AI CHAT BOT USING TENSORFLOW

PROJECT REPORT

OF MAJOR PRJECT

MASTER OF COMPUTER APPLICATIONS

SUBMITTED BY

Muneshwar Anchal
Batch Year – 2020-2022
Enrollment No. – U2049022

PROJECT SUPERVISOR – Dr. SARIKA YADAV

Centre of Computer Education & Training
Institute of Professional Studies
University of Allahabad, Prayagraj
Uttar Pradesh

Candidate's Declaration

This is certified that work, which is being presented in the project entitled Simple Chat Bot submitted by undersigned student of Master of Computer Application 4rd Semester in partial fulfillment for of 4rd Semester main project for Master of Computer Application is a record of our own work carried out under guidance and supervision of
Dr. Sarika Yadav.

This work has not submitted elsewhere for award of any other degree.

[illegible]

Date: 26/04/2022
Place: IPS, UoA, Prayagraj

Name and Signature of Student:
Muneshwar Anchal

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of any task Would be incomplete without the mention of people whose caseless corporation made it possible whose constant guidance and encouragement Crown all efforts with success.

I am grateful to our supervisor **Dr. Sarika Yadav** for the guidance, inspiration and constructive suggestions that help us in preparation of this project.

I also thanks who have helped in successful completion of the Project.

Date: 26/04/2022 Muneshwar Anchal

Place: IPS, UoA, Prayagraj MCA IV Semester

CONTENTS

1. Introduction

I. Abstract

In this paper we are going to introduce "AI Chat-bot" which is created through Google's TensorFlow. We are trying to develop a universal system with the ability to help our customers in such a way that reduces time and human resources.

A chatbot can be used anywhere a human is interacting with a computer system.

These are the areas where the fastest adoption is occurring:

1. Customer service

2. Sales and Marketing

Because it is economically available **24*7**.

The technology at the core of the rise of the chatbot is natural language processing ("NLP"). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations. This improvement in NLP is firing a great deal of additional research which should lead to continued improvement in the effectiveness of chatbots in the years to come.

.

II. Motivation

Social media has changed the way users approach customer service**.** Nearly half of the internet users are getting help through text services whether it is a tweet or Facebook messenger rather than calling toll free helpline customer support or drafting a detailed email. Twitter users send millions of requests to major companies monthly. With the rapid increase in the number of user requests, it has become increasingly challenging to process and respond to incoming requests.

To address this challenge, many organizations form dedicated customer service teams responding to user requests on social media. The team consists of dozens or even hundreds of human agents trained to address users' various needs [9].

However manually solving the request is time consuming and often fails user expectations. Recent studies show that 72% of users who contact a brand on Twitter expect a response within an hour [19]. Yet, our analysis of 1M conversations shows the average response time is 6.5

hours. This gap motivated us to explore the feasibility of chatbots for customer service on social media.

There has been a long history of chatbots powered by various techniques such as information retrieval and template rules [15]. Deep learning techniques have been recently applied to natural language generation; however, prior work focuses on general scenarios without specific contexts [7]. Lessons could also be informed by studies of social Q&A [5, 6, 13], where users may ask informational questions about products or services. Yet, it is not clear how such question types can be applied for customer service.

III. Problem Definition

Online shopping websites receive a great deal of inbound interaction and more than 75% of them are complaints or concerns. This in itself is not a problem; it’s a natural result of dealing in numbers. However, besides aiming for a high volume of sales, ecommerce websites must also aim for a high quality of customer support. So, when those customer complaints go unaddressed, it doesn’t bode well for the future of the business. So as an ecommerce business, how do you prevent customer issues from going unanswered? Hiring human support staff could be an option, but there’s a limit to how many queries these customer support reps would be able to address at a time. Add to this the restrictions of productive time limits, public holidays, and the mountainous costs of maintaining a fleet of 24/7 support staff.

2. Requirements

2.1 Software

Operating System: Windows or Linux

Technology: PYTHON, AIML, TENSORFLOW

2.2 Hardware

Processor: Pentium IV (minimum)

Hard Disk: 128GB

RAM: 256MB (minimum)

3. Objective

The goal of the system is to help the consumer to stay updated with trends. Artificial Intelligent is the fastest growing technology everywhere in the world, with the help of Artificial Intelligent and Knowledgeable database. We can make the transformation in the pattern matching and virtual assistance. This system is developing chat bot based on android system so with the combination of Artificial Intelligent Knowledgeable database and virtual assistance. We can develop such chat bot which will make a conversion between human and machine and will satisfy the question raised by user. The main motive of the project is to reduce the work load on the college's office staff and reduce the response time to a user's query.

3.1 Background and related work:

- **Existing System:**

Staples: is online retailer that sells office equipment. They use Facebook messenger to deliver product suggestions to customers, based on their previous brand behaviors.

H&M: is well known clothing retailer. They ask customers questions around their styles and offer them photo option to select from.

- **Disadvantage of existing System**

1. Pushes lots of Advertisement.
2. Don’t suggest appropriate article.
3. Need good interaction skill (Query must be in pre-defined format)

4. Chat bot system Architecture

Customer Service Chat bot via Deep Learning

The conversation between users and customer service agents on social media can be viewed as mapping one sequence of words representing the request to another sequence of words representing the response (see Figure 1). Deep learning techniques can be applied to learn the mapping from sequences to sequences [17].

Sequence-to-Sequence Learning

The core of the system consists of two LSTM neural networks: one as an encoder that maps a variable-length input sequence to a fixed-length vector, and the other as a decoder that maps the vector to a variable-length output sequence (Figure 1).

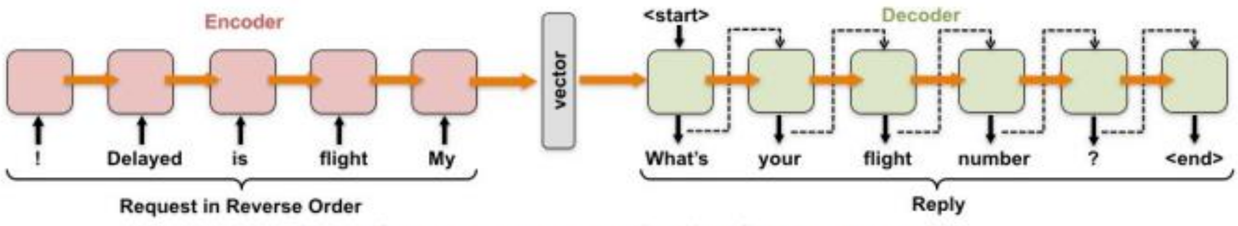


Figure 1. Sequence-to-sequence learning with LSTM neural networks.

5. Flow Chart

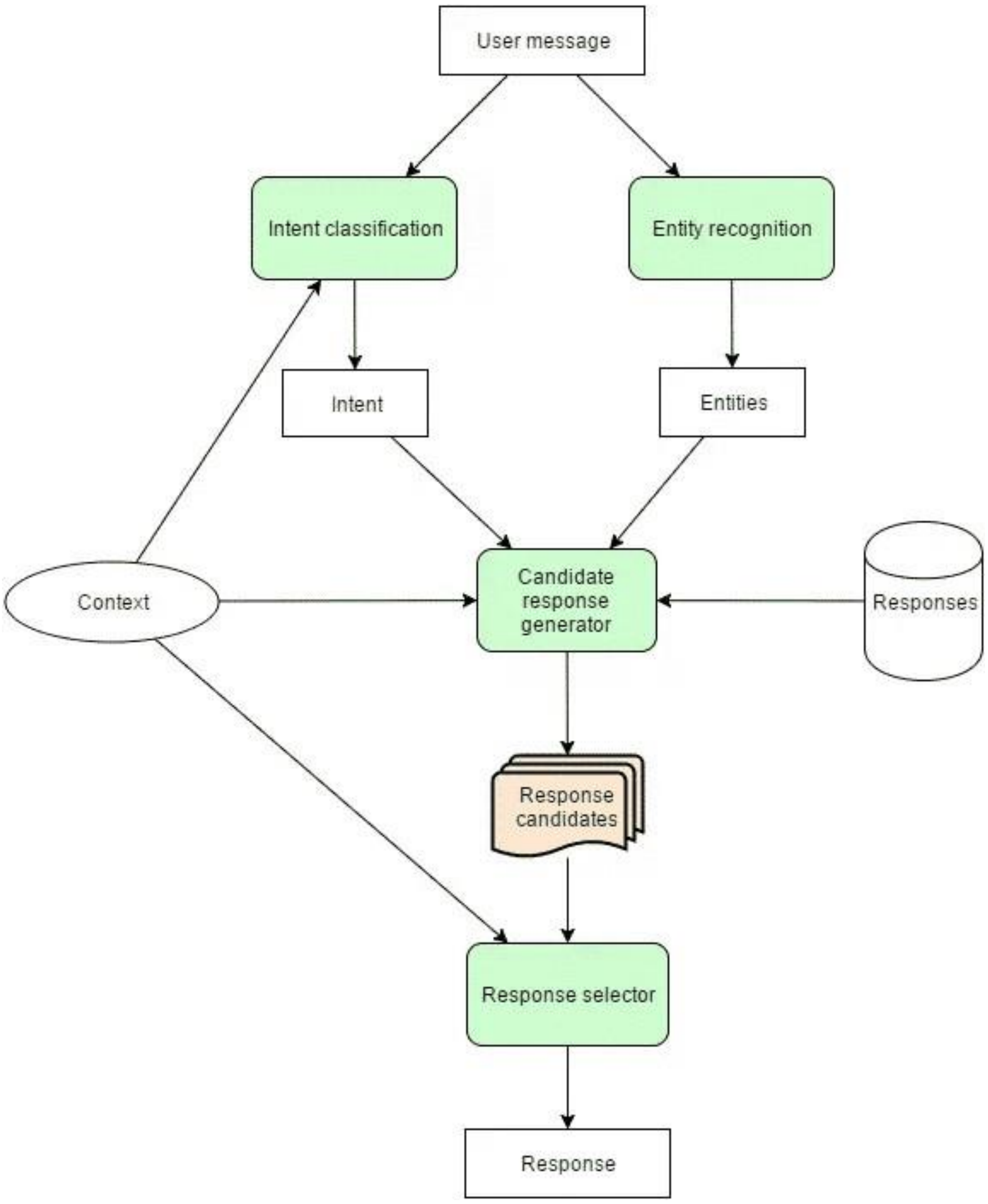


Figure 2 (Flow chart)

6. Some Important Terms:

Tokenization is a process of taking a set of text or text and breaking it up into its individual words or sentences. [6]

