सी डैक | प्रगत संगणन विकास केंद्र, नोएडा
CDAC | Centre for Development of Advanced Computing, Noida

MR. SHIVAM PANDEY

(Project Guide)

Submitted By

KARAN BISHT	PRN: 230320528008
MANISH SHARMA	PRN: 230320528012
SAIF KHAN	PRN: 230320528009

*Submitted In Partial Fulfillment of the Requirement for the Award of*Post Graduate Diploma in Artificial Intelligence (PG-DAI) Under the Guidance of

CONTENTS

INDEX	TITLE	PAGE NUMBER
I	Certificate	2
II	Acknowledgement	3
III	Abstract	4
IV	Methodology	4
V	Introduction to the problem statement and the possible solution	6
VI	Coding	7-17
VII	Results	18
VIII	Conclusion & Future Scope	19-20
IX	Reference & Bibliography	21

CERTIFICATE

This is to certify that Report entitled “A Natural Language Processing & Machine Learning based Personal Assistant with chat and voice commands” which is submitted by Karan bisht, Manish kumar sharma and khan saif mujahid in partial fulfillment of the requirement for the award of Post Graduate Diploma in Artificial Intelligence (PG-DAI) to CDAC, Noidais a record of the candidates own work carried out by them under my supervision.

The documentation embodies results of original work, and studies are carried out by the student themselves and the contents of the report do not from the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

ACKNOWLEDGEMENT

We would like to express our best sense of gratitude & endeavor with respect to Mr. Shivam pandey (Project Guide) CDAC, Noida for suggesting the problems scholarly guidance and expert supervision during the course of this project. Special thanks to Mr. Ravi Payal (Program Coordinator).

We are very thankful to Mr. Shivam Pandey (Project Guide) the project guide for constant simulated discussion, encouraging new ideas about this project.

ABSTRACT

In Today Fast Paced Era , the significance of a robust and effective personal assistance has grown immensely. The constant demands of both personal and professional life often exceed an individual's ability to efficiently handle multiple tasks, information of different domains and communication interaction. This research project, titled " Personalized A*I* Assistant with Chat and Voice Based Command " is a user - centred AI assistant which aims to tackle this critical requirement by designing a user interactive and personalized AI assistant.

OBJECTIVE

The core aim of this project is to develop a tailored assistant that effectively caters to the distinct needs and preference of every user. We acknowledge that a uniform approach is inadequate in today's diverse and ever-changing landscape. Consequently, our goal is to offer an AI assistant capable of seamless adaptation to individual demands, thereby elevating productivity and convenience.

Methodology

To achieve the objectives of the " Personalized AI Assistant with Chat and Voice Based Command" project, we utilize an extensive array of technologies and methodologies.

Natural Language Processing (NLP): The cornerstone of our assistant's capabilities resides in advanced NLP techniques. NLP is harnessed to facilitate chat-based interactions, voice recognition, and language comprehension. This empowers the assistant to understand user commands, inquiries, and conversations with a high level of precision and contextual awareness.

Speech-to-Text and Text-to-Speech Conversion: Effectively bridging the gap between spoken language and written text, we implement robust mechanisms for speech-to-text and text-to-speech conversion. This functionality allows the assistant to transform spoken commands into text for processing and generate spoken responses, ensuring a natural and intuitive user experience.

Integration with External APIs: To offer users a broad spectrum of functionalities and access to information, our assistant seamlessly integrates with external APIs such as Google, YouTube, Wikipedia, Google Play, and other relevant sources. This integration enables the assistant to fetch real-time data, play music, schedule meetings, retrieve news, and tap into a vast knowledge base, enriching the user experience with diverse capabilities.

Machine Learning: An integral aspect of our project involves the application of machine learning algorithms. These algorithms empower the assistant to continuously learn and adapt to user preferences, progressively personalizing interactions over time. Through machine learning, the assistant becomes more proficient at identifying patterns, suggesting content, and delivering responses that align with individual user requirements.

Flask Integration for User Interaction: Flask is user to establish a smooth user interaction interface within our project. Flask plays a pivotal role in crafting a user-centred web application, facilitating effortless communication between users and our AI assistant. This integration significantly improves the accessibility and user-friendliness of our AI assistant, furnishing a platform through which users can easily issue commands, receive responses, and engage with the assistant via a web-based interface. Flask's adaptable and lightweight framework has empowered us to fashion an interactive user experience, further elevating the functionality of our personalized assistant.

By integrating these technologies and techniques, our project strives to create a comprehensive and adaptable AI assistant capable of addressing diverse user requirements and providing a seamless, user-centered experience. This methodology ensures that our assistant remains at the forefront of technology, continuously evolving to meet the ever-changing needs of our users.

INTRODUCTION TO THE PROBLEM STATEMENT AND THE POSSIBLE SOLUTION

Problem Statement:

In our contemporary, high-speed world, many individuals encounter difficulties in efficiently handling their tasks, accessing information, and streamlining communication. Existing personal assistant solutions often fall short in addressing these challenges as they tend to be generic and lack the adaptability required to cater to the diverse and specific needs of individual users. Moreover, persistent concerns about data privacy and security pose significant obstacles to widespread adoption.

Proposed Solution:

To tackle the aforementioned issues, our project presents the following solutions:

Personalized: Aim to create an exceptionally adaptable AI assistant capable of learning and adjusting to the distinct preferences, routines, and demands of each user. This will empower the assistant to offer customized support, such as managing schedules, setting reminders, and suggesting content tailored to individual user preferences.

