LOW LEVEL DESIGN AND IMPLEMENTATION DOCUMENT

Regional Voice Assistant

UE18CS390B – Capstone Project Phase – 2

Submitted by:

P Deepak Reddy	**PES1201800344**
Chirag Rudresh	**PES1201801191**
Adithya A S	**PES1201801812**

Under the guidance of

Dr. Uma D
Designation
PES University

August - December 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

TABLE OF CONTENTS

Note:

Section 1	**Common for Prototype/Product Based and Research Projects**
Section 2 & 3	**Applicable for Prototype / Product Based Projects.**
Section 4	**Applicable for Research Projects.**
Appendix	**Provide details appropriately**

1. Introduction

1.1 Overview

The goal of the Low Level design is to visualize and understand the functioning of the regional voice assistant through algorithms, pseudo-code and representation of these functionalities in the form of class diagrams.

The regional assistant is divided into 5 main functional tasks - Splitting audio query, Speech-to-text, Next word prediction, POS tag prediction and finally integration into an app.

Each of these tasks further have functions and methods that ultimately transform a multilingual audio query into text, and recognising and translating the text query to generate the output to the user in an application.

The Low Level Design also shows the integration of these tasks through the simple visualisation diagrams like Class diagram, Sequence diagram and also understand the packages and architecture used to deploy these methods using Package and Deployment diagrams.

1.2 Purpose

The purpose of this low level design document is to show and understand the logical internal design of the actual program code. This is done on the basis of a high level design document. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.This provides an interface for all classes and helps to justify the choice of data structures used. The document gives a clear idea about the use of different algorithms and helps in maintenance and cost saving.

1.3 Scope

This low-level design document clearly specifies the way each component is implemented along with the pseudo code.The interaction between classes and dependencies among each component is shown. This also includes the interface design that the user interacts with. The use case diagrams,sequence diagrams and deployment diagrams for each component are specified. The results obtained for each component are shown.

2. Design Constraints, Assumptions, and Dependencies

- Using existing APIs to generate the query's output once speech is converted to text
- There is an assumption that API will generate an appropriate output to the query.
- Constraint on daily requests the existing API can handle
- Depending on the storage(Google Cloud, Firebase DB) for storing the queries and the results for models to learn from.

3. Design Description

3.1 Application Model

3.1.1 Description

This module describes the functionalities involved after the user records the audio and how the corresponding results are displayed to the user. Once the recording is completed,the following recording is uploaded to firebase storage with its timestamp. The backend then downloads the audio file with the most recent timestamp since that will be the most recent query from the user, deletes that audio file from the firebase storage and preprocess the audio file as required by the model. This preprocessed file is later sent to a speech-to-text model and output query is generated. The query is now converted to a single language with the help of google translate and then the links of top 5 results are taken from google search engine and this is saved in the firestore database. The app now displays this query along with the results for the user and the user can go to the corresponding result link or can delete that result. If the user is not satisfied with the results then they can now select the correct words predicted by the model to later further train the model and the corresponding query will be generated by sending the audio query directly to google translate.

3.1.2 Use case diagram

The following use case diagram shows different use cases the user is involved with while using the app. This includes all the elements that are displayed in the app as seen by the user

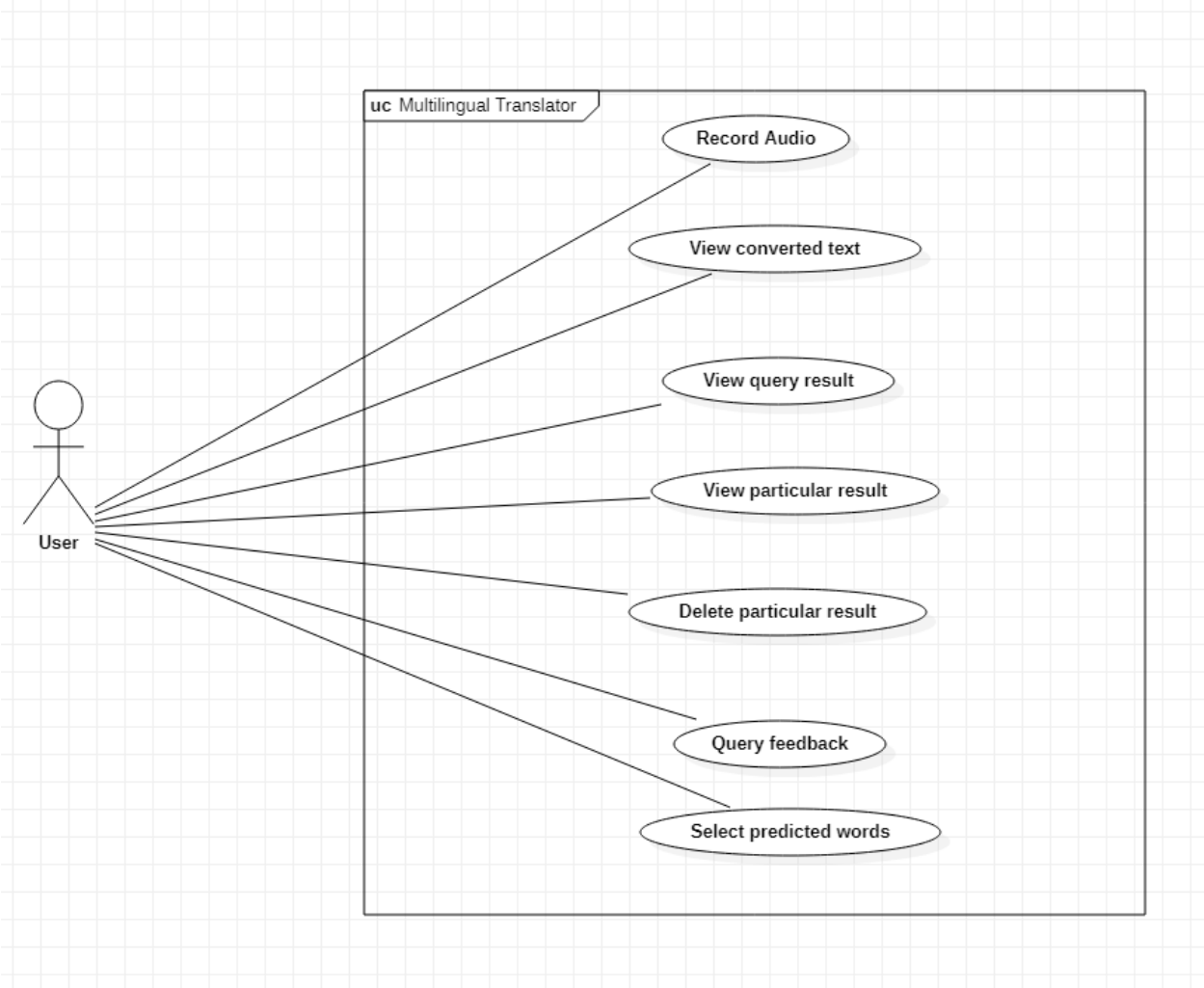


Fig 1 Use Case Diagram for application model

Use Case Item	Description
Record Audio	This allows user to record the input query
View converted Text	The multilingual translation of the audio to text from the model is shown
View query result	Top 5 results from google search for the corresponding query are shown. Each result is encapsulated in a container showing it's title,image and main description about that result.
View particular result	Among the top 5 results,the user can click onto any container to go to the corresponding link to view the entire page
Delete particular result	User can delete any result if he is unsatisfied with it
Query feedback	If the user is unsatisfied with the entire query then he can select 'Not satisfied' which helps in further training the model.
Select predicted words	If the user says he is not satisfied,then the list of predicted words will be shown for them to choose to further improve the model

