ARTIFICIAL INTELLIGENCE
CHATBOT FOR COLLEGE INFORMATION

A Report submitted

By

Aishwarya Mishra (170102263)
Abhinav Pathak (170102264)
Yashika Mittal (170102252)

Under the Guidance of
Dr. Sarvesh Vishwakarma
Professor of
CSE Department

In partial fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING, DIT UNIVERSITY, DEHRADUN

(State Private University through State Legislature Act No. 10 of 2013 of Uttarakhand and approved by UGC)

Mussoorie Diversion Road, Dehradun, Uttarakhand - 248009, India.

DIT UNIVERSITY

IMAGINE ASPIRE ACHIEVE

DECLARATION

The Project entitled **“ARTIFICIAL INTELLIGENCE CHATBOT FOR COLLEGE INFORMATION” is being certified** for the award of the **Degree of B.TECH** in Computer Science, submitted to **DIT University, Dehradun, Uttarakhand, India,** is an bonafide record of work carried out by Aishwarya Mishra, Abhinav Pathak and Yashika Mittal under the guidance of Dr.Sarvesh Vishwakarma.

The content in this Project/Thesis/Dissertation has not been presented for the award of any other degree or diploma to any University/Institution. The results entitled in this project report have not been submitted to any other University or Institute for any Degree or Diploma.

Students Name & Signature:

This is to certify that the above statement made by the candidate is correct to the best of my /our knowledge.

Date: Signature(s) of the Supervisor (s)

ACKNOWLEDGEMENT

First and foremost, we express our gratitude to the Almighty and the Management of DIT University for providing all the facilities needed for this project. We express our gratitude to the Head of Department, Prof. Vishal Bharti, for his immense support and encouragement. We express our gratitude to our project coordinator, Mr. Sarvesh Vishwakarma, Department of computer science for his valuable suggestions and guidance. I would like to extend my gratefulness to my panel member co-ordinator Ms. Nishtha Rawat, Assistant Professor, Department of Computer Science and Engineering for her continuous support and overwhelming responses.

Last but not the least, I wish to thank all the staff of Computer Science and Engineering Department for their help and support.

ABSTRACT

Chatbots are conversational robots which have been made for the ease of technology and human beings.They are becoming very popular these days and have are being used by my many companies and websites.They are an application of ARTFICIAL INTELLIGENCE which have been developed to ease the labour of human beings.

Chatbots make a conversation just like human beings.They respond to our questions just as a human being.They seem to be very helpful for medical domain,college websites etc.

In this project,we have developed two chatbots: one with Dialog-flow and other using NLP and SEQUENCE-TO SEQUENCE model.

TABLE OF CONTENTS

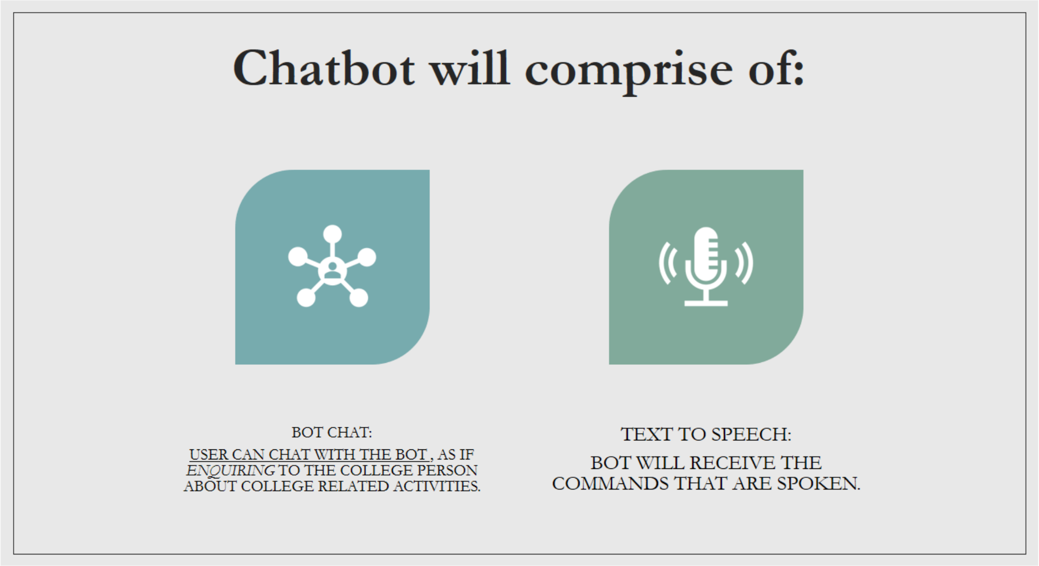
Title Page No.

CHAPTER 3 REFRENCES

1. INTRODUCTION

1.1. PURPOSE

1.1.1. Purpose of a Chatbot?



Chatbot will comprise of:

BOT CHAT:

USER CAN CHAT WITH THE BOT , AS IF *ENQUIRING* TO THE COLLEGE PERSON ABOUT COLLEGE RELATED ACTIVITIES.

TEXT TO SPEECH:

BOT WILL RECEIVE THE COMMANDS THAT ARE SPOKEN.

Chatbot is designed to grow a business more and for a better understanding of customer on industrial scale and it also helps to students in many ways such as student can have its own chatbot having his data about whatever he wants. There are many companies who are using chatbot as a user interface like facebook, college websites, telegram etc.

Also the chatbot will provide the information to customer which may be unknown to him and more qualities of the company.

1.1.2. Modules of Project

The project will be concluding 2 main modules:

- Using framework – Dialog Flow.
- Using NLP.

1.1.3. Chatbot will comprise of:

- ChatterBot : User can ask the questions or can chat with the chatbot regarding any topic he wants. For example : Our chatbot will provide all the information regarding the college. Whats the fee structure , the courses provided etc. Also chatbots can be made on real time data so that every new information can be included to train the data accordingly and it will response to it without being a fool.
- Speech Prototype : Bot will receive the commands that are spoken.

The bot will answer in both the ways he will speak along with the answer written in chat .

1.2.Working of Chatbot

There are two main funtions that are performed by chatbot :

- Analysis of user request
- Return the response as answer to question

Work Flow of Chatterbot

- **Analysis of user request** : the main task of chatbot is to understand the user what he wants to ask what the sentiments are related to question. If he didn't get that correctly we cannot consider it a perfect bot for use.

- **Return the response as answer to the question** : once the chatbot understood what a person wants to ask it find out the relevant response to it that can be produced as output on screen to show the user as its answer. There can be of three types on which chatterbot can produce the answer to:

1. The text or response predefined by user.

2. A text retrieved from a knowledge base that contains different answers

3. A contextualized piece of information based on data the user has provided

4. Data stored in database systems.

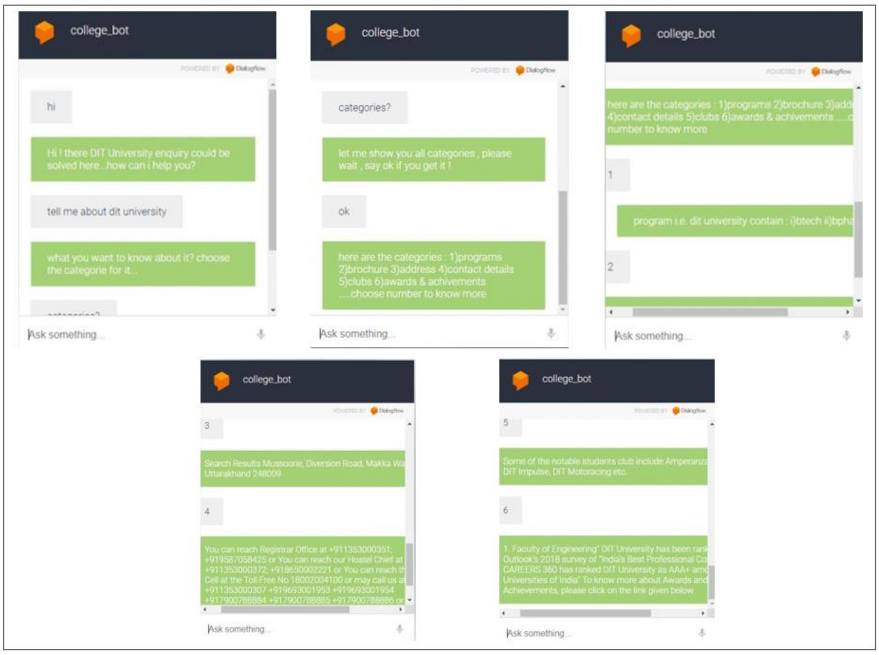
5. A disambiguating question that helps the Chabot to correctly understand the user's request.

1.2. Definition and overview

A Chabot can be defined as artificial intelligence (AI) software in which it is trained in such a way that can root out a conversation (or a chat) with a user through applications, messaging websites, and mobile apps or through the telephone.

This could be text based, spoken, non-verbal conversation. This Chabot will give answer to query about college related issues e.g.: (admission, subjects, streams, fee structure etc.).

Importance: A Chabot is also defined as one of the most advanced and perfect-promising way of interaction between person and machines. It seems as a face to face or person to person interaction. It is well defined and well explained in every term of it.



hi

Hi ! there DIT University enquiry could be solved here...how can i help you?

tell me about dit university

what you want to know about it? choose the categorie for it...

categories?

let me show you all categories , please wait , say ok if you get it !

ok

here are the categories : 1)programs 2)brochure 3)address 4)contact details 5)clubs 6)awards & achivements ....choose number to know more

1

program i.e. dit university contain : i)btech ii)bpha

2

3

Search Results Mussoorie, Diversion Road, Makka Wa Uttarakhand 248009

4

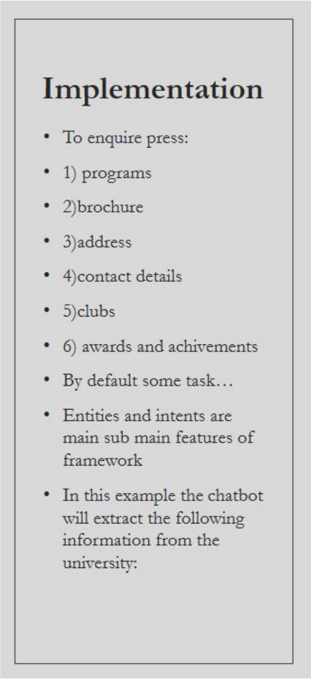
You can reach Registrar Office at +911353000351; +919587058425 or You can reach our Hostel Chief at +911353000372; +918650002221 or You can reach th Cell at the Toll Free No 18002004100 or may call us at +911353000307 +919693001953 +919693001954 +917900788884 +917900788885 +917900788886 or

5

Some of the notable students club include Amperanz DIT Impulse, DIT Motoracing etc.

6

1. Faculty of Engineering" DIT University has been rank Outlook's 2018 survey of "India's Best Professional Co CAREERS 360 has ranked DIT University as AAA+ amo Universities of India" To know more about Awards and Achievements, please click on the link given below.

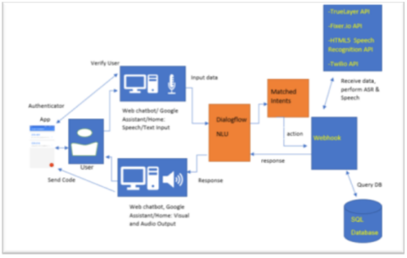


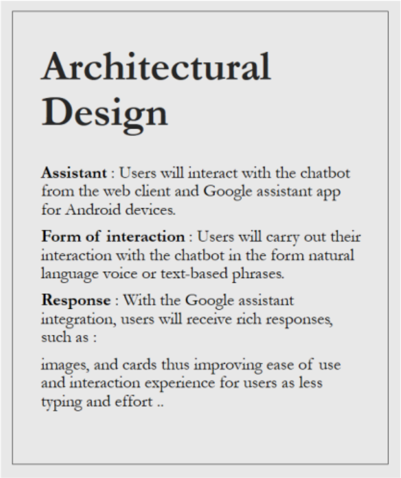
Implementation

- To enquire press:
- 1) programs
- 2)brochure
- 3)address
- 4)contact details
- 5)clubs
- 6) awards and achivements
- By default some task…
- Entities and intents are main sub main features of framework
- In this example the chatbot will extract the following information from the university:

2. DESCRIPTIONS

2.1. Architectural Design


-TrueLayer API
-Fixer.io API
-HTML5 Speech Recognition API
-Twilio API
Verify User
Input data
Authenticator App
Web chatbot/ Google Assistant/Home: Speech/Text Input
Matched Intents
Receive data, perform ASR & Speech
Dialogflow NLU
action
Webhook
User
response
Send Code
Response
Query DB
Web chatbot, Google Assistant/Home: Visual and Audio Output
SQL Database



Architectural Design

Assistant : Users will interact with the chatbot from the web client and Google assistant app for Android devices.

Form of interaction : Users will carry out their interaction with the chatbot in the form natural language voice or text-based phrases.

Response : With the Google assistant integration, users will receive rich responses, such as :

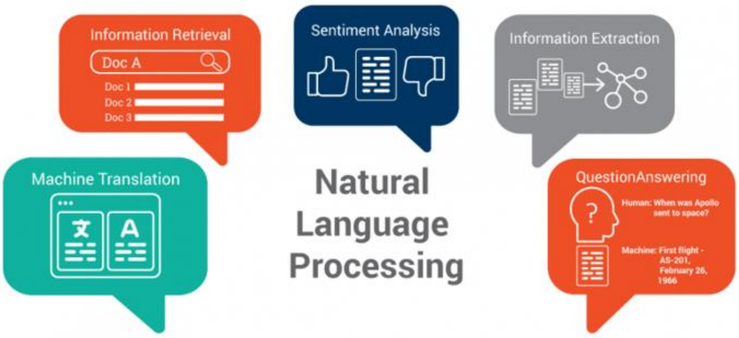
images, and cards thus improving ease of use and interaction experience for users as less typing and effort ..

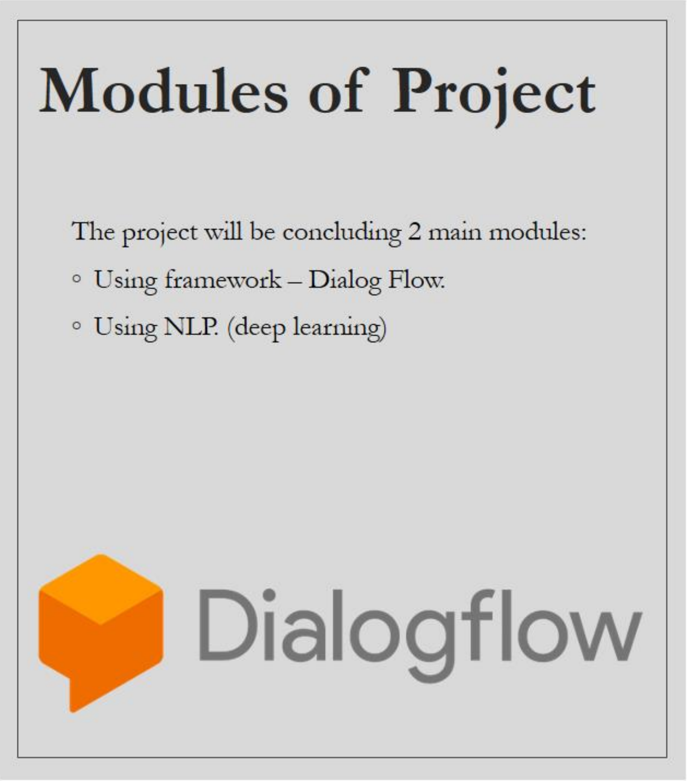
Person who is interacting with chatbot will interact or ask question with the chatbot from the web client. It is somewhat similar to google assistant that we found in our digital devices like mobile phones, laptops and everywhere at place where we operate the google itself. Users will ask the question from chatbot in the form any language can be said natural or text-based. With the Google assistant pre-defination, users will get mny responses, such as; images, and links to go through and interaction experience for users as less typing.

