Portable Gemini Terminal Using ESP32

[1]P.Sravan Kumar Reddy[2]P. Sreenivasula Reddy [3]Sk.Adeel Ahamed[3]Sd.Sameer Ali[4]S.Vishnu[5]M.Likhith Chandra

[12] Assistant Professor, ECE, Narayana Engineering College, Nellore, Andhra Pradesh
[3]UG Scholor, ECE, Narayana Engineering College, Nellore, Andhra Pradesh
[4] UG Scholor, ECE, Narayana Engineering College, Nellore, Andhra Pradesh
[5]UG Scholor, ECE, Narayana Engineering College, Nellore, Andhra Pradesh
[6]UG Scholor, ECE, Narayana Engineering College, Nellore, Andhra Pradesh

Abstract: *This project aims to develop a portable Gemini terminal using the ESP32 microcontroller, enabling real-time interaction with Google's Gemini API via text and voice input. By integrating conversational AI into a low-cost, low-power embedded system, this work demonstrates the feasibility of AI-driven IoT devices. The terminal leverages Python to process voice input and generate voice output, addressing key challenges in AI integration on resource-constrained hardware. This project highlights the potential of embedded systems in conversational AI applications and paves the way for intelligent, voice-controlled devices in IoT ecosystems. The system utilizes the ESP32's Wi-Fi capabilities to communicate with the Gemini API, ensuring seamless data exchange. A Python-based interface on a laptop or PC facilitates speech recognition and text-to-speech conversion, enhancing accessibility. The project explores the constraints of deploying AI models in embedded systems, focusing on latency, power consumption, and network dependency. Security considerations are addressed to safeguard API communication and user data. The results demonstrate the viability of AI-enhanced embedded systems for real-world applications, setting a foundation for future advancements in low-power intelligent devices.*

Keywords: *ESP32, Gemini API; conversational AI; IoT; embedded systems; voice recognition; text-to-speech; low-power computing; AI integration; real-time communication; Python; wireless connectivity.*

I. INTRODUCTION

The Internet has grown exponentially, leading to an increasing need for lightweight and privacy-focused communication protocols. One such emerging protocol is Gemini, which provides a minimalistic, text-oriented alternative to the traditional web. Unlike the HTTP protocol, which supports multimedia-heavy web pages, Gemini is designed for simplicity, security, and efficiency. It allows users to browse structured text-based content while avoiding the security risks associated with JavaScript and invasive advertising. As more users and developers become interested in privacy-preserving online experiences, the demand for compact, energy-efficient, and portable devices capable of accessing the Gemini protocol is growing.

This project aims to develop a Portable Gemini Terminal utilizing the ESP32 microcontroller and a PC-based Python program. The ESP32 serves as the network interface and user input/output controller, while the PC software provides voice recognition and speech synthesis capabilities, making the terminal more interactive and accessible. The integration of speech input and output extends the usability of this

device, enabling users to navigate Gemini content without relying solely on traditional text input methods. This combination of embedded hardware and software-based processing makes the project an innovative approach to lightweight browsing.

II. FUNCTIONAL OVERVIEW

Embedded systems:

An embedded system is a specialized computing system designed to perform a dedicated function. It consists of hardware and software embedded within a larger system, providing efficiency, reliability, and real-time processing capabilities. Embedded systems are widely used due to their low power consumption, cost-effectiveness, and adaptability for various applications.

This project utilizes the ESP32 microcontroller, connected to a laptop/PC via USB, to interact with Google's Gemini API in real-time. By integrating conversational AI with embedded hardware, this work explores the feasibility of AI-driven IoT devices.

Components of Embedded Systems:

1. Hardware:
    - ESP32 microcontroller (primary processing unit)
    - USB Interface (for communication between ESP32 and laptop/PC)
    - Laptop/PC (for speech processing and API interaction)
2. Application Software:
    - Python-based interface for speech-to-text and text-to-speech conversion
    - Serial communication via USB between ESP32 and laptop/PC
    - Task execution management for real-time processing
3. Real-Time Processing:
   The ESP32 captures user input, transmits it to the laptop/PC via USB, and receives AI-generated responses from the Gemini API. Since this system does not require a Real-Time Operating System (RTOS), task scheduling is managed efficiently through software optimizations.

