A.I. Avatar with GPT Technology

Submitted in partial fulfillment of the requirements
of the degree
BACHELOR OF ENGINEERING IN **ARTIFICIAL INTELLIGENCE & DATA SCIENCE**

By

Sanika M. Sarang 211101043/56
Ayushi S. Soni 211101032/64
Shreya L. Yewale 211101029/72

Supervisor
Prof. Prof. S. P. Bansu

Department of Artificial Intelligence & Data Science
A.C.Patil College of Engineering
Kharghar, Navi Mumbai
University of Mumbai
(AY 2023-24)

CERTIFICATE

This is to certify that the Mini Project entitled **" A.I. Avatar with GPT Technology"** is a bonafidework of **Sanika M. Sarang (56) , Ayushi S. Soni (64)** and **Shreya L. Yewale (72)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **"Bachelor of Engineering"** in **"Artificial Intelligence & Data Science" .**

(Prof. S. P. Bansu)
Supervisor

(Prof. S. P. Bansu)
Head of Department

(Dr. V. N. Pawar)
Principal

Mini Project Approval

This Mini Project entitled "**A.I. Avatar with GPT Technology"** by **Sanika M. Sarang (56) , Ayushi S. Soni (64)** and **Shreya L. Yewale (72)** is approved for the degree of **Bachelor of Engineering** in **Artificial Intelligence & Data Science.**

Examiners

1……
(Internal Examiner Name & Sign)

2……
(External Examiner name & Sign)

Date:

Place:

Contents

Abstract

The "AI Avatar with GPT Technology" project represents a pioneering fusion of advanced Natural Language Processing (NLP) and state-of-the-art Generative Pre-trained Transformer (GPT) technology (Re-engineered Bard), integrated with an interactive avatar. The objective is to create an innovative conversational interface capable of understanding user queries in text formats, generating contextually relevant responses, and providing an engaging user experience.

This project addresses the gap in research by exploring the seamless integration of GPT-based models with interactive avatars, enabling a dynamic and personalized interaction. By leveraging Bard 's language understanding capabilities and the avatar's visual representation, the project aims to redefine user engagement with artificial intelligence.

Ethical considerations are a central focus, ensuring user privacy, transparency, and responsible AI practices throughout development. Additionally, a comprehensive evaluation framework is established to assess the effectiveness and user satisfaction of the avatar-driven interface.

Through rigorous development, user testing, and iterative refinement, this project sets out to push the boundaries of conversational AI, offering a versatile and inclusive platform with wide-ranging applications in customer service, education, virtual tours, and more. The project's outcomes contribute to the advancement of human-computer interaction, paving the way for future innovations in the field.

ABBREVIATION TABLE:

Abbreviation	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
GenAI	Generative AI
re	Regular Expressions
pyttsx3	Text-to-Speech Synthesis Library
speech_recognition	Voice Input Handling Library
system	System-related Functions (from the os module)
Bard	Chatbot Class (Non-standard Library)
OS	Operating System
CPU	Central Processing Unit
RAM	Random Access Memory
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
IDE	Integrated Development Environment
SVG	Scalable Vector Graphics
NLP	Natural Language Processing
GPT	Generative Pre-trained Transformer
UI/UX	User Interface/User Experience
MDN	Mozilla Developer Network

Acknowledgments

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of the "AI Avatar with GPT Technology" project.

First and foremost, I extend my deepest thanks to my mentor and guide for their invaluable support, guidance, and mentorship throughout this project. Their expertise and insights have been instrumental in shaping the direction and execution of this endeavor.

I am also immensely grateful to the entire team involved in the development of this project. Each member's dedication, expertise, and collaborative spirit played a pivotal role in achieving our goals.

I would like to extend my thanks to the academic institutions, research organizations, and open-source communities whose resources and knowledge-sharing platforms proved invaluable in the course of this project.

Furthermore, I would like to express my appreciation to my peers, colleagues, and friends who provided encouragement, feedback, and constructive criticism, all of which greatly contributed to the project's success.

Lastly, I would like to acknowledge the support of my family for their unwavering encouragement and understanding throughout this journey.

This project would not have been possible without the collective effort and support of all those mentioned above. Thank you for being a part of this endeavor and for helping bring the "AI Avatar with GPT Technology" project to fruition.

1. Introduction

1.1 Introduction:

In an era defined by rapid technological advancements and an ever-growing reliance on digital interactions, the development and deployment of intelligent conversational agents have taken center stage. These agents, often referred to as chatbots, have revolutionized the way we engage with technology, enabling natural language interactions that span from answering customer inquiries to assisting users in their daily tasks. This report delves into the creation and enhancement of a chatbot system, with a particular focus on incorporating voice recognition and avatar features. AI avatar chat is a conversational agent that uses artificial intelligence to simulate human-like interactions. It is a type of chatbot that can engage in conversations with users through text, voice, or video. AI avatar chat can be used for a variety of purposes, such as customer service, education, and entertainment.

1.2 Motivation:

The motivation behind this project stems from the increasing demand for more intuitive and engaging human-computer interactions. Traditional text-based chatbots have already proven their worth in various applications, but the integration of voice recognition and avatars can elevate user experiences to unprecedented levels. Voice recognition enables users to interact with the system more naturally, while avatars add a visual and emotional dimension to the interaction, making it more relatable and engaging. These enhancements have the potential to make technology more accessible, particularly for those who may face barriers with text-based interfaces.

1.3 Problem Statement & Objectives:

The core problem we aim to address is the development of a versatile and user-friendly chatbot that seamlessly integrates voice recognition and avatars. This involves overcoming multiple challenges, including:

- Natural Language Processing to understand user input effectively.
- Natural and expressive text-to-speech capabilities to provide clear and engaging voice responses.
- The creation and integration of a dynamic avatar that can visually represent the agent's persona and emotions.

Our objectives are to:

1. Enhance User Interaction: Create a more engaging and interactive user experience through the use of an AI-powered avatar.
2. Education and Information Sharing: Use the avatar to provide informative responses, assist with learning, or offer guidance on specific topics.
3. Error Handling and Fallback Mechanisms: Implement strategies to gracefully handle situations where the avatar may not understand the user's input.
4. Ensure the system is user-friendly, secure, and scalable.

1.4 Organization of the Report:

- There would be a literature survey first, which deals with the information regarding the existing system, its drawbacks, and solutions provided by this project.
- The next is the proposed system of this project which has its introduction, the frameworks used and its architecture, its algorithm along with the snippets of its code and execution.
- Details about the hardware and software used to build it and its analysis.
- Followed by the code and the Results when different inputs where given.
- Lastly the future work of the project and it’s conclusion.

2. Literature Survey

2.1 Survey of Existing System:

The landscape of virtual assistants and conversational AI systems has seen significant advancements in recent years. Below is a survey of some existing systems and their notable features:

Google Assistant: Google Assistant is a widely used virtual assistant that leverages Google's powerful search capabilities and natural language processing. It can perform tasks like setting reminders, sending messages, providing directions, and more.

Amazon Alexa: Alexa is the virtual assistant powering Amazon's Echo devices. It is capable of performing a wide range of tasks, including controlling smart home devices, providing weather updates, and even playing games with users.

