System Design Document

For

The Kaldi ASR 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	V2.17

		Updating models (context, use case, DFD)	
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 3	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

TABLE OF CONTENTS

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.

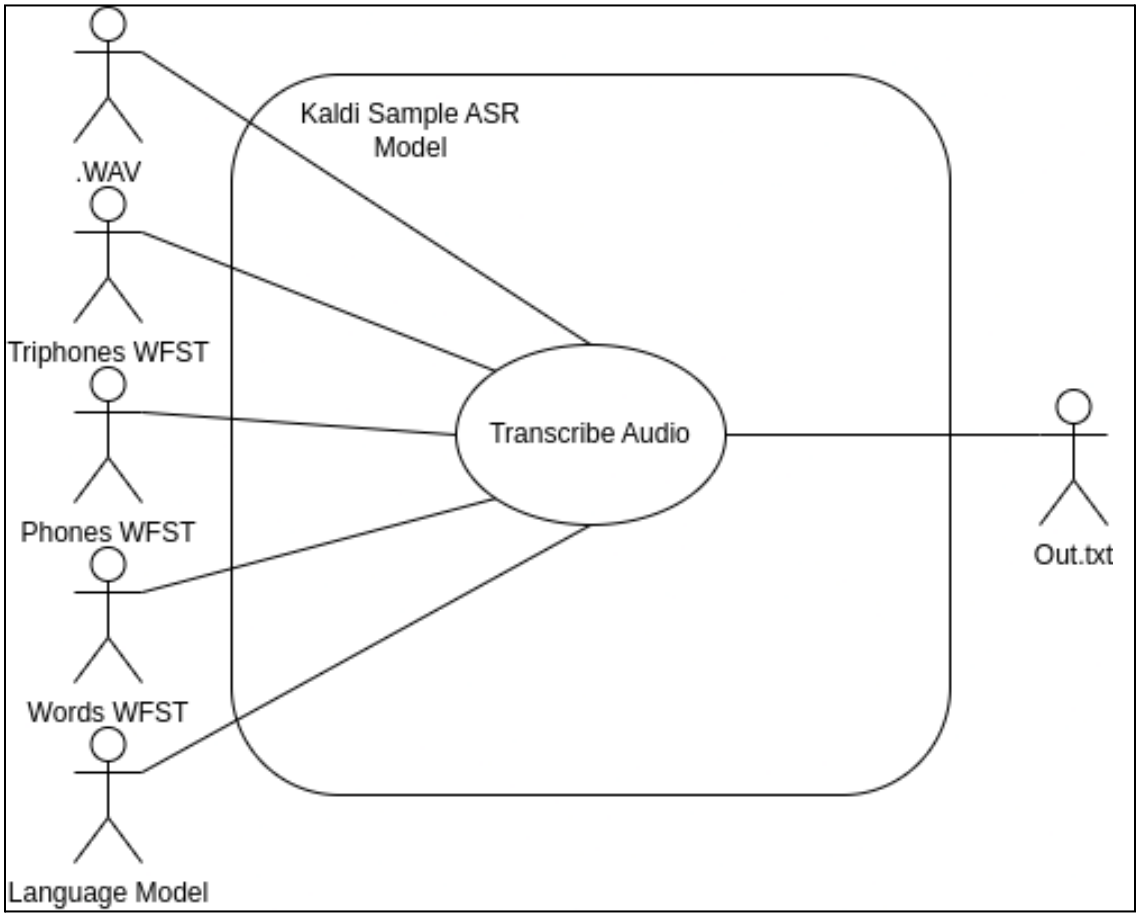

Figure 01: Sample ASR Model Use Case Diagram V4.1.1

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, and the complexity of speech recognition in general.

Some of the difficulties of working with Kaldi ASR Toolkit include its abhorrently large 12.5 GB installation package, its extremely high-level documentation, and its lack of a graphical user interface (GUI).

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 the 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.

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 master 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 Requirements Specification. 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.

