System Design Document

for

Kaldi ASR Team

Prepared by
Adam Gallub, Milan Haruyama, Tabitha O'Malley, David Serfaty, Tahmina Tisha

Revision History

Name	Date	Reasons For Change	Version
Tabitha, Milan, David, Max, Tisha, Adam	09/29/2023		V1.0
Tabitha	10/24/2023	Writing Section:1.2.1	V2.1
Tisha, Tabitha, Milan	10/24/2023	Rewriting the section: 2.2	V2.2
Tabitha, Tisha, Milan	10/24/2023	Writing the section, Rewriting, and editing: 1.2	V2.3
Tabitha	10/25/2023	Writing Sections: 2.1, 1.5	V2.4
Tabitha	10/26/2023	Writing Sections: 1.1, 1.2.2, 1.3	V2.5
David	10/26/2023	Writing Sections: 1.2, 1.5, 3.1, 3.2	V2.6
Milan	10/26/2023	Writing Sections: 1.1, 1.2, 1.5	V2.7
Adam	10/28/2023	Asking TA: 2.1 Write Section: 4.1	V2.8
Tisha	10/28/2023	Asking TA: 2.1 Writing Section: 2.1, 5.1	V2.9
Tabitha	10/29/2023	Writing/Rewriting Sections: : 1.2.1, 2.1, 2.2, 3.1, 3.2, 4, 4.1, 4.2, 5.2	V2.10
David	10/29/2023	Writing/Rewriting Sections: 1.2.1, 1.2.2, 1.2.3, 2.1. 2.2, 3.1, 3.2, 4, 4.1, 5, 5.1, 6	V2.11
Tisha	10/29/2023	Writing/Rewriting Section: 2.1	V2.12
Milan	10/29/2023	Editing All Sections	V2.13
Milan	10/30/2023	Editing All Sections	V2.14
Tabitha	10/30/2023	Rewriting Section: 2.1	V2.15
Tabitha	10/31/2023	Updating Models Editing Sections Writing Section: 4.2	V2.16
David	10/31/2023	Rewriting sections: 1.5, 5.2 Editing sections Updating models (context, use case, DFD)	V2.17
Milan	10/31/2023	Editing all sections	V2.18

Tabitha	11/05/2023	Class Diagram: 2.1	V3.1
David	11/05/2023	Class Diagram: 2.1	V3.2
Tisha	11/05/2023	Edited Section 2.2	V3.3
Adam	11/07/2023	Writing/Rewriting: 3.1, 3.2	V3.4
Tisha	11/07/2023	Writing/Rewriting 3.1	V3.5
Adam	11/07/2023	Writing/Rewriting 3.1	V3.5
Milan	11/07/2023	Editing all Sections, reformatting Table of Contents	V3.6
David	11/11/2023	Editing and Rewriting: 4.2 Updating DFD model	V3.7
Tabitha	11/11/2023	Editing and Rewriting: 4.2	V3.8
Tisha	11/11/2023	Editing 4.1	V3.9
Milan	11/11/2023	Editing all sections	V3.10
Milan	11/11/2023	Editing: 2.2	V3.11
Tisha	11/12/2023	Edited Section 4.1 (added the last Use Case)	V3.12
Tabitha	11/15/2023	Update all DFD Models Updating/Rewriting Section; 4.2 Editing/Adding: 5.1	V3.13
Tisha	11/16/2023	section 4.1 (needs to be checked)	V3.14
Tabitha	11/16/2023	Reading and Commenting Sections : 2-5 For accuracy to the current requirements Update Classes Diagram Update/Rewrite Section: 2.1	V3.15
David	11/18/2023	Editing and rewriting Use Cases Remaking Use Case Diagram V2.1.1 Making Use Case Diagram V2.2.2 Editing DFD V3.1.3, V3.2.2, V3.3.1 Rewriting and editing 4.1	V3.16
Tabitha	11/18/2023	Editing and rewriting Use Cases Remaking Use Case Diagram V2.1.1 Making Use Case Diagram V2.2.2 Editing DFD V3.1.3, V3.2.2, V3.3.1 Rewriting and editing 4.1	V3.17
Tisha	11/18/2023	Rewriting section 5.2 (needs to checked for accuracy)	V3.18
Milan	11/18/2023	Editing all sections	V3.19
David	11/19/2023	Editing 4.2, 2.1, 5.1, 5.2 Editing Class Diagram	V3.20

Tabitha	11/19/2023	Editing: 5.1, 5.2	V3.21
Milan	11/19/2023	Editing all sections	V3.22
Milan	11/20/2023	Reviewing/editing all sections	V3.23
Tabitha	11/22/2023	Section 1.2.3 and Figure 03	V3.23
Adam, Milan, Tabitha, David, Tahmina	01/22/2024		V4.0
David	01/23/2024	Updated 1.2.1 Model Edited 1.2.1	V4.1
Tabitha	01/23/2024	Highlighted sections to work on Edited 1.2.1	V4.1
Milan	01/25/2024	Updated Section 1.1 Updated Section 1.2	V4.2
Milan	01/29/2024	Reviewing all sections	V4.3
Milan	01/30/2024	Finished Section 1.1	V4.4
Tabitha	01/30/2024	Editing Section 4.2	V4.5
Tabitha	02/01/2024	Editing Section 4.2	V4.6
Milan	02/04/2024	Editing Section 1.2.2, 4.2	V4.7
Tabitha	02/04/2024	Editing Section 4.2	V4.8
Milan	02/05/2024	Reviewing/Editing all sections	V4.9
Tabitha	02/14/2024	Section 4.2.1, 4.2.2	V5.1
Tabitha	02/15/2024	Section 4.2.2	V5.3
Tabitha	02/22/2024	Section 2.1	V5.4
Milan	02/26/2024	Section 4.1	V5.5
Tabitha	02/26/2024	Section 2	V5.6
Tabitha	02/27/2024	Update Figure Numbers	V5.7
Milan	02/27/2024	Section 4.1	V5.8
Milan	02/29/2024	Section 4.1	V5.9
Milan	02/02/2024	Section 4.1	V5.10
Milan	02/03/2024	Section 4.1	V5.11
Milan	03/21/2024	Added page numbers Added revision comments	V6.0
Tabitha	03/23/2024	Updated all Models to current versions	V6.1

Milan	03/23/2024	Editing Section 2.1 Editing section 4.1	V6.2
Milan	04/04/2024	Editing Section 2.1 Editing Section 4.1	V6.3
Milan	04/09/2024	Editing Section 4.1	V6.4
Tabitha	04/09/2024	Editing Section 4.1	V6.5
Milan	04/13/2024	Finished Section 4.1	V6.6

TABLE OF CONTENTS

1 INTRODUCTION **7**
1.1 Purpose and Scope 7
1.2 Project Executive Summary **7**
1.2.1 System Overview **8**
1.2.2 Design Constraints **8**
1.2.3 Future Contingencies **9**
1.3 Document Organization **9**
1.4 Project References **10**
1.5 Glossary **11**
2 SYSTEM ARCHITECTURE **12**
2.1 System Software Architecture **12**
2.2 Internal Communications Architecture **22**
3 HUMAN-MACHINE INTERFACE **23**
3.1 Inputs **23**
3.2 Outputs **23**
4 DETAILED DESIGN **25**
4.1 Software Detailed Design **25**
4.1.1 Audio Data Preprocessing **26**
4.1.2 Audio Data Transcription **26**
4.1.3 Model Training **26**
4.2 Internal Communications Detailed Design **29**
4.2.1 Preparation **29**
4.2.2 Decoding **33**
4.2.3 Output **34**
5 EXTERNAL INTERFACES **36**
5.1 Interface Architecture **36**
5.2 Interface Detailed Design **36**
6 SYSTEM INTEGRITY CONTROLS **37**

SYSTEM DESIGN DOCUMENT

1 INTRODUCTION

1.1 Purpose and Scope

Learning to communicate with Air Traffic Control (ATC) is a daunting task for many new aircraft pilots. Aviation phraseology is often the biggest source of miscommunication (despite being designed to mitigate it) due to the highly intricate vernacular. The idiosyncrasies present within aviation phraseology require hundreds of hours of training for student pilots to learn. While there exist a few resources (e.g., the LiveATC website) for student pilots to study aviation phraseology, these resources do little to accommodate the major learning curve due to a lack of transcriptions of spoken speech for student pilots to read.

As such, the RTube web application shall bridge the learning gap faced by many student pilots by providing the ability to transcribe live ATC transmissions into text in real-time using an acoustic automatic speech recognition (ASR) model, as well as providing a live interface for users to track the flight paths of aircraft. With the ability to transcribe speech to text in real-time, student pilots can dramatically reduce the amount of training required to understand spoken aviation phraseology. In addition, the live interface helps students better understand different contexts in which specific phrases are used.

Currently, the RTube web application utilizes an end-to-end ASR model developed using the NVIDIA NeMo Toolkit. The Kaldi ASR Team aims to replace the end-to-end ASR model with an acoustic model developed using the Kaldi ASR Toolkit. The main advantage of replacing the end-to-end ASR model with an acoustic one is the faster training speed of the latter due to requiring less training data than end-to-end models. However, the improved training efficiency comes at the cost of a significantly more difficult development life cycle, as the Kaldi ASR Toolkit is highly complex and poorly documented for people unfamiliar with ASR model development.

1.2 Project Executive Summary

The Kaldi ASR Team solely focuses on the development of the acoustic ASR model that shall be utilized by the RTube web application. Included in this section of the document is an overview of the developed ASR model, design constraints for future acoustic ASR models, and general project contingencies.

1.2.1 System Overview

The Kaldi-based ASR system utilizes acoustic models that use phones and triphones to model pronunciation, a lexicon that converts pronunciations to words, and a language model that converts words into text. After the audio has been fully transcribed, the ASR model returns the text to the user.

