A PROJECT REPORT
ON
JARVIS: The AI voice Assistant

Submitted in the partial fulfilment of award of

BACHELOR OF TECHNOLOGY
(2023-2024)

Degree In

Computer Science and Engineering

Submitted To:
Dr. Nishant shrivastava

Submitted by:
Dilshan khan(8821103025)
Shadab khan(8821103016)
Mukul kumar sharma(8821103028)

DECLARATION

We do hereby declare that the report entitled "Jarvis-Personal-Assistant" submitted by us to japyee university, Anoopshahr inpartial of the requirement for the award of the degree of **B.TECH** in **COMPUTER SCIENCE AND ENGINEERING** is a record of bonafide project work carried out by us under the guidance of **Dr. Nishant shrivastva** and Department of Computer Science and Engineering.

Dilshan khan(8821103025)
Shadab khan(8821103016)
Mukul kumar sharma(8821103028)

CERTIFICATE

This is to certify that the project entitled “Jarvis: AI Voice Assistant” is a bonafide work undertaken by **Mr. Dilshan Khan** (8821103025), **Mr. Mukul Sharma** (8821103028), and **Mr. Shadab Khan** (8821103016) during the 5th Semester of B.Tech in Computer Science and Engineering at Jaypee University, Anoopshahr. The project was conducted under the guidance of **Dr. Nishant Shrivastva**.

This endeavor is presented in partial fulfillment of the requirements for the Degree of B.TECH. in **COMPUTER SCIENCE AND ENGINEERING** from Jaypee University, Anoopshahr.

Project Guide:

Dr. Nishant shrivastva

Department Of Computer
Science and Engineering

(Head of the Department)

INDEX

INTRODUCTION

An AI voice assistant, also referred to as a virtual or digital assistant, leverages voice recognition technology, natural language processing, and artificial intelligence (AI) to interact with users. Through advanced technology, the device processes user messages, deconstructs them, assesses their significance, and provides meaningful feedback.

Artificial intelligence enables these virtual assistants to engage in authentic conversations. They adeptly understand natural language voice commands and execute various tasks for users. These tasks encompass sending messages, answering phone calls, providing directions, delivering news and weather updates, accessing platforms like Google, YouTube, Stack Overflow, and more. Furthermore, they can answer queries, perform web scraping, play music, and undertake an array of activities.

Typically, AI voice assistants excel at executing straightforward tasks, such as adding events to calendars, offering information readily available through internet searches, or managing and monitoring smart home devices. They are capable of sending emails, setting alarms, providing weather reports, disclosing their location, conducting basic mathematical calculations, checking news, initiating music playback, and accessing different websites like Stack Overflow, YouTube, Facebook, and others.

These AI-powered marvels have become integral parts of our daily lives, streamlining routine activities and enhancing user experiences. As technology continues to evolve, the potential for AI voice assistants to contribute to our convenience and efficiency only expands.

PROBLEM DESCRIPTION

In the rapidly evolving technological landscape, virtual assistants such as Cortana, Siri, and Google Assistant have become integral parts of users' experiences on Windows, Android, and iOS platforms. These assistants seamlessly aid users in various tasks, making interactions more intuitive and efficient. However, there exists a notable gap in this paradigm — the Windows platform, often regarded as a cornerstone for developers, lacks a dedicated virtual assistant.

PURPOSE

The envisioned solution, Jarvis, aims to bridge this gap by providing a comprehensive AI assistant tailored specifically for the Windows platform. Jarvis will be designed to understand and respond to the intricacies of developer-centric tasks, offering a seamless and intelligent interaction experience.

1. Project Goal/Scope: Creating an Efficient Voice Assistant

The goal of our project is to develop a voice assistant designed to assist users in efficiently performing various tasks on their personal computers. The assistant operates through voice commands, minimizing the reliance on physical hardware. Key functionalities include opening applications and websites, playing media, providing time and date information, and personalized greetings based on the current time.

2. Integration of AI Technology for Enhanced Interactivity

We are actively integrating AI technology to enhance the interactivity and engagement of the assistant. The system's potential uses are vast, with programmable tasks ranging across various domains. As we continue development, our aim is to make the assistant a valuable tool, streamlining users' computer interactions and maximizing productivity.

3. Diverse Applications in Communication and Collaboration

In addition to its primary functions, AI virtual assistants can significantly improve communication And collaboration between individuals. The system can be programmed to translate languages, transcribe speech, and summarize text. Furthermore, it has the capability to facilitate group discussions and meetings.

4. Future Potential and Transformative Impact

As AI technology evolves, we anticipate witnessing more innovative and transformative applications for AI virtual assistants. These assistants have the potential to revolutionize how we live, work, and interact with the world around us.

Hardware Requirements:

Component	Recommendation
Processor (CPU)	i. For small to medium-scale projects: Modern multi-core processor (e.g., Intel Core i5 or i7, AMD Ryzen)
	ii. For larger projects: High-performance CPUs or GPUs for parallel processing (NVIDIA GPUs with CUDA)
Memory (RAM)	8 GB or more recommended. Larger models or datasets may require additional RAM.
Storage	SSDs are preferred. Storage amount depends on the dataset and models.
Graphics Processing Unit	Optional but beneficial for faster training and inference. NVIDIA GPUs commonly used.
Internet Connection	Reliable internet connection for downloading datasets, models, and updates.

