System Design Document

For

RTube Kaldi Research Team

Team members:
Tabitha O'Malley, Milan Haruyama, David Serfaty, Tahmina Tisha, Adam Gallub

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

TABLE OF CONTENTS

SYSTEM DESIGN DOCUMENT

1 INTRODUCTION

1.1 Purpose and Scope

As an aircraft pilot, learning to communicate with Air Traffic Control (ATC) is a daunting task. Despite being designed to mitigate miscommunication, aviation phraseology is highly intricate and requires hundreds of hours of training to learn its idiosyncrasies. While there exist a few resources (such as the website LiveATC) for student pilots to study aviation phraseology, these resources do little to accommodate the major learning curve present, especially since they do not provide live transcriptions of 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, as well as providing a live interface for users to track the flight paths of aircraft. The live speech-to-text transcription is performed using an automatic speech recognition (ASR) model developed using the Kaldi ASR Toolkit. 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.

1.2 Project Executive Summary

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

1.2.1 System Overview

The sample ASR model transcribes audio into text. The ASR model utilizes phone, triphone, and word databases to convert the received audio into words. The ASR model then uses a predictive model to build the sentence structure for the converted words. After the audio has been fully transcribed, the ASR model returns the text to the user.

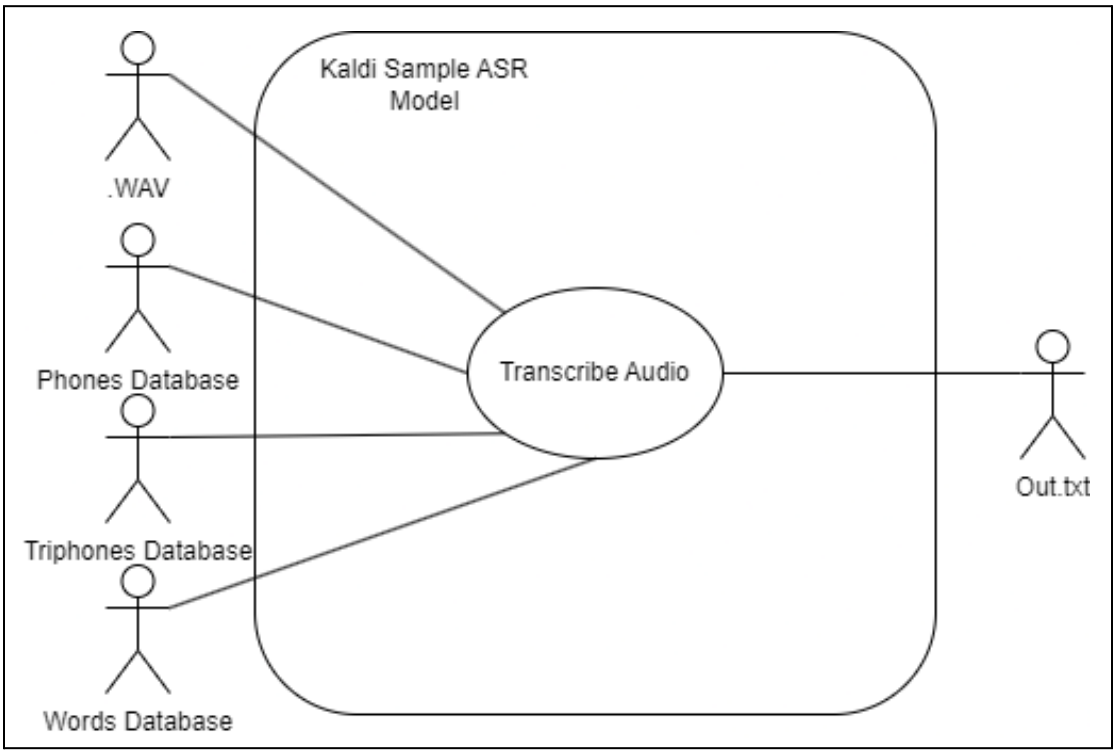

Figure 1: Sample ASR Model Use Case Diagram V2.1.1

1.2.2 Design Constraints

There are two constraints that limit the scope of any development done by the RTube Kaldi Research Team. Those are the complexity of speech recognition and the scale of the Kaldi ASR Toolkit. There are many challenges that speech recognition brings to the table, variability in speech, tone and pitch, accents, and even the speed at which the speaker talks. Each one of these brings a unique challenge. When it is speech recognition for an ATC application these challenges increase tenfold and even more challenges are added including the idiosyncrasies of ATC communications and static from the radios.