3.1.3 Class Diagram

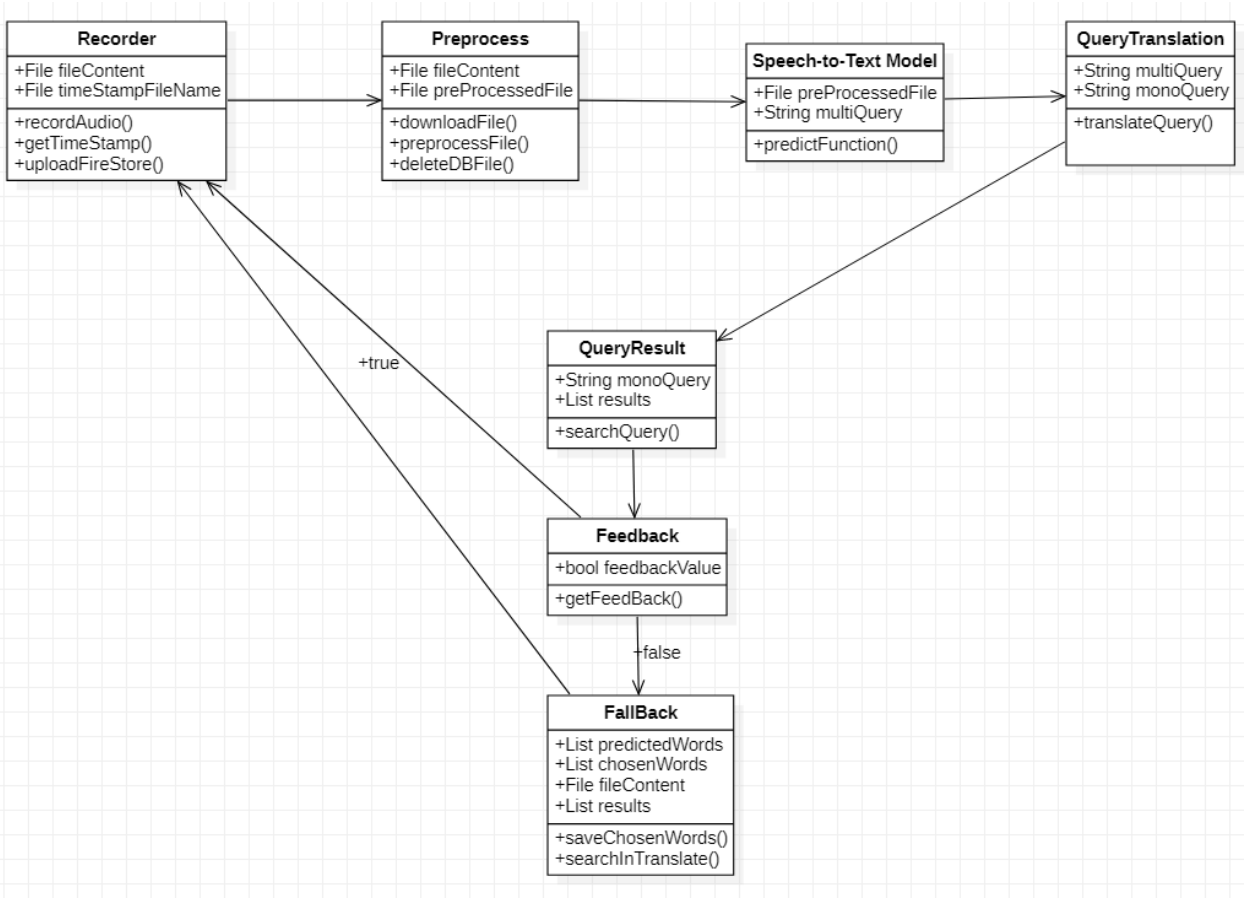


Fig 2 Class Diagram for application model

Class Name 1 :Recorder

Recorder class helps to get the multilingual audio input from the user and save it to the database.

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
File	fileContent	private	Null	Audio data file
File	timeStampfileContent	private	Null	Final Audio data file

recordAudio()

- Input - empty fileContent
- Output - fileContent with its location directory
- Parameters - fileContent
- Exceptions - Error in recording such as permission denied by user
- Pseudo-code

```
bool permission= getPermissionToRecord()
if(permission):
        record.start()
else:
        showErrorMessage()
```

getTimeStamp()

- Input - fileContent with its location directory
- Output - timeStampfileContent
- Parameters - fileContent

- Exceptions - Error with not finding the file in directory
- Pseudo-code

```
customPath=fileContent.path
if(customPath in directory):

        timestampfileContent.name=DateTime.now().millisecondsSinceEp
        och.toString()
else:
        showErrorMessage()
```

uploadFireStore()

- Input - timeStampFileContent
- Output - success after uploading
- Parameters - timeStampfileContent
- Exceptions - Error while uploading the file to firestore
- Pseudo-code

```
StorageReference storageRef =
FirebaseStorage.instance.ref().child(fileContent);

StorageUploadTask uploadTask = storageRef.putFile(
                timeStampfileContent,
                StorageMetadata(
                  contentType: 'audio/$_extension',
                ),
               );
if(uploadTask.success()):
```

```
        showSuccess()
else:
        showFailure()
```

Class Name 2: PreProcess

This class helps in changing the audio file into required format as specified by the model

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
File	fileContent	private	Timestamp of when the file was generated	Audio data file
File	preProcessedFile	public	Null	Changed format of the file

downloadFile()

- Input - fileContent
- Output - downloaded file
- Parameters - fileContent
- Exceptions - Error in getting permission to access db
- Pseudo-code

```
        bool permission= getAccessToDB()
        if(permission):
                fileContent=db.download(fileContent.name)
        else:
                showErrorMessage()
```

preProcessFile()

- Input - fileContent
- Output - preprocessedFile
- Parameters - fileContent
- Exceptions - Error with not finding the file in directory
- Pseudo-code

```
preProcessedFile=processAcordingtoModel(fileContent)
```

deleteDBFile()

- Input - fileContent
- Output - success after deleting
- Parameters - fileContent
- Exceptions - Error when getting access to database.Error when deleting the file
- Pseudo-code

```
bool permission= getAccessToDB()
if(permission):
        db.delete(fileContent)
        if(success()):
                showSuccessMessage()
        else:
                showFailureMessage()
else:
        showErrorMessage()
```

Class Name 3: Speech-To-Text Model

This class helps in getting the trained model and finding the output of the model

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
File	preProcessed File	public	Timestamp of when the file was generated	Preprocessed file content
String	multiQuery	public	Null	Text output of the model

predictFunction()

- Input - preProcessesFile
- Output - multiQuery
- Parameters - preProcessedFile
- Exceptions - Error within the model
- Pseudo-code

```
try:
        multiQuery=model.predit( preProcessedFile)
except:
        showErrorMessage()
```

Class Name 4: QueryTranslation

QueryTranslation translates the multilingual query of the user onto single language

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	multiQuery	public	Output of the model	Multilingual query of the user
String	monoQuery	public	Null	Translated single language query of the user

translateQuery()

- Input - multiQuery
- Output - monoQuery
- Parameters -multiQuery
- Exceptions - Error while importing libraries.
- Pseudo-code

```
try:
        import google
        monoQuery=google.translate(multiQuery)
except:
        showErrorMessage()
```

Class Name 5:QueryResult

This class shows the top 5 results of the query given by the user

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	monoQuery	public	Output from google translate	Single language query of the user
List	results	public	Null	Results generated by google search engine

searchQuery()

- Input - monoQuery
- Output - results
- Parameters -monoQuery
- Exceptions - Error while importing libraries.Too many requests error
- Pseudo-code

```
try:
        from google import search
        result=search(monoQuery,top5)
except:
        showErrorMessage()
```

Class Name 6:Feedback

This class helps to get feedback from the user to identify if the user is satisfied with the results or not.

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
bool	feedbackValue	public	Null	Value will be true if the user is satisfied with the results else false

getFeedBack()

- Input - feedbackValue
- Output - feedbackValue
- Parameters -feedbackValue
- Exceptions - When user does not select either of the options
- Pseudo-code

```
val=displayYesOrNo()
if  val==’Yes’:
        feedbackValue=True
        return feedbackValue
elif val==’No’:
        feedbackValue=False
        return feedbackValue
else:
        displayPromt()
```

Class Name 7:Fallback

Following class implements the fallback procedure that will happen when the feedback is negative from the user. The predicted words will be shown to the user where the user can select the correct words. Then these words will be sent to the google search engine. If the user isn't satisfied with the predicted words then the entire audio file will be sent to google API.

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
List	predictedWords	public	Predicted words from the model	List involves all the predicted words from the model
List	chosenWords	public	Null	This involves words as chosen by the user
File	fileContent	private	Timestamp of when the file was generated	Audio data file
List	results	public	Null	Results generated by google search engine

saveChosenWords()

- Input - predictedWords
- Output - chosenWords
- Parameters -predictedWords

- Exceptions - Error within the model
- Pseudo-code

```
switch  (userInput()):
        case  word1:
                chosenWords.append(word1)
                break
        case  word2:
                chosenWords.append(word2)
                break
        case  word3:
                chosenWords.append(word3)
                break
                .
                .
                .
                .
                .
case  wordn:
                chosenWords.append(wordn)
                break
default:
        showMessage("No predicted words are correct")
                break
```

searchInTranslate()

- Input - chosenWords,fileContent
- Output - results

- Parameters -chosenWords,fileContent
- Exceptions - Error when importing libraries
- Pseudo-code

```
if(len(chosenWords)==0):
    try:
        import google
        monoQuery=google.translate(fileContent)
        from google import search
        result=search(monoQuery,top5)
    except:
        showErrorMessage()
else:
    try:
        sentence=""
        for i in chosenWords:
            sentence+=i
            sentence+= ' '
        from google import search
        result=search(monoQuery,top5)
    except:
        showErrorMessage()
```