Software Requirements:

Component	Recommendation
Operating System	Linux (e.g., Ubuntu) preferred, but AI projects can be developed on Windows or macOS.
Python	Python 3.x is recommended.
Development Environment	Use a development environment like Anaconda or virtual environments for dependency management.
Integrated Development Environment (IDE)	Choose from popular options like VSCode, PyCharm, or text editors like Sublime Text.
Version Control	Utilize Git for tracking changes in your codebase.
AI Libraries and APIs	Depending on the project, integrate with specific AI libraries or APIs (e.g., TensorFlow, PyTorch, OpenAI).
Text-to-Speech (TTS) Libraries	If the AI involves speech interactions, use TTS libraries like Google Text-to-Speech or pyttx3.
Speech-to-Text (STT) Libraries	For speech recognition, consider STT libraries like Google Speech-to-Text or speech-recognition.

Tools and Techniques for Building an AI Assistant

1. Speech Recognition:

Library: speechRecognition

Description: This library enables the conversion of audio into text for further processing. It is a crucial component for understanding and interpreting user commands through voice input.

Installation:

```
pip install SpeechRecognition
```

2. Text-to-Speech Conversion:

Library: pyttsx3

Description: Pyttsx3 is a cross-platform text-to-speech library that is platform-independent. The significant advantage of using this library is its offline functionality, making it a reliable choice for converting text into spoken words.

Installation:

Pip install pyttsx3

3. Operating System Interaction:

Module: os (built-in in Python)

Description: The os module in Python provides essential methods for interacting with the operating system. This includes tasks such as creating files and directories, managing files and directories, handling input and output, managing environment variables, and process management. It plays a crucial role in the overall functionality of the voice assistant.

4. OpenAI Integration:

Library: openai

Description: OpenAI provides a powerful platform for building AI models and integrating them into applications. In the context of a voice-based assistant, OpenAI could be utilized for advanced natural language processing, contextual understanding, and providing more sophisticated responses to user queries.

Installation:

Pip install openai

5. Web Requests:

Library: requests

Description: The requests library is crucial for making HTTP requests, enabling the voice assistant to interact with web services, APIs, and fetch dynamic information from the internet.

Installation:

Pip install requests

6. Graphical User Interface (GUI) Development:

Library: Tkinter (built-in in Python)

Description: Tkinter is the standard GUI toolkit included with Python. It facilitates the creation of graphical user interfaces, allowing for the development of interactive and user-friendly components for the voice assistant.

7. Accessing Wikipedia Information:

Library: wikipedia

Description: The wikipedia library provides an interface for interacting with Wikipedia articles. It allows the voice assistant to access a vast repository of information, making it a valuable tool for knowledge retrieval.

Installation:

```
pip install wikipedia
```

8. Date and Time Handling:

Module: datetime (built-in in Python)

Description: The datetime module in Python provides functions to manipulate dates and times. It is essential for incorporating time-related features into the voice assistant, such as providing the current date and time.

9. Audio Input/Output:

Library: PyAudio

Description: PyAudio is a Python library for handling audio input and output. It is useful for capturing voice commands through a microphone and playing back audio responses. PyAudio is a valuable tool for building the voice interaction aspect of the assistant.

Installation:

```
pip install pyaudio
```

Techniques for Building an AI Assistant

1. Speech Recognition:

Description: Convert spoken language into text. This involves processing audio input to recognize and transcribe spoken words.

Tools/Technologies: SpeechRecognition library, Google Cloud Speech-to-Text, CMU Sphinx.

2. Natural Language Processing (NLP):

Description: Understand and interpret the meaning of text or spoken language. NLP techniques are used for tasks like sentiment analysis, entity recognition, and language understanding.

Tools/Technologies: spaCy, NLTK, Rasa NLU, Dialogflow.

3. Text-to-Speech (TTS):

Description: Convert text into spoken words. TTS is essential for enabling the assistant to communicate with users through voice.

Tools/Technologies: pyttsx3, gTTS (Google Text-to-Speech), Amazon Polly.

4. Intent Recognition:

Description: Identify the user's intent based on their input. This involves determining what action the user wants the assistant to perform.

Tools/Technologies: Rasa, Dialogflow, LUIS (Language Understanding Intelligent Service).

5. Dialog Management:

Description: Manage the flow of conversation and context. Keep track of the ongoing interaction to provide coherent and contextually relevant responses.

Tools/Technologies: Rasa, Microsoft Bot Framework, ChatterBot.

6. Knowledge Base Integration:

Description: Integrate with external knowledge bases or databases to enhance the assistant's knowledge and ability to answer user queries.

Tools/Technologies: Wikipedia API, Knowledge Graphs.

7. Machine Learning for Personalization:

Description: Use machine learning techniques to personalize the assistant's responses based on user preferences and historical interactions.

Tools/Technologies: TensorFlow, scikit-learn.

8. Sentiment Analysis:

Description: Analyze the sentiment of user input. Understand whether the user's tone is positive, negative, or neutral.

Tools/Technologies: TextBlob, VADER sentiment analysis.

9. Continuous Learning:

Description: Implement mechanisms for the assistant to learn and improve over time based on user feedback and new data.

Tools/Technologies: Reinforcement learning, online learning.

10. Error Handling and Graceful Degradation:

Description: Handle user queries that the assistant cannot understand or process gracefully, providing informative responses.

Tools/Technologies: Custom error handling logic.

These techniques are often combined and customized based on the specific requirements and functionalities of the AI assistant being developed. The choice of techniques depends on factors such as the complexity of the assistant, the desired user experience, and the available resources.

Module Design for AI Assistant

1. Introduction

1.1 Purpose

I. The purpose of this document is to outline the modular design of the AI Assistant system. The AI Assistant is designed to provide users with intelligent responses, perform various tasks, and adapt to user preferences over time.

