Sebastian version 2 chatbot for Skylanot

K C Praneethaa (21Z223)
Madhumitha D (21Z227)
Madhumitha DR (21Z228)
Roshini P (21Z245)
Sathya M (21Z253)

19Z610 - MACHINE LEARNING LABORATORY

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

BRANCH: COMPUTER SCIENCE AND ENGINEERING

Of Anna University

April 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

1. PROBLEM STATEMENT:

To develop a "Sebastian version 2 chatbot for Skylanot," an innovative voice based mobile application meticulously engineered to reignite student engagement and facilitate seamless access to essential college resources which will enable effortless communication between students and the app, significantly enhancing their overall experience.

2. DATASET DESCRIPTION:

A CSV file holds questions and answers from PSG's mandatory disclosures, covering financials, risk management, and sustainability. It offers stakeholders insights into PSG's operations, enhancing transparency and accountability. This dataset, used to train the BERT model, boosts the chatbot's accuracy in responding to inquiries about PSG's disclosures.

3. METHODOLOGY / MODELS USED:

Language Understanding:

- BERT is used to understand the meaning and context of user input. This includes analyzing text queries or commands provided by users to determine their intent and extract relevant information.
- BERT's bidirectional architecture allows it to capture contextual information from both the left and right contexts of each word in a sentence. This helps the chatbot better understand the nuances and intricacies of natural language.

Natural Language Processing (NLP):

- BERT provides state-of-the-art performance on various NLP tasks, including text classification, named entity recognition, and question answering.
- By leveraging BERT's pre-trained representations and fine-tuning them on our specific dataset, our chatbot can learn to perform tasks such as understanding user queries, providing relevant responses, and handling various conversation scenarios effectively.

Preference for BERT:

- BERT is preferred in our project likely due to its effectiveness and versatility in handling a wide range of NLP tasks.
- It has been pre-trained on large corpora of text data, which helps it capture a rich understanding of language semantics and syntax.

- Fine-tuning BERT on our specific dataset allows the chatbot to adapt and specialize to the domain or context of your application, improving its performance for our particular use case.
- Additionally, the availability of pre-trained BERT models through libraries like Hugging Face Transformers simplifies the implementation and integration process, making it an attractive choice for developers.

4. TOOLS USED:

- Development Environment: Python 3.x, Flask framework for backend development
- Natural Language Processing (NLP) Libraries: Used PyTorch for nlp and deep learning.
- Speech to Text and Text to Speech Libraries : Python pyttxs3
- Python flask - To serve as an interface between Backend - python and frontend - flutter
- Machine Learning Libraries: Transformers for pretrained Bert model
- Version Control: Git for collaborative development and version control
- Dependencies: Necessary Flutter packages for integrating chatbot functionality

5. CHALLENGES FACED:

- Identifying the right training model for chatbot
- Data Collection and Processing for training the chatbot
- Integration of voice with the chatbot
-

6. CONTRIBUTION OF TEAM MEMBERS:

Roll No	**Name**	**Contribution**
21Z223	K C Praneethaa	Front end - flutter code, Model Testing.
21Z227	Madhumitha D	Dataset preparation
21Z228	Madhumitha D R	Text based chatbot logic, flask server setup (integration of backend and frontend)
21Z245	Roshini P	Voice Integration (input and output) with text based chatbot logic
21Z253	Sathya M	Front end - flutter code, Data Preparation and Validation.