Project Modules:

Project is completed in two modules :

1) Implementation of chatbot through **Dialogflow**.
2) Implementation of chatbot through **NLP** (RNN – *Sequence to Sequence Model*)





Modules of Project

The project will be concluding 2 main modules:

- Using framework – Dialog Flow.
- Using NLP. (deep learning)

Dialogflow

2.2. Progress in work

2.2.1. Building of chatbot through dialogflow –

Dialog Flow: In this method we are going to follow the following steps which will lead in initiating the process. The steps are:

□ Intents will be made as:

1. Contexts

2. Training phases

3. Action and parameters

4. Responses

- Entities

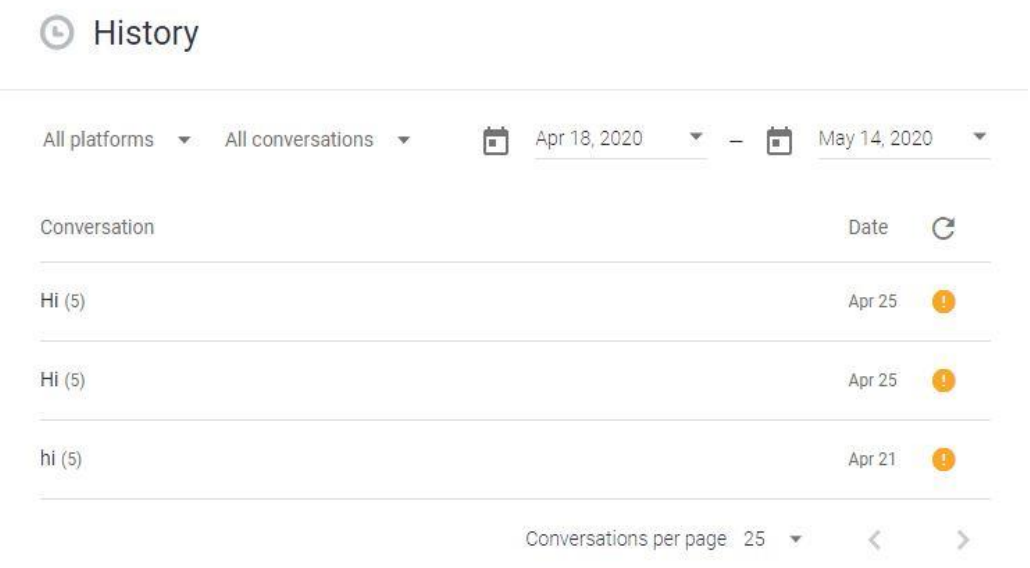
  It will contain all different types about what applicant want to enquire.

- Training

  It will make the bot learn about the statements which are asked frequently many times and answer default.

- History

  It will tell about the questions asked in past and if want to change the response or want to train can be done here.



History

All platforms ▾ All conversations ▾ Apr 18, 2020 ▾ – May 14, 2020 ▾

Conversation	Date
Hi (5)	Apr 25
Hi (5)	Apr 25
hi (5)	Apr 21

Conversations per page 25 ▾

- Fulfilment

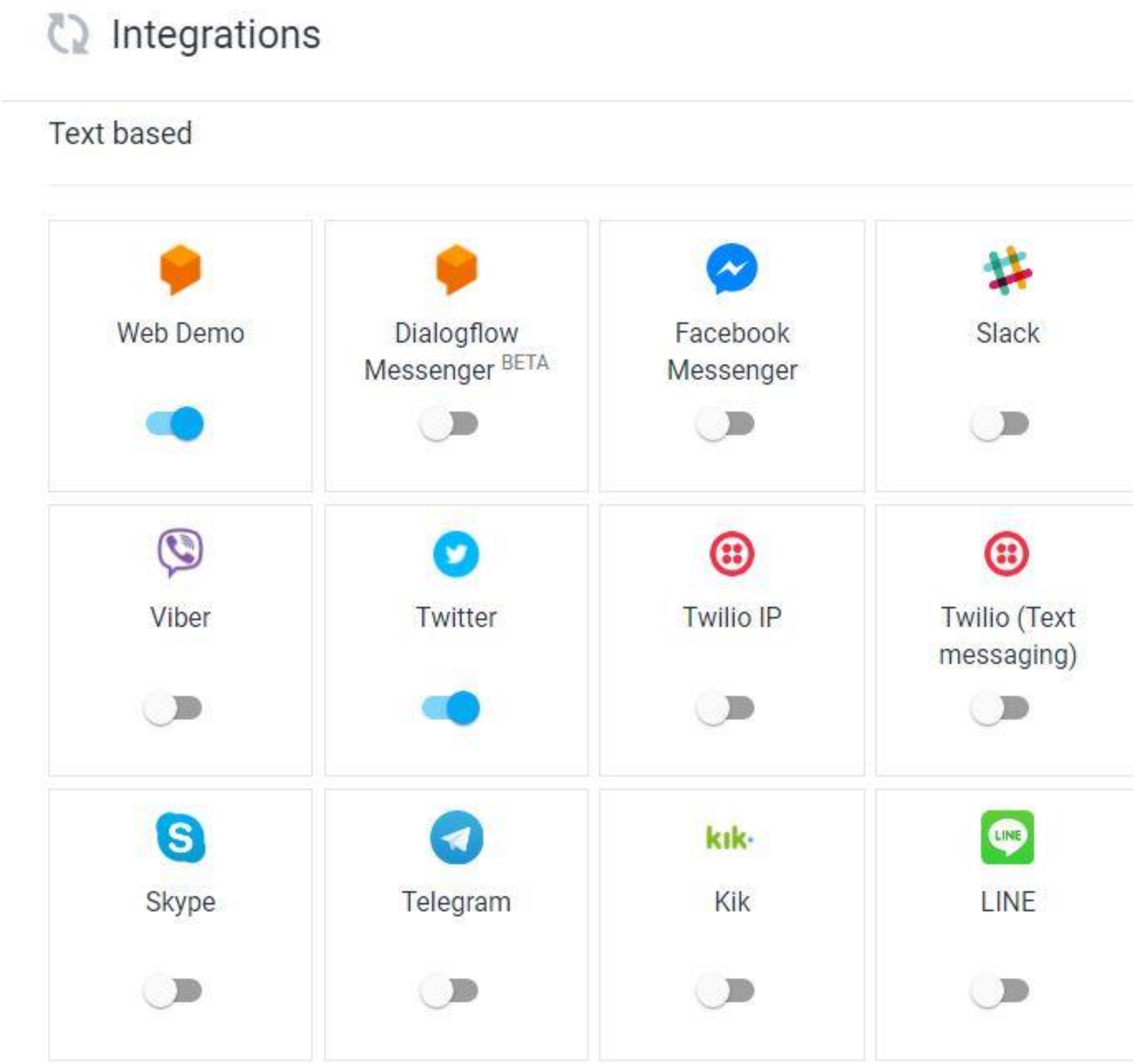
  The inline editor will contain the code of handling request by user and action taken by agent. It is part of training.

- Integrations

  It will build the assistant to reach the users through the required home page.

As we focused on interaction between user and bot so we should prepare the intent very smarty and carefully so that whenever there is need to define more answers or give more data to it , we should know in which part we have to do it. Intents are the basic steps of dialog-flow.

Once defined correctly the chatterbot cannot go out of content to give the answers as it is not trained on time database. Dialog is the collection of words, phrases, addresses, links and many more to report the output to user. The questions are general and normal that can be asked by the user. Another design part will contain fall-back intents in which if chatbot is unable to found the answer from the database given to it, it will return that he didn't get the question or he will come back later. Meanwhile the user at the back of it will train it on the input questions that are generally or mostly asked by the user so that there will be much less chances of error and strike to point what the user wants. The dialog also contain follow-up intents which gives the answer to what next question can be or if he understood the first response he should give some required given optional response to it. This provides a better way of interaction between chatterbot ans user.

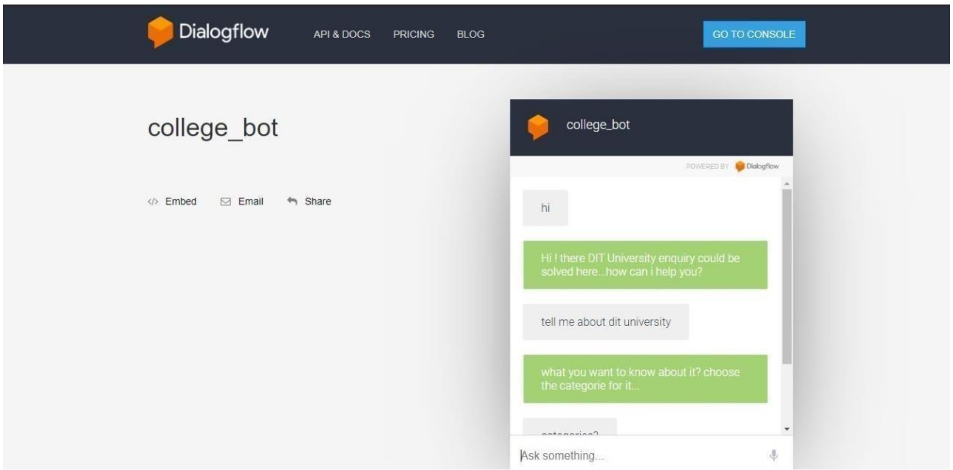


Integrations

Text based

- Web Demo ●
- Dialogflow Messenger BETA ○
- Facebook Messenger ○
- Slack ○
- Viber ○
- Twitter ●
- Twilio IP ○
- Twilio (Text messaging) ○
- Skype ○
- Telegram ○
- Kik ○
- LINE ○

In this example the chatbot will extract the following information from the university:
We have implemented it on webhook as it is free source .



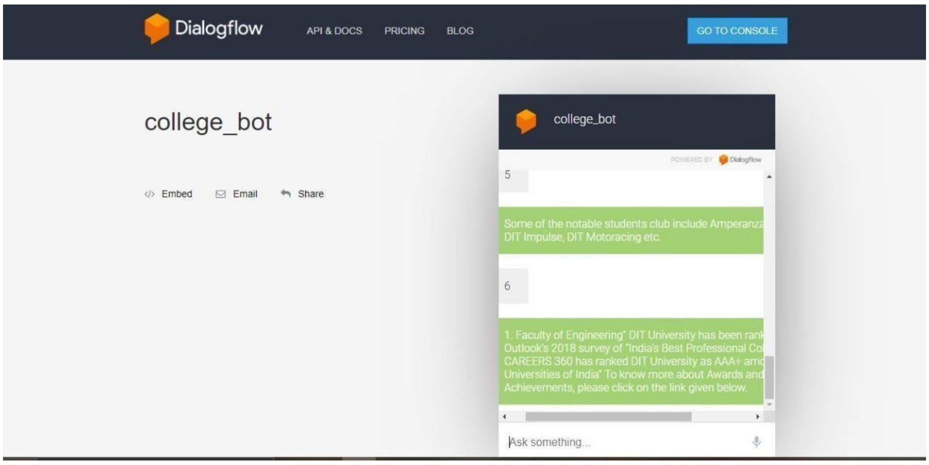
hi

Hi ! there DIT University enquiry could be solved here...how can i help you?

tell me about dit university

what you want to know about it? choose the categorie for it...

categories?

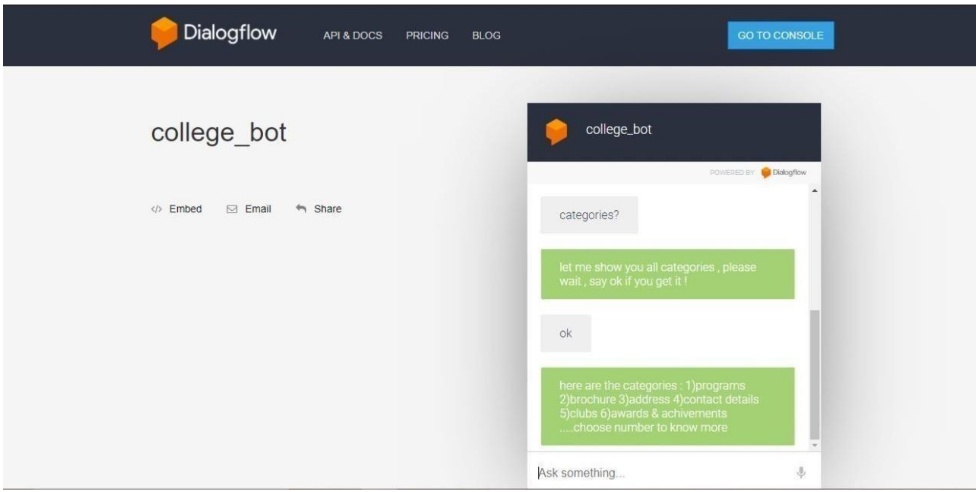


5

Some of the notable students club include Amperanza
DIT Impulse, DIT Motoracing etc.

6

1. Faculty of Engineering" DIT University has been rank
Outlook's 2018 survey of "India's Best Professional Co
CAREERS 360 has ranked DIT University as AAA+ amo
Universities of India" To know more about Awards and
Achievements, please click on the link given below.