Optimal Solutions: Implement cutting-edge natural language processing (NLP) and machine learning algorithms to ensure that the AI assistant comprehends user commands with remarkable accuracy and delivers meaningful responses promptly. We will also continuously refine and update these algorithms to enhance the overall user experience.

Versatility: Designed to accept both voice and chat-based commands, accommodating users by allowing them to interact with the assistant in the manner most convenient to them. This versatility ensures accessibility for users with diverse communication preferences and in various situations.

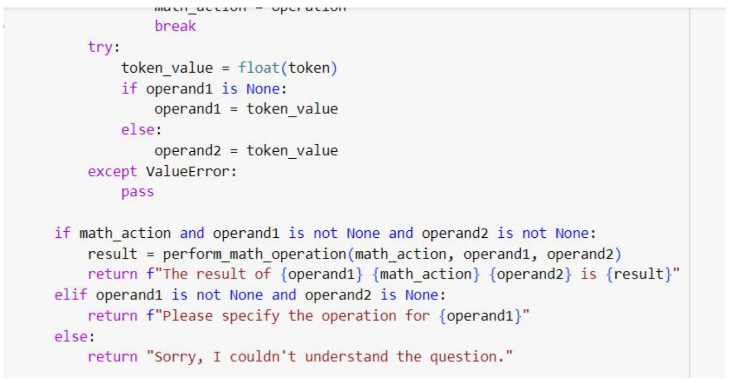
Data Privacy: Prioritize, data privacy and security by integrating robust encryption and user data protection measures. We will transparently communicate the data usage policy to users, obtain their informed consent, and grant them control over what data the assistant can access and store.

Database Management: Develop a secure and efficient database management system to store user-specific preferences and interactions while ensuring compliance with relevant data protection regulations and industry best practices.

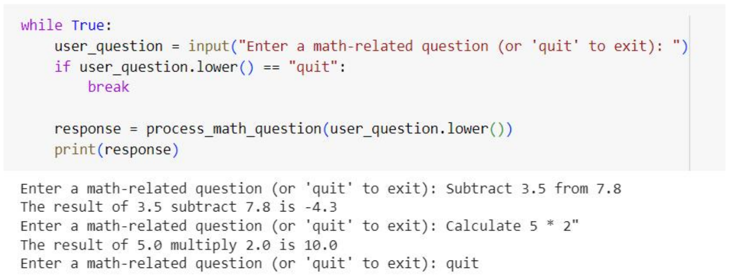
Continuous Improvement: Employing machine learning techniques, our AI assistant will continuously learn from user interactions, adapting its responses and suggestions over time. Regular updates to the AI model will incorporate the latest advancements in AI technology to keep the assistant at the forefront of capabilities.

CODING

Arithmetic model-

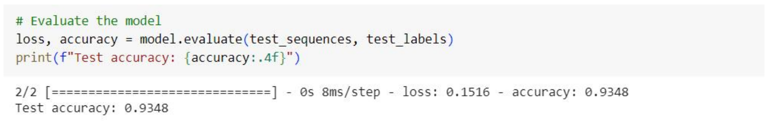


```
                math_action = operation
                break
        try:
            token_value = float(token)
            if operand1 is None:
                operand1 = token_value
            else:
                operand2 = token_value
        except ValueError:
            pass

    if math_action and operand1 is not None and operand2 is not None:
        result = perform_math_operation(math_action, operand1, operand2)
        return f"The result of {operand1} {math_action} {operand2} is {result}"
    elif operand1 is not None and operand2 is None:
        return f"Please specify the operation for {operand1}"
    else:
        return "Sorry, I couldn't understand the question."
```



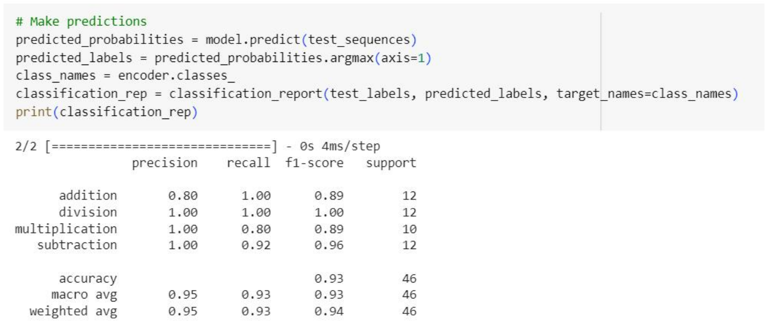
```
while True:
    user_question = input("Enter a math-related question (or 'quit' to exit): ")
    if user_question.lower() == "quit":
        break

    response = process_math_question(user_question.lower())
    print(response)
```

```
Enter a math-related question (or 'quit' to exit): Subtract 3.5 from 7.8
The result of 3.5 subtract 7.8 is -4.3
Enter a math-related question (or 'quit' to exit): Calculate 5 * 2"
The result of 5.0 multiply 2.0 is 10.0
Enter a math-related question (or 'quit' to exit): quit
```



```
# Evaluate the model
loss, accuracy = model.evaluate(test_sequences, test_labels)
print(f"Test accuracy: {accuracy:.4f}")
```