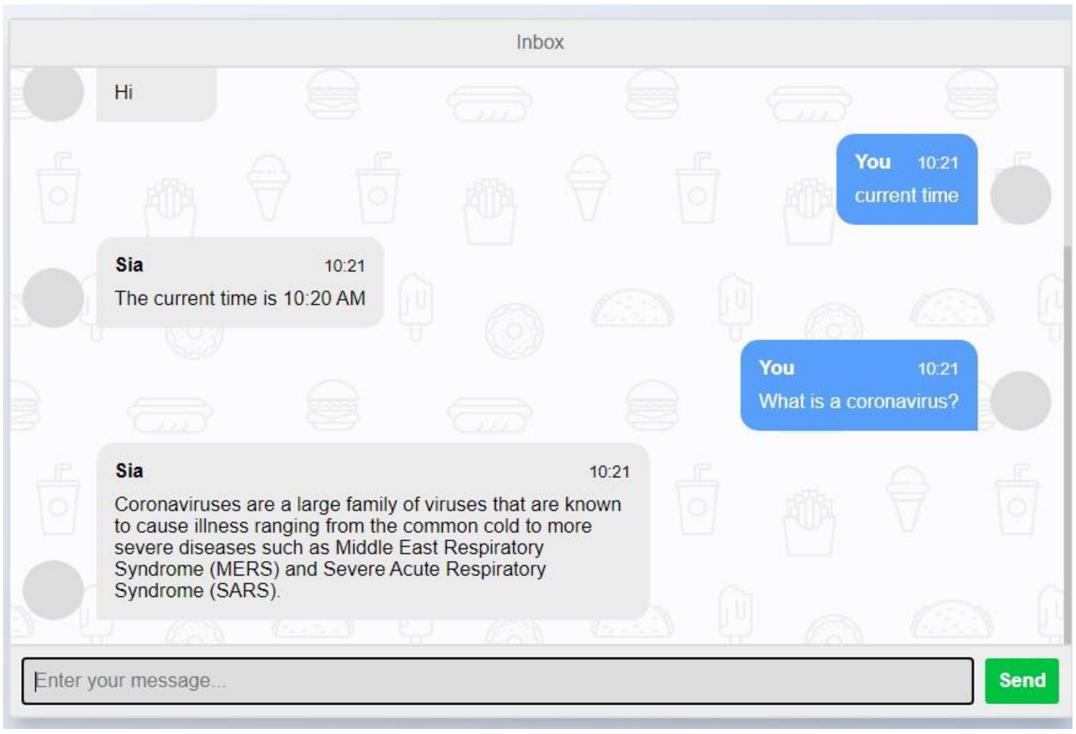
Lemmatization is the process of gathering the different inflected forms of a word so they can be dissected as a solitary item and is a variation of stemming. From there spell checker is used to identify and rectify spelling mistakes present in the query, then by using the sentence similarity and WordNet Algorithm a suitable response is explored in the knowledge database. [6]

WorldNet is a semantic and lexical database for the English language. It is used to group English words into the set of synonyms called syn_sets it gives short definitions and utilization models, and records various relations among these synonym sets or their individuals. If the response is found in the database it is displayed to the user, else the system notifies the admin about missing response in the database and gives a predefined response to the user. Admin can write the missing response into the database by logging into the admin block in website so that if the user asks the same query next time, he/she may get the suitable response. At the end of conversation, the college chatbot system collects the feedback from users to improve the system efficiency.
The functions of the user are to ask queries, provide feedback and so on. All the functions to be performed by the user are outlined below in detail as shown in Figure 2. [6]

7. Prototype:

We made a chatbot that we used as a prototype to investigate the research questions. During the design process we improved and tested the prototype. We tried to make it as helpful as we could manage within the time frames of the project by iterating multiple times.
It wasn't possible for me to use Python as back-end preprocessor, So I used FLASK frame work and then used Simple HTML web page to configure with flask and Python as Preprocessor.

In the making of the prototype, we also formed a persona for the chatbot to make the chatbot consistent in its language. This worked as a guideline in the design of the chatbot and was very helpful since it gave us a common understanding of the chatbots characteristics. We focused on building the chatbot as an engaging partner with a "happy tone" and a sense of humor, including GIFs to make the experience more fun and intriguing.


Inbox

Hi

You 10:21
current time

Sia 10:21
The current time is 10:20 AM

You 10:21
What is a coronavirus?

Sia 10:21
Coronaviruses are a large family of viruses that are known to cause illness ranging from the common cold to more severe diseases such as Middle East Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS).

Figure 3 (1st prototype)

8. Testing and Findings

In the beginning of our project, we wanted to test the previous version of our chatbot. This was late in the fall and most of the first-year students were familiar with a lot of the answers our chatbot could provide. We wanted to test this early version of the prototype to get input on what the chatbot could and could not answer in the future. After the test was completed, we had a short interview with the participants. The main purpose for this test was to see how the participants interacted with the prototype and find out if a chatbot could be suitable to find the information they needed. Before the testing we also carried out a pilot test to find immediate flaws in the plan.

8.1 Results from testing:

The first participant enjoyed talking to the bot, but stressed the fact that you had to "talk like "a dummy" for it to understand what you were asking. The participant pointed out that this really would have come in handy in his first weeks at the university, as he didn't always know who to ask - especially if he was in a hurry. He pointed out that the prototype needs to get more features like tell you current trend and discounts.

The second participant was a bit frustrated that the chatbot wasn't flexible enough. "I don't like having to guess what questions to ask". He would like more instructions to know how to get more out of the chatbot.

The third participant had also problems with understanding what the chatbot could do.

When given a hint for what the chatbot could do, the chatbot did not function properly. Here we tried to restart the system and then the chatbot displayed it´s welcome message what it could do. Afterwards it was clearer what the participant could ask it, but the chatbot did not always give the response that the participant wanted.

9. Discussion and Future work

Traditional customer service often emphasizes users' informational needs [9]; however, we found that over 40% of user requests on Twitter are emotional and they are not intended to seek specific information. This reveals a new paradigm of customer service interactions. One explanation is that, compared with calling the 1-800 number or writing an email, social media significantly lowers the cost of participation and allows more users to freely share their experiences with brands. Also, sharing emotions with the public is considered as one of the main motivations for using social media [1]. Future studies can examine how emotional requests are associated with users' motivation in the context of social media. Deep learning-based systems achieved similar performance as human agents in handling emotional requests, which represent a significant portion of user requests on social media. This finding opens new possibilities for integrating chatbots with human agents to support customer service on social media. For example, an automated technique can be designed to separate emotional and informational requests, and thus emotional requests can be routed to deep learning chatbots. The response speed can be greatly improved.

We observed that a deep learning-based system was able to learn writing styles from a brand and transfer them to another. Future work can explore the functionality in a more supervised fashion by filtering the training data with certain styles and specifying the target style for output sentences. This raises new opportunities of developing impression management tools on social media. As written text from brands and individual users affect how they are perceived on social media [18], such a tool can help them create images of themselves they wish to present.

Finally, chatbots on social media offer a new opportunity to provide individualized attention to users at scale and encourage interactions between users and brands, which can not only enhance brand performance but also help users gain social, information and economic benefits [3]. Future studies can be designed to understand how chatbots affect the relationship between users and brands in the long term.

10. Implementation

Step 1: Clean the data. We removed non-English requests and requests with images. All the @mentions were also removed in the training and testing data.

Step 2: Tokenize the data. We built a vocabulary of the most frequent 100K words in the conversations.

Step 3: Generate word-embedding features. We used the collected corpus to train word2vec models. Each word in the vocabulary was represented as a 640-dimension vector.

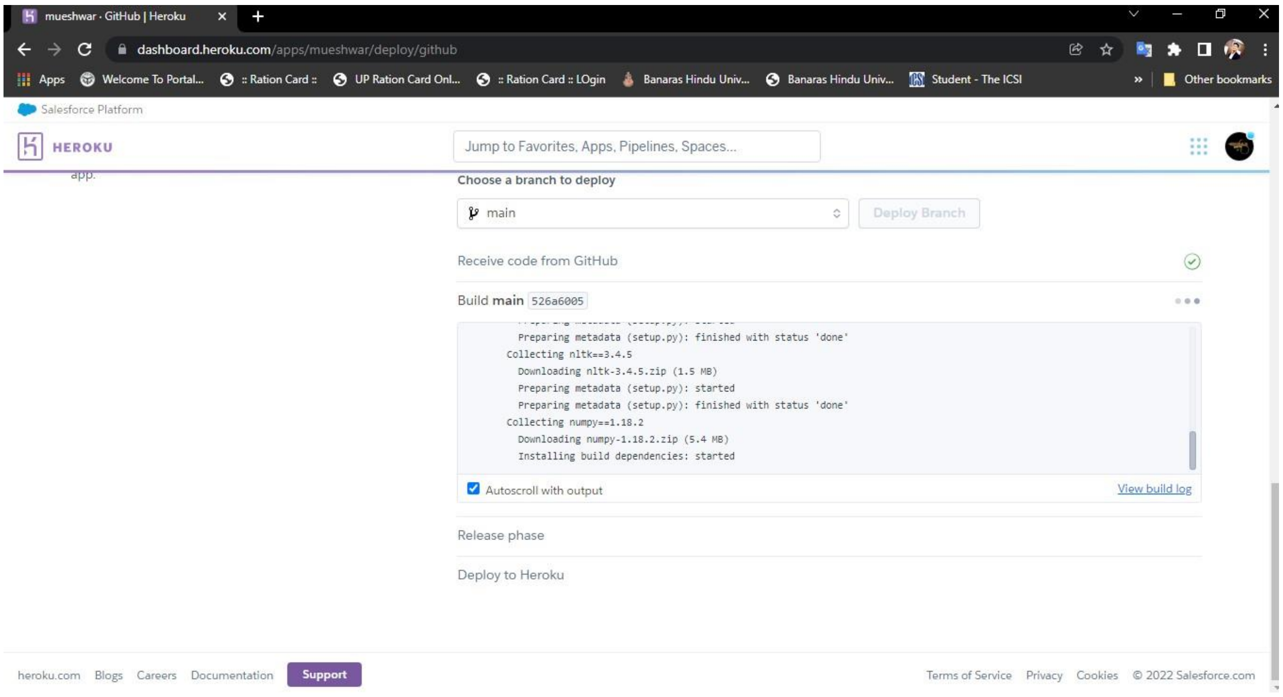
Step 4: Train LSTM networks. The input and output of LSTMs are vector representations of word sequences, with one word encoded or decoded at a time. In view of the clear advantage of deep LSTMs over shallow LSTMs in reported sequence-to-sequence tasks [17], we trained deep LSTMs jointly with 5 layers x 640 memory cells using stochastic gradient descent and gradient clipping.

11. Deployment

Most Heroku deployments are performed with Git. Heroku also supports Docker-based deployments. Additionally, you can deploy to Heroku via any of the following integrations:

- GitHub
- The Deploy to Heroku button
- Hashicorp Terraform

GitHub was best option for me to deploy my project, but it fails in installing some versions of the python libraries.


mueshwar · GitHub | Heroku
dashboard.heroku.com/apps/mueshwar/deploy/github
Salesforce Platform
HEROKU
Jump to Favorites, Apps, Pipelines, Spaces...

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub

Build **main** 526a6005

```
    Preparing metadata (setup.py): finished with status 'done'
  Collecting nltk==3.4.5
    Downloading nltk-3.4.5.zip (1.5 MB)
    Preparing metadata (setup.py): started
    Preparing metadata (setup.py): finished with status 'done'
  Collecting numpy==1.18.2
    Downloading numpy-1.18.2.zip (5.4 MB)
    Installing build dependencies: started
```

☑ Autoscroll with output — View build log

Release phase

Deploy to Heroku

heroku.com Blogs Careers Documentation Support — Terms of Service Privacy Cookies © 2022 Salesforce.com

Figure 4 (Error while deploying to Heroku)

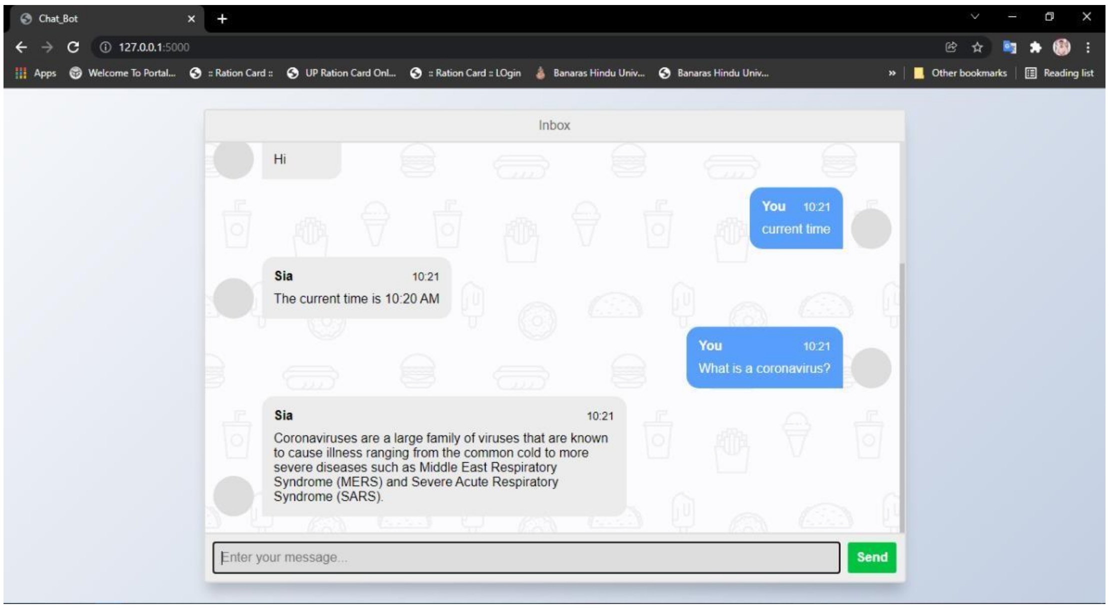
11. Future Scope and conclusion

This project can be used as WEB INTREGRATED chat bot which can be integrated with University of Allahabad, Prayagraj. Some of the characteristics of our chatbot was viewed as appropriate for the given context, like "casualness" and links to where the information was gathered. If the chatbot is to be furthered developed, this could be something to draw upon.

