EDUCATIONAL CHATBOT USING WEB SCRAPED DATA FOR INSTITUTE

Submitted in partial fulfillment of the
Requirement for the Master's degree
in
Information Technology

By
KEERTHANA P
21/PCSA/121

November 2022 – April 2023

Stella Maris College (Autonomous)
17, Cathedral Road,
Chennai-600086.

STELLA MARIS COLLEGE (Autonomous)
17, Cathedral Road,
Chennai-600086.
Master of Science (Information Technology)
(Affiliated to University of Madras)

BONAFIDE CERTIFICATE

This is to certify that this is a bonafide record of the project done by
KEERTHANA P

on
EDUCATIONAL CHATBOT USING WEB SCRAPED DATA FOR INSTITUTE
at
Stella Maris College (Autonomous)
November 2022- April 2023

This project was done by her in partial fulfillment of the requirements for the Master's degree in Information Technology

Head of Department **Internal Guide** **External Examiner**

ACKNOWLEDGEMENT

First of all, I am deeply indebted to God the Almighty for being the source of my strength, guide, confidence and inspiration.

With deep gratitude I sincerely thank my institution Stella Maris College that has given me the opportunity and confidence to complete my research project in this institution.

I express my deep sense of gratitude to my principal **Dr. (Sr.) Rosy Joseph, fmm, M.Sc., M.Phil., Ph.D.,** Stella Maris College, for the facilities provided by her in carrying out this work.

I would also like to thank my Head of the Department, **Ms. Blessy Boaz, M.Sc., M.Phil.,** and faculties of Department of computer Science for their constant support throughout this project.

I would like to sincerely thank my project guide, **Ms. Jeyapriya U, M.C.A., M.Phil., NET.,** for her constant support and encouragement without which I wouldn't have completed this project successfully.

KEERTHANA P

PLACE: Chennai

DATE:

ABSTRACT

This project aims to develop an educational AI chatbot to be used in the institute. AI chatbots use natural language processing (NLP) to help users to interact with web services or apps through text, graphics, or speech. Chatbots can understand natural human language, emulate human conversation, and automated regular tasks. The chatbot proposed in this project is designed to help students to get their queries solved on fingertips. Web scraping and transfer-learning are the techniques incorporated in developing this chatbot. This chatbot is advantageous over others due to its simplicity in terms of data preprocessing, training and testing. However, on the other hand, there are a few downsides such as, more time and space consumption.

Keywords: AI Chatbot, BERT, DPR, NLP, Web Scraping

ABBREVIATIONS

Abbreviations	Meaning
BERT	Bidirectional Encoder Representations from Transformers
DPR	Dense Passage Retriever
SQuAD	Stanford Question Answer Dataset
Cos_sim:	Cosine Similarity
NLP	Natural Language Processing

TABLE OF CONTENTS

LIST OF FIGURES

1.INTRODUCTION

In recent times, a growing number of studies have explored the ways and effects of chatbot applications in education. Several studies have revealed the benefits of using chatbots in academy settings. With the advancement of Artificial Intelligence (AI) technology, scholars have begun to apply machine learning and natural language technology to the development of chatbots, making their application in education a new topic of academic research. New technologies will enable chatbots to become smart guiding assistants in the future. This project implements a generative chatbots that is smarter than retrieval based chatbot, retrieval-based chatbot are based on predefined statements, where the responses are selected from a pool of responses. The generative chatbot is capable of understanding and processing the user query and generating appropriate responses. This chatbot extracts data from the website using python package Beautifulsoup, encoded them using DPR (dense passage retriever) model and store them in database. This chatbot is developed using transfer-learning model BERT (Bidirectional Encoder Representations from Transformers) which are finetuned on SQuAD dataset (Stanford Question Answer Dataset). This chatbot is developed as a web application using flask, ajax for backend and html, css, javascript for frontend. In addition, System stores the conversations history in csv file.

2. SYSTEM ANALYSIS

2.1 ABOUT THE TOOL

GOOGLE COLABORATORY(G-Colab):

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Colab notebooks are Jupyter notebooks that are hosted by Colab. Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **LaTeX** and more.

VISUAL STUDIO CODE (VS Code):

VS Code is a free code editor, which runs on the macOS, Linux, and Windows operating systems. VS Code is lightweight and should run on most available hardware and platform versions.

2.2 REQUIREMENT SPECIFICATION

VISUAL STUDIO CODE (VS Code): Version: 1.77.1

Hardware

Visual Studio Code is a small download (< 200 MB) and has a disk footprint of < 500 MB. VS Code is lightweight and should easily run on today's hardware.

recommended:

- 1.6 GHz or faster processor
- 1 GB of RAM

Platforms

VS Code is supported on the following platforms:

- Windows 8.0, 8.1 and 10, 11 (32-bit and 64-bit)
- OS X High Sierra (10.13+)
- Linux (Debian): Ubuntu Desktop 16.04, Debian 9
- Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 7, Fedora 34

GOOGLE COLABORATORY

The VM used for Colaboratory appears to have 13GB RAM and 2 vCPU when checking using psutil (so a n1-highmem-2 instance)

- 2-core Xeon 2.2GHz
- 25GB RAM
- 107GB Disk

maximum lifetime of a VM is 12 hours. Idle VMs time out after 90m.

PYTHON 3.10.11

Python is a programming language that lets you work quickly and integrate systems more effectively.

- **DPR**: Dense Passage Retrieval (DPR) is a set of tools and models for state-of-the-art open-domain Q&A research. It was introduced in Dense Passage Retrieval for Open-Domain Question Answering by Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, Wen-tau Yih.
  - **DPRContextEncoder: The bare** DPRContextEncoder transformer outputting pooler outputs as context representations.
  - **DPRContextEncoderTokenizer**: Construct a DPRContextEncoder tokenizer.

