Internet of Things Fundamentals

Subject Project

BS AI 6th Semester SP-25 (AIE-3079)

Date:26-6-26

Project Title: joona-the AI powered voice control system

Group Name/no.: 4 musketeers

Team Members:

Members	Registration no	Name	Signature
Member-1 (Leader)	22-ntu-cs-1341	**Fatima Ahmad**	
Member-2	**22-ntu-cs-1344**	**Maryam Munawar**	
Member-3	**22-ntu-cs-1342**	**Fatima Riaz**	
Member-4	**22-ntu-cs-1353**	**Marriam Sajjad**	

Contributions in % of each Team Members for each component					
		Member-1	Member-2	Member-3	Member-4
Distribution Components		Name	Name	Name	Name
Coding	ESP32coding	35	30	10	25
	Python Coding	40	40	10	10
UI Design					

Database	45	25	15	15
Cloud Integration				
IoT Gateway	35	50	5	10
Edge Processing				
Documentation	10	30	30	30
Presentation Design	5	35	30	30
Hardware integration	15	15	20	50
Hardware Testing	30	30	15	25
Hardware display	15	15	40	30
server	30	50	10	10

To be filled by the evaluator

Team-Based Evaluation (60 Marks)

Criteria	Obtained Marks	Out of
System Design & Architecture		10
Hardware Integration & Circuit Setup		10
IoT Gateway and Cloud Communication		10
Working Prototype Demonstration		10
Performance & Reliability Testing		10
Presentation		10
Total (Team-Based)		60

Individual-Based Evaluation (40 Marks per Member)

	Member 1	Member 2	Member 3	Member 4
Criteria				
Understanding of the Project & Role	/10	/10	/10	/10
Code Contribution and Explanation	/10	/10	/10	/10
Q/A VIVA	/10	/10	/10	/10
Documentation/Reporting & Communication	/10	/10	/10	/10
Total (Individual-Based)	/40	/40	/40	/40
Total Overall (60+40)	/100	/100	/100	/100
Weightage Lab Grade (50)				

Table of Contents

1. Abstract / Executive Summary

Joona is a real-time, voice-controlled IoT assistant built using the ESP32-S3 microcontroller. It enables users to control appliances like lights and fans through natural voice commands. On button press, the ESP32 records audio via an I2S microphone and sends it to a Flask server, where AssemblyAI transcribes it and an intent parser (with OpenAI fallback) interprets the command. Google TTS generates a spoken response, which is played back through an I2S speaker. Commands for device control trigger GPIO-based relays, while reminder tasks are stored in Firebase Firestore and triggered using APScheduler. Joona demonstrates the seamless integration of embedded hardware with cloud-based AI services for intelligent home automation.

2. Introduction

2.1 Background & Motivation

Voice-controlled systems like Alexa and Google Assistant have revolutionized smart living, but they are often costly, internet-dependent, and closed-source. For students, developers, and DIY enthusiasts, building an accessible and customizable alternative becomes essential. Joona was designed to bridge this gap by combining the ESP32-S3 microcontroller with advanced cloud-based AI services, enabling intelligent, real-time control of IoT devices using natural voice interaction.

2.2 Problem Statement

Traditional IoT systems often rely on manual inputs or mobile apps and lack natural language interaction. Most microcontrollers cannot easily integrate high-level speech recognition, AI-based intent detection, and real-time device control. There's a need for a low-cost, offline-friendly, customizable voice assistant that can perform localized smart tasks and provide responses similar to commercial assistants.

2.3 Project Goals

- Develop a smart assistant using ESP32-S3 with I2S mic/speaker and relay modules.
- Enable voice command capture and HTTP communication with a Flask server.
- Use AssemblyAI for Speech-to-Text, OpenAI for fallback intent recognition, and Google TTS for audio replies.
- Control home appliances like LEDs and fans through GPIO based on interpreted commands.
- Implement voice-based reminder storage and scheduled alerts using Firebase and APScheduler.

3. Literature Review

3.1 Relevant IoT / ESP32 Concepts

The ESP32-S3 microcontroller is a powerful and cost-effective platform ideal for IoT applications. It features built-in Wi-Fi and Bluetooth, dual-core processing, and real-time GPIO control, making it well-suited for tasks involving connectivity, automation, and rapid response. One of its key features is support for the I2S protocol, allowing high-fidelity audio data transfer, which is critical for capturing voice commands through a microphone and playing responses through a speaker. Combined with HTTP-based communication, cloud services, and real-time data handling, the ESP32-S3 provides a strong foundation for intelligent, voice-enabled IoT systems.

Key IoT/ESP32 Concepts used in this project:

- I2S protocol for microphone (input) and speaker (output) audio streaming
- GPIO relay control for devices like lights and fans
- HTTP communication between ESP32 and Flask server
- Cloud integration with AssemblyAI, OpenAI, and Google TTS
- Real-time task scheduling and remote reminder management via Firebase + APScheduler

3.2 Similar Projects / Research

Over the years, various open-source and academic initiatives have explored the integration of voice assistants with microcontrollers. Projects such as ESP-Skainet (Espressif's official voice SDK), Jasper (an offline Python-based assistant), and Arduino-compatible Google Assistant replicas demonstrate early efforts toward voice-enabled IoT. However, many of these are either limited by offline constraints or lack deep hardware control and customization. In contrast, Joona offers a comprehensive and flexible solution by combining the ESP32-S3 with modern cloud APIs and local hardware integration—delivering a low-cost, intelligent, and fully interactive assistant system.