Basic Structure of the Embedded System:

1. Input (User Text or Voice via Laptop/PC): The user inputs a query via keyboard or voice.
2. Processing (Laptop/PC & ESP32):
    - The laptop/PC converts speech to text and sends the query to the Gemini API.
    - The Gemini API processes the query and returns a response.
    - The laptop/PC sends the response back to the ESP32 via USB.
3. Output (ESP32 & Speaker):
    - The ESP32 processes the received response.
    - The response is converted into speech and played through a speaker.

USB Communication and Power Management

In this project, the ESP32 communicates with the laptop/PC via USB, eliminating the need for wireless transmission. The USB connection serves a dual purpose:

- Data Transfer: Enabling seamless communication between ESP32 and the Python-based interface on the laptop/PC.
- Power Supply: Providing the necessary power to the ESP32, reducing the need for an external battery.

To enhance portability, future iterations of the project may explore wireless connectivity and wireless power transfer, such as electromagnetic induction-based charging.

III. DESIGN

Existing system:

Traditional AI chatbots primarily rely on cloud-based APIs and require a continuous internet connection for processing. The common methods for accessing AI models, such as Google's Gemini, involve:

- Smartphones or PCs as the primary user interface.
- High-power hardware platforms like Raspberry Pi for running AI models locally.
- Voice-based AI assistants (e.g., Google Assistant, Siri) that operate entirely in the cloud and are not optimized for resource-constrained embedded systems.

These existing solutions have several limitations, including high power consumption, lack of portability, and constant internet dependency. Moreover, they are not well-suited for embedded AI applications, where lightweight, low-power, and cost-effective solutions are required.

Proposed system:

To address these limitations, this project proposes the development of a Gemini Terminal using ESP32 and a laptop/PC. Unlike traditional methods, this approach focuses on low-power embedded AI interaction by leveraging ESP32's capabilities.

The key features of the proposed system include:

1. ESP32 and Laptop/PC Integration via USB:
    - The ESP32 acts as an interface between the user and the Gemini API.
    - A USB connection ensures stable communication and power delivery.
2. Python-Based Program on the Laptop/PC:
    - Handles speech-to-text conversion for voice input.
    - Sends user queries to Google's Gemini API.
    - Receives and processes responses from Gemini.
    - Uses text-to-speech (TTS) to generate voice output.
3. Real-Time Query Processing:
    - Ensures low-latency responses for seamless AI interactions.
    - Maintains a lightweight, **cost-effective**, and **portable** AI-powered terminal.

Block diagram:

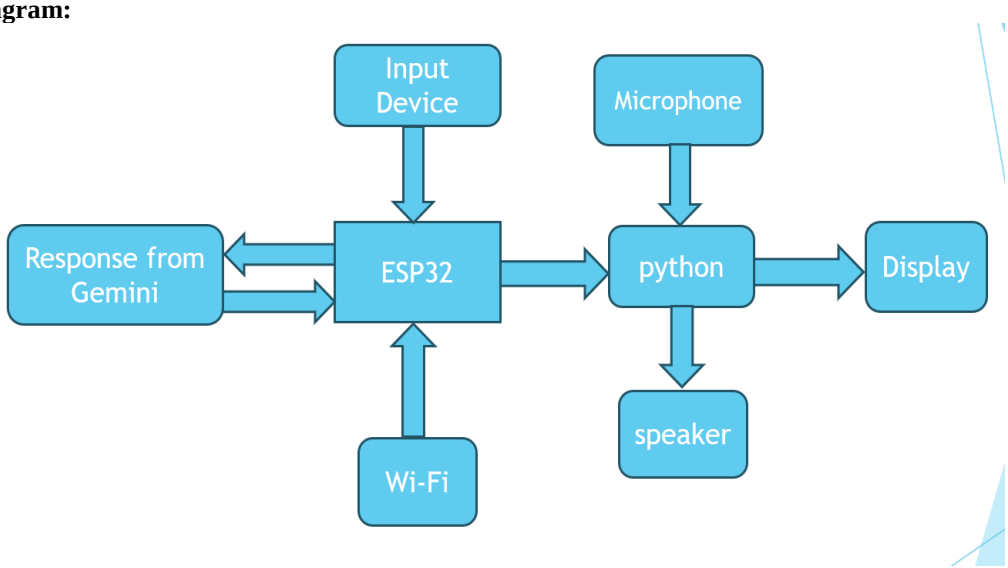

Fig 1: Block Diagram of Gemini Integration with ESP32 and Python

IV. WORKING

The working of each component in this project is explained in this section. Let's talk about them one by one:

Hardware Description:

The hardware setup consists of the following key components:

a) ESP32 Microcontroller

The ESP32 serves as the central embedded processing unit for user queries. It establishes USB-based communication with a laptop or PC, enabling seamless data exchange. Specifically, the ESP32 receives user input through serial communication and efficiently relays the processed responses to the Python program running on the connected laptop or PC.

Fig 2: ESP32 DEV Board

b) Laptop/PC

The system utilizes a Python-based interface to manage core functions, including speech recognition for user input, text processing to formulate queries, and text-to-speech conversion for audio output. This Python environment facilitates communication with an ESP32 microcontroller via USB, enabling the transmission of user queries and the reception of processed responses. Finally, the system presents the textual responses to the user directly within the VSCode environment on the screen, providing a visual interface for interaction.

c) USB Interface

The USB connection acts as a crucial wired link between the ESP32 and the Laptop/PC. This connection facilitates bidirectional data transfer, enabling the exchange of user queries and responses. Furthermore, the USB connection also serves as the power supply for the ESP32, streamlining the setup by eliminating the requirement for an external power source.

Fig 3: ESP32 Cable A to micro USB-A

d) Speaker (Optional for Voice Output)

An optional speaker component provides an audio output pathway, transforming AI-generated responses into audible speech. This feature enables a more natural, voice-based conversational experience, augmenting the system's interactivity. When combined with the visual display of text-based responses, the speaker significantly enhances the user's engagement and understanding.

Fig 4: Speaker for ESP32

SOFTWARE DESCRIPTION:

1) Arduino IDE for ESP32 Development:

The Arduino Software (IDE) is utilized for programming the ESP32 microcontroller, which serves as the core embedded processing unit in this Gemini interaction system. Programs written within the Arduino IDE are called "sketches," saved with the .ino file extension. The IDE's "sketchbook" feature provides a standardized location for storing these sketches, accessible via the File > Sketchbook menu or the Open button.

Key Arduino IDE Functions:

1. Verify: Checks the ESP32 code for compilation errors, reporting memory usage in the console.
2. Upload: Compiles the ESP32 code and uploads it to the connected ESP32 board via the USB connection, facilitating communication with the laptop/PC.
3. New: Creates a new sketch for ESP32 programming.
4. Open: Opens existing ESP32 sketches from the sketchbook.
5. Save: Saves the current ESP32 sketch.

Arduino IDE Setup for ESP32:

Step 1: ESP32 Board Integration:

- To program the ESP32, you need to add the ESP32 board support to the Arduino IDE. This is typically done through the Board Manager by adding the ESP32 board URLs in the Arduino IDE preferences.
- Once the ESP32 board support is installed, select the correct ESP32 board model from the Tools > Board menu.

Step 2: IDE COM Port Setup:

- Connect the ESP32 to the laptop/PC via USB. The Arduino IDE should automatically detect the connected ESP32 and assign it a COM port.
- Select the correct COM port from the Tools > Port menu. This establishes the communication channel for uploading code.

Step 3: Testing ESP32 Communication:

- A simple test to verify the ESP32 setup is to upload a basic "serial communication" sketch. This sketch will send data from the ESP32 to the serial monitor in the Arduino IDE, confirming that communication is established.
- For example, you can use a simple sketch that prints "Hello, ESP32!" to the serial monitor at regular intervals.