3.1.4 Sequence Diagram

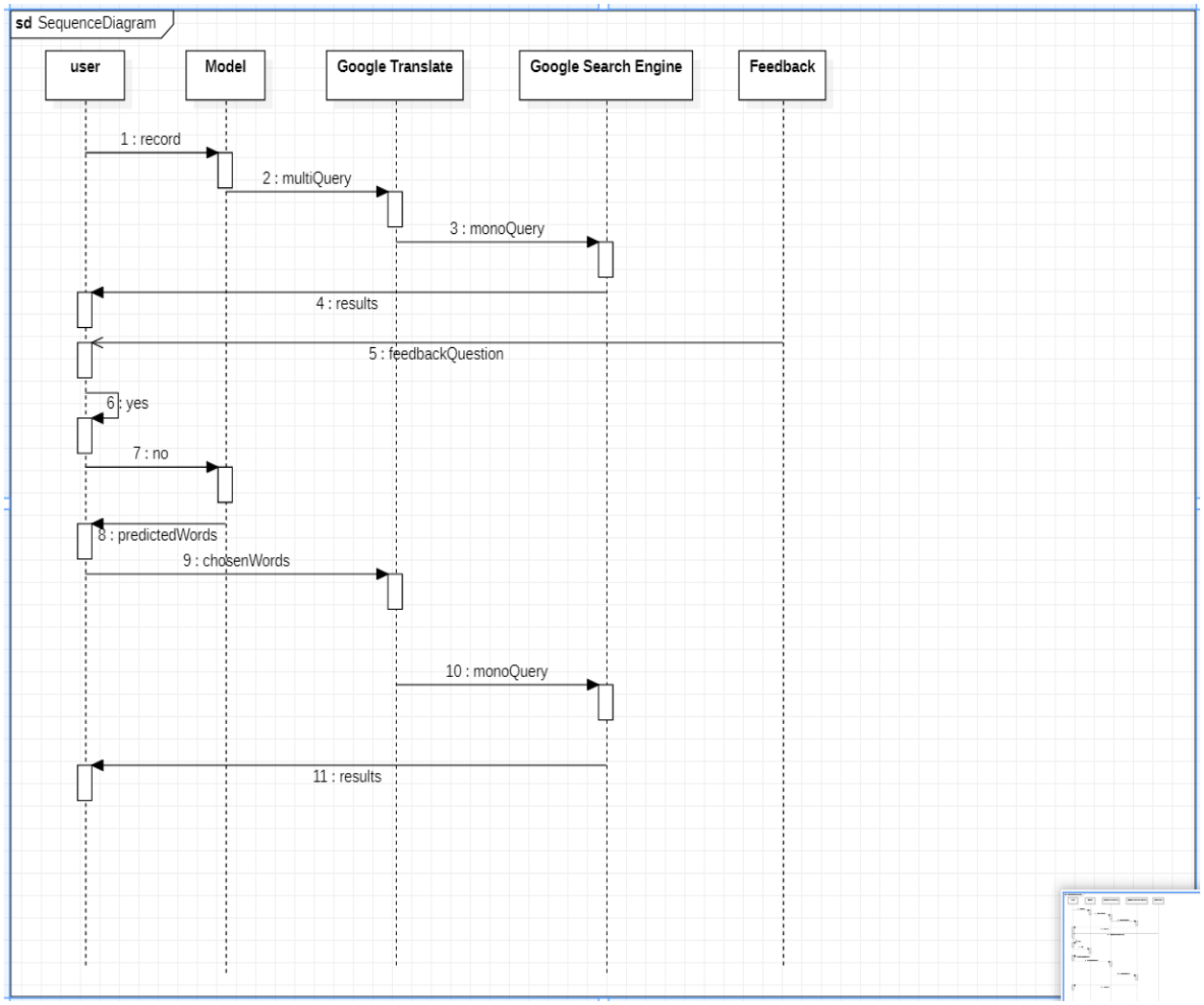


Fig 3 Sequence Diagram for application model

3.1.5 Packaging diagram

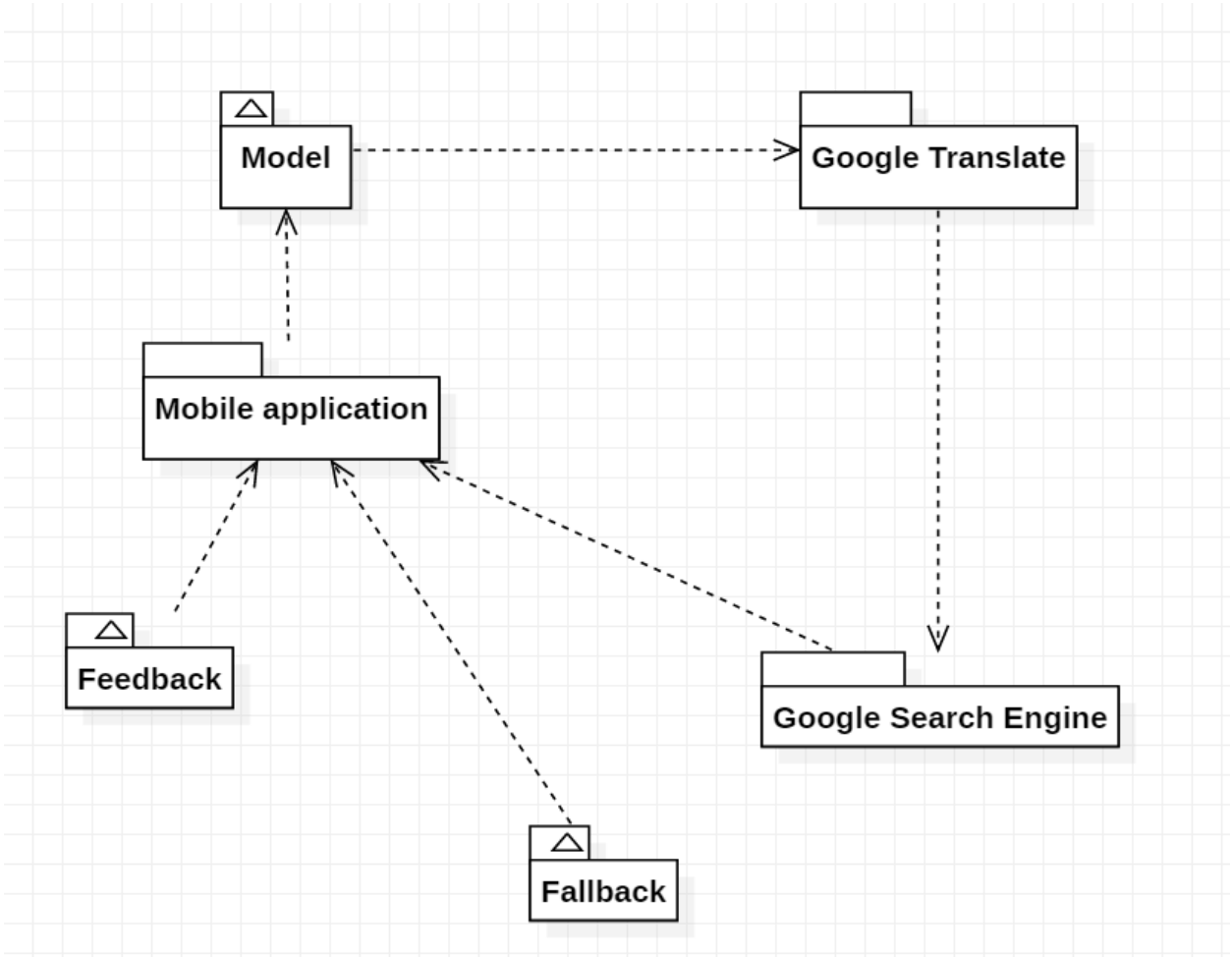


Fig 4 Packaging Diagram for application model

3.1.6 Deployment Diagram

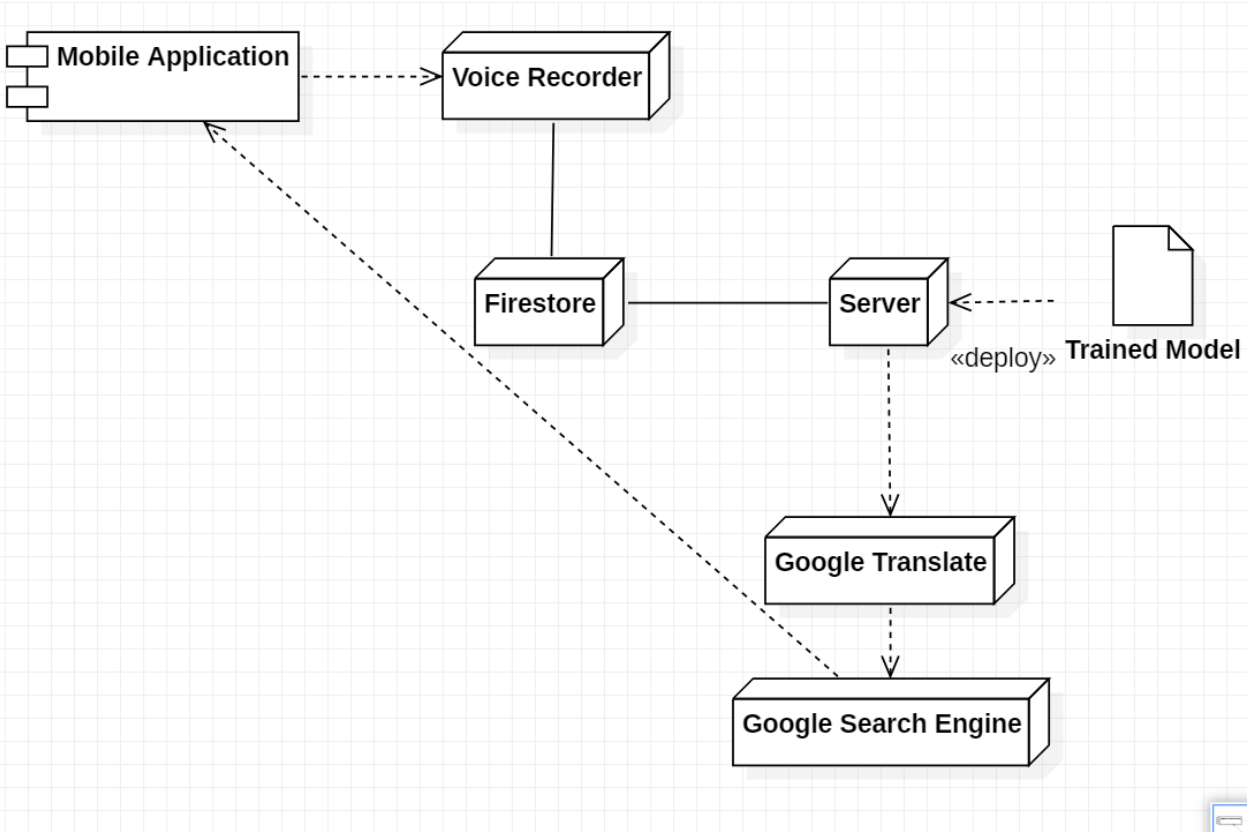


Fig 5 Deployment Diagram for application model

3.2 Splitting Module

3.2.1 Description

This module takes in a speech query as input and splits it into chunks based on word beginning and ending.So the number of chunks is nothing but the number of words in the speech query. These chunks will later be used to predict the uttered word.

