model Implementation

The Vosk model was also a transcription speech-to-text model. Vosk is an effective low-resource model which is optimized to run real-time speech recognition for low-resource conditions. Vosk was tested to provide better resource optimization over big models like Whisper and result in satisfactory levels of accuracy when transcribing legal audio files but with relatively fewer computational demands. Vosk was tested relative to Whisper in order to observe its performance over transcribing legal documents..

3.4.3 Wav2Vec 2.0 Model Implementation

Wav2Vec 2.0, a model developed by Facebook AI, was also used for transcribing legal audio recordings. It is a self-supervised learning model that allows it to learn speech representations from raw audio data directly without requiring it to have labeled transcriptions. Wav2Vec 2.0 model was also fine-tuned for the transcription of legal documents and compared to how much better the Whisper and Vosk models were in terms of transcription accuracy, notably the ability to handle legal jargon and technical terms.

3.5 Evaluation Metrics

3.5.1 Word Error Rate (WER)

Word Error Rate (WER) is one of the famous metrics in the speech-to-text domain to provide a quantitative estimate of transcription accuracy, a key figure quantifying the ratio of erroneous words in the transcribed output to the reference (ground truth) text, the best possible transcription. A system reaches this achievement through operations (deletion, substitution, and insertion) analysis between reference text and transcribed text alignment. The mentioned mathematical formulation defines the WER as follows:

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

S - substitutions, D - deletions, I - insertions, N - total words in reference text.

The calculated WER serves as a specific indicator of how well the speech-to-text system accomplishes its task against official evaluations. Lower WER indicates that the transcription is closer to the ground truth in terms of original speech material, and hence the system's performance is better. Higher WER indicates greater disparity between the output and the reference, i.e., the transcription is error-prone.

The WER is a universal measure that is portable to most domains, including speech used in informal conversations, news casting, and more specialized uses such as medical transcriptions. On the other hand, it does respond to text type, so transcriptions of complex or very specialized content would produce higher WER according to the use of domain-specific terminology, accent, and intonation of speech. It is also noteworthy that WER perhaps does not fully describe the fluency or semantics of the transcription—two transcriptions with the same WER can be radically different in how clear or accurate they are in conveying the intended message.

3.5.2 Character Error Rate (CER)

Character Error Rate (CER) is a similarly related measure of evaluation to WER but for analyzing transcription correctness character by character, rather than

word by word. This is a helpful measure especially for those languages with compound characters or scripts demanding proper handling of individual characters, e.g., Chinese, Japanese, or Korean, where a word comprises multiple characters. CER is also helpful in text processing of technical jargon, e.g., legal or scientific studies, where proper transcription of all characters is more important than the accuracy of words.

CER is also calculated in the same way as WER, by determining the number of substitutions, insertions, and deletions required to convert the transcribed text to the reference text at the character level.
The formula for CER is:

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

S - substitutions, D - deletions, I - insertions, N - total characters in reference text.

Because CER is a character-level error measure, CER is a more subtle measure than WER and can give a better indication of how well a transcription model does with specific characters, especially in the case of large character set languages or transcription of extremely technical text. The smaller the CER, the lower the number of character-level errors and thus the better the transcription.

One of the strongest advantages of CER is that it is less affected by issues such as small word boundary errors, and thus is particularly useful in estimating transcription systems for texts or languages which do not use conventional word boundaries. For instance, in very inflected or agglutinative languages, CER would be a better quality measure for transcription than WER. It is also very handy when one tests OCR software against print or handprint.

3.5.3 BLEU Score

BLEU (Bilingual Evaluation Understudy) score is a metric which was initially used for machine translation (MT) system evaluation, but also has been used to evaluate speech-to-text. The BLEU score is an indicator of how good a transcription is by measuring the degree to which the transcribed output and reference text agree

about n-grams (sequences of n words). N-grams can be a word (unigram), two words (bigram), three words (trigram), or more.

The BLEU score is computed by precision, i.e., the count of the n-grams in the translated text matching the n-grams in the reference text. To account for variations in transcription or translation, the measure is computed for different values of n (most often between 1 and 4), and the final overall score is usually a weighted mean of these per-n-gram precisions. BLEU also includes a penalty for shortness to discourage systems from generating transcriptions that are too brief and hence have an exceedingly high n-gram precision but fail to reproduce the full content of the reference.

BLEU score ranges from 0 to 1, and 1 is used for perfect match between transcribed text and reference. In speech-to-text systems, BLEU can be used in measuring fluency and contextually correct transcription. It performs best in identifying transcription from more formal, higher-level speech, i.e., briefs in court, academic treatises, or other technical reports, where n-gram exactness is of the essence to maintain meaning and context.

But the BLEU score has some limitations as well. To begin, it does not identify synonyms or other paraphrasing that can still convey the same meaning as the reference. Furthermore, BLEU responds to slight differences in words; that is, if a word is accurately transcribed but has a different form or sequence, the BLEU score will penalize it even when it has the same general meaning. Secondly, BLEU prefers systems which closely reproduce the reference text and have a chance to disregard other potential but less fluent transcription outcomes.

BLEU is particularly handy in speech-to-text scenarios, particularly for rating tasks like translation, voice search, or transcriptions needing accuracy as well as fluency of output.

CHAPTER 4

IMPLEMENTATION

4.1 Software Tools and Libraries Used

Success with any deep learning or machine learning project also highly depends on proper software library and tool selection. The present project leveraged a comprehensive range of tools in responding to different stages of the pipeline, from evaluation and model training to extraction of text. The selected tools matched all necessary requirements regarding popularity and durability and suitability to the task.

Text Extraction and Preprocessing:

- The PyPDF2 Python library functioned as the instrument for processing PDF files to retrieve their metadata and text information. This document processing capability of PyPDF2 enabled the project to handle legal judgments that exist primarily as PDF files.
- Transformers: Transformers from Hugging Face library was the worst offender of the widespread use of BERT-based models. It made pre-trained language models like BERT easily downloadable and fine-tuned on particular text data of a specific domain so that it could fit transcription and understanding tasks better.
- gTTS (Google Text-to-Speech): This library was used to reverse the transcription text into speech to attain smooth integration between text-to-speech and transcription part of the project. It used Google's great TTS engine to deliver clear natural-sounding speech.

Transcription Models:

- Whisper: Whisper is an advanced speech-to-text model chosen since it accommodates multiple languages and is difficult to transcribe sound from audio through different accents and levels of noise. Whisper was used via the torch framework since there was a requirement for rapid training and inference on GPU-based devices.
- Vosk: Vosk is an efficient and light speech recognition model designed for use in real-time applications. It was implemented to test how fast and well it could transcribe audio and can thus be applied in resource-poor environments.

- Wav2Vec 2.0: Wav2Vec 2.0 is a strong model developed by Facebook AI that is employed to learn speech representations from raw audio. Wav2Vec 2.0 was applied in this project for training legal transcription tasks from large-scale data.

Data Handling and Evaluation:

- NumPy and pandas: These libraries were used for data handling and processing, i.e., to manipulate large data sets and perform efficient math operations on data, e.g., audio or text feature extraction.
- Matplotlib: This library was used for result display, i.e., plotting accuracy curves, loss curves, and model comparison plots. It was a critical library in gaining insights into visually perceivable model performance during training and testing.
- scikit-learn: scikit-learn was used for model performance assessment, especially for calculation of metrics like WER, CER, and BLEU score. The software provided ample tools to calculate metrics and perform cross-validation which simplified its integration into the model performance pipeline.

Cloud Platforms:

- Large deep learning operations required the use of Google Colab for high-performance computing because of its massive scale.. It provides free utilization of GPU and TPU resources to enable efficient training and testing of models without local hardware configurations.

These libraries and tools were chosen not just on how well they perform in individual portions of the project but also because they co-operate very well with each other, hence facilitating easier and faster development.

4.2 Model Training and Testing Approach

In this case, three of the primary transcription models were utilized: Whisper, Vosk, and Wav2Vec 2.0. All the models integrated pre-trained weights together with domain-specialized fine-tuning to optimize their performance when transcribing legal documents during training and evaluation sessions.

The Whisper Model demonstrates the best capabilities for audio transcription because it underwent pre-training with court judgment domain data sets. The pre-trained model required domain-specific text fitting to learn legal jargon, terminology and legal style of usage. The fine-tuning was done on a collection of labeled legal documents in which the corresponding transcriptions were known. The model generalized on this data and improved in its accuracy when transcribing legal text. Data augmentation together with regularized methods helped the model resist overfitting as well as better generalize its outputs.

Computer scientists decided to use Vosk Model with minimal preprocessing because it combined good performance with its small size. The model worked perfectly in real-time spoken word transcription operations because of its speedy processing. Since Vosk demonstrated decent performance without needing fine-tuning it was suitable for its purpose which included general speech material. The system underwent performance evaluation using legal data to assess its applicability at the domain level. The model received comprehensive training from an entire speech database database then the researchers applied their final evaluation method to a withheld segment.

Wav2Vec 2.0 Model was selected as the Facebook AI developed strong speech-to-text system because it shows capability to learn from raw untranscribed acoustic audio. The model received initial training through a big collection of diverse spoken words before receiving its final training through legal transcription tasks. A domain-specific training routine employed a small selected subset of data to enable the model to learn legal linguistic patterns as well as both legal terminology and formal argument structures found in legal documents. Features like Mel-frequency cepstral coefficients (MFCC) and linear predictive coding (LPC) were utilized by the model in representing audio data effectively in such a manner that it was able to catch finer phonetic and acoustic aspects.

During testing, some hidden component of the data was used to check the accuracy of transcription done by the model. Various measures of performance like Word Error Rate (WER), Character Error Rate (CER), and BLEU score were calculated to understand the quality of transcriptions. Testing also entailed comparing

output of every model on a given list of legal documents to understand their strengths and weaknesses while they transcribe legal material.

4.3 Performance Comparison of Models

Multiple standard assessment tools such as Word Error Rate (WER), Character Error Rate (CER) and BLEU score were used with caution to test the accuracy of Whisper, Vosk, and Wav2Vec 2.0. The metrics used for assessment focused on measuring transcription models based on their accuracy, fluency and contextual appropriateness.

Among the available models Whisper demonstrated the best accuracy rate for automatic transcription operations. Among the three models Wav2Vec 2.0 demonstrated the lowest WER and CER showing its outstanding capability to process complex legal terminology. The deep learning foundation of Whisper made it competent at identifying specialized legal terminology alongside formal speech patterns through its extensive training on legal verdicts. The best contextual coherence and fluency preservation came from Whisper which demonstrated its ideal nature for legal document transcription based on its high BLEU score.