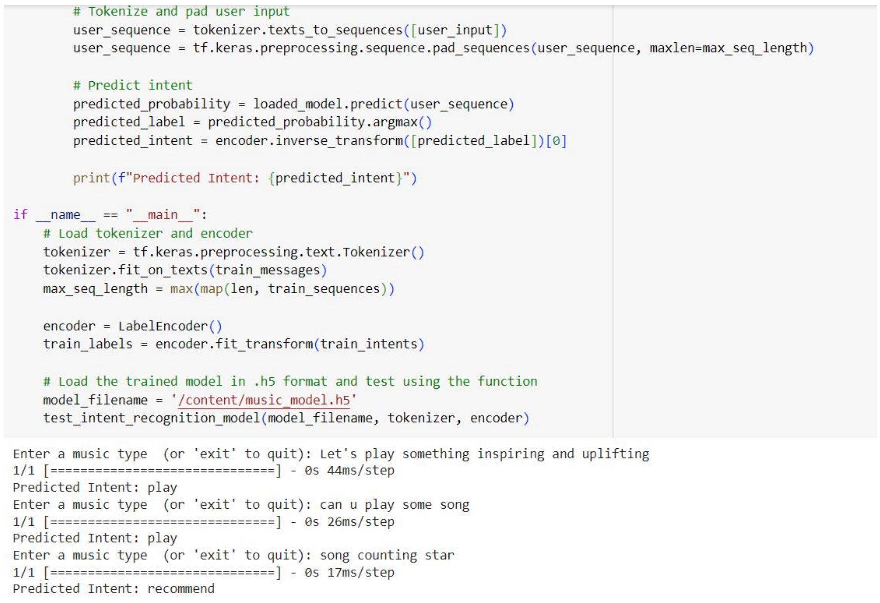
```
2/2 [==============================] - 0s 8ms/step - loss: 0.1516 - accuracy: 0.9348
Test accuracy: 0.9348
```



```
# Make predictions
predicted_probabilities = model.predict(test_sequences)
predicted_labels = predicted_probabilities.argmax(axis=1)
class_names = encoder.classes_
classification_rep = classification_report(test_labels, predicted_labels, target_names=class_names)
print(classification_rep)
```

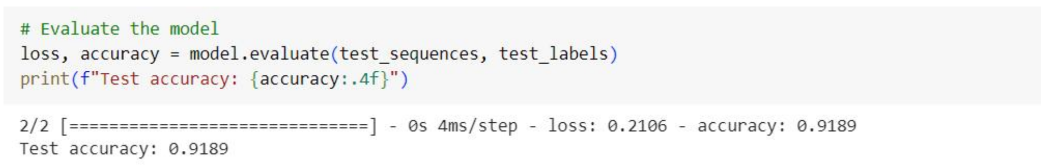
```
2/2 [==============================] - 0s 4ms/step
                precision    recall  f1-score   support

      addition       0.80      1.00      0.89        12
      division       1.00      1.00      1.00        12
multiplication       1.00      0.80      0.89        10
   subtraction       1.00      0.92      0.96        12

      accuracy                           0.93        46
     macro avg       0.95      0.93      0.93        46
  weighted avg       0.95      0.93      0.94        46
```

Play and Recommend-



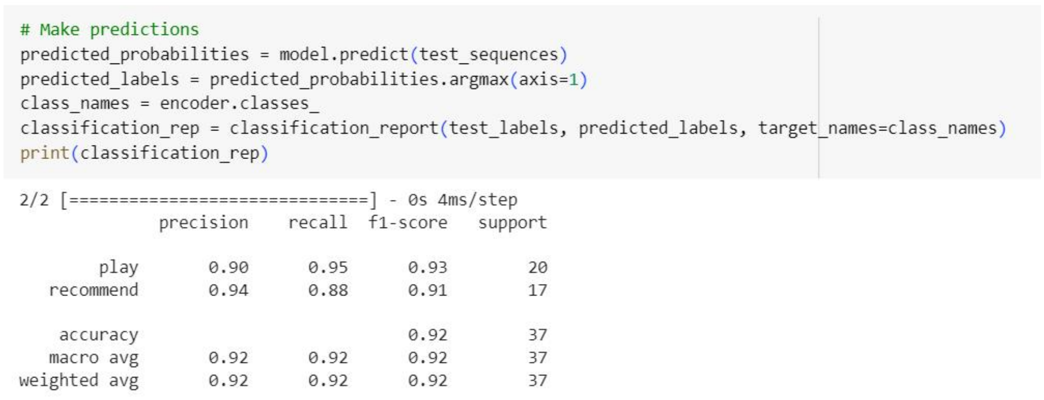
```
        # Tokenize and pad user input
        user_sequence = tokenizer.texts_to_sequences([user_input])
        user_sequence = tf.keras.preprocessing.sequence.pad_sequences(user_sequence, maxlen=max_seq_length)

        # Predict intent
        predicted_probability = loaded_model.predict(user_sequence)
        predicted_label = predicted_probability.argmax()
        predicted_intent = encoder.inverse_transform([predicted_label])[0]

        print(f"Predicted Intent: {predicted_intent}")

if __name__ == "__main__":
    # Load tokenizer and encoder
    tokenizer = tf.keras.preprocessing.text.Tokenizer()
    tokenizer.fit_on_texts(train_messages)
    max_seq_length = max(map(len, train_sequences))

    encoder = LabelEncoder()
    train_labels = encoder.fit_transform(train_intents)

    # Load the trained model in .h5 format and test using the function
    model_filename = '/content/music_model.h5'
    test_intent_recognition_model(model_filename, tokenizer, encoder)
```

```
Enter a music type  (or 'exit' to quit): Let's play something inspiring and uplifting
1/1 [==============================] - 0s 44ms/step
Predicted Intent: play
Enter a music type  (or 'exit' to quit): can u play some song
1/1 [==============================] - 0s 26ms/step
Predicted Intent: play
Enter a music type  (or 'exit' to quit): song counting star
1/1 [==============================] - 0s 17ms/step
Predicted Intent: recommend
```



```
# Evaluate the model
loss, accuracy = model.evaluate(test_sequences, test_labels)
print(f"Test accuracy: {accuracy:.4f}")
```