Due to the complexity of speech recognition, the magnitude of the Kaldi ASR Toolkit is enormous. The average desktop computer does not have the capability to run and sometimes even download the toolkit. The scale of Kaldi also brings the challenge of understanding. This is because the developers of Kaldi did not put a lot of time into writing documentation that could be understood. This means that as the research team goes through the Kaldi code it is a steep learning curve and requires a great deal of teamwork and documentation to learn how the toolkit works.

1.2.3 Future Contingencies

The Kaldi Research Team is hindered by a lack of coordination and connection with the NeMo team, and outdated hardware that is incapable of running the Kaldi ASR Toolkit.

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 sample ASR model included in the Kaldi ASR Toolkit. 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. Lastly, the System Integrity Control section shall cover any sensitive data that could threaten the integrity of the system if modified.

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

Kaldi Research Team Software Requirements Specification. Kaldi Research Team. 19 November 2023. Kaldi Research 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.

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

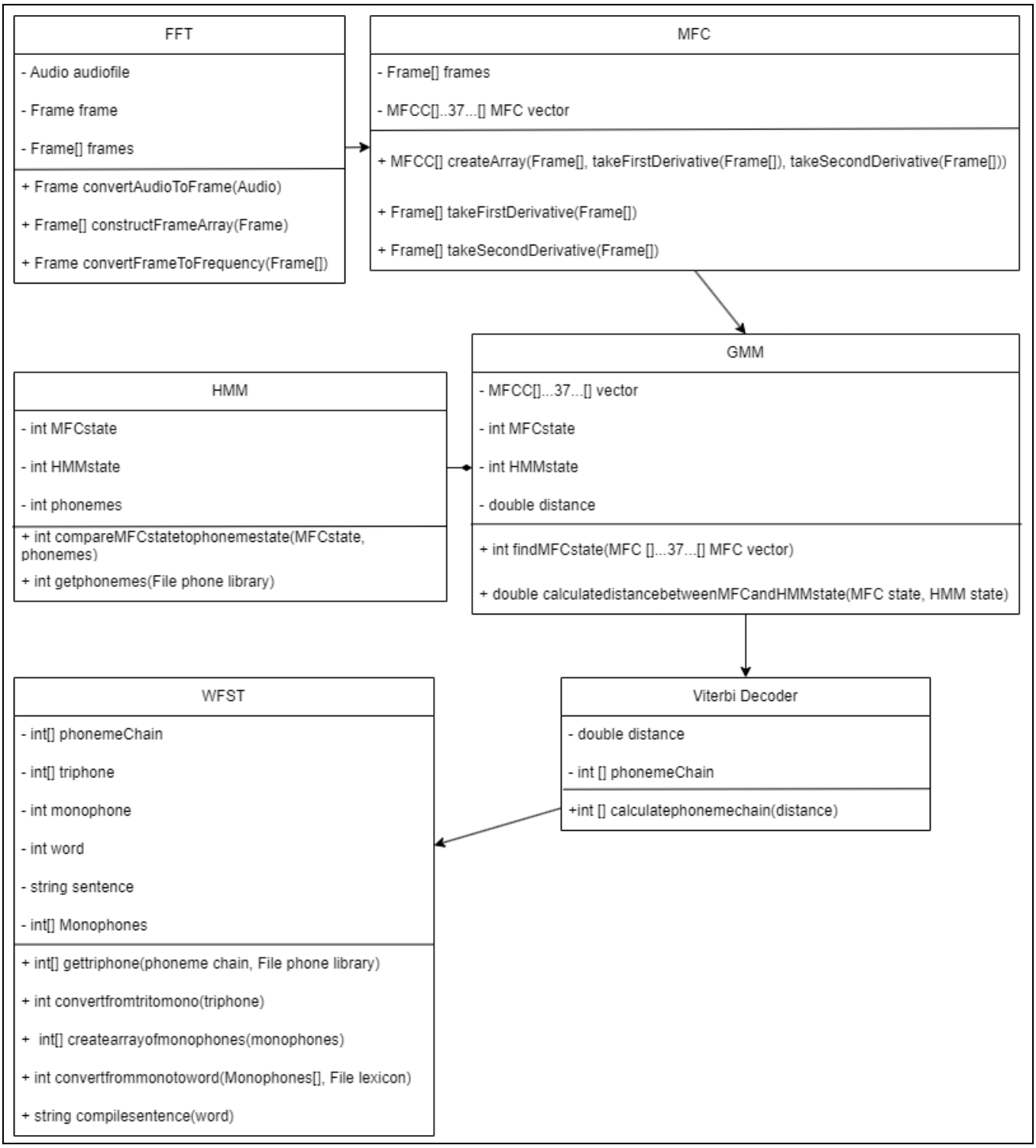

Figure 2: Sample ASR Model Class Diagram V3.2

Fast Fourier Transform (FFT): responsible for converting audio frames from the time-domain to the frequency-domain.