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

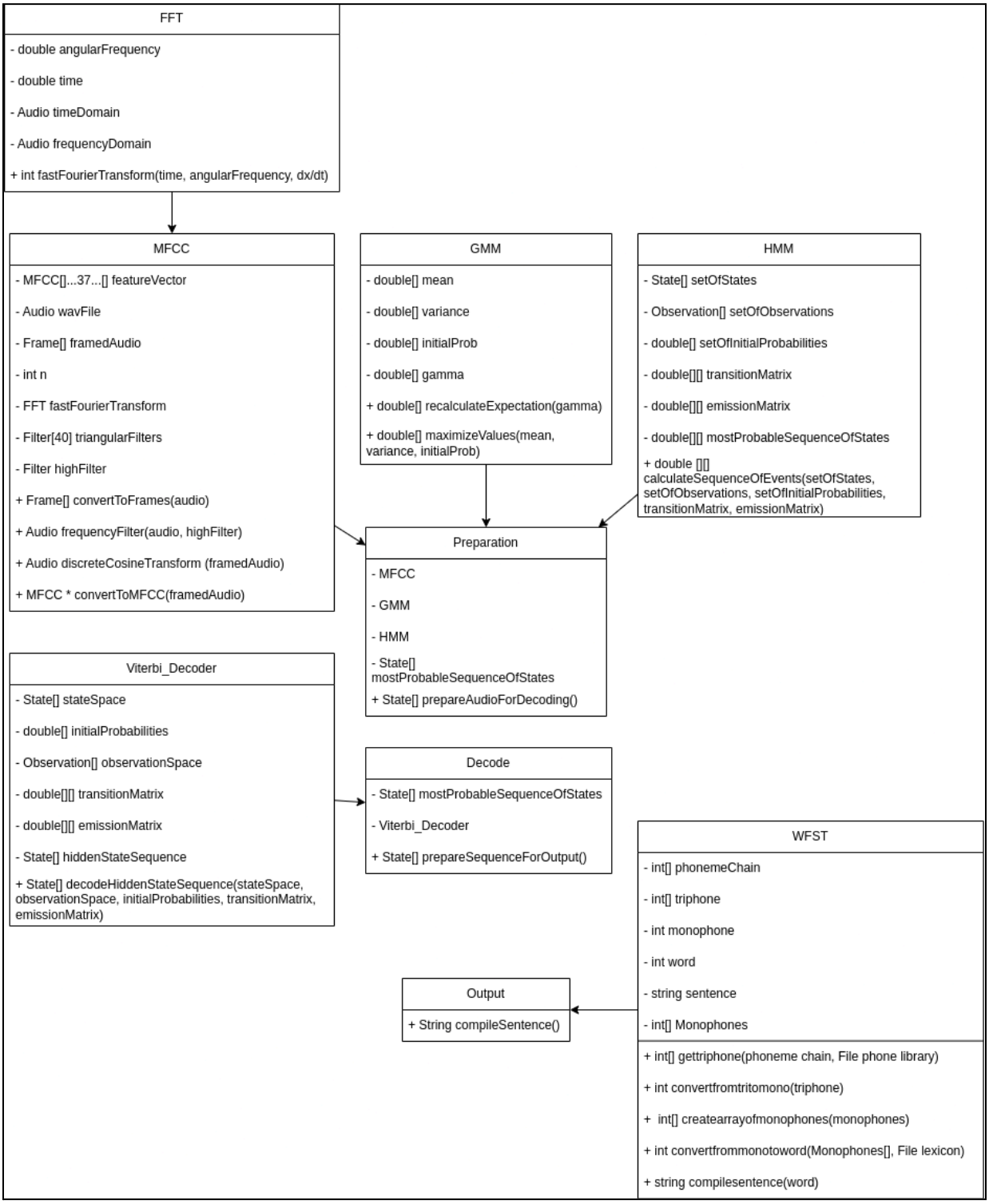

Figure 2: ASR Model Class Diagram V4.1.3

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

- Variables: Frame[] framedAudio, MFCC[]... 37…[] featureVector, Audio wavFile, int n, FFT fastFourierTransform, Filter[40] triangularFilters, Filter highFilter
- Methods: Frame[] convertToFrames(audio), Audio frequencyFilter(audio, highFilter), Audio discreteCosineTransform(framedAudio), MFCC * convertToMFCC(framedAudio)
- Inputs: .WAV audio file
- Outputs: 39-dimensional MFCC feature vector

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

- Variables: double angularFrequency, double time, Audio timeDomain, Audio frequencyDomain
- Methods: fastFourierTransform(time, angularFrequency, timeDomain)
- Inputs: time, timeDomain, angularFrequency
- Outputs: frequencyDomain

Gaussian Mixture Model (GMM): expectation maximization of the MFCC feature vector in reference to phoneme Gaussian distribution.