Apple Siri: Siri is Apple's virtual assistant available on iOS devices. It can perform tasks like sending messages, setting reminders, making calls, and providing information on a wide range of topics.

OpenAI GPT-3: GPT-3 is a powerful language generation model developed by OpenAI. It can generate human-like text based on a given prompt and has been used in various applications, including chatbots, content generation, and more.

Analysis:

These existing systems showcase the wide-ranging capabilities of virtual assistants and conversational AI. They are proficient in tasks ranging from general information retrieval to device control and personalized recommendations. However, there is still room for improvement, especially in creating more contextually aware and visually engaging interactions through the integration of avatars, which is the focus of the "AI Avatar with GPT Technology" project.

2.2 Limitation Existing system or research gap:

- Limited Visual Interaction:

  Existing virtual assistants primarily rely on text-based interactions, limiting the potential for visually engaging and dynamic communication. There is a notable gap in systems that integrate avatars for a more immersive user experience.

- Contextual Understanding Challenges:

  While current virtual assistants have made significant strides in natural language understanding, they may struggle with complex contexts or maintaining conversational coherence over extended interactions. This represents a research gap in achieving more robust contextual understanding.

- Evaluation Frameworks for Avatar Integration:

  There is a research gap in the development of comprehensive evaluation frameworks that assess the effectiveness, user satisfaction, and ethical implications of integrating avatars with conversational AI.

- Real-time Visual Synchronization:

  Achieving real-time lip sync and facial expressions in avatars remains a challenge. This limitation calls for research in real-time visual synchronization techniques for a more realistic avatar interaction.

- Offline Functionality:

  Most existing virtual assistants heavily rely on an internet connection. There is a gap in research regarding the development of virtual assistants that can function effectively in offline scenarios.

2.3 Mini Project Contribution:

	Sanika Sarang	Ayushi Soni	Shreya Yewale
Backend	✓		✓
Frontend		✓	✓
Avatar Animation and triggering logic	✓	✓	
Background Research for Project	✓	✓	✓

3. Proposed System

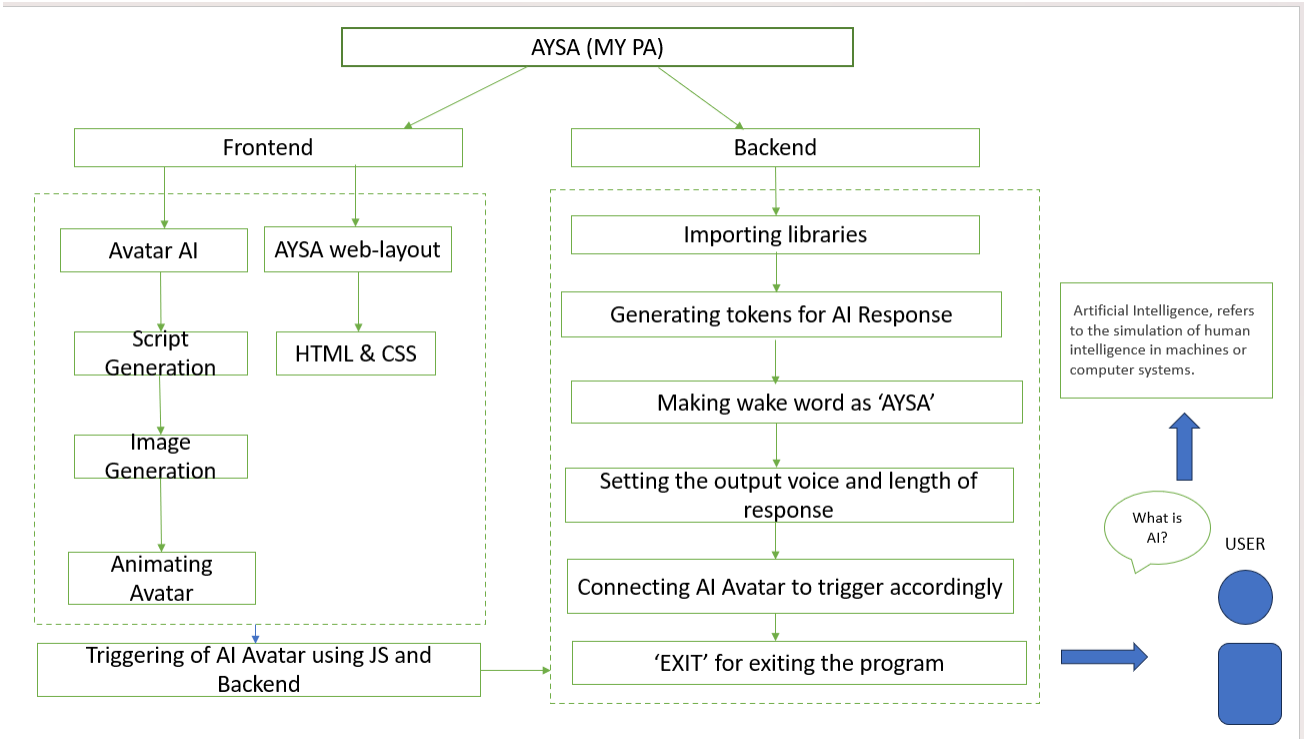
3.1 Introduction:

Our algorithm, the AI Response Generator, serves as an advanced conversational AI system capable of understanding and responding to user queries. By employing state-of-the-art natural language processing techniques and machine learning, it addresses the need for more natural and intelligent interactions between humans and machines. With applications ranging from virtual assistants to customer support chatbots, our algorithm's innovation lies in its ability to comprehend context, generate contextually relevant responses, and adapt to a variety of conversational scenarios, significantly improving the quality of human-computer interactions.

The AI Response Generator represents a sophisticated conversational AI system designed to effectively interpret and address user queries. Leveraging cutting-edge natural language processing techniques and machine learning, it caters to the growing demand for natural and intelligent interactions between individuals and automated systems. Its utility spans across various applications, serving as a key component in the development of virtual assistants and customer support chatbots.

Distinguished by its capacity to grasp contextual nuances, the algorithm excels in generating responses that align with the conversation's context, thereby enhancing the depth and quality of human-computer interactions. Its adaptive nature enables it to navigate through diverse conversational scenarios, demonstrating its versatility and effectiveness in catering to a wide range of user needs and inquiries.