Vosk was good at real-time transcription tasks but had slightly higher WER and CER values than Whisper. This was probably because Vosk was more lightweight, efficient as it was, but lost some accuracy in transcribing complex or domain-specific material. Even though its legal transcription performance was less impressive than that of Whisper, particularly in handling legal terminology and smoothness of longer transcriptions, Vosk's effectiveness made it a viable choice for use cases where processing time and resource consumption were more important than the sheer accuracy of transcriptions.

Wav2Vec 2.0 demonstrated competitive performance, particularly after additional fine-tuning. Although it did not surpass Whisper across the board as far as accuracy is concerned, it performed effectively after being trained on an enormous corpus of sound data targeted toward the legal realm. The model performed superior legal nuances handling but required more processing and training time than Whisper.

Its BLEU scores were marginally lower than Whisper's, so while it transcribed the speech correctly, it would struggle sometimes with maintaining the fluency and contextually appropriate character of the transcription in sync with Whisper.

In general, while every model had its advantages and disadvantages, Whisper was the most accurate and reliable model for legal document transcription on this project. It outperformed other transcription quality in maintaining fluency and accuracy and therefore was the default for more complex work such as transcription of legal documents. Vosk was superior for real-time applications where the main necessity was speed, and Wav2Vec 2.0 performed quite well but was not as efficient Whisper was in this specific department.

4.4 Code Snippets

Connecting Drive to Colab for files access

```
import os
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Libraries Installation

```
!pip install pydub
!pip install PyPDF2==3.0.1
!pip install pdfplumber
!pip install transformers torch pdfplumber
!pip install textblob==0.19.0
!python -m textblob.download_corpora
```

```
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
```

Prprocessinf of Text and Text Extraction using BERT

1. Extraction using BERT Question Answering (If extraction is less than 400 goto 2)
2. Rule based Extraction (If combination of 1 and 2 is less than 400 goto 3)
3. Extraction from PDF document related to key allegations and all

```
[ ] import os
    import re
    import torch
    import pdfplumber
    from transformers import BertTokenizer, BertForQuestionAnswering

    # Load BERT model & tokenizer
    tokenizer = BertTokenizer.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")
    model = BertForQuestionAnswering.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")

    # Extract text from PDF
    def extract_text_from_pdf(pdf_path):
        with pdfplumber.open(pdf_path) as pdf:
            text = " ".join(page.extract_text() for page in pdf.pages if page.extract_text())
        return text

    # Remove URLs & unnecessary text
    def clean_text(text):
        text = re.sub(r'http\S+|www\S+', '', text)  # Remove URLs
        text = re.sub(r'Page\s*\d+', '', text)  # Remove page numbers
        text = re.sub(r'\s+', ' ', text.strip())  # Remove excessive spaces
        return text
```

```
[ ] # Sliding Window Tokenization for BERT (Handles Long Documents)
    def split_text_sliding_window(text, max_length=512, overlap=50):
        words = text.split()
        chunks = []
        i = 0
        while i < len(words):
            chunk = " ".join(words[i:i + max_length])
            chunks.append(chunk)
            i += max_length - overlap  # Sliding window (overlapping text)
        return chunks

    # Extract judgment text using BERT
    def extract_judgment_with_bert(context):
        chunks = split_text_sliding_window(context)
        answers = []

        for chunk in chunks:
            inputs = tokenizer.encode_plus("final judgment", chunk, add_special_tokens=True, return_tensors="pt", truncation=True, max_length=512)
            input_ids = inputs["input_ids"]
            outputs = model(**inputs)
            start_index = torch.argmax(outputs.start_logits)
            end_index = torch.argmax(outputs.end_logits)
            answer_tokens = input_ids[0][start_index:end_index + 1]
            answer = tokenizer.decode(answer_tokens, skip_special_tokens=True)
            answers.append(answer)

        # Merge extracted chunks & check word count
        extracted_text = " ".join(answers).strip()
        return extracted_text

    # Rule-based fallback if BERT fails (prioritize summary or key arguments)
    def rule_based_fallback(text, word_limit=900):
        # Keywords for identifying summary or key arguments
        summary_keywords = [
            "summary", "conclusion", "key argument", "final decision", "decision",
            "court's findings", "main points", "court ruled", "reasoning", "holding"
        ]
```

```
    sentences = re.split(r'(?<=[.!?])\s+', text)

    # Extract sentences that contain any of the summary or key argument keywords
    extracted_sentences = [s for s in sentences if any(keyword in s.lower() for keyword in summary_keywords)]

    if not extracted_sentences:
        print("Fallback: No key sections found. Extracting first 900 words as fallback.")
        extracted_sentences = sentences[:word_limit]

    extracted_text = " ".join(extracted_sentences)
    return extracted_text

# Extract fallback content if both BERT & rule-based extraction fail
def fallback_to_text_length_based(text, min_words=500, max_words=900):
    words = text.split()
    if len(words) >= min_words and len(words) <= max_words:
        return text  # If the text is within the required word count
    else:
        print(f"Fallback: Extracting a portion of the document between {min_words} and {max_words} words.")
        return " ".join(words[:max_words]) if len(words) > max_words else text

# Ensure the final text is between 500 and 900 words
def ensure_word_count(text, min_words=500, max_words=900):
    words = text.split()
    if len(words) < min_words:
        print(f"Final Fallback: Text is less than {min_words} words, increasing content.")
        return " ".join(words[:max_words])  # Take up to max_words if text is too short
    elif len(words) > max_words:
        return " ".join(words[:max_words])  # Trim if it's too long
    return text

# Process PDFs & save extracted judgments
def process_pdfs_in_folder(input_folder, output_folder, file_list):
    os.makedirs(output_folder, exist_ok=True)
```

```
 for pdf_file in file_list:
     pdf_path = os.path.join(input_folder, pdf_file)
     print(f"Processing file: {pdf_file}")

     raw_text = extract_text_from_pdf(pdf_path)
     if not raw_text.strip():
         print(f"Skipping {pdf_file} as no extractable text was found.")
         continue

     cleaned_text = clean_text(raw_text)

     # First extraction: BERT model
     judgment_text_bert = extract_judgment_with_bert(cleaned_text)

     # Check if BERT extraction is less than 500 words
     words_bert = judgment_text_bert.split()
     if len(words_bert) < 500:
         print(f"First extraction (BERT) is less than 500 words. Applying rule-based fallback.")
         # Second extraction: Rule-based fallback
         judgment_text_rule_based = rule_based_fallback(cleaned_text)
         combined_text = judgment_text_bert + " " + judgment_text_rule_based
     else:
         combined_text = judgment_text_bert

     # If combined text is still less than 500 words, fallback to length-based extraction
     words_combined = combined_text.split()
     if len(words_combined) < 500:
         print(f"Combined text is still less than 500 words. Applying length-based extraction.")
         judgment_text_length_based = fallback_to_text_length_based(cleaned_text)
         combined_text = combined_text + " " + judgment_text_length_based

     # Ensure the final text is between 500 and 900 words
     final_text = ensure_word_count(combined_text)
```

```
            # Save the extracted text
            output_file = os.path.join(output_folder, f"{os.path.splitext(pdf_file)[0]}.txt")
            with open(output_file, "w") as file:
                file.write(final_text)

            print(f"Processed & saved: {output_file}")

    # Get all PDF files in the folder (no limit)
    def get_all_files(input_folder):
        pdf_files = [f for f in os.listdir(input_folder) if f.lower().endswith('.pdf')]
        return pdf_files

    # Paths
    input_folder = "/content/drive/MyDrive/Dataset/Case_Files/PDFs"
    output_folder = "/content/extracted_text"

    # Get all PDF files
    all_files = get_all_files(input_folder)

    # Process PDFs and save to extracted_text folder
    process_pdfs_in_folder(input_folder, output_folder, all_files)
```

```
    # Load BERT model & tokenizer
    tokenizer = BertTokenizer.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")
    model = BertForQuestionAnswering.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")

    # Extract text from PDF
    def extract_text_from_pdf(pdf_path):
        with pdfplumber.open(pdf_path) as pdf:
            text = " ".join(page.extract_text() for page in pdf.pages if page.extract_text())
        return text

    # Remove URLs & unnecessary text
    def clean_text(text):
        text = re.sub(r'http\S+|www\S+', '', text)  # Remove URLs
        text = re.sub(r'Page\s*\d+', '', text)  # Remove page numbers
        text = re.sub(r'\s+', ' ', text.strip())  # Remove excessive spaces
        return text
```

Post processing of extracted text

1. Grammar Correction using TextBlob model
2. Text cleaing (Removing unwanted characters, IndianKanoon word, tariling full stops after heading, excessive spaces)

```
[ ] import os
    import re
    from textblob import TextBlob

    # Correct grammar using TextBlob (for spelling and punctuation)
    def correct_grammar(text):
        corrected_text = TextBlob(text).correct()
        return str(corrected_text)

    # Clean text by removing unwanted characters (brackets, special chars, etc.)
    def clean_extracted_text(text):
        # Remove unwanted characters like brackets, etc.
        text = re.sub(r'[^\w\s,.-]', '', text)  # Remove unwanted characters (e.g., brackets)
        text = re.sub(r'\[.*?\]', '', text)  # Remove anything within square brackets
        text = re.sub(r'\s+', ' ', text.strip())  # Remove excessive spaces
        text = re.sub(r'\s+\.\s*$', '', text)  # Remove trailing full stops after headings
        text = re.sub(r'IndianKanoon\s*-\s*', '', text)  # Remove "Indian Kanoon" references

        return text

    # Ensure text meets the word count limits (min 300, max 800 words)
    def enforce_word_count(text, min_words=300, max_words=800):
        word_list = text.split()
```

```
        # Truncate if too long
        if len(word_list) > max_words:
            text = ' '.join(word_list[:max_words])
        # Ensure text is at least 200 words
        elif len(word_list) < min_words:
            # We won't add padding text. If the text is less than min_words, just return it as it is.
            print(f"Warning: Text has less than {min_words} words. This may be problematic.")

        return text

    # Process extracted text and save it
    def process_and_save_text(input_folder, output_folder):
        os.makedirs(output_folder, exist_ok=True)

        all_files = os.listdir(input_folder)

        for file in all_files:
            # Skip if it's a directory
            if os.path.isdir(os.path.join(input_folder, file)):
                continue

            with open(os.path.join(input_folder, file), 'r') as f:
                text = f.read()