Step 4: Uploading the Project Code:

- Once the ESP32 setup is verified, you can upload the code for your Gemini interaction system.
- This code will handle the serial communication with the Python program on the laptop/PC, receiving user queries and transmitting responses.

Step 5: Monitoring Serial Communication:

- Use the Serial Monitor in the Arduino IDE to observe the data being transmitted between the ESP32 and the laptop/PC. This helps in debugging and ensuring proper communication.

Step 6: Integrating with Python:

- Confirm that the ESP32 is successfully communicating with the Python program. This involves sending data from the ESP32 to the Python script and receiving responses back.

2) Python Interface for Gemini Interaction:

The Python component of this system acts as the central software interface, bridging the gap between the user and the Gemini AI. It runs on a laptop/PC and leverages various Python libraries to manage speech recognition, text processing, text-to-speech conversion, and communication with the ESP32 microcontroller.

Key Python Functionalities:

1. Speech Recognition:
    - Utilizes libraries like SpeechRecognition to convert user speech captured by a microphone into text.
    - Handles noise reduction and optimizes speech input for accurate transcription.
2. Text Processing:
    - Processes the transcribed text to formulate queries suitable for the Gemini AI.
    - May involve natural language processing (NLP) techniques to understand user intent and context.
    - Handles data formatting and prepares the query for transmission to the ESP32.
3. ESP32 Communication:
    - Establishes a serial communication link with the ESP32 via USB using libraries like pyserial.
    - Transmits the formatted query to the ESP32 for processing.
    - Receives responses from the ESP32 in text format.
4. Text-to-Speech (TTS):
    - Converts the text response received from the ESP32 into audible speech using libraries like pyttsx3 or Google Text-to-Speech (gTTS).
    - Provides an audio output through the speaker, enhancing the conversational experience.
5. Display Interface:
    - Displays the text response on the screen within the VSCode environment or a dedicated GUI.
    - Provides a visual representation of the interaction, complementing the audio output.

Python Setup and Execution:

Library Installation:

- Install necessary Python libraries using pip: “pip install SpeechRecognition pyserial pyttsx3“
- Ensure that any required audio drivers are installed for speech recognition and TTS.

Step 1: Serial Port Configuration:

- Identify the COM port assigned to the ESP32 and configure it in the Python script.
- Establish a serial connection with the ESP32 using the pyserial library.

Step 2: Speech Recognition Setup:

- Configure the microphone input and set up the SpeechRecognition library for accurate speech transcription.

Step 3: TTS Configuration:

- Initialize the chosen TTS library and configure the voice and speech rate.

Step 4: Main Execution Loop:

- The main Python script runs in a loop, continuously listening for user input, processing it, sending it to the ESP32, receiving responses, and providing audio and visual output.

Step 5: Error Handling:

- Implement error handling to manage potential issues such as serial communication errors, speech recognition failures, and TTS errors.

Step 6: Integration with VSCode:

- If using VSCode as the display interface, ensure that the Python script can interact with the VSCode environment to display the responses.

3) Gemini API Integration:

The Gemini API is the core AI engine used in this project to generate intelligent responses based on user queries. The Python interface communicates with the Gemini API, sending user inputs and receiving AI-generated responses in real time.

1. API Key and Authentication:

   To access the Gemini API, a unique API key is required for authentication. The API key is issued by Google and must be kept secure to prevent unauthorized access. The key is typically stored in an environment variable or a secure configuration file to avoid exposing it in the source code.

2. Create a Google Cloud Account:
   - Sign in to the Google Cloud Console and create a new project.
3. Enable the Gemini API:
   - Navigate to the "APIs & Services" section and enable the Gemini API for the project.
4. Generate an API Key:
   - In the "Credentials" section, create a new API key and copy it for use in the Python script.
5. Securely Store the API Key:
   - It is recommended to store the API key in an environment variable or a separate configuration file (e.g., .env) to enhance security.
   - Example of storing the API key in an environment variable (.env file): GEMINI_API_KEY = your_api_key_here
   - Then, access it in Python using the dotenv library:

     ```
     import os
     from dotenv import load_dotenv
     load_dotenv()
     API_KEY = os.getenv("GEMINI_API_KEY")
     ```

6. Sending Queries to the Gemini API:

   Once authenticated, the Python script formats user queries and sends them to the Gemini API using HTTP requests. The response is processed and displayed on the screen or converted into speech.