- Variables: Audio audioFile, Frame frame, Frame[] frames
- Methods: convertAudioToFrame(Audio), constructFrameArray(Frame), convertFrameToFrequency(Frame [])
- Inputs: WAV file
- Outputs: frequency-domain frames

Mel-Frequency Cepstrum (MFC): converts frequency-domain frames into a 39-dimensional MFCC feature vector represented as a 39-dimensional array.

- Variables: Frame[] frames, MFCC[]... 37…[]
- Methods: create array (Frame [], takeFirstDerivative(Frame []), takeSecondDervivative(Frame[])), takeFirstDerivative(Frame []), takeSecondDervivative(Frame[])
- Inputs: frequency-domain frames
- Outputs: 39-dimensional MFCC feature vector

Gaussian Mixture Model (GMM): finds the distance between the MFCC feature vector and the phoneme states.

- Variables: MFCC[]...37…[] MFC, int MFCstate, int HMMstate, double distance
- Methods: findMFCstate(MFCC[]...37…[] MFC), calculatedistancebetweenMFCandHMMstate(MFCstate, HMMstate)
- Inputs: MFC feature vector
- Outputs: Distance

Hidden Markov Model (HMM): finds the HMM state closest to the MFC state.

- Variables: double distance, int HMMstate, in phonemes
- Methods: compareMFCstatetophonemestate(MFCstate, phonemes), getphonemes(File phone library)
- Inputs: MFC state
- Outputs: HMM state

Viterbi Decoder: finds the phoneme chain based on the previously calculated distance.

- Variables: double distance, int HMMstate, in phonemes
- Methods: compareMFCstatetophonemestate(MFCstate, phonemes), getphonemes(File phone library)
- Inputs: MFC state
- Outputs: Phoneme chain

WFST (Weighted Finite State Transducer): responsible for converting the phoneme chain into sentences.

- Variables: int[] phonemeChain, int[] triphone, int monophone, int word, string sentence, int[] Monophones
- Methods: gettriphone(phonemeChain, File phone library), convertfromtritomono(triphone),createarrayof monophones(monophones), convertfrommonotoword(Monophones[], File lexicon), compilesentence(word)
- Inputs: Phoneme chain
- Outputs: Sentences

2.2 Internal Communications Architecture

The Kaldi ASR Toolkit provides users with a sample ASR model that receives audio files as input and outputs text transcriptions of the audio. The ASR model exclusively accepts WAV files and exclusively outputs text files. The 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 if they are found in the lexicon. If not, they shall be written in lowercase and shall be appended to the non-lexicon library. More thorough analyses of the sample ASR model are provided in Section 4 of this document.

