EE782 – Assignment 2

Voice-Controlled Guard Mode System

Student Name: Jishnuchandra Arepalli

Course: EE782 – Machine Learning Applications

Abstract

This project demonstrates a simple yet functional **voice-activated control system** capable of recognizing spoken commands and performing corresponding actions.
The system enables the user to activate or deactivate a "Guard Mode" using only their voice.
By integrating **speech-to-text conversion**, **browser-based audio capture**, and **state persistence**, the project illustrates how speech interfaces can be implemented within the Python and Google Colab ecosystem.
The work highlights the fusion of audio signal processing, natural-language interpretation, and file-based memory for a seamless, interactive experience.

---

1. Introduction

Voice-controlled systems are rapidly becoming central to modern human–computer interaction. From mobile assistants to smart-home devices, natural speech has emerged as the most intuitive way to issue commands.
This project explores those principles by developing a small-scale **speech-driven automation module** that listens for user commands such as *"active"* or *"shut down."*
Upon recognizing a valid phrase, the system toggles a persistent state variable known as **Guard Mode**, simulating the activation or deactivation of a security mechanism.

The notebook-based environment was chosen to highlight how interactive tools such as **Google Colab** can combine JavaScript (for microphone access) with Python (for backend logic).
This project provides both educational value and a foundation for larger applications in embedded voice control, smart devices, and AI-based monitoring systems.

---

2. Objectives

The main goals of this assignment were to:

1. Capture real-time speech input through a web browser.
2. Convert the recorded audio into text using Python's SpeechRecognition library.
3. Detect key words to determine whether to enable or disable Guard Mode.
4. Save the system's state to ensure persistence across notebook runs.

5. Demonstrate a minimal yet complete speech-controlled workflow suitable for further enhancement.

---

3. System Requirements

The system operates entirely within a Google Colab notebook and requires:

- **Python 3.10+**
- **SpeechRecognition** – speech-to-text processing
- **PyAudio** – audio-stream interface
- **FFmpeg** – file-format conversion (WebM → WAV)
- **IPython.display & google.colab.output** – to embed JavaScript for audio recording
- **Base64 and OS** modules for data handling and persistence

The setup section installs dependencies using the following commands:

!apt-get install -y portaudio19-dev

!pip install SpeechRecognition pyaudio

---

4. System Design and Implementation

4.1 Architecture Overview

The project consists of four main modules:

1. **Audio Recording Interface** – JavaScript code captures microphone input in the browser.
2. **Audio Conversion** – FFmpeg converts the recorded WebM file to WAV format.
3. **Speech Recognition** – Google's SpeechRecognition API processes the audio.
4. **State Management** – a small text file (guard_state.txt) stores whether Guard Mode is ON or OFF.

The system flow is linear and event-driven:
User → Record → Recognize → Evaluate Command → Save State → Feedback.

4.2 Audio Recording via JavaScript

Colab's front-end JavaScript function uses the **MediaRecorder API** to capture audio for a specified duration.
It returns a base64-encoded string, which Python decodes and stores as audio.webm.
This approach avoids the need for direct microphone access from Python, which is not supported in hosted notebook environments.

4.3 Audio Processing and Recognition

Once the .webm file is captured, FFmpeg converts it into .wav, suitable for the SpeechRecognition library:

```
!ffmpeg -i audio.webm -ac 1 -f wav audio.wav -y
```

The recognizer reads the audio file and uses Google's API to produce a text transcript.
This text is then converted to lowercase and matched against pre-defined trigger words.

4.4 Command Evaluation Logic

```
if text == "active":
    guard_mode = True
elif text == "shut down":
    guard_mode = False
else:
    print("Command not recognized")
```

A successful match toggles the system state and prints feedback to the user.
This logic can easily be extended to include more complex command sets.

4.5 Persistence Mechanism

The state is written to guard_state.txt after each command:

```
with open("guard_state.txt", "w") as f:
    f.write(str(guard_mode))
```

During initialization, the program reads this file to restore the last known mode.
This ensures continuity between sessions—mimicking a persistent configuration memory found in embedded systems.

---

5. Experimental Results

Sample runs of the system produced the following outputs:

Initial state: Guard mode is OFF

Recording for 3 seconds... Please speak now.

Heard: 'active'

STATUS: Guard mode activated.

Initial state: Guard mode is ON

Recording for 3 seconds... Please speak now.

Heard: 'shut down'

STATUS: Guard mode deactivated.

Observations

- The recognition accuracy was high for clear, single-word commands.
- Background noise occasionally produced partial misinterpretations.
- State persistence worked flawlessly between executions.
- The response time (record → recognize → output) averaged 3–4 seconds, acceptable for small applications.

---

6. Discussion

This project highlights how simple, low-level speech interaction can be achieved using freely available tools.
Although limited to two commands, the architecture supports easy extension.
Because the speech recognition relies on Google's online service, a stable internet connection is essential.
The modular design—separating recording, recognition, and logic—makes it adaptable to offline models such as **Vosk** or **Whisper** for improved privacy and latency.

The combination of JavaScript and Python demonstrates cross-language communication within a notebook, an important skill for integrating front-end and back-end functionalities in research prototypes.

---

7. Conclusion

The **Voice-Controlled Guard Mode System** achieves its intended objective of enabling hands-free control through spoken commands.
It successfully integrates **speech recognition**, **browser audio capture**, and **persistent state management** into a cohesive, interactive application.
This exercise provides practical exposure to natural-language interfaces and shows how such mechanisms can be incorporated into broader AI-driven automation systems.
Future work can extend this foundation into a multi-command voice assistant capable of executing complex sequences or controlling IoT devices.

---

8. Future Enhancements

1. **Add more commands** (e.g., "status," "restart," "record," "sleep mode").
2. **Integrate text-to-speech (TTS)** responses using pyttsx3 or gTTS.
3. **Add noise filtering** to improve robustness in real environments.
4. **Introduce a GUI** with libraries like Streamlit or Gradio.
5. **Enable offline operation** through local speech models.
6. **Connect to IoT devices** for real-world automation tasks.

9. GitHub Repository Link :

https://github.com/ajc1495/AI_GUARD_ROOM/tree/main