3.2 Architecture/ Framework:



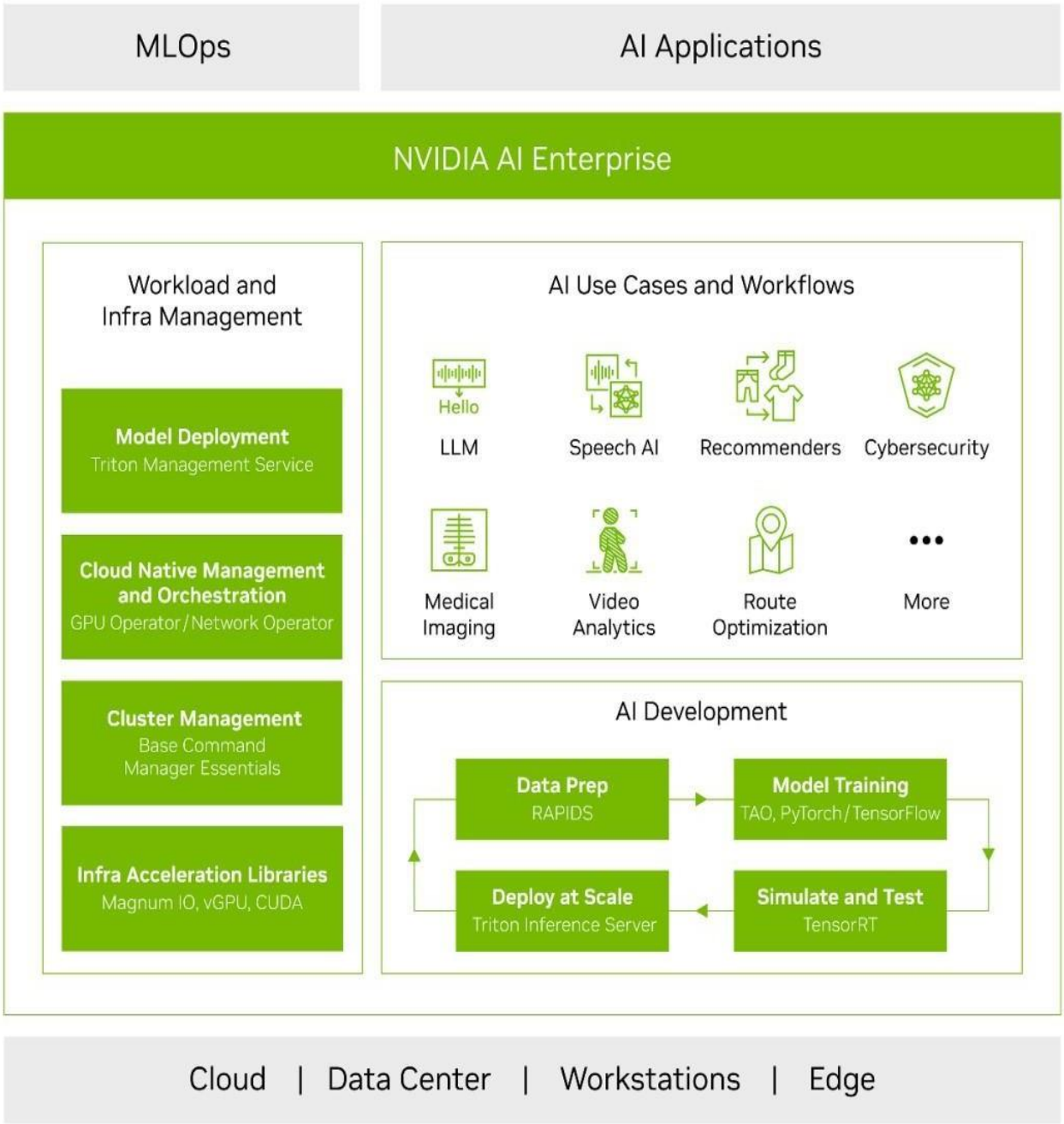
1. **Code for AI Response Generator:**

Framework:

- The code for the AI Response Generator primarily relies on Python and utilizes several external libraries and tools. While it doesn't explicitly use a specific software framework, it does make use of various libraries to achieve its functionality.

Architecture:

- It follows a client-server architecture where the client is the user interacting with the chatbot, and the server component includes the Chatbot class from the Bard library. The chatbot interacts with an API using authentication tokens to generate responses to user queries.
- The code demonstrates a state machine-like architecture where the chatbot transitions between voice-activated and non-voice-activated states based on user input. It also maintains conversation history to provide context for responses.
- Generative AI is a form of AI that can create new and original content, such as text, images, audio, video, music, art, and code. GenAI is based on various technical foundations, such as: model architecture, self-supervised pre-training, and generative modeling methods.
- Model architecture: The structure and design of the neural networks that generate the content. Examples of model architectures include transformers, convolutional neural networks, recurrent neural networks, and attention mechanisms.
- Self-supervised pretraining: The process of training a model on a large amount of unlabeled data is to learn general features and patterns that can be transferred to specific tasks. Examples of self-supervised pretraining methods include masked language modeling, contrastive learning, and denoising autoencoders.
- Generative modeling methods: The techniques and algorithms that enable a model to learn the probability distribution of the data and sample new data from it.


MLOps
AI Applications
NVIDIA AI Enterprise
Workload and Infra Management
Model Deployment
Triton Management Service
Cloud Native Management and Orchestration
GPU Operator / Network Operator
Cluster Management
Base Command Manager Essentials
Infra Acceleration Libraries
Magnum IO, vGPU, CUDA
AI Use Cases and Workflows
Hello
LLM
Speech AI
Recommenders
Cybersecurity
Medical Imaging
Video Analytics
Route Optimization
More
AI Development
Data Prep
RAPIDS
Model Training
TAO, PyTorch / TensorFlow
Deploy at Scale
Triton Inference Server
Simulate and Test
TensorRT
Cloud | Data Center | Workstations | Edge

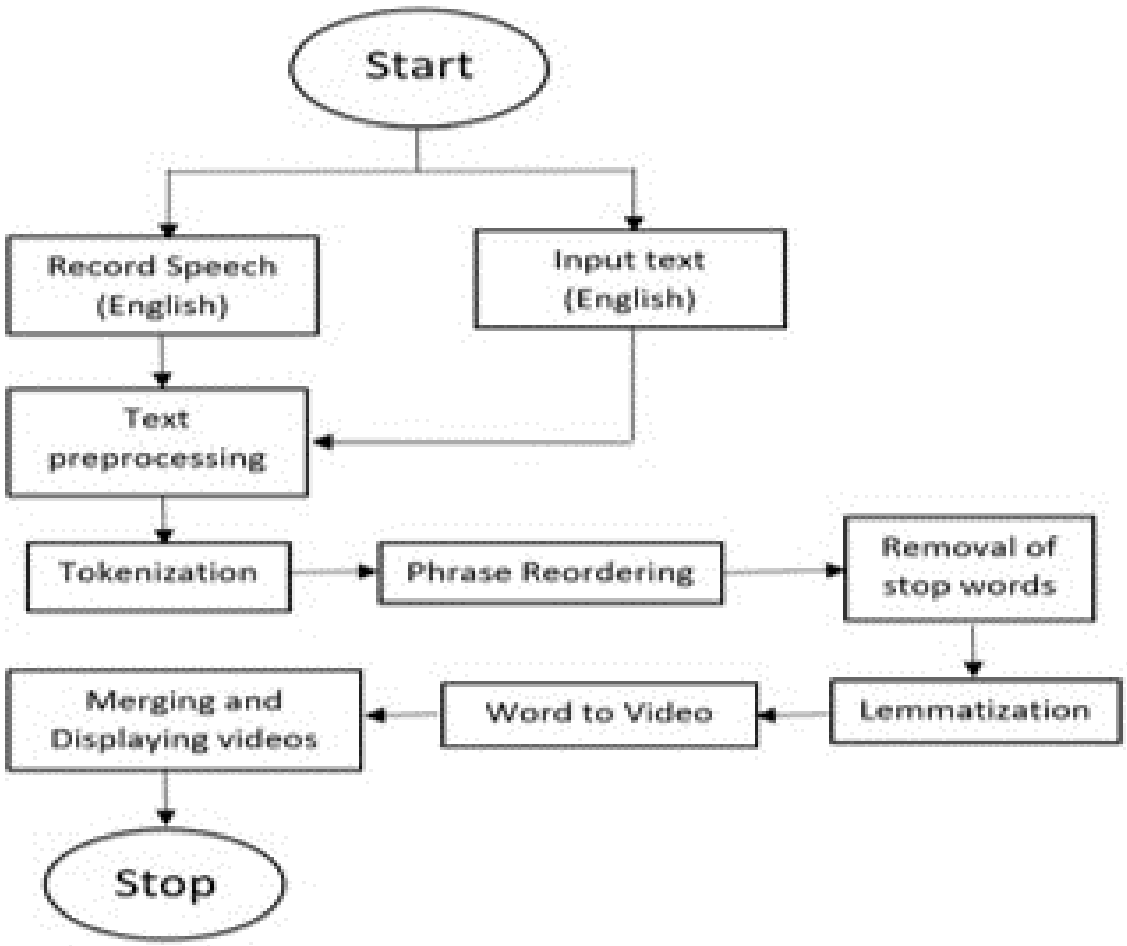
2. Code for Frontend and AI Avatar:

Framework:

- The code for the frontend and AI Avatar is primarily built using HTML and CSS for the user interface. While it doesn't explicitly use a specific front-end web framework like React or Angular, it leverages HTML and CSS for layout and styling, which can be considered the framework for the user interface.

Architecture:

- The architecture of the frontend and AI Avatar code can be described as a web-based user interface. It follows a client-server architecture where the client is the user's web browser, and the server (not shown in the provided code) would handle back-end logic, such as communication with the AI Response Generator.
- The user interface is structured using HTML and styled using CSS. It's designed to provide a responsive and interactive user experience. The architecture separates the user interface (HTML and CSS) from the back-end logic (not shown in the provided code), which communicates with the AI Response Generator to send and receive messages.



3.3 Algorithm and Process Design:

3.3.1 : Algorithm of AI Response Generator:

The chatbot can be activated by saying a wake word ("aysa" in this case), and it responds to user queries and commands. Here's an algorithmic overview of the code:

1. Import necessary libraries and modules:
   - re for regular expressions
   - pyttsx3 for text-to-speech synthesis
   - speech_recognition for handling voice input
   - system from the os module for system-related functionality
   - Bard for the Chatbot class (not a standard library)
   - playsound for playing audio files (not used in the provided code)

2. Define the API tokens and initialize the chatbot:
   - Initialize the Chatbot class with tokens for authentication.

3. Set some constants and variables:
   - MAX_OUTPUT_LENGTH: Maximum number of characters for the chatbot's response.
   - LANGUAGE_CODE: Language code for text-to-speech (English in this case).
   - wake_word: The word that activates the chatbot ("aysa" in this case).
   - Initialize r as a SpeechRecognizer for voice input.
   - Initialize voice_activated as False.
   - Initialize conversation_history as an empty string.

4. Define helper functions:
   - is_wake_word(text): Checks if the wake word is present in the input text.
   - clean_text(text): Removes any asterisks from the text.
   - limit_output(text, max_length): Limits the text to a maximum length.
   - speak(text, language): Uses text-to-speech to speak the provided text.

5. Define the prompt_bard(prompt, conversation_history) function:
   - Combines the current conversation history with the user's prompt.
   - Calls the chatbot to generate a response based on the combined conversation.
   - Returns the response content.

6. Create the main function:
   - Use a while loop to continuously interact with the chatbot.
   - If not voice-activated, listen for the wake word ("aysa").
   - Once the wake word is detected, set voice_activated to True and prompt the user to ask questions.
   - Continue listening for user input.
   - If the user says "exit" or "bye," the program exits.
   - If the user says "continue" and there is a conversation history, continue the conversation.
   - Otherwise, send the user's query to the chatbot, update the conversation history, and speak the chatbot's response.

7. Execute the main function when the script is run.

3.3.2 Algorithm for AI frontend and AI Avatar Generator:

1. **HTML Structure**:
    - The page has an HTML5 document structure with a **<head>** section for metadata and a **<body>** section for content.
    - It includes a link to an external stylesheet (**main.css**) and a favicon link.
    - The content of the page is divided into a flex container (** **) with two main sections: **.left** and **.right**.
    - Inside the **.left** section, there's a button for creating a new chat, a container for an avatar or video, and other elements.
    - The **.right** section seems to display information about "AYSA (MY PA)."
2. **CSS Styling**:
    - The CSS styles define the layout, colors, and positioning of various elements.
    - Flexbox (**display: flex**) is used to arrange the **.left** and **.right** sections side by side with equal height.
    - The background color, text color, padding, and other styles are defined for various elements.
    - The **.container** element has a maximum width, a background color, rounded corners, and a box shadow to give it a card-like appearance.
    - There are CSS styles for avatar positioning within the container.
    - Styling for a "New Chat" button is also provided.
3. **Interactive Elements**:
    - The HTML includes input elements for sending messages.
    - A "Send" button is provided with an associated SVG icon.
    - There's a ** ** element for displaying chat messages. However, the content for this element and its behavior appear to be controlled by JavaScript in an external script file (**script.js**).
4. **JavaScript**:
    - The behavior and functionality of this chat interface may be implemented in the **script.js** file. The script controls the visibility and content of the chat log and manages user interactions, such as sending messages.

3.4 Details of Hardware & Software:

1. Hardware Requirements:
    - Operating System: Specify the operating system (e.g., Windows 10, macOS, Linux) required to run the code or application.
    - Processor: Mention the processor type and speed (e.g., Intel Core i7, 2.5 GHz).
    - RAM (Memory): Indicate the minimum and recommended RAM requirements (e.g., 8 GB minimum, 16 GB recommended).
    - Storage: Mention the minimum available storage space required (e.g., 100 GB of free disk space).
    - Graphics: If applicable, describe the graphics card or GPU requirements (e.g., NVIDIA GeForce GTX 1060 or better).
    - Audio: Specify if any specific audio hardware is needed (e.g., microphone or speakers).
2. Software Requirements:
    - Programming Language: List the programming languages used in the code or application (e.g., Python, JavaScript).
    - External Libraries and Modules: Identify any external libraries, frameworks, or modules that need to be installed (e.g., TensorFlow, NumPy).
    - APIs: If the code interacts with external APIs, mention the APIs and any required API keys.
    - Operating System Compatibility: Specify if the code is platform-specific or compatible with multiple operating systems.
    - Web Browsers: If the project is a web application, indicate which web browsers are supported (e.g., Chrome, Firefox, Safari).
    - Development Tools: Mention any integrated development environments (IDEs) or code editors recommended for working with the code.
    - Database: If the application uses a database, specify the database system (e.g., MySQL, PostgreSQL).
3. Dependencies and Installation Instructions:
    - Provide instructions for installing or setting up any required software, libraries, or dependencies.
    - Mention any specific versions or configurations that are known to be compatible with the code or application.
4. Performance Considerations:
    - If there are specific performance requirements (e.g., real-time processing, high CPU usage), outline them.
    - Mention any hardware optimizations or considerations that may improve performance.
5. Security and Permissions:
    - Specify any security settings or permissions required to run the code (e.g., access to the internet, file system permissions).
6. Miscellaneous Requirements:
    - Include any other hardware or software considerations that are critical for the successful execution of the project or code.