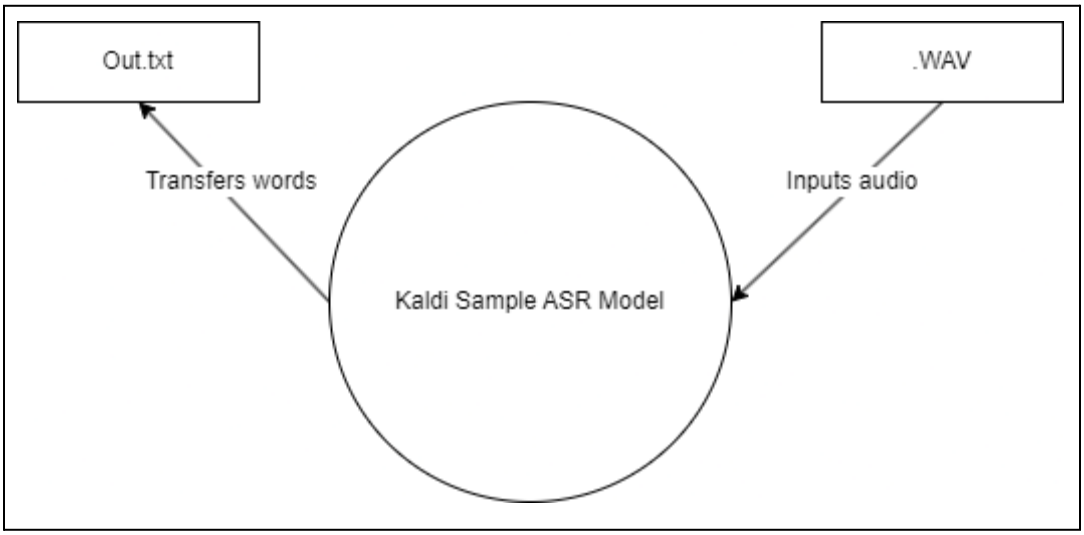

Figure 3: Sample ASR Model Level 0 Data Flow Diagram V1.2

3 HUMAN-MACHINE INTERFACE

3.1 Inputs

Input shall be given as a terminal command in the format, “python3 main.py filename.wav” (*Figure 4*). 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 4*). 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 5, 6, 7*), 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 5*). Calling multiple files of any type shall cause the script to throw an exception and print, “Too many arguments provided. Aborting” (*Figure 6*). 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 7*).

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 9*). 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 9*.

The program shall print out the guessed value of non-lexicon words in the terminal. For instance, Gettysburg (*Figure 9*) 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 4: 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 5: 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 6: 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 7: Sample ASR Model Unsuccessful Input; no file in directory exception