- Variables: double[] mean, double[] variance, double[] initialProb, double[] gamma
- Methods: double[] recalculateExpectation(gamma), double[] maximizeValues(mean, variance, initialProb)
- Inputs: Mean, Variance, Initial Probability
- Outputs: Gamma

Hidden Markov Model (HMM): Figures out the probability of the order of state changes based on observations.

- Variables: State[] setOfStates, Observation[] setOfObservations, double[] setOfInitialProbabilities, double[][] transitionMatrix, double[][] emissionMatrix, double[][] mostProbableSequenceOfStates
- Methods: double[][] calculateSequenceOfEvents(setOfStates, setOfObservations, setOfInitialProbabilities, transitionMatrix, emissionMatrix)
- Inputs: Set of all possible states, set of all observations, set of initial probabilities
- Outputs: Most probable sequence of states given observations

Viterbi Decoder: Uses calculations to find the state sequence.

- Variables: State[] stateSpace, double[] initialProbabilities, Observation[] observationSpace, double[][] transitionMatrix, double[][] emissionMatrix, State[] hiddenStateSequence
- Methods: State[] decodeHiddenStateSequence(stateSpace, observationSpace, initialProbabilities, transitionMatrix, emissionMatrix)
- Inputs: all the hmm stuff
- Outputs: most likely hidden state sequence

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 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.

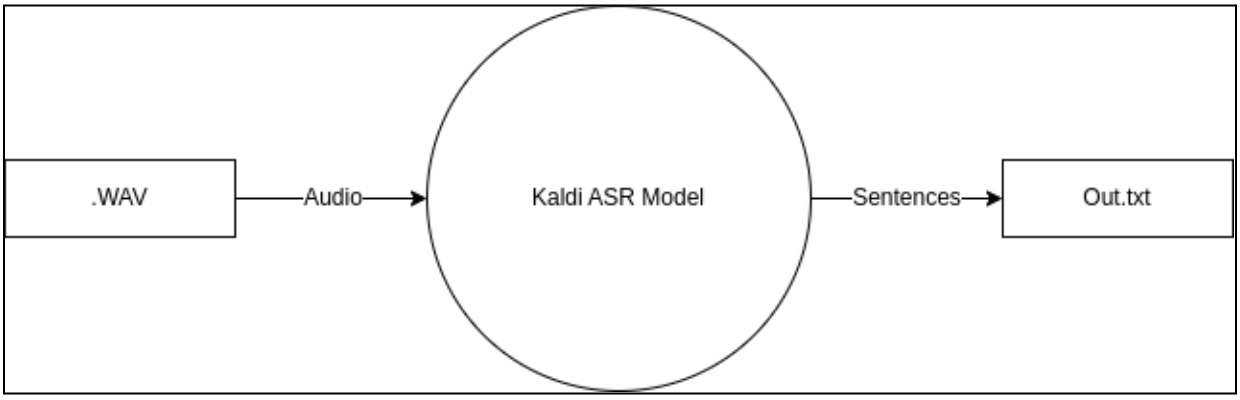

Figure 3: Acoustic ASR Model Level 0 Data Flow Diagram V4.1.1

3 HUMAN-MACHINE INTERFACE

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.

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.