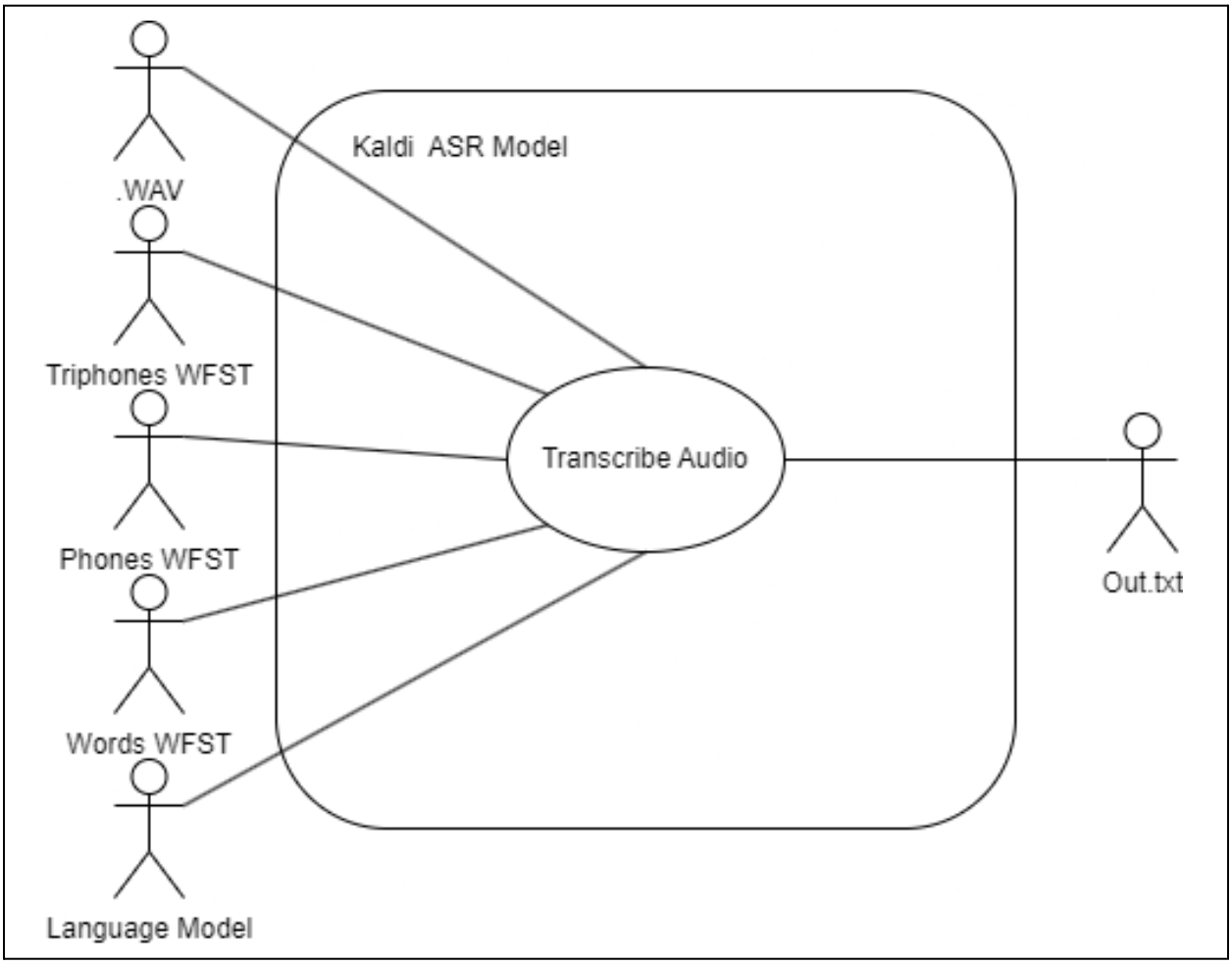

Figure 01: Kaldi ASR Model Use Case Diagram V5.1.1

1.2.2 Design Constraints

The primary constraints that limit the scope of any development done by the Kaldi ASR Team revolve around the difficulties of working with the Kaldi ASR Toolkit, the complexity of speech recognition in general, and the many applications and packages that have to be installed that are not documented.

Some of the difficulties of working with the Kaldi ASR Toolkit include its abhorrently large 12.5 GB installation package, its extremely high-level documentation, its lack of a graphical user interface (GUI), and the number of unlisted dependencies that are required to both install and run it. In addition, many of the helper applications and packages themselves are tedious to install and require a number of unlisted dependencies. Overall, setting up and using the Kaldi ASR Toolkit is its own laborious task, which is why a user installation manual is currently being written by the Kaldi ASR Team.

Speech recognition on its own must contend with the variability in tone, pitch, realization, and speed of spoken speech between different speakers. When used for the application of transcribing ATC vernacular, the complexity of speech recognition further increases due to the presence of noise interference during live radio transmissions (e.g., engine noise, turbulence, static, etc.), and the idiosyncrasies of aviation phraseology.

1.2.3 Future Contingencies

The nature of this project generates potential problems including but not limited to failure to properly train an acoustic ASR model; inadequate training of the ASR model due to dataset insufficiency; failure of the ASR model to perform live transcriptions; and so on.

Some possible solutions to the aforementioned hindrances include the use of more powerful machines to run the Kaldi ASR toolkit, and acquisition of a dataset that is either larger in size, more varied, or both.

Currently, the preprocessing portion of the ASR model has been planned. However, more research must be conducted to find a base model that works well with the preprocessing script in place. If no such model exists that produces an adequate word error rate (WER) with the current preprocessing script, the best inadequate model shall be used.

Lastly, due to the aforementioned difficulty of working with the Kaldi ASR Toolkit, the Kaldi ASR Team is actively researching to better understand the toolkit and aid in the development life cycle of the acoustic ASR model. Some team members are even taking a masters-level course revolving around the toolkit to further enhance their understanding

1.3 Document Organization

This document shall discuss system architecture, human-machine interfaces, detailed design, external interfaces, and system integrity control in their own respective sections. The System Architecture section shall discuss the high-level structure and functionality of the acoustic ASR model developed by the Kaldi ASR Teamt. The Human-Machine Interfaces section shall discuss the inputs and outputs of the aforementioned ASR model. The Detailed Design section shall discuss the lower level of detail of the ASR model. The External Interfaces section shall have a basic description of how other systems interface with the Kaldi ASR Toolkit and the ASR model. Lastly, the System Integrity Control section shall cover any sensitive data that could threaten the integrity of the system if modified, and measures taken by the Kaldi ASR Team to safeguard it.

1.4 Project References

Fagan, Jonathan. “What Is a Good Accuracy Level for Transcription?” *University Transcription Services*, 3 June 2020, www.universitytranscriptions.co.uk/what-is-a-good-accuracy-level-for-transcription/. Accessed 26 Oct. 2023

Kaldi ASR Team Product Vision Statement. Kaldi ASR Team. 19 September 2023. Kaldi ASR Team Drive.

Kaldi ASR Team Software Design Document. Kaldi ASR Team. 19 November 2023. Kaldi ASR Team Drive.

Ravihara, Ransaka. “Gaussian Mixture Model Clearly Explained.” *Medium*, Towards Data Science, 11 Jan. 2023, www.towardsdatascience.com/gaussian-mixture-model-clearly-explained-115010f7d4cf/.

“About Pandas.” *Pandas*, www.pandas.pydata.org/about/. Accessed 26 Oct. 2023.

“LiveATC FAQ.” *LiveATC.Net - Listen to Live Air Traffic Control over the Internet!*, www.liveatc.net/faq/. Accessed 26 Oct. 2023.

“A Python Library to Read/WRITE EXCEL 2010 Xlsx/XLSM Files¶.” *Openpyxl*, www.openpyxl.readthedocs.io/en/stable/. Accessed 26 Oct. 2023.

“What Is Kaldi?” *Kaldi*, www.kaldi-asr.org/doc/about.html/. Accessed 26 Oct. 2023.

“What Is Speech Recognition?” *IBM*, www.ibm.com/topics/speech-recognition/. Accessed 26 Oct. 2023.

Chester F. Carlson Center for Imaging Science | College of Science | RIT, www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf/. Accessed 1 Nov. 2023.

“About FFmpeg” *FFmpeg,* www.ffmpeg.org/about.html/. Accessed 19 Nov. 2023.

Linguistic Data Consortium. (1994). LDC94S14A: ATIS-3 Training Text, Version 1.1. University of Pennsylvania. https://catalog.ldc.upenn.edu/LDC94S14A. Accessed 25 February 2023

IDIAP Research Institute. (2024). Atco2 Corpus. GitHub. https://github.com/idiap/atco2-corpus. Accessed 25 February 2023

1.5 Glossary

Term	Definition
ATC	***Air Traffic Control;*** the service that elicits communications between pilots and helps to prevent air traffic accidents.
ASR	***Automatic Speech Recognition;*** the ability for computers to recognize and translate spoken speech.
CLI	***Command-Line Interface;*** text-based interface that allows interaction from the user to the computer program.
DNN	***Deep Neural Network;*** a machine learning technique that represents learning and processing data in artificial neural networks.
ERAU	***Embry Riddle Aeronautical University;*** an aviation-centered university located in Daytona Beach, Florida.
FFmpeg	***Linux utility;*** a portable open-source utility that allows users to decode, encode, transcode, multiplex, demultiplex, stream, filter, and play most human- or machine-made multimedia.
FFT	***Fast Fourier Transform;*** algorithm used to obtain the spectrum or . frequency content of a signal.
General American English	The most spoken variety of the English language in the United States.
GMM	***Gaussian Mixture Model;*** used to calculate the distance between the MFC feature vector and the HMM state.
HMM	***Hidden Markov Model;*** used to find the state locations of the phonemes..
IPA	***International Phonetic Alphabet:*** an alphabetic system of phonetic notation developed by the International Phonetic Association; used to represent speech sounds in a standardized format.
Lexicon	*pertaining to speech;* a library of words that is understood by the language model.
MFC	***Mel-Frequency Cepstrum;*** a representation of the short-term power spectrum of a sound
MFCCs	***Mel-Frequency Cepstral Coefficients;*** the coefficients that a MFC is comprised of
NLP	***Natural Language Processing;*** the culmination of computer science, linguistics, and machine learning.
Phone	*pertaining to speech;* a distinct speech sound or gesture;
Phoneme	*pertaining to speech;* a set of phones that can distinguish one word from another
Triphone	*pertaining to speech;* a sequence of three consecutive phonemes
WER	***Word Error Rate;*** the rate at which error in words occurs

2 SYSTEM ARCHITECTURE

2.1 System Software Architecture

This section shall provide a brief overview of each class of the Kaldi ASR Toolkit. A more thorough analysis of each class is provided in Section 4.1 of this document.