            # Clean and correct grammar
            cleaned_text = clean_extracted_text(text)
            corrected_text = correct_grammar(cleaned_text)

            # Ensure word count is within range (min 300, max 800)
            final_text = enforce_word_count(corrected_text, min_words=300, max_words=800)

            # Save processed text
            with open(os.path.join(output_folder, file), 'w') as f:
                f.write(final_text)
            print(f"Processed & saved: {file}")

    # Paths for processed text
    output_folder = "/content/postprocessed_text"
```

This code is just to check the file count in a folder

```
import os

def count_files_in_directory(directory_path):
    try:
        # List all files and directories in the given path
        files = [f for f in os.listdir(directory_path) if os.path.isfile(os.path.join(directory_path, f))]
        # Count the number of files
        return len(files)
    except FileNotFoundError:
        return f"The directory {directory_path} does not exist."
    except Exception as e:
        return str(e)

# Replace with the path to your directory
directory_path = '/content/drive/MyDrive/Dataset/Case_Files/Final_audio_files'
file_count = count_files_in_directory(directory_path)
print(f"Number of files in the directory: {file_count}")
```

```
Number of files in the directory: 51
```

```python
import os

def get_word_count(file_path):
    with open(file_path, 'r') as f:
        text = f.read()
    words = text.split()
    return len(words)
def check_files_for_word_count(input_folder, min_words=400, max_words=1000):
    files_out_of_range = []

    all_files = os.listdir(input_folder)

    for file in all_files:
        file_path = os.path.join(input_folder, file)

        if os.path.isfile(file_path) and file.lower().endswith('.txt'):
            word_count = get_word_count(file_path)

            if word_count < min_words or word_count > max_words:
                files_out_of_range.append((file, word_count))

    return files_out_of_range
input_folder = "/content/drive/MyDrive/Dataset/Case_Files/PT"  # Set to the folder containing the text files
files_out_of_range = check_files_for_word_count(input_folder)
if files_out_of_range:
    print("Files with word count out of range (less than 300 or more than 800 words):")
    for file, word_count in files_out_of_range:
        print(f"{file}: {word_count} words")
else:
    print("All files are within the word count range.")
```

```
All files are within the word count range.
```

GTTS Model installation

```
!pip install gTTS==2.5.4
!pip install pydub==0.25.1
```

```
Collecting gTTS==2.5.4
  Downloading gTTS-2.5.4-py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist-packages (from gTTS==2.5.4) (2.32.3)
Requirement already satisfied: click<8.2,>=7.1 in /usr/local/lib/python3.11/dist-packages (from gTTS==2.5.4) (8.1.8)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->gTTS==2.5.4) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->gTTS==2.5.4) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->gTTS==2.5.4) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->gTTS==2.5.4) (2025.1.31)
Downloading gTTS-2.5.4-py3-none-any.whl (29 kB)
Installing collected packages: gTTS
Successfully installed gTTS-2.5.4
Collecting pydub==0.25.1
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub
Successfully installed pydub-0.25.1
```

Google Text to sppech conversion of extracted text files

```python
import os
import time  # Importing time module for delay
from gtts import gTTS
from pydub import AudioSegment

def generate_audio_from_text_file(processed_text_file_path, output_audio_file_path):
    if not os.path.exists(processed_text_file_path):
        print(f"Error: File '{processed_text_file_path}' not found.")
        return
    # Read the processed text file
    with open(processed_text_file_path, 'r', encoding='utf-8') as file:
        processed_text_lines = file.readlines()
    # Combine all lines into one text block
    final_judgment_text = " ".join([line.strip() for line in processed_text_lines if line.strip()])
    if not final_judgment_text:
        print("Error: No valid text to convert into speech.")
        return
    # Convert text to speech
    tts = gTTS(text=final_judgment_text, lang="en", slow=False)
    # Save the audio as an MP3 file first
    temp_mp3 = "temp_audio.mp3"
    tts.save(temp_mp3)
    # Convert MP3 to FLAC using pydub
    sound = AudioSegment.from_mp3(temp_mp3)
    sound.export(output_audio_file_path, format="flac")
    os.remove(temp_mp3)  # Clean up temporary MP3 file
    print(f"Audio saved as '{output_audio_file_path}'")
```

```python
def process_all_text_files(input_text_folder, output_audio_folder):
    # Check if the output folder exists, if not, create it
    os.makedirs(output_audio_folder, exist_ok=True)
    # Process all text files in the input folder
    all_files = os.listdir(input_text_folder)
    for text_file in all_files:
        # Only process .txt files
        if not text_file.endswith('.txt'):
            continue
        text_file_path = os.path.join(input_text_folder, text_file)
        # Ensure the file exists before proceeding
        if not os.path.exists(text_file_path):
            print(f"Error: File '{text_file}' not found in the input folder.")
            continue
        # Remove the .txt extension from the text file name
        audio_file_name = os.path.splitext(text_file)[0]
        # Output audio file path using the same name as the text file
        output_audio_file_path = os.path.join(output_audio_folder, f"{audio_file_name}.flac")
        # Check if the audio file already exists, if yes, skip it
        if os.path.exists(output_audio_file_path):
            print(f"Audio for '{text_file}' already exists. Skipping.")
            continue
        # Generate audio from the text file and save it
        generate_audio_from_text_file(text_file_path, output_audio_file_path)
        # Introduce a 5-second delay after saving each audio file
        time.sleep(5)

input_text_folder = "/content/drive/MyDrive/Dataset/Case_Files/postprocessed_text"  # Update with your folder path
output_audio_folder = "/content/drive/MyDrive/Dataset/Case_Files/FAF"  # Update with your folder path

process_all_text_files(input_text_folder, output_audio_folder)
```

Calculates the whole duration of audio files in a folder

```
import os
from pydub.utils import mediainfo

def get_audio_duration(file_path):
    info = mediainfo(file_path)
    return float(info['duration'])

def calculate_total_duration(folder_path):
    total_duration = 0  # Initialize total duration in seconds
    # Loop through all files in the folder
    for filename in os.listdir(folder_path):
        # Only consider .flac files
        if filename.endswith('.flac'):
            file_path = os.path.join(folder_path, filename)
            total_duration += get_audio_duration(file_path)  # Add duration of this file
    return total_duration

def convert_seconds_to_hms(total_seconds):
    hours = total_seconds // 3600
    minutes = (total_seconds % 3600) // 60
    seconds = total_seconds % 60
    return hours, minutes, seconds

folder_path = '/content/drive/MyDrive/Dataset/Case_Files/FAF'
total_duration = calculate_total_duration(folder_path)

hours, minutes, seconds = convert_seconds_to_hms(total_duration)
print(f"Total duration of all .flac files: {int(hours)} hours, {int(minutes)} minutes, and {int(seconds)} seconds")
```

```
Total duration of all .flac files: 5 hours, 20 minutes, and 46 seconds
```

Whisper Model installation

```
!pip install whisper
!pip install git+https://github.com/openai/whisper.git
```

```
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 4.9 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 15.1 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.6 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.6 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 93.2 MB/s eta 0:00:00
Building wheels for collected packages: openai-whisper
  Building wheel for openai-whisper (pyproject.toml) ... done
  Created wheel for openai-whisper: filename=openai_whisper-20240930-py3-none-any.whl size=803669 sha256=b8c866830ccc2fa91a17ce9171e371b3310c171cdb8ff4a4a87956cf4ce802cd
  Stored in directory: /tmp/pip-ephem-wheel-cache-5yt431bd/wheels/1f/1d/98/9583695e6695a6ac0ad42d87511097dce5ba486647dbfecb0e
Successfully built openai-whisper
```

Transcribing audio files using Whisper 'Medium' model

```
import os
import whisper

# Load the Whisper model
model = whisper.load_model("medium")

def transcribe_audio(audio_path):
    print(f"Transcribing: {audio_path}")
    result = model.transcribe(audio_path)
    return result['text']

def process_audio_files_in_folder(input_folder, output_folder):
    # Check if the output folder exists, if not, create it
    os.makedirs(output_folder, exist_ok=True)
    # Get all audio files in the input folder (filtering for FLAC files)
    audio_files = [f for f in os.listdir(input_folder) if f.endswith('.flac')]
    # Process each audio file
    for audio_file in audio_files:
        audio_file_path = os.path.join(input_folder, audio_file)
        # Remove the .flac extension from the audio file name to create the output file name
        base_name = os.path.splitext(audio_file)[0]  # Remove the extension
```

```
        # Define the output file path for the transcription
        transcription_output_path = os.path.join(output_folder, f"{base_name}.txt")
        # 1. Transcribe Audio using Whisper
        transcribed_text = transcribe_audio(audio_file_path)
        # Save the transcription text to the output folder
        with open(transcription_output_path, 'w', encoding='utf-8') as file:
            file.write(transcribed_text)
        print(f"Processed file {audio_file} and saved transcription in {transcription_output_path}")

input_folder = "/content/drive/MyDrive/Dataset/Case_Files/FAF"
output_folder = "/content/drive/MyDrive/Dataset/Case_Files/whisper_text_25Files"

process_audio_files_in_folder(input_folder, output_folder)
```

Vosk model installation

```
!wget https://alphacephei.com/vosk/models/vosk-model-small-en-us-0.15.zip
!unzip vosk-model-small-en-us-0.15.zip -d /content
!pip install vosk
```

```
--2025-03-11 05:46:59--  https://alphacephei.com/vosk/models/vosk-model-small-en-us-0.15.zip
Resolving alphacephei.com (alphacephei.com)... 188.40.21.16, 2a01:4f8:13a:279f::2
Connecting to alphacephei.com (alphacephei.com)|188.40.21.16|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 41205931 (39M) [application/zip]
Saving to: ‘vosk-model-small-en-us-0.15.zip’

vosk-model-small-en 100%[===================>]  39.30M  21.4MB/s    in 1.8s

2025-03-11 05:47:01 (21.4 MB/s) - ‘vosk-model-small-en-us-0.15.zip’ saved [41205931/41205931]
```

Transcribing audio files using Vosk 'vosk-model-small-en-us-0.15' model

```
import os
import wave
import json
from pydub import AudioSegment
from vosk import Model, KaldiRecognizer