3.5 Code:

3.5.1 Code for AI response generater:

```
import pyttsx3
from Bard import Chatbot
from flask import Flask, request, render_template, jsonify

app = Flask(__name__, static_url_path='/static')
app.debug = True

token = 'yourtoken'
ts_token = 'yourtoken'

chatbot = Chatbot(token, ts_token)

MAX_OUTPUT_LENGTH = 250
LANGUAGE_CODE = 'en'

def clean_text(text):
    cleaned_text = text.replace('*', '')
    return cleaned_text

def limit_output(text, max_length=MAX_OUTPUT_LENGTH):
    if len(text) > max_length:
        return text[:max_length]
    return text

def speak(text, language=LANGUAGE_CODE):
    engine = pyttsx3.init()
    engine.setProperty('rate', 220)
    engine.setProperty('voice',
f'HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Speech_OneCore/Voices/Tokens/{languag
e}')
    cleaned_text = clean_text(text)
    limited_text = limit_output(cleaned_text)
    engine.say(limited_text)
    engine.runAndWait()

def prompt_bard(prompt, conversation_history):
    full_prompt = f"{conversation_history}\n{prompt}"
    response = chatbot.ask(full_prompt)
    return response['content']

@app.route('/', methods=['GET'])
def chat():
    return render_template('chat.html', conversation="", user_input="", response="")

@app.route('/api/chat', methods=['POST'])
def chat_api():
    user_input = request.json.get('user_input', '')
    if len(user_input.strip()) == 0:
        return jsonify({"response": "Empty prompt. Please enter again."})
```

```
    conversation_history = request.json.get('conversation_history', '')

    if "continue" in user_input.lower() and conversation_history:
        response = prompt_bard(user_input, conversation_history)
    else:
        response = prompt_bard(user_input, conversation_history)
        conversation_history += f"\n{user_input}\n{response}"

    speak(response)
    return jsonify({"response": response, "conversation_history": conversation_history})

if __name__ == '__main__':
    conversation_history = ""
    app.run()
```

3.5.2 : Code for frontend and AI Avatar:

```
<!DOCTYPE html>
<html>
<head>
  <title>Aysa.ai</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <style>

    .flex {
      display: flex;
      height: 100vh;
    }

    .left {
      flex: 20%;
      background-color: #262424;
      color: rgb(212, 54, 237);
      padding: 20px;
    }

    .right {
      flex: 80%;
      background-color: #5f5f7c;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      font-size: 1.2rem;
      font-weight: bold;
      text-align: center;
      overflow: hidden;
    }

    /* ... (your existing styles) ... */

    #user_input_container {
      width: 97%;
      background: rgb(184, 184, 246);
      border-radius: 7px;
      display: flex;
      align-items: center;
    }

    #user_input {
      flex: 1;
      border: none;
      background: transparent;
      padding-left: 30px;
    }

    #send {
      background: rgb(100, 106, 125);
      color: rgb(250, 252, 255);
```

```
            border: none;
            padding: 12px;
            margin-left: 10px;
            cursor: pointer;
        }
        .left, .right {

            color: rgb(45, 43, 45);
        }


       .user-message {
    font-size: 14px; /* Adjust the font size as needed */
    color: black;
    text-align: left;
    margin: 5px 0; /* Add spacing between user messages */
}

#loop-video {
        width: 280px;
        height: 560px;
         /* Add a border to the video container */

}
        #voiceButton {
            width: px;
            height: 24px;
            stroke: currentColor;
            stroke-width: 2px;
            stroke-linecap: "round";
            stroke-linejoin: "round";
        }

        #user_input_display {
            width: none;
            color: rgb(55, 51, 51);
            text-align: center;
            text-decoration: rgb(23, 23, 23);
        }
        .aysa-title {
            font-size: 2rem;
            font-weight: bold;
            color: lavender;

            text-align: center;
        }


        .newchat {
            background-color: rgb(119, 223, 249);
            width: 90%;
            border-color: white;
            display: flex;
            border: 1px solid white;
```

```
            padding-top: 0.5rem;

            padding-bottom: 0.5rem;

            margin-top: 0.5rem;
            margin-bottom: 0.5rem;
            padding-left: 2.5rem;
            padding-right: 2.5rem;
            gap: 0.5rem;
            background-color: transparent;
            color: rgb(91, 240, 251);
            align-items: center;
            cursor: pointer;
            font-size: 14px;
            border-radius: 0.37rem;
            opacity: 0.7;
        }

        .bg-chatblack-50 {
            background-color: #4f4f60;
        }

    </style>
</head>

<body style="overflow: hidden;">
    <div class="flex">
        <div class="left">
            AYSA
            <br>
            <button class="newchat">
                <svg stroke="currentColor" fill="none" stroke-width="2" viewBox="0 0
      24 24" stroke-linecap="round"
                    stroke-linejoin="round" class="h-4 w-4" height="1em" width="1em"
      xmlns="http://www.w3.org/2000/svg">
                    <line x1="12" y1="5" x2="12" y2="19"></line>
                    <line x1="5" y1="12" x2="19" y2="12"></line>
                </svg>
                New Chat
            </button>
            <div class="avatar-container"  >
                <video id="loop-video">
                <source src="{{ url_for('static', filename='vn.mp4') }}"
      type="video/mp4"  >
            </video>
            </div>
            <div class="container">

            </div>
        </div>
        <div class="right">
            <div class="aysa-title">AYSA</div>
            <div id="chat-container">

            </div>
```