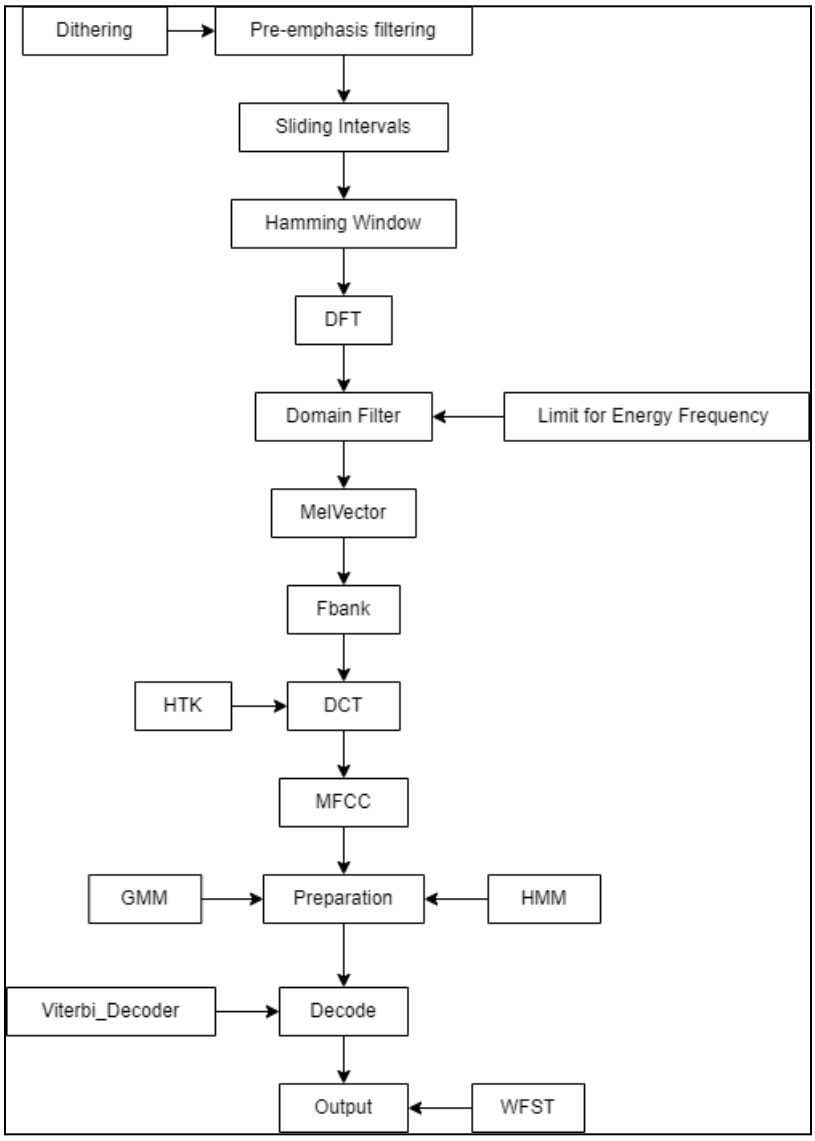

Figure 02: Simplified Class Diagram

2.1.1 Dithering

This class shall add Gaussian noise to the audio so that it never equals zero.

2.1.2 Pre-Emphasis Filtering

Ths class shall filter the audio to ensure a frequency range between 8 to 16 kHz.

2.1.3 Sliding Intervals
This class shall frame the audio into 25-millisecond intervals.

2.1.4 Hamming Window
The class shall window the data to prevent spectral leakage.

2.1.5 Discrete Fourier Transform (DFT)
This class shall convert the windowed data from the time domain to the frequency domain.

2.1.6 Limit for Energy Frequency
This class shall find the maximum energy frequency.

2.1.7 Domain Filter
This class shall filter the data so that it does not exceed the maximum energy

2.1.8 Mel Vector
This class shall convert the data into a mel vector.

2.1.9 Fbank
This class shall calculate the logarithm of the mel vector.

2.1.10 Hidden Markov Model Toolkit (HTK)
This class shall convert frequencies to melody numbers.

2.1.11 Discrete Cosine Transform (DCT)
This class shall mirror and smooth the data.

2.1.12 Mel-Frequency Cepstral Cepstrum (MFC)
This class shall convert frequency-domain frames into a 39-dimensional MFCC feature vector represented as a 39-dimensional array.

2.1.13 Gaussian Mixture Model (GMM)
This class shall calculate the expectation maximization of the MFCC feature vector in reference to phoneme Gaussian distribution.

2.1.14 Hidden Markov Model (HMM)
This class shall determine the hidden state sequence.

2.1.15 Preparation
This class shall prepare the data for decoding

2.1.16 Viterbi Decoder
This class shall find the most probable sequence of states.

2.1.17 Decode
This class shall prepare a sequence for output.

2.1.18 Weighted Finite State Transducer (WFST)
This class shall convert the sequence of states into sentences.

2.2 Internal Communications Architecture

The Kaldi ASR Toolkit provides users with the ability to create an acoustic ASR model that receives audio files as input and generates text transcriptions of the audio as output. The model exclusively accepts WAV files and exclusively outputs text files; all other inputted file types must be converted into WAV files using a file conversion utility (e.g., FFmpeg). The generated text files shall only contain words based on General American English spelling conventions; no punctuation or grammatical marks shall be included. The words shall be capitalized in the text file if they are found in the lexicon; if not, they shall be in lowercase, and appended to the non-lexicon library. More thorough analyses of the model created by the Kaldi ASR Team is provided in Section 4 of this document.

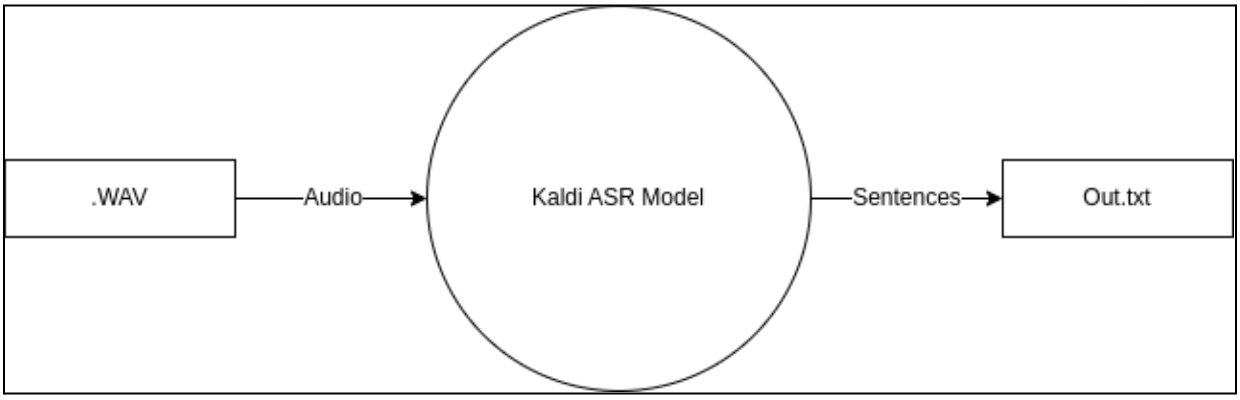

Figure 03: Acoustic ASR Model Level 0 Data Flow Diagram V4.1.1

3 HUMAN-MACHINE INTERFACE

Please note that the following subsections are in reference to the sample ASR model provided by the Kaldi ASR Toolkit. Once the Kaldi ASR Team develops its own acoustic ASR model to work alongside the preprocessing script, the following subsections shall be updated in reference to it.

3.1 Inputs

Input shall be given as a terminal command in the format, “python3 main.py filename.wav” (*Figure 04*). The argument, “python3”, calls the execution of a Python3 file. The file in question, “main.py, is a script that initiates the execution of the ASR model. Lastly, the argument, “filename.wav”, calls the WAV file that shall be transcribed by the ASR model. The file shall be located in the same directory as the script. In essence, the script shall only execute if it is called alongside a single WAV file in the Ubuntu terminal (*Figure 04*). Any input that does not adhere to this format shall cause the system to throw an exception and print a message in the terminal (*Figures 05, 06, 07*), explained in detail in the proceeding subsection.

Any input that does not adhere to this format shall cause the system to throw an exception and print a message in the terminal (*Figures 05, 06, 07*), explained in detail in the proceeding subsection.

3.2 Outputs

Calling any file type other than WAV shall cause the script to throw an exception and print, “Provided filename does not end in '.wav'” (*Figure 05*). Calling multiple files of any type shall cause the script to throw an exception and print, “Too many arguments provided. Aborting” (*Figure 06*). In the absence of a WAV file, the system shall attempt a traceback to the most recently called WAV file; if unsuccessful, the system shall throw a “ValueError” exception and print, “No .wav file in the root directory” (*Figure 07*).

Upon successful execution of the aforementioned Python3 script, the ASR model shall output transcribed into a text file called “out.txt” stored in the directory “/kaldi/egs/mini_librispeech/s5/”. The file shall allow users to read the transcription performed. The transcription performed shall have a minimum transcription accuracy of 80%. The written text shall be in Courier font, and shall consist of words based on General American English spelling conventions. If the words are found in the lexicon, they shall be written in uppercase. If not, they shall be written in lowercase. No grammatical or punctuation marks shall be included (*Figure 09*). Upon every execution of the Python3 script, “out.txt” is deleted from the directory and rewritten.

There shall be another output called “outrescore.txt”. This text file shall not appear in the final product, hence its absence in *Figure 08*.

The program shall print out the guessed value of non-lexicon words in the terminal. For instance, Gettysburg (*Figure 09*) shall be printed out and given a score. Another output in the terminal is the overall cost per frame of the non-lexicon word. Finally the system shall print out in the terminal how many non-lexicon words were given values and how many failed.

```
redhocus@LAPTOP-GTI83R2U:~/project/kaldi/egs/kaldi-asr-tutorial/s5$ python3 main.py gettysburg.wav
tree-info exp/chain_cleaned/tdnn_1d_sp/tree
tree-info exp/chain_cleaned/tdnn_1d_sp/tree
```

Figure 04: Sample ASR Model Successful Input (with code execution snippet)

```
redhocus@LAPTOP-GTI83R2U:~/project/kaldi/egs/kaldi-asr-tutorial/s5$ python3 main.py gettysburg.flac
Traceback (most recent call last):
  File "/home/redhocus/project/kaldi/egs/kaldi-asr-tutorial/s5/main.py", line 16, in <module>
    raise ValueError("Provided filename does not end in '.wav'")
ValueError: Provided filename does not end in '.wav'
```

Figure 05: Sample ASR Model Unsuccessful Input; non-WAV input exception

```
redhocus@LAPTOP-GTI83R2U:~/project/kaldi/egs/kaldi-asr-tutorial/s5$ python3 main.py gettysburg.wav gettysburg.wav
Traceback (most recent call last):
  File "/home/redhocus/project/kaldi/egs/kaldi-asr-tutorial/s5/main.py", line 18, in <module>
    raise ValueError("Too many arguments provided. Aborting")
ValueError: Too many arguments provided. Aborting
```