Key comparisons and advantages of Joona:

- Combines real-time audio streaming with ESP32-S3 and I2S for mic and speaker control
- Uses cloud-based STT (AssemblyAI), intent recognition (OpenAI), and TTS (Google)
- Includes reminder management via Firebase Firestore and APScheduler
- Supports manual activation via hardware button for efficiency and control
- Offers greater customization, flexibility, and hardware interaction than most open-source assistants

4. Methodology / System Design

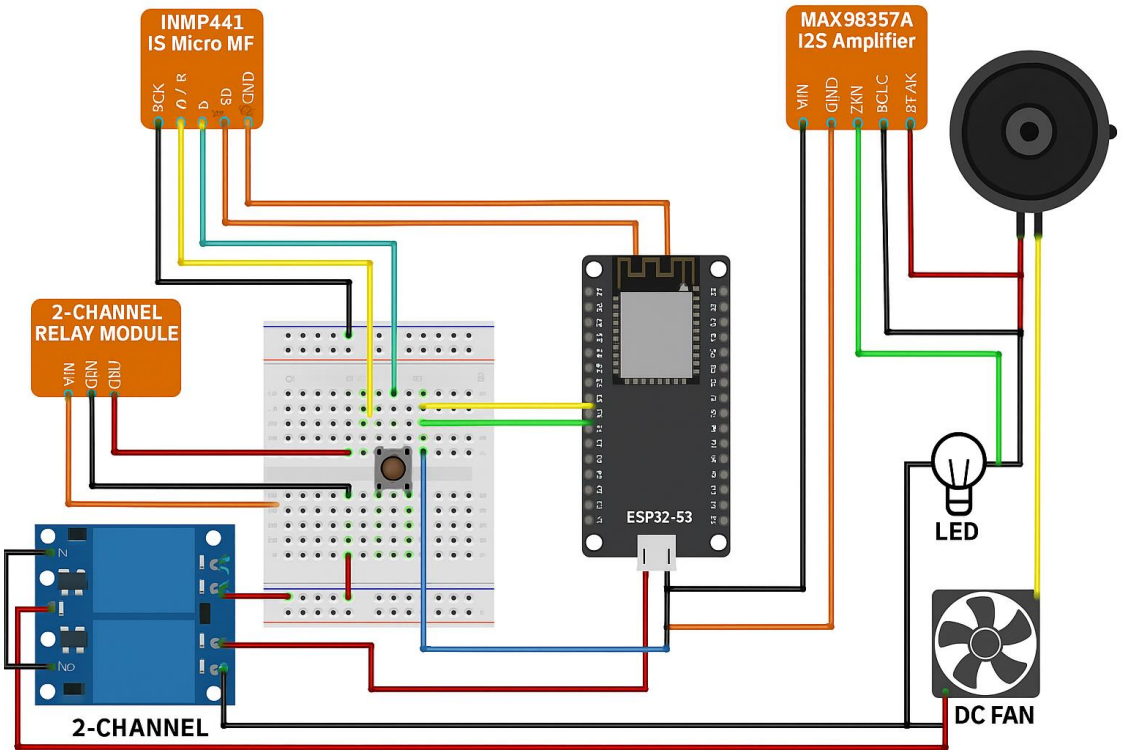
4.1. Hardware Components

The hardware components used in Joona are selected to enable real-time voice interaction, device control, and audio feedback. Below is the list of components:

Hardware List:

- ESP32-S3 development board (dual-core, I2S support)
- INMP441 I2S microphone module (for voice capture)
- MAX98357A I2S amplifier module with speaker (for audio playback)
- 2-Channel 5V Relay Module (to control light and fan)
- LED Light (as a controllable output device)
- DC Fan (as a controllable output device)
- Push Button (to trigger voice capture manually)
- Transistors & Resistors (for relay control logic)
- Jumper wires and breadboard (for prototyping)
- 5V Power Supply / USB