   Example API request using Python's requests library:

   Python:

   ```
   import requests
   API_URL = "https://api.gemini.google.com/v1/query"
   headers = {"Authorization": f"Bearer {API_KEY}"}
   data = {"prompt": "Hello, how can I assist you?"}

   response = requests.post(API_URL, json=data, headers=headers)
   print(response.json())  # Display the AI response
   ```

7. Integration with ESP32 Communication:
    - The Python script receives the user input (either text or speech), processes it, and sends the formatted query to the Gemini API.
    - The API response is then relayed to the ESP32 via USB communication, enabling real-time AI interaction.
    - The ESP32 transmits the received response to the laptop/PC, where it is displayed or converted into speech.
8. Error Handling and Rate Limits:
    - The Gemini API may impose rate limits, restricting the number of queries per minute. To handle this, the Python script includes error handling mechanisms such as retry logic and cooldown periods.
    - If the API key becomes invalid or expires, the system alerts the user and prompts for reauthentication.

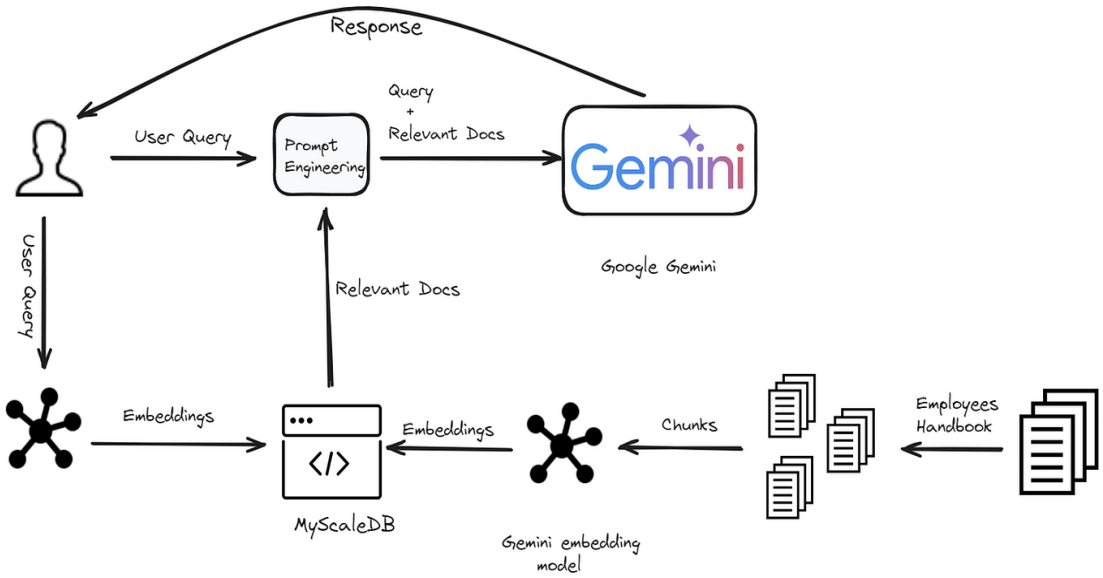

Fig 5: Building an ESP32 AI Q&A System Using Gemini API

This API integration allows the ESP32-based system to leverage the power of advanced AI models while maintaining a lightweight and efficient design. Future improvements may involve optimizing the API calls for better response times and reducing dependency on cloud services by exploring edge AI alternatives. Security and authentication measures can also be strengthened to prevent unauthorized access to the API key and sensitive data transmissions. Implementing **token-based authentication**, encrypting communication between the ESP32 and the Python interface, and securing API requests using OAuth or API gateways can ensure data integrity and protect user privacy.