3.2.2 Use case Diagram

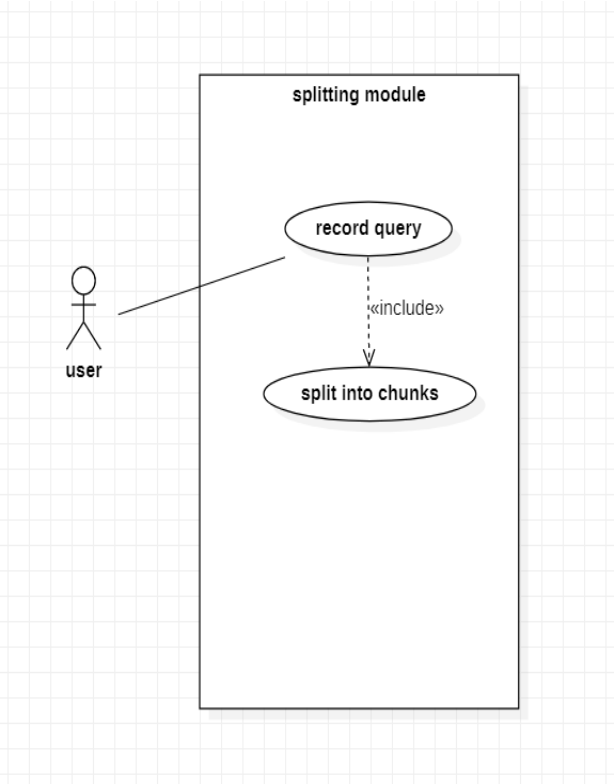


Fig 6 Use Case Diagram for splitting module

3.2.3 Class Diagram

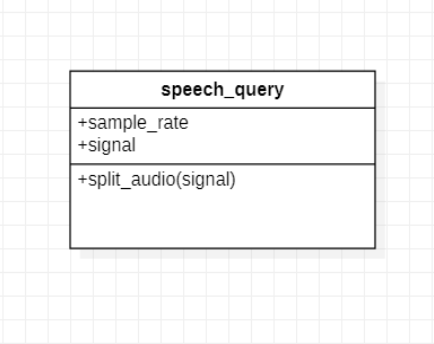


Fig 7 Class Diagram for splitting module

Class Name 1 : speech_query

The objects of this class depicts the signal which is obtained by reading the wav file using librosa library.

Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
integer	Sample_rate	public	22050	defines the number of samples per second (or per other unit) taken from a continuous signal to make a discrete or digital signal.
Numpy Array	signal	public	NULL	The audio file is depicted as a series of floating point values stored in Numpy Array which depicts the amplitude of the signal.

Methods

split_audio(signal)

Input - wav file

Output - list of chunks

Parameters - window_size

Exceptions - not required

Pseudo-code :

```
Window_size = 10000
Signal,sample_rate = librosa.read("path to wav file")
X = Moving_average(winodw_size)
result = split_at_minimas(X)
return result
```

3.2.4 Sequence Diagram

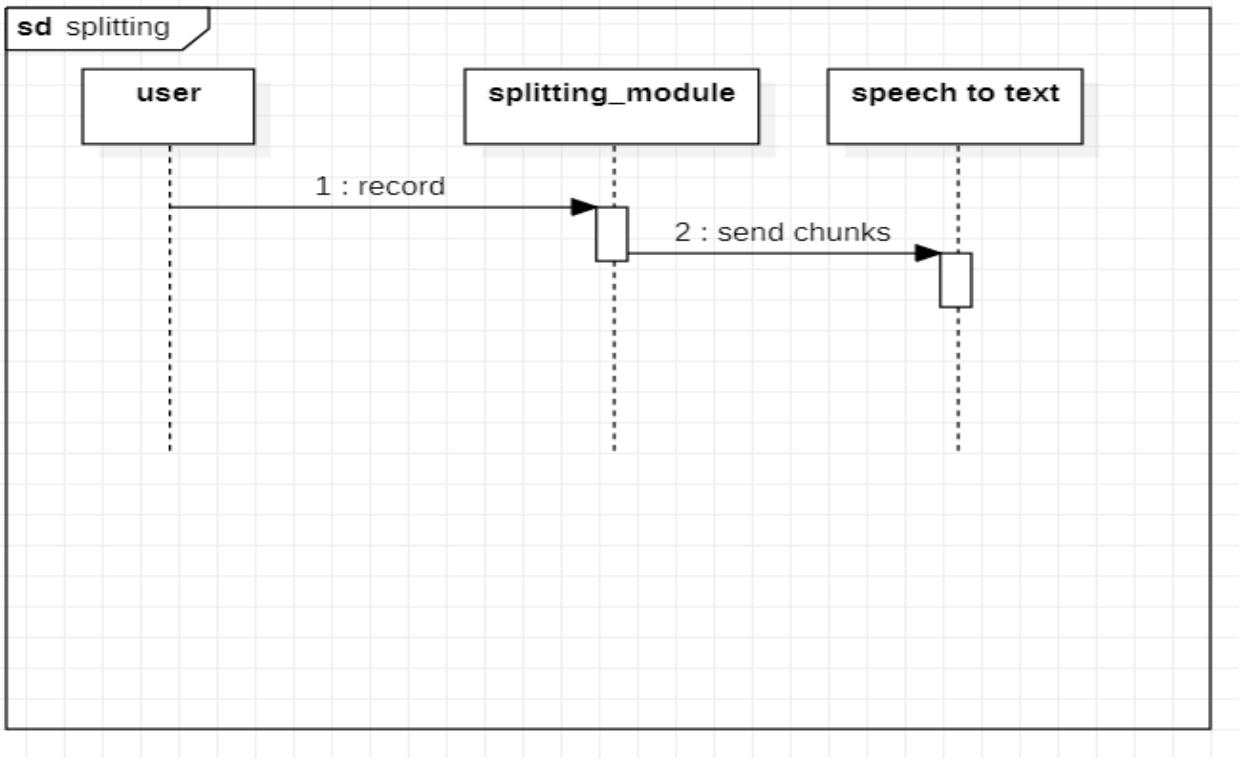


Fig 8 Sequence Diagram for splitting module

3.2.5 Packaging Diagram

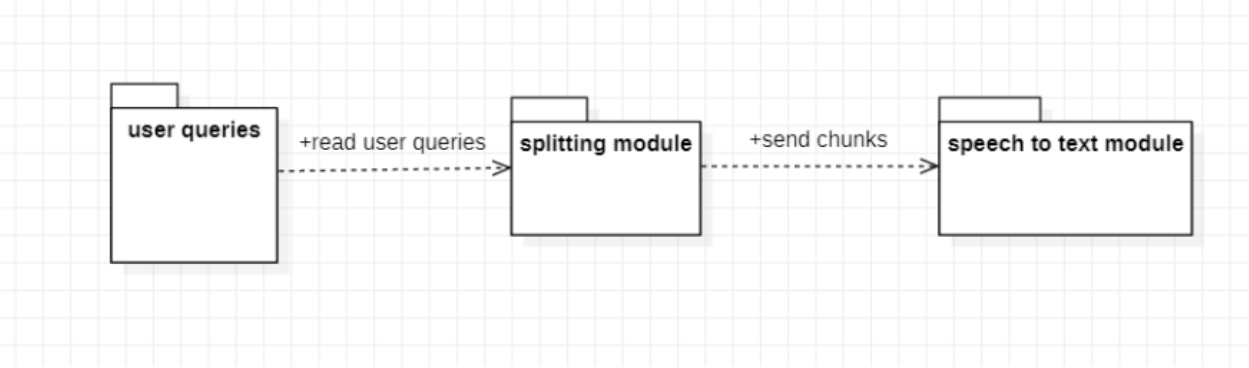


Fig 9 Packaging Diagram for splitting module

3.3 Speech to Text Module

3.3.1 Description

This module is responsible for converting the speech query into text.