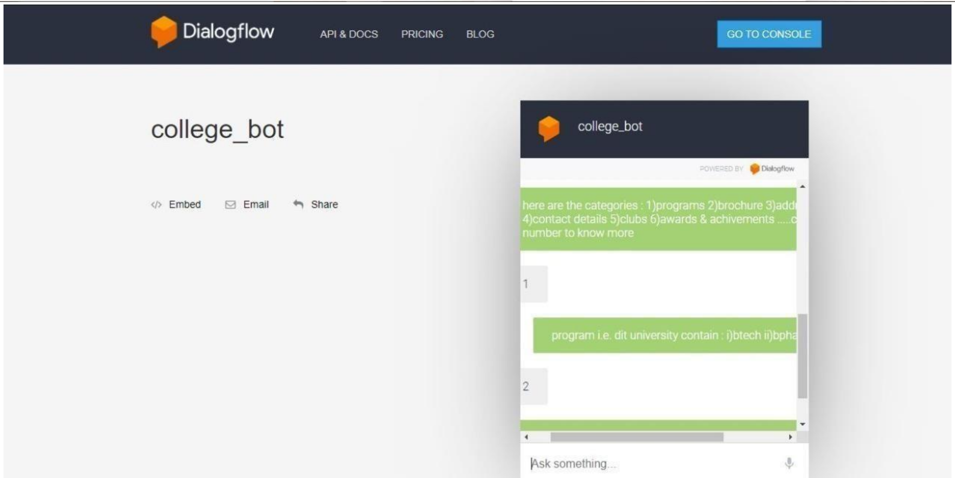


categories?

let me show you all categories , please wait , say ok if you get it !

ok

here are the categories : 1)programs 2)brochure 3)address 4)contact details 5)clubs 6)awards & achivements .....choose number to know more



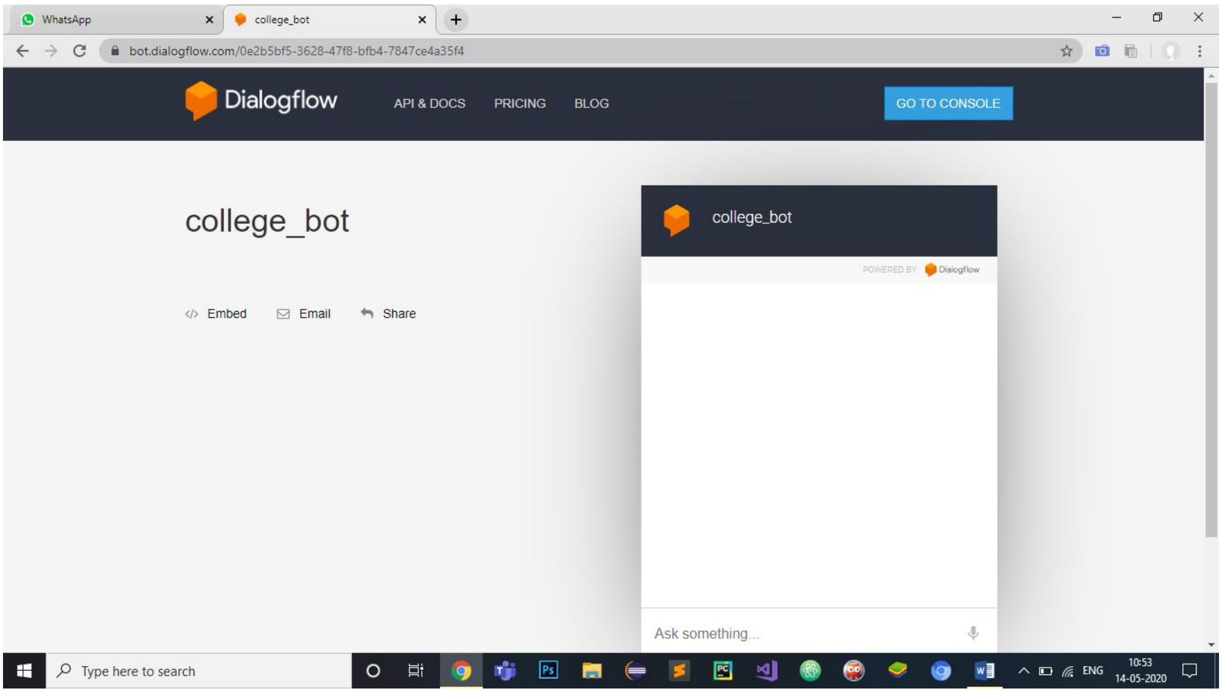
here are the categories : 1)programs 2)brochure 3)add 4)contact details 5)clubs 6)awards & achivements .....c number to know more

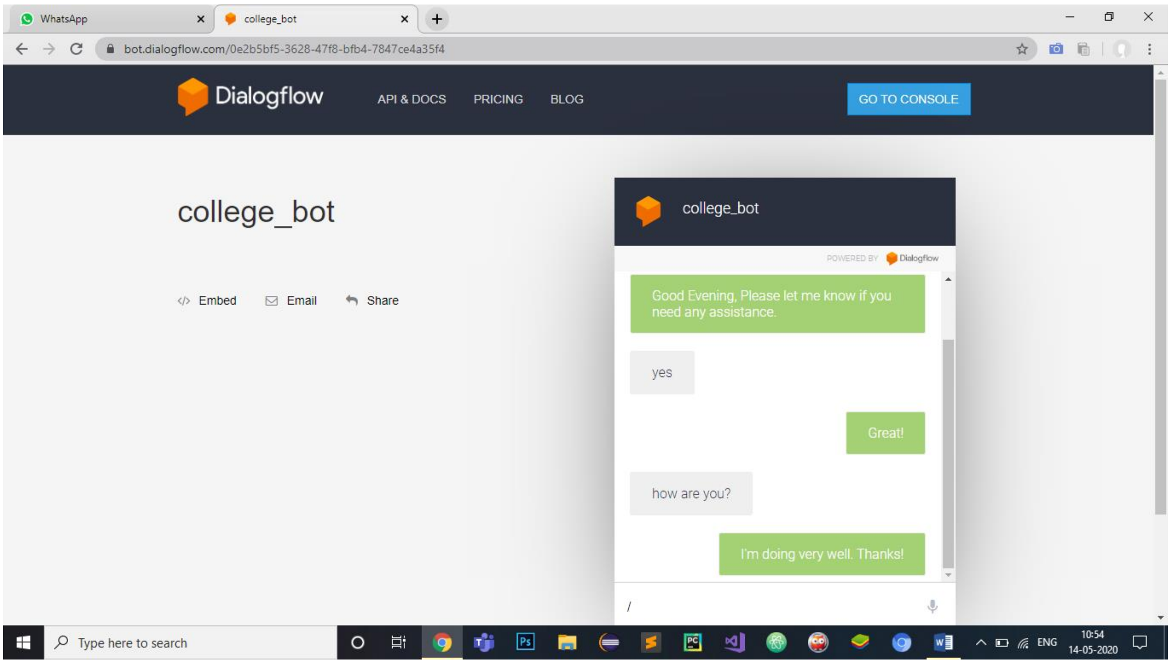
1

program i.e. dit university contain : i)btech ii)bpha

2

CHECK THE WORKING HERE:
https://bot.dialogflow.com/0e2b5bf5-3628-47f8-bfb4-7847ce4a35f4





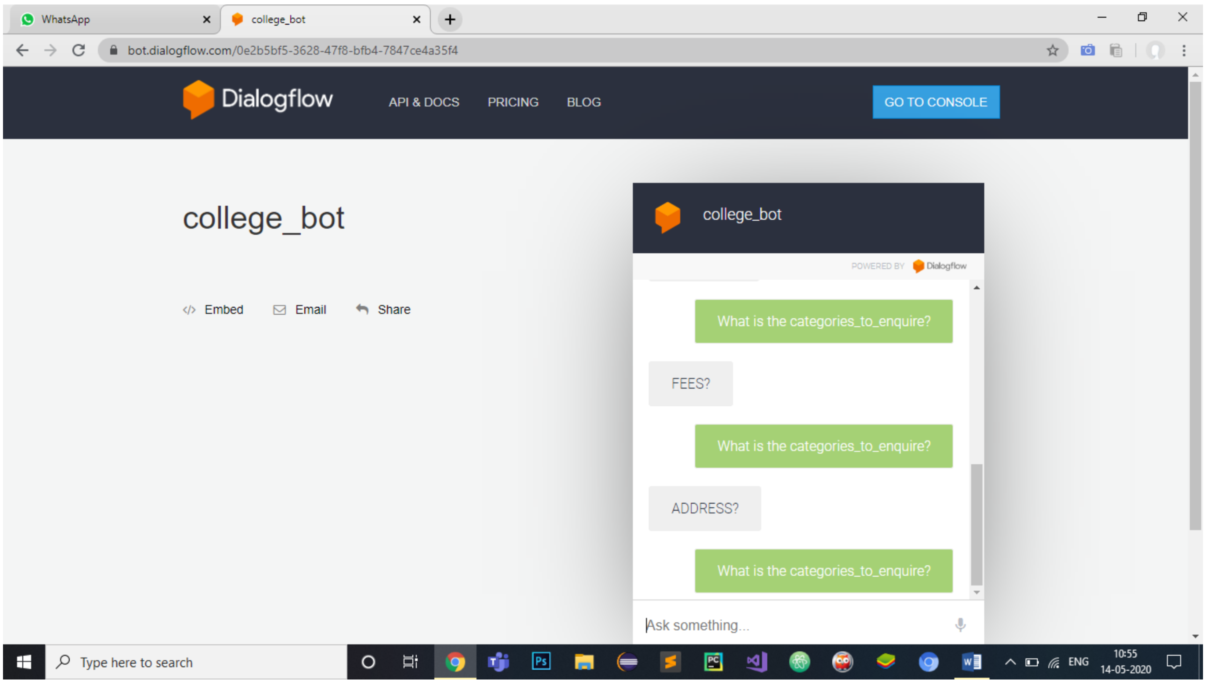
Good Evening, Please let me know if you need any assistance.

yes

Great!

how are you?

I'm doing very well. Thanks!



college_bot

college_bot

What is the categories_to_enquire?

FEES?

What is the categories_to_enquire?

ADDRESS?

What is the categories_to_enquire?

ANATOMY OF A CHATBOT

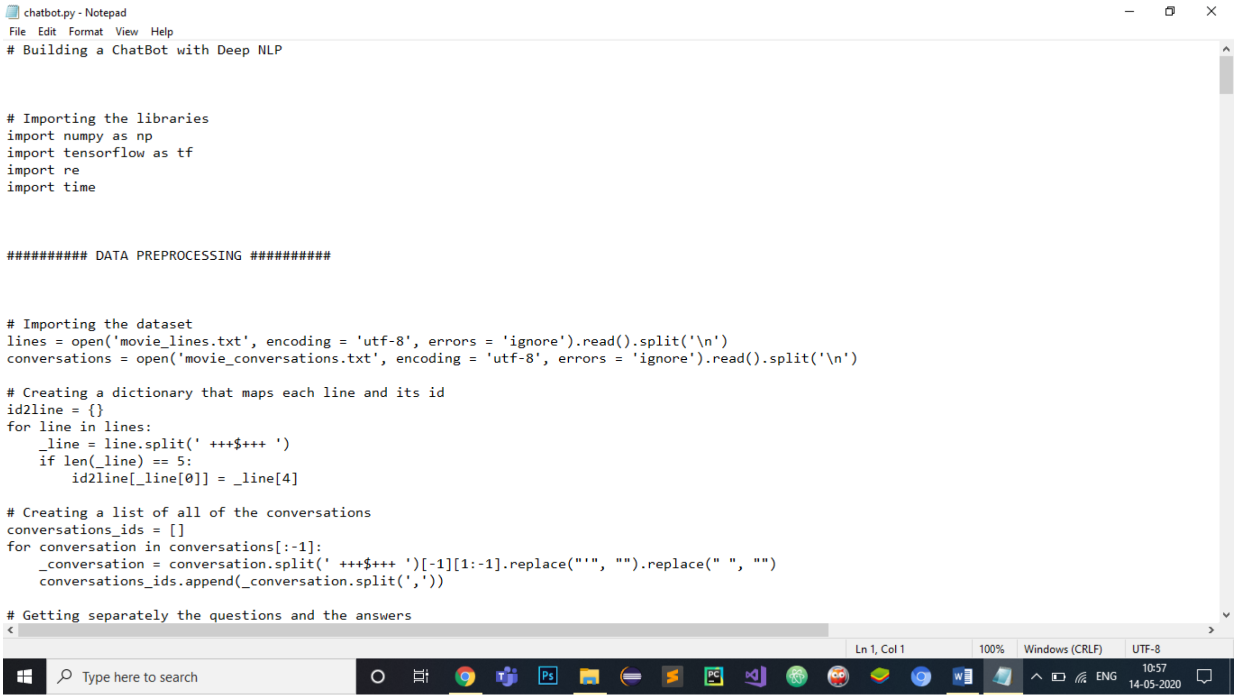
2.1.1. BUILDING OF CHATBOT WITH NLP –

Chat-bot have three phases to complete and get the output required. They are as follows:

1) Data Pre-processing
2) Sequence to Sequence Model
3) Data Training

DATA PRE-PROCESSING –

Data pre-processing involves the second module of the project. In this we undergo the process of developing the data. This module involves the use of various algorithms which help the bot to understand the dataset provided to it and helps in making the brain of the bot.



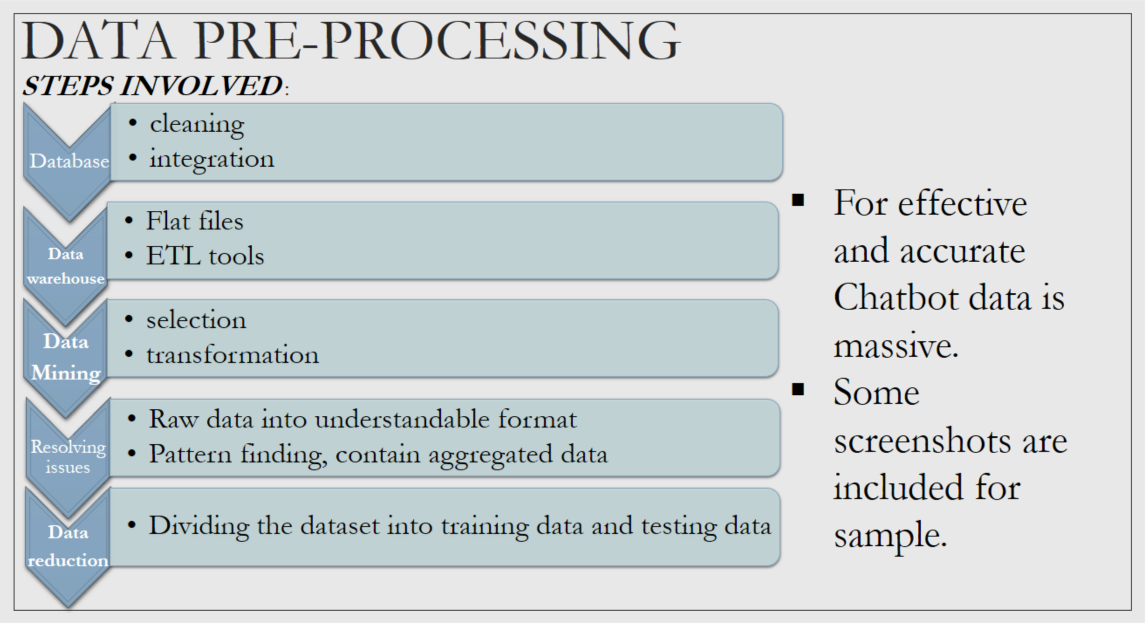
```
# Building a ChatBot with Deep NLP


# Importing the libraries
import numpy as np
import tensorflow as tf
import re
import time



########## DATA PREPROCESSING ##########



# Importing the dataset
lines = open('movie_lines.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')
conversations = open('movie_conversations.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')

# Creating a dictionary that maps each line and its id
id2line = {}
for line in lines:
    _line = line.split(' +++$+++ ')
    if len(_line) == 5:
        id2line[_line[0]] = _line[4]

# Creating a list of all of the conversations
conversations_ids = []
for conversation in conversations[:-1]:
    _conversation = conversation.split(' +++$+++ ')[-1][1:-1].replace("'", "").replace(" ", "")
    conversations_ids.append(_conversation.split(','))

# Getting separately the questions and the answers
```

DATASET COLLECTED:

To make a chatter bot effective and immensive so that it responds to all the queries, it need enormous tools for that purpose.

