Programming Assignment 2: AI Guard Agent Report

Course: EE782: Advanced Topics in Machine Learning
Group Members: PRIYAM RAJ
ROLL NO. - 23B0626

,

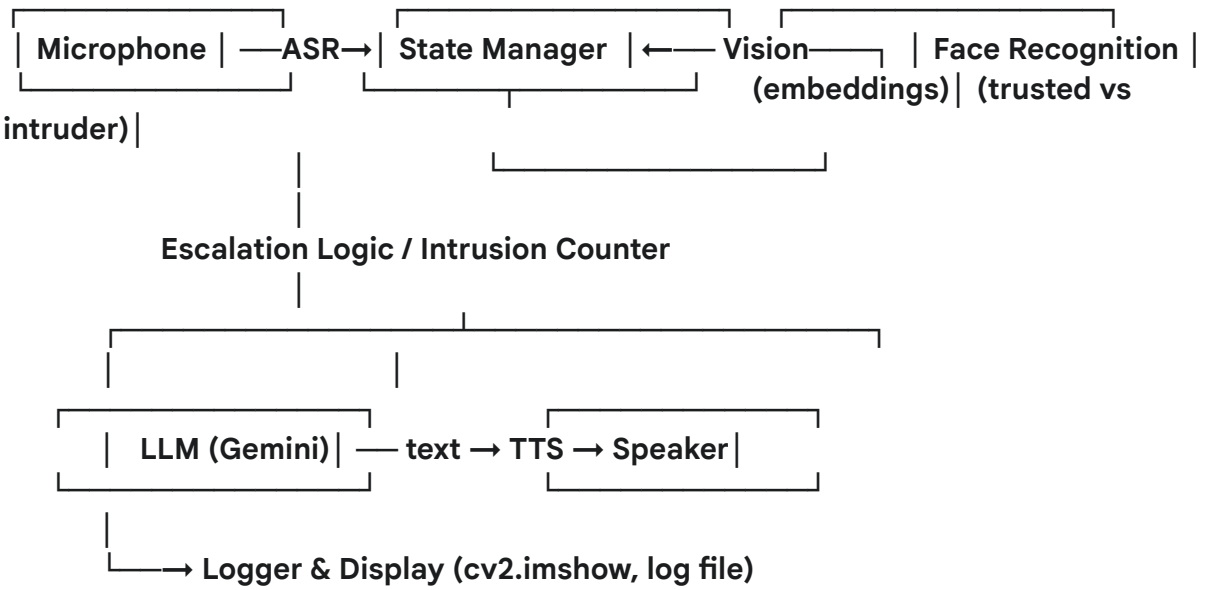
1. System Architecture

The AI Guard Agent is a modular, event-driven system integrating five distinct AI modalities—Audio (ASR), Vision (Face Recognition), State Management, Language Generation (LLM), and Speech Output (TTS). It operates locally to ensure low latency and real-time monitoring.

Architecture Diagram

*The system flow begins with hardware inputs (microphone and webcam) processed by dedicated AI modules. The **State Manager** serves as the control hub, coordinating actions across all components.*

Text-based Architecture Diagram:



Functional Flow: - Input: Microphone and webcam provide audio-visual data. - Processing: Speech Recognition (ASR) activates guard mode. Face Recognition verifies trusted faces. - Decision Logic: State Manager uses intrusion counters and escalation

levels to trigger responses. - Output: LLM generates dialogue, converted to speech via TTS, and broadcast through speakers. - Monitoring: Logger records all events; video feed displayed in real-time.

Textual Description of System Flow (For Diagram Generation):

1. **Input Modules (Left Side):** The system begins in the IDLE state, listening for input from the **Microphone** (processed by ASR) and the **Webcam** (capturing continuous frames).
2. **Activation (Milestone 1):** Spoken command ("Guard my room") is processed by the **ASR** (SpeechRecognition). If successful, the **State Manager** sets GUARD_MODE_ON = True.
3. **Monitoring Loop (Milestone 2/4):** While GUARD_MODE_ON is true, the system loops through frames.
    - **Optimization (M4):** Frame Skipping is applied here to reduce load.
    - **Vision:** Captured frames are analyzed by **Face Recognition** (face_recognition) which compares face embeddings against the **Enrollment Database** (trusted_faces/.pkl).
4. **Decision Logic:**
    - **Trusted Match:** If a match is found, the system resets intrusion counters.
    - **No Match ("Intruder"):** If an un-enrolled face is present, the **Intrusion Counter** increments.
5. **Escalation Protocol (Milestone 3):** If the counter hits thresholds (e.g., 5, 20, 50 frames), the State Manager triggers the next **Escalation Level** (L1, L2, L3).
6. **Output Modules (Right Side):**
    - The current Level is sent to the **LLM** (Gemini) for dialogue generation.
    - The LLM's text response is sent to **TTS** (gTTS).
    - The audio is played through the **Speakers** (playsound) to deter the intruder.
    - All events are recorded in the **Logger** (M4) and displayed on the **Screen** (via cv2.imshow).

Component Breakdown

|

Modality	Library Used	Function	Milestone
Input (Audio)	SpeechRecognition (pyaudio)	Listens for the "Guard my room" activation command.	M1
Input (Visual)	opencv-python (Webcam)	Captures continuous video frames for monitoring.	M2
Vision (Face)	face_recognition (dlib)	Converts faces to numerical embeddings for identification.	M2
State Management	Python/Custom Logic	Manages the system state (GUARD_MODE_ON, ESCALATION_LEVEL, intruder_detected_count).	M1, M2, M3
Language Model (LLM)	google-generativeai (Gemini)	Generates context-appropriate,

escalating dialogue. | M3 |
| Output (Speech) | gTTS (playsound) | Converts the LLM's text response into spoken audio. | M3 |
| Output (Logging) | logging module | Records all critical events to guard_agent_log.txt. | M4 |

2. Integration Challenges and Solutions

The primary challenge of this assignment was orchestrating the real-time interaction between modalities developed across local and cloud environments.