3.3.2 Use Case Diagram

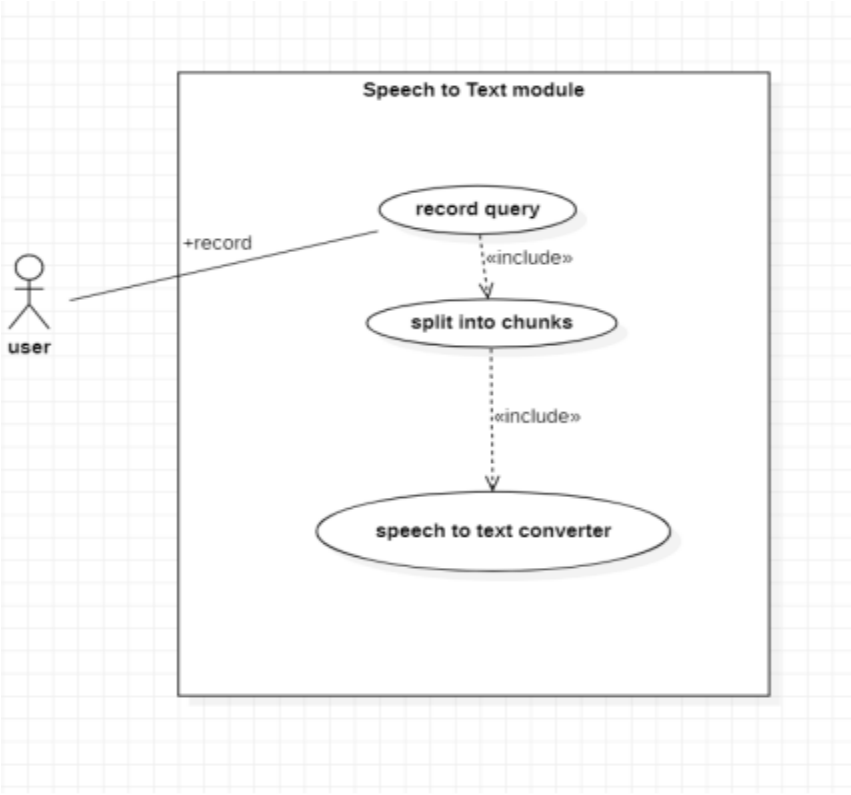


Fig 10 Use Case Diagram for speech to text module

3.3.3 Class Diagram

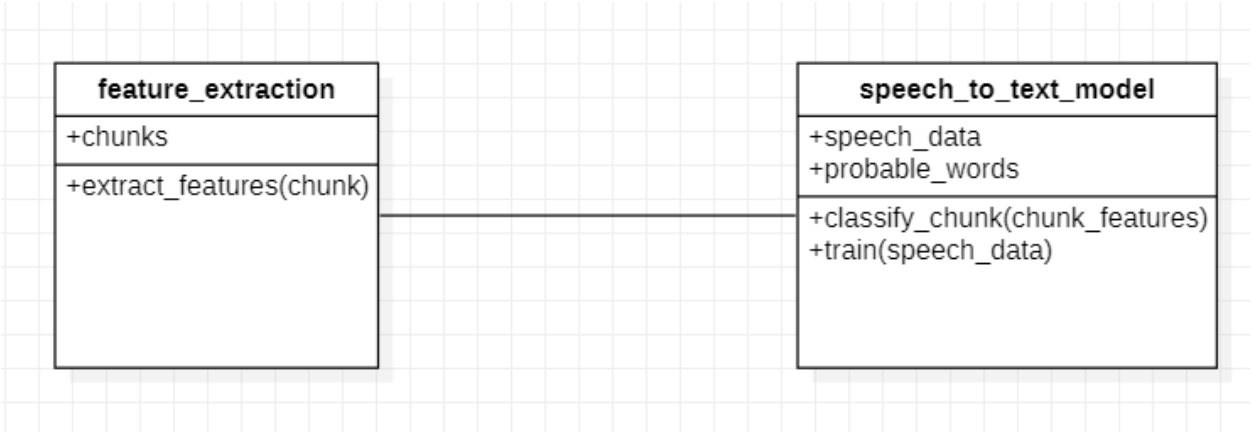


Fig 11 Class Diagram for speech to text module

Class Name : feature extraction

Data Type	Data Name	Access Modifiers	Initial Value	Description
List of chunks	chunks	public	Given by the splitting module	The signal for each chunk(word utterance) in the speech query is contained in a list.

Methods

extract_features(chunk)

This method takes each individual chunk from the list of chunks and extracts features for the same. MFCC features for each chunk are extracted using the librosa library.

Input - wav file

Output - list of chunks

Parameters - number of mfcc features

Exceptions - not required

Pseudo-code :

```
for chunk in chunks:
    librosa.extract_MFCC_features(chunk)
```

Class Name : speech to Text model

Data Type	Data Name	Access Modifiers	Initial Value	Description
Speech recordings in wav format	speech_data	public	-	All the speech recordings is stored in a directory and is accessed during training
List of Strings	Probable_words	public	-	All the words that may possibly occur next in the sentence are returned by the word prediction model.

Methods:

train(speech_data)

Input - all speech recordings

Output - list of chunks

Parameters - None

Exceptions - not required

Pseudo-code :

```
Model = create_NN_model()
Model.train(path_to_directory)
```

classify_chunk(chunk_features)

Input - chunk_features

Output - text format of the chunk

Parameters - None

Exceptions - not required

Pseudo-code :

```
X = Model.predict(chunk_features)

Return X
```

3.3.4 Sequence Diagram

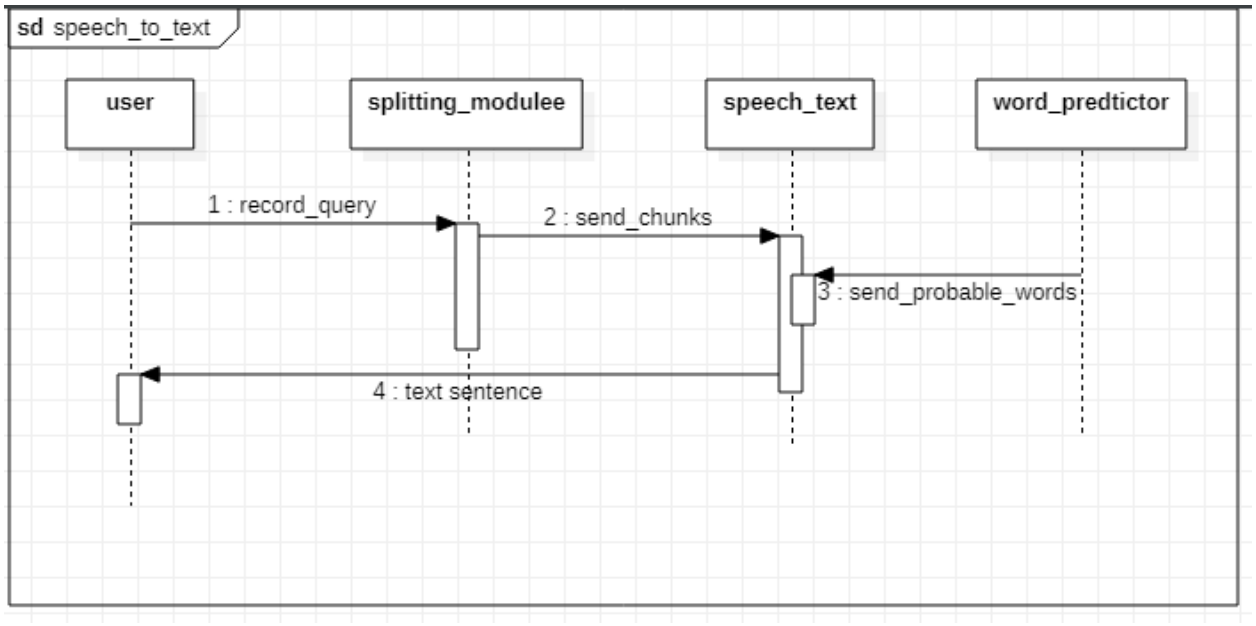


Fig 12 Sequence diagram for speech to text module

3.3.5 Deployment Diagram

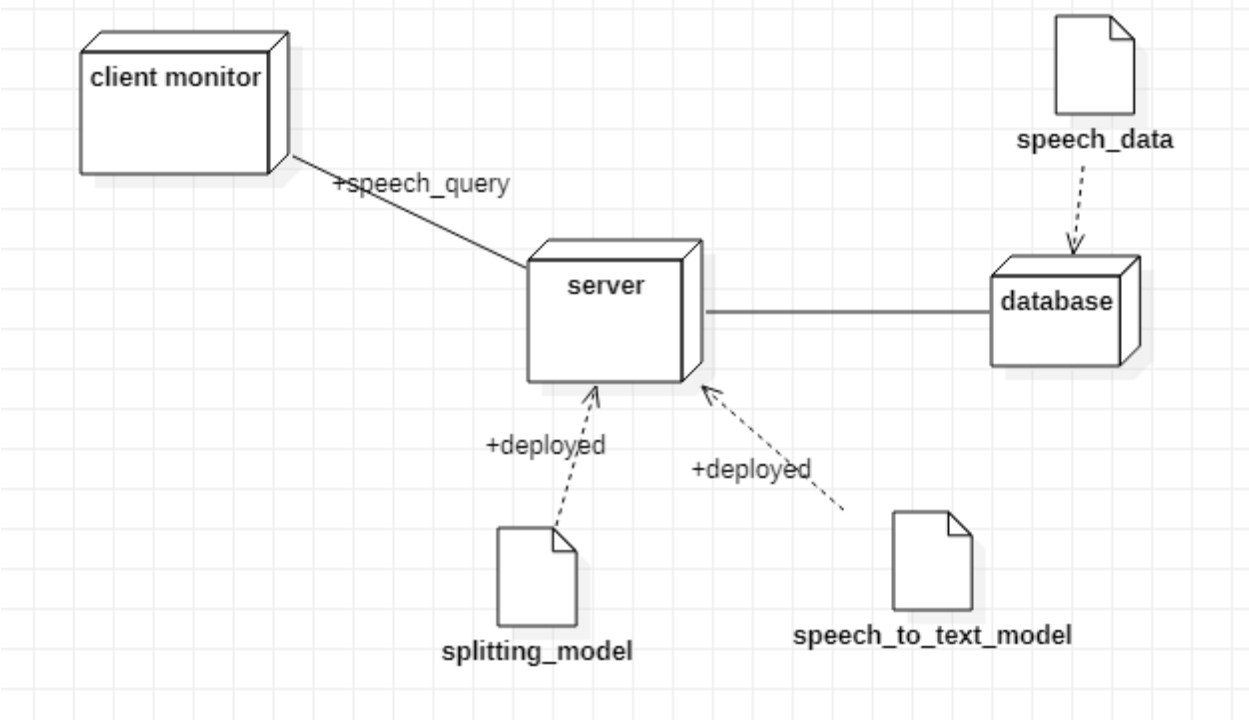


Fig 13 Sequence diagram for speech to text module

3.4 Word Prediction Model

3.4.1 Class Diagram

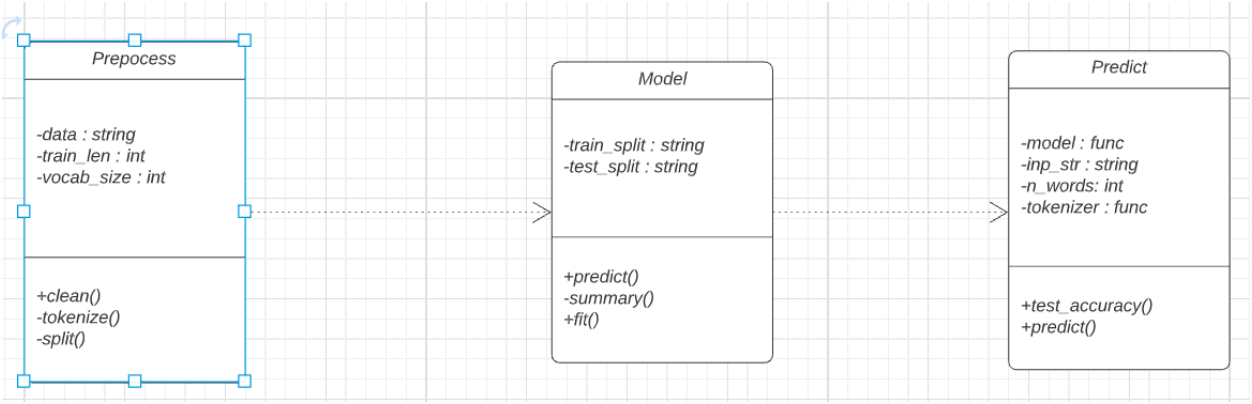


Fig 14 Class Diagram for word prediction model

Class Name 1 : Preprocess

Uses raw input data provided by the user and generates preprocessed, clean and tokenized text.