Preprocessing of data involves the following steps:

1) Import the libraries
2) Import the data set
3) Check for missing values
4) See the Categorical Values
5) Splitting the Dataset into training and test set
6) Feature scaling



DATA PRE-PROCESSING

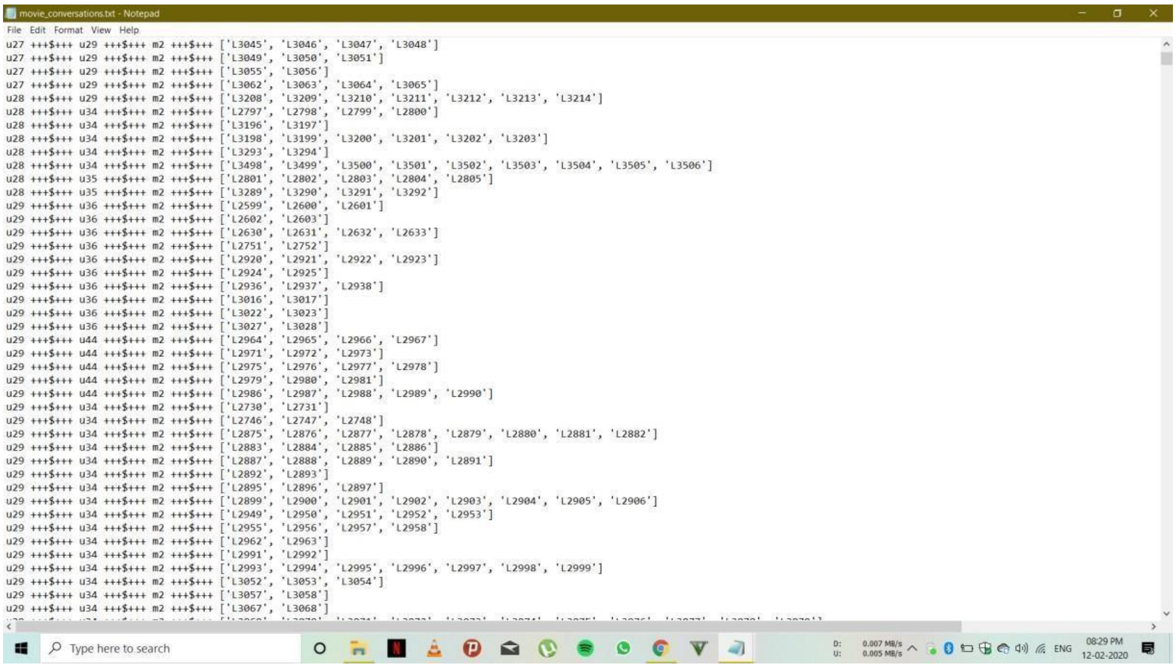
STEPS INVOLVED:

- cleaning
- integration
- Flat files
- ETL tools
- selection
- transformation
- Raw data into understandable format
- Pattern finding, contain aggregated data
- Dividing the dataset into training data and testing data

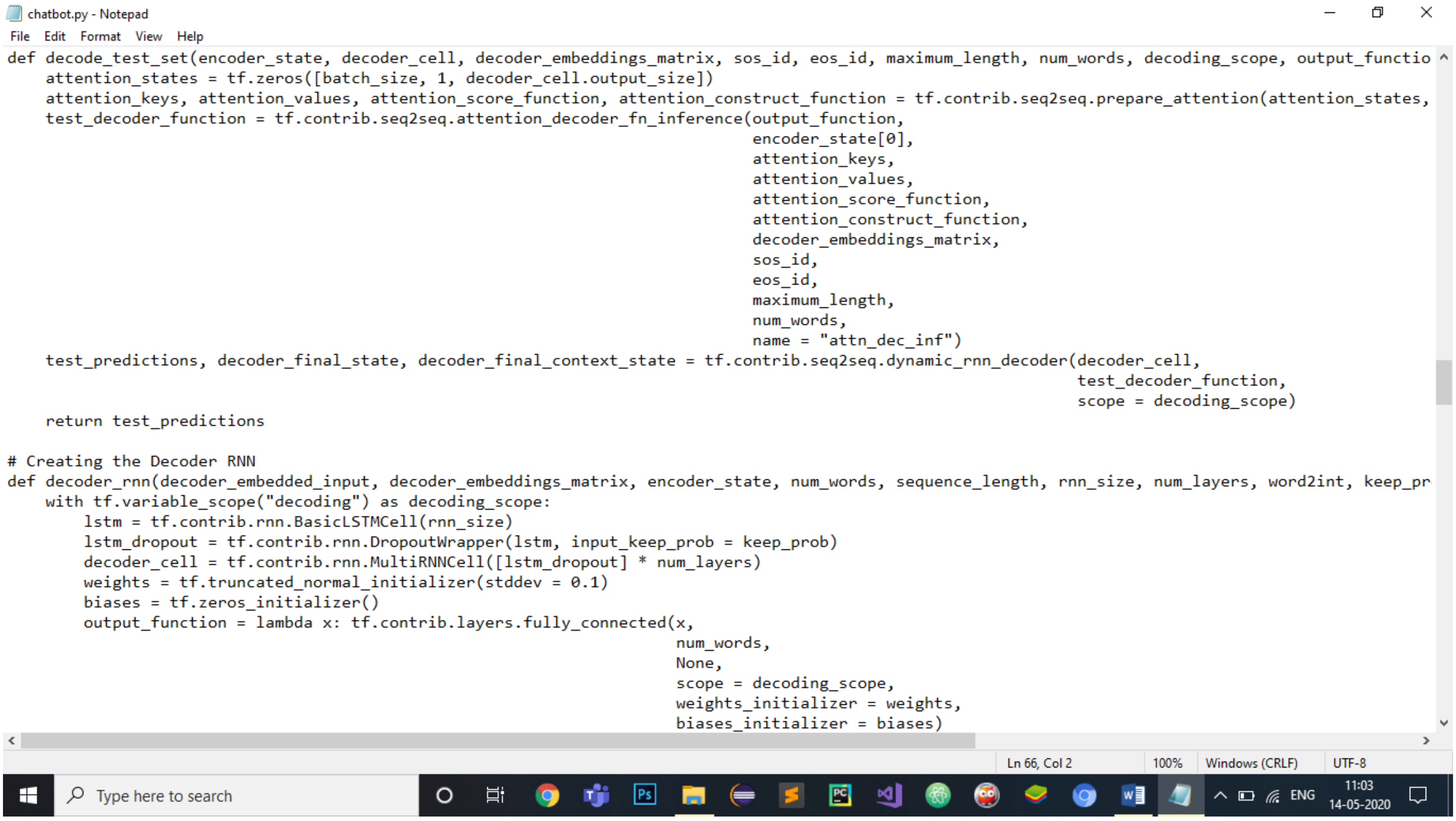
- For effective and accurate Chatbot data is massive.
- Some screenshots are included for sample.

COLLECTING DATASET:

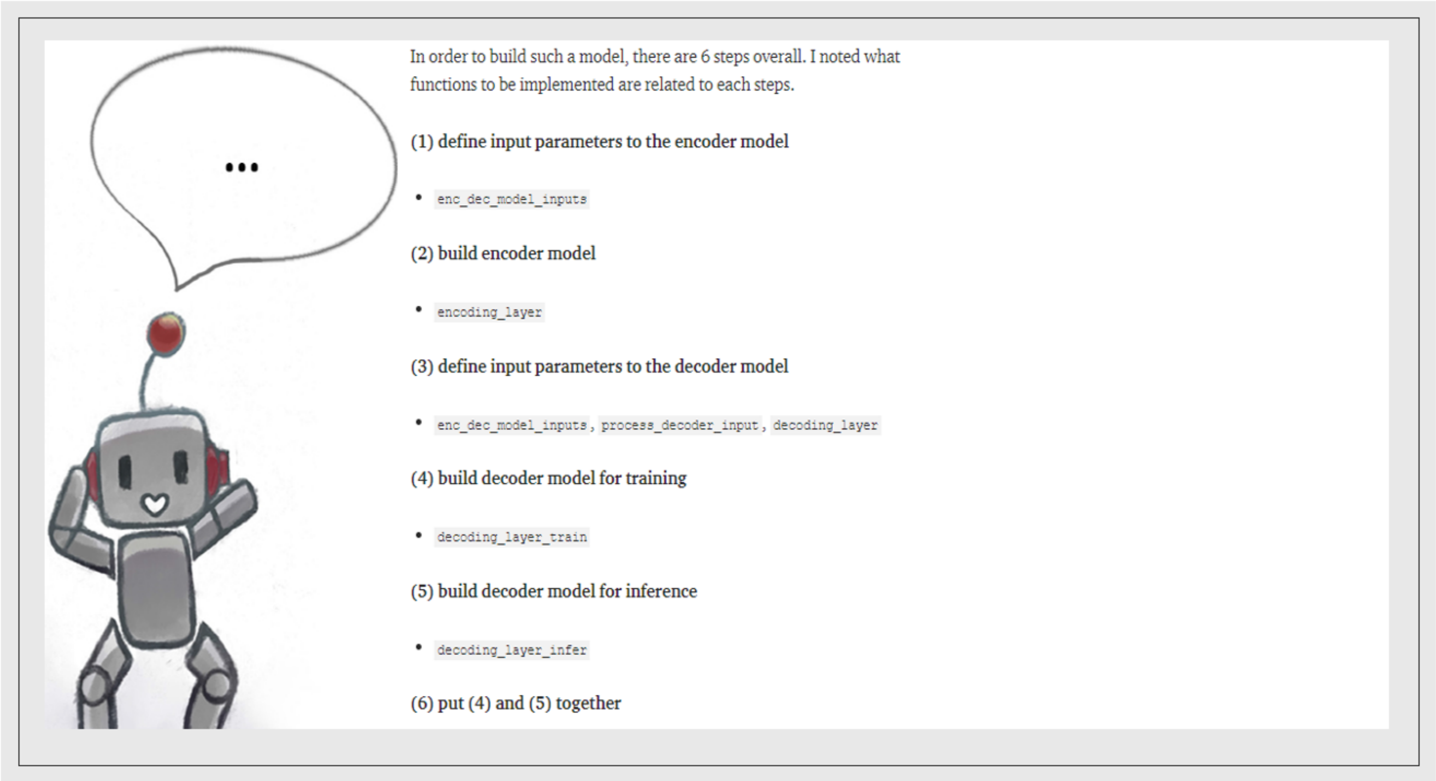
- Data is massive if we need a good built of chatbots.
- Some screenshots are included for sample.



```
u27 +++$+++ u29 +++$+++ m2 +++$+++ ['L3045', 'L3046', 'L3047', 'L3048']
u27 +++$+++ u29 +++$+++ m2 +++$+++ ['L3049', 'L3050', 'L3051']
u27 +++$+++ u29 +++$+++ m2 +++$+++ ['L3055', 'L3056']
u27 +++$+++ u29 +++$+++ m2 +++$+++ ['L3062', 'L3063', 'L3064', 'L3065']
u28 +++$+++ u29 +++$+++ m2 +++$+++ ['L3208', 'L3209', 'L3210', 'L3211', 'L3212', 'L3213', 'L3214']
u28 +++$+++ u34 +++$+++ m2 +++$+++ ['L2797', 'L2798', 'L2799', 'L2800']
u28 +++$+++ u34 +++$+++ m2 +++$+++ ['L3196', 'L3197']
u28 +++$+++ u34 +++$+++ m2 +++$+++ ['L3198', 'L3199', 'L3200', 'L3201', 'L3202', 'L3203']
u28 +++$+++ u34 +++$+++ m2 +++$+++ ['L3293', 'L3294']
u28 +++$+++ u34 +++$+++ m2 +++$+++ ['L3498', 'L3499', 'L3500', 'L3501', 'L3502', 'L3503', 'L3504', 'L3505', 'L3506']
u28 +++$+++ u35 +++$+++ m2 +++$+++ ['L2801', 'L2802', 'L2803', 'L2804', 'L2805']
u28 +++$+++ u35 +++$+++ m2 +++$+++ ['L3289', 'L3290', 'L3291', 'L3292']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L2599', 'L2600', 'L2601']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L2602', 'L2603']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L2630', 'L2631', 'L2632', 'L2633']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L2751', 'L2752']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L2920', 'L2921', 'L2922', 'L2923']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L2924', 'L2925']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L2936', 'L2937', 'L2938']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L3016', 'L3017']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L3022', 'L3023']
u29 +++$+++ u36 +++$+++ m2 +++$+++ ['L3027', 'L3028']
u29 +++$+++ u44 +++$+++ m2 +++$+++ ['L2964', 'L2965', 'L2966', 'L2967']
u29 +++$+++ u44 +++$+++ m2 +++$+++ ['L2971', 'L2972', 'L2973']
u29 +++$+++ u44 +++$+++ m2 +++$+++ ['L2975', 'L2976', 'L2977', 'L2978']
u29 +++$+++ u44 +++$+++ m2 +++$+++ ['L2979', 'L2980', 'L2981']
u29 +++$+++ u44 +++$+++ m2 +++$+++ ['L2986', 'L2987', 'L2988', 'L2989', 'L2990']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2730', 'L2731']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2746', 'L2747', 'L2748']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2875', 'L2876', 'L2877', 'L2878', 'L2879', 'L2880', 'L2881', 'L2882']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2883', 'L2884', 'L2885', 'L2886']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2887', 'L2888', 'L2889', 'L2890', 'L2891']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2892', 'L2893']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2895', 'L2896', 'L2897']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2899', 'L2900', 'L2901', 'L2902', 'L2903', 'L2904', 'L2905', 'L2906']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2949', 'L2950', 'L2951', 'L2952', 'L2953']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2955', 'L2956', 'L2957', 'L2958']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2962', 'L2963']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2991', 'L2992']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L2993', 'L2994', 'L2995', 'L2996', 'L2997', 'L2998', 'L2999']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L3052', 'L3053', 'L3054']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L3057', 'L3058']
u29 +++$+++ u34 +++$+++ m2 +++$+++ ['L3067', 'L3068']
```



```
def decode_test_set(encoder_state, decoder_cell, decoder_embeddings_matrix, sos_id, eos_id, maximum_length, num_words, decoding_scope, output_functio
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function = tf.contrib.seq2seq.prepare_attention(attention_states,
    test_decoder_function = tf.contrib.seq2seq.attention_decoder_fn_inference(output_function,
                                                                              encoder_state[0],
                                                                              attention_keys,
                                                                              attention_values,
                                                                              attention_score_function,
                                                                              attention_construct_function,
                                                                              decoder_embeddings_matrix,
                                                                              sos_id,
                                                                              eos_id,
                                                                              maximum_length,
                                                                              num_words,
                                                                              name = "attn_dec_inf")
    test_predictions, decoder_final_state, decoder_final_context_state = tf.contrib.seq2seq.dynamic_rnn_decoder(decoder_cell,
                                                                                                                test_decoder_function,
                                                                                                                scope = decoding_scope)
    return test_predictions

# Creating the Decoder RNN
def decoder_rnn(decoder_embedded_input, decoder_embeddings_matrix, encoder_state, num_words, sequence_length, rnn_size, num_layers, word2int, keep_pr
    with tf.variable_scope("decoding") as decoding_scope:
        lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
        lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
        decoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
        weights = tf.truncated_normal_initializer(stddev = 0.1)
        biases = tf.zeros_initializer()
        output_function = lambda x: tf.contrib.layers.fully_connected(x,
                                                                      num_words,
                                                                      None,
                                                                      scope = decoding_scope,
                                                                      weights_initializer = weights,
                                                                      biases_initializer = biases)
```



In order to build such a model, there are 6 steps overall. I noted what functions to be implemented are related to each steps.