- **DPRQuestionEncoder:** The bare DPRQuestionEncoder transformer outputting pooler outputs as question representations.
- **DPRQuestionEncoderTokenizer:** Constructs a DPRQuestionEncoder tokenizer.
- performance of the abouve model on retrieving top-k contexts accuracy (k ∈ {20, 100}) of SQuAD v1.1. dataset. Result for top-20 is 63.2 and top-100 is 77.2.

- **sentence_transformers.util :** This framework provides an easy method to compute dense vector representations for **sentences**, **paragraphs**, and **images**. The models are based on transformer networks like BERT / RoBERTa / XLM-RoBERTa etc. and achieve state-of-the-art performance in various tasks. Text is embedded in vector space such that similar text is close and can efficiently be found using cosine similarity.
- **Pickle:** The pickle module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.
- **Transformers (BertForQuestionAnswering & BertTokenizer):**

  BERT is a transformers model pretrained on a large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts. More precisely, it was pretrained with two objectives:

  - Masked language modeling (MLM): taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words. This is different from traditional

recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally mask the future tokens. It allows the model to learn a bidirectional representation of the sentence.

- Next sentence prediction (NSP): the models concatenates two masked sentences as inputs during pretraining. Sometimes they correspond to sentences that were next to each other in the original text, sometimes not. The model then has to predict if the two sentences were following each other or not.

This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks: if you have a dataset of labeled sentences for instance, you can train a standard classifier using the features produced by the BERT model as inputs.

This model has the following configuration:

- 24-layer
- 1024 hidden dimension
- 16 attention heads
- 336M parameters.

Pretrained model on English language using a masked language modeling (MLM) objective. Differently to other BERT models, this model was trained with a new technique: Whole Word Masking. In this case, all of the tokens corresponding to a word are masked at once. The overall masking rate remains the same. The training is identical -- each masked WordPiece token is predicted independently After pre-training, this model was fine-tuned on the SQuAD dataset with one of our fine-tuning scripts. See below for more information regarding this fine-tuning.

Evaluation: The results obtained are the following:

- f1 = 93.15
- exact_match = 86.91

- **Flask:**

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja has become one of the most popular Python web application frameworks.

Flask offers suggestions but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

3.SYSTEM DESIGN

3.1 PROPOSED WORK

The proposed system aims at creating an educational chatbot model to answer students' queries using the institute's web content. In this BERT and DPR transformer model are used for encoding data

3.2 METHODOLOGY

The process flow for Education AI Chatbot is given below.