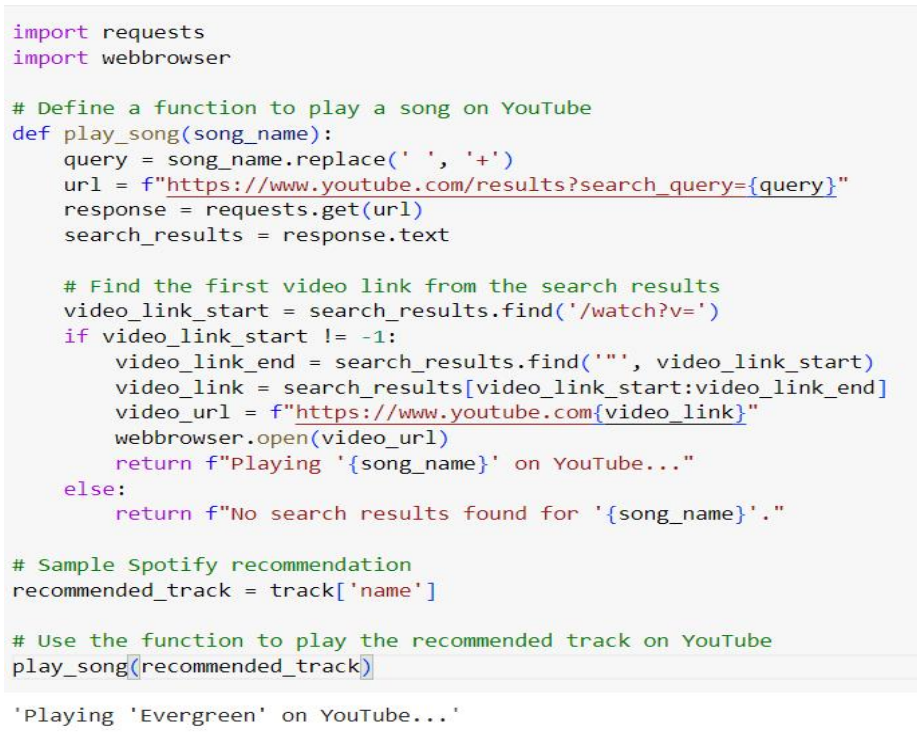
```
2/2 [==============================] - 0s 4ms/step - loss: 0.2106 - accuracy: 0.9189
Test accuracy: 0.9189
```



```
# Make predictions
predicted_probabilities = model.predict(test_sequences)
predicted_labels = predicted_probabilities.argmax(axis=1)
class_names = encoder.classes_
classification_rep = classification_report(test_labels, predicted_labels, target_names=class_names)
print(classification_rep)
```

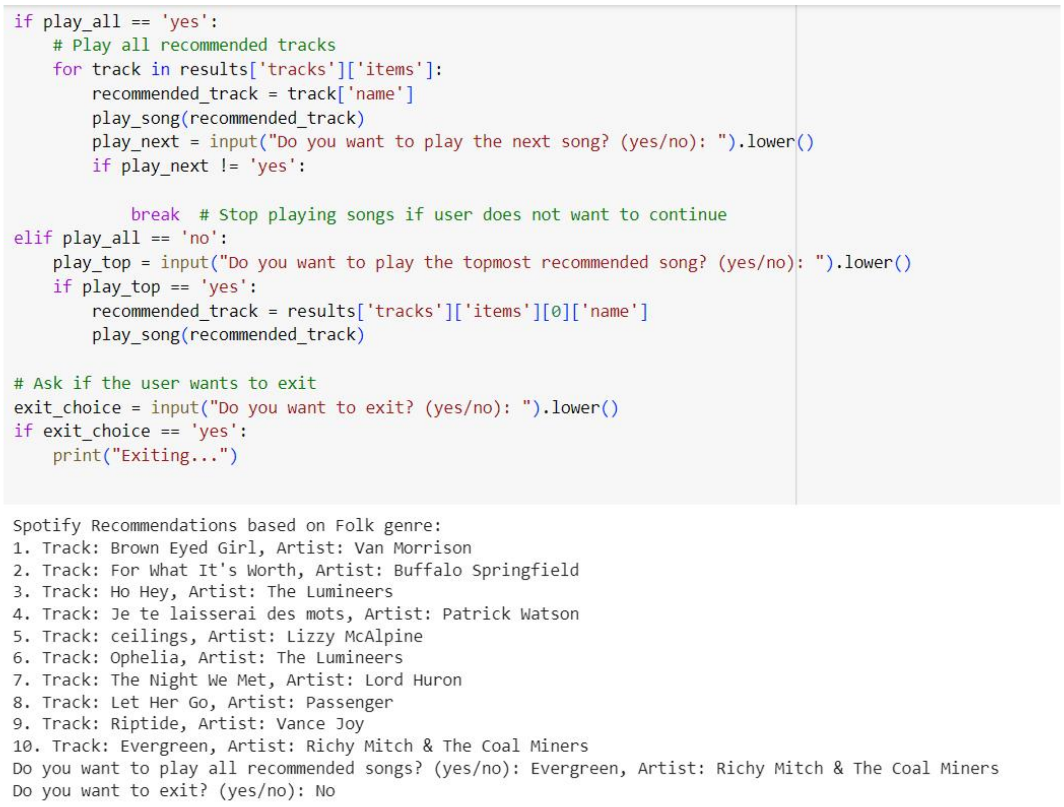
```
2/2 [==============================] - 0s 4ms/step
              precision    recall  f1-score   support

        play       0.90      0.95      0.93        20
   recommend       0.94      0.88      0.91        17

    accuracy                           0.92        37
   macro avg       0.92      0.92      0.92        37
weighted avg       0.92      0.92      0.92        37
```