1.2 Scope

II. This module design encompasses the core functionalities of the AI Assistant, including natural language processing, task execution, user interaction, and learning capabilities.

2. Modules

2.1 User Interaction Module

2.1.1 Purpose

I. Handles user input and provides output in a user-friendly manner.

2.1.2 Functions

I. Receive and process user input.

II. Generate human-like responses.

III. Manage user preferences and settings.

2.1.3 Dependencies

I. Natural Language Processing (NLP) Module.

2.2 Natural Language Processing (NLP) Module

2.2.1 Purpose

I. Analyzes and understands natural language input from users.

2.2.2 Functions

I. Tokenization and parsing of user input. II. Intent recognition. III. Named entity recognition.

2.2.3 Dependencies

I. External NLP libraries (e.g., spaCy, NLTK).

2.3 Task Execution Module

2.3.1 Purpose

I. Executes tasks based on user requests and preferences.

2.3.2 Functions

I. Task planning and scheduling.

II. Interaction with external APIs for task execution.

III. Error handling and recovery.

2.3.3 Dependencies

I. User Interaction Module.

II. External APIs for specific tasks (e.g., weather API, calendar API).

2.4 Learning and Adaptation Module

2.4.1 Purpose

Enables the AI Assistant to learn and adapt to user behavior over time.

2.4.2 Functions

I. User behavior analysis.

II. Model training for personalized responses.

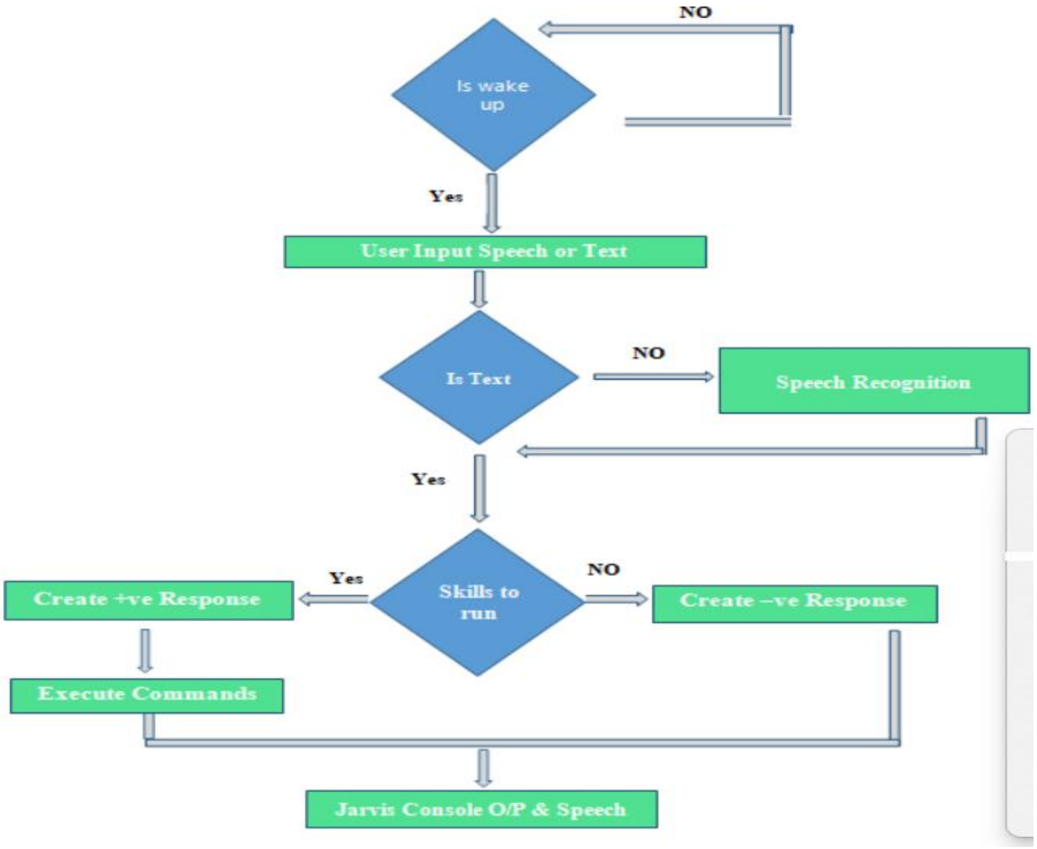
III. Continuous improvement of response accuracy.

2.4.3 Dependencies

I. User Interaction Module.

II. Data storage for user profiles and learning models.

2.5 Entity-Relationship (ER) Diagram



2.5.1 Purpose

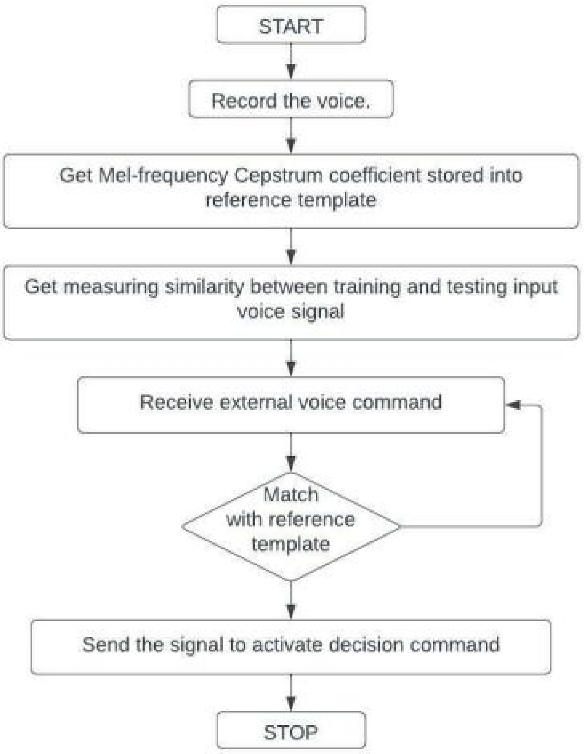
Illustrates the relationships between different entities in the AI Assistant system.