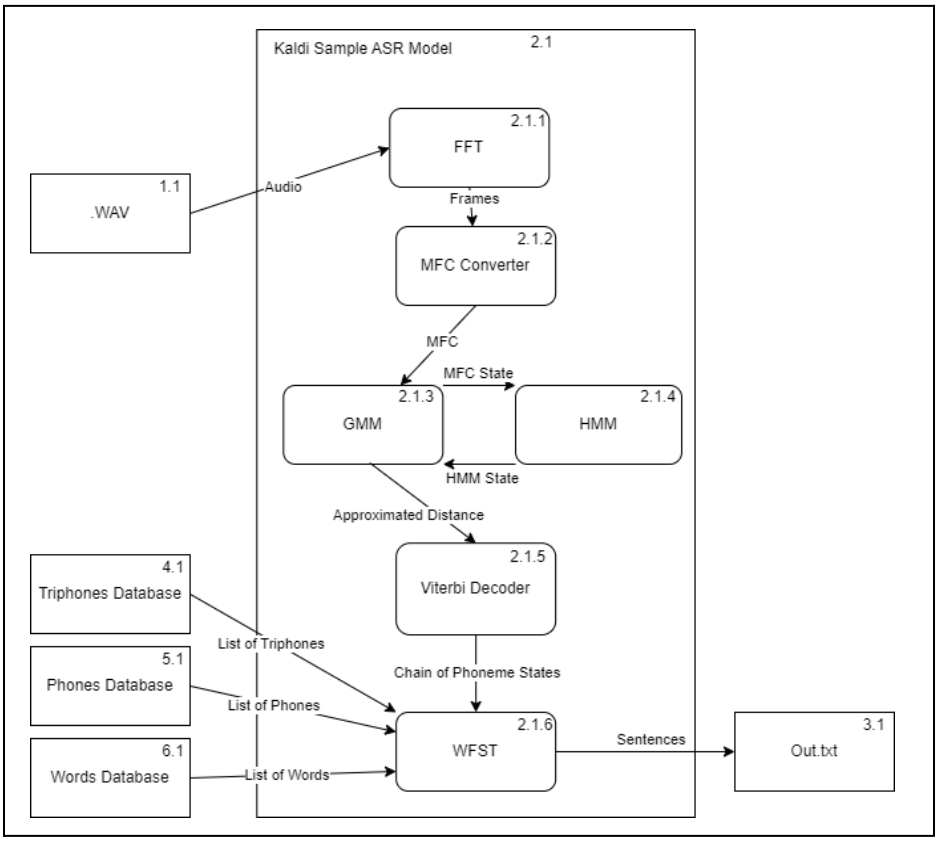

Figure 8: Sample ASR Model Level 1 Data Flow Diagram V3.1.3

```
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 9: Sample ASR Model Output

4 DETAILED DESIGN

4.1 Software Detailed Design

This section provides an overview of the ASR model in the form of the use case diagrams below. *Figure 9* illustrates how users and other actors shall interact with the ASR model. *Figure 10* illustrates how an ASR model is trained.

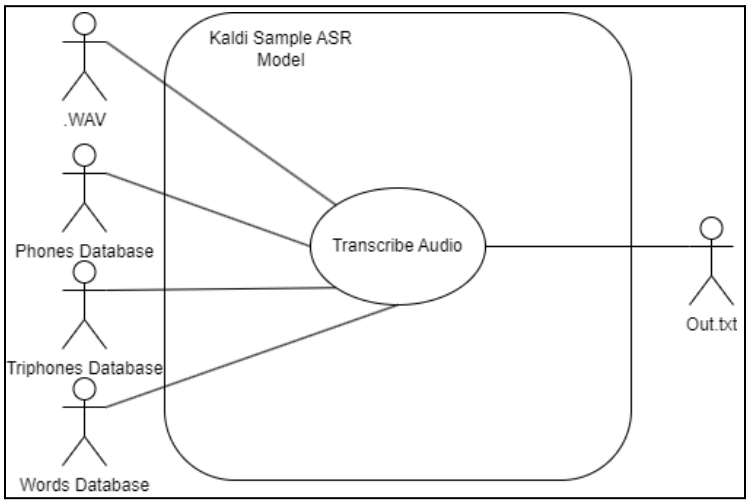

Figure 10: Kaldi ASR Toolkit Sample ASR Model Use Case Diagram V2.1.1

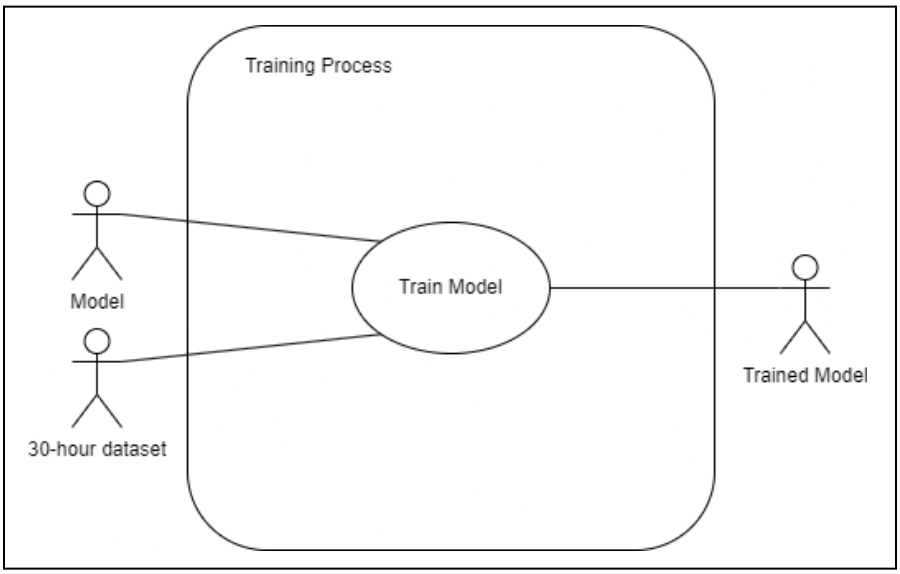

Figure 11: Kaldi ASR Model Training Use Case Diagram V2.2.1

Use Cases:

Use Case ID: UC1

Use Case Name: Transcribe Audio

Goal: The user inputs a WAVand the ASR model shall return a text file containing the transcribed audio called “out.txt”.

Actors: WAV File, Phones Database, Triphones Database, Words Database, “out.txt”

Precondition: Model is properly installed and trained

Post-Condition: “out.txt” is stored in /kaldi/egs/mini_librispeech/s5/

Trigger Condition: The user inputs a WAV file

Main Success Scenario:

Step	System Reaction	Step	Actor Action
2	Partitions audio data into 25-millisecond frames in 10-millisecond sliding intervals, allowing for three frames to overlap	1	Input WAV file
3	Converts frames from the time-domain to the frequency-domain		
4	Generates a 39-dimensional MFCC feature vector and populates it with the initial frequency- domain data, the first-order derivative of the data, and the second-order derivative of the data to show changes in the frequency-domain.		
5	Populates the MFCC feature vector with the initial frequency-domain data, the first-order derivative of the data, and the second-order derivative of the data to show changes in the frequency-domain.		
6	Compares MFCC feature vector data with HMM state		
7	Calculates the distance to the most probable phoneme based on the MFCC feature vector and the data sent by the HMM		
8	Constructs a chain of phoneme states based on the distance calculated by the GMM		