Figure 06: Sample ASR Model Unsuccessful Input; too many arguments exception

```
redhocus@LAPTOP-GTI83R2U:~/project/kaldi/egs/kaldi-asr-tutorial/s5$ python3 main.py
Traceback (most recent call last):
  File "/home/redhocus/project/kaldi/egs/kaldi-asr-tutorial/s5/main.py", line 10, in <module>
    FILE_NAME_WAV = glob.glob("*.wav")[0]
IndexError: list index out of range

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/redhocus/project/kaldi/egs/kaldi-asr-tutorial/s5/main.py", line 12, in <module>
    raise ValueError("No .wav file in the root directory")
ValueError: No .wav file in the root directory
```

Figure 07: Sample ASR Model Unsuccessful Input; no file in directory exception

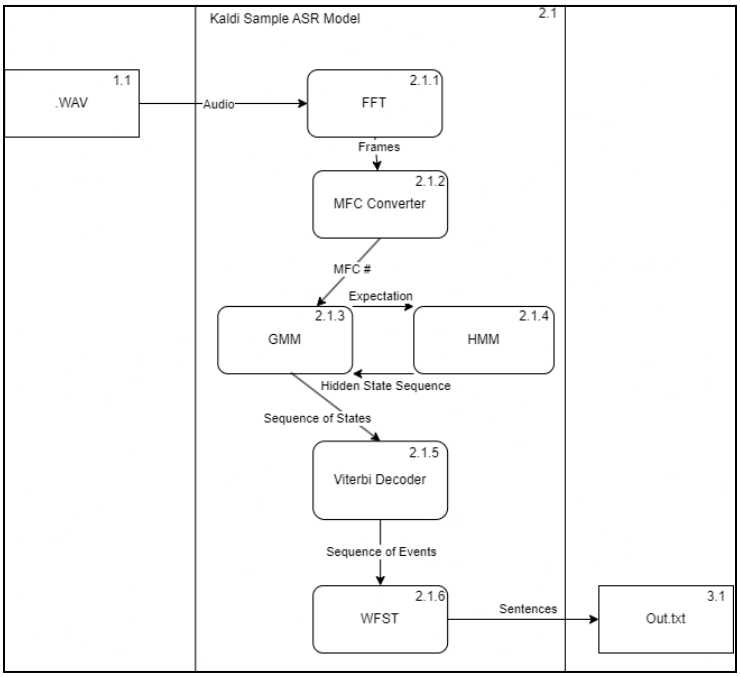

Figure 08: Sample ASR Model Level 1 Data Flow Diagram V5.1.1

```
gettysburg FOUR SCORE AND SEVEN YEARS AGO
OUR FATHERS BROUGHT FORTH ON THIS
CONTINENT A NEW NATION CONCEIVED A LIBERTY
AND DEDICATED TO THE PROPOSITION THAT ALL
MEN ARE CREATED EQUAL
```

Figure 09: Sample ASR Model Output

4 DETAILED DESIGN

4.1 Software Detailed Design

The following subsections provide a detailed description of the Mini-Librispeech corpus provided by the Kaldi ASR Toolkit shall be the model to be trained. *Figure 10* illustrates how users and other actors shall interact with the ASR model. *Figure 11* illustrates how the ASR model shall be trained. Below *Figure 11* is a detailed analysis of each class of the ASR model.

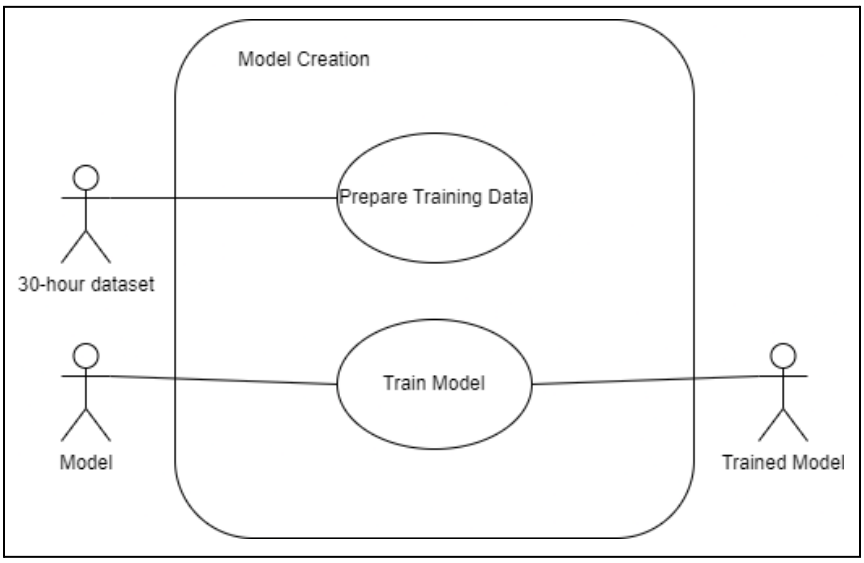

Figure 10: ASR Model Training Use Case Diagram V5.2.2

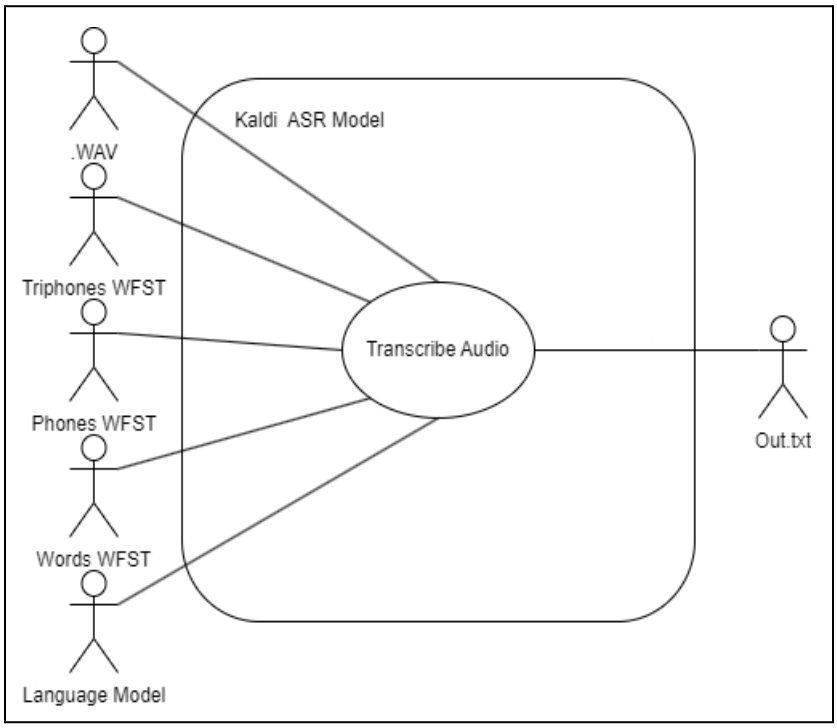

Figure 11: ASR Model Use Case Diagram V5.1.1

Dithering		
This class shall add Gaussian noise to the audio so that it never equals zero.		
Variables		
Inputs	**Outputs**	**Interal**
Audio wavFile	ditheredAudio	gaussianNoise
Methods		
addNoise(wavFile, gaussianNoise)		

Pre-emphasis filtering		
Ths class shall filter the audio to ensure a frequency range between 8 to 16 kHz.		
Variables		
Inputs	**Outputs**	**Interal**
ditheredAudio	filteredAudio	a
		n
Methods		
filterFrequencies(ditheredAudio, a, n)		

Sliding Intervals		
This class shall frame the audio into 25- millisecond intervals.		
Variables		
Inputs	**Outputs**	**Interal**
filteredAudio	framedAudio	n
Methods		
framingAudio(filteredAudio, n)		

Hamming Window		
The class shall window the data to prevent spectral leakage.		
Variables		
Inputs	**Outputs**	**Interal**
filteredAudio	framedAudio	n
		N
Methods		
WindowingAudio(framedAudio, N, n)		

Discrete Fourier Transform (DFT)		
This class shall convert the windowed data from the time domain to the frequency domain.		
Variables		
Inputs	**Outputs**	**Interal**
WindowAudio	frequencyDomainAudio	N
		K
		k
		j
Methods		
discreteForierTransform(WindowAudio, N, K, j, k)		

Limit for Energy Frequency		
This class shall find the maximum energy frequency.		
Variables		
Inputs	**Outputs**	**Interal**
	maxEnergyFrequency	kLower
		kUpper
Methods		
maxEnergyCalc(kLower, kUpper)		

Domain Filter		
This class shall filter the data so that it does not exceed the maximum energy		
Variables		
Inputs	**Outputs**	**Interal**
frequencyDomainAudio	filterDomainAudio	k
maxEnergyFrequency		K
Methods		
filterDomainfrequency(maxEnergyFrequency, k, K, frequencyDomainAudio)		

MelVector		
This class shall convert the data into a mel vector.		
Variables		
Inputs	**Outputs**	**Interal**
filterDomainAudio	melVector	
Methods		
vectorData(filterDomainAudio)		

Fbank		
This class shall calculate the logarithm of the mel vector.		
Variables		
Inputs	**Outputs**	**Interal**
melVector	logMelVector	
Methods		
Logarithm(melVector)		

HTK		
This class shall convert frequencies to melody numbers.		
Variables		
Inputs	**Outputs**	**Interal**
frequency	melodyNumber	
Methods		
calculateMelodyNumber(frequency)		

Discrete Cosine Transform (DCT)		
This class shall mirror and smooth the data.		
Variables		
Inputs	**Outputs**	**Interal**
melNumber	dctVector	k
logMelVector		M
Methods		
discreteCosineTransform(melodyNumber, logMelVector, k, M)		

Mel-Frequency Cepstrum (MFC)		
This class shall convert frequency-domain frames into a 39-dimensional MFCC feature vector represented as a 39-dimensional array.		
Variables		
Inputs	**Outputs**	**Interal**
dctVector	MFCC[]... 37…[] featureVector	
Methods		
convertToMFCC(dctVector)		

Gaussian Mixture Model (GMM)		
This class shall calculate the expectation maximization of the MFCC feature vector in reference to phoneme Gaussian distribution.		
Variables		
Inputs	**Outputs**	**Interal**
initalProb	gamma	
mean		
variance		
Methods		
recalculateExepectation(gamma)		
maximixizeValues(mean, variance, initialProb)		

Hidden Markov Model (HMM)		
This class shall determine the hidden state sequence.		
Variables		
Inputs	**Outputs**	**Interal**
setOfStates	hiddenStateSequence	transitionMatrix
setOfObservations		emissionMatrix
setOfInitialProbabilities		
Methods		
calculateSequenceOfEvents(setOfStates, setOfObservations, setOfInitialProbabilities, transitionMatrix, emissionMatrix)		