I will try to carry out this project for further developments and making it as expected.

12. Screenshots:

User Interface:



Hi

You 10:21
current time

Sia 10:21
The current time is 10:20 AM

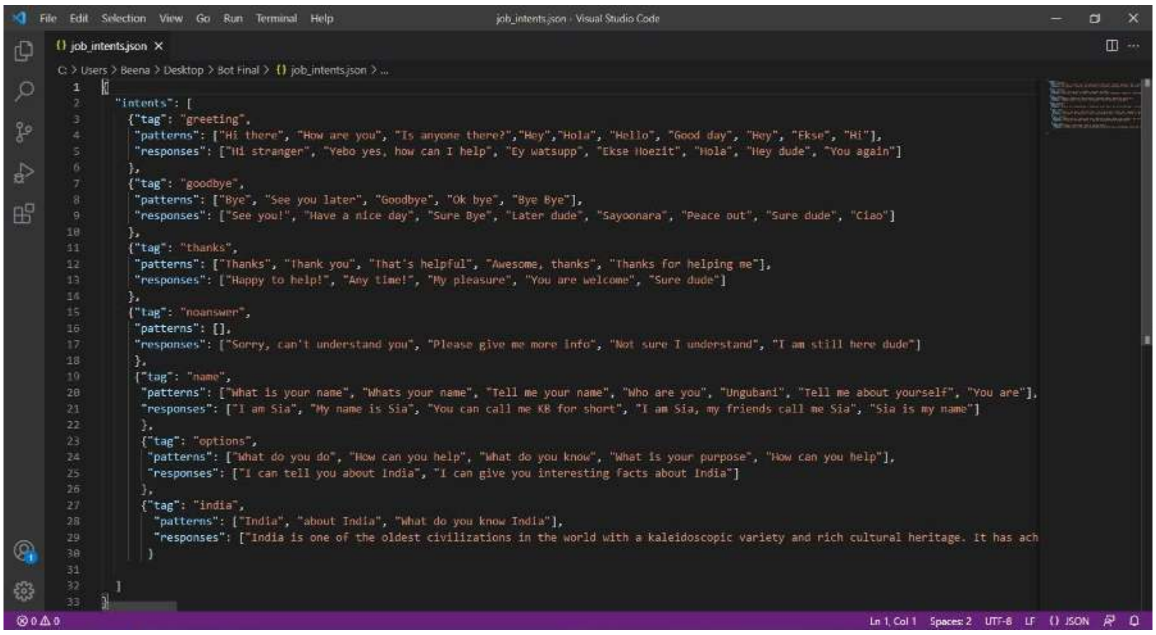
You 10:21
What is a coronavirus?

Sia 10:21
Coronaviruses are a large family of viruses that are known to cause illness ranging from the common cold to more severe diseases such as Middle East Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS).

Enter your message...

Figure 5 (user interface)

Data sets:


File Edit Selection View Go Run Terminal Help
job_intents.json - Visual Studio Code
job_intents.json
C: > Users > Beena > Desktop > Bot Final > job_intents.json > ...
Ln 1, Col 1 Spaces: 2 UTF-8 LF JSON

```
{
  "intents": [
    {"tag": "greeting",
     "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day", "Hey", "Ekse", "Hi"],
     "responses": ["Hi stranger", "Yebo yes, how can I help", "Ey watsupp", "Ekse Hoezit", "Hola", "Hey dude", "You again"]
    },
    {"tag": "goodbye",
     "patterns": ["Bye", "See you later", "Goodbye", "Ok bye", "Bye Bye"],
     "responses": ["See you!", "Have a nice day", "Sure Bye", "Later dude", "Sayoonara", "Peace out", "Sure dude", "Ciao"]
    },
    {"tag": "thanks",
     "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
     "responses": ["Happy to help!", "Any time!", "My pleasure", "You are welcome", "Sure dude"]
    },
    {"tag": "noanswer",
     "patterns": [],
     "responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand", "I am still here dude"]
     },
     {"tag": "name",
      "patterns": ["What is your name", "Whats your name", "Tell me your name", "Who are you", "Ungubani", "Tell me about yourself", "You are"],
      "responses": ["I am Sia", "My name is Sia", "You can call me KB for short", "I am Sia, my friends call me Sia", "Sia is my name"]
      },
      {"tag": "options",
       "patterns": ["What do you do", "How can you help", "What do you know", "What is your purpose", "How can you help"],
       "responses": ["I can tell you about India", "I can give you interesting facts about India"]
      },
      {"tag": "india",
        "patterns": ["India", "about India", "What do you know India"],
        "responses": ["India is one of the oldest civilizations in the world with a kaleidoscopic variety and rich cultural heritage. It has ach
       }

    ]
}
```

Figure 6 (Sample Indents for basic training)

