CHAPTER 1

INTRODUCTION

Artificial intelligence demonstrates our potential to think like humans when applied to robots. In this case, a computer system is built in a way that usually calls for human intervention. Since Python is a newly developed language, writing a voice assistant script with it is simple. The user can customize the assistant's instructions to suit their needs. Voice recognition refers to Alexa, Siri, and so on. Speech Recognition is an API in Python that lets us translate speech to text. Creating my own helper was an intriguing effort. It got simpler to use voice commands to accomplish common chores like playing music and launching your preferred IDE, as well as to send emails without typing a single word and search on Google without opening a browser. The present state of technological advancements allows them to accomplish any work as successfully as, if not more so than, us. I discovered while working on this project that artificial intelligence (AI) reduces human labor and saves time in all fields.

Because artificial intelligence is being used by the voice assistant, the results are extremely accurate and effective. The assistant eliminates the need for typing entirely and acts as a second person who we are speaking with and requesting to execute tasks for. This helps to save human labor and time while completing any activity. Although the assistant is just as good as a human one, we might argue that it is more productive and efficient for any given activity. The libraries and packages utilized in the creation of this assistance minimize time and concentrate on time complications.

Among its features are its ability to send emails and view PDFs. It's capable of texting on WhatsApp. It can launch notepad, your preferred IDE, command prompt, etc. It has a music player. It can look up topics on Wikipedia for you. It may launch web browser tabs for websites such as Google, YouTube, and others. It may provide you with a desktop reminder of your choosing and a weather prediction. It is capable of some simple dialogue.

The PyCharm IDE was used to build all of the py files for this project, along with other tools and

technologies.In addition, I included the following libraries and modules in my project. pywhatkit, pyjokes, pyPDF2, pyautogui, pyQt, SpeechRecognition, Datetime, Wikipedia, and pyttsx3, among others. In order to engage with JARVIS, I have developed a live graphical user interface (GUI), which provides an engaging visual for the interaction.

1.1 Current Framework

There are a lot of virtual assistants on the market right now, including Alexa from Amazon, Siri from Apple, and Google Assistant. Because of these systems' deep integration into their corresponding ecosystems, users may interact with them across devices with ease. They do, however, have several drawbacks, including a reliance on particular technology, privacy issues, and a lack of customization choices for users who want to personalize the assistant to suit their own requirements.

Current solutions often need constant internet access and frequently transmit user data to cloud servers, posing privacy risks. Moreover, they are frequently closed-source, which restricts developers' ability to alter or add functionality beyond what the manufacturer offers.

1.2 The Suggested Framework

The suggested solution is to use Python to create a highly configurable personal assistant in order to overcome the shortcomings of current virtual assistants. With an emphasis on offline functionality and privacy, this assistant will provide customers more freedom and control.

The following are the suggested system's salient features:

- **Open-Source and Customizable:** Because the assistant will be open-source, users and developers will be able to add to and alter its features to better meet their requirements.

- **Offline Functionality:**

    - Unlike many current virtual assistants that rely on constant internet access, this assistant will function offline for its core operations, thereby addressing privacy concerns and ensuring reliability in areas with limited internet connectivity.
    - This feature makes the assistant more secure, as sensitive data and commands are processed locally without the need to transmit information to cloud servers.

- **Enhanced Privacy:**

    - By minimizing the dependence on cloud services, the assistant ensures that user data remains private and secure. Users have complete control over their data, reducing the risk of unauthorized access and data breaches.
    - Privacy-conscious users can trust that their interactions with the assistant are not being monitored or recorded by third parties.

- **Modular Architecture:**

    - The system will be designed with a modular architecture, allowing for easy addition, removal, and modification of features. This enables users to customize the assistant's capabilities based on their specific requirements.
    - Modules can be developed for various functionalities such as email management, media control, web browsing, and more, providing a flexible and scalable solution.