2.5.2 Components

I. Entities representing key elements (e.g., User, Task, Interaction).

II. Relationships depicting connections between entities.

2.6 Flowchart


START
Record the voice.
Get Mel-frequency Cepstrum coefficient stored into reference template
Get measuring similarity between training and testing input voice signal
Receive external voice command
Match with reference template
Send the signal to activate decision command
STOP

2.6.1 Purpose

Visual representation of the overall flow of processes in the AI Assistant system.

2.6.2 Description

I. The flowchart provides a graphical depiction of how user input is processed, tasks are executed, and the AI Assistant adapts over time.

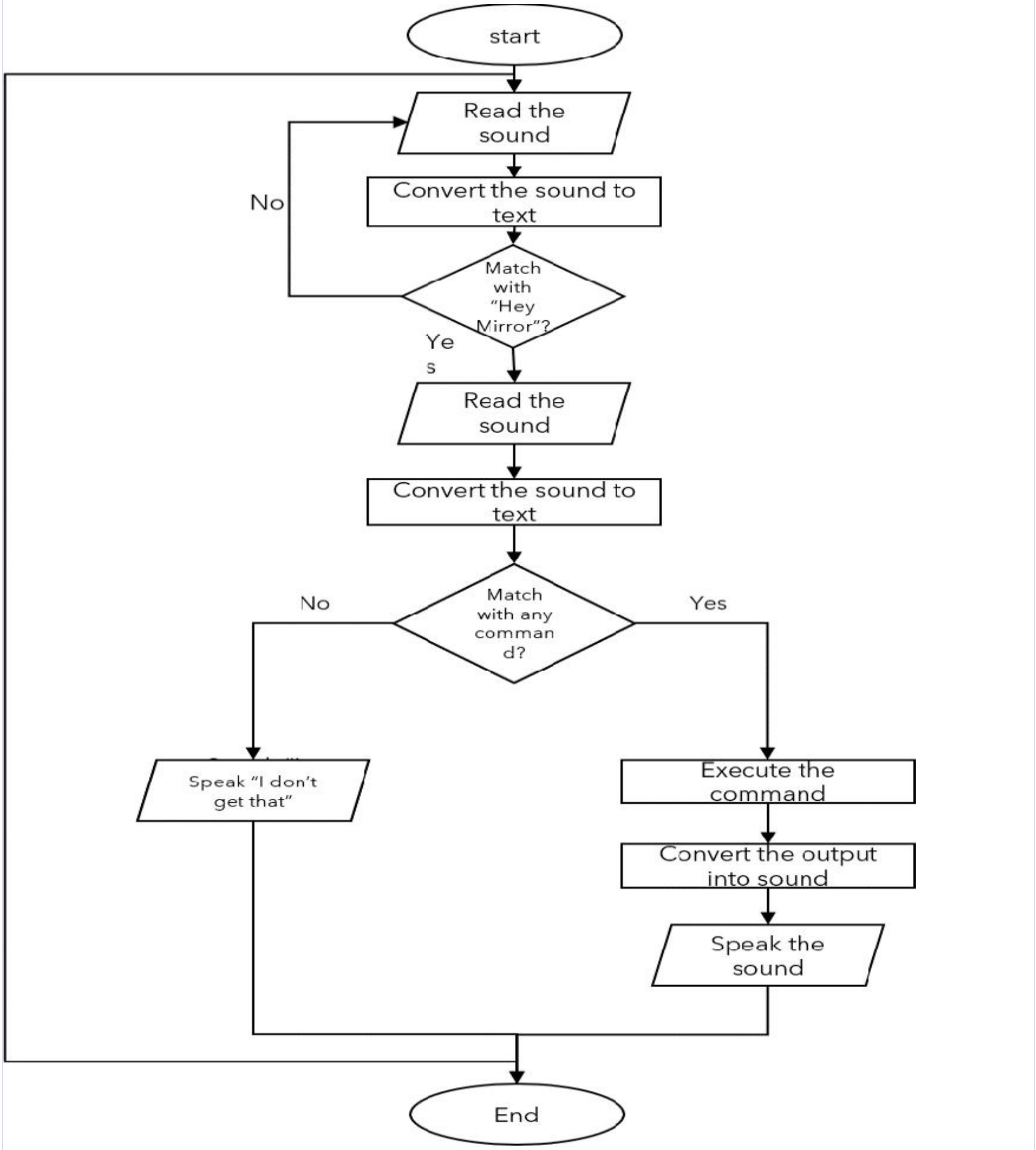
II. It showcases the sequential steps and decision points in the system's workflow.

3 Interaction

3.1Interaction Flow

I. Describe the flow of information and control between modules during typical user interactions**.**

3.2 Data Flow. Illustrate how data is passed between modules, including input and **output formats.**



Implementation Details for AI Jarvis Assistant

Language used: Python 3

Import modules

- Modules used :
- pyttsx3 (imports voices and has functions related to
- speaking)
- datetime (#not important .)
- speech_recognition (to convert speech to text)
- wikipedia (to access Wikipedia information)
- webbrowser (to manipulate web browsing operations)
- os (for just os.clear())
- webbrowser (for playing songs on youtube)
- Openai(know about any educational knowledge eg:calculation,coding)
- Requests(for fetching data)
- Tkinter (for graphical user interface)

1. Define Functionalities:

1.1 Voice Recognition:

Voice recognition is a fundamental functionality that empowers Jarvis to understand and process user commands through spoken language. This involves integrating a robust voice recognition library or API, such as Google Speech Recognition, to effectively convert audio input into textual commands.