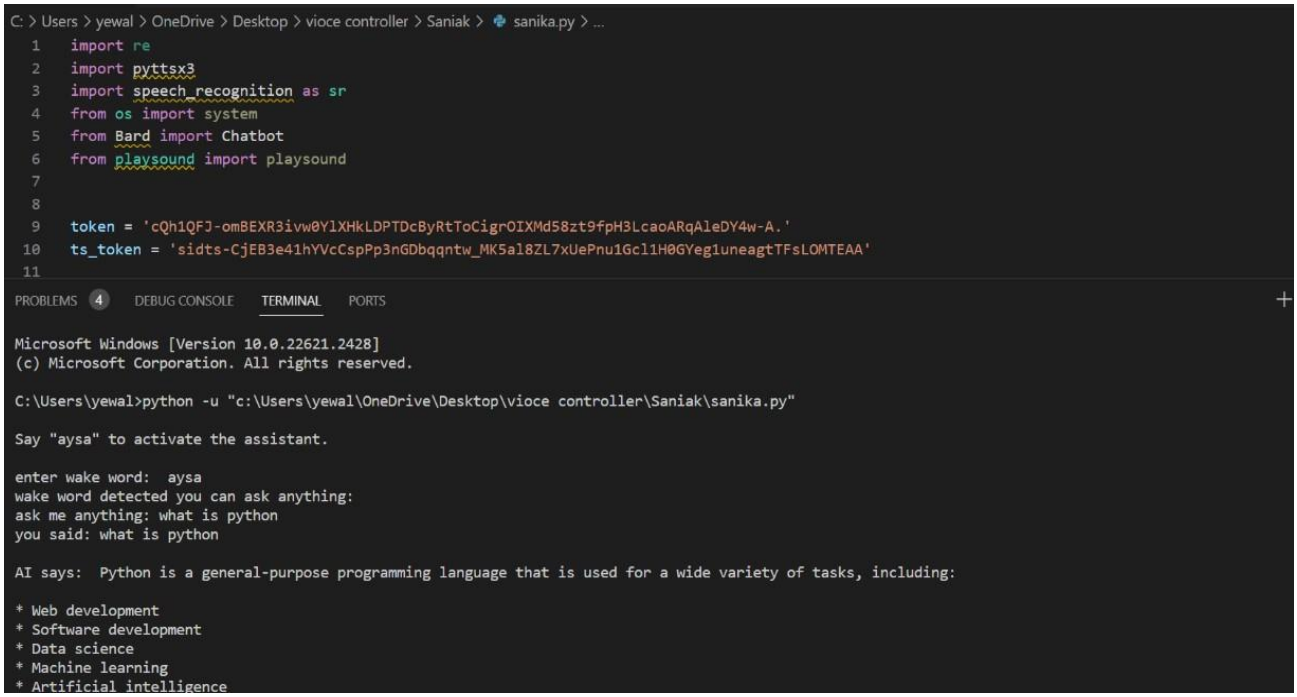
```
        <div id="user_input_container" style="position: fixed; bottom: 0; right: 0;

width: 77%; background: rgb(184, 184, 246); border-radius: 7px; display: flex;
    align-items: center;">
            <input id="user_input" class="input_text" placeholder="ask me"

name="text">
            <button id="send" class="relative -left-35 pl-10" > <!-- Remove the
    clearInput() function from the button -->
                <svg stroke="currentColor" fill="none" stroke-width="3" viewBox="0
    0 24 24" stroke-linecap="round" stroke-linejoin="round" class="h-4 w-4 mr-1"
    height="1em" width="1em" xmlns="http://www.w3.org/2000/svg">
                    <line x1="22" y1="2" x2="11" y2="13"></line>
                    <polygon points="22 2 15 22 11 13 2 9 22 2"></polygon>
                </svg>
            </button>
        </div>
    </div>
</div>

<script>

function speakText(response) {
  // Check if the SpeechSynthesis API is available in the browser
  if ('speechSynthesis' in window) {
    // Create a new SpeechSynthesisUtterance object
    var message = new SpeechSynthesisUtterance();

    // Set the text to be spoken
    message.text = response;
    message.voice = speechSynthesis.getVoices()[0];
    // Use the default voice or specify a voice
    // message.voice = speechSynthesis.getVoices()[0]; // Use the default voice


    message.rate = 1.0;
    message.pitch = 1.0;
    message.volume = 1.0;

    // Speak the text
    window.speechSynthesis.speak(message);
  } else {
    console.log('SpeechSynthesis API is not supported in this browser.');
  }
}
    $(document).ready(function () {
        $("#send").click(function () {
            var user_input = $("#user_input").val();
            var conversation_history = $("#conversation").text();

            // Append user input to the chat-container
            $("#chat-container").append("<div>User: " + user_input + "</div>");

            $.ajax({
                type: 'POST',
```

```
                url: '/api/chat', // Replace with your server endpoint
                data: JSON.stringify({ "user_input": user_input,

"conversation_history": conversation_history }),

                contentType: 'application/json',
                dataType: 'json',
                success: function (data) {
                    // Append chatbot response text to the chat-container
                    $("#chat-container").append("<div>AYSA: " + data.response +
    "</div");

                    // Update conversation history
                    $("#conversation").text(data.conversation_history);

                    // Clear user input field
                    $("#user_input").val("");

                    // Scroll to the bottom of the chat-container
                    var chatContainer = document.getElementById("chat-container");
                    chatContainer.scrollTop = chatContainer.scrollHeight;

                    // Show the hidden video element
                    var chatbotVideo = document.getElementById("loop-video");
                    chatbotVideo.style.display = "block";

                    // Play the video and stop it after 14 seconds

                    chatbotVideo.play();
                    setTimeout(function() {
                        chatbotVideo.pause();
                    }, 14000); // 14 seconds in milliseconds

                    // Hide the fallback image
                    var fallbackImage = document.getElementById("fallback-image");
                    fallbackImage.style.display = "none";

                },
                error: function () {
                    // If there's an error, hide the video and show the fallback image
                    var chatbotVideo = document.getElementById("loop-video");
                    chatbotVideo.style.display = "none";

                    var fallbackImage = document.getElementById("fallback-image");
                    fallbackImage.style.display = "block";
                }
            });
        });
    });
  </script>

</body>
    </html>
```

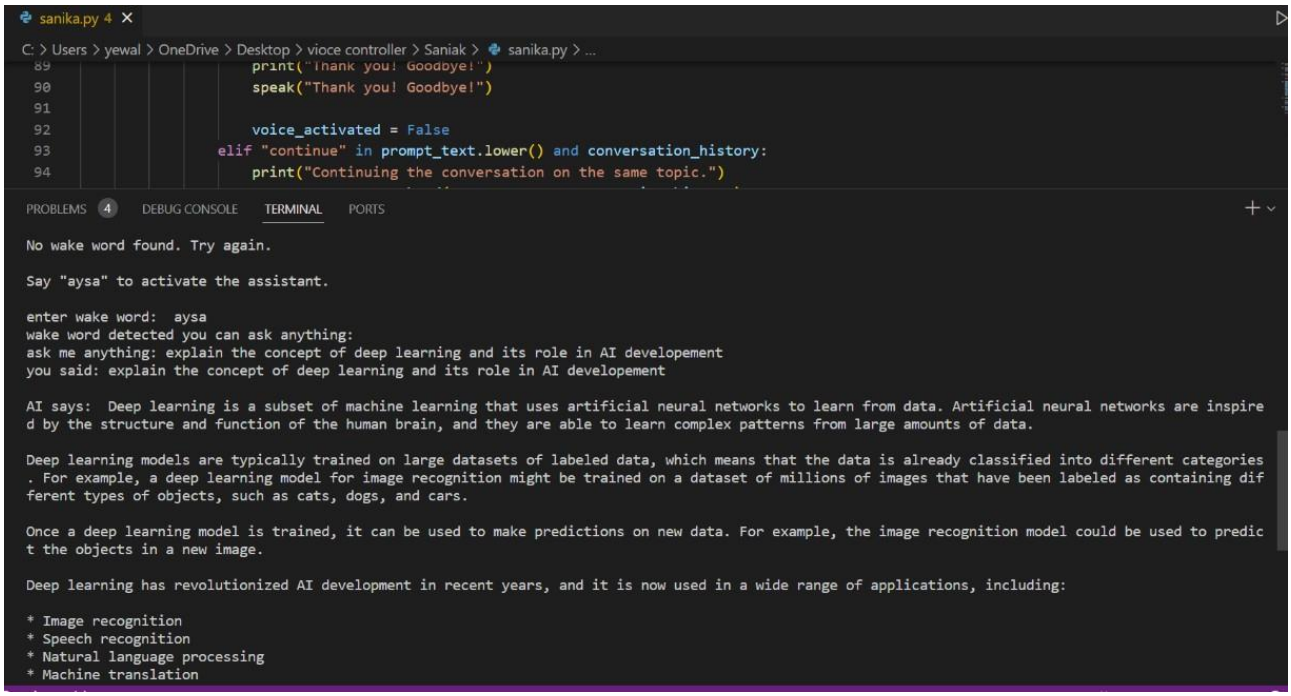
3.6 Experiment and Results:

3.6.1: AI response generator:

Outputs generated according to inputs



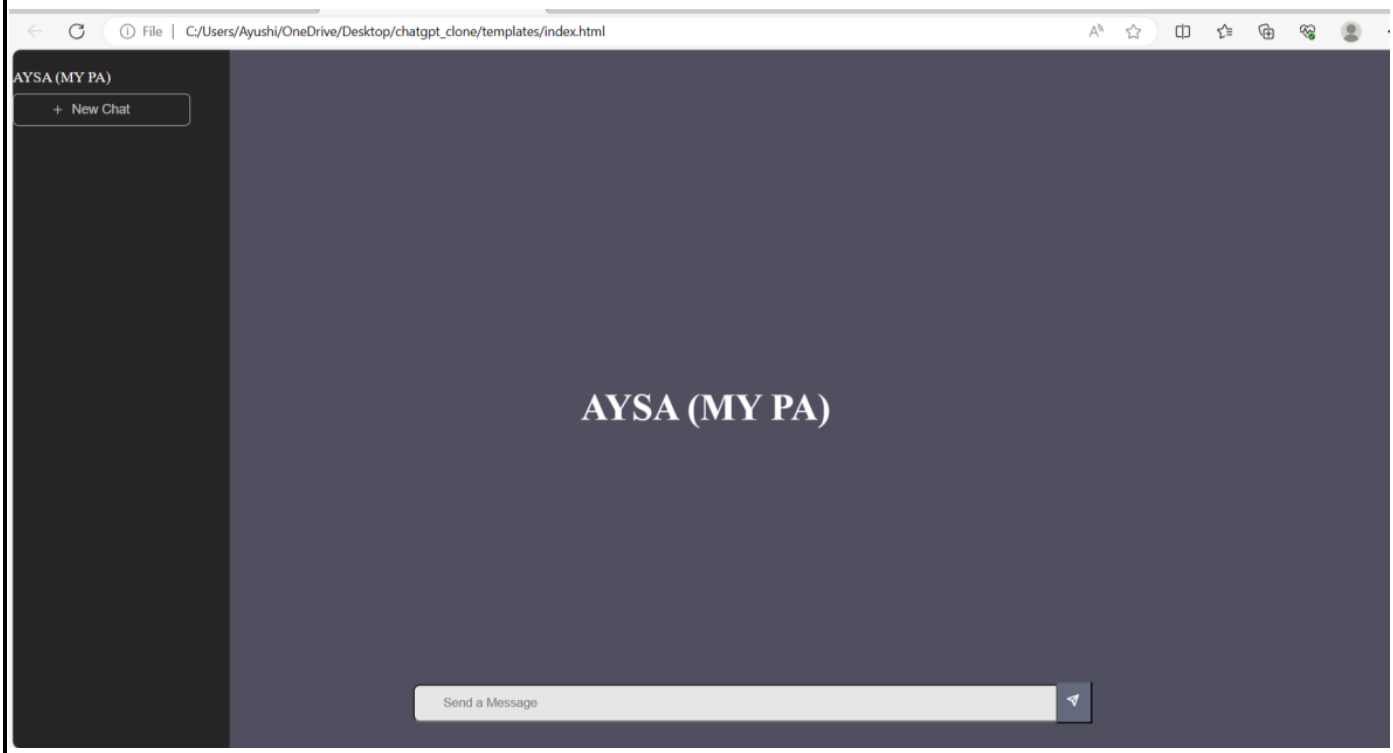
```
C: > Users > yewal > OneDrive > Desktop > vioce controller > Saniak > sanika.py > ...
  1   import re
  2   import pyttsx3
  3   import speech_recognition as sr
  4   from os import system
  5   from Bard import Chatbot
  6   from playsound import playsound
  7
  8
  9   token = 'cQh1QFJ-omBEXR3ivw0YlXHkLDPTDcByRtToCigrOIXMd58zt9fpH3LcaoARqAleDY4w-A.'
 10   ts_token = 'sidts-CjEB3e41hYVcCspPp3nGDbqqntw_MK5al8ZL7xUePnu1Gcl1H0GYeg1uneagtTFsLOMTEAA'
 11

PROBLEMS 4   DEBUG CONSOLE   TERMINAL   PORTS

Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\yewal>python -u "c:\Users\yewal\OneDrive\Desktop\vioce controller\Saniak\sanika.py"

Say "aysa" to activate the assistant.

enter wake word:  aysa
wake word detected you can ask anything:
ask me anything: what is python
you said: what is python

AI says:  Python is a general-purpose programming language that is used for a wide variety of tasks, including:

* Web development
* Software development
* Data science
* Machine learning
* Artificial intelligence
```