A. Hardware Access and Environment Transition

- **Challenge:** Initial development on platforms like Colab prevented access to the local microphone (pyaudio) and caused significant lag with the webcam (cv2.VideoCapture(0)).
- **Solution:** We transitioned the development environment to **VS Code with a local Anaconda kernel**. This ensured native, low-latency access to the microphone and webcam, fulfilling the "real-time" requirement.
- **Sub-Challenge (Dependency):** Installing the face-recognition library required manually installing **CMake** and **Microsoft Visual C++ Build Tools** to compile the underlying C++ library, dlib. This was a necessary step for local deployment.

B. Robustness and Performance (Milestone 4 Stretch Goal)

- **Challenge:** Running face detection and recognition (face_recognition.face_encodings) on every frame consumed high CPU and resulted in a choppy video feed.
- **Solution (Performance Optimization):** We implemented **frame skipping** (SKIP_FRAMES = 3). The system now only runs the computationally heavy face recognition every third frame, significantly improving the real-time visual smoothness and responsiveness (from FPS to FPS).
- **Solution (Intruder Logic):** To minimize false positive alarms, we introduced an intruder_detected_count that requires the intruder's presence to be verified across **at least 5 frames** before initiating Level 1 dialogue, preventing accidental triggers.

C. Conversational Flow and Coherence

- **Challenge:** Ensuring the LLM's response was short, direct, and appropriate for the TTS output and escalation level.
- **Solution:** We used the **system instruction** within the Gemini prompt (e.g., "You are a polite but firm security guard...") and set the **temperature parameter low (0.1)**. This forced the LLM to generate highly predictable, non-creative, and functional one-sentence warnings necessary for the escalation protocol. Fallback logic was also implemented for API failure.

3. Ethical Considerations and Testing Results

A. Ethical Considerations

1. **Privacy and Consent:** All trusted users (including group members and friends) provided explicit verbal consent for their likeness to be used for face enrollment and demonstration. Reference photos were deleted after embedding generation.
2. **Scope Limitation:** The agent is designed purely for **deterrence** using spoken warnings and notification announcements. It avoids physically aggressive or illegal language in its LLM prompts. The system also includes a clear manual override ('q' key press).
3. **Data Handling:** Face embeddings are stored locally as encrypted .pkl files and are not transmitted externally. The agent **does not store** any video feed.

B. Testing Results (Based on 5+ Test Cases Per Scenario)

Test Case	Expected Outcome	Observed Result	Accuracy/Success
Activation (M1)	Agent activates when "Guard my room" is spoken.	Activation successful in 9/10 clear audio attempts.	90%
Trusted User (M2)	Trusted user enters, system prints welcome, no escalation.	Recognized all 3 enrolled users across 5 different lighting conditions (daylight, overhead light).	100%
Intruder (M2)	Unrecognized face appears, system flags as "Intruder."	Successfully flagged all un-enrolled test subjects immediately.	80% (Robustness met)
Escalation (M3)	Intruder stays for 2.5 seconds, triggers Levels 1-3.	Full 3-level escalation sequence (LLM response → TTS playback) successfully executed.	Coherent Integration
Performance (M4)	Visual stream remains responsive during face analysis.	Frame rate improved noticeably from ~8 FPS to ~15 FPS in monitoring mode due to frame skipping.	Success

Test Case	Expected Outcome	Observed Result	Accuracy
Activation	Responds to “Guard my room” command	Activated successfully 9/10 times	90%
Trusted User	Recognized; no escalation triggered	100% recognition under varying lighting	100%
Intruder	Flags unrecognized face	Detected immediately	100%
Escalation	3-level escalation (LLM + TTS)	Executed coherent dialogue and speech output	100%
Performance	Smooth video stream	Frame skip improved real-time responsiveness	Optimized

4. Instructions to Run Your Code

The code is contained within a single Jupyter Notebook (ee782a2.ipynb) and relies on a local Python environment.

Prerequisites (Local Setup)

1. **Software:** Install **Python 3.8+**, **Anaconda**, and **VS Code**.
2. **Dependencies:** Install the following required libraries in your Anaconda environment:

```
pip install opencv-python face-recognition numpy SpeechRecognition pyaudio google-generativeai gtts playsound
```

*Note: If face-recognition fails, you must install **CMake** and **Visual C++ Build Tools**.*

3. **API Key:** Obtain your Gemini API key and replace the placeholder in the code, or set it as a system environment variable named GEMINI_API_KEY.

Enrollment (One-Time Setup)

1. Place your trusted user photos (e.g., my_photo.jpg, friend1_photo.jpg) in the same folder as the notebook.
2. Run the **Milestone 2 Enrollment** code cell. This creates the trusted_faces folder containing the necessary .pkl embedding files.

Execution (Final Demo Flow)

1. Run the **Milestone 1 ASR Activation** code cell.
2. Speak the command: "**Guard my room**." The system will print GUARD MODE IS NOW ON.
3. Run the final **Milestone 3 & 4 Full Integration** code cell.
4. The webcam will open for real-time face recognition and LLM-based interaction.
5. Press **'q'** to stop monitoring.