Music Recommend Model-



```
import requests
import webbrowser

# Define a function to play a song on YouTube
def play_song(song_name):
    query = song_name.replace(' ', '+')
    url = f"https://www.youtube.com/results?search_query={query}"
    response = requests.get(url)
    search_results = response.text

    # Find the first video link from the search results
    video_link_start = search_results.find('/watch?v=')
    if video_link_start != -1:
        video_link_end = search_results.find('"', video_link_start)
        video_link = search_results[video_link_start:video_link_end]
        video_url = f"https://www.youtube.com{video_link}"
        webbrowser.open(video_url)
        return f"Playing '{song_name}' on YouTube..."
    else:
        return f"No search results found for '{song_name}'."

# Sample Spotify recommendation
recommended_track = track['name']

# Use the function to play the recommended track on YouTube
play_song(recommended_track)
```

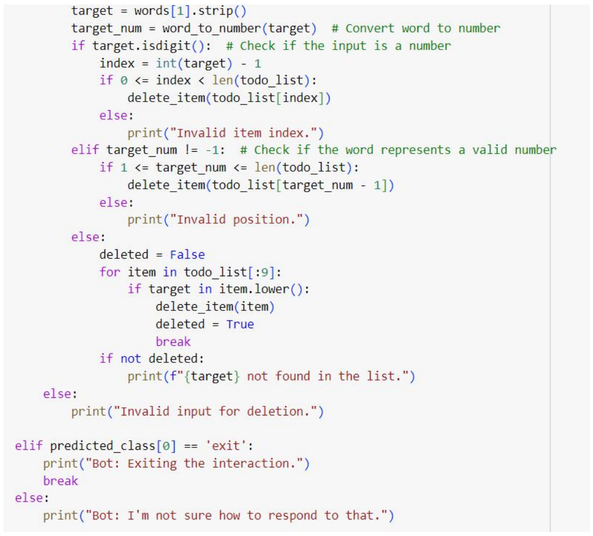
```
'Playing 'Evergreen' on YouTube...'
```



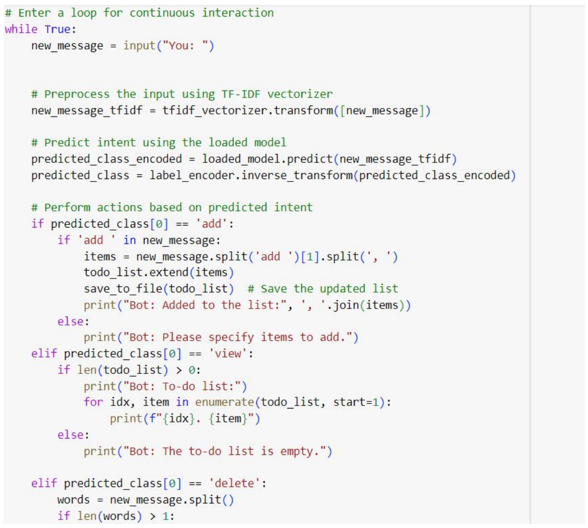
```
if play_all == 'yes':
    # Play all recommended tracks
    for track in results['tracks']['items']:
        recommended_track = track['name']
        play_song(recommended_track)
        play_next = input("Do you want to play the next song? (yes/no): ").lower()
        if play_next != 'yes':

            break  # Stop playing songs if user does not want to continue
elif play_all == 'no':
    play_top = input("Do you want to play the topmost recommended song? (yes/no): ").lower()
    if play_top == 'yes':
        recommended_track = results['tracks']['items'][0]['name']
        play_song(recommended_track)

# Ask if the user wants to exit
exit_choice = input("Do you want to exit? (yes/no): ").lower()
if exit_choice == 'yes':
    print("Exiting...")
```

```
Spotify Recommendations based on Folk genre:
1. Track: Brown Eyed Girl, Artist: Van Morrison
2. Track: For What It's Worth, Artist: Buffalo Springfield
3. Track: Ho Hey, Artist: The Lumineers
4. Track: Je te laisserai des mots, Artist: Patrick Watson
5. Track: ceilings, Artist: Lizzy McAlpine
6. Track: Ophelia, Artist: The Lumineers
7. Track: The Night We Met, Artist: Lord Huron
8. Track: Let Her Go, Artist: Passenger
9. Track: Riptide, Artist: Vance Joy
10. Track: Evergreen, Artist: Richy Mitch & The Coal Miners
Do you want to play all recommended songs? (yes/no): Evergreen, Artist: Richy Mitch & The Coal Miners
Do you want to exit? (yes/no): No
```