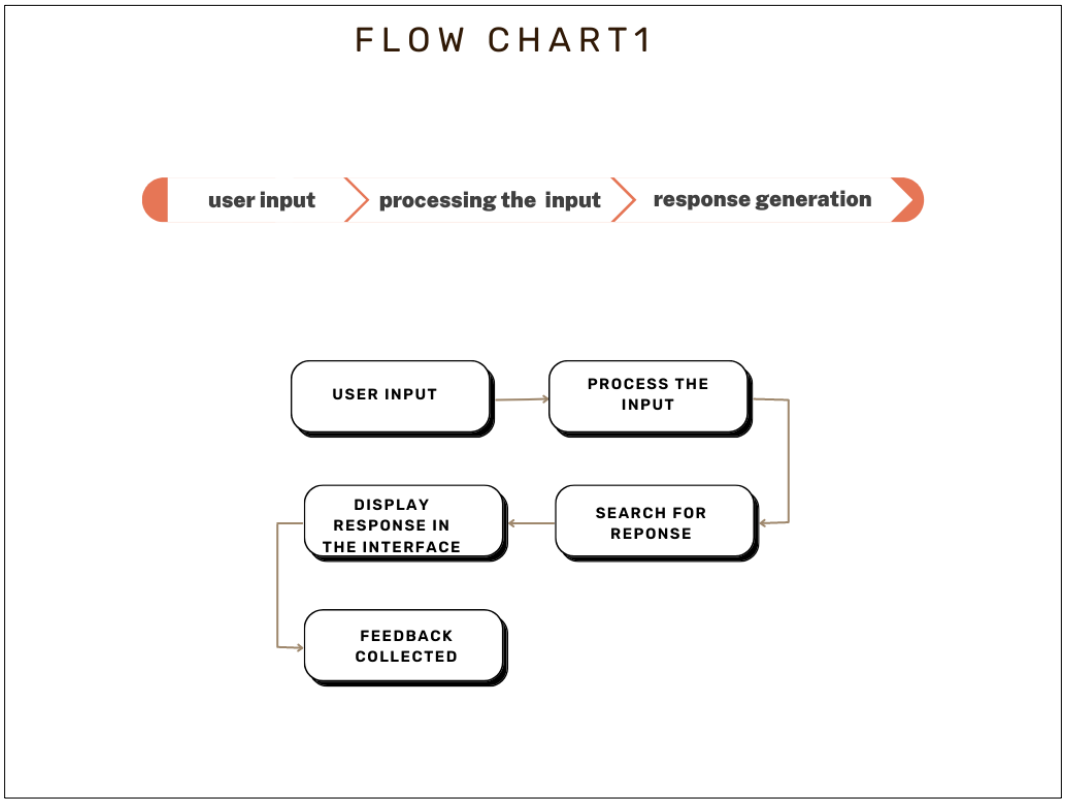

Figure 1: Process Flow of SMC-BOT

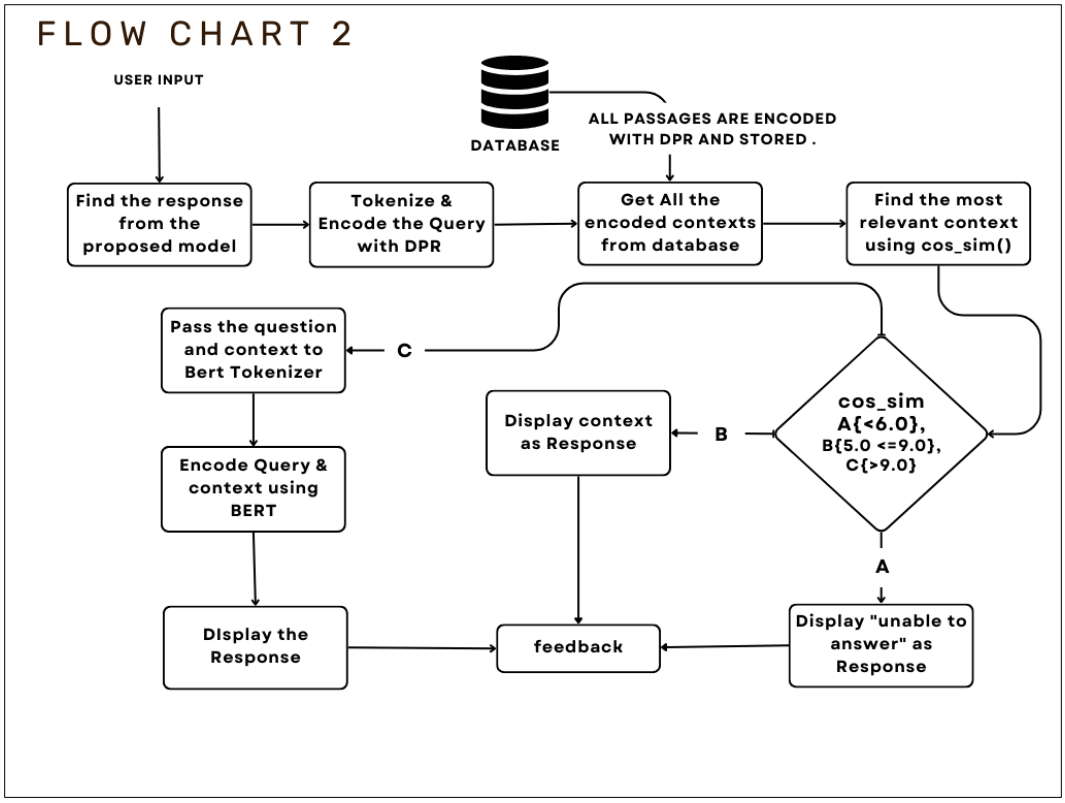

Figure 2: Chatbot model

The above figure describes a significant part of the working model.

3.3 IMPLEMENTATION OF MODEL

The proposed work involves two major steps:

i. Dataset Generation
ii. Building Educational AI Chatbot

DATASET GENERATION:

This proposed work depends on historical question answer dataset. Due to inadequate or poor question answer dataset, it involves generating dataset by scraping the institutes' website, the generated contents are split into less than 100 words per block, later encoded using DPRContextEncoder and stored in pickle file.