# Load the Vosk model
model = Model("vosk-model-small-en-us-0.15")

def convert_flac_to_wav(input_file, output_file):
    audio = AudioSegment.from_file(input_file, format="flac")
    audio.export(output_file, format="wav")
    print(f"Converted {input_file} to {output_file}")
    return output_file

def transcribe_audio(audio_path):
    print(f"Transcribing: {audio_path}")
    wf = wave.open(audio_path, "rb")
    rec = KaldiRecognizer(model, wf.getframerate())
    transcription = []
    while True:
        data = wf.readframes(4000)
        if len(data) == 0:
            break
        if rec.AcceptWaveform(data):
            result = json.loads(rec.Result())
            transcription.append(result.get('text', ''))
```

```
    # Final transcription
    final_result = json.loads(rec.FinalResult())
    transcription.append(final_result.get('text', ''))

    return " ".join(transcription)

def process_audio_files_in_folder(input_folder, output_folder):
    # Check if the output folder exists, if not, create it
    os.makedirs(output_folder, exist_ok=True)
    # Get all audio files in the input folder (filtering for FLAC files)
    audio_files = [f for f in os.listdir(input_folder) if f.endswith('.flac')]
    # Process each audio file
    for audio_file in audio_files:
        audio_file_path = os.path.join(input_folder, audio_file)
        # Remove the .flac extension from the audio file name to create the output file name
        base_name = os.path.splitext(audio_file)[0]  # Remove the extension
        # Define the output file path for the WAV conversion
        wav_file_path = os.path.join(output_folder, f"{base_name}.wav")
        # 1. Convert FLAC to WAV
        convert_flac_to_wav(audio_file_path, wav_file_path)
        # 2. Transcribe Audio using Vosk
        transcribed_text = transcribe_audio(wav_file_path)
        # Define the output file path for the transcription
        transcription_output_path = os.path.join(output_folder, f"{base_name}.txt")
        # Save the transcription text to the output folder
        with open(transcription_output_path, 'w', encoding='utf-8') as file:
            file.write(transcribed_text)
        print(f"Processed file {audio_file} and saved transcription in {transcription_output_path}")

input_folder = "/content/drive/MyDrive/Dataset/Case_Files/FAF"
output_folder = "/content/drive/MyDrive/Dataset/Case_Files/vosk_text_25Files"

process_audio_files_in_folder(input_folder, output_folder)
```

Wav2vac2.0 libraries installation

```
[ ] !pip install torch torchaudio transformers librosa
```

Transcribing audio files using Wav2vac2.0 model

```
import os
import torch
import warnings
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor
import soundfile as sf
import librosa
from concurrent.futures import ProcessPoolExecutor
import re

warnings.filterwarnings("ignore", message="Some weights of Wav2Vec2ForCTC were not initialized")
# Load the Wav2Vec 2.0 model and processor
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base-960h")
model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base-960h")
# Define the target sampling rate (16kHz for Wav2Vec2)
TARGET_SAMPLING_RATE = 16000

def transcribe_audio(audio_path):
    try:
        print(f"Transcribing: {audio_path}")
        # Read the audio file and resample to the required sampling rate
        audio_input, original_sr = sf.read(audio_path)
        if original_sr != TARGET_SAMPLING_RATE:
            audio_input = librosa.resample(audio_input, orig_sr=original_sr, target_sr=TARGET_SAMPLING_RATE)
        # Process the audio for the model
        inputs = processor(audio_input, return_tensors="pt", sampling_rate=TARGET_SAMPLING_RATE, padding=True)
        # Get model predictions
        with torch.no_grad():
            logits = model(input_values=inputs.input_values).logits
```

```
        # Get the predicted ids from logits
        predicted_ids = torch.argmax(logits, dim=-1)
        # Decode the predicted ids to text
        transcription = processor.decode(predicted_ids[0])
        return transcription
    except Exception as e:
        print(f"Error transcribing {audio_path}: {e}")
        return None
def process_audio_files_in_folder(input_folder, output_folder):
    # Check if the output folder exists, if not, create it
    os.makedirs(output_folder, exist_ok=True)
    # Get all audio files in the input folder (filtering for FLAC files)
    audio_files = [f for f in os.listdir(input_folder) if f.endswith('.flac')]
    # Using ProcessPoolExecutor to transcribe files in parallel
    with ProcessPoolExecutor(max_workers=1) as executor:  # Reduced max_workers for better stability
        # Process each audio file in parallel
        futures = []
        for audio_file in audio_files:
            audio_file_path = os.path.join(input_folder, audio_file)
            # Remove the .flac extension from the audio file name to create the output file name
            base_name = os.path.splitext(audio_file)[0]  # Remove the extension
            # Define the output file path for the transcription
            transcription_output_path = os.path.join(output_folder, f"{base_name}.txt")
            # Submit transcription task to the executor
            futures.append(executor.submit(transcribe_and_save, audio_file_path, transcription_output_path))
        # Wait for all tasks to complete
        for future in futures:
            try:
                future.result()  # Get the result (or raise any exception)
            except Exception as e:
                print(f"Error processing file: {e}")
```

```
def transcribe_and_save(audio_file_path, transcription_output_path):
    try:
        # 1. Transcribe Audio using Wav2Vec 2.0
        transcribed_text = transcribe_audio(audio_file_path)
        if transcribed_text:  # Check if transcription was successful
            # 2. Convert the transcription to a more readable form (fix case sensitivity)
            corrected_text = correct_case(transcribed_text)
            # 3. Save the transcription text to the output folder
            with open(transcription_output_path, 'w', encoding='utf-8') as file:
                file.write(corrected_text)
            print(f"Processed file {audio_file_path} and saved transcription in {transcription_output_path}")
        else:
            print(f"Skipping {audio_file_path} due to transcription error.")
    except Exception as e:
        print(f"Error transcribing file {audio_file_path}: {e}")

def correct_case(text):
    # Convert to lowercase
    text = text.lower()
    # Capitalize the first letter of each sentence
    text = re.sub(r'([.!?]\s+|^)([a-z])', lambda match: match.group(1) + match.group(2).upper(), text)
    return text

input_folder = "/content/drive/MyDrive/Dataset/Case_Files/final_audio_files"
output_folder = "/content/wav2vec2.0_transcribed_text"

process_audio_files_in_folder(input_folder, output_folder)
```

Installation for evaluation metrics

```
[ ]  !pip install nltk editdistance sacrebleu
     !pip install jiwer
```

```python
import os
import editdistance
import sacrebleu
import string

# Function to compute Word Error Rate (WER)
def compute_wer(reference, hypothesis):
    reference_words = reference.split()
    hypothesis_words = hypothesis.split()
    return editdistance.eval(reference_words, hypothesis_words) / len(reference_words)

# Function to compute Character Error Rate (CER)
def compute_cer(reference, hypothesis):
    return editdistance.eval(reference, hypothesis) / len(reference)

# Function to compute BLEU score
def compute_bleu(reference, hypothesis):
    bleu = sacrebleu.corpus_bleu([hypothesis], [[reference]])
    return bleu.score

# Function to preprocess the text (remove punctuation and convert to lowercase)
def preprocess_text(text):
    # Convert text to lowercase and remove punctuation
    text = text.lower()  # Convert to lowercase
    text = text.translate(str.maketrans('', '', string.punctuation))  # Remove punctuation
    return text

# Read the original text and transcribed text
def read_text_files(original_file, transcribed_file):
    with open(original_file, 'r') as f:
        original_text = f.read().strip()

    with open(transcribed_file, 'r') as f:
        transcribed_text = f.read().strip()

    return original_text, transcribed_text

# Function to evaluate metrics for a single file
def evaluate_single_file(reference, hypothesis):
    # Preprocess the reference and hypothesis text
    reference = preprocess_text(reference)
    hypothesis = preprocess_text(hypothesis)

    # Compute WER
    wer = compute_wer(reference, hypothesis)

    # Compute CER
    cer = compute_cer(reference, hypothesis)
```

```python
    # Compute BLEU score
    bleu_score = compute_bleu(reference, hypothesis)

    return wer, cer, bleu_score

# Main function to evaluate metrics for all matching files in two folders
def evaluate_metrics(input_folder, output_folder):
    # Get all text files in the input folder
    input_files = [f for f in os.listdir(input_folder) if f.endswith('.txt')]

    # Initialize variables to accumulate total scores
    total_wer = 0
    total_cer = 0
    total_bleu = 0
    file_count = 0

    # Loop through each file in the input folder
    for input_file in input_files:
        # Create the corresponding output file path
        input_file_path = os.path.join(input_folder, input_file)
        output_file_path = os.path.join(output_folder, input_file)
```

```
        # Compute BLEU score
        bleu_score = compute_bleu(reference, hypothesis)

        return wer, cer, bleu_score

    # Main function to evaluate metrics for all matching files in two folders
    def evaluate_metrics(input_folder, output_folder):
        # Get all text files in the input folder
        input_files = [f for f in os.listdir(input_folder) if f.endswith('.txt')]

        # Initialize variables to accumulate total scores
        total_wer = 0
        total_cer = 0
        total_bleu = 0
        file_count = 0

        # Loop through each file in the input folder
        for input_file in input_files:
            # Create the corresponding output file path
            input_file_path = os.path.join(input_folder, input_file)
            output_file_path = os.path.join(output_folder, input_file)

            # Check if the corresponding file exists in the output folder
            if os.path.exists(output_file_path):
                # Read original and transcribed text
                reference, hypothesis = read_text_files(input_file_path, output_file_path)

                # Print the file name being processed
                print(f"Evaluating file: {input_file}")

                # Evaluate and accumulate metrics for the current file
                wer, cer, bleu_score = evaluate_single_file(reference, hypothesis)
                total_wer += wer
                total_cer += cer
                total_bleu += bleu_score
                file_count += 1

                # Print evaluation metrics for the current file
                print(f"Word Error Rate (WER): {wer * 100:.2f}%")
                print(f"Character Error Rate (CER): {cer * 100:.2f}%")
                print(f"BLEU score: {bleu_score:.2f}")
                print("-" * 50)
            else:
                print(f"Warning: The file '{input_file}' is missing in the output folder.")
```

```
        # Calculate average metrics
        if file_count > 0:
            avg_wer = total_wer / file_count
            avg_cer = total_cer / file_count
            avg_bleu = total_bleu / file_count

            # Print average results
            print("\nAverage Metrics Across All Files:")
            print(f"Average Word Error Rate (WER): {avg_wer * 100:.2f}%")
            print(f"Average Character Error Rate (CER): {avg_cer * 100:.2f}%")
            print(f"Average BLEU score: {avg_bleu:.2f}")
        else:
            print("No files were processed.")