1.2 Natural Language Processing (NLP):

NLP is a crucial aspect that allows Jarvis to comprehend and interpret user instructions in a human-like manner. Advanced pre-trained models like OPENAI or GPT-3 are utilized to process user input, enabling the system to understand context, extract entities, and discern the underlying meaning.

1.3 Task Automation:

Jarvis is designed to perform various tasks seamlessly, such as sending emails, opening applications, and fetching information. This involves the implementation of modular components tailored for specific tasks. Conditional statements and algorithms are employed to determine the appropriate action based on the user's input.

1.4 User Interface:

A user-friendly interface is developed to facilitate interaction between users and Jarvis. This interface can be graphical or text-based, providing a platform for users to input commands, receive responses, and interact seamlessly with Jarvis.

2. Procedural Steps:

2.1 Step 1: Voice Recognition

I. Integration of a reliable voice recognition library or API.

II. Configuration of the system to capture user voice commands using a microphone.

2.2 Step 2: Natural Language Processing

i. Implementation of a pre-trained NLP model (e.g., OPENAI, GPT-3).

ii. Utilization of named entity recognition techniques to extract relevant information from user input.

2.3 Step 3: Task Automation

I. Modular implementation for specific tasks like opening applications, and web scraping.

II. Use of conditional statements and algorithms to determine appropriate actions based on user input.

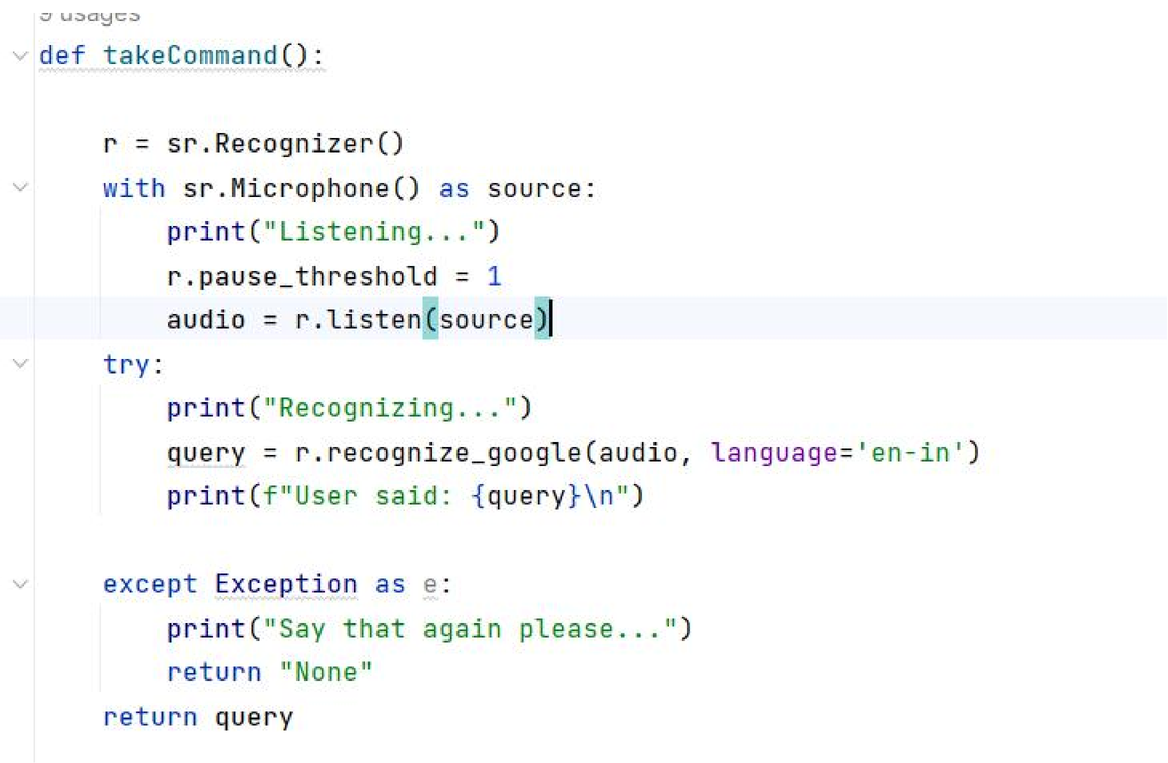
2.5 Step 5: User Interface

I. Creation of an intuitive and user-friendly interface for users to interact with Jarvis.

II. Display of relevant information and responses in a clear and understandable format.

3. Coding:

3.1 Takecommand fuction(it’s responsible to take command by the user)



```
9 usages
def takeCommand():

    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
        r.pause_threshold = 1
        audio = r.listen(source)
    try:
        print("Recognizing...")
        query = r.recognize_google(audio, language='en-in')
        print(f"User said: {query}\n")

    except Exception as e:
        print("Say that again please...")
        return "None"
    return query
```