10	Builds triphones using the phoneme chains from the Viterbi Decoder	9	Access Triphone database
12	Converts triphones to monophones based on context-dependency	11	Access Phone database
14	Builds words using the monophones	13	Access Word database
15	Uses the previous two words to predict the following word		

16	Uses the language model to assemble the words into sentences		
17	Transfers sentences to a text file	18	Outputs “out.txt”

Exceptions (Alternative Scenarios of Failure):

1. User inputs non WAVaudio file
2. User does not input anything
3. Transcription failure

Alternate Scenarios:

ALT 1: Step 1: The system does not accept any other audio files than WAV.
Step 1.1: The system throws an exception.
Step 1.2: Return to step 1.

ALT 2: Step 0: The system shall not execute if not WAVis imputed.
Step 0.1: Continue to wait

ALT 3: Step 2-16: The system fails to transcribe audio.
Step 2-16.1: The system displays an exception for the failed step.
Step 2-16.2: Return to the failed step.

Use Case ID: UC2

Use Case Name: Train Model

Goal: The ASR model shall be trained using the 30-hour training data

Actors: Model, Training Data

Pre: ASR model exists, training data exists

Post: ASR model is trained

Main Success Scenario:

Step	System Reaction	Step	Actor Action
		1	Input model
3	The model shall train itself to recognize speech and sound through changes in frequency by using the training data	2	Input training data
4	Attempts to match the labels and continues training until improvement plateaus		
5	Return trained model		

Exceptions (Alternative Scenarios of Failure):

No possible exceptions due to assumption of adequate training data as per customer’s specifications.

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 9:1.1)* is processed by the ASR model (*Figure 9:2.1)*.

The following six paragraphs outline the data preparation process, which includes: the WAV file; Fast Fourier Transforms (FFTs); Mel-Frequency Cepstrum (MFC) and Mel-Frequency Cepstral Coefficients (MFCCs); Gaussian Mixture Models (GMMs); Hidden Markov Models (HMMs); Viterbi Decoder; and Weighted Finite State Transducers (WFSTs).