7. ANNEXURE I:

Model training python file

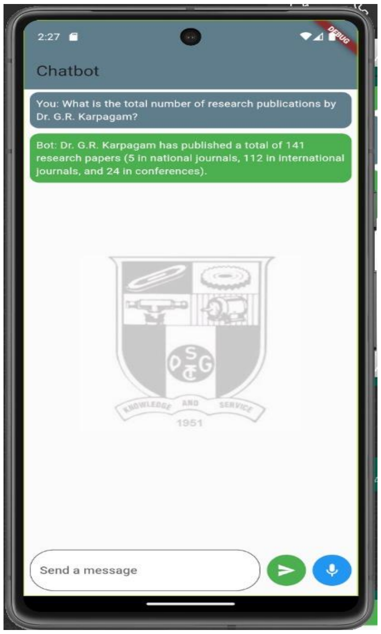
```
import csv
import os
import torch
from transformers import BertTokenizer, BertModel
from scipy.spatial.distance import cosine
import speech_recognition as sr
import pyttsx3

# Load pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")

# Function to process natural language query and return 1-D embedding
def process_query(query):
    inputs = tokenizer(query, return_tensors="pt", max_length=512, truncation=True)
    with torch.no_grad():
        outputs = model(**inputs)
    embeddings = outputs.last_hidden_state.mean(dim=1)
    return embeddings.squeeze()  # Ensure the output is 1-D

# Preprocess dataset to generate 1-D embeddings for each question
def preprocess_dataset(file_name):
    question_embeddings = []
    questions = []
    answers = []
    if os.path.exists(file_name):
        with open(file_name, 'r', encoding='ISO-8859-1') as file:
            csv_reader = csv.reader(file)
            header = next(csv_reader)  # Skip the header row
            for row in csv_reader:
                questions.append(row[0])
                answers.append(row[1])
                # Process each question to get 1-D embedding
                embeddings = process_query(row[0]).numpy()
question_embeddings.append(embeddings)
    else:
        print(f"File '{file_name}' not found.")
    return questions, answers, question_embeddings

# Find the most similar question based on 1-D embeddings
def find_most_similar_question(query_embedding, question_embeddings, questions, answers):
    min_distance = float('inf')
    most_similar_index = -1
    for index, q_embedding in enumerate(question_embeddings):
        # Ensure both embeddings are 1-D before comparing
        distance = cosine(query_embedding, q_embedding)
        if distance < min_distance:
            min_distance = distance
            most_similar_index = index
    if most_similar_index != -1:
        return answers[most_similar_index]
    else:
        return None
# Function to convert text to speech
def speak_text(command):
    engine = pyttsx3.init()
    engine.say(command)
    engine.runAndWait()

# Initialize the recognizer
r = sr.Recognizer()

# Load your dataset and generate 1-D embeddings for questions
file_name = 'Questions_answers.csv'  # Adjust to your CSV file path
questions, answers, question_embeddings = preprocess_dataset(file_name)

# Function to process text input
def process_text_input(user_input):
    # Process the user's query to get 1-D embedding
    query_embedding = process_query(user_input).numpy()
```

```
# Find the most similar question and its answer
based on embeddings
    found_answer =
find_most_similar_question(query_embedding,
question_embeddings, questions, answers)

# Display the answer
    if found_answer:
        print("Answer:", found_answer)
        speak_text("The answer is: " +
found_answer)
    else:
        print("No answer found.")
        speak_text("Sorry, I couldn't find an
answer.")

# Function to process voice input
def process_voice_input():
    try:
        print("Listening:")
        with sr.Microphone() as source:
            r.adjust_for_ambient_noise(source,
duration=0.2)
            audio = r.listen(source)
            text = r.recognize_google(audio).lower()
            print("You said:", text)
            speak_text("You said: " + text)
            process_text_input(text)  # Process the
recognized text
    except sr.RequestError as e:
        print("Could not request results;
{0}".format(e))
    except sr.UnknownValueError:
        print("Unknown error occurred.")

# Save the trained BERT model
model_save_path = "bert_voice_model.pth"
torch.save(model.state_dict(), model_save_path)
print("Model saved at:", model_save_path)

# Ask for input only once
user_input = input("Type your query or type
'voice' to switch to voice input: ").lower()
if user_input == "voice":
    process_voice_input()  # Start voice input
mode
else:
    process_text_input(user_input)  # Process text
input
```