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

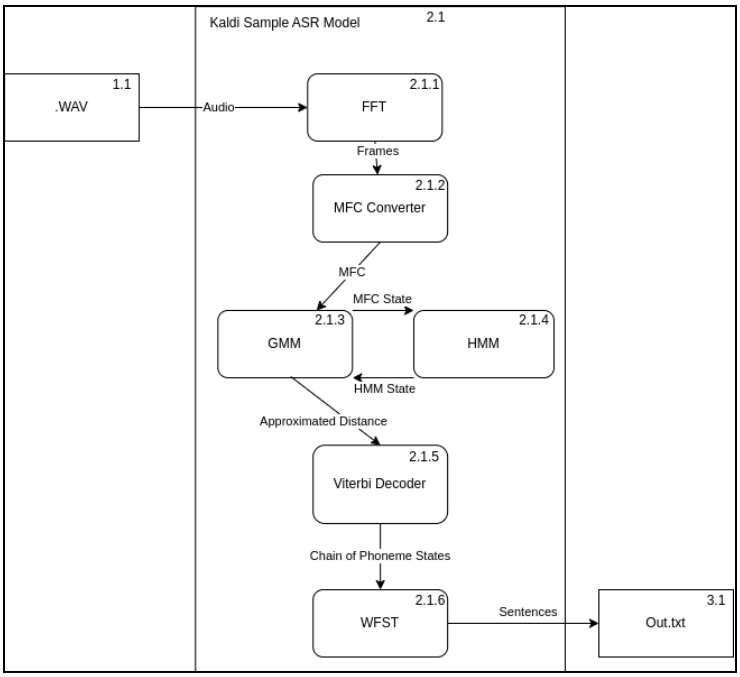

Figure 8: Sample ASR Model Level 1 Data Flow Diagram V4.1.1

```
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 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 10* illustrates how users and other actors shall interact with the ASR model. *Figure 11* illustrates how an ASR model is trained.

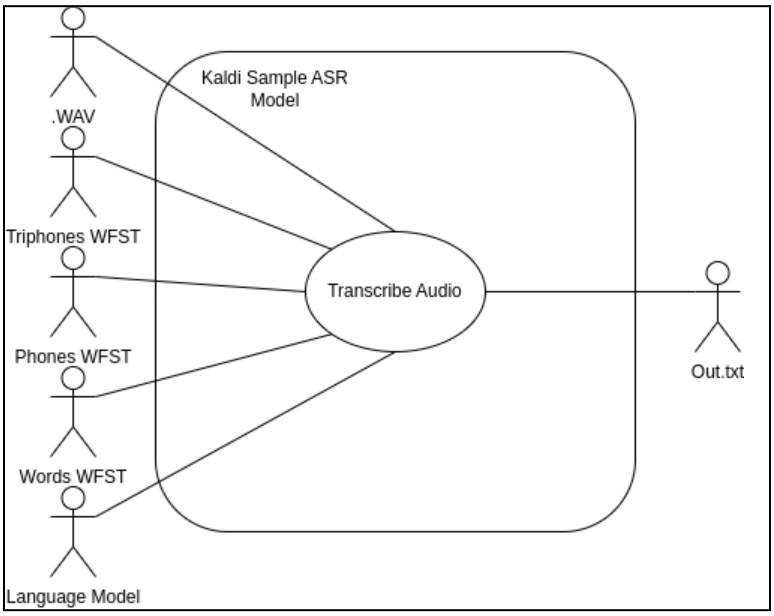

Figure 10: Kaldi ASR Toolkit Sample ASR Model Use Case Diagram V4.1.1

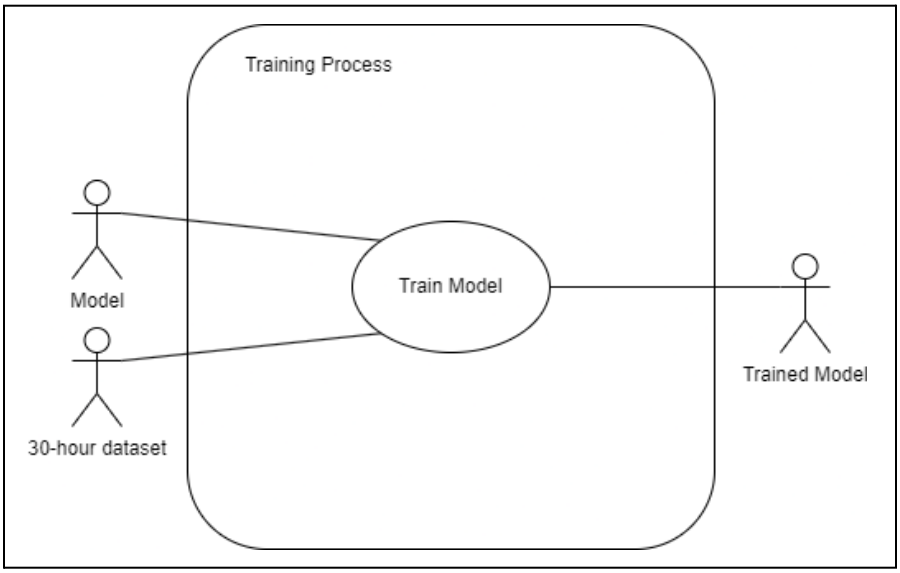

Figure 11: Kaldi ASR Model Training Use Case Diagram V2.2.1

Use Cases:

Use Case ID: UC1

Use Case Name: Transcribe Audio

Goal: The user inputs an audio file in the .WAV format and the ASR model shall return a text file containing the transcribed audio called “out.txt”.

Actors: WAV file, Phones WFST, Triphones WFST, Words WFST, Language Model, “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: The model outputs a text file

Step	Actor Action	Step	System Reaction
1	Inputs WAV file	2	Partitions audio data into 25-millisecond long frames in 10-millisecond long sliding intervals, allowing for three frames to overlap
		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.
6	Feeds to Triphone WFST	5	Calculates the distance to the most probable phoneme based on the MFCC feature vector and the GMM data to produce a chain of HMM states that represents triphones.