The sample ASR model first inputs the WAV files into a FFT (*Figure 9: 2.1.1*) . The FFT partitions the file into overlapping 25-millisecond frames. The frames are obtained in 10-millisecond sliding intervals starting at time equals zero milliseconds, represented by the formula $[10n,\ 10n + 25]$. The audio data is then converted from the time-domain to the frequency-domain.

The MFC Converter (*Figure 9: 2.1.2*) creates a 39-dimensional MFCC feature vector used to represent a MFC. Each dimension is indexed in 10 millisecond increments (e.g., the first index represents time equals 10 milliseconds). Then, the Converter partitions the aforementioned frames into twelve segments, and appends an additional "common" segment that represents the total power of the twelve segments. The MFC Converter then converts each of the thirteen segments into MFCCs. The first- and second-order derivatives of each MFCC are derived afterwards. The Converter then populates the MFCC feature vector with the aforementioned data, using the first set of thirteen dimensions (i.e., indices 0 through 12) to store the original thirteen MFCC; the second set of thirteen dimensions (i.e., indices 13 through 25) store the first-order derivatives of the original thirteen MFCCS; and lastly, the third set of thirteen dimensions (i.e., indices 26 through 38) store the second-order derivatives of the original thirteen MFCCs.

A Gaussian Mixture Model (*Figure 9: 2.1.3*) shall represent the MFCC feature vector of each frame. A Hidden Markov Model (*Figure 9: 2.1.4*) shall represent the relationship between the MFCC vector and the phonemes due to the phonemes evolving in a chain order and being unobservable.

The decoding operations shall be performed by a Viterbi Decoder and a set of four Weighted Finite State Transducers (WFSTs) on a high-level. The Viterbi Decoder (*Figure 9: 2.1.5*) shall take in the distance between the MFCC vector and the GMM and return a chain of phonemes states. The WFSTs (*Figure 9: 2.1.6*) shall convert the chain of phonemes into text.