Flask application to serve as a backend

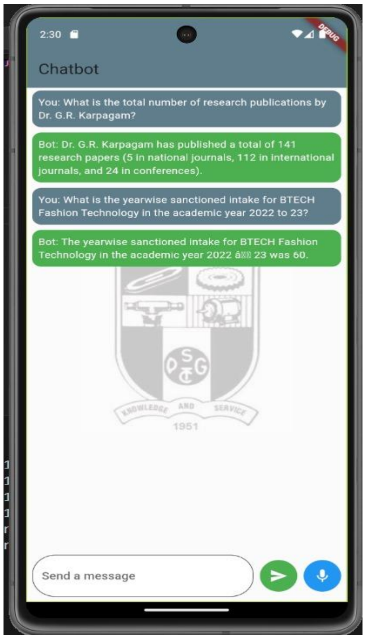
```
from flask import Flask, request, jsonify
import torch
from transformers import BertTokenizer,
BertModel
from scipy.spatial.distance import cosine
import csv
import os
import speech_recognition as sr

app = Flask(__name__)

# Load pre-trained BERT model and tokenizer
tokenizer =
BertTokenizer.from_pretrained("bert-base-
uncased")
model = BertModel.from_pretrained("bert-base-
uncased")
model.eval()

# Function to process natural language query and
return 1-D embedding
def process_query(query):
    inputs = tokenizer(query, return_tensors="pt",
max_length=512, truncation=True)
    with torch.no_grad():
        outputs = model(**inputs)
    embeddings =
outputs.last_hidden_state.mean(dim=1)
    return embeddings.squeeze().numpy()  #
Ensure the output is numpy array

# Function to find the most similar question based
on 1-D embeddings
def
find_most_similar_question(query_embedding,
question_embeddings, questions, answers):
    min_distance = float('inf')
    most_similar_index = -1
    for index, q_embedding in
enumerate(question_embeddings):
        # Ensure both embeddings are 1-D before
comparing
        distance = cosine(query_embedding,
q_embedding)
        if distance < min_distance:
            min_distance = distance
            most_similar_index = index
    if most_similar_index != -1:
        return answers[most_similar_index]
    else:
        return None

# Function to convert speech to text
def speech_to_text():
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
        audio = r.listen(source)
    try:
        query = r.recognize_google(audio)
        print("You said:", query)
        return query
    except sr.UnknownValueError:
        print("Sorry, could not understand audio.")
        return None
    except sr.RequestError:
        print("Sorry, could not request results.")
        return None

# Route to handle query processing
@app.route('/query', methods=['POST'])
def query():
    file_name = 'C:\\Users\DELL\OneDrive\Desktop\datasets\Questions_answers.csv'  # Adjust to your CSV file path
    if not os.path.exists(file_name):
        return jsonify({'message': f"File '{file_name}' not found."}), 404

    # Load your dataset and generate 1-D embeddings for questions
    questions = []
    answers = []
    question_embeddings = []
    with open(file_name, 'r', encoding='ISO-8859-1') as file:
        csv_reader = csv.reader(file)
        header = next(csv_reader)  # Skip the header row
        for row in csv_reader:
            questions.append(row[0])
            answers.append(row[1])
            # Process each question to get 1-D embedding
            embeddings = process_query(row[0])

            question_embeddings.append(embeddings)
    input_type = request.json.get('input_type')
    if input_type == 'text':
        query = request.json.get('query')
    elif input_type == 'voice':
        query = speech_to_text()
    else:
        return jsonify({'message': 'Invalid input type.'}), 400
    if query:
        query_embedding = process_query(query)
        found_answer = find_most_similar_question(query_embedding, question_embeddings, questions, answers)
        if found_answer:
            return jsonify({'answer': found_answer}), 200
        else:
            return jsonify({'message': 'No answer found.'}), 404
    else:
        return jsonify({'message': 'Query not provided.'}), 400

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

8. ANNEXURE II:



You: What is the total number of research publications by Dr. G.R. Karpagam?

Bot: Dr. G.R. Karpagam has published a total of 141 research papers (5 in national journals, 112 in international journals, and 24 in conferences).