8	Feeds to Monophone WFST	7	Builds triphones from the chain of HMM states using the Viterbi Decoder.
10	Feeds to Word WFST	9	Converts triphones to monophones based on context-dependency
12	Feeds to Language Model	11	Builds words using the monophones
		13	Uses the previous two words to predict the following word
		14	Uses the Language Model to assemble the words into sentences
16	Outputs “out.txt”	15	Transfers sentences to a text file

Exceptions (Alternative Scenarios of Failure):

1. User inputs non WAV audio 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 WAV is imputed.
Step 0.1: Continue to wait

ALT 3: Step 2-15: The system fails to transcribe audio.
Step 2-15.1: The system displays an exception for the failed step.
Step 2-15.2: 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	Actor Action	Step	System Reaction
1	Input model		
2	Input training data	3	The model shall train itself to recognize speech and sound through changes in frequency by using the 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 13:1.1)* is processed by the ASR model (*Figure 13:2.1)*.

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 1: 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 padded 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 2: 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:

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

Formula 3: Hamming Window

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

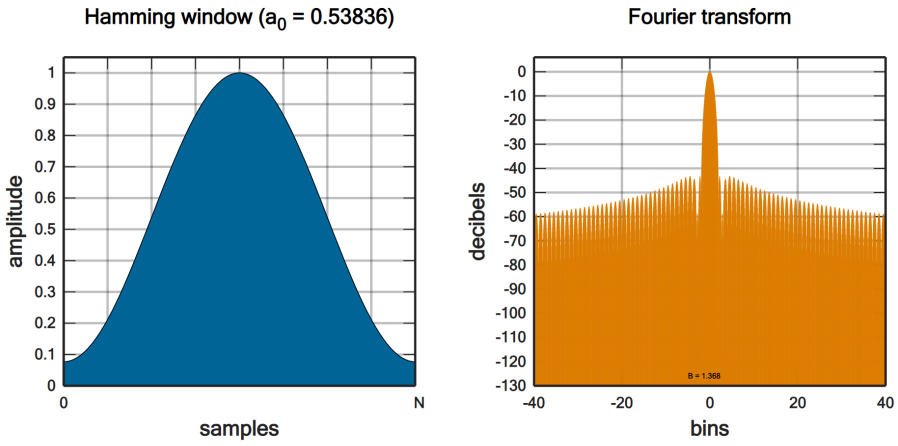

Figure 12: Example of a Hamming Window

The audio data is then converted from the time-domain to the frequency-domain using a fast Fourier transform (FFT) (*Figure 12: 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 4: 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 5: 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 6: 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 7: 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 8: 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 9: 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 13: 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 13: 2.1.4*) shall calculate the probability of the sequence of observable events, 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.

4.2.2 Decoding

The decoding operations shall be performed by a Viterbi decoder. The Viterbi decoder (*Figure 13: 2.1.5*) shall take in the initial probability distribution -- the probability that it shall start in that state; the sequence of observable events -- the order of observations based on time; the emission matrix -- the probability of the observation being generated from any given state *i*; the transition matrix, the probability of going to the next state; and the State Space -- the set of HMM state changes. It returns the most likely hidden state sequence.