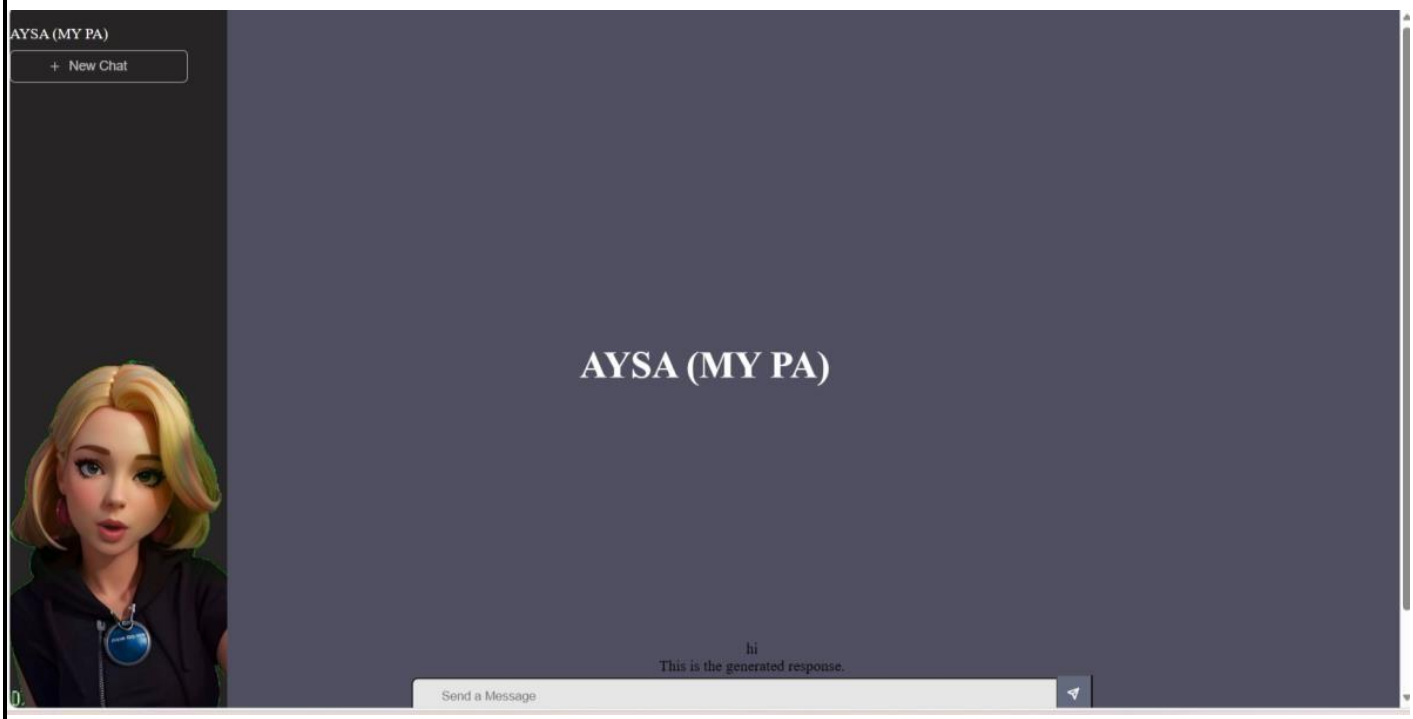


```
sanika.py 4
C: > Users > yewal > OneDrive > Desktop > vioce controller > Saniak > sanika.py > ...
 90                  speak("Thank you! Goodbye!")
 91
 92                  voice_activated = False
 93              elif "continue" in prompt_text.lower() and conversation_history:
 94                  print("Continuing the conversation on the same topic.")

PROBLEMS 4   DEBUG CONSOLE   TERMINAL   PORTS

No wake word found. Try again.

Say "aysa" to activate the assistant.

enter wake word:  aysa
wake word detected you can ask anything:
ask me anything: explain the concept of deep learning and its role in AI developement
you said: explain the concept of deep learning and its role in AI developement

AI says:  Deep learning is a subset of machine learning that uses artificial neural networks to learn from data. Artificial neural networks are inspire
d by the structure and function of the human brain, and they are able to learn complex patterns from large amounts of data.

Deep learning models are typically trained on large datasets of labeled data, which means that the data is already classified into different categories
. For example, a deep learning model for image recognition might be trained on a dataset of millions of images that have been labeled as containing dif
ferent types of objects, such as cats, dogs, and cars.

Once a deep learning model is trained, it can be used to make predictions on new data. For example, the image recognition model could be used to predic
t the objects in a new image.

Deep learning has revolutionized AI development in recent years, and it is now used in a wide range of applications, including:

* Image recognition
* Speech recognition
* Natural language processing
* Machine translation
```

3.6.2 : Frontend and AI Avatar:

AI Avatar triggering once the input is given and the output is generated





3.8 Conclusion and Future work:

Future Work:

Integration with OpenAI API: To enhance the AI Response Generator's capabilities, we can explore the integration of the OpenAI API. While this may entail additional costs, it will provide access to state-of-the-art language models and the potential for more context-aware and human-like responses. Voice Recognition for Input: To offer a more seamless and user-friendly experience, implementing voice recognition for input is imperative. This will enable users to interact with the system using their voice, making it more accessible and convenient.

Voice Output Enhancement: Building on the current text-to-speech output, we can further improve the quality and naturalness of the voice output. This might involve exploring advanced speech synthesis technologies and voice modulation to make the responses sound more human-like.

Conversation History and Context Management: The system can be enhanced to maintain and manage conversation history and context more effectively. This will allow for more coherent and context-aware responses, making conversations feel more natural and engaging.

Scalability and Cost Optimization: As the system evolves, it's important to focus on scalability and cost optimization. This includes efficient resource utilization and monitoring of API usage to manage operational expenses effectively. Security and Privacy Considerations: As the system handles voice input, it's essential to implement robust security and privacy measures to protect user data and maintain confidentiality.

User Interface Enhancements: In the Frontend and AI Avatar, consider further improvements in user interface design, responsiveness, and accessibility features to ensure a seamless user experience.

Usability Testing: Conduct usability testing to gather user feedback and iteratively improve the system based on real user experiences and suggestions.

Avatar’s lip synchronization:In our quest to advance the AI Response Generator, synchronization of the AI Avatar's lip movements with the voice output is a crucial feature for enhancing the user experience. This synchronization between the voice output and the avatar's lip movements is pivotal for creating a more realistic and engaging interaction. By ensuring that the AI Avatar's lip movements align with the spoken responses, we can significantly elevate the sense of naturalness and realism in the conversation.

Conclusion: In the current implementation, we have developed an AI Response Generator that leverages the capabilities of the Bard API, enabling text-based interactions with users. This system has demonstrated its effectiveness in understanding and responding to user queries, contributing to more natural and intelligent interactions between humans and machines. However, it's important to note that the reliance on the Bard API incurs certain operational costs, which need to be considered. Furthermore, the Frontend and AI Avatar have provided a user-friendly interface for engaging with the AI Response Generator, enhancing the user experience. However, the user input has been limited to textual queries, and the output has been in the form of text-to-speech.

AI Response Generation:

- Natural Language Processing (NLP) and Chatbots: Jurafsky, D., & Martin, J. H. (2020). Speech and Language Processing (3rd ed.). Pearson.

  Manning, C. D., & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. MIT Press.
- Chatbot Development and Frameworks: Documentation and resources for libraries like OpenAI's GPT models, Dialogflow, Rasa, etc.
- Voice Recognition: Documentation for speech recognition libraries and APIs like SpeechRecognition, Google Speech Recognition, etc.
- Text-to-Speech (TTS): Documentation for TTS libraries like pyttsx3, gTTS (Google Text-to-Speech), etc.

Frontend and Avatar Generator:

References:

- HTML, CSS, and JavaScript: MDN Web Docs (https://developer.mozilla.org/en-US/): Comprehensive resources on web development.
- Flexbox Layout: CSS-Tricks Guide on Flexbox (https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
- Interactive Web Elements: W3Schools (https://www.w3schools.com/): Tutorials on HTML forms, buttons, and interactive elements.
- UI/UX Design: Nielsen Norman Group (https://www.nngroup.com/): Articles on user experience design.
- SVG (Scalable Vector Graphics): SVGMDN Web Docs (https://developer.mozilla.org/en-US/docs/Web/SVG): Reference for SVG elements and attributes.