3.2 Python Code for Voice Recognition:

```
import speech_recognition as sr


def speech_to_text():
    r = sr.Recognizer()


    try:
        with sr.Microphone() as source:
            print("Say something:")
            audio = r.listen(source)


        voice_data = r.recognize_google(audio)
        print("You said:", voice_data)
        return voice_data


    except sr.UnknownValueError:
        print("Error: Could not understand audio")

    except sr.RequestError as e:
        print(f"RequestError: {e}")

    except OSError as e:
        print(f"OSError: {e}")
        print("No default input device available. Check your microphone settings.")
        return None
```

3.3 Python Code for GUI:

```
root = Tk()
root.title("AI Assistant")
root.geometry("550x675")
root.resizable(False, False)
root.config(bg="#6F8FAF")
frame = LabelFrame(root, padx=100, pady=7, borderwidth=3)
frame.config(bg="#6F8FAF", relief="raised")
frame.grid(row=0, column=1, padx=55, pady=10)
image_path = "jarvish.jpg"
original_image = Image.open(image_path)
resized_image = original_image.resize((300, 300))
image = ImageTk.PhotoImage(resized_image)
image_label = Label(frame, image=image)
image_label.grid(row=1, column=0, pady=20)
text = Text(root, font=('courier 10 bold'), bg="#356696")
text.place(x=100, y=375, width=375, height=100)
entry = Entry(root, justify=CENTER)
entry.place(x=100, y=500, width=350, height=30)
```

```
button1 = Button(root, text="Ask", bg="#356696", pady=16, padx=40, bd=3,
relief=SOLID, command=Ask)
button1.place(x=100, y=550, width=100, height=30)

button2 = Button(root, text="Delete", bg="#356696", pady=16, padx=40, bd=3,
relief=SOLID, command=delete)
button2.place(x=220, y=550, width=100, height=30)
button3 = Button(root, text="Send", bg="#356696", pady=16, padx=40, bd=3,
relief=SOLID, command=send)
button3.place(x=340, y=550, width=100, height=30)

root.mainloop()
```

3.4 Python Code for Task Automation:

```
import webbrowser
def action(user_data):
    user_data = user_data.lower()
    if "what is your name" in user_data:
        return "My name is virtual assistant"
    elif "hello" in user_data or "hey" in user_data:
        return "Hey, sir. How can I help you?"
    elif "good morning" in user_data:
        return "Good morning, sir."
    elif "time now" in user_data:
        current_time = datetime.datetime.now()
        time_str = f"Hour: {current_time.hour}, Minute: {current_time.minute}"
        return f"Time now: {time_str}"
    elif "shutdown" in user_data:
        return "OK, sir. Shutting down."
    elif "play music" in user_data:
        webbrowser.open("https://gaana.com/")
        return "Gaana.com is now ready for you."
    elif "youtube" in user_data:
        webbrowser.open("https://youtube.com/")
        return "YouTube.com is now ready for you."

    elif "open google" in user_data:
        webbrowser.open("https://google.com/")
        return "Google.com is now ready for you."
```

3.5 python code for weather information

```
                import requests

api_key = '9013d84cc21a448d0cf15ca130f68949'
url = 'http://api.openweathermap.org/data/2.5/weather?q={}&appid={}'
def get_weather_info(city):
    result = requests.get(url.format(city, api_key))
    if result.status_code == 200:
        data = result.json()
        city_name = data['name']
        country = data['sys']['country']
        weather_desc = data['weather'][0]['description']
        temperature = data['main']['temp']
        min_temperature = data['main']['temp_min']
        max_temperature = data['main']['temp_max']
        humidity = data['main']['humidity']
        weather_info = {
            'city': city_name,
            'country': country,
            'weather_desc': weather_desc,
            'temperature': temperature,
            'min_temperature': min_temperature,
            'max_temperature': max_temperature,
            'humidity': humidity,
        }

        return weather_info
    else:
        return None
```

3.6 python code for openai:

```
     import openai
import text_to_speech
openai.api_key = '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'
response_spoken = False

def get_ai_response(query):
    global response_spoken
    try:
        if not response_spoken:
            response = openai.Completion.create(
                engine="text-davinci-002",
                prompt=query,
                max_tokens=50
            )
            ai_response = response.choices[0].text.strip()
            text_to_speech.speak_text(ai_response)
            response_spoken = True  # Set the flag to True after speaking once
            return ai_response
        else:
            return "AI response already spoken"

    except Exception as e:
        print(f"Error in OpenAI query: {e}")
        return "Error in OpenAI query"
```