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 13: 2.1.5*) shall give the Hidden Markov Model (*Figure 14: 2.1.6.1*) the chain of monophones which shall be used along with the Triphone Database (*Figure 14: 4.1*) to find the triphone. Which shall be passed to the context dependence (*Figure 14: 2.1.6.2*), which shall convert the triphones into monophones using the data from the phone database (*Figure 14: 5.1*). The monophones are then passed to the lexicon (*Figure 14: 2.1.6.3*), which uses the database of words (*Figure 14: 6.1*) to convert the monophones into words. The language model (*Figure 14: 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.

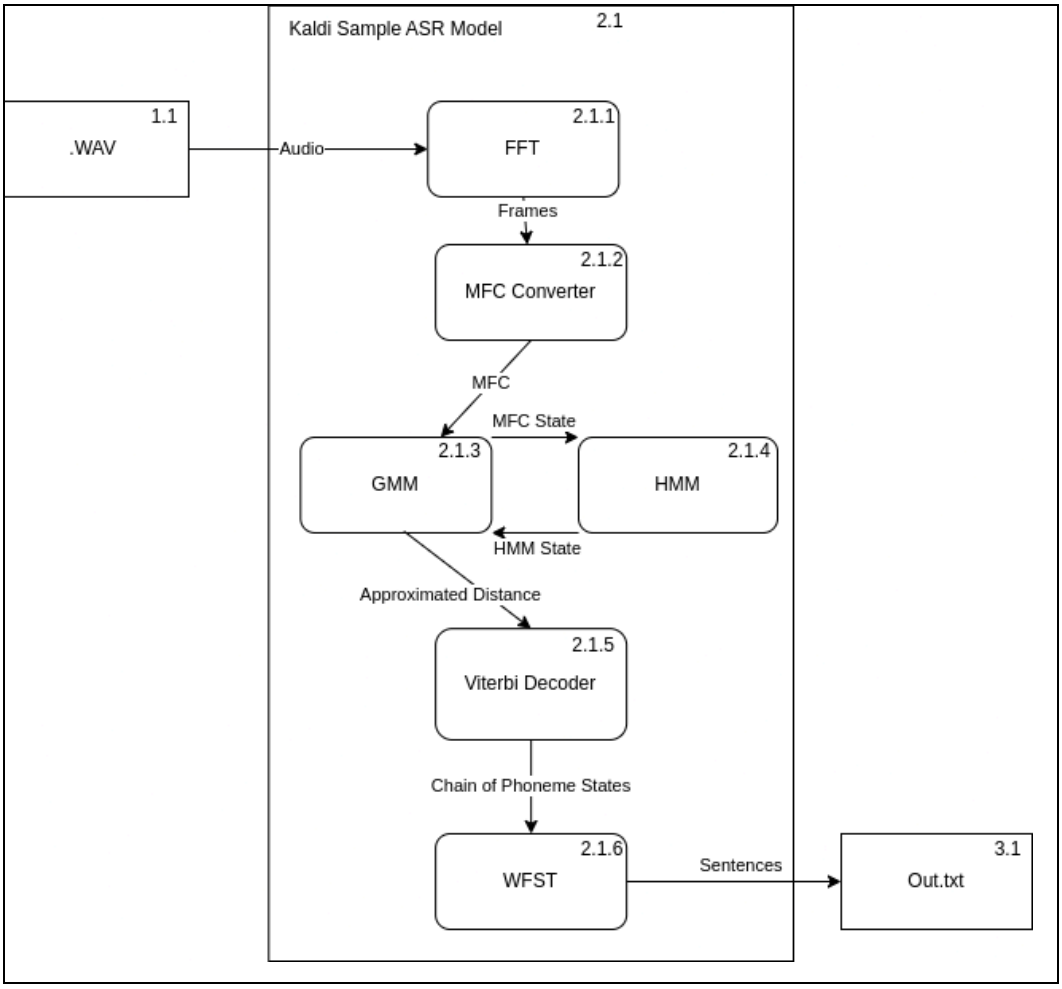

Figure 13: Kaldi ASR Toolkit Sample ASR Model Level 1 Data Flow Diagram V4.1.1

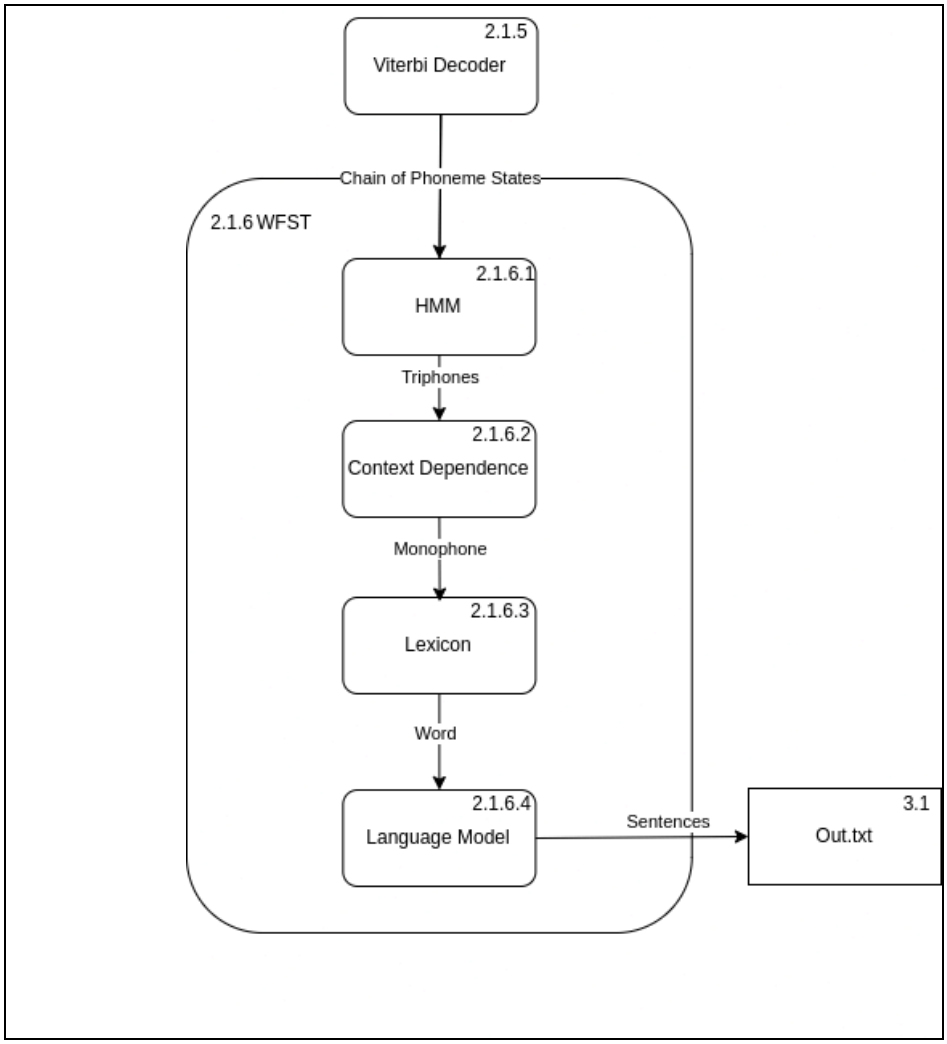

Figure 14: Kaldi ASR Toolkit Sample ASR Model Level 2 Data Flow Diagram V4.2.1

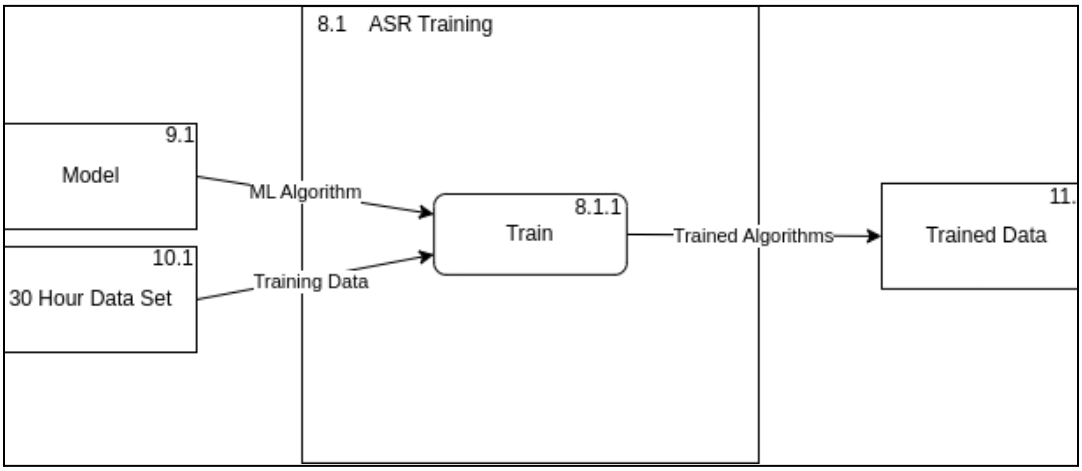

Figure 15: Kaldi ASR Toolkit Training Level 1 Data Flow Diagram V4.3.1

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 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)

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.).