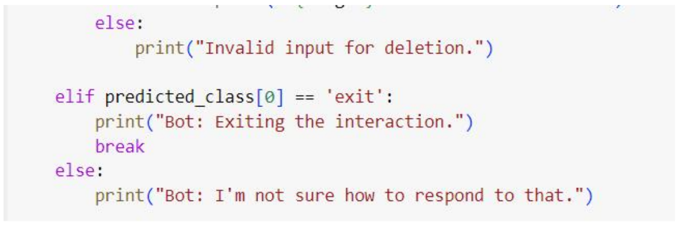
To Do List Model-



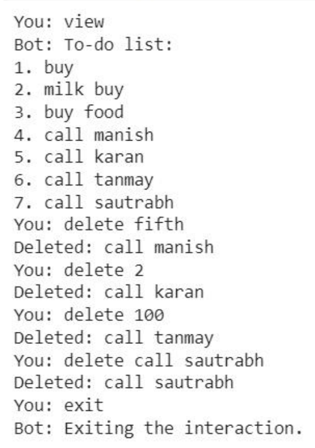
```
            target = words[1].strip()
            target_num = word_to_number(target)  # Convert word to number
            if target.isdigit():  # Check if the input is a number
                index = int(target) - 1
                if 0 <= index < len(todo_list):
                    delete_item(todo_list[index])
                else:
                    print("Invalid item index.")
            elif target_num != -1:  # Check if the word represents a valid number
                if 1 <= target_num <= len(todo_list):
                    delete_item(todo_list[target_num - 1])
                else:
                    print("Invalid position.")
            else:
                deleted = False
                for item in todo_list[:9]:
                    if target in item.lower():
                        delete_item(item)
                        deleted = True
                        break
                if not deleted:
                    print(f"{target} not found in the list.")
        else:
            print("Invalid input for deletion.")

    elif predicted_class[0] == 'exit':
        print("Bot: Exiting the interaction.")
        break
    else:
        print("Bot: I'm not sure how to respond to that.")
```



```
# Enter a loop for continuous interaction
while True:
    new_message = input("You: ")


    # Preprocess the input using TF-IDF vectorizer
    new_message_tfidf = tfidf_vectorizer.transform([new_message])

    # Predict intent using the loaded model
    predicted_class_encoded = loaded_model.predict(new_message_tfidf)
    predicted_class = label_encoder.inverse_transform(predicted_class_encoded)

    # Perform actions based on predicted intent
    if predicted_class[0] == 'add':
        if 'add ' in new_message:
            items = new_message.split('add ')[1].split(', ')
            todo_list.extend(items)
            save_to_file(todo_list)  # Save the updated list
            print("Bot: Added to the list:", ', '.join(items))
        else:
            print("Bot: Please specify items to add.")
    elif predicted_class[0] == 'view':
        if len(todo_list) > 0:
            print("Bot: To-do list:")
            for idx, item in enumerate(todo_list, start=1):
                print(f"{idx}. {item}")
        else:
            print("Bot: The to-do list is empty.")

    elif predicted_class[0] == 'delete':
        words = new_message.split()
        if len(words) > 1:
```

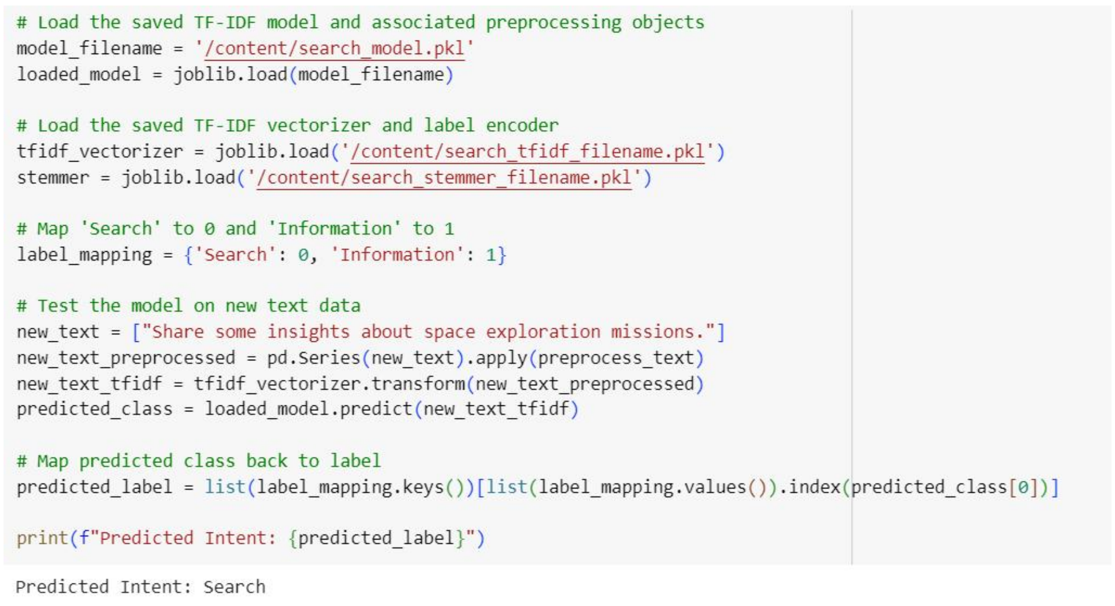


```
        else:
            print("Invalid input for deletion.")

    elif predicted_class[0] == 'exit':
        print("Bot: Exiting the interaction.")
        break
    else:
        print("Bot: I'm not sure how to respond to that.")
```



```
You: view
Bot: To-do list:
1. buy
2. milk buy
3. buy food
4. call manish
5. call karan
6. call tanmay
7. call sautrabh
You: delete fifth
Deleted: call manish
You: delete 2
Deleted: call karan
You: delete 100
Deleted: call tanmay
You: delete call sautrabh
Deleted: call sautrabh
You: exit
Bot: Exiting the interaction.
```