V.RESULTS

In this working model, the integration of ESP32, USB communication, and Google's Gemini API has successfully demonstrated the feasibility of an AI-powered embedded system for real-time conversational interaction. The project establishes a seamless connection between the ESP32 microcontroller and a laptop/PC via USB, enabling efficient query processing and response generation. The ESP32 functions as an intermediary, transmitting user queries to the Gemini API and receiving AI-generated responses. The responses are displayed on the laptop/PC screen within the VSCode environment, ensuring a user-friendly interface. Additionally, the Python program handles speech-to-text (STT) and text-to-speech (TTS) conversions, allowing both text-based and voice-based interactions.

This implementation effectively transforms the ESP32 into a portable AI terminal, leveraging its low power consumption and compact form factor. The use of USB communication eliminates the need for Wi-Fi connectivity, making the system more stable and power-efficient. The project successfully validates the potential of embedded AI applications, paving the way for future enhancements such as wireless communication, battery-powered operation, and standalone AI assistants.

Furthermore, the modular and scalable architecture of this system allows for the seamless integration of additional functionalities, including offline AI processing and edge computing capabilities. The ESP32's versatility and cost-effectiveness make it an ideal candidate for further research and experimentation in embedded AI applications. Future improvements may involve the addition of a dedicated display module, reducing dependency on an external PC, or incorporating a wake-word detection system for hands-free operation.

Beyond personal AI terminals, this system has the potential for broader applications in IoT, robotics, and assistive technologies. By integrating AI-driven decision-making, the ESP32 could be deployed in smart environments to process commands, provide voice-based assistance, or control connected devices efficiently. Additionally, by optimizing the response latency and refining the speech recognition accuracy, the system can be further enhanced for real-time, natural interactions.

The successful development and testing of this AI-powered embedded system affirm the feasibility of compact, low-power AI devices for portable use. By leveraging advancements in hardware acceleration and AI models, this concept can be expanded into various real-world applications, including educational tools, personal assistants, and AI-enhanced embedded systems. The results indicate a promising direction for future research and innovation in the field of AI-driven microcontroller-based applications.

ESP32 OUTPUT:

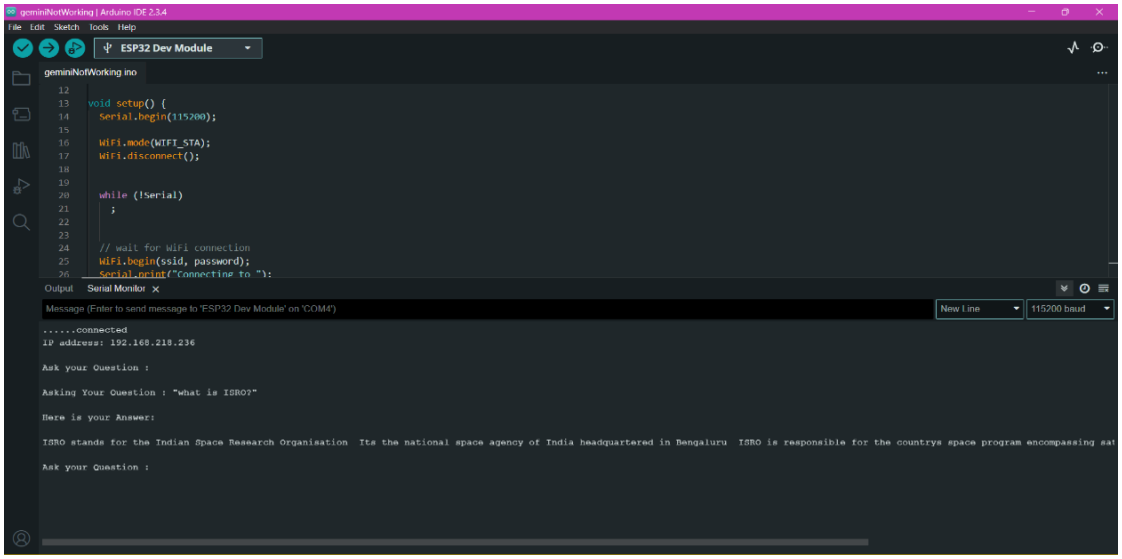

Fig 6: ESP32 output in Arduino IDE

PYTHON OUTPUT:

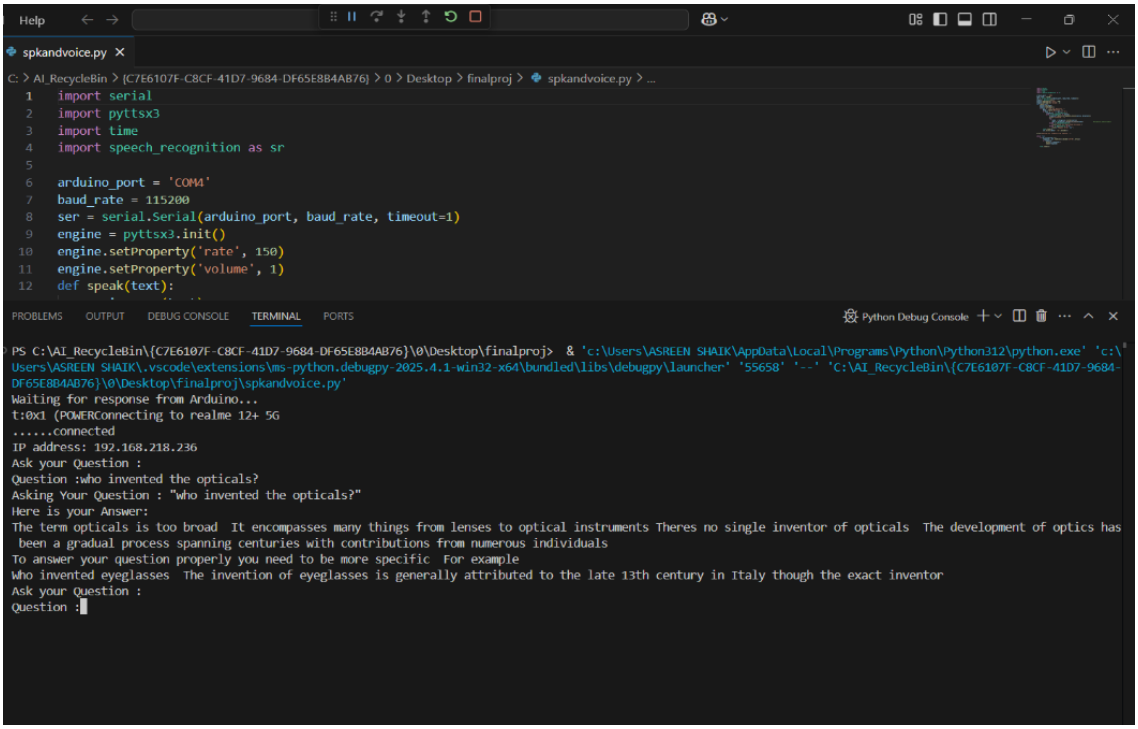

Fig 7: ESP32 Output for Python Program

VI. CONCLUSION

This project successfully demonstrates the feasibility of an AI-powered embedded system by integrating ESP32, USB communication, and Google's Gemini API. By leveraging the ESP32's low-power architecture and real-time processing capabilities, the system provides a cost-effective and portable AI terminal that facilitates seamless human-machine interaction. The project effectively establishes a stable, USB-based communication link between the ESP32 microcontroller and a laptop/PC, eliminating the dependency on Wi-Fi while ensuring low-latency query processing. The integration of Python-based speech-to-text (STT) and text-to-speech (TTS) functionalities further enhances the usability of the system, allowing both text-based and voice-based AI interactions.

The findings of this project highlight the potential of embedded AI applications in IoT, showcasing how resource-constrained microcontrollers can efficiently interface with advanced AI models like Google's Gemini. The lightweight and energy-efficient nature of this system makes it a viable candidate for future applications in smart IoT devices, voice-controlled assistants, and AI-driven embedded solutions. Future improvements, such as wireless communication, standalone operation with a display, and battery-powered functionality, could further enhance the practicality and mobility of the system. This project lays a strong foundation for the next generation of embedded AI solutions, bridging the gap between low-power microcontrollers and cloud-based AI capabilities.