CHAPTER 2

SYSTEM DEIGN

2.1 INFORMATION TRANSFER

For interactions between the user and the assistant to be seamless and effective, the personal assistant system's data flow is essential. The flow consists of several important phases, each involving certain parts and procedures. This is an in-depth analysis of the data flow within the system:

1. Audio Input:

- **Part Number:** Microphone
- **Procedure:** First, the user's spoken command is recorded by the system via a microphone. The system's principal input device is the microphone.
- **Specifics:** The circuitry in the microphone records the user's voice in analog waves and transforms them into a digital signal. The voice recognition module then receives this digital signal for additional processing.

2. Speech Recognition:

- Speech Recognition Library (such as the SpeechRecognition library in Python) is a component.
- **Procedure:** The system turns the recorded voice input into text by using a speech recognition library. This is an important step because it converts spoken words into a format that the system can comprehend and use.
- **Specifics**: After processing the digital audio stream, the speech recognition library extracts the linguistic content. It recognizes the words and phrases that

the user speaks using a variety of models and algorithms. The text that results from speech recognition is sent to the command processing module.

3. Processing Commands:

- **Component:** Command Processing Module
- **Process:** This module determines the proper course of action by analyzing the text input. In order to find words and phrases that meet preset instructions, the text must be parsed.
- **Specifics:** Natural language processing (NLP) methods are used by the command processing module to interpret the context and intent of user commands. It maps the detected text to certain actions using a preset set of instructions and rules. The system recognizes the terms "weather" and "today" and maps them to the weather prediction function, for example, if the user asks, "What's the weather like today?"

4. Execution of Action:

- **Part:** Module for Action Execution Procedure Using the appropriate library or API, the system does the necessary task based on the command found in the previous phase. In this phase, the particular function linked to the command is executed.
- **Specifics:** To complete the intended job, the action execution module communicates with a number of libraries and APIs. As an illustration:
    - In the event that the command is to send an email, the system connects to an SMTP server via the smtplib library and sends the email.

- When the command is to play music, the OS library is used by the system to find and launch the music file.
- If the command is to obtain weather information, the system calls a weather API using the requests library to get the current weather.

5. Voice Output:

- **Component**: Text-to-Speech (TTS) Engine (e.g., `pyttsx3` library in Python)
- **Process**: After executing the action, the system provides feedback to the user through a text-to-speech engine, confirming the action or providing the requested information.

- **Details**: The TTS engine converts the text response generated by the system into spoken words. It synthesizes the text into an audio signal that is played back to the user through speakers. For example, if the user asked for the current time, the system would respond with "The current time is 3:45 PM" using the TTS engine.

CHAPTER 3

SPECIFICATION OF SOFTWARE

3.1 Libraries for Python

Several Python libraries are used in the personal assistant's implementation; each one offers unique characteristics that enhance the system's overall capabilities. Below is a thorough explanation of every library that was used in the project:

1. Pyttsx3:

- **Objective:** Text-to-Speech (TTS)
- **Conversion Specifications:** pyttsx3 is an offline text-to-speech conversion library compatible with Windows, macOS, and Linux. To translate text into spoken words, it makes use of the TTS engines that are accessible on the relevant platforms. The assistant may offer the user audio feedback and replies by customizing the speech pace, volume, and voice thanks to the library.
- **Use Case:** Delivering auditory feedback, reading aloud from text, and speaking in response.

2. Speech Recognition:

- **Goal:** Speech Recognition and Processing
- **Specifications:** User voice input is captured and processed using the SpeechRecognition library. It is compatible with several voice recognition engines and APIs, such as Microsoft Bing Voice Recognition, CMU Sphinx, Google Web voice API, and others. The assistant can comprehend and interpret voice instructions thanks to the library's ability to translate spoken words into text.
- **Use Case:** Speech to text conversion for command processing, voice command capture.

3. Wikipedia:

- **Goal:** Wikipedia Synopsis Fetching
- **Details:** To obtain Wikipedia material, use the straightforward interface provided by the Wikipedia library. The assistant may get subject summaries from Wikipedia thanks to this feature, which makes it a helpful tool for instant information retrieval. The library has the ability to look for publications, get article summaries, and obtain in-depth data on certain subjects.
- **Use Case:** Extracting data from Wikipedia and giving prompt responses to user inquiries.