**Other data set has been downloaded from reddit_data set for training purposes .

13. Code Overview

• Chatbot.py

```
from __future__ import print_function

import numpy as np
import tensorflow as tf

import argparse
import os
import pickle
import copy
import sys
import html

from utils import TextLoader
from model import Model

def main():
    assert sys.version_info >= (3, 3), \
    "Must be run in Python 3.3 or later. You are running {}".format(sys.version)
    parser = argparse.ArgumentParser()
    parser.add_argument('--save_dir', type=str, default='models/reddit',
                    help='model directory to store checkpointed models')
    parser.add_argument('-n', type=int, default=500,
                    help='number of characters to sample')
    parser.add_argument('--prime', type=str, default=' ',
                    help='prime text')
    parser.add_argument('--beam_width', type=int, default=2,
                    help='Width of the beam for beam search, default 2')
    parser.add_argument('--temperature', type=float, default=1.0,
                    help='sampling temperature'
                    '(lower is more conservative, default is 1.0, which is neutral)')
    parser.add_argument('--topn', type=int, default=-1,
                     help='at each step, choose from only this many most likely characters;'
                     'set to <0 to disable top-n filtering.')
    parser.add_argument('--relevance', type=float, default=-1.,
                    help='amount of "relevance masking/MMI (disabled by default):"'
                    'higher is more pressure, 0.4 is probably as high as it can go without'
                    'noticeably degrading coherence;'
                    'set to <0 to disable relevance masking')
    args = parser.parse_args()
    sample_main(args)

def get_paths(input_path):
    if os.path.isfile(input_path):
        # Passed a model rather than a checkpoint directory
        model_path = input_path
        save_dir = os.path.dirname(model_path)
    elif os.path.exists(input_path):
        # Passed a checkpoint directory
        save_dir = input_path
        checkpoint = tf.train.get_checkpoint_state(save_dir)
        if checkpoint:
            model_path = checkpoint.model_checkpoint_path
        else:
            raise ValueError('Checkpoint not found in {}.'.format(save_dir))
    else:
        raise ValueError('save_dir is not a valid path.')
    return model_path, os.path.join(save_dir, 'config.pkl'), os.path.join(save_dir,
'chars_vocab.pkl')

def sample_main(args):
    model_path, config_path, vocab_path = get_paths(args.save_dir)
    # Arguments passed to sample.py direct us to a saved model.
    # Load the separate arguments by which that model was previously trained.
    # That's saved_args. Use those to load the model.
    with open(config_path, 'rb') as f:
        saved_args = pickle.load(f)
    # Separately load chars and vocab from the save directory.
    with open(vocab_path, 'rb') as f:
        chars, vocab = pickle.load(f)
    # Create the model from the saved arguments, in inference mode.
    print("Creating model...")
    saved_args.batch_size = args.beam_width
    net = Model(saved_args, True)
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    # Make tensorflow less verbose; filter out info (1+) and warnings (2+) but not errors (3).
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
    with tf.Session(config=config) as sess:
        tf.global_variables_initializer().run()
        saver = tf.train.Saver(net.save_variables_list())
        # Restore the saved variables, replacing the initialized values.
        print("Restoring weights...")
        saver.restore(sess, model_path)
        chatbot(net, sess, chars, vocab, args.n, args.beam_width,
            args.relevance, args.temperature, args.topn)

def initial_state(net, sess):
    # Return freshly initialized model states.
    return sess.run(net.zero_state)

def forward_text(net, sess, states, relevance, vocab, prime_text=None):
    if prime_text is not None:
        for char in prime_text:
            if relevance > 0.:
                # Automatically forward the primary net.
                _, states[0] = net.forward_model(sess, states[0], vocab[char])
                # If the token is newline, reset the mask net state; else, forward it.
                if vocab[char] == '\n':
                    states[1] = initial_state(net, sess)
                else:
                    _, states[1] = net.forward_model(sess, states[1], vocab[char])
            else:
                _, states = net.forward_model(sess, states, vocab[char])
    return states

def sanitize_text(vocab, text): # Strip out characters that are not part of the net's vocab.
    return ''.join(i for i in text if i in vocab)

def initial_state_with_relevance_masking(net, sess, relevance):
    if relevance <= 0.: return initial_state(net, sess)
    else: return [initial_state(net, sess), initial_state(net, sess)]

def possibly_escaped_char(raw_chars):
    if raw_chars[-1] == ';':
        for i, c in enumerate(reversed(raw_chars[:-1])):
            if c == ';' or i > 8:
                return raw_chars[-1]
            elif c == '&':
                escape_seq = "".join(raw_chars[-(i + 2):])
                new_seq = html.unescape(escape_seq)
                backspace_seq = "".join(['\b'] * (len(escape_seq)-1))
                diff_length = len(escape_seq) - len(new_seq) - 1
                return backspace_seq + new_seq + "".join([' '] * diff_length) + "".join(['\b'] *
diff_length)
    return raw_chars[-1]

def chatbot(net, sess, chars, vocab, max_length, beam_width, relevance, temperature, topn):
    states = initial_state_with_relevance_masking(net, sess, relevance)
    while True:
        user_input = input('\n> ')
        user_command_entered, reset, states, relevance, temperature, topn, beam_width =
process_user_command(
            user_input, states, relevance, temperature, topn, beam_width)
        if reset: states = initial_state_with_relevance_masking(net, sess, relevance)
        if not user_command_entered:
            states = forward_text(net, sess, states, relevance, vocab, sanitize_text(vocab, "> " +
user_input + "\n>"))
            computer_response_generator = beam_search_generator(sess=sess, net=net,
                initial_state=copy.deepcopy(states), initial_sample=vocab[' '],
                early_term_token=vocab['\n'], beam_width=beam_width,
forward_model_fn=forward_with_mask,
                forward_args={'relevance':relevance, 'mask_reset_token':vocab['\n'],
'forbidden_token':vocab['>'],
                                'temperature':temperature, 'topn':topn})
            out_chars = []
            for i, char_token in enumerate(computer_response_generator):
                out_chars.append(chars[char_token])
                print(possibly_escaped_char(out_chars), end='', flush=True)
                states = forward_text(net, sess, states, relevance, vocab, chars[char_token])
                if i >= max_length: break
            states = forward_text(net, sess, states, relevance, vocab, sanitize_text(vocab, "\n> "))

def process_user_command(user_input, states, relevance, temperature, topn, beam_width):
    user_command_entered = False
    reset = False
    try:
        if user_input.startswith('--temperature '):
            user_command_entered = True
            temperature = max(0.001, float(user_input[len('--temperature '):]))
            print("[Temperature set to {}]".format(temperature))
        elif user_input.startswith('--relevance '):
            user_command_entered = True
            new_relevance = float(user_input[len('--relevance '):])
            if relevance <= 0. and new_relevance > 0.:
                states = [states, copy.deepcopy(states)]
            elif relevance > 0. and new_relevance <= 0.:
                states = states[0]
            relevance = new_relevance
            print("[Relevance disabled]" if relevance <= 0. else "[Relevance set to
{}]".format(relevance))
        elif user_input.startswith('--topn '):
            user_command_entered = True
            topn = int(user_input[len('--topn '):])
            print("[Top-n filtering disabled]" if topn <= 0 else "[Top-n filtering set to
{}]".format(topn))
        elif user_input.startswith('--beam_width '):
            user_command_entered = True
            beam_width = max(1, int(user_input[len('--beam_width '):]))
            print("[Beam width set to {}]".format(beam_width))
        elif user_input.startswith('--reset'):
            user_command_entered = True
            reset = True
            print("[Model state reset]")
    except ValueError:
        print("[Value error with provided argument.]")
    return user_command_entered, reset, states, relevance, temperature, topn, beam_width

def consensus_length(beam_outputs, early_term_token):
    for l in range(len(beam_outputs[0])):
        if l > 0 and beam_outputs[0][l-1] == early_term_token:
            return l-1, True
        for b in beam_outputs[1:]:
            if beam_outputs[0][l] != b[l]: return l, False
    return l, False

def scale_prediction(prediction, temperature):
    if (temperature == 1.0): return prediction # Temperature 1.0 makes no change
    np.seterr(divide='ignore')
    scaled_prediction = np.log(prediction) / temperature
    scaled_prediction = scaled_prediction - np.logaddexp.reduce(scaled_prediction)
    scaled_prediction = np.exp(scaled_prediction)
    np.seterr(divide='warn')
    return scaled_prediction

def forward_with_mask(sess, net, states, input_sample, forward_args):
    # forward_args is a dictionary containing arguments for generating probabilities.
    relevance = forward_args['relevance']
    mask_reset_token = forward_args['mask_reset_token']
    forbidden_token = forward_args['forbidden_token']
    temperature = forward_args['temperature']
    topn = forward_args['topn']

    if relevance <= 0.:
        # No relevance masking.
        prob, states = net.forward_model(sess, states, input_sample)
    else:
        # states should be a 2-length list: [primary net state, mask net state].
        if input_sample == mask_reset_token:
```

```
        # Reset the mask probs when reaching mask_reset_token (newline).
        states[1] = initial_state(net, sess)
      primary_prob, states[0] = net.forward_model(sess, states[0], input_sample)
      primary_prob /= sum(primary_prob)
      mask_prob, states[1] = net.forward_model(sess, states[1], input_sample)
      mask_prob /= sum(mask_prob)
      prob = np.exp(np.log(primary_prob) - relevance * np.log(mask_prob))
    # Mask out the forbidden token (">") to prevent the bot from deciding the chat is over)
    prob[forbidden_token] = 0
    # Normalize probabilities so they sum to 1.
    prob = prob / sum(prob)
    # Apply temperature.
    prob = scale_prediction(prob, temperature)
    # Apply top-n filtering if enabled
    if topn > 0:
      prob[np.argsort(prob)[:-topn]] = 0
      prob = prob / sum(prob)
    return prob, states

def beam_search_generator(sess, net, initial_state, initial_sample,
    early_term_token, beam_width, forward_model_fn, forward_args):
    '''Run beam search! Yield consensus tokens sequentially, as a generator;
    return when reaching early_term_token (newline).
    Args:
      sess: tensorflow session reference
      net: tensorflow net graph (must be compatible with the forward_net function)
      initial_state: initial hidden state of the net
      initial_sample: single token (excluding any seed/priming material)
        to start the generation
      early_term_token: stop when the beam reaches consensus on this token
        (but do not return this token).
      beam_width: how many beams to track
      forward_model_fn: function to forward the model, must be of the form:
        probability_output, beam_state =
              forward_model_fn(sess, net, beam_state, beam_sample, forward_args)
        (Note: probability_output has to be a valid probability distribution!)
      tot_steps: how many tokens to generate before stopping,
        unless already stopped via early_term_token.
    Returns: a generator to yield a sequence of beam-sampled tokens.'''
    # Store state, outputs and probabilities for up to args.beam_width beams.
    # Initialize with just the one starting entry; it will branch to fill the beam
    # in the first step.
    beam_states = [initial_state] # Stores the best activation states
    beam_outputs = [[initial_sample]] # Stores the best generated output sequences so far.
    beam_probs = [1.] # Stores the cumulative normalized probabilities of the beams so far.
    while True:
        # Keep a running list of the best beam branches for next step.
        # Don't actually copy any big data structures yet, just keep references
        # to existing beam state entries, and then clone them as necessary
        # at the end of the generation step.
        new_beam_indices = []
        new_beam_probs = []
        new_beam_samples = []

        # Iterate through the beam entries.
        for beam_index, beam_state in enumerate(beam_states):
            beam_prob = beam_probs[beam_index]
            beam_sample = beam_outputs[beam_index][-1]

            # Forward the model.
            prediction, beam_states[beam_index] = forward_model_fn(
                    sess, net, beam_state, beam_sample, forward_args)

            # Sample best_tokens from the probability distribution.
            # Sample from the scaled probability distribution beam_width choices
            # (but not more than the number of positive probabilities in scaled_prediction).
            count = min(beam_width, sum(1 if p > 0. else 0 for p in prediction))
            best_tokens = np.random.choice(len(prediction), size=count,
                                    replace=False, p=prediction)
            for token in best_tokens:
                prob = prediction[token] * beam_prob
                if len(new_beam_indices) < beam_width:
                    # If we don't have enough new_beam_indices, we automatically qualify.
                    new_beam_indices.append(beam_index)
                    new_beam_probs.append(prob)
                    new_beam_samples.append(token)
                else:
                    # Sample a low-probability beam to possibly replace.
                    np_new_beam_probs = np.array(new_beam_probs)
                    inverse_probs = -np_new_beam_probs + max(np_new_beam_probs) +
min(np_new_beam_probs)
                    inverse_probs = inverse_probs / sum(inverse_probs)
                    sampled_beam_index = np.random.choice(beam_width, p=inverse_probs)
                    if new_beam_probs[sampled_beam_index] <= prob:
                        # Replace it.
                        new_beam_indices[sampled_beam_index] = beam_index
                        new_beam_probs[sampled_beam_index] = prob
                        new_beam_samples[sampled_beam_index] = token
        # Replace the old states with the new states, first by referencing and then by copying.
        already_referenced = [False] * beam_width
        new_beam_states = []
        new_beam_outputs = []
        for i, new_index in enumerate(new_beam_indices):
            if already_referenced[new_index]:
                new_beam = copy.deepcopy(beam_states[new_index])
            else:
                new_beam = beam_states[new_index]
                already_referenced[new_index] = True
            new_beam_states.append(new_beam)
            new_beam_outputs.append(beam_outputs[new_index] + [new_beam_samples[i]])
        # Normalize the beam probabilities so they don't drop to zero
        beam_probs = new_beam_probs / sum(new_beam_probs)
        beam_states = new_beam_states
        beam_outputs = new_beam_outputs
        # Prune the agreed portions of the outputs
        # and yield the tokens on which the beam has reached consensus.
        l, early_term = consensus_length(beam_outputs, early_term_token)
        if l > 0:
            for token in beam_outputs[0][:l]: yield token
            beam_outputs = [output[l:] for output in beam_outputs]
        if early_term: return

if __name__ == '__main__':
    main()
```