The class has the following methods:

. Clean() : this method takes the raw input data and removes spaces, punctuations and new lines characters. What remains is a single string which is comma separated containing all the words of the input data in the order that it existed in.

.Tokenize() : uses the NLTK word_tokenize() to set tokens to each word in the cleaned data. This results in a mapping between the tokens and the words present in the vocabulary.

. Split() : the tokenized word sequences are split into train_length number of words for training and the tokenized data is split into sequences for training. This is then assigned as training_data and the rest of the data is allocated for testing.

Class Name 2: Model

This class is the creation of a DNN using packages from keras that will be used to learn and train from the tokenized sequences to find and add weights to important words following prior words and form an accurate learning from the patterns.

The methods used in this class are:

. Predict() : this method takes in the padded query data and generates the next word from the learning that the model has done through training data.

. Summary() : is a method to visualize the various layers, input shapes and expected output shapes of the model created.

. Fit() : takes the input training data, the expected outputs for the training data and passes it through the model created for a specified number of epochs. Every epoch the weights and bias are automatically corrected by an optimizer called 'Adam'.

Class Name 3 : Predict

This class is used to generate the final output and test the model for accuracy with test data.

The methods used for this class are:

. Test_accuracy() : the method uses a cross-entropy equation to calculate the error between the generated output and the expected output of the model.

. **Predict() :** given the number of next words to be predicted, the predict method runs the input sequence through the model to generate the next possible 'n' words.

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	data	public	raw input multilingual sentences	contains raw data from multilingual dataset
Int	train_len	public	0	Number of words to train model
Int	vocab_size	public	0	Number of unique words in the input data set

3.5 POS Tag Prediction Model

3.5.1 Class Diagram

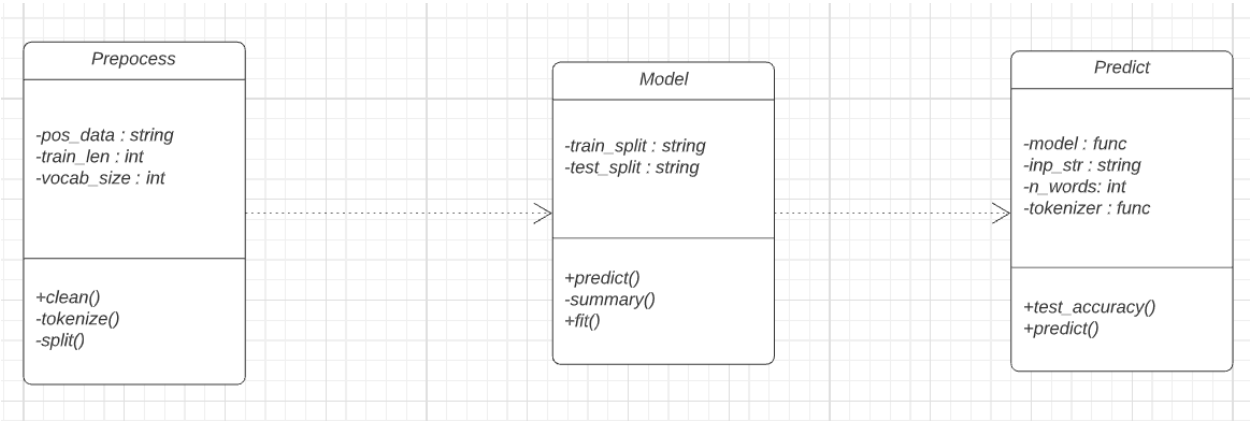


Fig 15 Class diagram for POS tag prediction module

Class Name 1 : Preprocess -

Uses POS tagged input data provided by the user and generates preprocessed, clean and tokenized text.

The class has the following methods:

. Clean() : this method takes the POS tagged input data and removes spaces, punctuations and new lines characters. What remains is a single string which is comma separated containing all the POS tags of the input word data in the order that it occurs in.

.Tokenize() : uses the NLTK word_tokenize() to set tokens to each POS tag in the cleaned data. This results in a mapping between the tokens and the words present in the vocabulary.

. Split() : the tokenized word sequences are split into train_length number of words for training and the tokenized data is split into sequences for training. This is then assigned as training_data and the rest of the data is allocated for testing.

pseudo-code:

```
clean() = data.lower()
tokenize = clean.word_tokenize()
vocab_size = len(tokenizer[train_len])
split() = Tokenizer().fit_text_to_sequences(tokenize)
```

Class Name 2: Model -

This class is the creation of a DNN using packages from keras that will be used to learn and train from the tokenized sequences to find and add weights to important words following prior words and form an accurate learning from the patterns.

The methods used in this class are:

. Predict() : this method takes in the padded POS query and generates the next word from the learning that the model has done through training data.

. Summary() : is a method to visualize the various layers, input shapes and expected output shapes of the model created.

. Fit() : takes the input training data, the expected outputs for the training data and passes it through the model created for a specified number of epochs. Every epoch the weights and bias are automatically corrected by an optimizer called 'Adam'.

Class Name 3 : Predict -

This class is used to generate the final output and test the model for accuracy with test data.

The methods used for this class are:

. Test_accuracy() : the method uses a cross-entropy equation to calculate the error between the generated output and the expected output of the model.

. **Predict() :** given the number of next POS to be predicted, the predict method runs the input sequence through the model to generate the next possible 'n' words.

pseudo-code:

```
clean() = data.lower()
tokenize = clean.word_tokenize()
vocab_size = len(tokenizer[train_len])
split() = Tokenizer().fit_text_to_sequences(tokenize)
```

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	pos_tag	public	Tagged POS words	POS tag of each kannada and english word
Int	train_len	public	0	Number of words to train model
Int	vocab_size	public	0	Number of unique words in the input data set