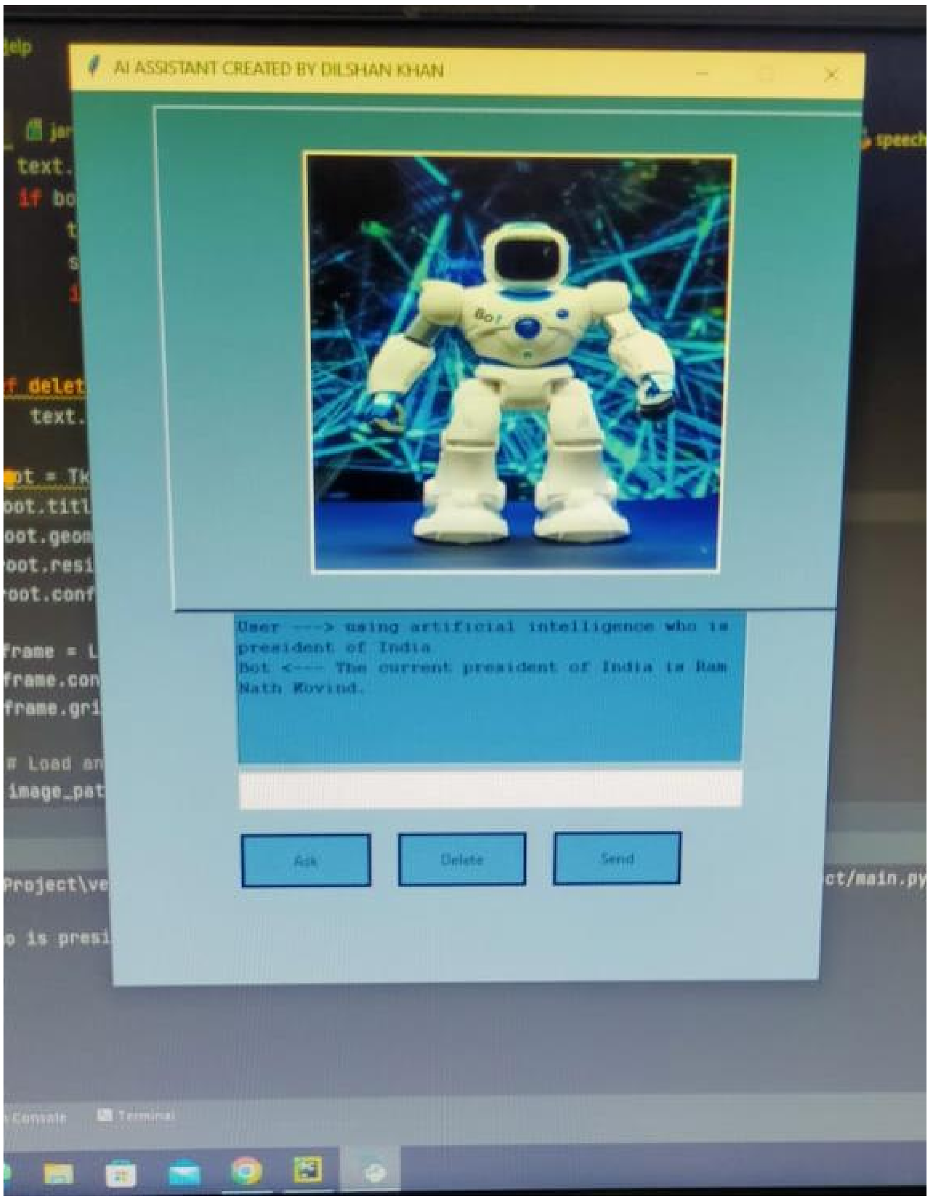
3.7 complete code:

```
        from tkinter import *
from PIL import ImageTk, Image
import speech_to_text
import text_to_speech
import action

# Define the speak_text function
def speak_text(text):
    engine = text_to_speech.pyttsx3.init()
    rate = engine.getProperty('rate')
    engine.setProperty('rate', rate - 70)
    try:
        engine.say(text)
        engine.runAndWait()
    except Exception as e:
        print(f"Error in text-to-speech: {e}")

def Ask():
    user_data = speech_to_text.speech_to_text()
    bot_value = action.action(user_data)
    text.insert(END, 'User ---> ' + user_data + "\n")
    if bot_value is not None:
        text.insert(END, "Bot <--- " + bot_value + "\n")
        text_to_speech.speak_text(bot_value)
        if bot_value.lower() == "ok sir":
            root.destroy()
def send():
    sent = entry.get()
    bot = action.action(sent)
    text.insert(END, 'User ---> ' + sent + "\n")
    if bot is not None:
        text.insert(END, "Bot <--- " + bot + "\n")
        speak_text(bot)  # Speak the bot's response
        if bot.lower() == "ok sir":
            root.destroy()


def delete():
    text.delete('1.0', END)


root = Tk()
root.title("AI Assistant")
root.geometry("550x675")
root.resizable(False, False)
root.config(bg="#6F8FAF")


frame = LabelFrame(root, padx=100, pady=7, borderwidth=3)
frame.config(bg="#6F8FAF", relief="raised")
frame.grid(row=0, column=1, padx=55, pady=10)
image_path = "jarvish.jpg"
original_image = Image.open(image_path)
```

```
resized_image = original_image.resize((300, 300))
image = ImageTk.PhotoImage(resized_image)
image_label = Label(frame, image=image)
image_label.grid(row=1, column=0, pady=20)
text = Text(root, font=('courier 10 bold'), bg="#356696")
text.place(x=100, y=375, width=375, height=100)
entry = Entry(root, justify=CENTER)
entry.place(x=100, y=500, width=350, height=30)
button1 = Button(root, text="Ask", bg="#356696", pady=16, padx=40, bd=3, relief=SOLID, command=Ask)
button1.place(x=100, y=550, width=100, height=30)
button2 = Button(root, text="Delete", bg="#356696", pady=16, padx=40, bd=3, relief=SOLID, command=delete)
button2.place(x=220, y=550, width=100, height=30)
button3 = Button(root, text="Send", bg="#356696", pady=16, padx=40, bd=3, relief=SOLID, command=send)
button3.place(x=340, y=550, width=100, height=30)

root.mainloop()
import requests
api_key = '9013d84cc21a448d0cf15ca130f68949'
url = 'http://api.openweathermap.org/data/2.5/weather?q={}&appid={}'
def get_weather_info(city):
    result = requests.get(url.format(city, api_key))
    if result.status_code == 200:
        data = result.json()
        city_name = data['name']
        country = data['sys']['country']
        weather_desc = data['weather'][0]['description']
        temperature = data['main']['temp']
        min_temperature = data['main']['temp_min']
        max_temperature = data['main']['temp_max']
        humidity = data['main']['humidity']
        weather_info = {
            'city': city_name,
            'country': country,
            'weather_desc': weather_desc,
            'temperature': temperature,
            'min_temperature': min_temperature,
            'max_temperature': max_temperature,
            'humidity': humidity,
        }
        return weather_info
    else:
        return None

import openai
import text_to_speech
openai.api_key = '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'
response_spoken = False
def get_ai_response(query):
    global response_spoken
    try:
        if not response_spoken:
            response = openai.Completion.create(
                engine="text-davinci-002",
                prompt=query,
                max_tokens=50
            )
            ai_response = response.choices[0].text.strip()
            text_to_speech.speak_text(ai_response)  # Speak the AI response
            response_spoken = True  # Set the flag to True after speaking once
            return ai_response
        else:
            return "AI response already spoken"

    except Exception as e:
        print(f"Error in OpenAI query: {e}")
        return "Error in OpenAI query"
import pyttsx3
def speak_text(text):
    engine = pyttsx3.init()
    rate = engine.getProperty('rate')  # Corrected method name
    engine.setProperty('rate', rate - 70)  # Corrected method name and value
    try:
        engine.say(text)
        engine.runAndWait()
    except Exception as e:
        print(f"Error in text-to-speech: {e}")
     elif "open google" in user_data:
        webbrowser.open("https://google.com/")
        return "Google.com is now ready for you."
    elif "weather" in user_data:
        default_location = "delhi"
        ans = weather.get_weather_info(default_location)
        if ans:
            text_to_speech.speak_text(f"Weather in {default_location}:
{ans['temperature']}°C, {ans['weather_desc']}")
            return f"Weather in {default_location}: {ans['temperature']}°C,
{ans['weather_desc']}"
```

```
    elif "using artificial intelligence" in user_data:
        ai_query = user_data.replace("using artificial intelligence", "").strip()
        # For other queries, use OpenAI to generate a response
        ai_response = ai_module.get_ai_response(ai_query)
        if ai_response:
            text_to_speech.speak_text(ai_response)
            return ai_response
        else:
            return "Error in generating AI response’’
    else:
        return "I'm sorry, I didn't understand that."
import speech_recognition as sr
def speech_to_text():
    r = sr.Recognizer()
    try:
        with sr.Microphone() as source:
            print("Say something:")
            audio = r.listen(source)
        voice_data = r.recognize_google(audio)
        print("You said:", voice_data)
        return voice_data
    except sr.UnknownValueError:
        print("Error: Could not understand audio")
        return None
    except sr.RequestError as e:
        print(f"RequestError: {e}")
        return None
    except OSError as e:
        print(f"OSError: {e}")
        print("No default input device available. Check your microphone settings.")
    return None
```