- **Model.py**

```
import tensorflow as tf
from tensorflow.python.ops import rnn_cell
from tensorflow.python.ops import nn_ops
from tensorflow.python.ops import variable_scope as vs
from tensorflow.python.framework import ops
from tensorflow.contrib import rnn

from tensorflow.python.util.nest import flatten

import numpy as np

class PartitionedMultiRNNCell(rnn_cell.RNNCell):
    """RNN cell composed sequentially of multiple simple cells."""
```

```
  def __init__(self, cell_fn, partition_size=128, partitions=1, layers=2):
    """Create a RNN cell composed sequentially of a number of RNNCells.
    Args:
      cell_fn: reference to RNNCell function to create each partition in each layer.
      partition_size: how many horizontal cells to include in each partition.
      partitions: how many horizontal partitions to include in each layer.
      layers: how many layers to include in the net.
    """
    super(PartitionedMultiRNNCell, self).__init__()

    self._cells = []
    for i in range(layers):
      self._cells.append([cell_fn(partition_size) for _ in range(partitions)])
    self._partitions = partitions

  @property
  def state_size(self):
    # Return a 2D tuple where each row is the partition's cell size repeated `partitions`
times,
    # and there are `layers` rows of that.
    return tuple(((layer[0].state_size,) * len(layer)) for layer in self._cells)

  @property
  def output_size(self):
    # Return the output size of each partition in the last layer times the number of partitions
per layer.
    return self._cells[-1][0].output_size * len(self._cells[-1])

  def zero_state(self, batch_size, dtype):
    # Return a 2D tuple of zero states matching the structure of state_size.
    with ops.name_scope(type(self).__name__ + "ZeroState", values=[batch_size]):
      return tuple(tuple(cell.zero_state(batch_size, dtype) for cell in layer) for layer in
self._cells)

  def call(self, inputs, state):
    layer_input = inputs
    new_states = []
    for l, layer in enumerate(self._cells):
      # In between layers, offset the layer input by half of a partition width so that
      # activations can horizontally spread through subsequent layers.
      if l > 0:
        offset_width = layer[0].output_size // 2
        layer_input = tf.concat((layer_input[:, -offset_width:], layer_input[:, :-
offset_width]),
            axis=1, name='concat_offset_%d' % l)
            # Create a tuple of inputs by splitting the lower layer output into partitions.
            p_inputs = tf.split(layer_input, len(layer), axis=1, name='split_%d' % l)
            p_outputs = []
            p_states = []
            for p, p_inp in enumerate(p_inputs):
                with vs.variable_scope("cell_%d_%d" % (l, p)):
                    p_state = state[l][p]
                    cell = layer[p]
                    p_out, new_p_state = cell(p_inp, p_state)
                    p_outputs.append(p_out)
                    p_states.append(new_p_state)
            new_states.append(tuple(p_states))
            layer_input = tf.concat(p_outputs, axis=1, name='concat_%d' % l)
        new_states = tuple(new_states)
        return layer_input, new_states

def _rnn_state_placeholders(state):
    """Convert RNN state tensors to placeholders, reflecting the same nested tuple
structure."""
    # Adapted from @carlthome's comment:
    # https://github.com/tensorflow/tensorflow/issues/2838#issuecomment-302019188
    if isinstance(state, tf.contrib.rnn.LSTMStateTuple):
        c, h = state
        c = tf.placeholder(c.dtype, c.shape, c.op.name)
        h = tf.placeholder(h.dtype, h.shape, h.op.name)
        return tf.contrib.rnn.LSTMStateTuple(c, h)
    elif isinstance(state, tf.Tensor):
        h = state
        h = tf.placeholder(h.dtype, h.shape, h.op.name)
        return h
    else:
        structure = [_rnn_state_placeholders(x) for x in state]
        return tuple(structure)

class Model():
    def __init__(self, args, infer=False): # infer is set to true during sampling.
        self.args = args
        if infer:
            # Worry about one character at a time during sampling; no batching or BPTT.
            args.batch_size = 1
            args.seq_length = 1

        # Set cell_fn to the type of network cell we're creating -- RNN, GRU, LSTM or NAS.
        if args.model == 'rnn':
            cell_fn = rnn_cell.BasicRNNCell
        elif args.model == 'gru':
            cell_fn = rnn_cell.GRUCell
        elif args.model == 'lstm':
            cell_fn = rnn_cell.BasicLSTMCell
        elif args.model == 'nas':
            cell_fn = rnn.NASCell
        else:
            raise Exception("model type not supported: {}".format(args.model))

        # Create variables to track training progress.
        self.lr = tf.Variable(args.learning_rate, name="learning_rate", trainable=False)
        self.global_epoch_fraction = tf.Variable(0.0, name="global_epoch_fraction",
trainable=False)
        self.global_seconds_elapsed = tf.Variable(0.0, name="global_seconds_elapsed",
trainable=False)

        # Call tensorflow library tensorflow-master/tensorflow/python/ops/rnn_cell
        # to create a layer of block_size cells of the specified basic type (RNN/GRU/LSTM).
        # Use the same rnn_cell library to create a stack of these cells
        # of num_layers layers. Pass in a python list of these cells.
        # cell = rnn_cell.MultiRNNCell([cell_fn(args.block_size) for _ in
range(args.num_layers)])
        # cell = MyMultiRNNCell([cell_fn(args.block_size) for _ in range(args.num_layers)])
        cell = PartitionedMultiRNNCell(cell_fn, partitions=args.num_blocks,
            partition_size=args.block_size, layers=args.num_layers)

        # Create a TF placeholder node of 32-bit ints (NOT floats!),
        # of shape batch_size x seq_length. This shape matches the batches
        # (listed in x_batches and y_batches) constructed in create_batches in utils.py.
        # input_data will receive input batches.
        self.input_data = tf.placeholder(tf.int32, [args.batch_size, args.seq_length])

        self.zero_state = cell.zero_state(args.batch_size, tf.float32)

        self.initial_state = _rnn_state_placeholders(self.zero_state)
        self._flattened_initial_state = flatten(self.initial_state)

        layer_size = args.block_size * args.num_blocks

        # Scope our new variables to the scope identifier string "rnnlm".
        with tf.variable_scope('rnnlm'):
            # Create new variable softmax_w and softmax_b for output.
            # softmax_w is a weights matrix from the top layer of the model (of size layer_size)
            # to the vocabulary output (of size vocab_size).
            softmax_w = tf.get_variable("softmax_w", [layer_size, args.vocab_size])
```

```
            # softmax_b is a bias vector of the ouput characters (of size vocab_size).
            softmax_b = tf.get_variable("softmax_b", [args.vocab_size])
            # Create new variable named 'embedding' to connect the character input to the base
layer
            # of the RNN. Its role is the conceptual inverse of softmax_w.
            # It contains the trainable weights from the one-hot input vector to the lowest layer
of RNN.
            embedding = tf.get_variable("embedding", [args.vocab_size, layer_size])
            # Create an embedding tensor with tf.nn.embedding_lookup(embedding,
self.input_data).
            # This tensor has dimensions batch_size x seq_length x layer_size.
            inputs = tf.nn.embedding_lookup(embedding, self.input_data)

        # TODO: Check arguments parallel_iterations (default uses more memory and less
time) and
        # swap_memory (default uses more memory but "minimal (or no) performance
penalty")
        outputs, self.final_state = tf.nn.dynamic_rnn(cell, inputs,
                initial_state=self.initial_state, scope='rnnlm')
        # outputs has shape [batch_size, max_time, cell.output_size] because time_major ==
false.
        # Do we need to transpose the first two dimensions? (Answer: no, this ruins
everything.)
        # outputs = tf.transpose(outputs, perm=[1, 0, 2])
        output = tf.reshape(outputs, [-1, layer_size])
        # Obtain logits node by applying output weights and biases to the output tensor.
        # Logits is a tensor of shape [(batch_size * seq_length) x vocab_size].
        # Recall that outputs is a 2D tensor of shape [(batch_size * seq_length) x layer_size],
        # and softmax_w is a 2D tensor of shape [layer_size x vocab_size].
        # The matrix product is therefore a new 2D tensor of [(batch_size * seq_length) x
vocab_size].
        # In other words, that multiplication converts a loooong list of layer_size vectors
        # to a loooong list of vocab_size vectors.
        # Then add softmax_b (a single vocab-sized vector) to every row of that list.
        # That gives you the logits!
        self.logits = tf.matmul(output, softmax_w) + softmax_b
        if infer:
            # Convert logits to probabilities. Probs isn't used during training! That node is never
calculated.
            # Like logits, probs is a tensor of shape [(batch_size * seq_length) x vocab_size].
            # During sampling, this means it is of shape [1 x vocab_size].
            self.probs = tf.nn.softmax(self.logits)
        else:
            # Create a targets placeholder of shape batch_size x seq_length.
            # Targets will be what output is compared against to calculate loss.
        self.targets = tf.placeholder(tf.int32, [args.batch_size, args.seq_length])
        # seq2seq.sequence_loss_by_example returns 1D float Tensor containing the log-
perplexity
        # for each sequence. (Size is batch_size * seq_length.)
        # Targets are reshaped from a [batch_size x seq_length] tensor to a 1D tensor, of the
following layout:
        #   target character (batch 0, seq 0)
        #   target character (batch 0, seq 1)
        #   ...
        #   target character (batch 0, seq seq_len-1)
        #   target character (batch 1, seq 0)
        #   ...
        # These targets are compared to the logits to generate loss.
        # Logits: instead of a list of character indices, it's a list of character index probability
vectors.
        # seq2seq.sequence_loss_by_example will do the work of generating losses by
comparing the one-hot vectors
        # implicitly represented by the target characters against the probability distrutions in
logits.
        # It returns a 1D float tensor (a vector) where item i is the log-perplexity of
        # the comparison of the ith logit distribution to the ith one-hot target vector.

        loss = nn_ops.sparse_softmax_cross_entropy_with_logits(
            labels=tf.reshape(self.targets, [-1]), logits=self.logits)

        # Cost is the arithmetic mean of the values of the loss tensor.
        # It is a single-element floating point tensor. This is what the optimizer seeks to
minimize.
        self.cost = tf.reduce_mean(loss)
        # Create a tensorboard summary of our cost.
        tf.summary.scalar("cost", self.cost)

        tvars = tf.trainable_variables() # tvars is a python list of all trainable TF Variable
objects.
        # tf.gradients returns a list of tensors of length len(tvars) where each tensor is
sum(dy/dx).
        grads, _ = tf.clip_by_global_norm(tf.gradients(self.cost, tvars),
                args.grad_clip)
        optimizer = tf.train.AdamOptimizer(self.lr) # Use ADAM optimizer.
        # Zip creates a list of tuples, where each tuple is (variable tensor, gradient tensor).
        # Training op nudges the variables along the gradient, with the given learning rate,
using the ADAM optimizer.
        # This is the op that a training session should be instructed to perform.
        self.train_op = optimizer.apply_gradients(zip(grads, tvars))
        #self.train_op = optimizer.minimize(self.cost)
        self.summary_op = tf.summary.merge_all()

    def add_state_to_feed_dict(self, feed_dict, state):
        for i, tensor in enumerate(flatten(state)):
            feed_dict[self._flattened_initial_state[i]] = tensor

    def save_variables_list(self):
        # Return a list of the trainable variables created within the rnnlm model.
        # This consists of the two projection softmax variables (softmax_w and softmax_b),
        # embedding, and all of the weights and biases in the MultiRNNCell model.
        # Save only the trainable variables and the placeholders needed to resume training;
        # discard the rest, including optimizer state.
        save_vars = set(tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES,
scope='rnnlm'))
        save_vars.update({self.lr, self.global_epoch_fraction, self.global_seconds_elapsed})
        return list(save_vars)

    def forward_model(self, sess, state, input_sample):
        '''Run a forward pass. Return the updated hidden state and the output probabilities.'''
        shaped_input = np.array([[input_sample]], np.float32)
        inputs = {self.input_data: shaped_input}
        self.add_state_to_feed_dict(inputs, state)
        [probs, state] = sess.run([self.probs, self.final_state], feed_dict=inputs)
        return probs[0], state

    def trainable_parameter_count(self):
        total_parameters = 0
        for variable in tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
scope='rnnlm'):
            shape = variable.get_shape()
            variable_parameters = 1
            for dim in shape:
                variable_parameters *= dim.value
            total_parameters += variable_parameters
        return total_parameters
```