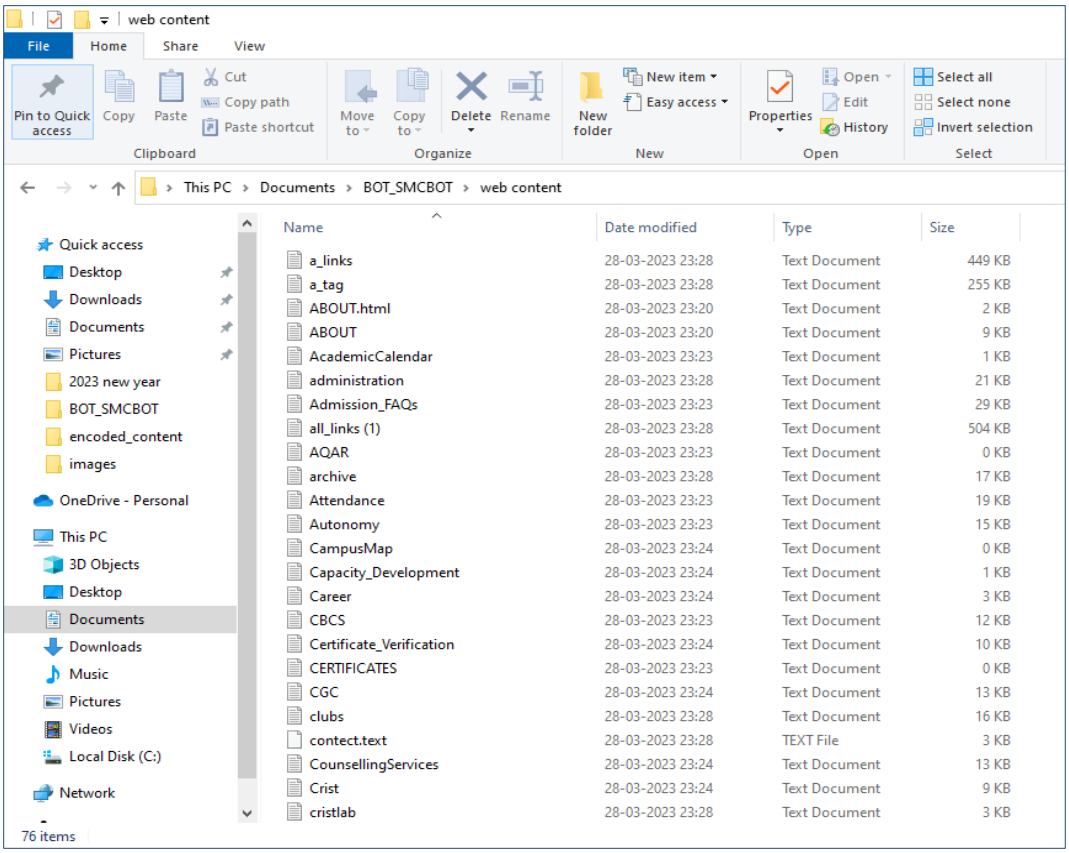

Figure 3: How web scrapped contents are stored

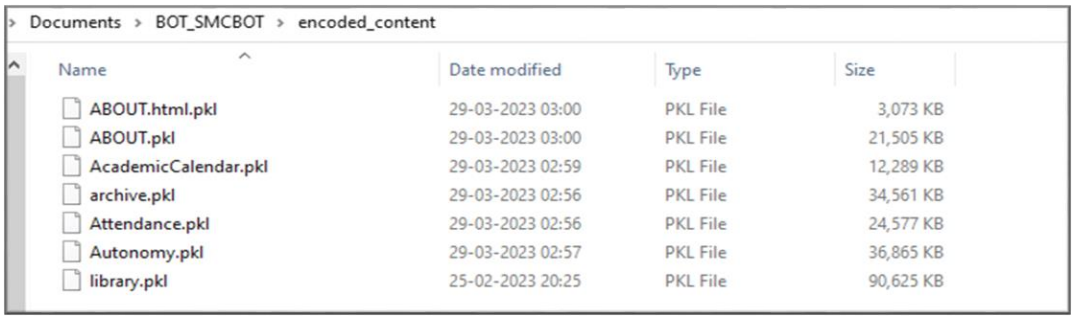

Figure 4:How DRP context encoded contents are stored

BUILDING EDUCATIONAL AI CHATBOT

The chatbot application is hosted as a web application, which helps in answering students' queries. The query is then passed to the backend for processing, then the question is tokenized and encoded by using DPRQuestionTokenizer and DPRQuestionEncoder respectively. The encoded context is retrieved from the database which are stored as pickle files, the collected contexts are compared against the encoded question.

The response generation involves following steps:

After collecting the input from the user side and

1. Check whether the input is a general conversation, if yes, return the response which is hardcoded.
2. If not,
    a. Check whether the it's a known query, if yes, search the CSV file and return the corresponding response.

	A	B	C
1	Question	Response	feedback
2	hi	Hola human, please tell me your queries	yes
3	how are you	Hi, how are you? I am great thanks! Please tell me you	yes
4	where is Postgraduate lik	the top floor	yes
5	what is stellarchives	Planned and designed by Ms. Gita Balachandran, forr	yes
6			
7			
8			
9			

Figure 5:The positive feebacks are stored in CSV file and this file is used to identify the known queries.

b. Else, pass the input to DRP question encoder for encoding, and compare the encoded vectors against all available encoded contexts using cosine similarity cos_sim() . If the resulting vector of cos_sim() is

i. Less than 0.6 return “unable to answer” as response

```
def basic_ques(q):
    q = q.lower()
    l1 = ["Hi there","hi","Hola","Hello","Hello there","Hya","Hya there"]
    l2 = ["Hi there","Hola","Hi human, please tell me your qurries","Hello human, please tell me your qurries","Hola h
    l3 = ["how are you","Hi how are you","hru","h r u","Hello how are you","Hola how are you","How are you doing","Hop
    l4 = ["Hello, I am great, how are you? Please tell me your querries","Hello, how are you? I am great thanks! Pleas
    l5 = ["What is your name","What could I call you","What can I call you","What do your friends call you","Who are y
    l6 = ["You can call me SMCBOT","You may call me SMCBOT","Call me SMCBOT"]
    l7 = ["OK thank you","OK thanks","OK","Thanks","Thankyou","Thank you","That's helpful"]
    l8 = ["No problem!","Happy to help!","Any time!","My pleasure"]
    l9 = ["Thanks bye","bye","byeeeee","byyye","Thanks for the help goodbye","Thank you bye","Thank you, goodbye","Tha
    l10 = ["No problem, goodbye","Not a problem! Have a nice day","Bye! Come back again soon."]
    resp = [[l1,l2],[l3,l4],[l5,l6],[l7,l8],[l9,l10]]
    for ar in resp:
        if q in [item.lower() for item in ar[0]]:
            ans = random.choice(ar[1])
            return ans
    return None
```

Figure 6: hardcoded responses

ii. Between 0.6 and 0.9, return the context as the response

```
ABOUT.pkl
tensor([[-0.0035,  0.4323,  0.0479,  ..., -0.6383, -0.6838,  0.1781],
        [ 0.2856,  0.3295,  0.2040,  ..., -0.8055, -0.4550, -0.0058],
        [ 0.0743,  0.4859,  0.0109,  ..., -0.7663, -0.1507,  0.2199],
        ...,
        [ 0.2856,  0.3295,  0.2040,  ..., -0.8055, -0.4550, -0.0058],
        [ 0.0743,  0.4859,  0.0109,  ..., -0.7663, -0.1507,  0.2199],
        [-0.1896,  0.2827, -0.2693,  ...,  0.0510,  0.0723,  0.1806]],
       grad_fn=<AliasBackward0>)
filename & result: ABOUT
 tensor([[0.4712, 0.4578, 0.4717, 0.3494, 0.4712, 0.4578, 0.4717, 0.3494, 0.4712,
         0.4578, 0.4717, 0.3494, 0.4712, 0.4578, 0.4717, 0.3494, 0.4712, 0.4578,
         0.4717, 0.3494, 0.4712, 0.4578, 0.4717, 0.3494, 0.4712, 0.4578, 0.4717,
         0.3494]], grad_fn=<MmBackward0>)
filename & value ABOUT
 tensor(0.4717, grad_fn=<MaxBackward1>)
max tensor(0.4717, grad_fn=<MaxBackward1>)
```

Figure 7: searching for similar context from all available pickle files and selecting top context with highest value.

```
FILE NAME library
 ARGMAX 1
127.0.0.1 - - [10/Apr/2023 16:48:49] "GET /get?msg=CBCS HTTP/1.1" 200 -
```

Figure 8: Returning the selected context.

iii. Greater than 0.9 pass the question and the context to the BERT model. Tokenize both the question and context using BERT tokenizer and pass the resulting value to BERT encoder, which returns the starting and ending index of the answer from the given context.