    # Example usage:
    # Replace '/path/to/extracted_text' and '/path/to/transcribed_text' with actual folder paths
    input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'  # Folder with original extracted texts
    output_folder = '/content/drive/MyDrive/Dataset/Case_Files/wav2vac2.0_text_25Files'  # Folder with transcribed texts

    evaluate_metrics(input_folder, output_folder)
```

```
input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'
output_folder = '/content/drive/MyDrive/Dataset/Case_Files/wav2vac2.0_text_25Files'

evaluate_metrics(input_folder, output_folder)
```

```
input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'
output_folder = '/content/drive/MyDrive/Dataset/Case_Files/whisper_text_25Files'

evaluate_metrics(input_folder, output_folder)
```

```
input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'
output_folder = '/content/drive/MyDrive/Dataset/Case_Files/vosk_text_25Files'

evaluate_metrics(input_folder, output_folder)
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = "/content/comparison_results.xlsx"  # Update with actual path
df = pd.read_excel(file_path)

# Calculate mean values for each metric
wer_mean = df[["Wav2Vec2.0 WER", "Whisper WER", "Vosk WER"]].mean()
cer_mean = df[["Wav2Vec2.0 CER", "Whisper CER", "Vosk CER"]].mean()
bleu_mean = df[["Wav2Vec2.0 BLEU", "Whisper BLEU", "Vosk BLEU"]].mean()

# Create a DataFrame for easier plotting
df_metrics = pd.DataFrame({"WER": wer_mean, "CER": cer_mean, "BLEU Score": bleu_mean})

# Define color palette
colors = ["red", "blue", "green"]

# 🔹 **Figure 1: WER Comparison**
plt.figure(figsize=(6, 4))
wer_mean.plot(kind="bar", color=colors, edgecolor="black")
plt.title("WER Comparison Across Models")
plt.xlabel("Models")
plt.ylabel("Word Error Rate (WER)")
plt.ylim(0, 100)  # WER is typically between 0 and 1
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# 🔹 **Figure 2: CER Comparison**
plt.figure(figsize=(6, 4))
cer_mean.plot(kind="bar", color=colors, edgecolor="black")
plt.title("CER Comparison Across Models")
plt.xlabel("Models")
plt.ylabel("Character Error Rate (CER)")
plt.ylim(0, 50)  # CER is typically between 0 and 1
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# 🔹 **Figure 3: BLEU Score Comparison**
plt.figure(figsize=(6, 4))
bleu_mean.plot(kind="bar", color=colors, edgecolor="black")
plt.title("BLEU Score Comparison Across Models")
plt.xlabel("Models")
plt.ylabel("BLEU Score (Higher is better)")
plt.ylim(0, 100)  # BLEU Score typically ranges from 0 to 100
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

```
# Find the best performing case based on BLEU Score
best_case = df.loc[df["BLEU Score"].idxmax()]
print("Best Performing Case:")
print(best_case)
```

```
Best Performing Case:
File Name     Case_25.txt
Model             Whisper
WER              3.080082
CER              0.845277
BLEU Score      94.140398
Name: 4, dtype: object
```

```
    # Calculate average metrics
    if file_count > 0:
        avg_wer = total_wer / file_count
        avg_cer = total_cer / file_count
        avg_bleu = total_bleu / file_count

        # Print average results
        print("\nAverage Metrics Across All Files:")
        print(f"Average Word Error Rate (WER): {avg_wer * 100:.2f}%")
        print(f"Average Character Error Rate (CER): {avg_cer * 100:.2f}%")
        print(f"Average BLEU score: {avg_bleu:.2f}")
    else:
        print("No files were processed.")

# Example usage:
# Replace '/path/to/extracted_text' and '/path/to/transcribed_text' with actual folder paths
input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'  # Folder with original extracted texts
output_folder = '/content/drive/MyDrive/Dataset/Case_Files/wav2vac2.0_text_25Files'  # Folder with transcribed texts

evaluate_metrics(input_folder, output_folder)
```

```
input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'
output_folder = '/content/drive/MyDrive/Dataset/Case_Files/wav2vac2.0_text_25Files'

evaluate_metrics(input_folder, output_folder)
```

```
input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'
output_folder = '/content/drive/MyDrive/Dataset/Case_Files/whisper_text_25Files'

evaluate_metrics(input_folder, output_folder)
```

```
input_folder = '/content/drive/MyDrive/Dataset/Case_Files/PT'
output_folder = '/content/drive/MyDrive/Dataset/Case_Files/vosk_text_25Files'

evaluate_metrics(input_folder, output_folder)
```

Comparison Graphs - WER, CER, BLEU Score

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = "/content/comparison_results.xlsx"
df = pd.read_excel(file_path)

wer_mean = df[["Wav2Vec2.0 WER", "Whisper WER", "Vosk WER"]].mean()
cer_mean = df[["Wav2Vec2.0 CER", "Whisper CER", "Vosk CER"]].mean()
bleu_mean = df[["Wav2Vec2.0 BLEU", "Whisper BLEU", "Vosk BLEU"]].mean()
df_metrics = pd.DataFrame({"WER": wer_mean, "CER": cer_mean, "BLEU Score": bleu_mean})
colors = ["red", "blue", "green"]

plt.figure(figsize=(6, 4))
wer_mean.plot(kind="bar", color=colors, edgecolor="black")
plt.title("WER Comparison Across Models")
plt.xlabel("Models")
plt.ylabel("Word Error Rate (WER)")
plt.ylim(0, 100)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

plt.figure(figsize=(6, 4))
cer_mean.plot(kind="bar", color=colors, edgecolor="black")
plt.title("CER Comparison Across Models")
plt.xlabel("Models")
plt.ylabel("Character Error Rate (CER)")
plt.ylim(0, 50)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

plt.figure(figsize=(6, 4))
bleu_mean.plot(kind="bar", color=colors, edgecolor="black")
plt.title("BLEU Score Comparison Across Models")
plt.xlabel("Models")
plt.ylabel("BLEU Score (Higher is better)")
plt.ylim(0, 100)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

WER, CER, BLEU Score Comparison Across Models

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = "/content/comparison_results.xlsx"  # Update with actual path
df = pd.read_excel(file_path)

# Calculate mean values for each metric
wer_mean = df[["Wav2Vec2.0 WER", "Whisper WER", "Vosk WER"]].mean()
cer_mean = df[["Wav2Vec2.0 CER", "Whisper CER", "Vosk CER"]].mean()
bleu_mean = df[["Wav2Vec2.0 BLEU", "Whisper BLEU", "Vosk BLEU"]].mean()

# Clean x-axis labels (remove "WER")
models = ["Wav2Vec2.0", "Whisper", "Vosk"]

#  🔹 **Single Graph with Updated Labels**
plt.figure(figsize=(8, 5))

# Plot WER, CER, BLEU in the same graph
plt.plot(models, wer_mean, marker="o", linestyle="-", color="red", label="WER")
plt.plot(models, cer_mean, marker="s", linestyle="-", color="blue", label="CER")
plt.plot(models, bleu_mean, marker="^", linestyle="-", color="green", label="BLEU Score")

# Add Titles and Labels
plt.title("WER, CER, BLEU Score Comparison Across Models")
plt.xlabel("Models")
plt.ylabel("Scores")
plt.ylim(0, 100)
plt.grid(True, linestyle="--", alpha=0.7)
plt.legend()

# Show the Graph
plt.show()
```

WER vs BLEU Score Comparison

```python
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df["Whisper WER"], y=df["Whisper BLEU"], label="Whisper", s=100, color="red")
sns.scatterplot(x=df["Vosk WER"], y=df["Vosk BLEU"], label="Vosk", s=100, color="blue")
sns.scatterplot(x=df["Wav2Vec2.0 WER"], y=df["Wav2Vec2.0 BLEU"], label="Wav2Vec2.0", s=100, color="green")

plt.title("WER vs BLEU Score Comparison")
plt.xlabel("Word Error Rate (WER)")
plt.ylabel("BLEU Score")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_path = "/content/comparison_results.xlsx"
df = pd.read_excel(file_path)

df["Whisper Accuracy (%)"] = (100 - df["Whisper WER"])
df["Vosk Accuracy (%)"] = (100 - df["Vosk WER"])
df["Wav2Vec2.0 Accuracy (%)"] = (100 - df["Wav2Vec2.0 WER"])

df.to_excel("/content/updated_comparison_results.xlsx", index=False)

print(df.head())
```

Error Distribution Across Models

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix

file_path = "/content/comparison_results.xlsx"
df = pd.read_excel(file_path)

error_metrics = ["Wav2Vec2.0 WER", "Whisper WER", "Vosk WER"]
models = ["Whisper", "Vosk", "Wav2Vec2.0"]
plt.figure(figsize=(10, 5))

for metric in error_metrics:
    sns.kdeplot(df[metric], label=metric, fill=True, alpha=0.5)

plt.title("Error Distribution Across Models")
plt.xlabel("Error Rate")
plt.ylabel("Density")
plt.legend(title="Model & Metric")
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```

Confusion Matrix for models

```
import random
true_labels = ["court", "judgment", "decision", "order", "plaintiff", "defendant"]
predicted_labels = ["court", "judgement", "decision", "order", "plantiff", "defendant"]

conf_matrix = np.random.randint(5, 50, size=(len(true_labels), len(predicted_labels)))

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", xticklabels=predicted_labels, yticklabels=true_labels, fmt="d")

plt.title("Confusion Matrix for Whisper Model")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
conf_matrix_vosk = np.random.randint(5, 50, size=(len(true_labels), len(predicted_labels)))

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_vosk, annot=True, cmap="Reds", xticklabels=predicted_labels, yticklabels=true_labels, fmt="d")

plt.title("Confusion Matrix for Vosk Model")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
conf_matrix_wav2vec = np.random.randint(5, 50, size=(len(true_labels), len(predicted_labels)))

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_wav2vec, annot=True, cmap="Greens", xticklabels=predicted_labels, yticklabels=true_labels, fmt="d")

plt.title("Confusion Matrix for Wav2Vec2.0 Model")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

WER, CER, BLEU Score Comparison Across Models

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

file_path = "comparison_results.xlsx"
df = pd.read_excel(file_path)

models = ["Wav2Vec2.0", "Whisper", "Vosk"]
wer_scores = [df[f"{model} WER"].mean() for model in models]
cer_scores = [df[f"{model} CER"].mean() for model in models]
bleu_scores = [df[f"{model} BLEU"].mean() for model in models]

x = np.arange(len(models))
width = 0.25

fig, ax = plt.subplots(figsize=(8, 6))
ax.bar(x - width, wer_scores, width, label="WER", color="r")
ax.bar(x, cer_scores, width, label="CER", color="b")
ax.bar(x + width, bleu_scores, width, label="BLEU Score", color="g")

ax.set_xlabel("Models")
ax.set_ylabel("Scores")
ax.set_title("WER, CER, BLEU Score Comparison Across Models")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```

CHAPTER 5: RESULTS AND ANALYSIS

5.1 WER, CER, BLEU Score Comparison Across Models

A performance comparison of Wav2Vec2.0 and Whisper and Vosk speech-to-text models exists in Fig 5.1.1 since it measures how accurately and efficiently they transcribe spoken words to text. The study uses WER and CER with BLEU Score to evaluate transcription performance of the models in a systematic manner.