Preparation		
This class shall prepare the data for decoding.		
Variables		
Inputs	**Outputs**	**Interal**
MFCC	hiddenStateSequence	
GMM		
HMM		
Methods		
prepareAudioForDecoding()		

Viterbi_Decoder		
This class shall find the most probable sequence of states.		
Variables		
Inputs	**Outputs**	**Interal**
stateSpace	mostProbableSequenceOfState	transitionMatrix
initialProbabilities		emissionMatrix
- observationSpace		
Methods		
calculateSquenceOfEvents(stateSpace, observerationSpace, initailProbablities, transitionMatrix, emissionMatrix)		

Decode		
This class shall prepare a sequence for output.		
Variables		
Inputs	**Outputs**	**Interal**
Viterbi_Decoder	mostProbableSequenceOfState	
Methods		
prepareSequenceForOutput()		

WFST		
This class shall convert the sequence of states into sentences.		
Variables		
Inputs	**Outputs**	**Interal**
mostProbableSequenceOfState	sentance	triphone
		monophone
		word
		Monophones
Methods		
gettriphone(mostProbableSequenceOfState)		
convertfromtritomono(triphone)		
createarrayof monophones(monophones)		
convertfrommonotoword(Monophones[], File lexicon)		
compilesentence(word)		

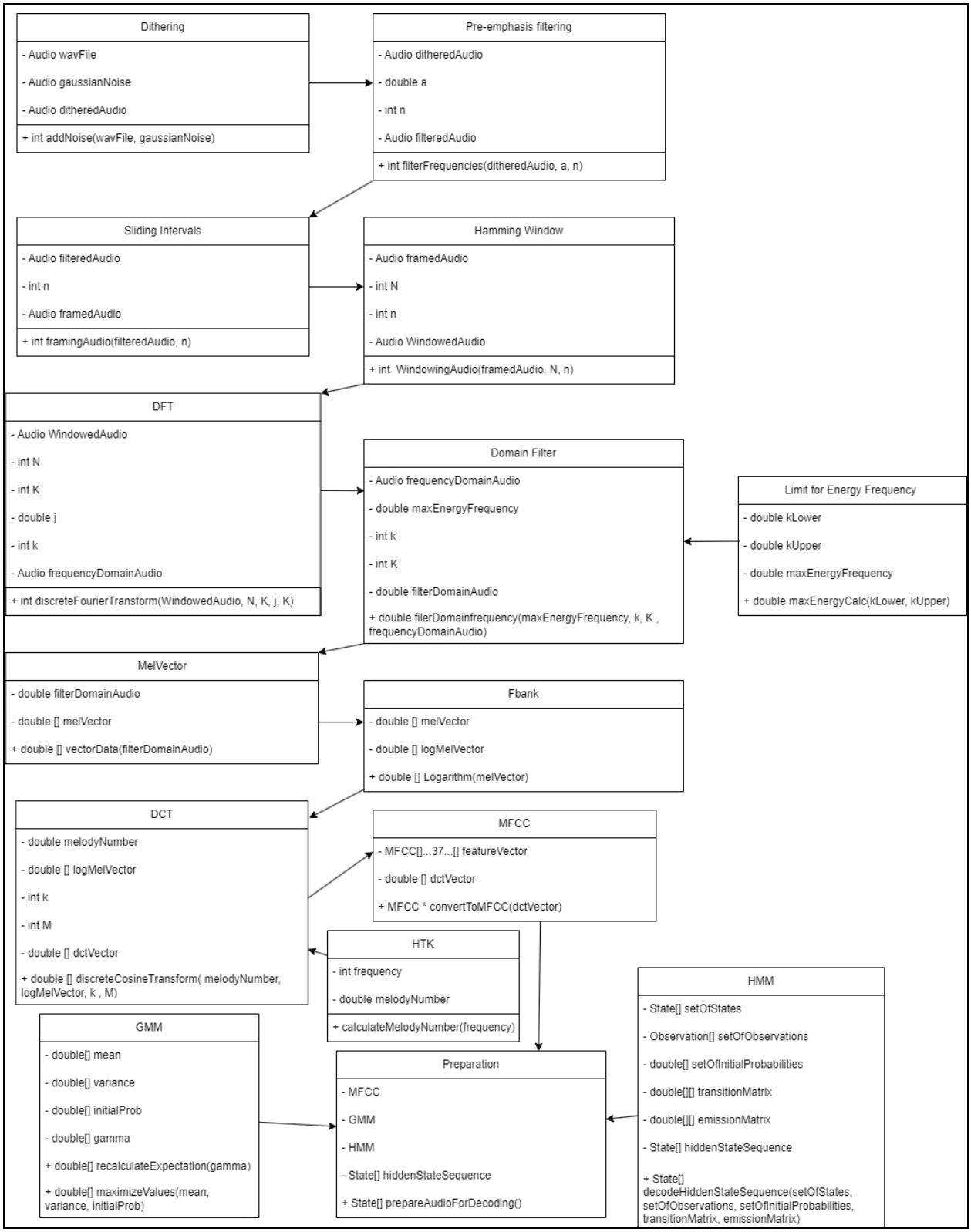

Figure 12: ASR Model Preparation Stage Class Diagram V5.2.2

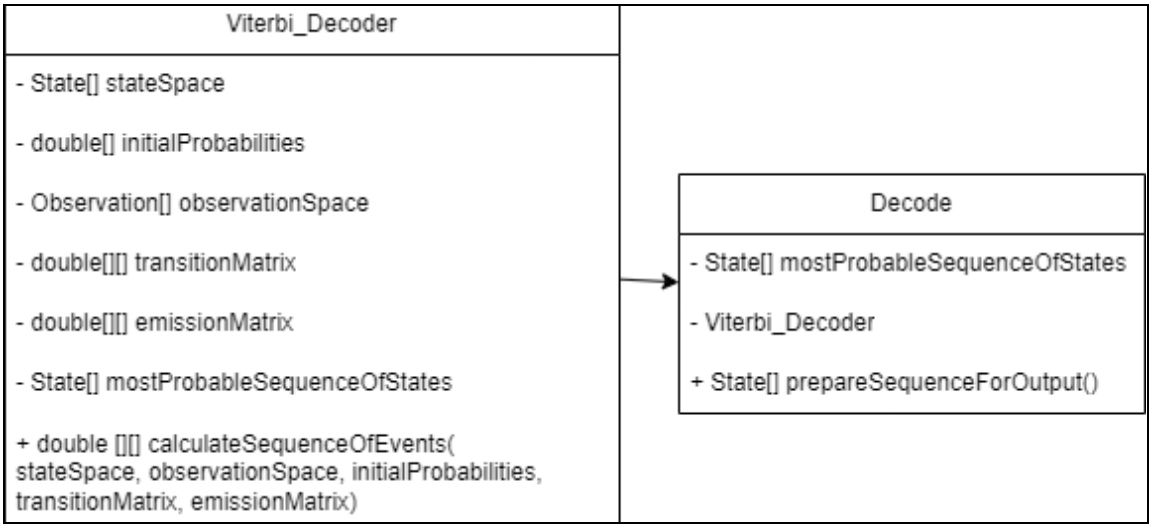

Figure 13: ASR Model Decode Stage Class Diagram V5.3.2

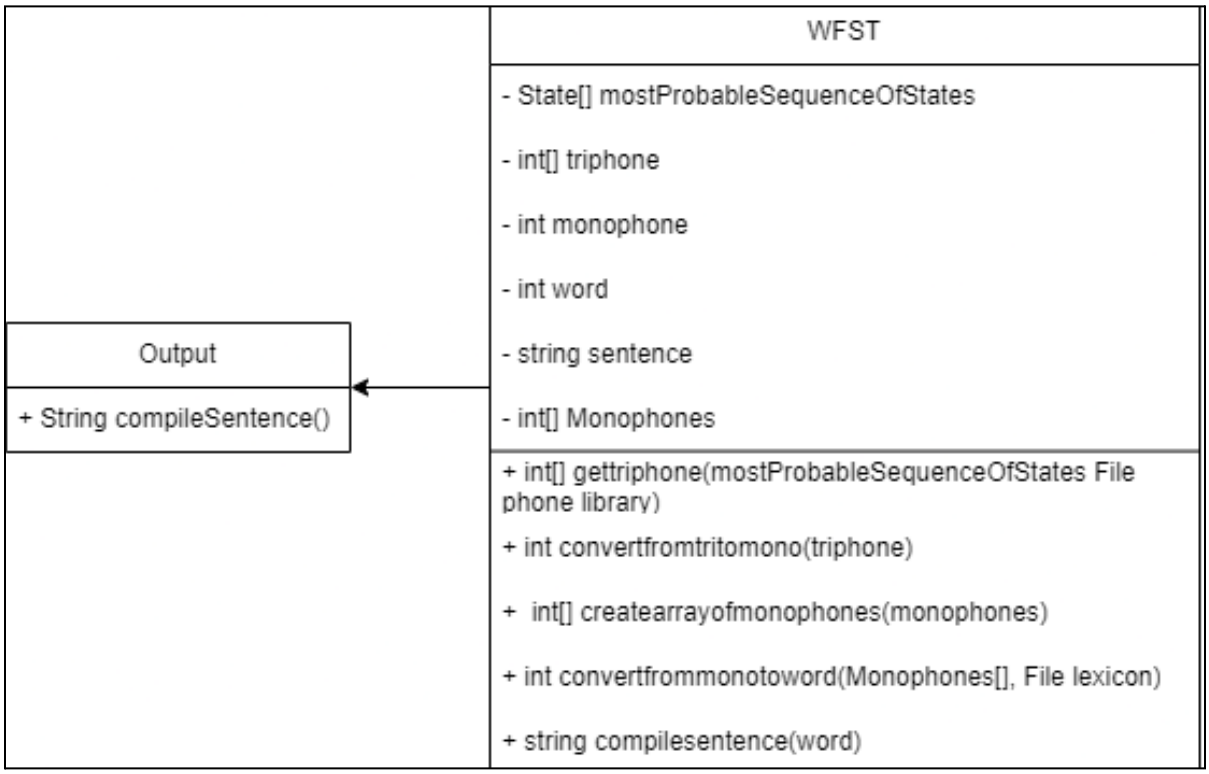

Figure 14: ASR Model Output Stage Class Diagram V5.4.2

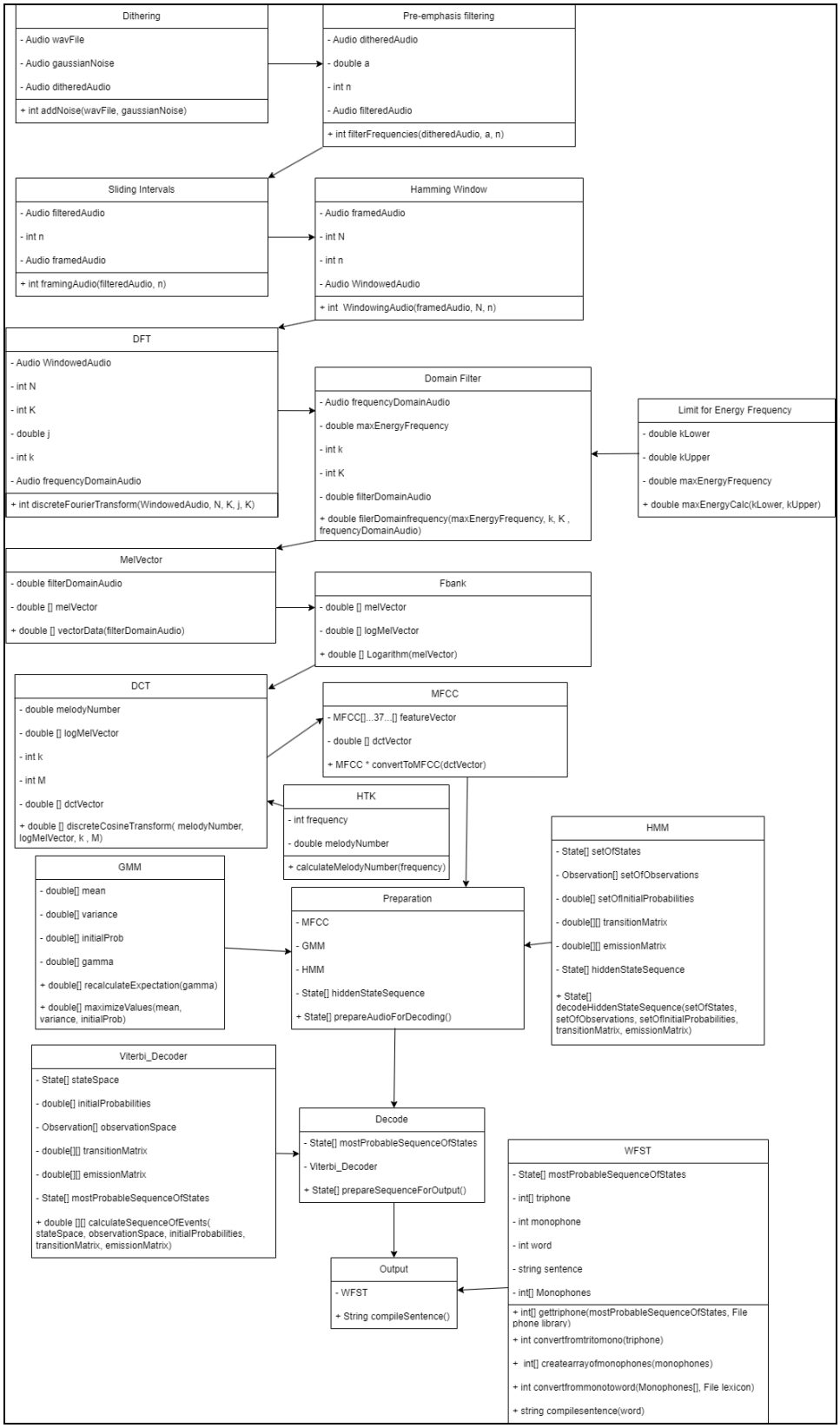

Figure 15: ASR Model Class Diagram V5.1.3

4.1.1 Audio Data Preprocessing

In order for the model to properly transcribe any audio data, the data must first be preprocessed. Preprocessing is performed via the ATCO2 corpus created by Juan Pablo Zuluaga of the Swiss Federal Institute of Technology in Lausanne. ATCO2 provides a script written in Bash that is designed to preprocess the Air Traffic Control Corpus (ATC0) for compatibility with the Kaldi ASR Toolkit. ATC0 is provided by the Linguistic Data Consortium (LDC) and consists of a 30-hour ATC dataset that includes audio files and verified text transcriptions of the communications between pilots and ATC controllers. ATCO2 is designed to be run in the Linux operating system.

The first stage of the preprocessing script initializes the bulletin with the options"-euo pipefail". The "-e" option causes the script to exit immediately if a command returns a non-zero exit status. The "-u" option treats references to undefined variables as errors. Lastly, the "-o pipefail" option causes a pipeline to return a non-zero exit status if any command in the pipeline fails. In short, any errors or failures shall cause the script to terminate early.

The second stage of the preprocessing script initializes data paths used to read from specific directories, and used to create new directories to store output data. These paths include the directories used to store transcripts in ATC0, and directories in ATCO2.

The third stage of the preprocessing script runs an external script that parses the command line for additional options. It allows for users to customize the behavior of the preprocessing script.

The fourth stage of the preprocessing script runs another external script that normalizes all of the text in the transcriptions. Normalization includes conversion to lowercase, removal of punctuation marks, and handling special characters. The same script is used in later stages of preprocessing.

The fifth stage of the preprocessing script checks if all dependencies are installed on the Linux machine. These include the "sph2pipe" program created by LDC, and the SoX utility. Each of these are used for sound file conversion.

The sixth stage of the preprocessing script processes portions of the audio data such as transcriptions and audio files. Various commands such as "find", "grep", "awk", and "sed" are used to perform operations such as generating WAV files from SPHERE files, parsing transcripts, and extracting speaker information. The processed data is stored in appropriate files and directories for further analysis and experimentation.

The seventh stage of the preprocessing script extracts speaker information from the dataset. Information such as speaker IDs are extracted by iterating over the transcript files, identifying speaker roles based upon predefined criteria, and obtaining metadata. This information is then generated onto files and saved in the appropriate directories.

The penultimate stage of the preprocessing script processes transcripts and text from the dataset. It once again normalizes text, converts words into lowercase, and links acronyms together using underscores to help standardize the data.