(1) define input parameters to the encoder model

- `enc_dec_model_inputs`

(2) build encoder model

- `encoding_layer`

(3) define input parameters to the decoder model

- `enc_dec_model_inputs`, `process_decoder_input`, `decoding_layer`

(4) build decoder model for training

- `decoding_layer_train`

(5) build decoder model for inference

- `decoding_layer_infer`

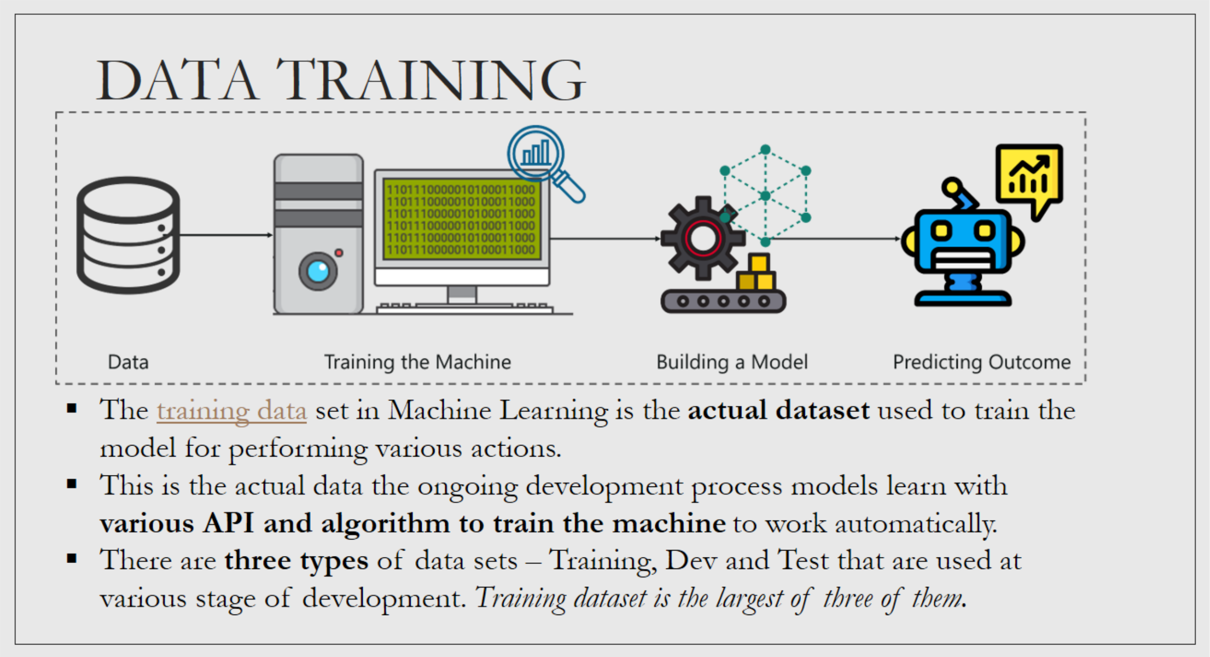
(6) put (4) and (5) together

DATA TRAINING –

The third phase of the project includes the data training.It is one of the most time-taking phase of all the modules.

The data training involves the development of the bot using the content of data that we have already given.This includes the working of APIs that help in learning from previous inputs and outputs.

This is the most enhancement based phase as it can take months to develop on larger basis.



DATA TRAINING

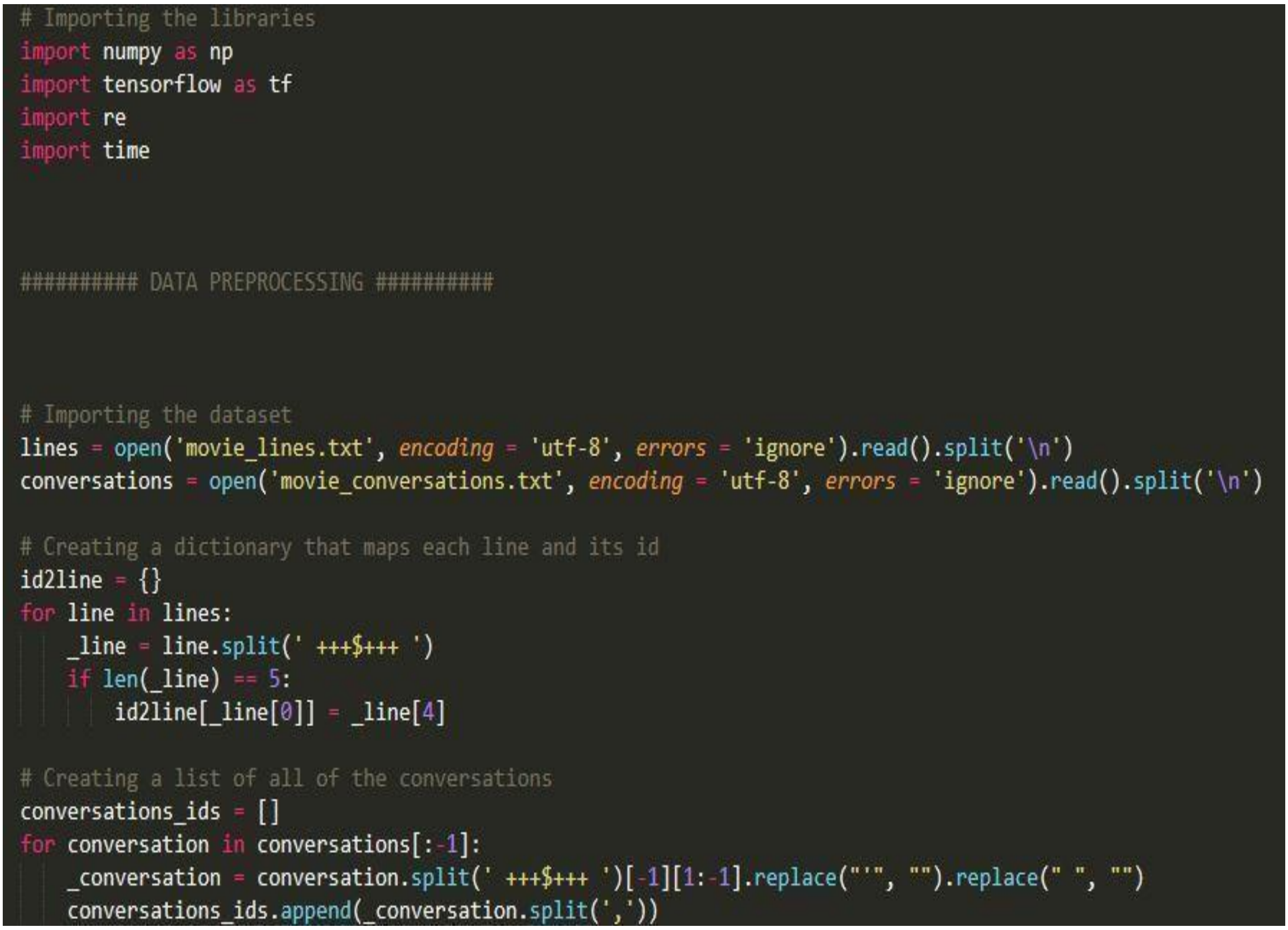
Data → Training the Machine → Building a Model → Predicting Outcome

- The training data set in Machine Learning is the **actual dataset** used to train the model for performing various actions.
- This is the actual data the ongoing development process models learn with **various API and algorithm to train the machine** to work automatically.
- There are **three types** of data sets – Training, Dev and Test that are used at various stage of development. *Training dataset is the largest of three of them.*

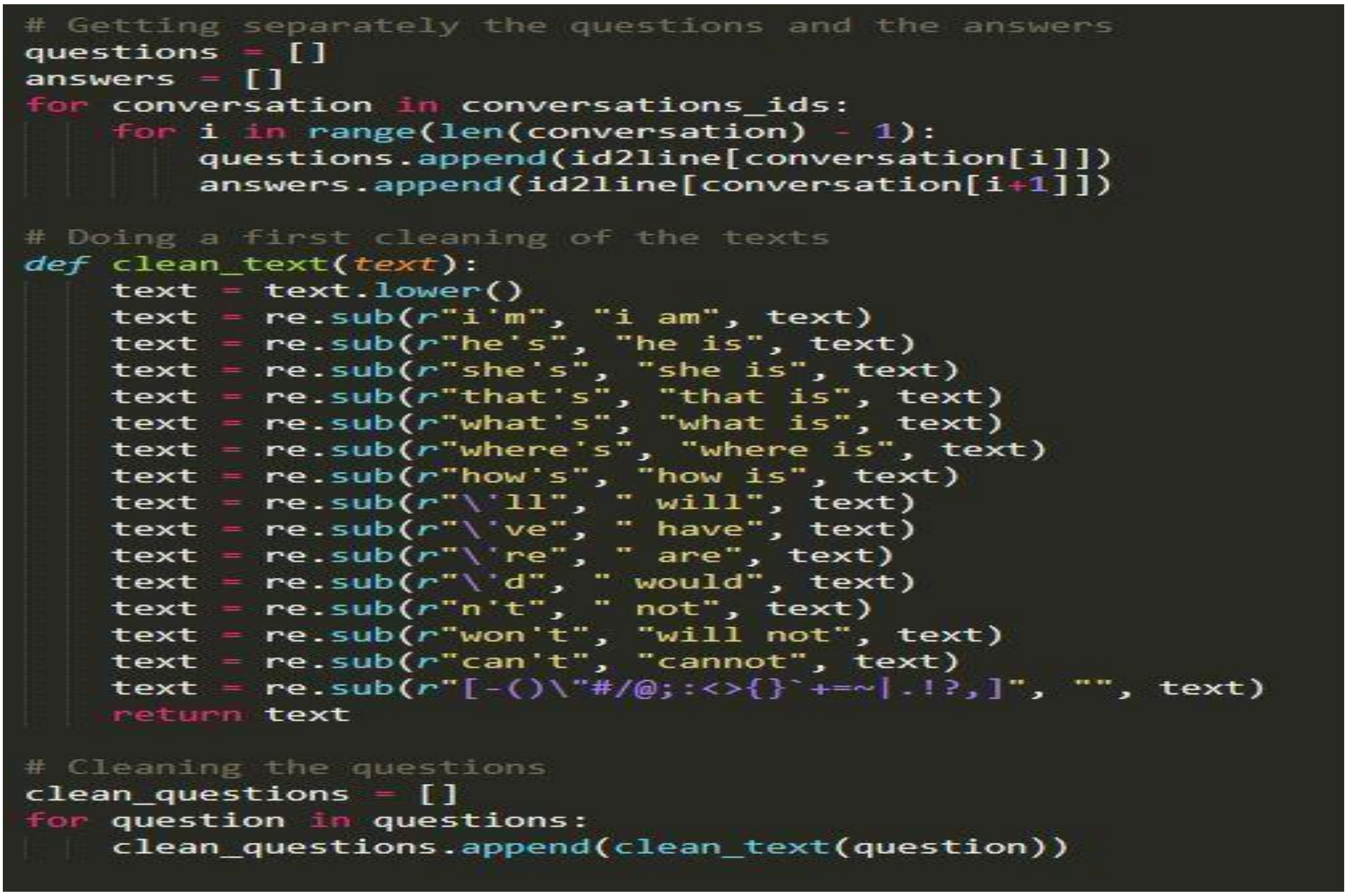
2.2. Feasilibilty and Methodology

The methodology of the project can be justified by the following code:

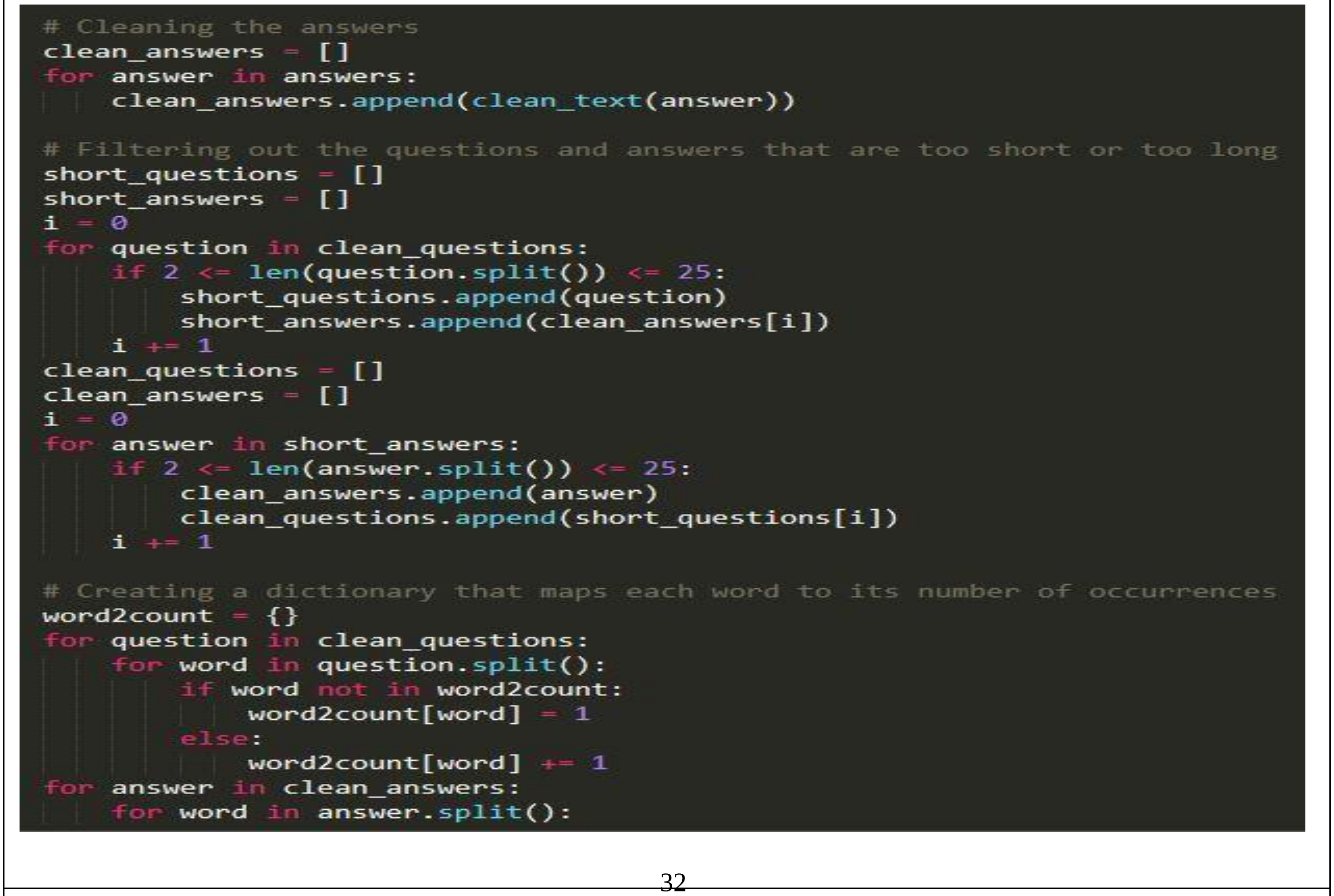
CODE FOR CHATBOT:



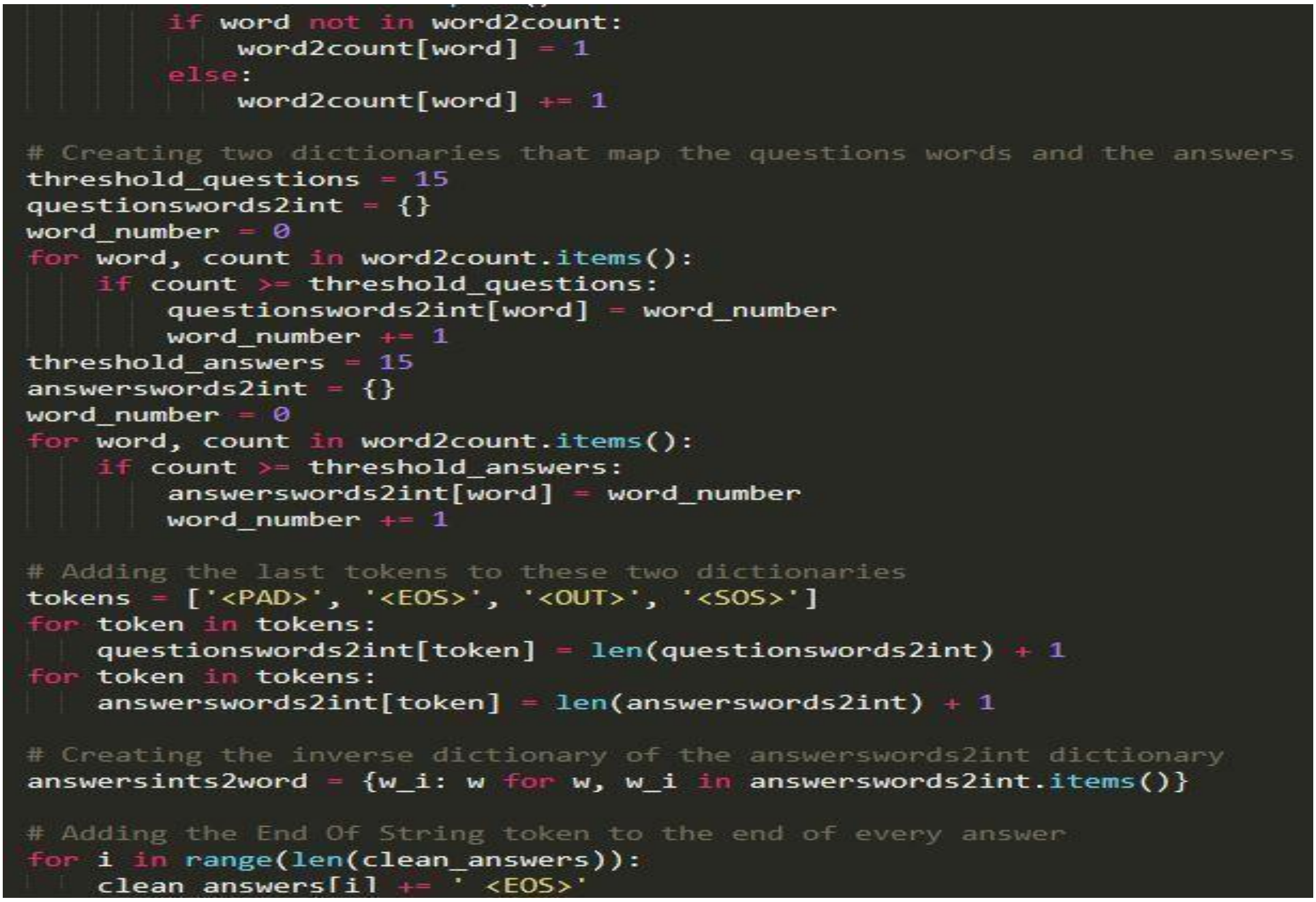
```python
# Importing the libraries
import numpy as np
import tensorflow as tf
import re
import time



########## DATA PREPROCESSING ##########



# Importing the dataset
lines = open('movie_lines.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')
conversations = open('movie_conversations.txt', encoding = 'utf-8', errors = 'ignore').read().split('\n')

# Creating a dictionary that maps each line and its id
id2line = {}
for line in lines:
    _line = line.split(' +++$+++ ')
    if len(_line) == 5:
        id2line[_line[0]] = _line[4]

# Creating a list of all of the conversations
conversations_ids = []
for conversation in conversations[:-1]:
    _conversation = conversation.split(' +++$+++ ')[-1][1:-1].replace("'", "").replace(" ", "")
    conversations_ids.append(_conversation.split(','))
```



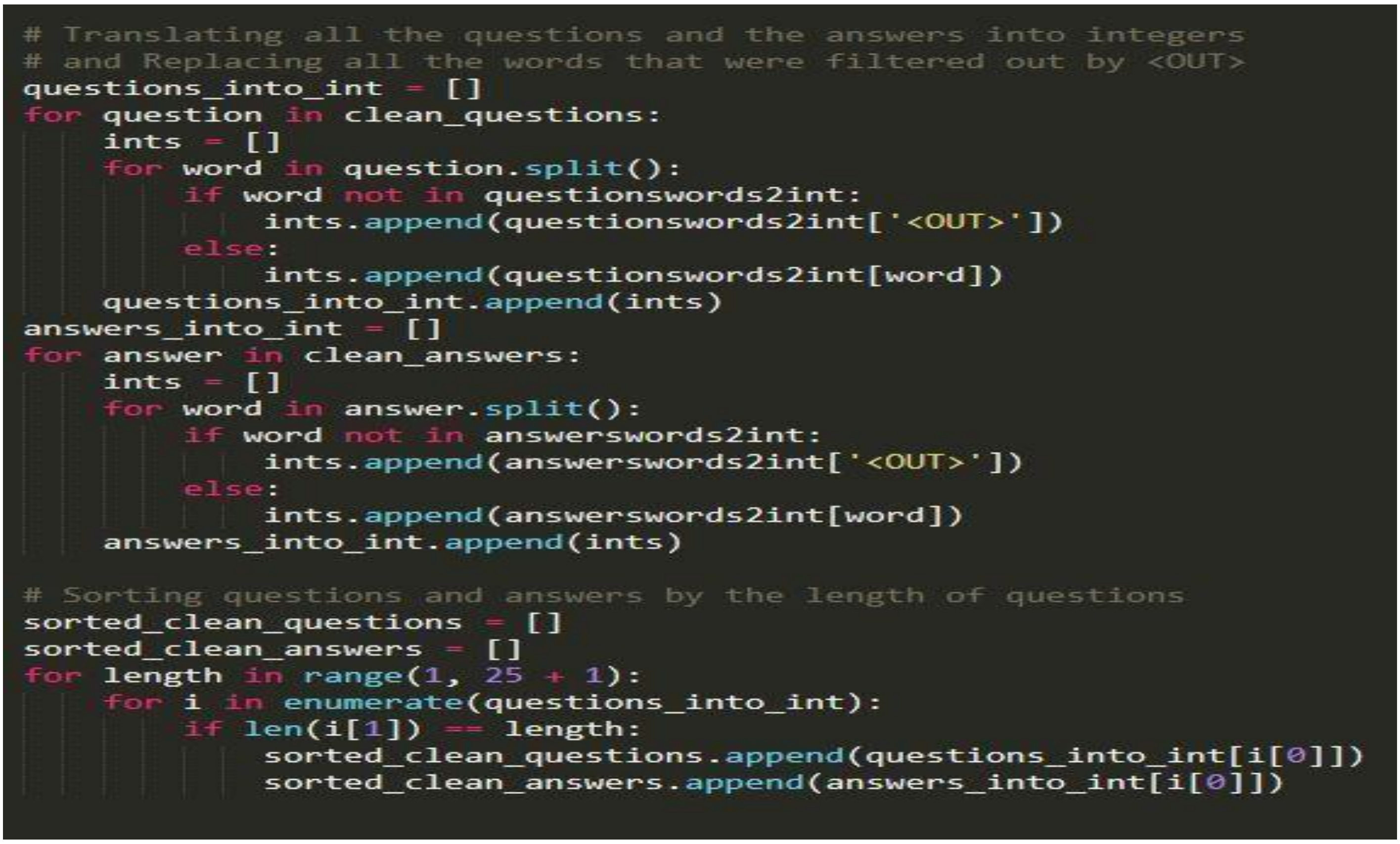
```python
# Getting separately the questions and the answers
questions = []
answers = []
for conversation in conversations_ids:
    for i in range(len(conversation) - 1):
        questions.append(id2line[conversation[i]])
        answers.append(id2line[conversation[i+1]])

# Doing a first cleaning of the texts
def clean_text(text):
    text = text.lower()
    text = re.sub(r"i'm", "i am", text)
    text = re.sub(r"he's", "he is", text)
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"what's", "what is", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"how's", "how is", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"n't", " not", text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "cannot", text)
    text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", text)
    return text

# Cleaning the questions
clean_questions = []
for question in questions:
    clean_questions.append(clean_text(question))
```



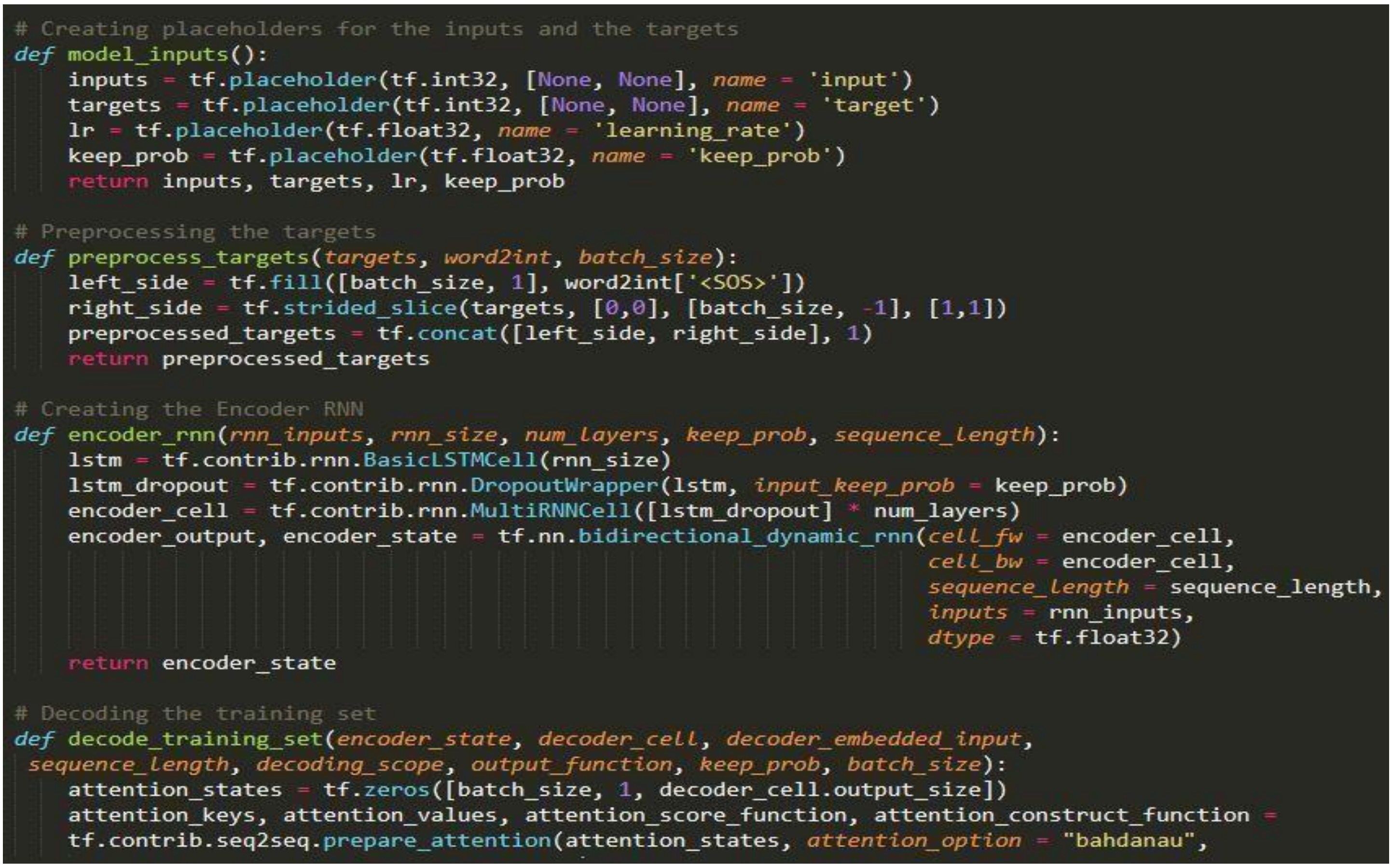
```python
# Cleaning the answers
clean_answers = []
for answer in answers:
    clean_answers.append(clean_text(answer))

# Filtering out the questions and answers that are too short or too long
short_questions = []
short_answers = []
i = 0
for question in clean_questions:
    if 2 <= len(question.split()) <= 25:
        short_questions.append(question)
        short_answers.append(clean_answers[i])
    i += 1
clean_questions = []
clean_answers = []
i = 0
for answer in short_answers:
    if 2 <= len(answer.split()) <= 25:
        clean_answers.append(answer)
        clean_questions.append(short_questions[i])
    i += 1

# Creating a dictionary that maps each word to its number of occurrences
word2count = {}
for question in clean_questions:
    for word in question.split():
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1
for answer in clean_answers:
    for word in answer.split():
```



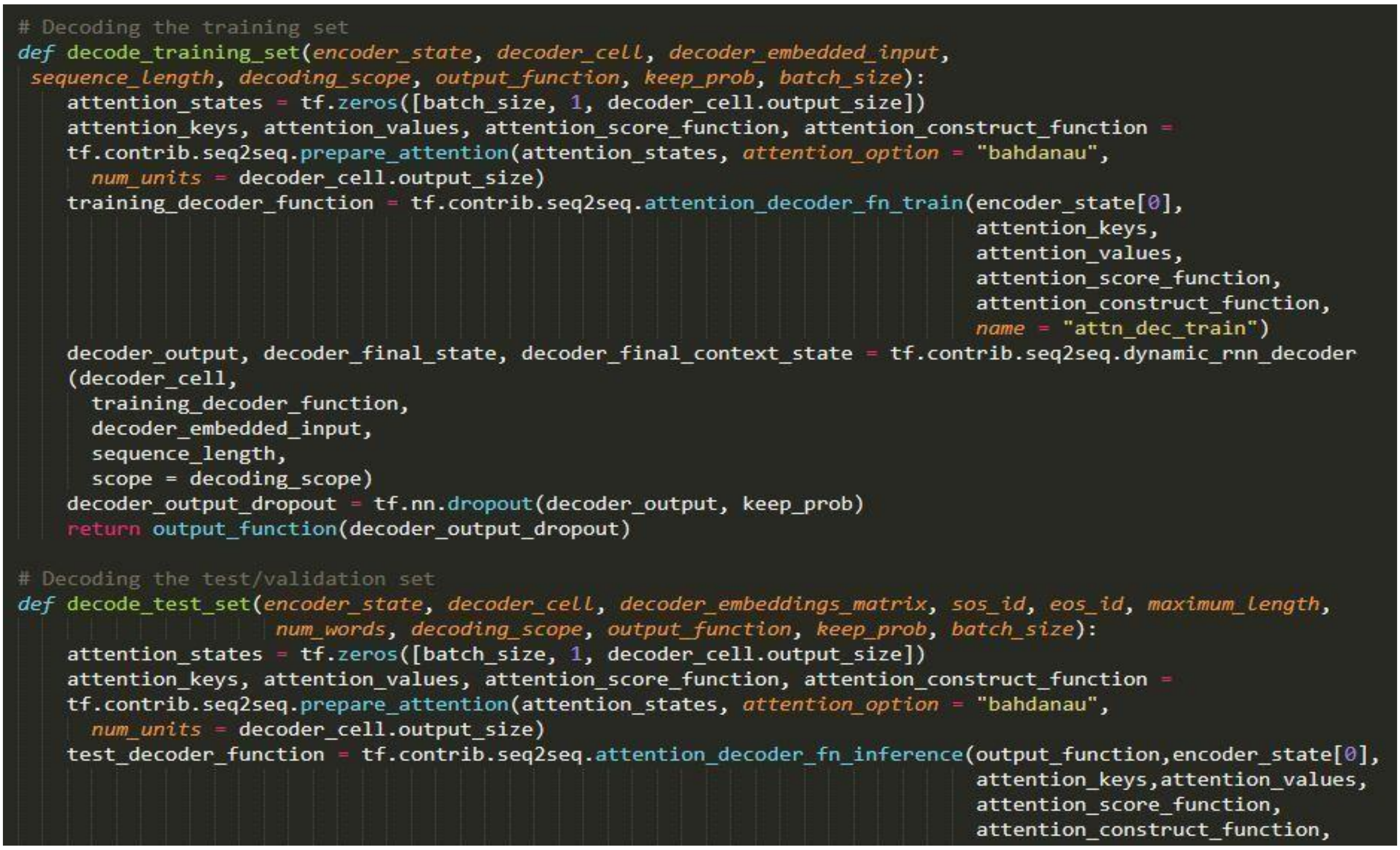
```
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1

# Creating two dictionaries that map the questions words and the answers
threshold_questions = 15
questionswords2int = {}
word_number = 0
for word, count in word2count.items():
    if count >= threshold_questions:
        questionswords2int[word] = word_number
        word_number += 1
threshold_answers = 15
answerswords2int = {}
word_number = 0
for word, count in word2count.items():
    if count >= threshold_answers:
        answerswords2int[word] = word_number
        word_number += 1

# Adding the last tokens to these two dictionaries
tokens = ['<PAD>', '<EOS>', '<OUT>', '<SOS>']
for token in tokens:
    questionswords2int[token] = len(questionswords2int) + 1
for token in tokens:
    answerswords2int[token] = len(answerswords2int) + 1

# Creating the inverse dictionary of the answerswords2int dictionary
answersints2word = {w_i: w for w, w_i in answerswords2int.items()}

# Adding the End Of String token to the end of every answer
for i in range(len(clean_answers)):
    clean_answers[i] += ' <EOS>'
```



```
# Translating all the questions and the answers into integers
# and Replacing all the words that were filtered out by <OUT>
questions_into_int = []
for question in clean_questions:
    ints = []
    for word in question.split():
        if word not in questionswords2int:
            ints.append(questionswords2int['<OUT>'])
        else:
            ints.append(questionswords2int[word])
    questions_into_int.append(ints)
answers_into_int = []
for answer in clean_answers:
    ints = []
    for word in answer.split():
        if word not in answerswords2int:
            ints.append(answerswords2int['<OUT>'])
        else:
            ints.append(answerswords2int[word])
    answers_into_int.append(ints)

# Sorting questions and answers by the length of questions
sorted_clean_questions = []
sorted_clean_answers = []
for length in range(1, 25 + 1):
    for i in enumerate(questions_into_int):
        if len(i[1]) == length:
            sorted_clean_questions.append(questions_into_int[i[0]])
            sorted_clean_answers.append(answers_into_int[i[0]])
```