3.5.2 Sequence Diagram

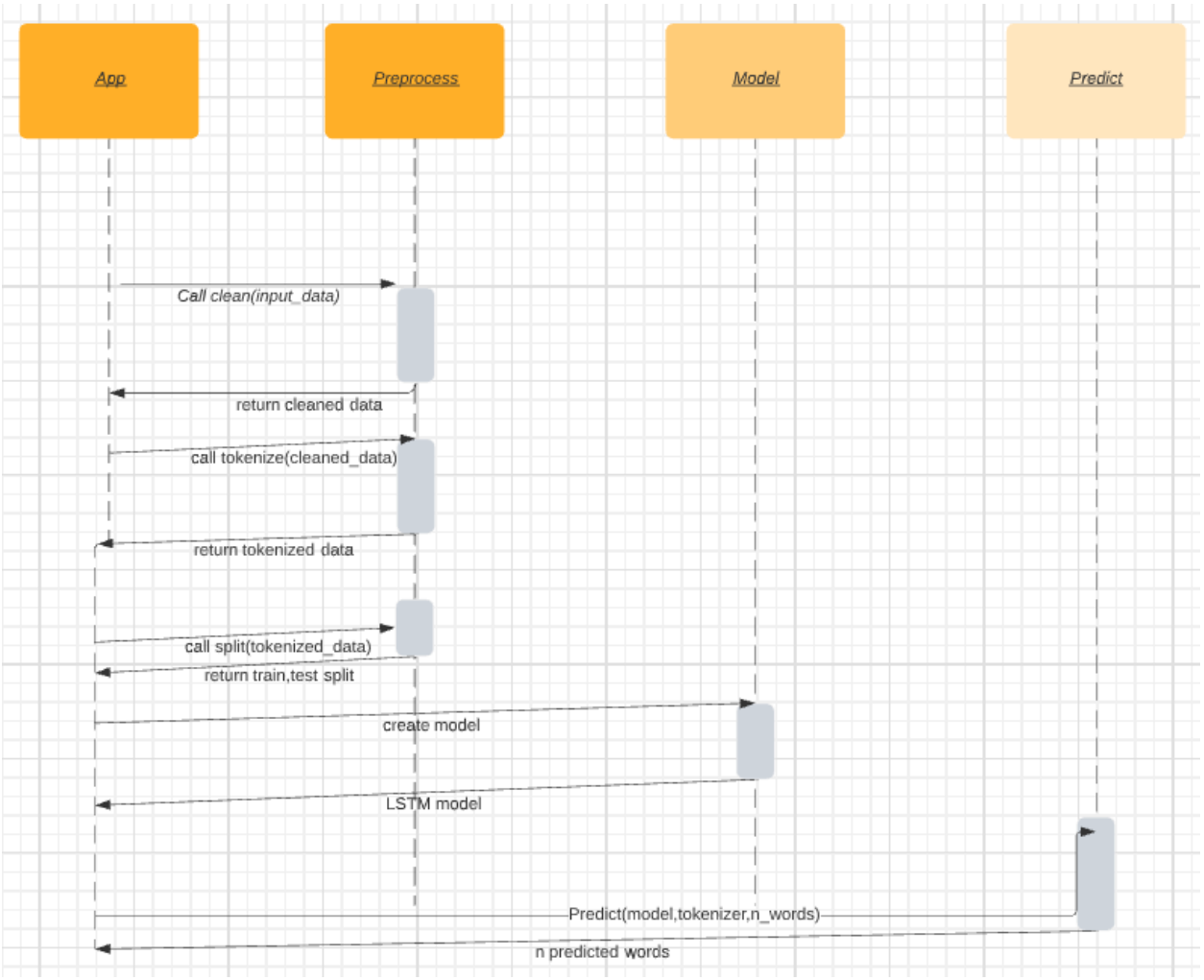


Fig 15 Sequence diagram for POS tag prediction module

3.5.3 Deployment Diagram

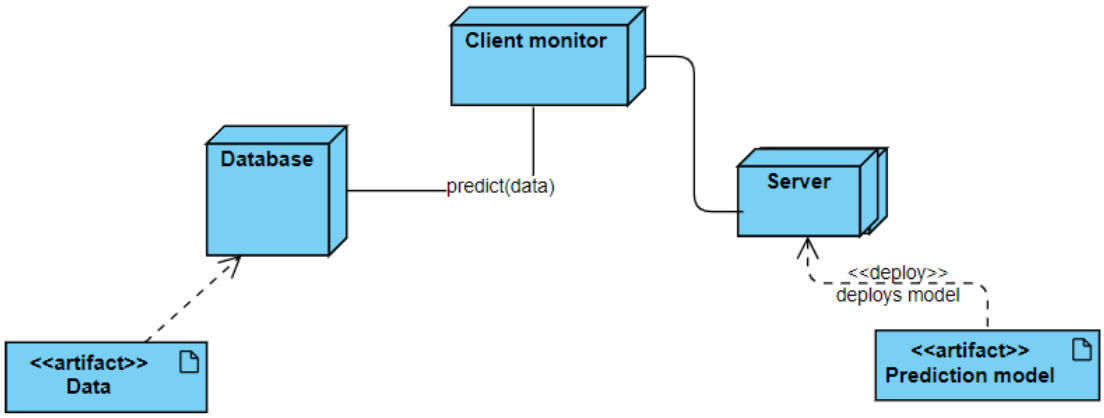


Fig 16 Sequence diagram for POS tag prediction module

4. Proposed Methodology / Approach

The approach used for recognition and translation of multilingual audio query is as follows :

- The input audio wav file containing the query is split into individual wav files each containing the words of the query.
- These wav files are then passed to a predictive model that uses a deep learning model to map the audio to text and generate the text output for the corresponding words.
- The accuracy of the speech-to-text model is further increased using a next-word prediction model and a POS tag prediction model.

  Both these prediction models take in a sequence of words, tags respectively and use a RNN to generate the next possible ‘n’ words,tags that follow.

  These words,tags are then used to decrease the search space for the speech-to-text conversion model for better results.

- The multilingual text query is passed through to Google Translation API to get the corresponding monolingual query which is then passed to the Search Engine to get the appropriate output.
- The entire process of recording the audio query to display the results is integrated into a user-friendly application.

The main constraint of this method is that the training audio and textual multilingual query dataset needed for the speech-to-text conversion model is very high(in terms of hundreds of thousands), which is not feasible with the team size and the time constraint. But this can be overcome, by expanding the dataset by generating recordings of different age groups,gender and language dialects.

This methodology also depends on the performance of the Google translation API and the Search Engine for the accuracy of the translation and output. The application depends on the storage constraints of the Database used as well as the limit on the amount of audio and textual queries that can be stored.

4.1 Algorithm and Pseudocode

4.1.1 Splitting model

Smoothing average for the signal using a window of size 1000 is calculated and stored in smooth_list variable. On this list , The minimas are found

```
minimas = []
i = 1
while isNaN(smooth_list[i]):
    i = i + 1

print(i)
window = 10000

while i + window < len(smooth_list):
    #print("hi")
    m = min(smooth_list[i:i+window])
    idx = smooth_list.index(m)
    #print(m,idx,smooth_list[i],smooth_list[i+window-1])

    if m != smooth_list[i] and m != smooth_list[i+window-1]:
        if len(minimas) != 0  and abs(idx - minimas[-1][0]) > 5000:
            minimas.append([idx,m])
        if len(minimas) == 0:
            minimas.append([idx,m])

    i = i + window

print(minimas)
```

4.1.2 Speech to Text model

A deep learning model is created with two hidden layers. The MFCC feature vector will be passed as input to the input layer and the last layer will contain as many neurons as we want to classify from.

The hidden layers are activated with ReLu whereas the last layer has activation function softmax.

```
model=Sequential()
###first layer
model.add(Dense(100,input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###second layer
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###third layer
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.5))

###final layer
model.add(Dense(num_labels))
model.add(Activation('softmax'))
```

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 100)               4100
_________________________________________________________________
activation_24 (Activation)   (None, 100)               0
_________________________________________________________________
dropout_18 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_25 (Dense)             (None, 200)               20200
_________________________________________________________________
activation_25 (Activation)   (None, 200)               0
_________________________________________________________________
dropout_19 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_26 (Dense)             (None, 100)               20100
_________________________________________________________________
activation_26 (Activation)   (None, 100)               0
_________________________________________________________________
dropout_20 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_27 (Dense)             (None, 5)                 505
_________________________________________________________________
activation_27 (Activation)   (None, 5)                 0
=================================================================
```

4.1.3 Word and POS tag prediction model

```
model = Sequential()
model.add(Embedding(vocabulary_size, seq_len, input_length=seq_len))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(50,activation='relu'))
model.add(Dense(vocabulary_size, activation='softmax'))
print(model.summary())
# compile network
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(train_inputs,train_targets,epochs=500,verbose=1)
model.save("mymodel.h5")
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 3, 3)              4452
_________________________________________________________________
lstm (LSTM)                  (None, 3, 50)             10800
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 50)                2550
_________________________________________________________________
dense_1 (Dense)              (None, 1484)              75684
=================================================================
Total params: 113,686
Trainable params: 113,686
Non-trainable params: 0
_________________________________________________________________
None
```

4.1.4 Application Implementation algorithm

Application code

```
recordAudio=record.start()

record.stop()

sendAudioToBackend(recordAudio)

results=sendReultsToApp(results)

takeFeedback()

if(feedback==positive):
        continue
else:
        askFallBackFromBackend()
        showFallBack
```

Backend code :

```
fileAudio=recvFromApp()

preProcessedAudio=audio.preProcess(fileAudio)

multiQuery=model.predict(preProcessedAudio)

monoQuery=google.translate(multiQuery)

results=google.search(monoQuery)

sendReultsToApp(results)



if(recvFallBackFromApp()){
```

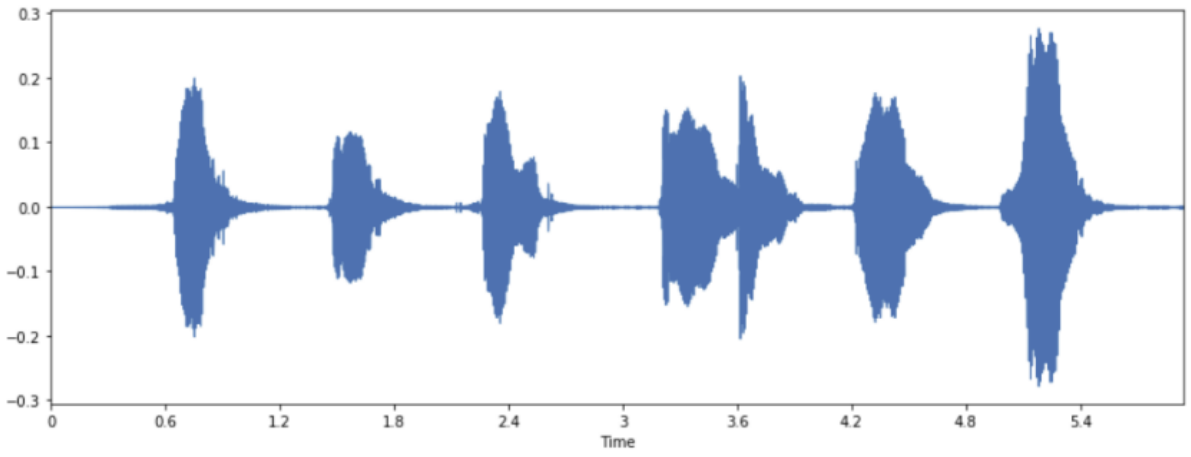
```
                sendFallBack()

        }
```

4.2 Implementation and Results

4.2.1 Splitting model

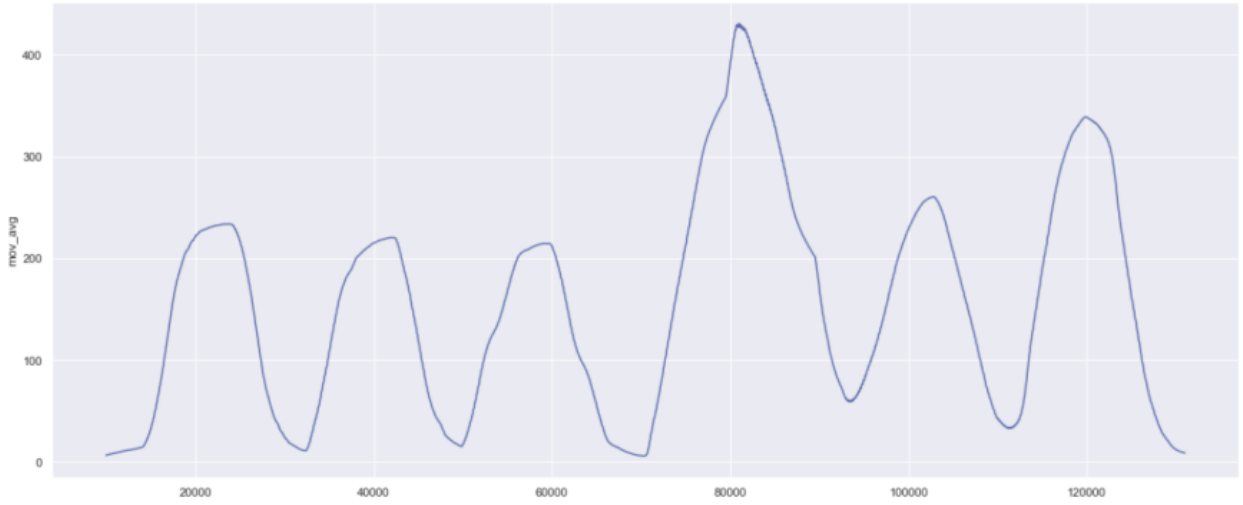
Here is visualisation of the wav file before and after smoothing for speech which says "how to painting on gode"

Before smoothing, The word "painting" part of the wav file would show characteristics of two different words



After smoothing:

The word “painting” part is now smoothed and now looks like a single word utterance as we desired.



Indexes where the sentence has to be split:

```
print(minimas)
```

```
9999
[[32369, 11.05272972598896], [49817, 15.221263921066324], [69983, 6.132479933303117], [93381, 58.94959320834991], [109987, 42.5
95168354146296]]
```

4.2.2 Deep learning model for biased dataset(only 2 voices)

```
In [293]: from tensorflow.keras.callbacks import ModelCheckpoint
          from datetime import datetime

          num_epochs = 300
          num_batch_size = 4

          model.fit(X_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/300
40/40 [==============================] - 1s 12ms/step - loss: 69.8481 - accuracy: 0.0999 - val_loss: 11.2026 - val_accuracy:
0.0750
Epoch 2/300
40/40 [==============================] - 0s 5ms/step - loss: 37.7951 - accuracy: 0.1595 - val_loss: 3.3224 - val_accuracy: 0.
2000
Epoch 3/300
40/40 [==============================] - 0s 6ms/step - loss: 22.7985 - accuracy: 0.0936 - val_loss: 2.9853 - val_accuracy: 0.
1250
Epoch 4/300
40/40 [==============================] - 0s 6ms/step - loss: 15.8517 - accuracy: 0.1571 - val_loss: 2.9889 - val_accuracy: 0.
2000
Epoch 5/300
40/40 [==============================] - 0s 5ms/step - loss: 13.2711 - accuracy: 0.1265 - val_loss: 2.2602 - val_accuracy: 0.
2000
Epoch 6/300
40/40 [==============================] - 0s 5ms/step - loss: 10.9069 - accuracy: 0.0643 - val_loss: 2.2343 - val_accuracy: 0.
1750
Epoch 7/300
40/40 [==============================] - 0s 4ms/step - loss: 7.6222 - accuracy: 0.1197 - val_loss: 2.2733 - val_accuracy: 0.0
```

```
In [294]: test_accuracy=model.evaluate(X_test,y_test,verbose=0)
          print(test_accuracy[1])
```

```
0.9750000238418579
```

4.2.3 Word and POS tag prediction models

Accuracy:

```
Epoch 497/500
198/198 [==============================] - 1s 3ms/step - loss: 0.3989 - accuracy: 0.8436
Epoch 498/500
198/198 [==============================] - 1s 4ms/step - loss: 0.3881 - accuracy: 0.8443
Epoch 499/500
198/198 [==============================] - 1s 3ms/step - loss: 0.3962 - accuracy: 0.8442
Epoch 500/500
198/198 [==============================] - 1s 4ms/step - loss: 0.3941 - accuracy: 0.8398
```

Results:

```
for i in (model.predict(pad_encoded)[0]).argsort()[-3:][::-1]:
  pred_word = tokenizer.index_word[i]
  print("Next word suggestion:",pred_word)
```

```
how to hang
[2, 1, 901] [[  2   1 901]]
Next word suggestion: painting
Next word suggestion: tooka
Next word suggestion: a
```

4.2.4 Application UI

Application development was done on Flutter using concepts of object oriented programming to make an efficient app. All the classes as stated in the design description were designed using camel case and pascal case naming convention.

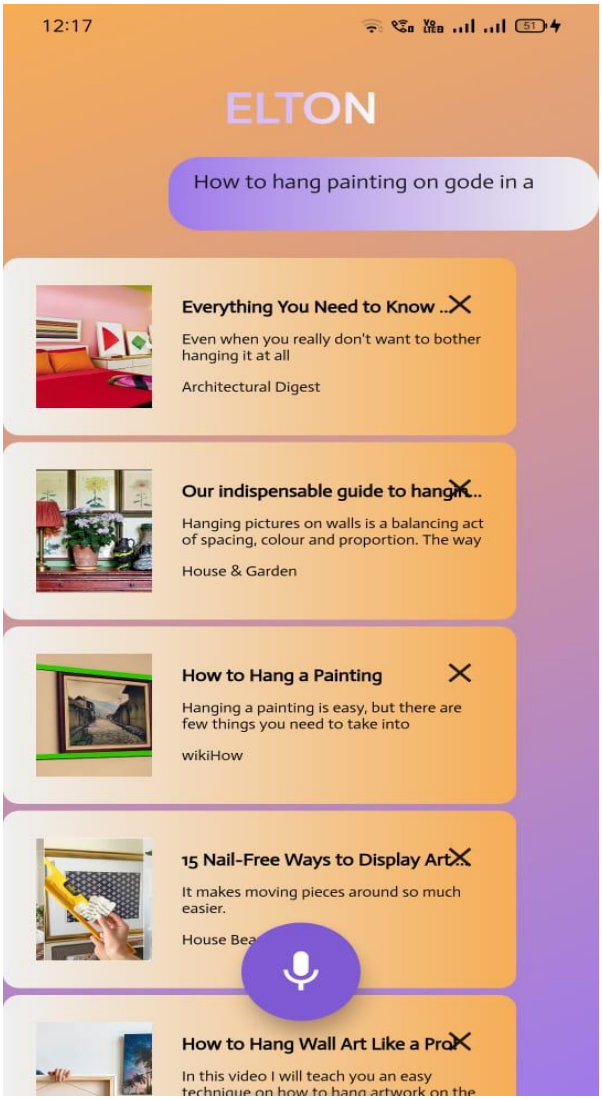


Fig 17 User Interface

4.2.5 Backend Implementation

The working of backend is as shown in the flow diagram below

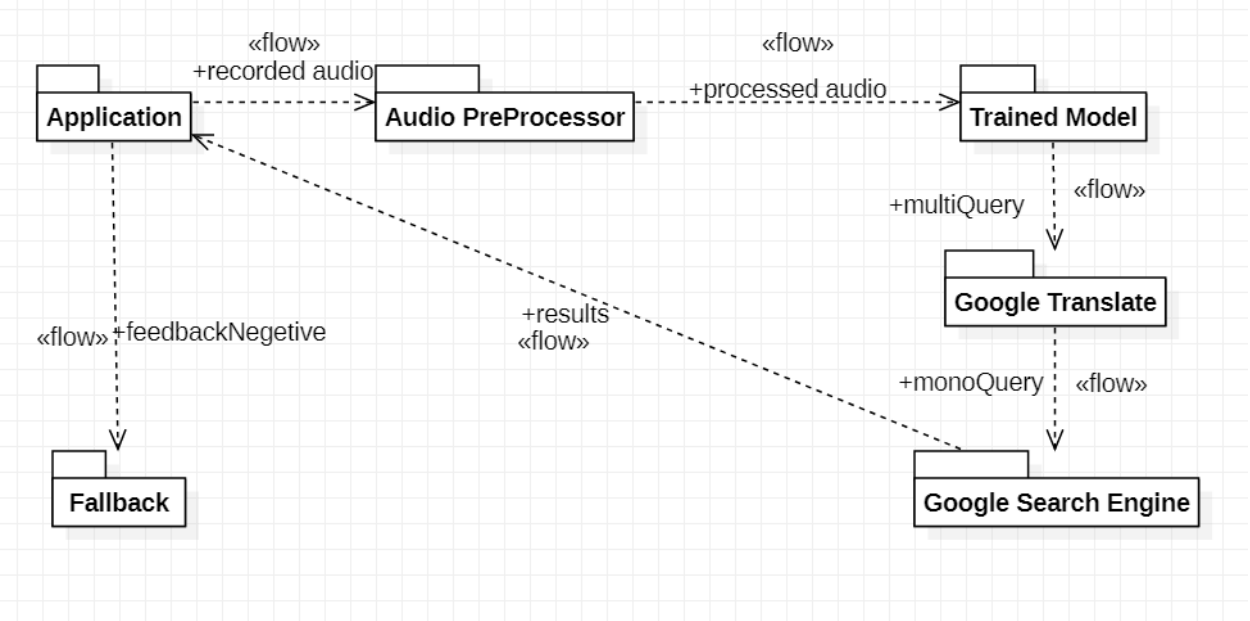


Fig 18 Backend implementation

Appendix A: Definitions, Acronyms and Abbreviations

API : An application programming interface is a connection between computers or between computer programs.

Firebase : Firebase is a platform developed by Google for creating mobile and web applications

Librosa : Librosa is a Python package for music and audio analysis.

Numpy : NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

WAV : Waveform Audio File Format is an audio file format standard, developed by IBM and Microsoft, for storing an audio bitstream on PCs.

MFCC : In sound processing, the mel-frequency cepstrum is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

Tensorflow : TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Keras : Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library

DNN : Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning.

NLTK : The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

POS : Parts of speech

RNN : A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior.

Appendix B: References

- Flutter documentation
- Firebase documentation
- Project Plan
- Cloud API documentation
- Capstone Review 1- ppt
- Survey of existing models
- Project Requirement Specifications
- Capstone Review 2 - ppt
- Project Plan

- High Level Document
- http://www.speech.cs.cmu.edu/tools/lmtool-new.html

Appendix C: Record of Change History

#	Date	Document Version No.	Change Description	Reason for Change
1.	23-09-21	1	Initial Document	
2.				
3.				

Appendix D: Traceability Matrix

Project Requirement Specification Reference Section No. and Name.	DESIGN / HLD Reference Section No. and Name.	LLD Reference Section No. Name
4.3Software Requirements	11 . Design Details	3.1 Application Model
1.Product Perspective	1.4 Product Perspective	1.1 Overview
4.3.1 Google Cloud API	2.2 Google API working and drawbacks	3.1 Application Model
2.3 General constraints, assumptions and dependencies	3.3 Constraints, Assumptions and Dependencies	2.General Constraints,Assumptions and Dependencies
3.1 User Interface	7. User Interface Diagram	4.2.4 Application UI
4. External Interface Requirements	3. Design Considerations	3. Design Description