- **Train.py**

```
import numpy as np
import tensorflow as tf

import argparse
import time, datetime
import os
import pickle
import sys

from utils import TextLoader
from model import Model

def main():
    assert sys.version_info >= (3, 3), \
    "Must be run in Python 3.3 or later. You are running {}".format(sys.version)

    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str, default='data/scotus',
                    help='data directory containing input.txt')
    parser.add_argument('--save_dir', type=str, default='models/new_save',
                    help='directory for checkpointed models (load from here if one is already
present)')
    parser.add_argument('--block_size', type=int, default=2048,
                    help='number of cells per block')
    parser.add_argument('--num_blocks', type=int, default=3,
                    help='number of blocks per layer')
    parser.add_argument('--num_layers', type=int, default=3,
                    help='number of layers')
    parser.add_argument('--model', type=str, default='gru',
                    help='rnn, gru, lstm or nas')
    parser.add_argument('--batch_size', type=int, default=40,
                    help='minibatch size')
    parser.add_argument('--seq_length', type=int, default=40,
                    help='RNN sequence length')
    parser.add_argument('--num_epochs', type=int, default=50,
                    help='number of epochs')
    parser.add_argument('--save_every', type=int, default=5000,
                    help='save frequency')
    parser.add_argument('--grad_clip', type=float, default=5.,
                    help='clip gradients at this value')
    parser.add_argument('--learning_rate', type=float, default=1e-5,
                    help='learning rate')
    parser.add_argument('--decay_rate', type=float, default=0.975,
                    help='how much to decay the learning rate')
    parser.add_argument('--decay_steps', type=int, default=100000,
                    help='how often to decay the learning rate')
    parser.add_argument('--set_learning_rate', type=float, default=-1,
                    help='reset learning rate to this value (if greater than zero)')
    args = parser.parse_args()
    train(args)

def train(args):
    # Create the data_loader object, which loads up all of our batches, vocab dictionary, etc.
    # from utils.py (and creates them if they don't already exist).
    # These files go in the data directory.
    data_loader = TextLoader(args.data_dir, args.batch_size, args.seq_length)
    args.vocab_size = data_loader.vocab_size

    load_model = False
    if not os.path.exists(args.save_dir):
        print("Creating directory %s" % args.save_dir)
        os.mkdir(args.save_dir)
    elif (os.path.exists(os.path.join(args.save_dir, 'config.pkl'))):
        # Trained model already exists
        ckpt = tf.train.get_checkpoint_state(args.save_dir)
        if ckpt and ckpt.model_checkpoint_path:
            with open(os.path.join(args.save_dir, 'config.pkl'), 'rb') as f:
                saved_args = pickle.load(f)
                args.block_size = saved_args.block_size
                args.num_blocks = saved_args.num_blocks
                args.num_layers = saved_args.num_layers
                args.model = saved_args.model
                print("Found a previous checkpoint. Overwriting model description arguments to:")
                print(" model: {}, block_size: {}, num_blocks: {}, num_layers: {}".format(
                    saved_args.model, saved_args.block_size, saved_args.num_blocks,
saved_args.num_layers))
                load_model = True

    # Save all arguments to config.pkl in the save directory -- NOT the data directory.
    with open(os.path.join(args.save_dir, 'config.pkl'), 'wb') as f:
        pickle.dump(args, f)
    # Save a tuple of the characters list and the vocab dictionary to chars_vocab.pkl in
    # the save directory -- NOT the data directory.
    with open(os.path.join(args.save_dir, 'chars_vocab.pkl'), 'wb') as f:
        pickle.dump((data_loader.chars, data_loader.vocab), f)

    # Create the model!
    print("Building the model")
    model = Model(args)
    print("Total trainable parameters: {:,d}".format(model.trainable_parameter_count()))

    # Make tensorflow less verbose; filter out info (1+) and warnings (2+) but not errors (3).
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```
config = tf.ConfigProto(log_device_placement=False)
#config.gpu_options.allow_growth = True
with tf.Session(config=config) as sess:
    tf.global_variables_initializer().run()
    saver = tf.train.Saver(model.save_variables_list(), max_to_keep=3)
    if (load_model):
        print("Loading saved parameters")
        saver.restore(sess, ckpt.model_checkpoint_path)
    global_epoch_fraction = sess.run(model.global_epoch_fraction)
    global_seconds_elapsed = sess.run(model.global_seconds_elapsed)
    if load_model: print("Resuming from global epoch fraction {:.3f},"
            " total trained time: {}, learning rate: {}".format(
            global_epoch_fraction,
            datetime.timedelta(seconds=float(global_seconds_elapsed)),
            sess.run(model.lr)))
    if (args.set_learning_rate > 0):
        sess.run(tf.assign(model.lr, args.set_learning_rate))
        print("Reset learning rate to {}".format(args.set_learning_rate))
    data_loader.cue_batch_pointer_to_epoch_fraction(global_epoch_fraction)
    initial_batch_step = int((global_epoch_fraction
            - int(global_epoch_fraction)) * data_loader.total_batch_count)
    epoch_range = (int(global_epoch_fraction),
            args.num_epochs + int(global_epoch_fraction))
    writer = tf.summary.FileWriter(args.save_dir, graph=tf.get_default_graph())
    outputs = [model.cost, model.final_state, model.train_op, model.summary_op]
    global_step = epoch_range[0] * data_loader.total_batch_count + initial_batch_step
    avg_loss = 0
    avg_steps = 0
    try:
        for e in range(*epoch_range):
            # e iterates through the training epochs.
            # Reset the model state, so it does not carry over from the end of the previous epoch.
            state = sess.run(model.zero_state)
            batch_range = (initial_batch_step, data_loader.total_batch_count)
            initial_batch_step = 0
            for b in range(*batch_range):
                global_step += 1
                if global_step % args.decay_steps == 0:
                    # Set the model.lr element of the model to track
                    # the appropriately decayed learning rate.
                    current_learning_rate = sess.run(model.lr)
                    current_learning_rate *= args.decay_rate
                    sess.run(tf.assign(model.lr, current_learning_rate))
                    print("Decayed learning rate to {}".format(current_learning_rate))
                start = time.time()
                # Pull the next batch inputs (x) and targets (y) from the data loader.
                x, y = data_loader.next_batch()

                # feed is a dictionary of variable references and respective values for
initialization.
                # Initialize the model's input data and target data from the batch,
                # and initialize the model state to the final state from the previous batch, so that
                # model state is accumulated and carried over between batches.
                feed = {model.input_data: x, model.targets: y}
                model.add_state_to_feed_dict(feed, state)

                # Run the session! Specifically, tell TensorFlow to compute the graph to calculate
                # the values of cost, final state, and the training op.
                # Cost is used to monitor progress.
                # Final state is used to carry over the state into the next batch.
                # Training op is not used, but we want it to be calculated, since that calculation
                # is what updates parameter states (i.e. that is where the training happens).
                train_loss, state, _, summary = sess.run(outputs, feed)
                elapsed = time.time() - start
                global_seconds_elapsed += elapsed
                writer.add_summary(summary, e * batch_range[1] + b + 1)
                if avg_steps < 100: avg_steps += 1
                avg_loss = 1 / avg_steps * train_loss + (1 - 1 / avg_steps) * avg_loss
                print("{:,d} / {:,d} (epoch {:.3f} / {}), loss {:.3f} (avg {:.3f}), {:.3f}s" \
                    .format(b, batch_range[1], e + b / batch_range[1], epoch_range[1],
                        train_loss, avg_loss, elapsed))
                # Every save_every batches, save the model to disk.
                # By default, only the five most recent checkpoint files are kept.
                if (e * batch_range[1] + b + 1) % args.save_every == 0 \
                        or (e == epoch_range[1] - 1 and b == batch_range[1] - 1):
                    save_model(sess, saver, model, args.save_dir, global_step,
                            data_loader.total_batch_count, global_seconds_elapsed)
        except KeyboardInterrupt:
            # Introduce a line break after ^C is displayed so save message
            # is on its own line.
            print()
        finally:
            writer.flush()
            global_step = e * data_loader.total_batch_count + b
            save_model(sess, saver, model, args.save_dir, global_step,
                    data_loader.total_batch_count, global_seconds_elapsed)

def save_model(sess, saver, model, save_dir, global_step, steps_per_epoch,
global_seconds_elapsed):
    global_epoch_fraction = float(global_step) / float(steps_per_epoch)
    checkpoint_path = os.path.join(save_dir, 'model.ckpt')
    print("Saving model to {} (epoch fraction {:.3f})...".format(checkpoint_path,
global_epoch_fraction),
        end='', flush=True)
    sess.run(tf.assign(model.global_epoch_fraction, global_epoch_fraction))
    sess.run(tf.assign(model.global_seconds_elapsed, global_seconds_elapsed))
    saver.save(sess, checkpoint_path, global_step = global_step)
    print("\rSaved model to {} (epoch fraction {:.3f}).   ".format(checkpoint_path,
global_epoch_fraction))

if __name__ == '__main__':
    main()
```

- **Index.html**

```
<!DOCTYPE html>
<html lang="en">

<head>
  <style>
    :root {
    --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
    --msger-bg: #fff;
    --border: 2px solid #ddd;
    --left-msg-bg: #ececec;
    --right-msg-bg: #579ffb;
  }

  html {
    box-sizing: border-box;
  }

  *,
  *:before,   *:after {
margin: 0;
padding: 0;     box-
sizing: inherit;
  }

  body {     display: flex;
color: #666;     justify-
content: center;     align-
items: center;     height:
100vh;
```

```
    background-image: var(--body-bg);
    font-family: Helvetica, sans-serif;

  }

  .msger {     display: flex;     flex-
flow: column wrap;     justify-
content: space-between;     width:
100%;     max-width: 867px;
margin: 25px 10px;     height:
calc(100% - 50px);     border:
var(--border);     border-radius:
5px; background: var(--msger-
bg);
    box-shadow: 0 15px 15px -5px rgba(0, 0, 0, 0.2);
  }

  .msger-header {   /* display:
  flex; */   font-size: medium;
  justify-content: space-between;
    padding: 10px;     text-align:
center;     border-bottom: var(--
border);     background: #eee;
color: #666;
  }

  .msger-chat {
flex: 1;
overflow-y: auto;
    padding: 10px;
  }
  .msger-chat::-webkit-scrollbar {
    width: 6px;
  }
  .msger-chat::-webkit-scrollbar-track {
background: #ddd;
  }
  .msger-chat::-webkit-scrollbar-thumb {
    background: #bdbdbd;
  }
  .msg {     display: flex;
align-items: flex-end;
    margin-bottom: 10px;
  }
  .msg-img {     width: 50px;
height: 50px;     margin-right:
10px;     background: #ddd;
background-repeat: no-repeat;
background-position: center;
background-size: cover;
    border-radius: 50%;
  }
  .msg-bubble {     max-
width: 450px;     padding:
15px;     border-radius:
15px; background: var(--
left-msg-bg);
  }
  .msg-info {   display: flex;
  justify-content: space-between;
  align-items: center;   margin-
  bottom: 10px;
  }   .msg-info-name {
margin-right: 10px;
    font-weight: bold;
  }
  .msg-info-time {     font-
size: 0.85em;
  }

  .left-msg .msg-bubble {
    border-bottom-left-radius: 0;
  }

  .right-msg {     flex-
direction: row-reverse;
  }
  .right-msg .msg-bubble {
background: var(--right-msg-bg);
color: #fff;     border-bottom-right-
radius: 0;
  }
  .right-msg .msg-img {
    margin: 0 0 0 10px;
  }

  .msgger-inputarea {
display: flex;     padding:
10px;     border-top: var(--
border);
```

```
    background: #eee;
  }
  .msger-inputarea * {
padding: 10px;     border:
none;     border-radius:
3px;     font-size: 1em;
  }
  .msger-input {
flex: 1;
    background: #ddd;
  }
  .msger-send-btn {
    margin-left: 10px;
    background: rgb(0, 196, 65);
    color: #fff;   font-
  weight: bold;   cursor:
  pointer;
    transition: background 0.23s;
  }
  .msger-send-btn:hover {
background: rgb(0, 180, 50);
  }

  .msger-chat {
    background-color: #fcfcfe;
    background-image: url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' width='260' height='260' viewBox='0 0 260
260'%3E%3Cg fill-rule='evenodd'%3E%3Cg fill='%23dddddd'
fillopacity='0.4'%3E%3Cpath d='M24.37 16c.2.65.39 1.32.54 2H21.17l1.17
2.34.45.9.24.11V28a5 5 0 0 1-2.23 8.94l-.02.06a8 8 0 0 1-7.75 6h-20a8 8 0 0 1-7.74-6l-.02-
.06A5 5 0
0 1-17.45 28v-6.76l-.79-1.58-.44-.9.9-.44.63-.32H-20a23.01 23.01 0 0 1 44.37-2zm-36.82
2a1 1 0 0 0-.44.1l-3.1 1.56.89 1.79 1.31-.66a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0 .9 0l2.21-
1.1a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0 .9 0l2.21-1.1a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0
.86.02l2.88-1.27a3 3 0 0 1 2.43 0l2.88 1.27a1 1 0 0 0 .85-.02l3.1-1.55-.89-1.79-1.42.71a3 3
0 0 1-2.56.06l-2.77-1.23a1 1 0 0 0-.4-.09h-.01a1 1 0 0 0-.4.09l-2.78 1.23a3 3 0 0 1-2.56-
.06l-2.3-1.15a1 1 0 0 0-.45-.11h-.01a1 1 0 0 0-.44.1L.9 19.22a3 3 0 0 1-2.69 0l-2.2-1.1a1 1
0 0 0-.45-.11h-.01a1 1 0 0 0-.44.1l-2.21 1.11a3 3 0 0 1-2.69 0l-2.2-1.1a1 1 0 0 0-.45-
.11h.01zm0-2h-4.9a21.01 21.01 0 0 1 39.61 0h-2.09l-.06-.13-.26.13h-32.31zm30.35
7.68l1.36-
.68h1.3v2h-36v-1.15l.34-.17 1.36-.68h2.59l1.36.68a3 3 0 0 0 2.69 0l1.36-
.68h2.59l1.36.68a3 3 0 0 0 2.69 0L2.26 23h2.59l1.36.68a3 3 0 0 0
2.56.06l1.67.74h3.23l1.67.74a3 3 0 0 0 2.56-.06zM-13.82 27l16.37 4.91L18.93 27h-
32.75zm-.63
2h.34l16.66 5 16.67-5h.33a3 3 0 1 1 0 6h-34a3 3 0 1 1 0-6zm1.35 8a6 6 0 0 0 5.65 4h20a6 6
```