4. web browser:

- **Goal:** Using the Default Web Browser to Open URLs
- **Details:** To open URLs in the built-in web browser, use the webbrowser library, a standard Python package. It can launch a URL-specified web page and supports a number of different browsers. This library makes it easier to search and browse the web by enabling the assistant to go to certain websites when the user requests it.
- **Use Cases:** Accessing internet resources, launching webpages, and doing web searches.

5. OS:

- The goal is to communicate with the operating system.
- **Details:** An interface to communicate with the operating system is provided by the standard Python module known as the os library. It has features for managing directories and files, carrying out system tasks, and managing environment variables. The assistant can launch apps, handle files, and carry out other system-level functions thanks to this library.

- **Use Case:** Launching programs, organizing files and folders, and running commands on the system.

6. Smtplib:

- **Goal:** Emailing people
- **Details:** Emails are sent using the Simple Mail Transfer Protocol (SMTP) using the smtplib library. Email sending and email server connection are managed by it. The assistant can create and send emails automatically thanks to the library, which makes it a helpful tool for alerts and communication.
- **Use Case:** Managing email correspondence and sending emails.

7. PyWhatKit:

- **Goal:** Programmatically Sending WhatsApp Messages
- **Details:** The assistant can send WhatsApp messages programmatically thanks to the pywhatkit package. It sends messages to certain contacts on the WhatsApp online version at predetermined periods. This library gives the assistant a communication channel so it may use WhatsApp to send instant messages.
- **Use Case:** Improving communication skills, scheduling message delivery, and sending WhatsApp messages requests.

8. win10toast:

- Making Desktop Notifications is the goal.
- **Specifics:** On Windows 10, win10toast is a library for making desktop alerts. With customized texts, symbols, and durations, the assistant may now show toast notifications. This library is helpful for alerting users visually and for creating reminders.
- **Use Case:** Improving user engagement, displaying desktop reminders.

CHAPTER 4

IMPLEMENTATION WORK DETAIL

4.1 Applications in Real Life

There are many practical uses for the personal assistant created using Python and other libraries, which may greatly increase productivity and make everyday activities easier. These cross-domain apps provide customers an adaptable tool that may be used for a variety of purposes.

1. **Office Assistance:** The personal assistant may serve as a virtual secretary in a professional context, assisting office workers in better time and task management. By linking with calendar services, the assistant may arrange meetings and make sure that none are missed. It may send emails on the user's behalf, which is especially helpful for brief correspondence or in situations when the user is preoccupied. Efficient time management may be achieved by setting reminders for significant dates or appointments. Furthermore, the assistant may automate repetitive jobs like organizing files, making to-do lists, and producing reports, freeing up employees to concentrate on more important duties that demand their attention.
2. **Educational Support:** For students, the personal assistant acts as a vital instrument for academic achievement. It can look up material on Wikipedia and other websites, offering concise responses and thorough explanations on a range of subjects. Textbooks and other study materials may be read by the assistant in PDF format, which helps students retain knowledge without straining their eyes. The assistant helps students stick to their study plans and makes sure they allot enough time for each topic by creating study reminders. The helper may also improve the whole learning experience by offering direction, resolving issues, and breaking down topics in homework assignments.
3. **Home Automation**: The integration of the personal assistant with smart home devices converts it into a central center for home automation. Voice commands

enable users to operate security systems, thermostats, and lighting, resulting in a smooth and practical home environment. For instance, the assistant may lock doors at night for increased security, change the temperature settings according to user preferences, and turn off lights in empty rooms. This degree of automation improves safety and energy efficiency in addition to comfort.

4. **Entertainment:** The personal assistant also caters to entertainment demands, delivering a hands-free experience for consumers. It can react to voice requests to play certain songs, albums, or playlists. It can play music from a local library or via streaming services like Spotify. The assistant makes it simple for customers to view their preferred TV series and films by opening streaming services like Netflix and YouTube. It may also provide details on TV series, films, and celebrities, thereby serving as an informed companion to improve the user's entertainment experience.

4.2 Data Implementation and Program Execution

The personal assistant must be implemented in a few crucial phases, each of which is necessary to guarantee the efficient and seamless operation of the system.