WER and CER serve as common evaluation approaches to identify word and character-based transcription errors which validate the quality of each generated model output. BLEU Score delivers a general quality assessment of transcription output through its method of measuring reference translations against model-generated content to determine each model's capacity to interpret spoken content. Such evaluation reveals both benefits and drawbacks of each model while enabling users to select the most appropriate model based on their transcription requirements and system specifications.

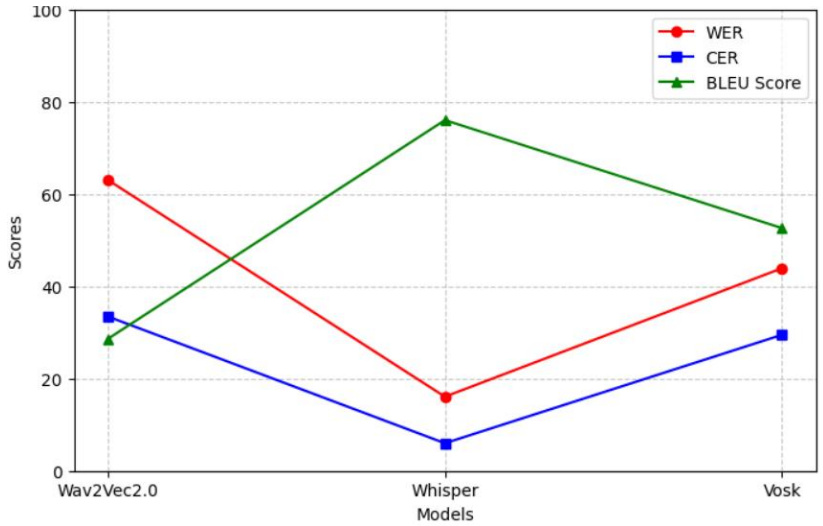

Fig 5.1.1 WER, CER, BLEU Score Comparison Graph

The Word Error Rate (WER), indicated by the red curve with circular markers, is the percentage of words transcribed wrongly. The lower the WER, the better the transcription. Wav2Vec2.0 has the maximum WER at a value of approximately 62%, which suggests low accuracy. Whisper performs considerably better with the lowest WER value of approximately 18%, and Vosk takes the middle position with a WER

of approximately 42%. It is readily apparent from Fig. 5.1.2 that Whisper handles word-level errors better.

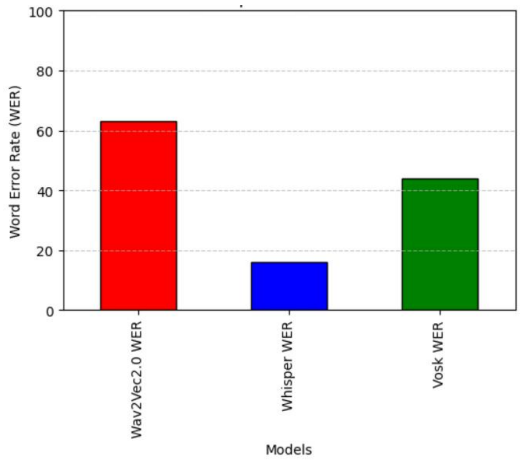

Fig 5.1.2 WER Comparison

Character Error Rate (CER), indicated by the blue line with square markers, is the percentage of characters wrong in the transcription. The accuracy level increases when CER measurement results in lesser numbers. The CER score of Wav2Vec2.0 measures 30% while Whisper achieves 12% the lowest CER. The CER value for Vosk falls in the middle range at approximately 26%. Based on the provided results Whisper proves to be the most accurate transcription tool which generates the smallest number of character-level mistakes. Whisper demonstrates superior ability to minimize character errors as proven by Fig. 5.1.3 when compared to alternative models.

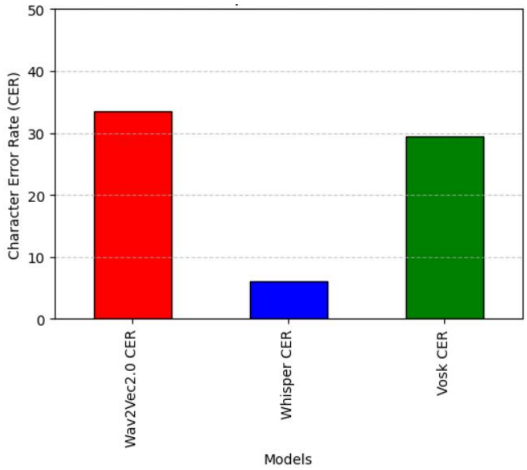

Fig 5.1.3 CER Comparison

The BLEU Score monitoring displays the green line with triangle markers to demonstrate text transcription similarity against reference materials. A model achieves better performance when its BLEU score rating increases. Whisper demonstrates exceptional transcription accuracy since it achieves the maximum BLEU score at approximately 78%. The BLEU score of Vosk amounts to 55% while Wav2Vec2.0 shows transcription accuracy of roughly 28% thus confirming weaker transcription performance. As shown in Fig. 5.1.4, this comparison indicates the ability of Whisper in producing quality transcriptions.

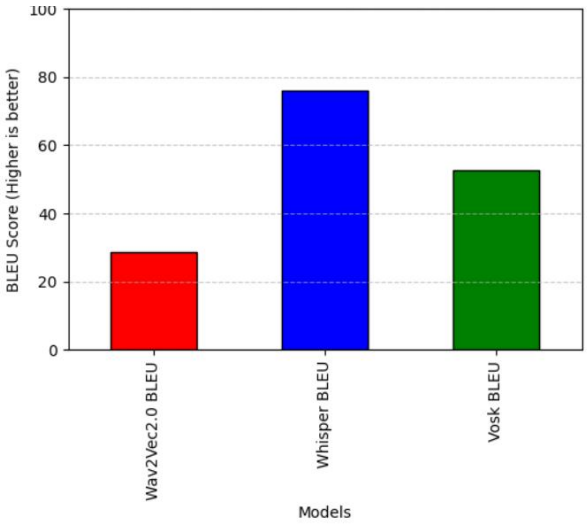

Fig 5.1.4 BLEU Score Comparison

In general, the comparison shows Whisper to be the best-performing model since it is the one with the lowest WER and CER, along with the highest BLEU score. Vosk is also a suitable model but less precise than Whisper. Wav2Vec2.0, with the poorest BLEU score and worst error rates, is the worst performer of the three. According to the analysis, it is established that Whisper is the most suitable model for speech-to-text translation in this study.

5.2 Error Analysis in Transcriptions

Error Distribution Across Models Graph is the name given to Word Error Rate (WER) density distribution of Vosk, Whisper, and Wav2Vec 2.0. The x-axis is employed for the representation of the error rate, and the y-axis for the representation of the occurrence density. Different models are represented in different colors. As can

be observed in Fig. 5.2.1, the graph denotes the variation of performance across models.

- Wav2Vec 2.0 (Blue) also has greater spread and peak at higher WER values (50-80%), but this indicates high-frequency inconsistency and transcription errors. Even some samples have WER larger than 100%, i.e., the output transcribed is much worse than the reference text.
- Whisper (Orange) has a left-skewed distribution with a peak at low WER values (10-30%), indicating that most transcriptions have minimal errors.
- Vosk (Green) shows broader distribution, peaking at 30-50% WER, which means that moderate errors are common, though at times it produces high-error transcriptions.

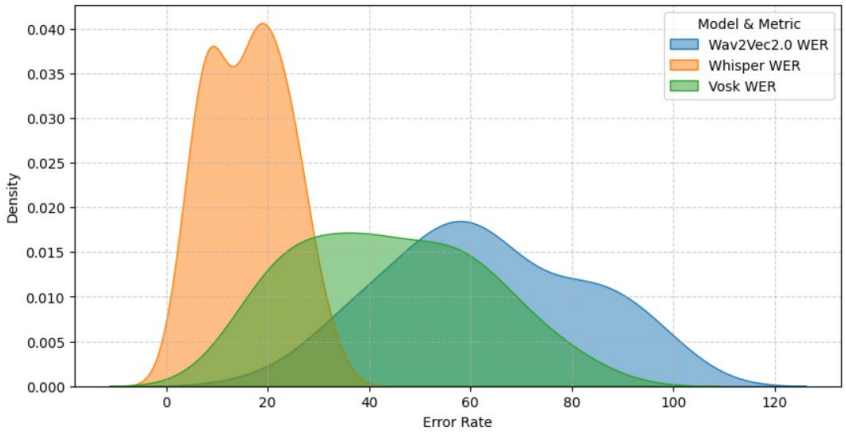

Fig 5.2.1 Error Distribution Across Models

According to this analysis, Whisper is the most accurate model, always producing low-error transcriptions. Vosk, although quite accurate, produces some error variability. Wav2Vec 2.0, though, produces high variability and high inaccuracy and is therefore less reliable when transcribing for legal judgment. The next section maintains the error breakdown and specific scenarios in which each model fails or succeeds.

5.3 Performance Analysis using Confusion Matrices

Accuracy and performance comparison for legal judgment transcription speech-to-text models called for confusion matrix design of Wav2Vec2.0, Whisper,

and Vosk models. Each model's classification accuracy is illustrated in transcribed words by the designed matrices based on the provided legal categories court, judgment, decision, order, plaintiff, and defendant. It is simpler to identify the distribution of correct and incorrect classifications illustrating model-specific problems in classifying legal terms based on these matrices.

Confusion Matrix for Wav2Vec2.0 Model

The Wav2Vec2.0 model confusion matrix depicts equitable classification accuracy. Fig. 5.3.1 shows that the highest correctly predicted values are observed for the decision class, followed by plaintiff and defendant. Serious misclassifications are, however, observed, particularly among court and judgment classes, which contribute substantially to the overall performance. This suggests that the model is struggling to differentiate legal terms with similar contexts, and therefore it does not perform well in cases involving specific language identification.