4.2. Circuit Diagram (Fritzing/Proteus/Other) with Labels



4.3. Software Design

4.3.1. System Workflow

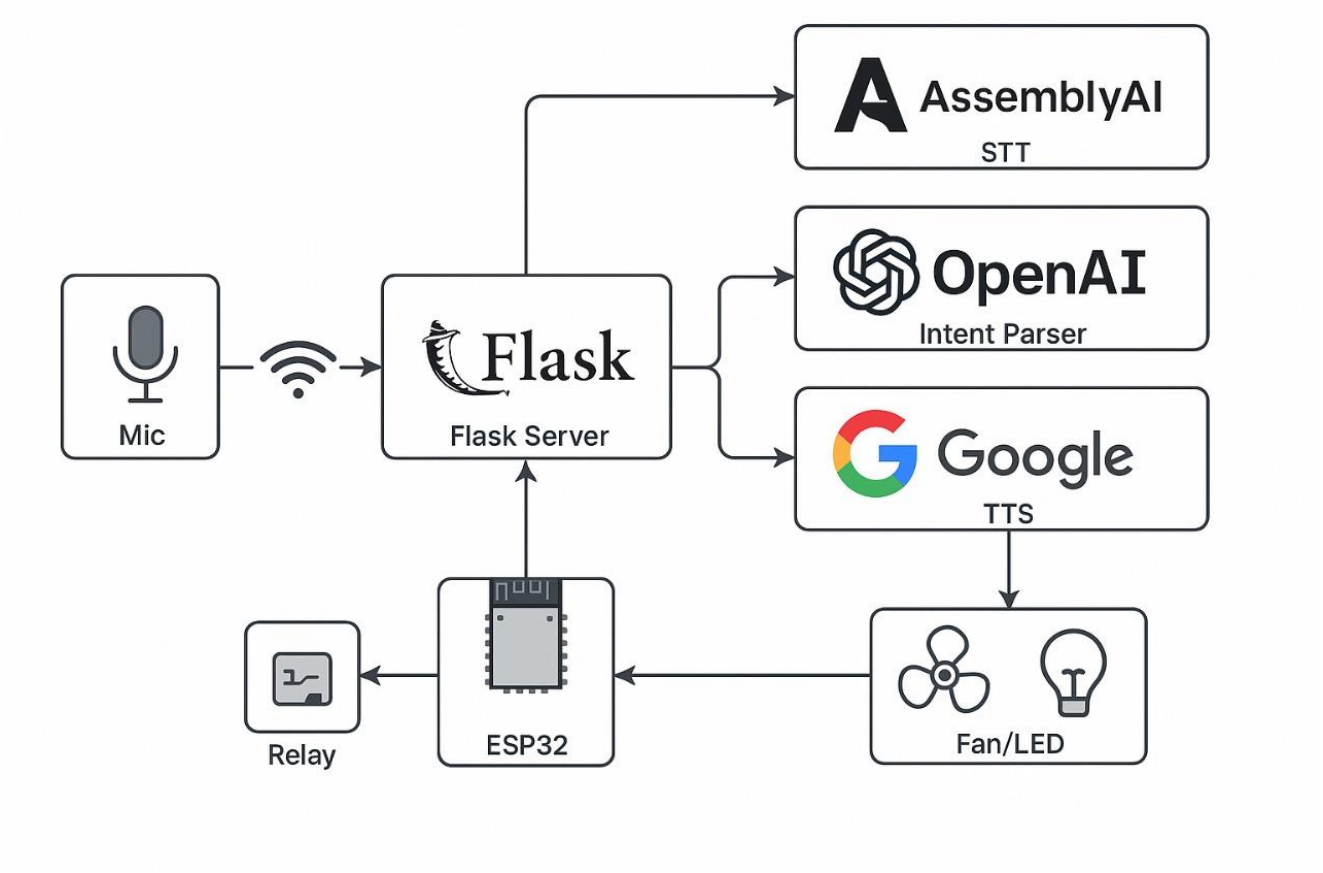
The Joona Voice-Controlled IoT Assistant follows a step-by-step voice interaction and device control pipeline. The system is designed for low-latency processing, combining edge hardware (ESP32-S3) with cloud-based AI services for natural language understanding and feedback. The entire workflow is initiated manually through a push-button on the ESP32-S3.

System Flow:

1. 🟢 Button Press (on ESP32-S3)
   The user manually presses a physical button, triggering the ESP32 to begin audio capture.
2. 🎙️ Audio Capture (via INMP441 Mic)
   The ESP32-S3 records voice input through the I2S microphone for a fixed duration (e.g., 3 seconds).
3. 🌐 HTTP Audio Upload
   The recorded audio is packaged in WAV format and sent via HTTP POST to the Flask server.
4. 🧠 Speech-to-Text (AssemblyAI)
   The server sends the audio to AssemblyAI's STT API to obtain a transcribed text.
5. 🔍 Intent Detection (Rule-Based + OpenAI fallback)
   The transcribed text is analyzed to identify the intent (e.g., control device, set reminder).
   Fallback queries are handled using OpenAI GPT to ensure intelligent conversation.
6. 🔁 Response Generation
   Based on intent:
   - If device control → a structured JSON is sent back to ESP32 to operate relays.
   - If reminder → data is stored in Firestore, and APScheduler checks for timed triggers.
   - A voice reply is generated using Google TTS and saved as an MP3.
7. ⬇️ Audio Download & Playback
   ESP32 downloads the MP3 response from the Flask server and plays it through the MAX98357A I2S amplifier and speaker.
8. 🔌 Relay Activation
   If the intent includes device control, ESP32 activates/deactivates GPIO pins to control lights or fan via relay module.

This workflow ensures seamless interaction between the user and the system, providing real-time responses and reliable device automation via voice commands.