1. **Environment Setup:** The first stage in the implementation process is setting up the development environment. This entails using pip, the Python package installer, to install the required Python libraries. The required libraries include `pyttsx3` for text-to-speech conversion, `SpeechRecognition` for capturing and processing speech input, `wikipedia` for fetching summaries from Wikipedia, `webbrowser` for opening URLs, `os` for interacting with the operating system, `smtplib` for sending emails, `PyPDF2` for reading PDF files, `pywhatkit` for sending WhatsApp messages, `requests` for making HTTP requests, and `win10toast` for creating desktop notifications. The proper installation and fulfillment of all dependencies are essential to the assistant's seamless operation.
2. **Voice Recognition:** Utilizing the `SpeechRecognition` library, the next stage entails recording and processing user instructions. To guarantee that the microphone records the user's speech correctly, configuration of the microphone is

necessary. To provide a satisfying user experience, the system must properly manage voice recognition problems, such as background noise or ambiguous speech. After that, the recorded audio is transformed into text, which is used as the input for further.

3. **Command Processing:** Once the voice is transformed into text, the system analyzes the input to identify the appropriate action. To do this, the text must be parsed, keywords must be found, and the text must meet predetermined instructions. To provide flexibility, the command processing phase needs to take several command variants into consideration. "Play music" is one command that might alternatively mean "start my music" or "put on some tunes." These fluctuations must be handled by the system, and it must react properly.
4. **Task Execution:** The system uses the appropriate library or API to carry out the required action based on the processed command. For instance, if the user wishes to send an email, the system utilizes the `smtplib` library to connect to the email server and transmit the message. If the user asks for the weather forecast, the system utilizes the `requests` library to get data from a weather API. Using the relevant libraries and APIs, other tasks, including launching programs or managing smart home appliances, are managed in a similar manner.