0 0 0 5.66-4H-13.1z'/%3E%3Cpath id='path6_fill-copy' d='M284.37 16c.2.65.39 1.32.54
2H281.17l1.17 2.34.45.9-.24.11V28a5 5 0 0 1-2.23 8.94l-.02.06a8 8 0 0 1-7.75 6h-20a8 8 0
0 1-7.74-6l-.02-.06a5 5 0 0 1-2.24-8.94v-6.76l-.79-1.58-.44-.9.9-.44.63-.32H240a23.01
23.01 0 0 1 44.37-2zm-36.82 2a1 1 0 0 0-.44.1l-3.1 1.56.89 1.79 1.31-.66a3 3 0 0 1 2.69
0l2.2 1.1a1 1 0 0 0 .9 0l2.21-1.1a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0 .9 0l2.21-1.1a3 3 0 0 1
2.69 0l2.2 1.1a1 1 0 0 0 .86.02l2.88-1.27a3 3 0 0 1 2.43 0l2.88 1.27a1 1 0 0 0 .85-
.02l3.11.55-.89-1.79-1.42.71a3 3 0 0 1-2.56.06l-2.77-1.23a1 1 0 0 0-.4-.09h-.01a1 1 0 0 0-
.4.09l2.78 1.23a3 3 0 0 1-2.56-.06l-2.3-1.15a1 1 0 0 0-.45-.11h-.01a1 1 0 0 0-.44.1l-2.21
1.11a3 3 0 0 1-2.69 0l-2.2-1.1a1 1 0 0 0-.45-.11h-.01a1 1 0 0 0-.44.1l-2.21 1.11a3 3 0 0 1-
2.69 0l-2.2-
1.1a1 1 0 0 0-.45-.11h-.01zm0-2h-4.9a21.01 21.01 0 0 1 39.61 0h-2.09l-.06-.13-
.26.13h32.31zm30.35 7.68l1.36-.68h1.3v2h-36v-1.15l.34-.17 1.36-.68h2.59l1.36.68a3 3 0 0
0 2.69
0l1.36-.68h2.59l1.36.68a3 3 0 0 0 2.69 0l1.36-.68h2.59l1.36.68a3 3 0 0 0
2.56.06l1.67.74h3.23l1.67.74a3 3 0 0 0 2.56-.06zM246.18 27l16.37 4.91L278.93 27h-
32.75zm-.63
2h.34l16.66 5 16.67-5h.33a3 3 0 1 1 0 6h-34a3 3 0 1 1 0-6zm1.35 8a6 6 0 0 0 5.65 4h20a6 6
0 0 0 5.66-4H246.9z'/%3E%3Cpath d='M159.5 21.02A9 9 0 0 0 151 15h-42a9 9 0 0 0-8.5
6.02 6 6 0 0 0 .02 11.96A8.99 8.99 0 0 0 109 45h42a9 9 0 0 0 8.48-12.02 6 6 0 0 0
.0211.96zM151 17h-42a7 7 0 0 0-6.33 4h54.66a7 7 0 0 0-6.33-4zm-9.34 26a8.98 8.98 0 0 0
3.34-7h-2a7 7 0 0 1-7 7h-4.34a8.98 8.98 0 0 0 3.34-7h-2a7 7 0 0 1-7 7h-4.34a8.98 8.98 0 0
0 3.34-7h-2a7 7 0 0 1-7 7h-7a7 7 0 1 1 0-14h42a7 7 0 1 1 0 14h-9.34zM109 27a9 9 0 0 0-
7.48 4H101a4 4 0 1 1 0-8h58a4 4 0 0 1 0 8h-.52a9 9 0 0 0-7.48-4h-42z'/%3E%3Cpath
d='M39 115a8 8 0 1 0 0-16 8 8 0 0 0 0 16zm6-8a6 6 0 1 1-12 0 6 6 0 0 1 12 0zm-3-
29v2h8v-6H40a4 4 0 0 0-4 4v10H22l-1.33 4-.67 2h2.19L26 130h26l3.81-40H58l-.67-2L56
84H42v-6zm-4-4v10h2V74h8v-2h-8a2 2 0 0 0-2 2zm2 12h14.56l.67
2H22.77l.672H40zm13.8 4H24.2l3.62 38h22.36l3.62-38z'/%3E%3Cpath d='M129 92h-
6v4h-6v4h-
6v14h-3l.24 2 3.76 32h36l3.76-32 .24-2h-3v-14h-6v-4h-6v-4h-8zm18 22v-12h-
4v4h3v8h1zm-3 0v-6h-4v6h4zm-6 6v-16h-4v19.17c1.6-.7 2.97-1.8 4-3.17zm-6
3.8V100h4v23.8a10.04 10.04 0 0 0 4 0zm-6-.63V104h-4v16a10.04 10.04 0 0 0 4 3.17zm-6-
9.17v-6h-
4v6h4zm-6 0v-8h3v-4h-4v12h1zm27-12v-4h-4v4h3v4h1v-4zm-6 0v-8h-4v4h3v4h1zm-6-
4v-4h-4v8h1v-4h3zm-6 4v-4h-4v8h1v-4h3zm7 24a12 12 0 0 0 11.83-10h7.92l-3.53
30h32.44l-3.53-30h7.92A12 12 0 0 0 130 126z'/%3E%3Cpath d='M212 86v2h-4v-2h4zm4
0h-
2v2h2v-2zm-20 0v.1a5 5 0 0 0-.56 9.65l.06.25 1.12 4.48a2 2 0 0 0 1.94 1.52h.01l7.02
24.55a2 2 0 0 0 1.92 1.45h4.98a2 2 0 0 0 1.92-1.45l7.02-24.55a2 2 0 0 0 1.95-1.52L224.5
96l.06-.25a5 5 0 0 0-.56-9.65V86a14 14 0 0 0-28 0zm4 0h6v2h-9a3 3 0 1 0 0 6H223a3 3 0
1 0 0-6H220v-2h2a12 12 0 1 0-24 0h2zm-1.44 14l-1-4h24.88l-1 4h-22.88zm8.95 26l-
6.8624h18.7l-6.86 24h-4.98zM150 242a22 22 0 1 0 0-44 22 22 0 0 0 0 44zm24-22a24 24 0
1 148 0 24 24 0 0 1 48 0zm-28.38 17.73l2.04-.87a6 6 0 0 1 4.68 0l2.04.87a2 2 0 0 0 2.5-
.82l1.14-1.9a6 6 0 0 1 3.79-2.75l2.15-.5a2 2 0 0 0 1.54-2.12l-.19-2.2a6 6 0 0 1
1.454.46l1.45-1.67a2 2 0 0 0 0-2.62l-1.45-1.67a6 6 0 0 1-1.45-4.46l.2-2.2a2 2 0 0 0-1.55-
2.13l2.15-.5a6 6 0 0 1-3.8-2.75l-1.13-1.9a2 2 0 0 0-2.5-.8l-2.04.86a6 6 0 0 1-4.68 0l-2.04-
.87a2 2