4.3.2. Flowchart or Architecture Diagram



4.3.3. Tools & Libraries Used

The Joona project utilizes a mix of embedded system libraries, cloud APIs, and backend frameworks to enable seamless voice interaction and IoT control. Below is a categorized breakdown of the tools and libraries:

🔧 Embedded (ESP32-S3 – Arduino):

- WiFi.h – for internet connectivity
- HTTPClient.h – for making HTTP requests to the server
- driver/i2s.h – for I2S audio communication with mic and speaker
- ArduinoJson – for parsing JSON responses from the server
- digitalWrite / pinMode – for controlling relays (LED/Fan)
- Core IDE: Arduino IDE

💻 Backend (Flask Server – Python):

- Flask – lightweight web server to handle HTTP requests and responses
- apscheduler – to schedule time-based reminders (every minute check)
- firebase_admin – to interact with Firestore database
- requests – to call external APIs (AssemblyAI, OpenAI, Google TTS)
- python-dotenv – to manage environment variables securely

☁️ Cloud APIs / Services:

- AssemblyAI – for Speech-to-Text (STT) transcription of user voice
- OpenAI GPT – for intent fallback, general conversation handling
- Google Text-to-Speech – for generating audio responses in MP3 format
- Firebase Firestore – to store and trigger voice-based reminders

🛠️ Development Tools:

- Arduino IDE – for writing and flashing ESP32 firmware
- Thonny / VS Code – for Python server development
- Postman – for testing API endpoints
- Fritzing – for circuit diagram design

5.Implementation

5.1. Step-by-Step Setup (Wiring, Configurations)

1. Wiring Connections:

🔊 I2S Microphone (INMP441)

- SCK (BCLK) → GPIO12
- WS (L/R Clock) → GPIO11
- SD (Data Out) → GPIO10
- VCC → 3.3V
- GND → GND

🔈 I2S Speaker (MAX98357A)

- BCLK → GPIO7
- LRC → GPIO9
- DIN → GPIO6
- VCC → 5V
- GND → GND

🔌 Relay Module (for fan & light)

- IN1 → GPIO21 (Light)
- IN2 → GPIO20 (Fan)
- VCC → 5V
- GND → GND
- Relay outputs connected to fan and LED with transistor+diode safety circuit.

⭕ Push Button

- One terminal → GPIO (user-defined, e.g., GPIO0)
- Other terminal → GND
- Add 10kΩ pull-down resistor if needed.

2. Initial Configurations:

- Configure I2S input for microphone and output for speaker.
- Set GPIOs for relay and button.
- Connect to Wi-Fi on ESP32.
- Start Flask server and ensure AssemblyAI, OpenAI, and Google TTS keys are set.
- Test Firebase connection for storing reminders.

5.2 Code Snippets (with Comments)

ESP32 (C++ – Arduino IDE)

Wi-Fi Setup:
Connects ESP32-S3 to Wi-Fi for cloud communication.

```
void setup_wifi() {
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
}
Serial.println("✅ WiFi connected");
}
```

I2S Microphone Initialization:
Configures ESP32 to read audio from the INMP441 mic using I2S.

```
void setup_i2s_mic() {
i2s_config_t config = { ... };
i2s_pin_config_t pin_config = { ... };
i2s_driver_install(I2S_PORT, &config, 0, NULL);
i2s_set_pin(I2S_PORT, &pin_config);
}
```

I2S Speaker Setup:
Enables ESP32 to play audio (MP3 response) via MAX98357A amplifier.

```
void setup_i2s_speaker() {
i2s_config_t config = { ... };
i2s_pin_config_t pin_config = { ... };
i2s_driver_install(I2S_SPK_PORT, &config, 0, NULL);
i2s_set_pin(I2S_SPK_PORT, &pin_config);
}
```

Stream Audio to Flask Server:
Captures audio and POSTs it to Flask for transcription and intent recognition.

```
void streamAudioToServer() {
connect to server;
send WAV header;
```

read 3 sec audio from I2S mic;
POST audio over HTTP;
read response JSON;
if response_audio exists → play via I2S speaker;
if intent → control GPIO relays.
}

Device Control (Relay Activation):

void controlDeviceFromServer(JsonDocument& doc) {
if (intent == "turn_on_device" && "light") → digitalWrite(relayLightPin, LOW);
if (intent == "turn_off_device" && "fan") → digitalWrite(relayFanPin, HIGH);
}

Flask Server (Python)

Audio Upload Endpoint (/upload-audio):
Receives WAV file from ESP32, transcribes it, parses intent, and responds with text and TTS audio.

@app.route('/upload-audio', methods=['POST'])
def upload_audio():
Save WAV file;
Call AssemblyAI to get transcript;
Parse intent using OpenAI & rules;
If reminder → store in Firestore;
Generate TTS reply with Google;
Return transcript, intent, response, and audio file path.

TTS Audio Playback Endpoint:

@app.route('/play-audio/<filename>')
def play_audio(filename):
return send_from_directory(TTS_FOLDER, filename)

Scheduler to Trigger Reminders:

scheduler = BackgroundScheduler()
scheduler.add_job(check_reminders, 'interval', minutes=1)

def check_reminders():
fetch reminder time from Firestore;
compare with current time;
print reminder if due.