Additionally, the modular nature of this implementation allows for easy scalability and adaptation to various use cases. For example, integrating a dedicated touch display and haptic feedback mechanisms could improve user interaction, making the system more intuitive and accessible. Furthermore, optimizing the firmware to support lightweight on-device AI models could enable offline processing, reducing reliance on cloud-based AI services and enhancing privacy and security.

Beyond personal AI terminals, this system holds significant promise for industrial automation, healthcare, and assistive technology applications. In medical settings, a similar embedded AI system could assist individuals with disabilities through voice-guided interfaces or provide real-time translation services. The use of AI-driven microcontrollers in smart homes and robotics could further automate everyday tasks, improving convenience and efficiency.

By refining and expanding upon this proof of concept, future iterations could incorporate improved energy management strategies, allowing for extended battery life in portable use cases. Enhancements in AI model efficiency and hardware acceleration could also contribute to better response times and real-time decision-making capabilities. This project serves as an important step towards the integration of AI into embedded systems, demonstrating how microcontrollers can be transformed into intelligent, interactive devices.

The successful implementation and validation of this system reinforce the feasibility of AI-powered microcontrollers in real-world applications. As AI continues to evolve, the development of embedded AI systems like this will play a crucial role in shaping the future of intelligent and autonomous devices. The insights gained from this project will serve as a valuable foundation for further research, innovation, and practical deployments in the field of AI-driven embedded computing.

VII. REFERENCES

[1] W. Madesh, *Building an Arduino-Based AI Q&A System Using Gemini API*, Medium, 2024. [Online]. Available: https://medium.com/@waranmadesh826/building-an-arduino-based-ai-q-a-system-using-gemini-api-b93fdeebc7c4.

[2] B.S. Sarjerao, A. Prakasarao, "A Low Cost Smart Pollution Measurement System Using REST API and ESP32" 3rd International Conference for Convergence in Technology, I2CT 2018, November 2018.

[3] H. Du et al., "Enabling AI-generated content (AIGC) services in wireless edge networks," 2023.

[4] A. V. Bapat and L. K. Nagalkar, "Phonetic Speech Analysis for Speech to Text Conversion," 2008 IEEE Region 10 and the Third International Conference on Industrial and Information Systems, 2008.

[5] L. Ouyang et al., "Training language models to follow instructions with human feedback," in Proc. Adv. Neural Inf. Process. Syst., 2022.

[6] K. Madani, A. M. Rinaldi, and C. Russo, Combining linked open data and multimedia knowledge base for digital cultural heritage robotic applications. In 2021 IEEE International Symposium on Multimedia (ISM), pp. 196-203, IEEE, November, 2021.

[7] S. Alam, K. Islam, N. Sharmila, Z. R. Sovon, and R. M. Rahman, Image Captioning-Bangladesh's Heritage Perspective Using Deep Learning. In 2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), pp. 1-8, IEEE, June, 2022.

[8] Q. Li, Intelligent intangible cultural heritage innovation platform under the background of big data and virtual systems. In 2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS), pp. 560-563, IEEE, February, 2022.

[9] S. F. Rashid, and R. P. Qasha, Cloud-Based Big Data Approach for Managing Multi-Types Data of Cultural Heritage. In 2022 International Conference on Communications, Information, Electronic and Energy Systems (CIEES), pp. 1-6, IEEE, November, 2022.

[10] Radford A, Wu J, Amodei D, Amodei D, Clark J, Brundage M, et al. Better language models and their implications. OpenAI Blog https://openai com/blog/better-language-models. 2019.

[11] L. Podo, M. Ishmal, and M. Angelini, "Vi(E)va LLM! A conceptual stack for evaluating and interpreting generative AI-based visualizations," 2024.

[12] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright CL, Mishkin P, et al. Training language models to follow instructions with human feedback. arXiv preprint arXiv:220302155. 2022.

[13] B.S. Sarjerao, A. Prakasarao, "A Low Cost Smart Pollution Measurement System Using REST API and ESP32" 3rd International Conference for Convergence in Technology, I2CT 2018, November 2018.