DPR Context encoder: returns **pooler_output** (torch.FloatTensor of shape (batch_size, embeddings_size)) — The DPR encoder outputs the *pooler_output* that corresponds to the context representation. The last layer hidden state of the first token of the sequence (classification token) further processed by a Linear layer. This output is to be used to embed contexts for nearest neighbors' queries with questions embeddings. (768-dimensional vector)

DPR Question encoder: returns **pooler_output** (torch.FloatTensor of shape (batch_size, embeddings_size)) — The DPR encoder outputs the *pooler_output* that corresponds to the question representation. The last layer hidden state of the first token of the sequence (classification token) further processed by a Linear layer. This output is to be used to embed questions for nearest neighbors' queries with context embeddings. (768-dimensional vector)

```
def get_context(xq):
  res_lst =dict()
  for i, xq_vec in enumerate(xq.pooler_output):
      for root, dirs, files in os.walk(f"C:/Users/HP/Documents/BOT_SMCBOT/encoded_content/"):
          for file in files:
              print(file)
              filename, extension = os.path.splitext(file)
              if extension == '.pkl':
                  x = pickle.load(open(f"C:/Users/HP/Documents/BOT_SMCBOT/encoded_content/{filename}.pkl", 'rb'))
                  xb = transformers.models.dpr.modeling_dpr.DPRContextEncoderOutput(torch.Tensor(x))
                  print(xb.pooler_output)
                  probs = cos_sim(xq_vec, xb.pooler_output)
                  print("filename & result:",filename,"\n",probs)
                  print("filename & value",filename,"\n",torch.max(probs))
                  print("max",torch.max(probs))
                  print("\n\n\n")
                  res_lst[filename] = [torch.argmax(probs).item(),torch.max(probs)]

      filenm = next(iter(res_lst))
      v = list(res_lst.values())

      for key in res_lst:
          if res_lst[key][1] > res_lst[filenm][1]:
              filenm = key
              armx = res_lst[key][0]
```

Figure 9:Loading the contexts from the database and returning most similar context

The similarity between the question and the context in calculated using cos_sim() cosine similarity, one advantage of cosine similarity is its low complexity, especially for sparse vectors: only the non-zero coordinates need to be considered.

Cosine similarity is the cosine of the angle between the vectors; that is, it is the *dot product* of the vectors divided by the product of their lengths. It follows that the cosine similarity does not depend on the magnitudes of the vectors, but only on their angle. The cosine similarity always belongs to the interval $[-1,1]$, where 1 indicates that the two texts are the same. where -1 indicates that the two texts are completely different. Anywhere in between will be considered how close it is to 1, that much similar.to compute the semantic similarity between the question and context.

from sentence_transformers.util import cos_sim

probs = cos_sim(xq_vec, xb.pooler_output)

$$sim(q,p) = E_Q(q)^T E_P(p)$$

Where, xq_vec is pooler_output of question and xb for context.

The cosine similarity result will be considered for further computation, if the value falls anywhere between 0.6 and 0.9 that context will be considered as bot response and if the value is above 0.9 then the question and context will be passed to BERT model for further processing and if the falls below 0.6 its considered response not found.

If the similarity is above 0.9, both will be tokenized using BERT Tokenizer and fed to BERT Encoder model.

This model returns a very precise answer in 2 or 3 words instead of a passage, which will be displayed in the chatbot interface as chatbot response. The feedback will be collected from the student for every response and stored in a csv file to finetune the chatbots' future responses.

```
C:\Users\HP\Documents\BOT_SMCBOT>c:/Users/HP/Documents/BOT_SMCBOT/venv/Scripts/python.exe c:/Users/HP/Documents/BO
T_SMCBOT/cbot.py
ctx_model
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from.
It may result in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'DPRQuestionEncoderTokenizer'.
The class this function is called from is 'DPRContextEncoderTokenizer'.
qus_model
```

Figure 10: loading the models

4.IMPLEMENTATION

4.1. DATASET GENERATION

Required packages:

```
import requests as req

import re

from bs4 import BeautifulSoup
```

Source Code:

```
def split_passage(pasg):
  print(pasg)
  words = pasg.split()
  ctex = []
  text = ""
  #print(len(words))
  #print(len(words)//100)
  i = 0
  for word in words:
    if i%100 == 0 and i!= 0:
      ctex.append(text)
      text = ""
    text += " "+ word
    i += 1
  ctex.append(text)
  print("LEN_of words",len(words))
  print("len of ctex:",len(ctex))
  return ctex



resp = req.get('https://stellamariscollege.edu.in/')
html = resp.text
```

```
soup = BeautifulSoup(html,'lxml') # lxml gives better experience than default html
with open('a_tag.txt' , 'a+') as cxts:
  tag_a = soup.find_all('a')
  print(len(tag_a))
  for i in range(len(tag_a)):
  #print(tag_a[i])
    cxts.write(tag_a[i].get('href'))
    #cxts.write(tag_p[i].)
    cxts.write('\n')



print('WEB SCRAPPIIIIING OVER')
with open("a_tag.txt",'r') as tg_a:
  a = tg_a.read().split('\n')
  for i in range(len(a)):
    link = a[i]
    if not link.startswith("javascript") and not link.__contains__('#') and not
link.__contains__('/assets/') :
      if link.startswith("http"):
        with open("links.txt","a+") as l:
          l.write(a[i]+"\n")
```

```
      else:
        with open("a_links.txt","a+") as al:
          al.write(root+a[i]+'\n')



with open("a_links.txt",'r') as all:
  aa = all.read().split('\n')
with open("links.txt",'r') as ll:
  ll = ll.read().split('\n')
with open("all_links.txt","a+") as lks:
  for i in range(len(aa)):
    lks.write(aa[i]+"\n")
  for i in range(len(ll)):
    lks.write(ll[i]+"\n")
with open("all_links.txt","r") as lks:
  Links = set(lks.read().split("\n"))
j= 0
for link in Links:
  if link == "" :
    continue
  if root not in link:
    continue
```

```
try:
  resp = req.get(link)
  print(link)
  html = resp.text
  if link == " ":
    continue
  if link.split('/')[-1] == "":
    title = "home"
  else:
    title  = link.split('/')[-1]
  with open(f'{title}.txt',"a+") as file:
    main_file = open("all_content.txt","a+")
    soup = BeautifulSoup(html,'lxml')
    tag_p = soup.find_all('p')


    print(len(tag_p))
    if len(tag_p) == 0:
      j+=0
      continue



    for i in range(len(tag_p)):
```

```
        l = split_passage(tag_p[i].text)
        for j in range(len(l)):
          file.write(str(l[j]))
          main_file.write(str(l[j]))
          file.write('\n')
          main_file.write('\n')
      except:
        pass
```