The last stage of the preprocessing script organizes the processed data and creates training and testing splits. It prepares necessary files such as segment files, utterance-to-speaker mapping files, and text files for each subset; creates dialogues based on callsign identifiers; and filters empty utterances before splitting the data into train and test sets to ensure that the dataset is properly partitioned for training and evaluation purposes.

4.1.2 Audio Data Training

The ASR model must then be trained using the data that was prepared in the steps outlined in the previous section. Audio data preprocessing serves as the first stage of training. The proceeding stages are described below.

The second stage runs a script to extract MFCC features from the data, computes a cepstral mean and variance normalization (CMVN), and creates a data subset containing the 500 shortest utterances to be used for the next stage.

The third stage runs a script to train a monophone model using the aforementioned 500 shortest utterances and the language model (LM) created in the first stage. This stage also runs a script to train the alignments using the monophone model.

The fourth stage runs a script to train the triphone model using an MFCC feature vector containing 13 MFCCs, their derivatives, and their second derivatives, totalling to a 39-dimensional vector.

The fifth stage runs a script to train the triphone model using linear discriminant analysis (LDA) and a maximum likelihood linear transform (MLLT). These techniques are used to improve the feature extraction process, which is a critical step in building accurate ASR models.

The sixth stage runs a script to train the triphone model using speaker adaptation training (SAT).

The seventh stage runs a script to generate single path linear lattices for each utterance using the alignment and the LM. These lattices are generated in their own directory. The script also saves a word pronunciation count onto a text file called “pron_counts_nowb.txt”. The text file contains the counts of pronunciations, which is generated by aligning training data, not from the original text, and is used in the next step. This stage also runs a script to create a dictionary with pronunciation probabilities, which is then used to create a new language model called ConstArpa. Finally, this stage computes the training alignments using the SAT model.

The final stage runs a script to decode multiple datasets using the SAT model and the small trigram LM. Then, the decoded lattices can be rescored using the medium trigram LM or the ConstraArpa LM., resulting in varying word error rates (WERs).

4.2 Internal Communications Detailed Design

The sample ASR model's internal communications start with the user entering the aforementioned command, "python3 main.py file_name.wav", into the Ubuntu terminal. Upon successful execution of the Python3 file, the WAV file (*Figure 17:1.1)* is processed by the ASR model (*Figure 17:2.1)*.

The following three sections outline the data preparation process, which includes the WAV file, the mel-frequency cepstrum (MFC) and mel-frequency cepstral coefficients (MFCCs), Gaussian mixture models (GMMs), and the hidden Markov models (HMMs); the decoding process, which includes the Viterbi decoder; and the output processes, which includes weighted finite state transducers (WFSTs).

4.2.1 Preparation

The ASR model first prepares the WAV file for decoding by formatting the data, converting it from the time-domain to the frequency-domain, and then converting it to a vector.

The first step is dithering the audio with Gausian noise in order to ensure the audio never has a frequency of zero. Next, the audio is filtered to accommodate for higher frequencies using the following formula:

$$x_p[n] = x[n] - ax[n-1]$$

Formula 01: Pre-emphasis filtering

where a is a value between 0.95 and 0.97, n is the sample, and $x[n]$ is the input signal at n. This allows the energy that is typically varied in human speech to be equalized.

The dithered and filtered audio is then partitioned into overlapping 25-millisecond frames, which is the most common frame size. The frames are obtained in 10-millisecond sliding intervals starting at time equals zero milliseconds using the following interval notation:

$$[10n,\ 10n + 25]$$

Formula 02: Sliding intervals

where n is the number of intervals from the beginning.

Windowing is then used to reduce spectrum leakage. The Hamming window, unlike many other types of windows, ensures that the values shall never equal zero. The following formula is used:

$$w_{Hamm}[n] = 0.54 - 0.46\,cos(\frac{2\pi n}{N-1})$$

Formula 03: Hamming Window

where N is the length of the window, and n equals N - 1.