5.3. Challenges & Solutions

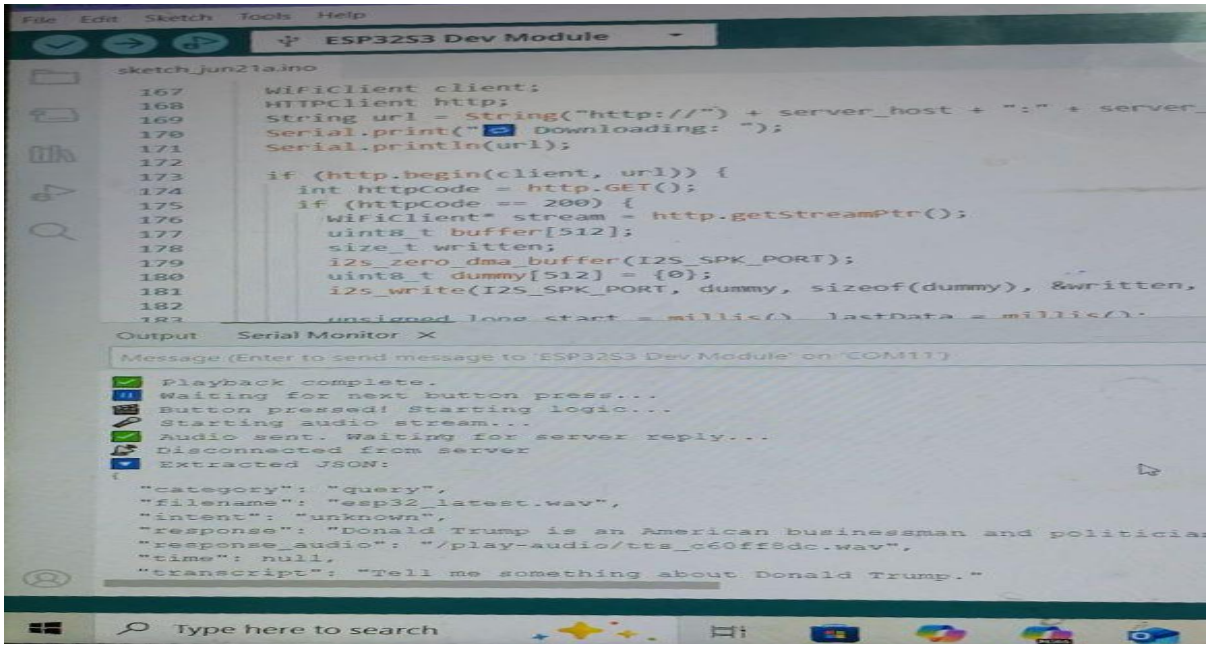
During the development of the Joona IoT Voice Assistant, we encountered several technical and integration challenges. Below is a summary of key obstacles and the solutions we implemented:

Challenge 1: I2S Microphone and Speaker Syncing
Problem: The ESP32-S3 had to switch between I2S receive (mic) and transmit (speaker) modes, which caused playback issues.
Solution: We implemented i2s_driver_uninstall() and reinitialized the speaker driver before every playback, ensuring smooth switching and avoiding conflicts.
Challenge 2: Slow or Corrupted Audio Transmission
Problem: Streaming raw audio from ESP32 to Flask server via HTTP sometimes failed or resulted in partial data.
Solution: We buffered audio in chunks and sent a proper WAV header to ensure that the Flask server received valid audio formats for AssemblyAI processing.
Challenge 3: Intent Misclassification
Problem: The rule-based parser occasionally failed for complex or ambiguous sentences.
Solution: We added a fallback mechanism using OpenAI's chat model for better interpretation and natural responses when confidence was low.
Challenge 4: Reminder Time Parsing
Problem: Users could say reminder times in various formats (e.g., "set alarm for 6 PM", "remind me at 6"), making it difficult to extract structured time.
Solution: We used regular expressions and cleaned the time strings to normalize formats and improve matching with the APScheduler trigger time.
Challenge 5: Playback Cutoff on ESP32
Problem: Audio responses from the server were being cut short during speaker playback.
Solution: We used i2s_zero_dma_buffer() and added dummy writes to flush the audio buffer before actual playback.
Challenge 6: Button-Based Audio Trigger (Future Addition)
Problem: Initially, audio recording started automatically in setup().
Solution: We planned to integrate a push-button to control when audio capture starts, ensuring better user control and energy efficiency.