4.2 BUILDING EDUCATIONAL AI CHATBOT

Required packages :

```
from sentence_transformers.util import cos_sim
from transformers import DPRContextEncoder, DPRContextEncoderTokenizer, DPRQuestionEncoder, DPRQuestionEncoderTokenizer
import transformers
import os
import numpy as np
import pickle
import torch
import pandas as pd
import numpy as np
import random,string,re
import nltk
```

```
from nltk.corpus import stopwords

from transformers import BertForQuestionAnswering

from transformers import BertTokenizer
```

Source Code:

```
ctx_model = DPRContextEncoder.from_pretrained('facebook/dpr-ctx_encoder-single-nq-base')

ctx_tokenizer = DPRContextEncoderTokenizer.from_pretrained('facebook/dpr-ctx_encoder-single-nq-base')

print("qus_model")

question_model = DPRQuestionEncoder.from_pretrained('facebook/dpr-question_encoder-single-nq-base')

question_tokenizer = DPRQuestionEncoderTokenizer.from_pretrained('facebook/dpr-question_encoder-single-nq-base')

stop_words = set(stopwords.words('english'))
```

```
x = pickle.load(open("C:/Users/HP/Documents/chatbot-project/file_ctx.pkl", 'rb'))
#To load saved model from local directory
```

```
pooled_output = x

xb =
transformers.models.dpr.modeling_dpr.DPRContextEncoderOutput(torch.Tensor(
x))

type(xb)
```

```
model = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-
word-masking-finetuned-squad')

tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-
masking-finetuned-squad')
```

```
def question_answer(question, text):

    #tokenize question and text in ids as a pair

    input_ids = tokenizer.encode(question, text)
```

```
#string version of tokenized ids

tokens = tokenizer.convert_ids_to_tokens(input_ids)


#first occurence of [SEP] token

sep_idx = input_ids.index(tokenizer.sep_token_id)



#number of tokens in segment A - question

num_seg_a = sep_idx+1



#number of tokens in segment B - text

num_seg_b = len(input_ids) - num_seg_a


#list of 0s and 1s

segment_ids = [0]*num_seg_a + [1]*num_seg_b

#print("segment_ids: ",segment_ids)

assert len(segment_ids) == len(input_ids)
```

```
    #model output using input_ids and segment_ids
    output = model(torch.tensor([input_ids]),
token_type_ids=torch.tensor([segment_ids]))

    #reconstructing the answer
    answer_start = torch.argmax(output.start_logits)
    answer_end = torch.argmax(output.end_logits)



    answer = str()
    if answer_end >= answer_start:

        answer = tokens[answer_start]
        for i in range(answer_start+1, answer_end+1):
            if tokens[i][0:2] == "##":
                answer += tokens[i][2:]
            else:
                answer += " " + tokens[i]
    #else:
    if answer.startswith("[CLS]"):
        answer = "Unable to find the answer to your question."
    return ("{}".format(answer.capitalize()))
```

```
def get_context(xq):
  res_lst =dict()
  for i, xq_vec in enumerate(xq.pooler_output):
        for root, dirs, files in
os.walk(f"C:/Users/HP/Documents/BOT_SMCBOT/encoded_content/"):
            for file in files:
                print(file)
                filename, extension = os.path.splitext(file)
                if extension == '.pkl':
                    x =
pickle.load(open(f"C:/Users/HP/Documents/BOT_SMCBOT/encoded_content/{fi
lename}.pkl", 'rb'))
                    xb =
transformers.models.dpr.modeling_dpr.DPRContextEncoderOutput(torch.Tensor(
x))
                    print(xb.pooler_output)
                    probs = cos_sim(xq_vec, xb.pooler_output)
```

```
                print("filename & result:",filename,"\n",probs)
                print("filename & value",filename,"\n",torch.max(probs))
                print("max",torch.max(probs))
                print("\n\n\n")
                res_lst[filename] = [torch.argmax(probs).item(),torch.max(probs)]

        filenm = next(iter(res_lst))
        v = list(res_lst.values())


        for key in res_lst:
            if res_lst[key][1] > res_lst[filenm][1]:
                filenm = key
                armx  = res_lst[key][0]

        print("FILE NAME",filenm,"\n ARGMAX",armx)
        with open(f"C:/Users/HP/Documents/BOT_SMCBOT/web
content/{filenm}.txt",errors="ignore") as fs:
                const = fs.read().split("\n")
                if res_lst[filenm][1] < 0.6:
                    return None,None
                elif res_lst[filenm][1] >= 0.6 and res_lst[filenm][1] < 0.9:
```

```
            return (const[armx], "same")
        else:
            return (const[armx],"different")

            #argmax = torch.argmax(probs)
    #print(context[argmax])
    #return context[argmax]


def basic_ques(q):
    q = q.lower()