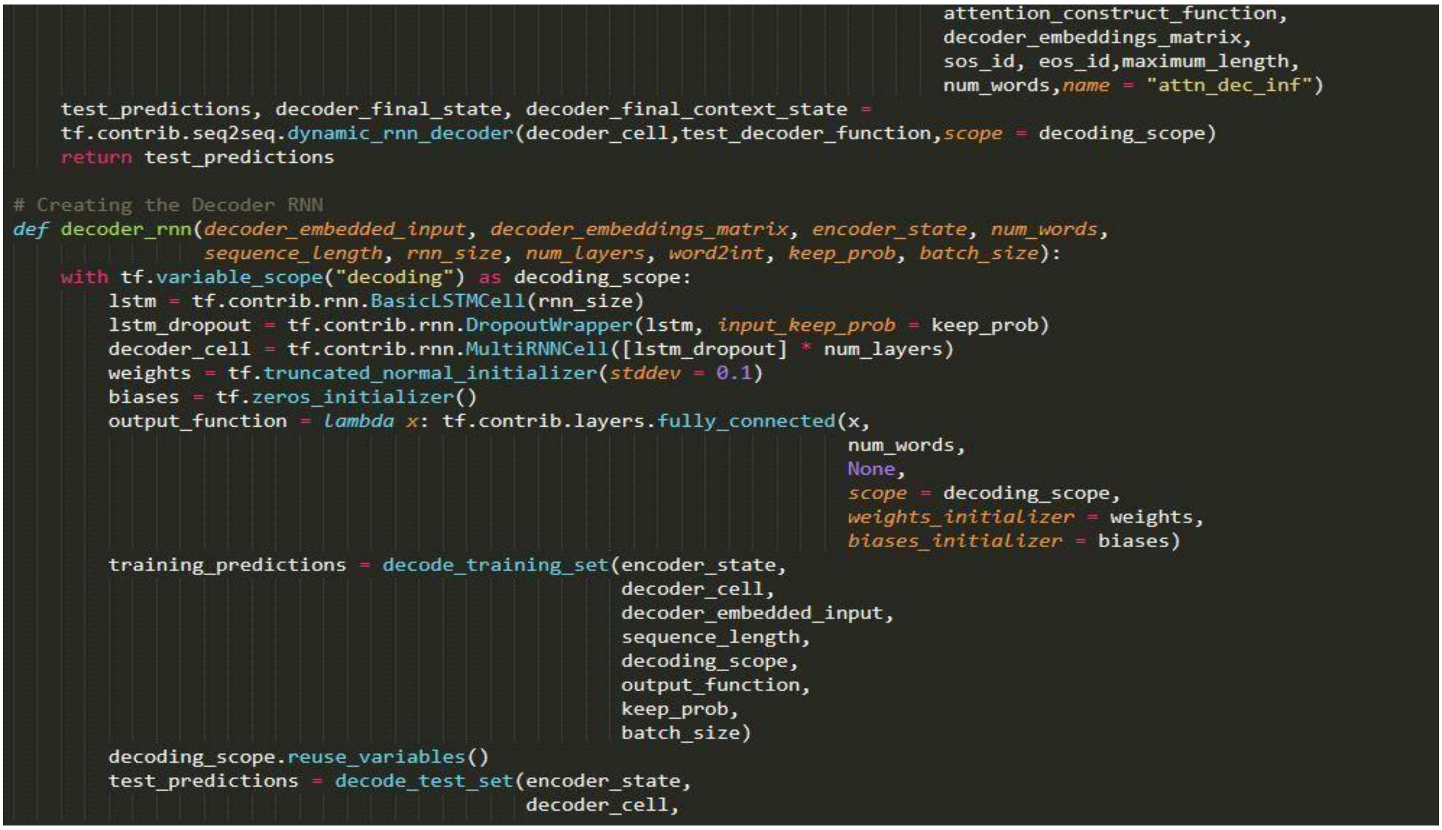


```
# Creating placeholders for the inputs and the targets
def model_inputs():
    inputs = tf.placeholder(tf.int32, [None, None], name = 'input')
    targets = tf.placeholder(tf.int32, [None, None], name = 'target')
    lr = tf.placeholder(tf.float32, name = 'learning_rate')
    keep_prob = tf.placeholder(tf.float32, name = 'keep_prob')
    return inputs, targets, lr, keep_prob

# Preprocessing the targets
def preprocess_targets(targets, word2int, batch_size):
    left_side = tf.fill([batch_size, 1], word2int['<SOS>'])
    right_side = tf.strided_slice(targets, [0,0], [batch_size, -1], [1,1])
    preprocessed_targets = tf.concat([left_side, right_side], 1)
    return preprocessed_targets

# Creating the Encoder RNN
def encoder_rnn(rnn_inputs, rnn_size, num_layers, keep_prob, sequence_length):
    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
    encoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
    encoder_output, encoder_state = tf.nn.bidirectional_dynamic_rnn(cell_fw = encoder_cell,
                                                                    cell_bw = encoder_cell,
                                                                    sequence_length = sequence_length,
                                                                    inputs = rnn_inputs,
                                                                    dtype = tf.float32)
    return encoder_state

# Decoding the training set
def decode_training_set(encoder_state, decoder_cell, decoder_embedded_input,
 sequence_length, decoding_scope, output_function, keep_prob, batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
    tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
```

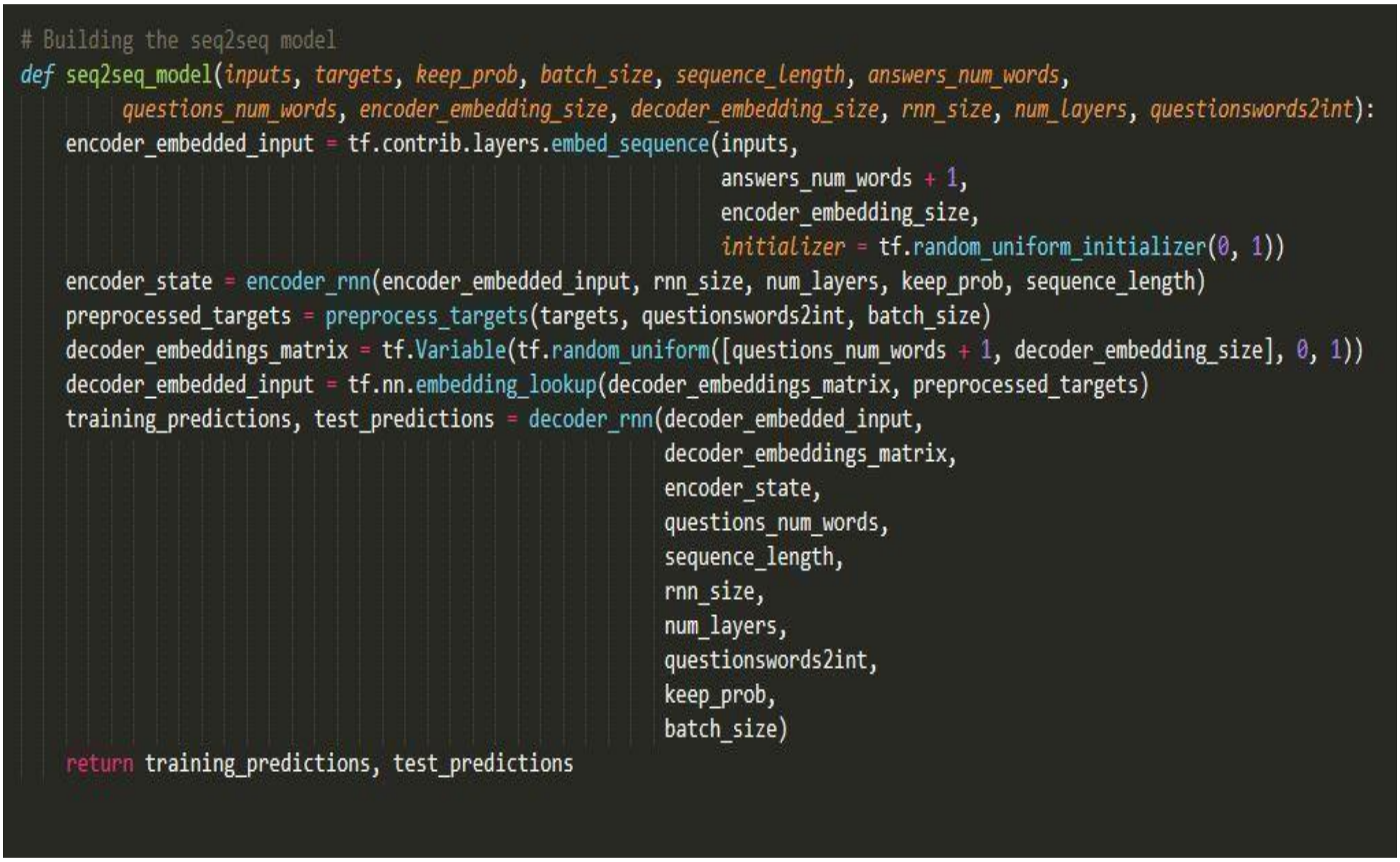


```
# Decoding the training set
def decode_training_set(encoder_state, decoder_cell, decoder_embedded_input,
 sequence_length, decoding_scope, output_function, keep_prob, batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
    tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
```

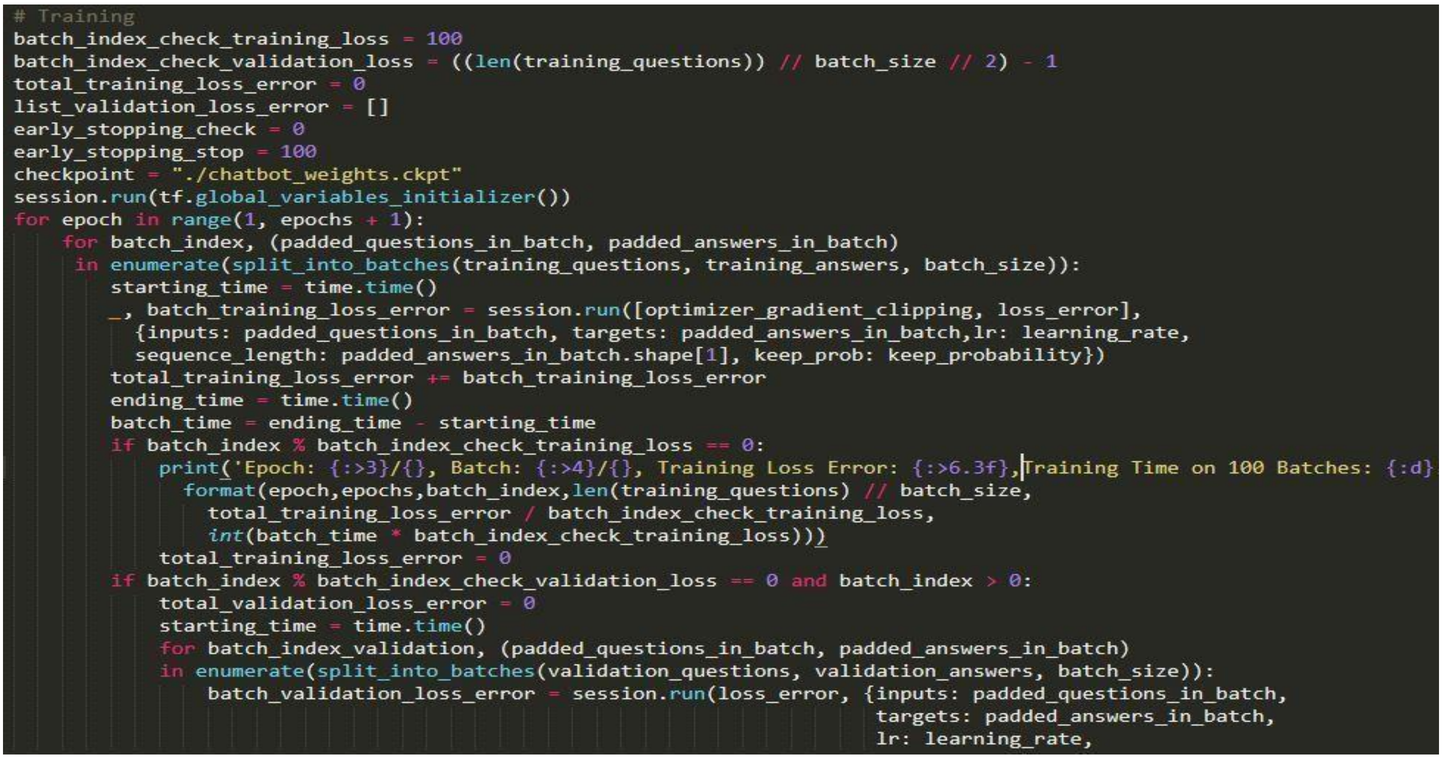
```
      num_units = decoder_cell.output_size)
    training_decoder_function = tf.contrib.seq2seq.attention_decoder_fn_train(encoder_state[0],
                                                                              attention_keys,
                                                                              attention_values,
                                                                              attention_score_function,
                                                                              attention_construct_function,
                                                                              name = "attn_dec_train")
    decoder_output, decoder_final_state, decoder_final_context_state = tf.contrib.seq2seq.dynamic_rnn_decoder
    (decoder_cell,
      training_decoder_function,
      decoder_embedded_input,
      sequence_length,
      scope = decoding_scope)
    decoder_output_dropout = tf.nn.dropout(decoder_output, keep_prob)
    return output_function(decoder_output_dropout)

# Decoding the test/validation set
def decode_test_set(encoder_state, decoder_cell, decoder_embeddings_matrix, sos_id, eos_id, maximum_length,
                     num_words, decoding_scope, output_function, keep_prob, batch_size):
    attention_states = tf.zeros([batch_size, 1, decoder_cell.output_size])
    attention_keys, attention_values, attention_score_function, attention_construct_function =
    tf.contrib.seq2seq.prepare_attention(attention_states, attention_option = "bahdanau",
      num_units = decoder_cell.output_size)
    test_decoder_function = tf.contrib.seq2seq.attention_decoder_fn_inference(output_function,encoder_state[0],
                                                                              attention_keys,attention_values,
                                                                              attention_score_function,
                                                                              attention_construct_function,
```