CHAPTER 5

COMPLETE CODE

```
import pyttsx3
import speech_recognition as sr
import datetime
import wikipedia
import webbrowser
import os
import smtplib
from email.message import EmailMessage
import pywhatkit as kit
import requests
from win10toast import ToastNotifier

# Initialize the TTS engine
engine = pyttsx3.init('sapi5')
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)

def speak(audio):
    engine.say(audio)
    engine.runAndWait()

def wishMe():
    hour = int(datetime.datetime.now().hour)
    if hour >= 0 and hour < 12:
        speak("Good Morning!")
    elif hour >= 12 and hour < 18:
        speak("Good Afternoon!")
    else:
        speak("Good Evening!")
    speak("I am Jarvis Sir. Please tell me how may I help you")
```

```
def takeCommand():
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
        r.pause_threshold = 0.5
        audio = r.listen(source)

    try:
        print("Recognizing...")
        query = r.recognize_google(audio, language='en-in')
        print(f"User said: {query}\n")
    except sr.UnknownValueError:
        print("Sorry, I did not understand that. Please say that again...")
        return "None"
    except sr.RequestError:
        print("Request failed; check your network connection.")
        return "None"
    except Exception as e:
        print(f"Error: {e}")
        return "None"
    return query.lower()

def search_wikipedia(query):
    speak('Searching on Wikipedia...')
    query = query.replace("wikipedia", "")
    results = wikipedia.summary(query, sentences=2)
    speak("According to Wikipedia")
    print(results)
    speak(results)

def open_website(url):
    webbrowser.open(url)

def play_music():
    music_dir = 'D:\\songs'
```

```
    songs = os.listdir(music_dir)
    if songs:
        os.startfile(os.path.join(music_dir, songs[0]))
    else:
        speak("No songs found in the directory.")

def sendEmail(to, content):
    try:
        msg = EmailMessage()
        msg.set_content(content)
        msg['Subject'] = 'Your Subject'
        msg['From'] = 'youremail@gmail.com'
        msg['To'] = to

        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.ehlo()
        server.starttls()
        server.login('youremail@gmail.com', 'your-password')
        server.send_message(msg)
        server.close()
        speak("Email has been sent!")
    except Exception as e:
        print(f"Failed to send email: {e}")
        speak("Sorry, I am not able to send this email")


def send_whatsapp_message(number, message):
    try:
        kit.sendwhatmsg_instantly(f"+{number}", message)
        speak("Message has been sent!")
    except Exception as e:
        print(f"Failed to send WhatsApp message: {e}")
        speak("Sorry, I am not able to send this WhatsApp message")

def open_application(app_name):
```

```
    try:
        if app_name == "command prompt":
            os.system("start cmd")
        elif app_name == "notepad":
            os.system("start notepad")
        elif app_name == "spotify":
            codePath = "C:\\Users\\Pankaj
Baraiya\\AppData\\Roaming\\Spotify\\Spotify.exe"
            os.startfile(codePath)
        else:
            speak("Application not recognized.")
    except Exception as e:
        print(f"Failed to open application: {e}")
        speak("Sorry, I am not able to open this application")


def set_reminder(title, message):
    try:
        toaster = ToastNotifier()
        toaster.show_toast(title, message, duration=10)
        speak("Reminder has been set!")
    except Exception as e:
        print(f"Failed to set reminder: {e}")
        speak("Sorry, I am not able to set the reminder")

def basic_conversation(query):
    if 'how are you' in query:
        speak("I am fine, thank you. How are you?")

    elif 'your name' in query:
        speak("I am Jarvis, your personal assistant.")

    elif 'who created you' in query:
        speak("I was created by a Pankaj under guidance of ashok sir.")
```

```
    else:
        speak("I am sorry, I don't know how to respond to that.")

if __name__ == "__main__":
    wishMe()
    while True:
        query = takeCommand()

        if 'wikipedia' in query:
            search_wikipedia(query)

        elif 'open youtube' in query:
            open_website("youtube.com")

        elif 'open google' in query:
            open_website("google.com")

        elif 'open stackoverflow' in query:
            open_website("stackoverflow.com")

        elif 'play music' in query:
            play_music()

        elif 'the time' in query:
            strTime = datetime.datetime.now().strftime("%H:%M:%S")
            speak(f"Sir, the time is {strTime}")

        elif 'open spotify' in query:
            codePath = "C:\\Users\\Pankaj
Baraiya\\AppData\\Roaming\\Spotify\\Spotify.exe"
            os.startfile(codePath)

        elif 'email to harry' in query:
            try:
                speak("What should I say?")
```

```
                content = takeCommand()
                to = "harryyourEmail@gmail.com"
                sendEmail(to, content)
            except Exception as e:
                print(f"Error: {e}")
                speak("Sorry, I am not able to send this email")

        elif 'send whatsapp message' in query:
            try:
                speak("To which number should I send the message?")
                number = takeCommand()
                speak("What is the message?")
                message = takeCommand()
                send_whatsapp_message(number, message)
            except Exception as e:
                print(f"Error: {e}")
                speak("Sorry, I am not able to send this WhatsApp message")

        elif 'open command prompt' in query:
            open_application("command prompt")

        elif 'open notepad' in query:
            open_application("notepad")


        elif 'set reminder' in query:
            speak("What is the title of the reminder?")
            title = takeCommand()
            speak("What is the message?")
            message = takeCommand()
            set_reminder(title, message)

        elif 'how are you' in query or 'your name' in query or 'who created you'
in query:
            basic_conversation(query)
```

CHAPTER 6

CONCLUSION

6.1 Restriction

Although the personal assistant exhibits a great deal of usefulness and capability, it's critical to recognize the existing constraints limiting its effectiveness and user experience:

1. **Offline Functionality:** Several functionalities rely on internet access, which is one of the main drawbacks. Online resources are necessary for tasks like retrieving weather predictions and doing Wikipedia searches. Because of this dependence, several capabilities are not available offline, which reduces the assistant's utility in situations when an internet connection is not available. Future iterations may concentrate on adding offline functionality to these features, maybe using local databases or cached data.
2. **Speech Recognition Accuracy:** A number of variables, including as the user's accent, the microphone's quality, and the existence of background noise, might influence speech recognition accuracy. Strong accented users or those working in loud situations may find it more difficult to be accurate, which might result in miscommunication and improper command processing. Although the accuracy of the voice recognition libraries available today is rather excellent, further development and optimization will be needed to get better results across a wider range of scenarios and user types. Accuracy might be improved by using more complex speech recognition models or by letting users train the system using their unique speech patterns.
3. **Dependency on APIs:** The assistant depends on external APIs for a number of its functions, including weather predictions. The availability and dependability of the assistant's functions may be limited by these APIs' use restrictions, membership requirements, or unavailability. Concerns about data security and privacy are also raised by reliance on third-party services. It might be advantageous to investigate the usage of numerous API providers for redundancy, local caching of frequently used data, or even creating internal APIs to lessen reliance on outside services in order to

6.2 The Amount of Upcoming Workring

Although the personal assistant in its present form is a useful tool with many real-world uses, there is still much room for advancement in order to improve both its functionality and user experience:

1. **Enhanced Speech Recognition:** The accuracy and responsiveness of the assistant might be greatly increased by including more sophisticated speech recognition models. This involves providing support for a greater variety of languages, accents, and dialects to increase the system's inclusivity and accessibility for users everywhere. More advanced models that are more suited to a variety of speech patterns and environmental circumstances may be produced via research in machine learning and natural language processing. Accuracy might be further improved by introducing user-specific training, in which the assistant gradually becomes used to each user's speech.
2. **Offline Capabilities:** Improving the personal assistant's usefulness in situations when internet connectivity is restricted or unavailable would require developing offline capabilities for essential features. This might include building offline-accessible local databases for meteorological data or Wikipedia summaries that are updated while the system is online. A more comprehensive offline experience might also result from the adoption of edge computing technologies, which process data locally on the user's device rather than via cloud services.
3. **Better User Interaction:** In the future, it can be the goal to make the user's interactions with the assistant more conversational and natural. Improving natural language processing skills to comprehend and reply to more intricate inquiries and follow-up questions is one way to do this. Creating a user interface that is more intuitive would accommodate a wider variety of user preferences and wants, including incorporating touch interactions and visual features.
4. **Expanded Functionality:** The personal assistant's usefulness would rise with the addition of new features and the expansion of current ones. Integration with more smart home appliances might be one of the future upgrades, offering more complete home

automation features. More advanced teaching tools, individualized learning plans, and platform integration for improved educational assistance would be beneficial as well. The assistant's function as an entertainment center would be improved by increasing the variety of entertainment alternatives, such as content recommendations based on user preferences.

5. **Security Enhancements:** Future development should concentrate on fortifying the security mechanisms put in place in the assistant, given the growing concerns about data security and privacy. This entails giving consumers more control over their data, guaranteeing compliance with data protection laws, and storing and transmitting data using cutting-edge encryption methods. User confidence and trust in the system would increase with the development of an open privacy policy and the implementation of safe authentication procedures for all interactions with outside services.

In conclusion, even if the personal assistant of today has many helpful features, recognizing its shortcomings and considering potential future improvements might result in a more sophisticated, precise, and intuitive system. In an ever-changing technology context, the assistant will continue to be relevant and beneficial to consumers via ongoing expansions and upgrades.

REFERENCES/ BIBLIOGRAPHY

1. Python Software Foundation. Python Language Reference, version 3.8. Available at http://www.python.org

2. pyttsx3 documentation. Available at https://pyttsx3.readthedocs.io/en/latest/

3. SpeechRecognition documentation. Available at https://pypi.org/project/SpeechRecognition/

4. Wikipedia-API documentation. Available at https://wikipedia.readthedocs.io/en/latest/code.html

5. webbrowser documentation. Available at https://docs.python.org/3/library/webbrowser.html

6. smtplib documentation. Available at https://docs.python.org/3/library/smtplib.html

7. PyPDF2 documentation. Available at https://pythonhosted.org/PyPDF2/

8. pywhatkit documentation. Available at https://pypi.org/project/pywhatkit/

9. requests documentation. Available at https://docs.python-requests.org/en/latest/

10. win10toast documentation. Available at https://github.com/jithurjacob/Windows-10-Toast-Notifications