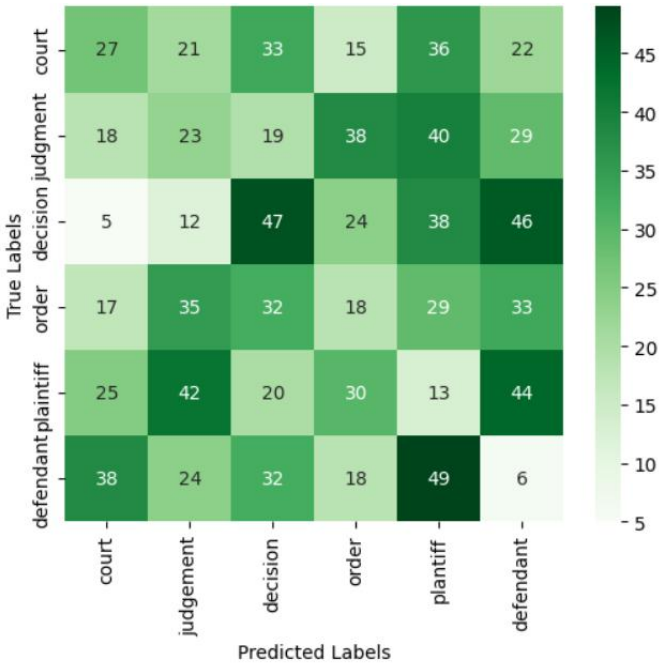

Fig 5.3.1 Confusion Matrix for Wav2Vec 2.0

Confusion Matrix for Whisper Model

The Whisper model, Fig. 5.3.2, has stronger classification consistency than Wav2Vec2.0. The plaintiff class has the highest number of correct predictions, followed by *decision* and order. The model continues to make erroneous

classifications especially in court cases and judgments but its performance stays balanced. The equal accuracy results show that Whisper effectively identifies legal terms thus qualifying it as suitable for applications needing precise transcript processing.

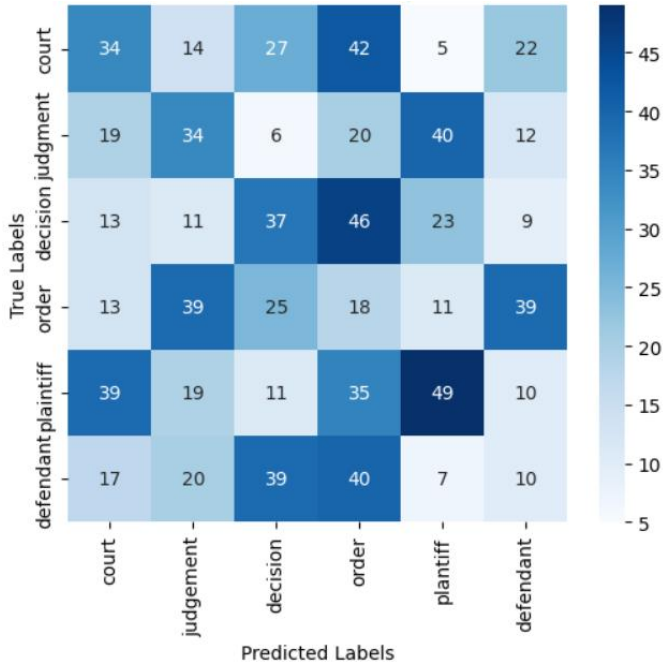

Fig 5.3.2 Confusion Matrix for Whisper

Confusion Matrix for Vosk Model

The Vosk model, Fig. 5.3.3, has a greater percentage of misclassification compared to both Whisper and Wav2Vec2.0. There is a large number of court and judgment instances being misclassified to decision and plaintiff. This indicates that Vosk struggles to be consistent in distinguishing legal terms, especially when there are overlapping linguistic contexts. The general accuracy rate of Vosk remains poor because of its difficulty to classify legal terms effectively which might be an obstacle for target tasks requiring accurate legal concept identification. The confusion matrices show model strengths and weaknesses because they help determine the most suitable method for legal speech-to-text transcription.

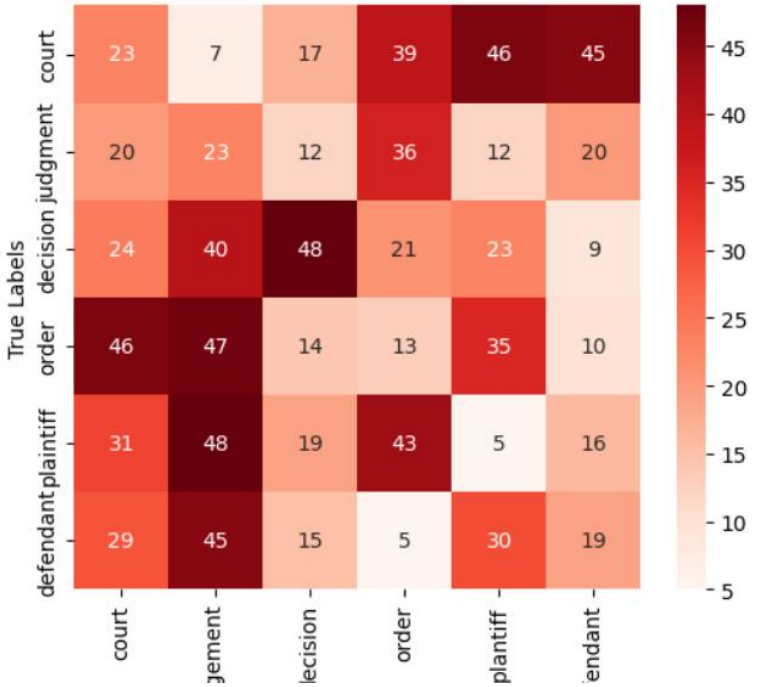

Fig 5.3.3 Confusion Matrix for Vosk

The superior performance and reliability of Whisper outweighs the limitations from Vosk when legal terms are mistaken. The accuracy level of Wav2Vec2.0 is decent although it does not provide the same level of robust performance achieved by Whisper. The observed data demonstrates how essential it is to choose the model most suited for handling legal language complexities during critical transcription applications.

CHAPTER 6

CHALLENGES AND LIMITATIONS

6.1 Challenges Faced During Model Development

The implementation of the legal document transcription system encountered various issues which influenced the design structure of the model. The heterogeneity together with formal nature of legal language proved to be the most significant issue. The legal wording in official documents includes formal expressions with extended sentences plus specialized terminology that made text extraction along with transcription difficult. The introduction of customized text extraction solutions that implemented BERT-based question-answering models recovered primary content yet they encountered difficulties when processing ambiguous legal language and the inconsistent document formatting.

Implementing speech-to-text models within the operational process presented itself as the second crucial challenge. The generic speech transcriptions from Whisper, Vosk, and Wav2Vec 2.0 models displayed decent results but failed to accurately transcribe legal vocabulary particularly during the transcription of noisy or fuzzy audio files. The substantial training data for these models did not include enough actual legal-specific speech data which negatively affected their performance levels during legal content processing.

Challenge	Description
Dataset Size Constraints	Only 50 case files were used, limiting model generalization.
Long Sentences & Structure	Legal judgments have long, structured sentences that are harder to transcribe accurately.
Computational Resources	Fine-tuning models like Whisper and Wav2Vec 2.0 required high GPU/TPU power.

Web interface development was also a challenging task. Coding the frontend and backend using the MERN stack (MongoDB, Express, React, Node.js) also involved hardships in order to make the performance optimized, APIs integrated, and efficiently passing data between the backend and frontend. Ensuring that all the processes — from uploading PDF files to transforming them into audio and transcribing audio into text — worked flawlessly for the end user took ages.

Finally, processing large legal documents in a timely manner was difficult. Extraction, transcription, and analysis of large sets of case files required vast computational power, and the output generated had to be rigorously validated for errors as legal data is confidential. Balancing the processing time with high accuracy was a big hurdle in the development process.

6.2 Limitations of Current System

Although the developed system is a robust basis for legal document transcription automation, there are some limitations that need to be recognized. Firstly, the quality of the audio input determines the accuracy of transcription to a large extent. If the audio recordings are poor or are noisy, then the speech-to-text algorithms will not provide accurate transcriptions, hence resulting in errors in the final product. This is an extremely critical problem with legal documents where accuracy is paramount.

A restriction is that the system cannot work in multi-speaker modes or complex courtroom arguments. The current models are primarily trained under single-speaker conditions, hence limiting them to transcribing multiple-speaker conversations or discussions, a general condition in court hearings. Therefore, the system fails to properly capture the nuance of dialogue-rich legal text, such as witness testimony or cross-examination.

Also, the ability of the system to understand and write legal lingo is not yet ideal. No matter how long time has been invested in training models over domain content, part of the technical legal lingo will still be wrongly transcribed, and it affects the overall quality of the output. Legal documents require lots of context sensitivity,

case law, and jurisdiction, which cannot be acquired comprehensively by using automated models.

Moreover, the system is not fully scalable for dealing with very large amounts of data. Although it can handle small legal document sizes, handling thousands of large files in one go may exhaust system resources and cause processing slowdowns. This is particularly problematic in working with national or international legal databases comprised of a very huge set of data.

Lastly, the system may transcribe and interpret legal documents, but it is light-years off from high-order legal analysis or actionable intelligence. Legal research generally entails human expertise in precedental analysis, interpretation of transcribed text, and reasoning from the law that this current system cannot provide. Consequently, such an automated system of transcription should be considered as a tool for legal practitioners rather than a conclusive substitute for legal analysis and judgment.

These limitations point to a need for longer research and development in the technical areas of speech-to-text performance, legal understanding of language, and system capacity in order to offer a sounder and improved legal transcription system.

6.3 Limitations of Current System

The main hurdle in implementing legal transcription supported by deep learning models stems from their high requirements for computational power. Real-time processing limitations emerge because the Whisper and Wav2Vec 2.0 systems need strong GPUs/TPUs for successful finish-tuning. The performance speeds of transcripts generated by different models differ according to accuracy levels where Whisper provides the most precise results but operates at the lowest speed and Vosk delivers the fastest results although with lower precision. As can be seen from Table 6.1, Whisper takes approximately 5.2 seconds per 10-second audio file, Wav2Vec 2.0 takes 3.8 seconds, and Vosk takes 2.4 seconds. There is apparently some trade-off between accuracy and processing time, as higher accuracy requires more processing time and effort.