    l1 = ["Hi there","hi","Hola","Hello","Hello there","Hya","Hya there"]
    l2 = ["Hi there","Hola","Hi human, please tell me your queries","Hello human, please tell me your queries","Hola human, please tell me your queries"]
    l3 = ["how are you","Hi how are you","hru","h r u","Hello how are you","Hola how are you","How are you doing","Hope you are doing well","Hello hope you are doing"]
    l4 = ["Hello, I am great, how are you? Please tell me your queries","Hello, how are you? I am great thanks! Please tell me your queries","Hello, I am good thank you, how are you? Please tell me your queries","Hi, I am great, how are you? Please tell me your queries","Hi, how are you? I am great thanks! Please tell me
```

```
your queries","Hi, I am good thank you, how are you? Please tell me your queries","Hi, good thank you, how are you? Please tell me your queries"]

    l5 = ["What is your name","What could I call you","What can I call you","What do your friends call you","Who are you","Tell me your name","What is your real name","What is your real name please","What's your real name","Tell me your real name","Your real name","Your real name please","Your real name please"]

    l6 = ["You can call me SMCBOT","You may call me SMCBOT","Call me SMCBOT"]

    l7 = ["OK thank you","OK thanks","OK","Thanks","Thankyou","Thank you","That's helpful"]

    l8 = ["No problem!","Happy to help!","Any time!","My pleasure"]

    l9 = ["Thanks bye","bye","byeeeee","byyye","Thanks for the help goodbye","Thank you bye","Thank you, goodbye","Thanks goodbye","Thanks good bye"]

    l10 = ["No problem, goodbye","Not a problem! Have a nice day","Bye! Come back again soon."]

    l11 = ["anouncement","National Scholarship Scheme","NSP","Academic Calendar","calendar","scholarship"]

    l12 = ["https://stellamariscollege.edu.in/examannounce"]

    l13 = ["LATEST NEWS","latest info","news"]

    l14 = ["https://stellamariscollege.edu.in/"]

    resp = [[l1,l2],[l3,l4],[l5,l6],[l7,l8],[l9,l10],[l11,l12],[l13,l14]]

    for ar in resp:

        if q in [item.lower() for item in ar[0]]:
```

```
            ans = random.choice(ar[1])
            return ans
    return None



def puct_remove(strs):
    out = re.sub('[%s]' % re.escape(string.punctuation), '', strs)
    strs = ""
    for r in out.split():
        if  r.lower() not in stop_words:
            print("r:",r)
            strs += r
    print(strs)
    return strs



def qna(question):
    question = puct_remove(question) # removes puct and stop words
    b_ans = basic_ques(question)
    if b_ans == None:
```

```
        df = pd.read_csv("C:/Users/HP/Documents/BOT_SMCBOT/fdbkY.csv")
        for i in range(len(df)):
            if puct_remove(df.at[i,'Question'].lower()) ==
puct_remove(question.lower()):
                prev_ans = df.at[i,'Response']
                return prev_ans
            else:
                xq_tokens = question_tokenizer(question, max_length=256,
padding='max_length',
                                truncation=True, return_tensors='pt')
                xq = question_model(**xq_tokens)
                text,chk = get_context(xq)
                if chk == "different":
                    ans  = question_answer(question, text)
                elif chk == "same":
                    ans = text
                else:
                    ans = "unable to find answer"



                return ans
    else:
        return b_ans
```

```
from flask import Flask, render_template, request,json,jsonify
import csv

app = Flask(__name__)
app.static_folder = 'static'
@app.route("/")
def home():
    return render_template("index.html")
@app.route("/get",methods=['POST','GET'])
def get_bot_response():
    if request.method =='POST':
        data =request.values
        for d in data:
            d = json.loads(d)
            vals = list(d.values())
            if vals[-1] == 'yes':
                with open('fdbkY.csv','a+') as fb:
                    csv_wtr = csv.writer(fb)
                    csv_wtr.writerow(d.values())
                fb.close()
                return jsonify("success")
```

```
            elif vals[-1] =='no':
                with open('fdbkN.csv','a+') as fb:
                    csv_wtr = csv.writer(fb)
                    csv_wtr.writerow(d.values())
                fb.close()
                #return jsonify("success")
                #return redirect(url_for("main.home"))
                #return redirect('/get')
                return jsonify("success")



    else:
        userText = request.args.get('msg')
        return qna(userText)

if __name__ == "__main__":
    app.run(debug=True)
```