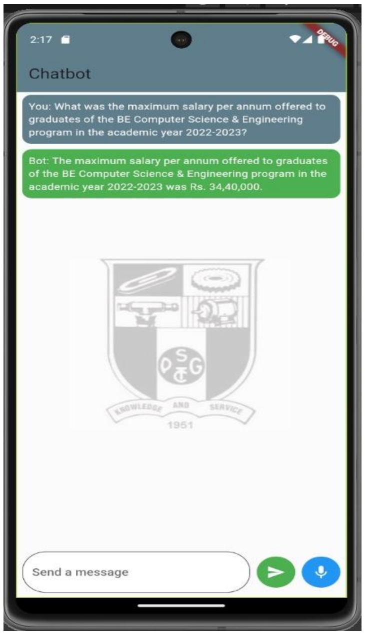


You: What is the total number of research publications by Dr. G.R. Karpagam?

Bot: Dr. G.R. Karpagam has published a total of 141 research papers (5 in national journals, 112 in international journals, and 24 in conferences).

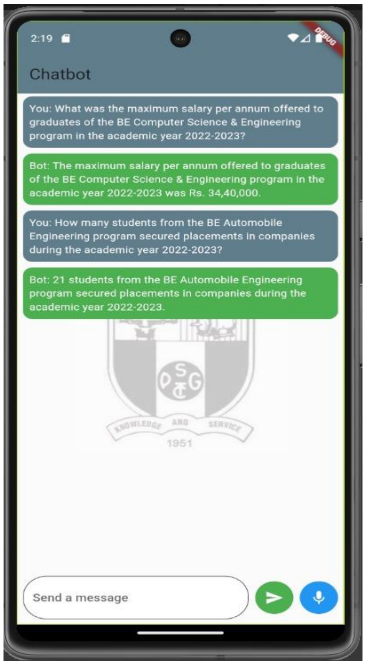
You: What is the yearwise sanctioned intake for BTECH Fashion Technology in the academic year 2022 to 23?

Bot: The yearwise sanctioned intake for BTECH Fashion Technology in the academic year 2022 â 23 was 60.



You: What was the maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023?

Bot: The maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023 was Rs. 34,40,000.

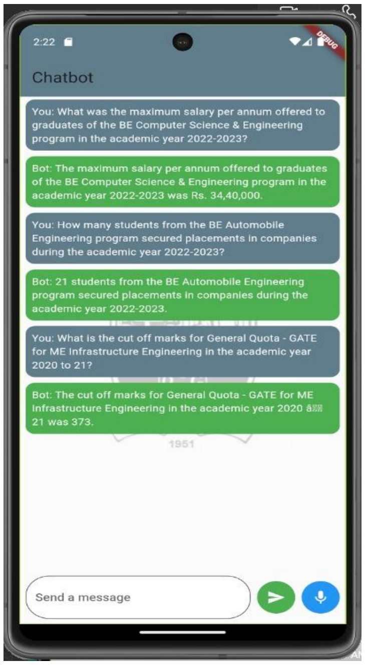


You: What was the maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023?

Bot: The maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023 was Rs. 34,40,000.

You: How many students from the BE Automobile Engineering program secured placements in companies during the academic year 2022-2023?

Bot: 21 students from the BE Automobile Engineering program secured placements in companies during the academic year 2022-2023.



You: What was the maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023?

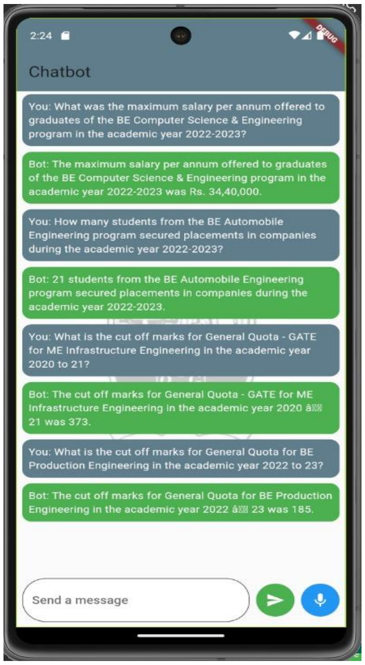
Bot: The maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023 was Rs. 34,40,000.

You: How many students from the BE Automobile Engineering program secured placements in companies during the academic year 2022-2023?

Bot: 21 students from the BE Automobile Engineering program secured placements in companies during the academic year 2022-2023.

You: What is the cut off marks for General Quota - GATE for ME Infrastructure Engineering in the academic year 2020 to 21?

Bot: The cut off marks for General Quota - GATE for ME Infrastructure Engineering in the academic year 2020 â 21 was 373.



You: What was the maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023?

Bot: The maximum salary per annum offered to graduates of the BE Computer Science & Engineering program in the academic year 2022-2023 was Rs. 34,40,000.

You: How many students from the BE Automobile Engineering program secured placements in companies during the academic year 2022-2023?

Bot: 21 students from the BE Automobile Engineering program secured placements in companies during the academic year 2022-2023.

You: What is the cut off marks for General Quota - GATE for ME Infrastructure Engineering in the academic year 2020 to 21?

Bot: The cut off marks for General Quota - GATE for ME Infrastructure Engineering in the academic year 2020 â 21 was 373.

You: What is the cut off marks for General Quota for BE Production Engineering in the academic year 2022 to 23?

Bot: The cut off marks for General Quota for BE Production Engineering in the academic year 2022 â 23 was 185.

9. REFERENCES:

1. https://nitinkchauhan.medium.com/create-a-voice-chatbot-in-python-using-nltk-speech-recognition-google-text-to-speech-445468c083f8
2. https://www.analyticsvidhya.com/blog/2021/10/complete-guide-to-build-your-ai-chatbot-with-nlp-in-python/
3. https://blog.tensorflow.org/2019/05/transformer-chatbot-tutorial-with-tensorflow-2.html
4. https://handsonai.medium.com/build-a-chat-bot-from-scratch-using-python-and-tensorflow-fd189bcfae45
5. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7283081/
6. https://www.infobip.com/blog/whatsapp-chatbot-quick-guide
7. https://www.turing.com/kb/build-customized-chatbots-using-gpt4
8. https://docs.automationanywhere.com/bundle/enterprise-v11.3/page/enterprise/topics/aae-architecture-implementation/bot-design-guidelines.html
9. https://arxiv.org/abs/1810.04805
10. https://github.com/google-research/bert