The WFSTs include a HMM, a Context-Dependency Model (CDM), a Language Model (LM), and a Lexicon. They are described below.

The Viterbi Decoder (*Figure 10: 2.1.5*) shall give the Hidden Markov Model (*Figure 10: 2.1.6.1*) the chain of monophones which shall be used along with the Triphone Database (*Figure 10: 4.1*) to find the triphone. Which shall be passed to the Context Dependence (*Figure 10: 2.1.6.2*), which shall convert the triphones into monophones using the data from the Phone Database (*Figure 10: 5.1*). The monophones are then passed to the Lexicon (*Figure 10: 2.1.6.3*), which uses the database of words (*Figure 10: 6.1*) to convert the monophones into words. The Language Model (*Figure 10: 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.

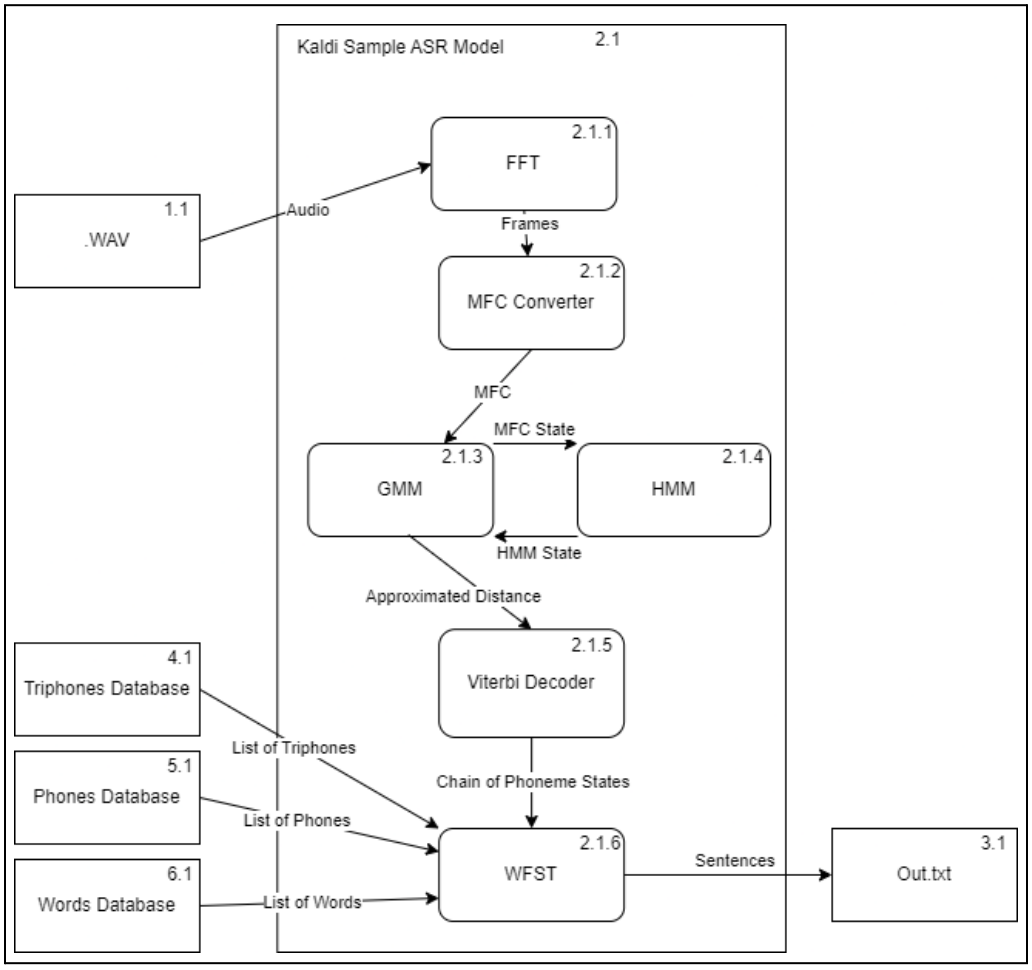

Figure 12: Kaldi ASR Toolkit Sample ASR Model Level 1 Data Flow Diagram V3.1.3

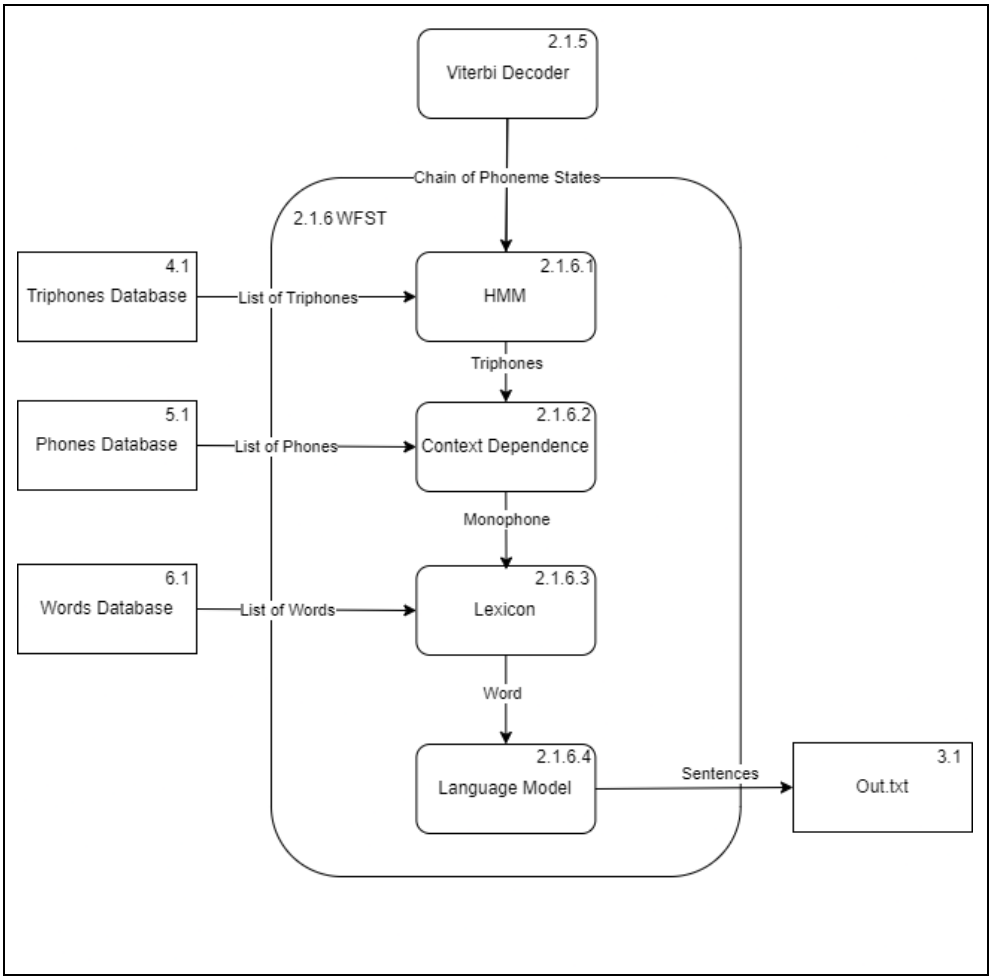

Figure 13: Kaldi ASR Toolkit Sample ASR Model Level 2 Data Flow Diagram V3.2.2

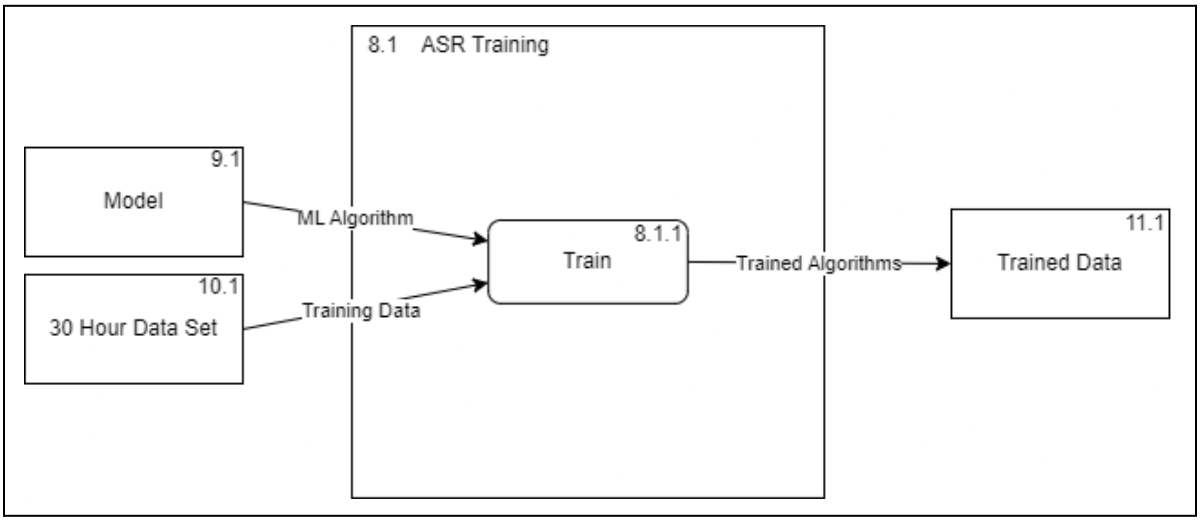

Figure 14: Kaldi ASR Toolkit Sample ASR Model Level 2 Data Flow Diagram V3.3.1

5 EXTERNAL INTERFACES

Outside the purview of the Kaldi Research Team is the development of the RTube web application. Once completed, the RTube web application shall integrate the data gathered by the Kaldi Research Team at the end of development.

5.1 Interface Architecture

The RTube web application in development by the RTube NeMo Team shall serve as a medium for a future ASR model developed by the RTube Kaldi Research Team. The web application shall allow the hypothetical 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 15*), and “python3 main.py filename.wav” (*Figure 16*), 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.

```
redhocus@LAPTOP-GTI83R2U:~/project/kaldi/egs/kaldi-asr-tutorial/s5$ ffmpeg -i gettysburg.flac gettysburg.wav
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
```

Figure 15: Execution of FFmpeg utility (with code execution snippet); conversion of FLAC to WAV

```
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 16: Sample ASR Model Successful Input (with code execution snippet)