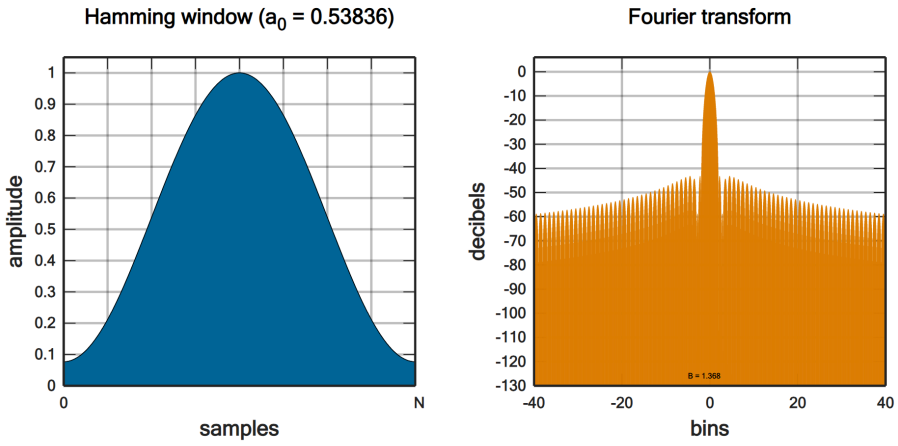

Figure 16: Example of a Hamming Window

The audio data is then converted from the time-domain to the frequency-domain using a fast Fourier transform (FFT) (*Figure 17: 2.1.1*). A FFT is a compilation of algorithms that use the discrete Fourier transform (DFT) formula to account for the edges. As such, the complexity of a FFT is represented by the function $O(K\ log_2(K))$, as opposed to the complexity of the DFT, which is represented by the function $O(K^2)$. The discrete Fourier transform formula is as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{K}kn}$$

Formula 04: DFT

where $x[n]$ is the windowed speech signal, N is the length, $K \geq N$, $j = \sqrt{-1}$, $k = \frac{\omega_k K}{2\pi}$.

The Mel filter bank is used to convert from frequency to melody numbers using the following formula:

$$m = 2595 \cdot log_{10}(1 + \frac{f}{700})$$

Formula 05: HTK formula

where m is the melody number and f is the frequency in hertz.

Frequency domain is then filtered using the following formulae:

$$F_m[k] = \begin{cases} \frac{k-k_{m-}}{k_m-k_{m-}}, & \text{for } k_{m-} \le k \le k_m, \\ \frac{k_{m+}-k}{k_{m+}-k_m}, & \text{for } k_m \le k \le k_{m+}, \\ 0, & \text{otherwise.} \end{cases}$$

Formula 06: Limit for energy frequency

where $F_m[k]$ is the maximum energy frequency, k_{m-} is the lower limit and k_{m+} is the upper limit.
In Formula 7,

$$a_m = \sum_{k=0}^{K/2+1} F_m[k]|X[k]|^2$$

Formula 07: Frequency domain filter

where $F_m[x]$ is from Formula 6, and $X[k]$ is from the FFT represented by Formula 4.

$$v_{mel} = [a_0, a_1, \ldots, a_{M-1}]$$

Formula 08: Vector for Mel filter

where v_{mel} is the vector for the mel filter.

Taking the logarithm of the Mel vector creates the Fbank, which compresses the range of the signal and simplifies the next steps.

$$v_{Fbank} = log\, v_{mel} = [v_0, v_1, \ldots, v_{M-1}]$$

Formula 09: Log of Vector

where v_{mel} is from Formula 8.

The Discrete Cosine Transform (DCT) makes a sequence even or symmetric with respect to the origin using the following formula:

$$c[k] = \sum_{m=0}^{M-1} v[m]\, cos\, (\pi\, (m + 0.5)\, k/M)$$

Formula 10: Discrete Cosine Transform

where $v[m]$ is from Formula 9, m is from Formula 5, $k = M - 1$, and M is the number of dimensions in the vector from Formula 9.

The mel-frequency cepstrum (MFC) reduces the dimensions from the DCT to twelve mel-frequency cepstral coefficients (MFCCs). A thirteenth MFCC is used to represent the short time energy of the speech signal. The delta (Δ) and the delta-delta (ΔΔ) of the 13 MFCCs are calculated

$$c_{\Delta}[k,l] = \frac{\sum_{n=1}^{N} n(\, c[\, k, l+m\,] - c\,[\, k, l-m])}{2\sum_{n=1}^{N} n^2}$$

Formula 11: Derivative of MFCC

A Gaussian mixture model (*Figure 17: 2.1.3*) shall find the expectation maximization of the MFCC feature vector in reference to phoneme Gaussian distribution by initializing μ_k -- the mean of the component; Σ_k -- variance of the component; and π_k -- the initial probability. Which it uses Formula 13 to calculate the $\gamma(Z_{nk})$ probability that the observation is int class k given the observation x_a. It shall continue to recompute μ_k , Σ_k , π_k , and $\gamma(Z_{nk})$ until it has been maximized.

$$Z_{nk} = \begin{cases} 1, \ if \ x_a \ in \ class \ k \\ 0, \ if \ not \end{cases}$$

Formula 12: Z_{nk}

$$N(\mu_k\,,\,\Sigma_k) = \sum_{k=1}^{K} \pi_k \, N(x \mid \mu_k\,,\,\Sigma_k)$$

Formula 13: Gaussian Mixture Model

A hidden Markov model (*Figure 18: 2.1.4*) shall calculate the hidden state sequence, the transition matrix - the probability of going to the next state, the emission matrix - the probability of the observation being generated from the i^{th} state. The HMM matches the waveform of the MFCC vector to the states for the phonemes, which is three per phoneme; it shall remain in a state until the waveform matches the state waveform at that specific point.

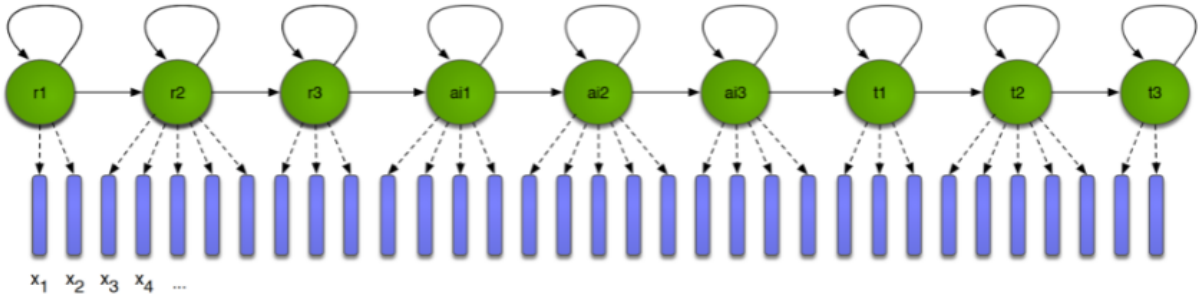

Figure 17: State comparison to the waveform points

4.2.2 Decoding

The decoding operations shall be performed by a Viterbi decoder. The Viterbi decoder (*Figure 17: 2.1.5*) shall receive the initial probability distribution (V_j) -- the probability that it shall start in that state; the sequence of observable events (B_j)- the order of observations based on time; the emission matrix (b_j) -- the probability of the observation being generated from any given state *i*; the transition matrix (a_{ij}) – the probability of going to the next state; and the State Space (x_i) -- the set of HMM state changes. It returns the probability of the sequence of events.

Initialization:

The Viterbi decoder initializes the start state and the probability of that state occurring at time equals zero. In this instance, the probability equals 1 because the probability of the current state being the start state is guaranteed. It is also assumed that the path back would be to the initial state (State 01).

$$For\ j\ =\ 1, V_j(0)\ =\ 1$$
$$Otherwise,\ V_j(0)\ =\ 0$$

Formula 14: Initialization for Viterbi decoder for the probability of the start state

$$B_j(0)\ =\ 1\ ,\ \forall j$$

Formula 15: Initialization for Viterbi decoder for the path back to the previous state

Recursion:

The Viterbi decoder then loops back and forth through the possible sequences to find the most probable current state and what the previous state would be.

$$V_j(t)\ =\ max_{i=1}^{J} V_i(t-1) a_{ij} b_j(x_i)$$

Formula 16: Recalculation of current probability of each state at time t

$$B_j(t)\ =\ arg\,max_{i=1}^{J} V_i(t-1) a_{ij} b_j(x_i)$$

Formula 17: Recalculation of the path back at time t

Termination:

The goal is to maximize the probability of each of the states in a sequence of states. When maximization is achieved, the decoding process is complete.

$$P\ *=\ V_E\ =\ max_{i=1}^{J} V_i(T) a_{iE}$$

$$s\ *_T\ =\ B_E\ =\ arg\,max_{i=1}^{J} V_i(T) a_{iE}$$

Formula 18: Termination of the probability of the sequence of events

4.2.3 Output

The WFSTs include a HMM, a context-dependency model (CDM), a language model (LM), and a lexicon.

The Viterbi decoder (*Figure 18: 2.1.5*) shall pass the chain of monophones onto the hidden Markov Model (*Figure 19: 2.1.6.1*), which shall be used along with the triphone Database (*Figure 18: 4.1*) to find the triphone. Which shall be passed to the context dependence (*Figure 19: 2.1.6.2*), which shall convert the triphones into monophones using the data from the phone database (*Figure 18: 5.1*). The monophones are then passed to the lexicon (*Figure 19: 2.1.6.3*), which uses the database of words (*Figure 19: 6.1*) to convert the monophones into words. The language model (*Figure 19: 2.1.6.4*), takes the two previous words' index numbers and predicts the next word. Finally, the sentences are written into "out.txt", which is returned to the user.

The Kaldi ASR Toolkit shall train the ASR models using the training data labels so the model can recognize sounds and speech components. The model shall continually iterate until the accuracy and improvement plateau. Upon no noticeable improvement the training stops and the trained model is released.