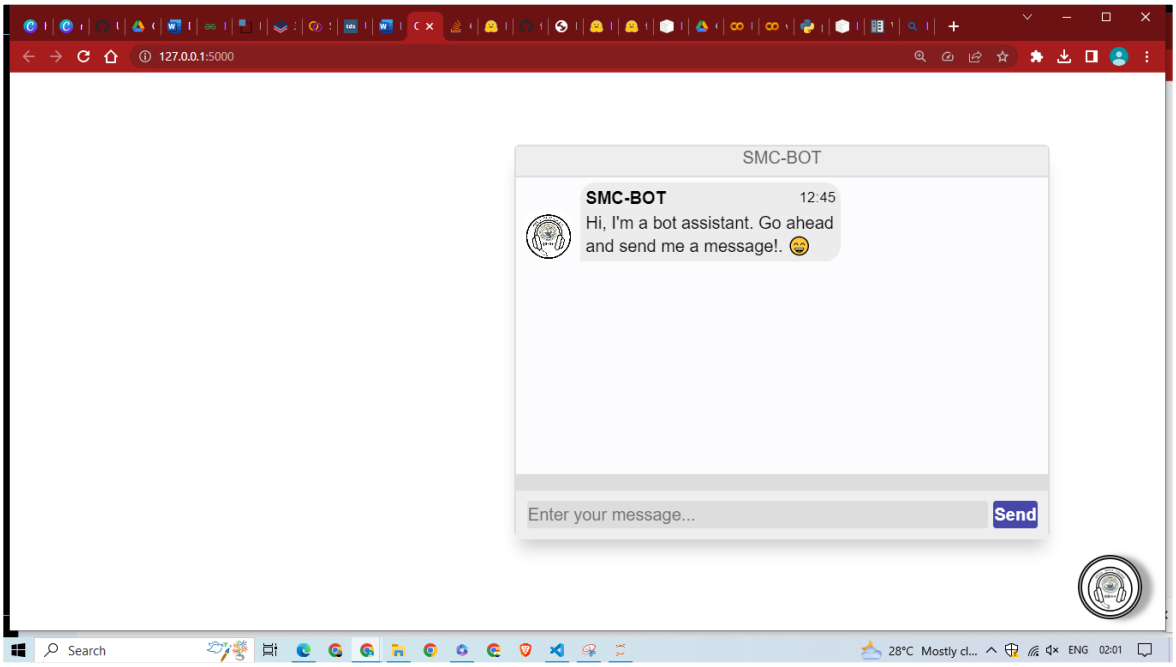

Figure 11:welcome message from chatbot

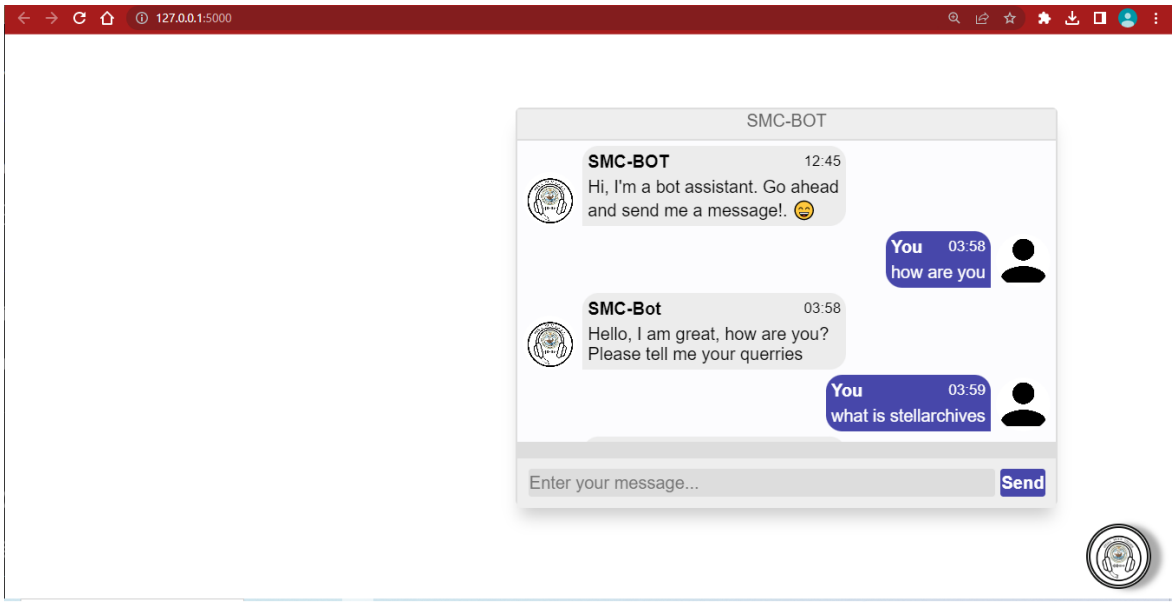

Figure 12: Friendly conversation with chatbot

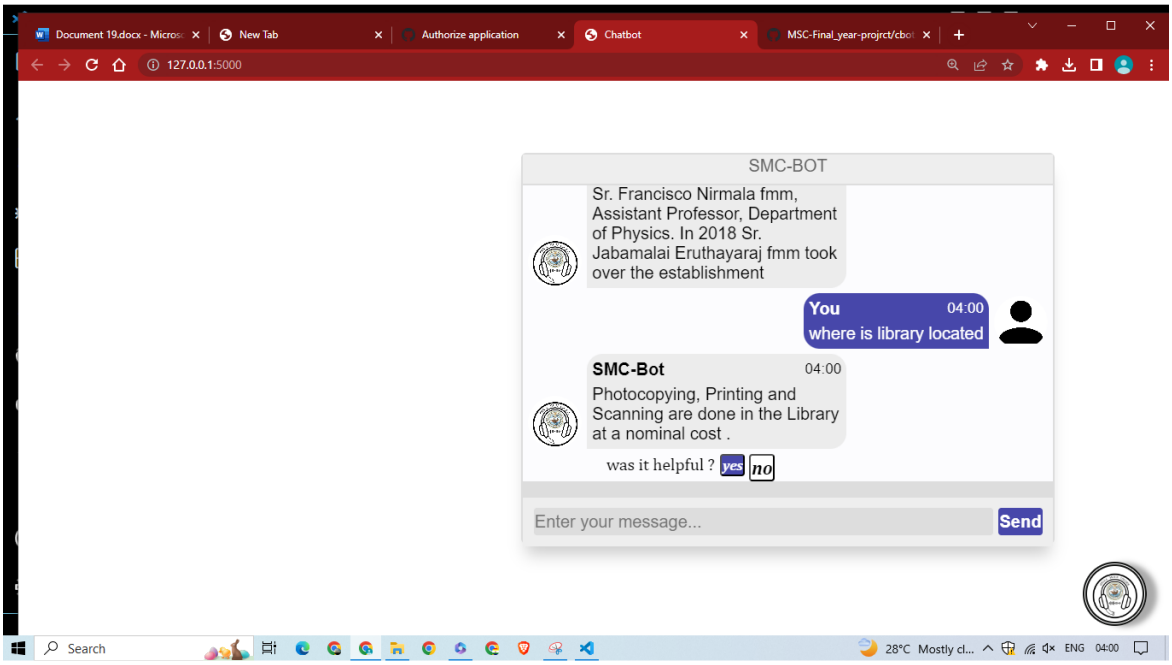

Figure 13: Collecting feed after every response

5. DISCUSSION & CONCLUSION

The Chatbot developed aids students in solving their queries at ease. The modules helped in the realization of the Chatbot involve BERT and DPR. Feedback that is collected from the students at the end of each response generated by the Chatbot is used for fine-tuning the Chatbot's future responses and analyses. The DPR model requires high RAM space for encoding and increased time to generate the response is due to minimal amount of RAM space and search through all files to find an optimal match of response for the query raised by the student. The future work is to incorporate audio and graphical response.