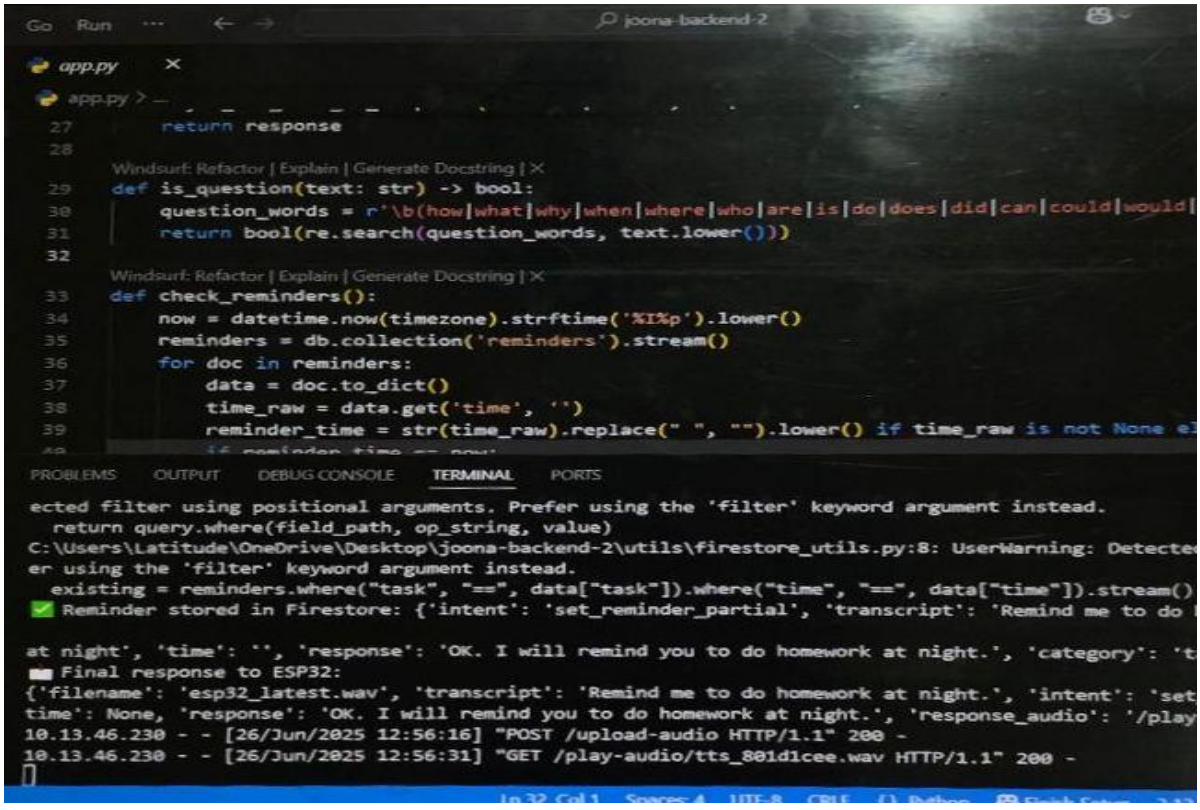
6. Results & Discussion

6.1. Screenshots / Serial Monitor Outputs

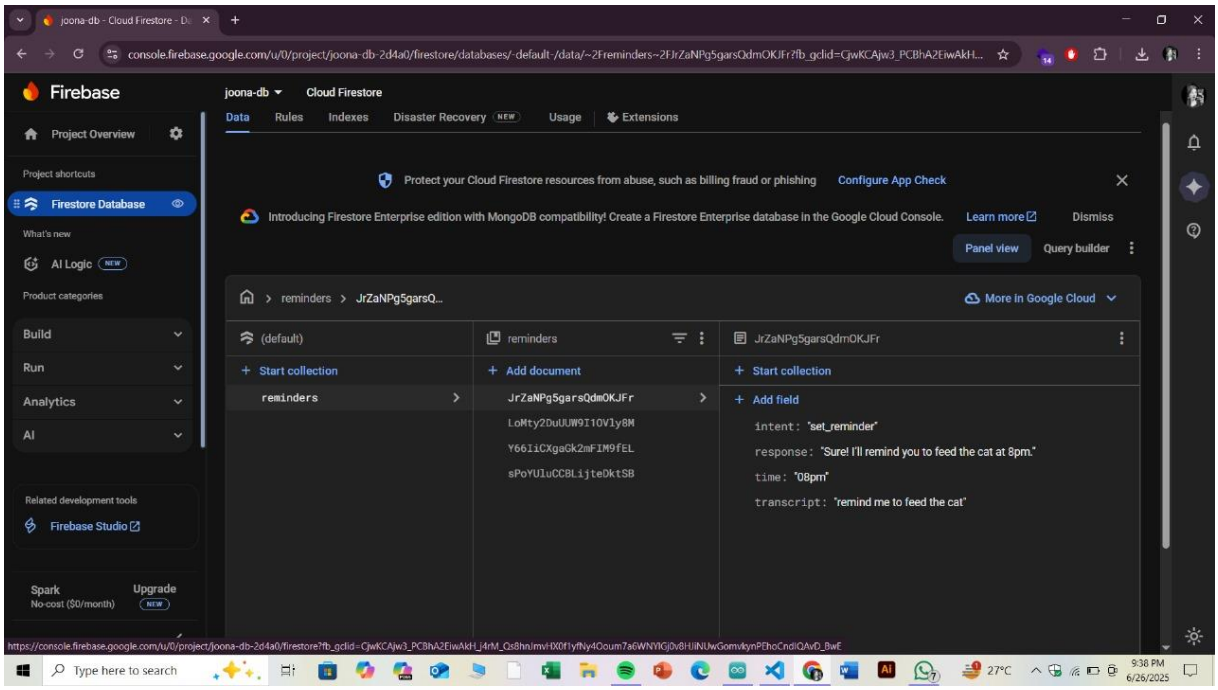
Serial Monitor Outputs



Terminal Output



Firebase Output



6.2. Performance & Response Time Analysis

To evaluate the responsiveness and reliability of the Joona voice assistant system, we conducted time-based measurements from voice input to device actuation and voice response playback. Below is a summary of the observed timings and overall performance:

📊 Response Time Breakdown:

- Audio Capture Duration (ESP32): ~3 seconds (fixed buffer)
- Audio Upload to Server (via HTTP): ~1–2 seconds (depending on Wi-Fi)
- Speech-to-Text Conversion (AssemblyAI): ~1.5–3 seconds
- Intent Parsing + TTS Generation (OpenAI + Google TTS): ~2–3 seconds
- Audio File Download from Server: ~1–2 seconds
- Audio Playback Duration: variable (depends on length of TTS output)
- Device Actuation Delay (Relay ON/OFF): < 300 ms (almost instant)

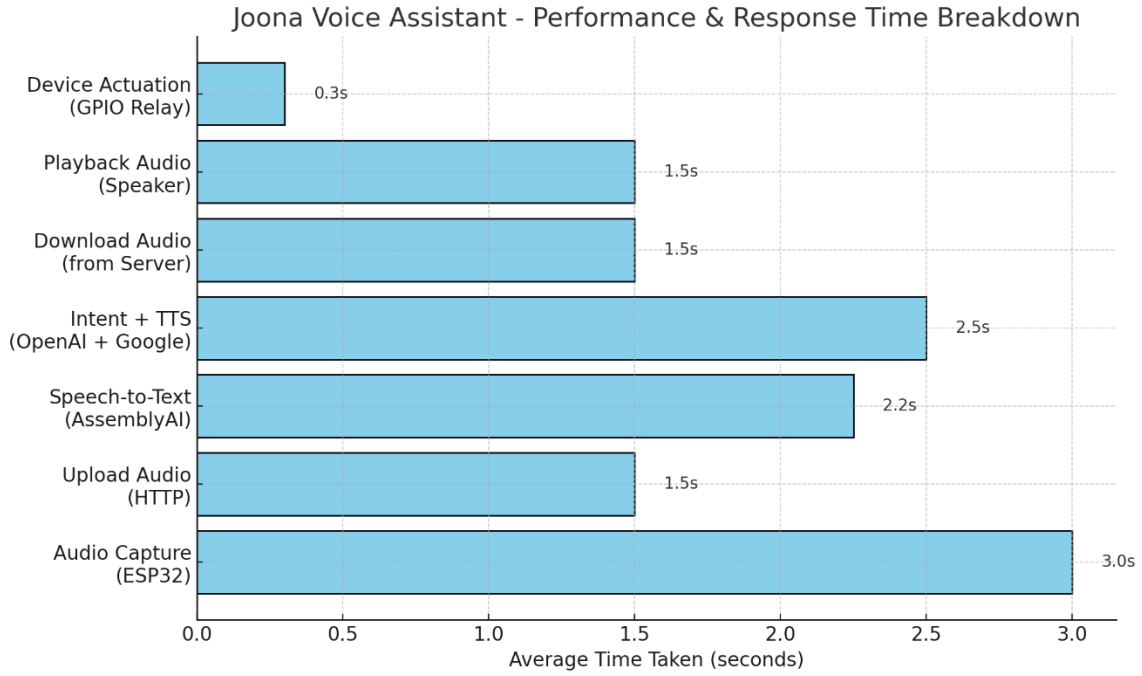
⏱️ Total Round-Trip Time:
≈ 8 to 12 seconds (from button press to voice reply + action)

📌 Observations:

- The most significant delay occurred during cloud processing (STT & TTS).
- Local device control via GPIO pins was nearly instantaneous.
- Response time can vary based on internet speed and server load.

✅ Overall Evaluation:

- The assistant offers acceptable real-time performance for home automation.
- Network stability is essential for consistent operation.
- Buffering and playback techniques ensured smooth audio delivery.



6.3. Comparison with Expected Outcomes

Our goal was to develop a real-time voice-controlled IoT assistant capable of understanding natural language, responding intelligently, and controlling hardware devices like lights and fans. Overall, the implemented system successfully met most expectations, with some minor trade-offs observed:

✅ Expected Outcomes Achieved:

- Accurate speech-to-text conversion using AssemblyAI, even in moderately noisy environments.
- Correct intent recognition through OpenAI and rule-based fallback logic.
- Successful relay control of devices (lights and fan) through ESP32 GPIO.
- Seamless playback of TTS-generated audio responses on the speaker.
- Functional real-time reminder scheduling via Firebase Firestore and APScheduler.

⚠️ Observed Gaps & Differences:

- Occasional delays (2–5 seconds) due to network latency in STT and TTS processing.
- Mic sensitivity limited very quiet or distant voice recognition.
- System requires stable Wi-Fi; no offline fallback implemented yet.
- Current implementation lacks dynamic multi-user handling.