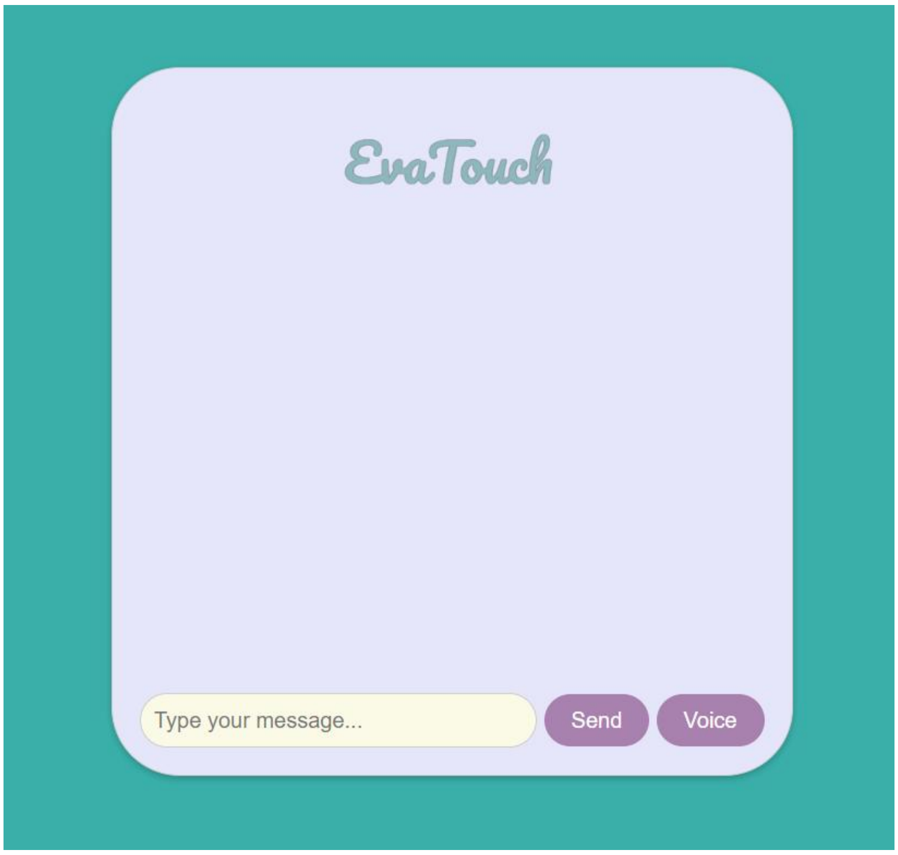
Search and Information model-

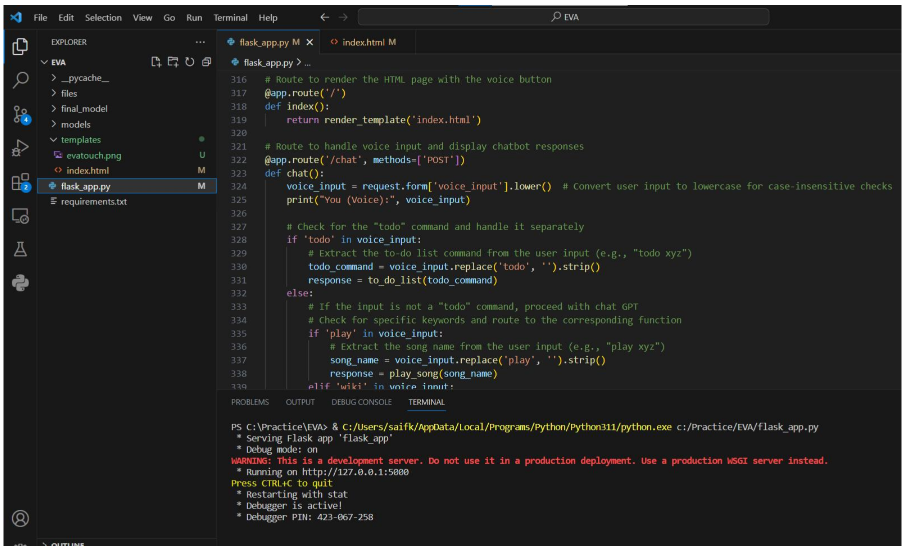


```python
# Load the saved TF-IDF model and associated preprocessing objects
model_filename = '/content/search_model.pkl'
loaded_model = joblib.load(model_filename)

# Load the saved TF-IDF vectorizer and label encoder
tfidf_vectorizer = joblib.load('/content/search_tfidf_filename.pkl')
stemmer = joblib.load('/content/search_stemmer_filename.pkl')

# Map 'Search' to 0 and 'Information' to 1
label_mapping = {'Search': 0, 'Information': 1}

# Test the model on new text data
new_text = ["Share some insights about space exploration missions."]
new_text_preprocessed = pd.Series(new_text).apply(preprocess_text)
new_text_tfidf = tfidf_vectorizer.transform(new_text_preprocessed)
predicted_class = loaded_model.predict(new_text_tfidf)

# Map predicted class back to label
predicted_label = list(label_mapping.keys())[list(label_mapping.values()).index(predicted_class[0])]

print(f"Predicted Intent: {predicted_label}")
```

```
Predicted Intent: Search
```