0 0 0-2.5.82l-1.14 1.9a6 6 0 0 1-3.79 2.75l-2.15.5a2 2 0 0 0-1.54 2.12l.19 2.2a6 6 0 0 1-1.45
4.46l-1.45 1.67a2 2 0 0 0 0 2.62l1.45 1.67a6 6 0 0 1 1.45 4.46l-.2 2.2a2 2 0 0 0 1.55
2.13l2.15.5a6 6 0 0 1 3.8 2.75l1.13 1.9a2 2 0 0 0 2.5.8zm2.82.97a4 4 0 0 1 3.12 0l2.04.87a4
4 0 0 0 4.99-1.62l1.14-1.9a4 4 0 0 1 2.53-1.84l2.15-.5a4 4 0 0 0 3.09-4.24l-.2-2.2a4 4 0 0 1
.97-2.98l1.45-1.67a4 4 0 0 0 0-5.24l-1.45-1.67a4 4 0 0 1-.97-2.97l.2-2.2a4 4 0 0 0-3.09-
4.25l-2.15-.5a4 4 0 0 1-2.53-1.84l-1.14-1.9a4 4 0 0 0-5-1.62l-2.03.87a4 4 0 0 1-3.12 0l2.04-
.87a4 4 0 0 0-4.99 1.62l-1.14 1.9a4 4 0 0 1-2.53 1.84l-2.15.5a4 4 0 0 0-3.09 4.24l.2
2.2a4 4 0 0 1-.97 2.98l-1.45 1.67a4 4 0 0 0 0 5.24l1.45 1.67a4 4 0 0 1 .97 2.97l-.2 2.2a4 4 0
0 0 3.09 4.25l2.15.5a4 4 0 0 1 2.53 1.84l1.14 1.9a4 4 0 0 0 5 1.62l2.03-.87zM152 207a1 1 0
1 1 2 0 1 1 0 0 1-2 0zm6 2a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-11 1a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-
6 0a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm3-5a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-8 8a1 1 0 1 1 2 0 1 1 0 0
1-2 0zm3 6a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm0 6a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm4 7a1 1 0 1 1 2 0
1 1 0 0 1-2 0zm5-2a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm5 4a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm4-6a1 1 0
1 1 2 0 1 1 0 0 1-2 0zm6-4a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-4-3a1 1 0 1 1 2 0 1 1 0 0 1-2
0zm4-3a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-5-4a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-24 6a1 1 0 1 1 2 0 1
1 0 0 1-2 0zm16 5a5 5 0 1 0 0-10 5 5 0 0 0 0 10zm7-5a7 7 0 1 1-14 0 7 7 0 0 1 14 0zm86-
29a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm19 9a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-
11zm-14 5a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm-25 1a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm5
4a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm9 0a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-11zm15
1a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm12-2a1 1 0 0 0 0 2h2a1 1 0 0 0 02h-2zm-
11-14a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm-19 0a1 1 0 0 0 0 2h2a1 1 0
0 0 0-2h-2zm6 5a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm-25 15c0-.47.01-
.94.031.4a5 5 0 0 1-1.7-8 3.99 3.99 0 0 1 1.88-5.18 5 5 0 0 1 3.4-6.22 3 3 0 0 1 1.46-1.05 5 5
0 0 1 7.76-3.27A30.86 30.86 0 0 1 246 184c6.79 0 13.06 2.18 18.17 5.88a5 5 0 0 1 7.76
3.27 3 3
0 0 1 1.47 1.05 5 5 0 0 1 3.4 6.22 4 4 0 0 1 1.87 5.18 4.98 4.98 0 0 1-1.7 8c.02.46.03.93.03
1.4v1h-62v-1zm.83-7.17a30.9 30.9 0 0 0-.62 3.57 3 3 0 0 1-.61-4.2c.37.28.78.49
1.23.63zm1.49-4.61c-.36.87-.68 1.76-.96 2.68a2 2 0 0 1-.21-3.71c.33.4.73.75 1.17
1.03zm2.32-4.54c-.54.86-1.03 1.76-1.49 2.68a3 3 0 0 1-.07-4.67 3 3 0 0 0 1.56 1.99zm1.14-
1.7c.35-.5.72-.98 1.1-1.46a1 1 0 1 0-1.1 1.45zm5.34-5.77c-1.03.86-2 1.79-2.9 2.77a3 3 0 0
0-1.11-.77 3 3 0 0 1 4-2zm42.66 2.77c-.9-.98-1.87-1.9-2.9-2.77a3 3 0 0 1 4.01 2 3 3 0 0
01.1.77zm1.34 1.54c.38.48.75.96 1.1 1.45a1 1 0 1 0-1.1-1.45zm3.73 5.84c-.46-.92-.95-
1.821.5-2.68a3 3 0 0 0 1.57-1.99 3 3 0 0 1-.07 4.67zm1.8 4.53c-.29-.9-.6-1.8-.97-2.67.44-
.28.84.63 1.17-1.03a2 2 0 0 1-.2 3.7zm1.14 5.51c-.14-1.21-.35-2.4-.62-3.57.45-.14.86-.35
1.23.63a2.99 2.99 0 0 1-.6 4.2zM275 214a29 29 0 0 0-57.97 0h57.96zM72.33 198.12c-.21-
.32-
.34-.7-.34-1.12v-12h-2v12a4.01 4.01 0 0 0 7.09 2.54c.57-.69.91-1.57.91-2.54v-
12h2v12a1.99 1.99 0 0 1-2 2 2 2 0 0 1-1.66-.88zM75 176c.38 0 .74-.04 1.1-.12a4 4 0 0 0
6.19
2.4A13.94 13.94 0 0 1 84 185v24a6 6 0 0 1-6 6h-3v9a5 5 0 1 1-10 0v-9h-3a6 6 0 0 1-6-6v-
24a14 14 0 0 1 14-14 5 5 0 0 0 5 5zm-17 15v12a1.99 1.99 0 0 0 1.22 1.84 2 2 0 0 0 2.44-
.72c.21-.32.34-.7.34-1.12v-12h2v12a3.98 3.98 0 0 1-5.35 3.77 3.98 3.98 0 0 1-.65.3V209a4
4 0 0 0 4 4h16a4 4 0 0 0 4-4v-24c.01-1.53-.23-2.88-.72-4.17-.43.1-.87.161.28.17a6 6 0 0 1-
5.2-3 7 7 0 0 1-6.47-4.88A12 12 0 0 0 58 185v6zm9 24v9a3 3 0 1 0 6 0v9h-

```
6z'/%3E%3Cpath d='M-17 191a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm19 9a1 1 0 0 1 11h2a1 1
0 0 1 0 2H3a1 1 0 0 1-1-1zm-14 5a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm-25 1a1 1 0 0
0 0 2h2a1 1 0 0 0 0-2h-2zm5 4a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm9 0a1 1 0 0 1 1-1h2a1 1 0
0 1 0 2h-2a1 1 0 0 1-1-1zm15 1a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm12-2a1 1 0
0 0 0 2h2a1 1 0 0 0 0-2H4zm-11-14a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm-19
0a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm6 5a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-
1zm25 15c0-.47.01-.94.03-1.4a5 5 0 0 1-1.7-8 3.99 3.99 0 0 1 1.88-5.18 5 5 0 0 1 3.4-6.22 3
3 0
0 1 1.46-1.05 5 5 0 0 1 7.76-3.27A30.86 30.86 0 0 1-14 184c6.79 0 13.06 2.18 18.17 5.88a5
5 0 0 1 7.76 3.27 3 3 0 0 1 1.47 1.05 5 5 0 0 1 3.4 6.22 4 4 0 0 1 1.87 5.18 4.98 4.98 0 0 11.7
8c.02.46.03.93.03 1.4v1h-62v-1zm.83-7.17a30.9 30.9 0 0 0-.62 3.57 3 3 0 0 1-
.614.2c.37.28.78.49 1.23.63zm1.49-4.61c-.36.87-.68 1.76-.96 2.68a2 2 0 0 1-.21-
3.71c.33.4.73.75 1.17 1.03zm2.32-4.54c-.54.86-1.03 1.76-1.49 2.68a3 3 0 0 1-.07-4.67 3 3 0
0 0 1.56 1.99zm1.14-1.7c.35-.5.72-.98 1.1-1.46a1 1 0 1 0-1.1 1.45zm5.34-5.77c-1.03.86-2
1.79-2.9 2.77a3 3 0 0 0-1.11-.77 3 3 0 0 1 4-2zm42.66 2.77c-.9-.98-1.87-1.9-2.9-2.77a3 3 0
0 1 4.01 2 3 3 0 0 0-1.1.77zm1.34 1.54c.38.48.75.96 1.1 1.45a1 1 0 1 0-1.1-1.45zm3.73
5.84c-.46-.92-.95-1.82-1.5-2.68a3 3 0 0 0 1.57-1.99 3 3 0 0 1-.07 4.67zm1.8 4.53c-.29-.9-
.61.8-.97-2.67.44-.28.84-.63 1.17-1.03a2 2 0 0 1-.2 3.7zm1.14 5.51c-.14-1.21-.35-2.4-
.623.57.45-.14.86-.35 1.23-.63a2.99 2.99 0 0 1-.6 4.2zM15 214a29 29 0 0 0-57.97
0h57.96z'/%3E%3C/g%3E%3C/g%3E%3C/svg%3E");
  }

  </style>

  <meta charset="UTF-8">
  <title>Chat_Bot</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
var dt = new Date();
 document.getElementById('date-time').innerHTML=dt
  </script>
</head>

<body>
 <!-- partial:index.partial.html --> <section
 class="msger">
   <header class="msger-header">
     <div class="msger-header-title">
       </i> Inbox </i>
     </div>
   </header>

   <main class="msger-chat">
```

```
     <div class="msg left-msg">
       <div class="msg-img" style="background-image:
url(https://image.flaticon.com/icons/svg/327/327779.svg)"></div>

       <div class="msg-bubble">
         <div class="msg-info">
           <div class="msg-info-name">Sia</div>
           <div class="msg-info-time" id="date-time"></div>
</div>

         <div class="msg-text">
           Hi, welcome! Go ahead and send me a message.
         </div>
       </div>
      </div>

    </main>

    <form class="msger-inputarea">
      <input type="text" class="msger-input" id="textInput" placeholder="Enter your
message...">
      <button type="submit" class="msger-send-btn">Send</button>
    </form>
  </section>
  <!-- partial -->
  <script src='https://use.fontawesome.com/releases/v5.0.13/js/all.js'></script>
<script>

    const msgerForm = get(".msger-inputarea");
const msgerInput = get(".msger-input");
    const msgerChat = get(".msger-chat");



    const BOT_IMG = "https://www.pinpng.com/pngs/m/53-532121_source-
wwwantrixacademy-com-report-female-female-person.png";
    const PERSON_IMG = "https://www.pinpng.com/pngs/m/53-531868_person-icon-
pngtransparent-png.png";     const BOT_NAME = "Sia";
    const PERSON_NAME = "You";

   msgerForm.addEventListener("submit", event => {
      event.preventDefault();
     const msgText = msgerInput.value;
if (!msgText) return;
```

```
    appendMessage(PERSON_NAME, PERSON_IMG, "right", msgText);
msgerInput.value = "";
    botResponse(msgText);
  });

   function appendMessage(name, img, side, text) {
     //   Simple solution for small apps
const msgHTML = ` <div
class="msg ${side}-msg">
  <div class="msg-img" style="background-image: url(${img})"></div>

  <div class="msg-bubble">
    <div class="msg-info">
      <div class="msg-info-name">${name}</div>
      <div class="msg-info-time">${formatDate(new Date())}</div>
    </div>

    <div class="msg-text">${text}</div>
  </div>
</div>
`;

     msgerChat.insertAdjacentHTML("beforeend", msgHTML);
     msgerChat.scrollTop += 500;
    }

   function botResponse(rawText) {

     // Bot Response
     $.get("/get", { msg: rawText }).done(function (data) {
console.log(rawText);         console.log(data);
       const msgText = data;
       appendMessage(BOT_NAME, BOT_IMG, "left", msgText);

     });

    }

   // Utils
   function get(selector, root = document) {
     return root.querySelector(selector);
    }

   function formatDate(date) {
    const h = "0" + date.getHours();
const m = "0" + date.getMinutes();

     return `${h.slice(-2)}:${m.slice(-2)}`;
   }


 </script>

</body>

</html>
```

MILESTONE

Sr. No.	Project Activity	Estimated Start Date	Estimated End Date
1	Synopsis	20 March, 2022	24 March, 2022 (submitted)
2	Project Completed (75%)	26 April, 2022	26 April, 2022 (still in progress)
3	Repost Submission	26 April, 2022	26 April, 2022 (submitted)

MEETING WITH SUPERVISOR

Date of the Meet	Mode	Remarks	Supervisors' Sign
18 March 2022	Offline	Search and implement some idea	
21 March 2022	Offline	Discussed issues	
25 March 2022	Offline	Checked progress	
06 April 2022	Offline	Reconstruction and improvement Idea	
09 April 2022	Offline	Received valuable suggestion	
16 April 2022	Offline	Checked progress and presentation of system	
26 April 2022	Offline	Project presentation	

References:

1. Natalya N. Bazarova, Yoon Hyung Choi, Victoria Schwanda Sosik, Dan Cosley, and Janis Whitlock. Social Sharing of Emotions on Facebook: Channel Differences, Satisfaction, and Replies. In Proc. of CSCW, 2015, 154-164.

2. Joan-Isaac Biel, Oya Aran, and Daniel Gatica-Perez. You Are Known by How You Vlog: Personality Impressions and Nonverbal Behavior in Youtube. In Proc. of ICWSM, 2011, 446-449.

3. Keith S Coulter, Johanna Gummerus, Veronica Liljander, Emil Weman, and Minna Pihlström. Customer Engagement in a Facebook Brand Community. Management Research Review, 2012, 35, 9, 857-877.

4. Ethan Fast, Binbin Chen, and Michael S. Bernstein. Empath: Understanding Topic Signals in Large-Scale Text. In Proc. of CHI, 2016, 4647-4657.

5. F. Maxwell Harper, Daniel Moy, and Joseph A. Konstan. Facts or Friends?: Distinguishing Informational and Conversational Questions in Social Q&A Sites. In Proc. of CHI, 2009, 759-768.

6. F. Maxwell Harper, Daphne Raban, Sheizaf Rafaeli, and Joseph A. Konstan. Predictors of Answer Quality in Online Q&A Sites. In Proc. of CHI, 2008, 865-874.

7. Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, and Vivek Ramavajjala. Smart Reply: Automated Response Suggestion for Email. In Proc. of KDD, 2016, 955-964.

8. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and Their Compositionality. In Proc. of NIPS, 2013, 3111-3119.

9. Keith B Murray. A Test of Services Marketing Theory: Consumer Information Acquisition Activities. The Journal of Marketing, 1991, 10-25.

10. NIELSEN. State of the Media: Social Media Report. 2011.

11. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A Method for Automatic Evaluation of Machine Translation. In Proc. of ACL, 2002, 311-318.

12. James W Pennebaker, Martha E Francis, and Roger J Booth. Linguistic Inquiry and Word Count: Liwc 2001. Mathway: Lawrence Erlbaum Associates, 2001, 71, 2001.

13. Tiziano Piccardi, Gregorio Convertino, Massimo Zancanaro, Ji Wang, and Cedric Archambeau. Towards Crowd-Based Customer Service: A Mixed-Initiative Tool for Managing Q&a Sites. In Proc. of CHI, 2014, 2725-2734.

14. Leyland F Pitt, Richard T Watson, and C Bruce Kavan. Service Quality: A Measure of Information Systems Effectiveness. Management Information Systems Quarterly, 1995, 173-187.

15. Alan Ritter, Colin Cherry, and William B Dolan. DataDriven Response Generation in Social Media. In Proc. of EMNLP, 2011, 583-593.

16. Anselm L Strauss Qualitative Analysis for Social Scientists. Cambridge University Press, 1987.

17. Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In Proc. of NIPS, 2014, 3104-3112.

18. http://blog.hubspot.com/marketing/twitter-responsetime-data.

19. http://lucene.apache.org

20. https://www.reddit.com/r/datasets/comments/65o7py/updated_reddit_comment_dataset_as_torrents/