🎯 Overall, the final system closely aligns with the envisioned goals of a responsive, real-time conversational IoT interface using low-cost embedded hardware and AI services.

7.Testing & Validation / Limitations

7.1. Test Cases

Test Case	Description	Expected Result	Actual Result	Status
TC1	Voice command to turn on light	Light turns on	Light turned on successfully	✅ Pass
TC2	Voice command to turn off fan	Fan turns off	Fan turned off successfully	✅ Pass
TC3	Set reminder for specific time	Reminder stored in Firebase	Reminder triggered on time	✅ Pass
TC4	Unknown query (e.g., general question)	Assistant responds with fallback	OpenAI generated response played	✅ Pass
TC5	Weak Wi-Fi connection	Audio transmission may fail	Retry mechanism works, occasional lag	⚠️ Partial
TC6	Button-controlled recording	Starts/stops recording as expected	Working correctly	✅ Pass

7.2. Limitations

- Requires stable Wi-Fi: Inconsistent connections can delay communication with the server.

- Audio processing delay: Response time depends on network speed and external API processing.

- Dependency on external APIs (AssemblyAI, OpenAI, Google TTS): If these services are unavailable, functionality is limited.

- Limited offline capability: System cannot function without internet.

- No voice wake word support: Assistant activates only on button press instead of passive listening.

8.Conclusion & Future Work

8.1. Key Takeaways

Joona represents a successful implementation of an intelligent, voice-controlled IoT assistant powered by the ESP32-S3 microcontroller. The system effectively bridges embedded hardware with cloud-based AI services to offer a seamless, real-time user experience. By leveraging I2S-enabled audio streaming, HTTP communication, and relay control, Joona captures voice commands, interprets their intent using AssemblyAI

and OpenAI APIs, and delivers appropriate responses through Google TTS — all while interacting with physical devices like lights and fans.

Moreover, the integration of Firebase Firestore and APScheduler enabled Joona to support real-time reminders and scheduled task execution, enhancing its practical use for smart home or automation environments. The system also included manual button-based triggering, which provides additional control flexibility in scenarios where hands-free operation is not optimal.

This project showcases how low-cost hardware like the ESP32-S3 can be combined with modern AI tools to build powerful and responsive conversational systems, setting a foundation for more advanced human-machine interfaces.

- Real-time voice interaction enabled via I2S microphone and speaker modules with ESP32-S3.

- Cloud integration with AssemblyAI (STT), OpenAI (intent fallback), and Google TTS (voice feedback).

- Reliable device control through GPIO relays for light and fan.

- Firebase-based reminder system with time-based triggers using APScheduler.

- Balanced use of hardware and software for robust and scalable IoT deployment.

8.2. Future Improvements (e.g., offline wake word, OLED display)

- Offline Capability: Integrate lightweight ML models (e.g., TensorFlow Lite) for on-device intent classification, enabling partial functionality without internet access.

- Wake-Word Detection: Implement a passive wake-word listener ("Hey Joona") to start recording without a physical button.

- Multi-Device Expansion: Enable multi-node communication via MQTT or Firebase, allowing Joona to manage multiple rooms or device clusters.

- User Authentication: Add speaker recognition or simple user profiles to personalize responses and restrict specific actions.

- Mobile App Integration: Develop a companion app for remote device control, push notifications for reminders, and audio command history.

- Enhanced Audio Processing: Use digital filters or pre-processing (e.g., noise suppression, automatic gain control) to improve accuracy of speech recognition.

- Energy Optimization: Implement sleep modes and efficient task scheduling to make the system suitable for battery-powered deployment.

9. References

- AssemblyAI – Speech-to-Text API
https://www.assemblyai.com/

- OpenAI API – GPT for Natural Language Understanding
https://platform.openai.com/

- Google Cloud Text-to-Speech API
https://cloud.google.com/text-to-speech

- Firebase Firestore – Real-time Database
https://firebase.google.com/docs/firestore

- ESP32-S3 Technical Reference Manual
https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf

- Arduino IDE – Programming Platform
https://www.arduino.cc/en/software

- ESP-IDF / ESP32 I2S Configuration Docs
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2s.html

- Microphone (INMP441) Datasheet & Setup
https://components101.com/microphones/inmp441-mems-microphone

- MAX98357A I2S Audio Amplifier Module Datasheet
https://learn.adafruit.com/adafruit-max98357-i2s-class-d-mono-amp

- APScheduler – Python Job Scheduler
https://apscheduler.readthedocs.io/en/stable/

- Python Flask Framework
https://flask.palletsprojects.com/

- YouTube Tutorial – Voice Controlled ESP32 Project
https://www.youtube.com/watch?v=3C5KxZrAJeU

- GitHub Repository – ESP32 I2S Microphone & Audio Projects
https://github.com/espressif/esp-adf

10.Links

https://github.com/Marriam-Sajjad11/IOT-LABs

https://github.com/fatima9093/IOT_lab

https://github.com/mry3am/IOT-LABS.git

https://github.com/fatimachaudhary-netizen/IOT-PROJECT.git