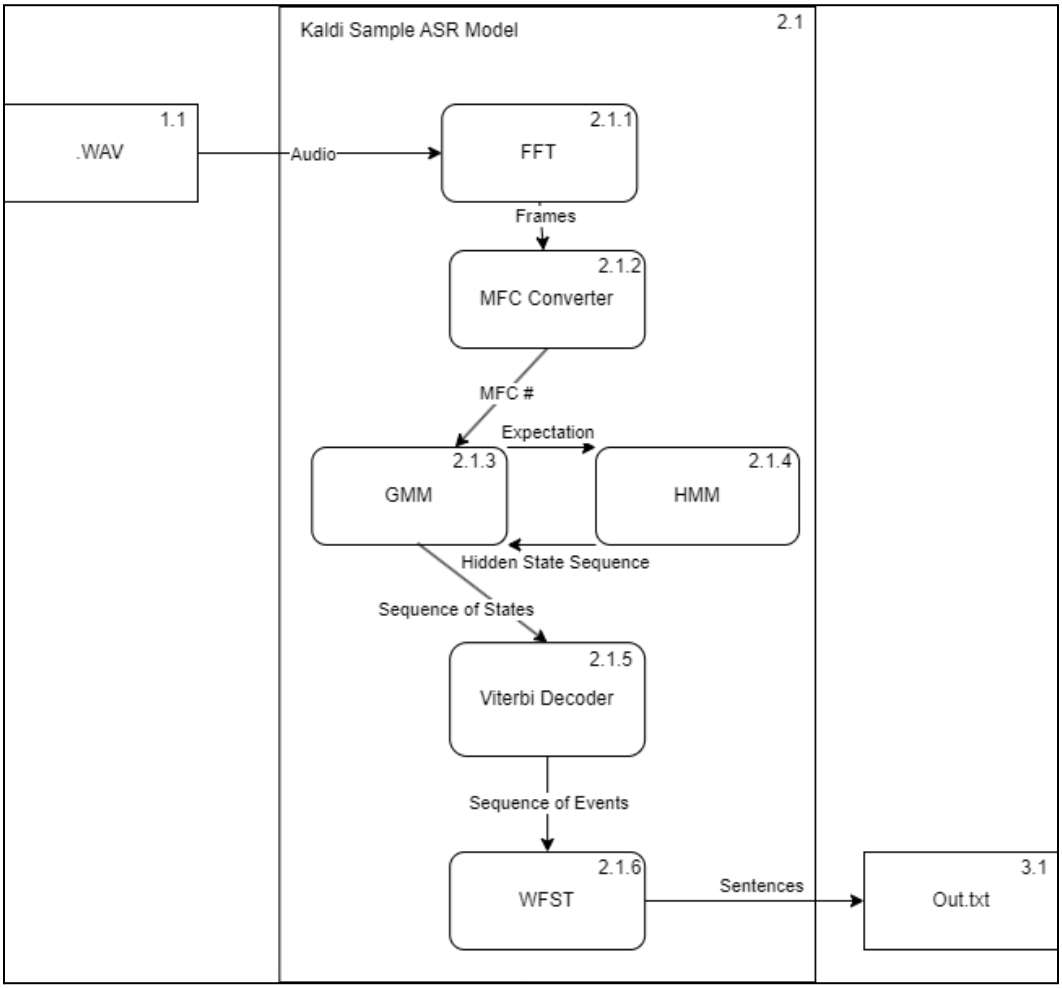

Figure 18: Kaldi ASR Toolkit Sample ASR Model Level 1 Data Flow Diagram V5.1.1

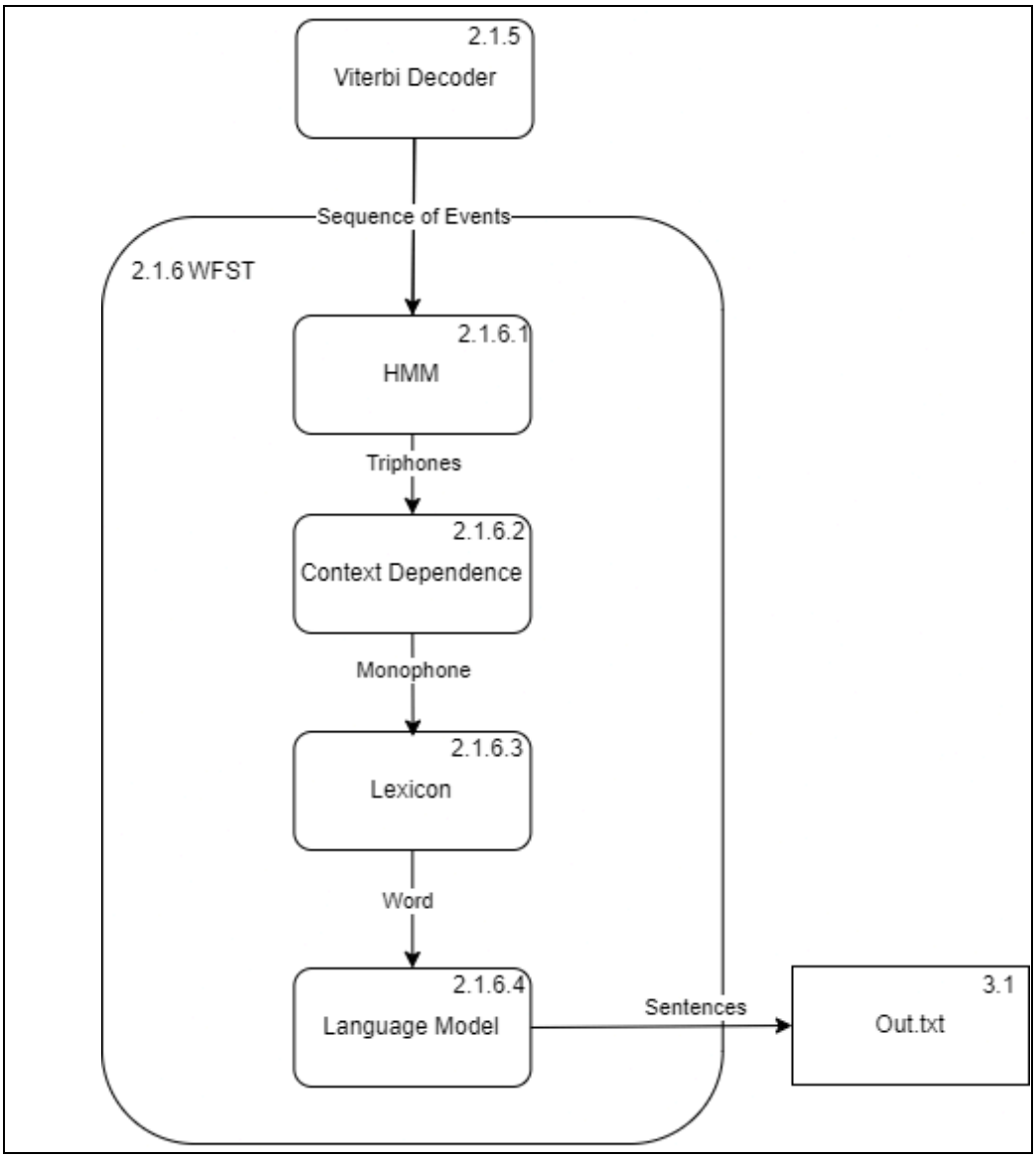

Figure 19: Kaldi ASR Toolkit Sample ASR Model Level 2 Data Flow Diagram V5.2.2

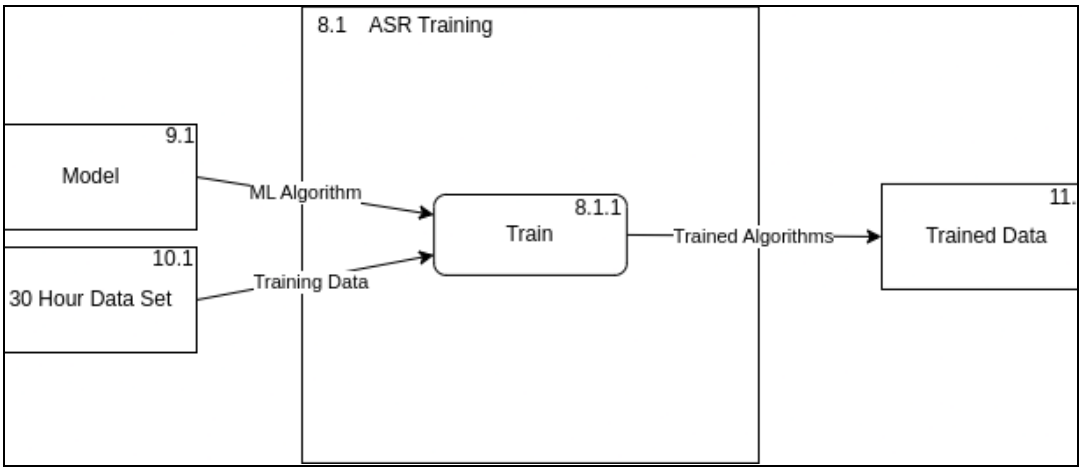

Figure 20: Kaldi ASR Toolkit Training Level 1 Data Flow Diagram V4.3.1

5 EXTERNAL INTERFACES

The Kaldi ASR Team is solely focused on developing an acoustic ASR model. Once the model is completed, it shall be integrated in the RTube web application. Outside the purview of the Kaldi ASR Team is the development of the web application itself -- this task is delegated to the NeMo Team.

It should be noted the following subsections are in reference to the sample ASR model provided by the Kaldi ASR Toolkit. Once the Kaldi ASR Team develops its own acoustic ASR model, the following subsections shall be updated in reference to it.

5.1 Interface Architecture

The RTube web application in development by the RTube NeMo Team shall serve as a medium for a future acoustic ASR model developed by the Kaldi ASR Team. The web application shall allow the hypothetical acoustic ASR model to run within it via the Python Flask API. It should be noted, however, that the development, let alone the implementation, of the aforementioned interface architecture is expected to span over a year into the future.

5.2 Interface Detailed Design

The Kaldi ASR Toolkit uses a command-line interface (CLI) to accept input and generate output. The two commands of importance are “ffmpeg -i filename.filetype filename.wav” (*Figure 21*), and “python3 main.py filename.wav” (*Figure 22*), seen previously in Section 3.1 of this document. These commands are used to convert audio files into WAV files via the FFmpeg Linux utility, and to call a Python3 script that executes the sample ASR model, respectively.

The first command is composed of three arguments and a flag. The argument, “ffmpeg”, calls the execution of the FFmpeg utility. The “-i” flag denotes that the next argument, “filename.filetype”, is the input audio file. Lastly, the result of the conversion is specified by the argument, “filename.wav”, which shall ideally have the same filename as the input file.

The second command is also composed of three arguments. As mentioned previously in Section 3.1 of this document, the argument, “python3”, calls the execution of a Python3 file. The file in question, “main.py, is a script that initiates the execution of the sample ASR model. Lastly, the argument, “filename.wav”, calls the WAV file that shall be transcribed by the sample ASR model. The file shall be located in the same directory as the script.

Figure 21: Execution of FFmpeg utility (with code execution snippet); conversion of FLAC to WAV

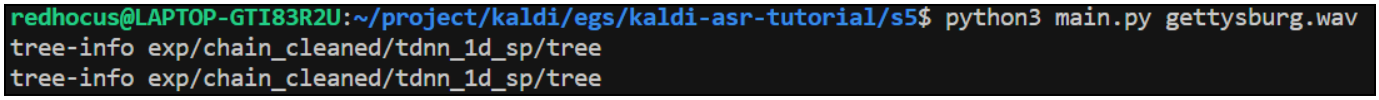

Figure 22: Sample ASR Model Successful Input (with code execution snippet)

6 SYSTEM INTEGRITY CONTROLS

All training data used in the development of the acoustic ASR model shall be password protected. Only people involved in the development of the model shall have access to the training data (e.g., product owner, developers, etc.).