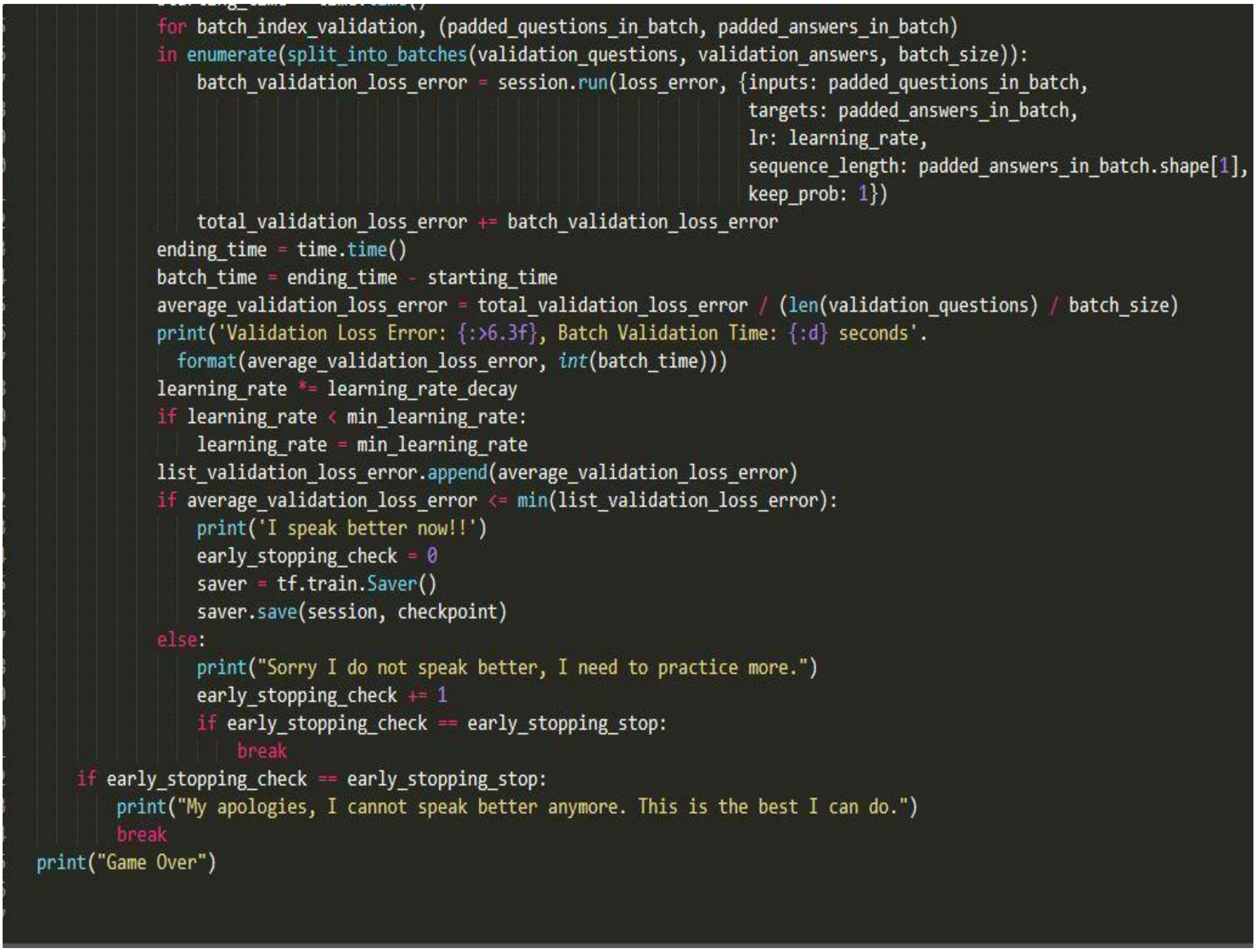
```
                                                                              attention_construct_function,
                                                                              decoder_embeddings_matrix,
                                                                              sos_id, eos_id,maximum_length,
                                                                              num_words,name = "attn_dec_inf")
    test_predictions, decoder_final_state, decoder_final_context_state =
    tf.contrib.seq2seq.dynamic_rnn_decoder(decoder_cell,test_decoder_function,scope = decoding_scope)
    return test_predictions

# Creating the Decoder RNN
def decoder_rnn(decoder_embedded_input, decoder_embeddings_matrix, encoder_state, num_words,
                sequence_length, rnn_size, num_layers, word2int, keep_prob, batch_size):
    with tf.variable_scope("decoding") as decoding_scope:
        lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
        lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
        decoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
        weights = tf.truncated_normal_initializer(stddev = 0.1)
        biases = tf.zeros_initializer()
        output_function = lambda x: tf.contrib.layers.fully_connected(x,
                                                                      num_words,
                                                                      None,
                                                                      scope = decoding_scope,
                                                                      weights_initializer = weights,
                                                                      biases_initializer = biases)
        training_predictions = decode_training_set(encoder_state,
                                                   decoder_cell,
                                                   decoder_embedded_input,
                                                   sequence_length,
                                                   decoding_scope,
                                                   output_function,
                                                   keep_prob,
                                                   batch_size)
        decoding_scope.reuse_variables()
        test_predictions = decode_test_set(encoder_state,
                                           decoder_cell,
```



```
# Building the seq2seq model
def seq2seq_model(inputs, targets, keep_prob, batch_size, sequence_length, answers_num_words,
        questions_num_words, encoder_embedding_size, decoder_embedding_size, rnn_size, num_layers, questionswords2int):
    encoder_embedded_input = tf.contrib.layers.embed_sequence(inputs,
                                                              answers_num_words + 1,
                                                              encoder_embedding_size,
                                                              initializer = tf.random_uniform_initializer(0, 1))
    encoder_state = encoder_rnn(encoder_embedded_input, rnn_size, num_layers, keep_prob, sequence_length)
    preprocessed_targets = preprocess_targets(targets, questionswords2int, batch_size)
    decoder_embeddings_matrix = tf.Variable(tf.random_uniform([questions_num_words + 1, decoder_embedding_size], 0, 1))
    decoder_embedded_input = tf.nn.embedding_lookup(decoder_embeddings_matrix, preprocessed_targets)
    training_predictions, test_predictions = decoder_rnn(decoder_embedded_input,
                                                         decoder_embeddings_matrix,
                                                         encoder_state,
                                                         questions_num_words,
                                                         sequence_length,
                                                         rnn_size,
                                                         num_layers,
                                                         questionswords2int,
                                                         keep_prob,
                                                         batch_size)
    return training_predictions, test_predictions
```



```
# Training
batch_index_check_training_loss = 100
batch_index_check_validation_loss = ((len(training_questions)) // batch_size // 2) - 1
total_training_loss_error = 0
list_validation_loss_error = []
early_stopping_check = 0
early_stopping_stop = 100
checkpoint = "./chatbot_weights.ckpt"
session.run(tf.global_variables_initializer())
for epoch in range(1, epochs + 1):
    for batch_index, (padded_questions_in_batch, padded_answers_in_batch)
     in enumerate(split_into_batches(training_questions, training_answers, batch_size)):
        starting_time = time.time()
        _, batch_training_loss_error = session.run([optimizer_gradient_clipping, loss_error],
          {inputs: padded_questions_in_batch, targets: padded_answers_in_batch,lr: learning_rate,
          sequence_length: padded_answers_in_batch.shape[1], keep_prob: keep_probability})
        total_training_loss_error += batch_training_loss_error
        ending_time = time.time()
        batch_time = ending_time - starting_time
        if batch_index % batch_index_check_training_loss == 0:
            print('Epoch: {:>3}/{}, Batch: {:>4}/{}, Training Loss Error: {:>6.3f},Training Time on 100 Batches: {:d}
              format(epoch,epochs,batch_index,len(training_questions) // batch_size,
                total_training_loss_error / batch_index_check_training_loss,
                int(batch_time * batch_index_check_training_loss)))
            total_training_loss_error = 0
        if batch_index % batch_index_check_validation_loss == 0 and batch_index > 0:
            total_validation_loss_error = 0
            starting_time = time.time()
            for batch_index_validation, (padded_questions_in_batch, padded_answers_in_batch)
            in enumerate(split_into_batches(validation_questions, validation_answers, batch_size)):
                batch_validation_loss_error = session.run(loss_error, {inputs: padded_questions_in_batch,
                                                                       targets: padded_answers_in_batch,
                                                                       lr: learning_rate,
```



```
            for batch_index_validation, (padded_questions_in_batch, padded_answers_in_batch)
            in enumerate(split_into_batches(validation_questions, validation_answers, batch_size)):
                batch_validation_loss_error = session.run(loss_error, {inputs: padded_questions_in_batch,
                                                                       targets: padded_answers_in_batch,
                                                                       lr: learning_rate,
                                                                       sequence_length: padded_answers_in_batch.shape[1],
                                                                       keep_prob: 1})
                total_validation_loss_error += batch_validation_loss_error
            ending_time = time.time()
            batch_time = ending_time - starting_time
            average_validation_loss_error = total_validation_loss_error / (len(validation_questions) / batch_size)
            print('Validation Loss Error: {:>6.3f}, Batch Validation Time: {:d} seconds'.
              format(average_validation_loss_error, int(batch_time)))
            learning_rate *= learning_rate_decay
            if learning_rate < min_learning_rate:
                learning_rate = min_learning_rate
            list_validation_loss_error.append(average_validation_loss_error)
            if average_validation_loss_error <= min(list_validation_loss_error):
                print('I speak better now!!')
                early_stopping_check = 0
                saver = tf.train.Saver()
                saver.save(session, checkpoint)
            else:
                print("Sorry I do not speak better, I need to practice more.")
                early_stopping_check += 1
                if early_stopping_check == early_stopping_stop:
                    break
    if early_stopping_check == early_stopping_stop:
        print("My apologies, I cannot speak better anymore. This is the best I can do.")
        break
print("Game Over")
```

INNOVATION AND USEFULNESS:

1) These chatterbots are responsive in nature.They respond in a similar manner like human beings.

2) This AI based application is really innovative and will prove to be very useful in communication fields.

3) These chatter bots are immensive in terms of holding data.

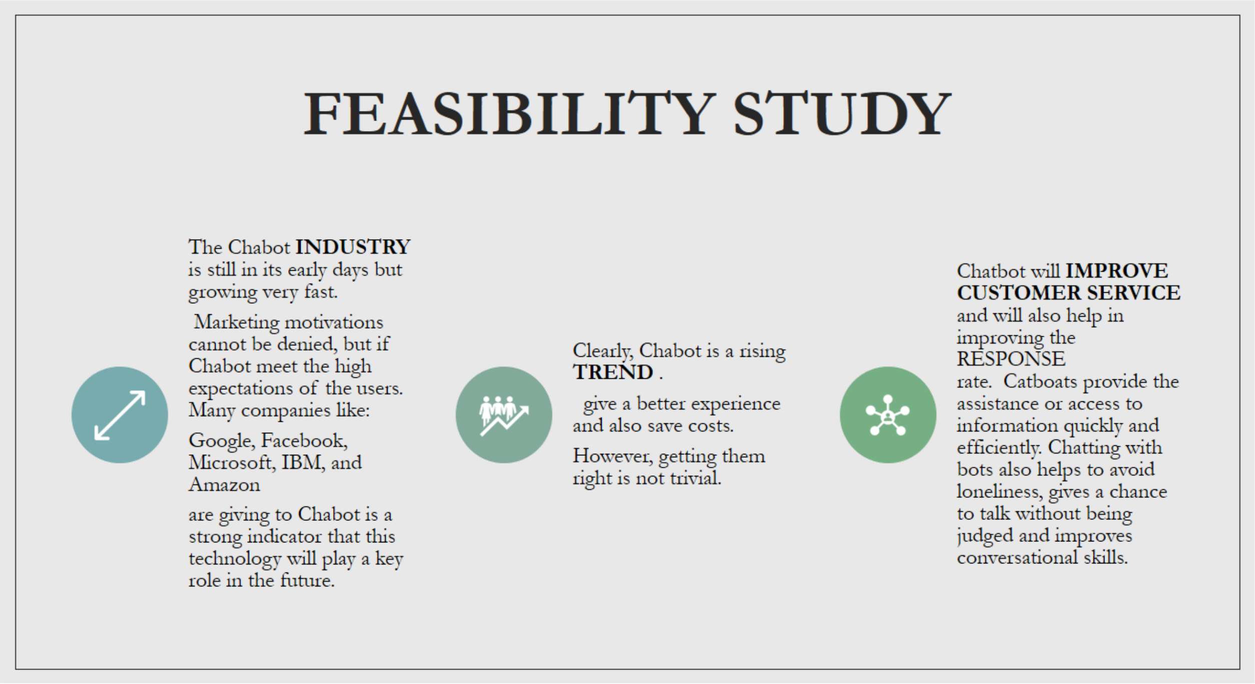
4) These bots are said to be human-like bots which provide general information about the website of companies,colleges,industries.

5) The innovation in field of ML and AI of these chatter bots is going to be a boon soon.

3.1. FEASIBILITY STUDY

The chatter bots are growing really well these days.Industries are making use of these human talking like robots so that the information can be given to anybody who enquires about them on various internet resources.The official websites generally have a pop-up of these chatbots on the screen so that the user can take general amout of information about that particular industry,college,company or any professional institute.

The chatbots are being seen having a bright future and thus their feasibilities are really high.Being applicable on various sites,they will prove to be a boon for the technological aspects world-wide.



FEASIBILITY STUDY

The Chabot **INDUSTRY** is still in its early days but growing very fast.

Marketing motivations cannot be denied, but if Chabot meet the high expectations of the users. Many companies like:

Google, Facebook, Microsoft, IBM, and Amazon

are giving to Chabot is a strong indicator that this technology will play a key role in the future.

Clearly, Chabot is a rising **TREND** .

give a better experience and also save costs.

However, getting them right is not trivial.

Chatbot will **IMPROVE CUSTOMER SERVICE** and will also help in improving the RESPONSE rate. Catboats provide the assistance or access to information quickly and efficiently. Chatting with bots also helps to avoid loneliness, gives a chance to talk without being judged and improves conversational skills.

REFERENCE AND BIBLIOGRAPHY:

- https://console.dialogflow.com/api-client/#/agent/8929b00d-30b7-468e- 94f0c8ff8f76426e/fulfillment
- https://www.youtube.com
- https://www.youtube.com/results?search_query=nlp+for+chatbot

https://www.youtube.com/results?search_query=dialogflow+chatbot+tutorial