4. output screen:



```
User ---> using artificial intelligence who is
president of India
Bot <--- The current president of India is Ram
Nath Kovind.
```

Jarvis Output Screen

Description: The output screen serves as a visual representation of Jarvis's actions. It displays the results of user commands, showcases feedback messages, and may indicate learning messages.

This comprehensive overview establishes the foundation for implementing an AI Jarvis assistant with a focus on voice recognition, NLP, task automation, and a user interface. The code snippets provided serve as a starting point, and customization can be applied based on specific use cases and preferences. Iterative improvements and enhancements can be made to continually refine Jarvis's functionality and user experience.

Limitations and Future Enhancement for AI Jarvis Assistant

Limitations:

1. Dependency on Internet Connectivity:

AI Jarvis Assistant heavily relies on internet connectivity for certain tasks such as fetching real-time information, accessing online databases, or performing web scraping. Lack of internet access may limit its functionality in offline scenarios.

2. Speech Recognition Challenges:

The accuracy of voice recognition can be affected by background noise, accents, or unclear pronunciation. Improvements in handling diverse speech patterns and environmental factors are crucial for a more robust user experience.

3. Limited Context Understanding:

While AI Jarvis uses advanced NLP models, its context understanding is not flawless. It may struggle with complex queries or maintaining context over extended conversations, leading to misinterpretations or incorrect responses.

4. Security Concerns:

Handling sensitive information such as personal data, emails, or user preferences raises security concerns. Ensuring robust encryption and privacy measures is imperative to prevent unauthorized access.

5. Task-Specific Learning Time:

The learning capability of Jarvis may require significant time and user interactions to adapt to specific tasks and preferences. Enhancements in quick adaptability and efficient learning algorithms are essential.

Future Enhancements:

1. Offline Functionality:

Implementing offline functionality would enhance Jarvis's usability in scenarios where internet connectivity is limited. This involves incorporating local databases or offline modes for essential tasks.

2. Advanced Speech Recognition:

Investing in cutting-edge speech recognition technologies, such as deep learning models tailored for diverse accents and noise environments, can significantly improve the accuracy of voice commands.

3. Contextual Understanding Improvements:

Further development of NLP models with a focus on contextual understanding will enable Jarvis to handle complex queries, maintain context over extended conversations, and provide more accurate responses.

4. Enhanced Security Measures:

Implementing advanced encryption techniques, secure user authentication, and regular security audits will fortify Jarvis against potential security threats, ensuring the confidentiality of user data.

5. Efficient Learning Algorithms:

Optimizing the learning algorithms to reduce the time required for Jarvis to adapt to user preferences and tasks will contribute to a more seamless and personalized user experience.

References

1. Website referred

- www.greeksforgreeks.com
- www.codewithharry.com
- www.stackoverflow.com

2. Youtube channel referred

- Code with harry
- Provider
- Kaushik shreshta

3. **Book referred**

- Python gui programming with tkinter by Alan D. Moore