RESULTS

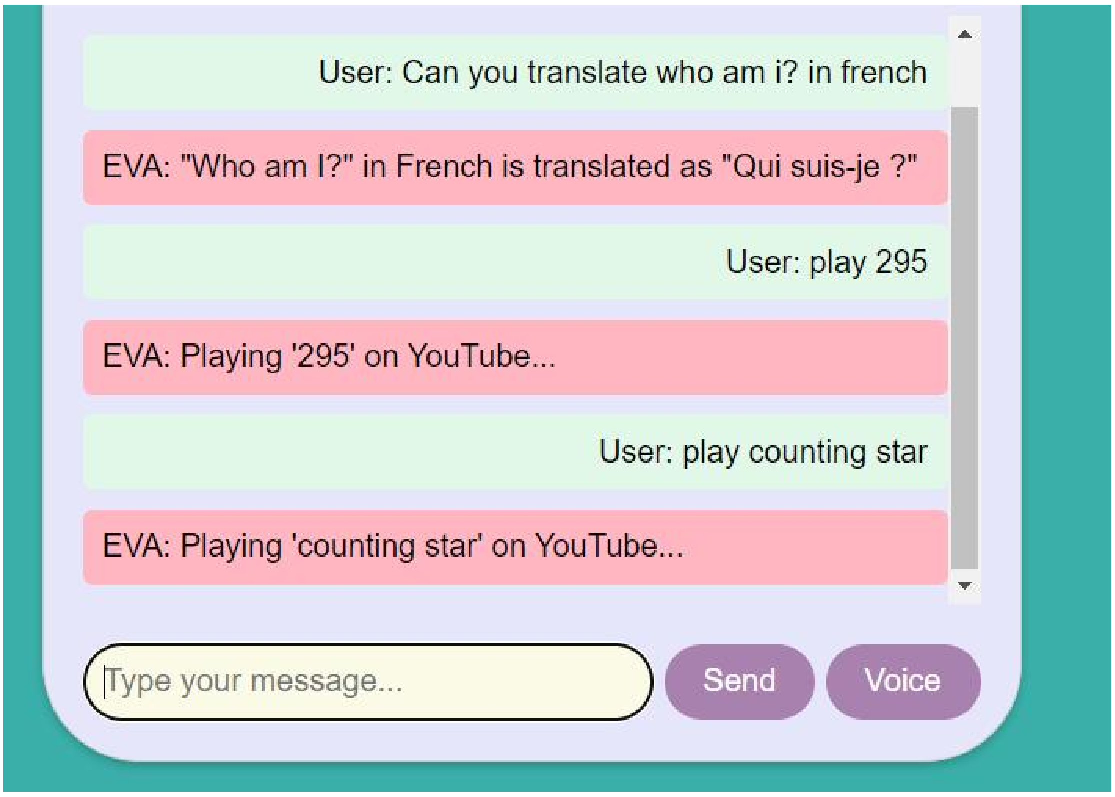




```
# Route to render the HTML page with the voice button
@app.route('/')
def index():
    return render_template('index.html')

# Route to handle voice input and display chatbot responses
@app.route('/chat', methods=['POST'])
def chat():
    voice_input = request.form['voice_input'].lower()  # Convert user input to lowercase for case-insensitive checks
    print("You (Voice):", voice_input)

    # Check for the "todo" command and handle it separately
    if 'todo' in voice_input:
        # Extract the to-do list command from the user input (e.g., "todo xyz")
        todo_command = voice_input.replace('todo', '').strip()
        response = to_do_list(todo_command)
    else:
        # If the input is not a "todo" command, proceed with chat GPT
        # Check for specific keywords and route to the corresponding function
        if 'play' in voice_input:
            # Extract the song name from the user input (e.g., "play xyz")
            song_name = voice_input.replace('play', '').strip()
            response = play_song(song_name)
        elif 'wiki' in voice_input:
```

```
PS C:\Practice\EVA> & C:/Users/saifk/AppData/Local/Programs/Python/Python311/python.exe c:/Practice/EVA/flask_app.py
 * Serving Flask app 'flask_app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 423-067-258
```

.


User: Can you translate who am i? in french

EVA: "Who am I?" in French is translated as "Qui suis-je ?"

User: play 295

EVA: Playing '295' on YouTube...

User: play counting star

EVA: Playing 'counting star' on YouTube...

CONCLUSION AND FUTURE SCOPE

In summary, project has set out to create a comprehensive and personalized AI assistant capable of integrating chat and voice interactions, performing arithmetic calculations, managing tasks, playing music, fetching information, scheduling appointments, delivering news and quotes, generating offering translations, and incorporating self-learning capabilities. This endeavor marks a notable advancement in addressing the productivity and convenience challenges encountered by individuals in today's fast-paced world, successfully achieving our objective of building an adaptable and user-centred AI assistant.

Looking ahead, there are several promising avenues for the future expansion and enhancement of this project:

Integration with Voice and Chat Interfaces - This Assistant could seamlessly integrate into various applications and devices, such as smart-phones, home automation systems, and automobiles, enabling users to interact with it effortlessly across different contexts. The project establishes the groundwork for a more interconnected and personalized future, where AI assistants become essential tools that simplify tasks, boost productivity, and ensure that technology caters to each user's unique preferences and needs.

Healthcare Applications: There is significant potential in leveraging the capabilities of our assistant for health-care purposes. It could be used to monitor medication schedules, provide health-related information, and offer valuable support in managing one's health.

In conclusion, our project not only fulfills its current objectives but also opens the door to an exciting future of possibilities, where AI assistance becomes an integral part of our daily lives, enriching our experiences and making them more efficient and tailored to individual needs.

REFERENCES

- Flask Documentation. https://flask.palletsprojects.com
- NLTK (Natural Language Toolkit) Documentation. https://www.nltk.org
- Wikipedia API Documentation. https://pypi.org/project/wikipedia-api/
- YouTube API Documentation. https://developers.google.com/youtube/registering_an_application
- NewsAPI Documentation. https://newsapi.org/docs
- Chatgpt https://www.researchgate.net/publication/367161545_Chatting_about_ChatGPT_How_may_AI_and_GPT_impact_academia_and_libraries
- OpenAI https://openai.com/research/overview