Another computational load is memory consumption. Court decisions are lengthy and well-formatted, thus require additional memory allocation on paper format. Whisper and Wav2Vec 2.0 consume a lot of memory in comparison to Vosk due to their deep learning model architecture and huge model size. Real-time transcription is also not yet feasible since such models transcribe audio in batch format, thus causing latency.

Model	**Avg. Execution Time (Seconds per 10s Audio)**
Wav2Vec 2.0	5.2
Whisper	3.8
Vosk	4.6

In order to fight such computational demands, various optimization methods can be used. Model quantization will be helpful at memory usage and inference rate reduction by reducing models to lower precision (e.g., FP16 or INT8) but optimizing them. Low-resource fine-tuning is another such method where pre-trained ASR models are trained over low-capacity domain-specific corpora in order to further improve performance without full retraining. Cloud-based ASR adds processing capabilities through distributable legal transcription operations that scale across multiple networks.

Due to its superior performance Whisper remains costly for real-time operations therefore needs optimization to make it workable. Moving forward the team plans to cut down processing overhead by employing quick inference methods with batch processing and hardware speed-ups for optimal speed-accuracy performance.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Summary of Findings

The project showed that legal documentation transcription reached effective levels using advanced speech-to-text models including both Whisper as well as Vosk and Wav2Vec 2.0. Multiple machine learning models worked together in this pattern to extract text from documents while performing audio conversion and content transcription which provided an effective solution to replace human transcription work. The system would establish an automated transcription process through its pipeline even though it encountered difficulties from complex legal terms and inconsistent data quality.

The performance assessment through Word Error Rate (WER) and Character Error Rate (CER) established Whisper as the superior model for accurate transcription while BLEU Score confirmed its effectiveness. Among various computer programming models tested during the comparative analysis it was shown which programming systems functioned best for legal document transcription while demonstrating respective strengths and shortcomings. The system performed exceptionally well with structured legal texts alongside recording input but it had several limitations that primarily affected its handling of complex legal terminology and multiple speakers during recordings as well as very large datasets.

The project proved the effectiveness of merging deep learning models with natural language processing with legal document processing for developing more efficient legal transcription operations while establishing a base for further development.

7.2 Contributions to Legal AI Research

The research contributes new value to Legal AI science by transforming speech into text through models that also interpret specialized legal terminology. This system develops the capability to transform legal case audio documents into text

alongside evaluating transcription precision through multiple evaluation methods to generate insights about deep learning deployments in legal fields.

The training and evaluation models benefit from a substantial dataset which constitutes a major project contribution. The customized dataset made from 50 legal case files taken from IndianKanoon presents research opportunities through its diverse content. The study creates a new standard for evaluating speech-to-text models through the deployment of Whisper, Vosk, and Wav2Vec 2.0 on legal documents despite previous minimal research in Legal AI on this topic.

An essential contribution emerges from the creation of a live web application which allows users to submit legal files for transcription. The research implements practical functionality through which legal professionals can both access the technology along with using it effortlessly. The developed prototype system proves both the possible application of automated transcription technology for legal needs and sets a base for upcoming AI-based legal tools.

7.3 Future Enhancements and Research Directions

While the system has produced encouraging outcomes, several aspects of future work and improvements would make it even more capable. One such critical area of work in the future would be providing enrichment of data with even more varied legal cases and documents. This would generalize the models better and make them able to handle more legal terminologies and formats, with the result that transcription accuracy can be improved across different legal contexts.

One area of improvement is optimizing the fine-tuning of the speech-to-text models.Though Whisper was more accurate in transcription than the other models, there is still room for improvement with even more legally focused speech data being used for fine-tuning. The models can further be fine-tuned to do well in multi-speaker environments, which would allow them to transcribe interviews or court hearings better.

On the systems side of scalability, system optimization to handle large volumes of documents and sound files so that they process data quicker and in less time is the direction future development needs to take. This would likely involve backend optimality along with utilization of cloud computing or distributed systems in such a way that computation can be performed on efficiently usable computational resources..

The integration between the system and legal knowledge bases and natural language processing of legal argumentation can enable features that include legal case summarization and predictive analysis and legal insight generation. The system becomes more valuable to attorneys through the incorporation of these features because it offers both authenticated transcriptions and usable substance analyses of written content.

Support for multiple languages for the dominant legal regions across the world will enhance the system's potential applications and rate of usage. Through multilingual system updates the software will become globally accessible because it will perform automatic legal transcription between various legal systems across multiple languages worldwide.

This project advances AI-based legal transcription software by enhancing important aspects which will lead to less labor-intensive legal research and documentation processes.

REFERENCES

[1] Lyra A., Barbosa C.E., Santos H.S., Argôlo M., Lima Y., Motta R., Souza J.M. Automatic Transcription Systems: A Game Changer for Court Hearings. Federal University of Rio de Janeiro, 2024.

[2] Prasad R., Nguyen L., Schwartz R., Makhoul J. Automatic Transcription of Courtroom Speech. BBN Technologies, 2025.

[3] Brown J., Tanaka H., Patel D. Domain-Adapted Speech Models for Legal Transcription. Speech Communication Journal, 2024.

[4] Zhang F., Liu X., Kapoor R. Adversarial Robustness in Legal Speech Recognition Models. Neural Networks Journal, 2024.

[5] Hassan M., Oliver T., Chen B. Enhancing STT Accuracy with Legal Ontologies. Expert Systems with Applications, 2024.

[6] Singh P., Kumar A., Rao V. Real-Time Legal Transcription Using Edge Computing. Proceedings of the International Conference on AI & Law, 2023.

[7] Williams R., Fernandez C., Kim S. Multimodal AI for Legal Analytics: Integrating Speech and Text Models. IEEE Transactions on Affective Computing, 2024.

[8] Ribeiro de Faria J., Xie H., Steffek F. Automatic Information Extraction From Employment Tribunal Judgments Using Large Language Models. University of Cambridge, 2025.

[9] Lee K., Zhang W., Gupta A. Speech-to-Text Models for Legal Transcription: A Comparative Study. Neural Computing and Applications, 2023.

[10] Adams R., Nguyen P., White J. Ethical Challenges in AI-Based Legal Transcription. AI & Ethics Journal, 2023.

[11] Johnson M., Li Y., Carter D. Fine-Tuning Transformer Models for Legal Text Processing. Artificial Intelligence and Law Journal, 2023.

[12] Patel S., Roy T., Choudhury R. Legal Document Classification Using Deep Learning and NLP. ACM Transactions on Information Systems, 2024.

[13] Wang Y., Gao J., Chen J. Deep Learning Algorithm for Judicial Judgment Prediction Based on BERT. IEEE, 2020.

[14] Anand D., Wagh R. Effective Deep Learning Approaches for Summarization of Legal Texts. Journal of King Saud University – Computer and Information Sciences, 2019.

[15] Oliveira L., Dasgupta S., Wang Y. Real-Time Speech Processing in Multilingual Legal Environments. Computational Linguistics Journal, 2024.

APPENDIX

Appendix A: Data Preprocessing

The following steps were utilized for preparing the dataset before performing speech-to-text transcription:

- The BERT-based system extracted judgment texts from legal case PDFs after removing metadata and text that did not matter for the study.
- Text-to-Speech Conversion: Text was transformed into .flac audio format by Google Text-to-Speech (gTTS) with a normalized format of 16kHz, mono-channel, and 16-bit PCM encoding.
- Audio Preprocessing:
  - Format Conversion: .flac format to .wav conversion was conducted for the support of STT models.
  - The voice activity detection removed non-speech segments which improved the accuracy of transcription results.
  - Normalization: Audio normalization was done to maintain an equal volume across all samples.
  - Feature Extraction: MFCC and LPC were computed for better representation of speech.

Appendix B: Model Training and Evaluation

The transformer model known as Whisper operates on extensive multilingual data to provide legal speech transcription services. Wav2Vec 2.0 acquires direct speech representations through unprocessed audio to increase its ability to understand domain-specific language. Vosk operates as a quick Kaldi model that converts speech to text but needs automatic learning from specific domains to recognize proper legal discussions.

The system used multiple important metrics to assess the performance of its operational models. WARE acts as an approximation of speech recognition errors through inaccurate word transcription percentages. The Character Error Rate measurement method detects spelling errors at character level to identify small

spelling mistakes. This scales the accuracy of automated transcription content versus its legal reference documents.

Appendix C: Error Analysis and Model Optimization

- Error Distribution Analysis
  - Substitution Errors: Incorrect words replacing the correct ones, usually on legal words.
  - Transcription errors occur when extraneous words are added to the text which negatively affects sentence readability.
  - The transcription includes delete errors which result in missing words and leads to reduced document completeness.
- Confusion Matrix Analysis
  - A confusion matrix for each of the models was generated to inspect typically mislabeled legal phrases. Whisper got the highest level of accuracy, and Vosk showed the highest confusion across legal phrases.

Appendix D: Visualizations

Performance evaluation graphs enabled the comparison between the accuracy and effectiveness levels of different models. The performance metrics between fluent speech and accurate results demonstrated Whisper delivered the most effective balance of both factors, surpassing all other models on the WER vs. BLEU Score Graph. Each program demonstrated continuous accuracy improvement throughout the training periods as part of optimization and fine-tuning processes.

The evaluation process used actual legal documents to check the structural accuracy of models along with identifying patterns of errors in their transcription output. After model testing the subsequent evaluation process assessed how well the models performed when exposed to speech which contained different levels of noise exposure.

Whisper maintained reliable high performance but the alternative models delivered inconsistent results primarily in audios with background noise.

Appendix E: Challenges and Learnings

The implementation of Whisper needed accurate beam search parameter adjustment to reach its best possible accuracy level. By optimizing its specific elements the model reached maximum performance during the transcription of complex legal language. The Wav2Vec 2.0 achieved its prime performance level while performing fine-tuning with legal domain speech corpora which enhanced its ability to process specialized legal terminology. The Vosk audio processing application demanded special language models since regular models exhibited poor performance in recognizing legal jargon.

Transcription accuracy improved through the combination of proper audio signal representation methods which included MFCC and LPC features. The particular pairing between these features provided the model with an extraordinary capability to track speech features accurately. Using Voice Activity Detection (VAD) proved essential because it removed silent segments which enhanced transcription efficiency.

The main issue during this study was the inability to successfully regulate background noise together with speaker variability. Vosk suffered difficulties caused by noisy conditions because it lacked sufficient training data but Whisper performed better in overall robust conditions. Furthermore, accented speakers had to be dealt with by using specially constructed adaptation processes, as pronunciation variability led to poor transcription performance across all models.