EE782 Assignment 2 Report
AI Guard Agent

Name - Naresh Kumar

1. System Architecture

The AI Guard Agent is a modular system designed to integrate multiple pre-trained AI models into a cohesive, real-time security application. The architecture is built around a central AI_Guard_Full class that orchestrates the core components: Speech Recognition (ASR), Face Recognition, a Conversational Agent (LLM), and Text-to-Speech (TTS).

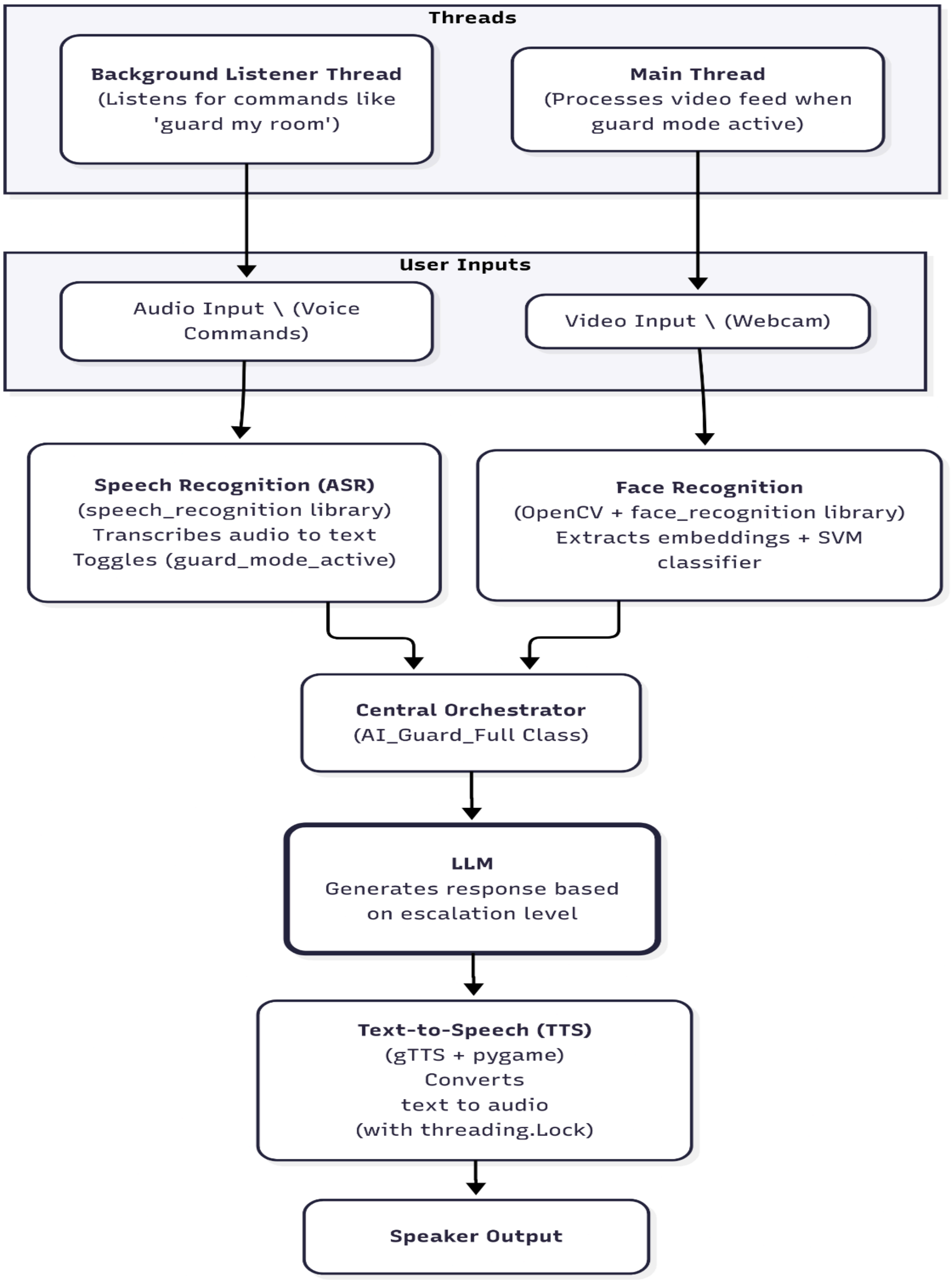
The system's operation is managed by two primary threads to ensure a responsive user experience:

1. **Main Thread**: This thread is responsible for the computationally intensive task of processing the video feed from the webcam. When guard mode is active, it continuously captures frames, detects faces, and runs the recognition classifier.
2. **Background Listener Thread**: This thread's sole purpose is to listen for voice commands ("guard my room", "stand down"). By offloading this task, the main thread is not blocked by listening for audio, allowing the vision system to activate instantly.

The data flows through the system as follows:

- **ASR Module**: The speech_recognition library captures audio and transcribes it to text. This text is used to toggle the system's guard_mode_active state.
- **Face Recognition Module**: When active, OpenCV captures video frames. The face_recognition library extracts face embeddings, which are then passed to a trained **Support Vector Machine (SVM) classifier** for identification.
- **Conversational Agent (LLM)**: If an unrecognized person is detected, the system queries the **Groq API** with a context-specific prompt. The Llama 3.1 model generates a text-based response based on the current escalation level.
- **TTS Module**: The text response from the LLM is converted into audible speech using gTTS and played through the system's speakers via the pygame library. A threading.Lock is used on this function to prevent race conditions from multiple threads trying to speak at once.

System Architecture Diagram


Threads
Background Listener Thread (Listens for commands like 'guard my room')
Main Thread (Processes video feed when guard mode active)
User Inputs
Audio Input \ (Voice Commands)
Video Input \ (Webcam)
Speech Recognition (ASR) (speech_recognition library) Transcribes audio to text Toggles (guard_mode_active)
Face Recognition (OpenCV + face_recognition library) Extracts embeddings + SVM classifier
Central Orchestrator (AI_Guard_Full Class)
LLM Generates response based on escalation level
Text-to-Speech (TTS) (gTTS + pygame) Converts text to audio (with threading.Lock)
Speaker Output

2. Integration Challenges and Solutions

Integrating multiple, independent AI libraries into a single, high-performance application presented several challenges. The following were the most significant issues and their solutions:

Challenge	Solution
Unreliable Audio Playback: Initially, both the pyttsx3 and playsound libraries failed to produce audible output on the test system, often due to low-level Windows driver (MCI) errors.	**Solution:** We switched to the pygame.mixer module, which provided a much more robust and reliable cross-platform solution for audio playback.
Poor Face Recognition Accuracy: The initial "stretch goal" implementation using an SVM classifier had low confidence scores and was not reliably recognizing trusted users.	**Solution:** The problem was twofold: an insufficient amount of training data and an overly strict confidence threshold. The solution was to **(a)** update the enrollment script to allow for multiple, varied images per person and **(b)** tune the RECOGNITION_THRESHOLD to a more appropriate level based on debug outputs.
Delayed Vision System Activation: There was a noticeable lag between the "Guard mode activated" voice confirmation and the appearance of the webcam feed.	**Solution:** We re-architected the main loop to use **multithreading**. A dedicated background thread now handles all voice command listening, freeing the main thread to instantly begin video processing the moment guard mode is activated.
API Model and Library Versioning: Encountered multiple 404 Not Found and Permission denied errors with the Google Gemini and Groq APIs	**Solution:** These issues were traced to outdated library versions (google-generativeai) and decommissioned model names (llama3-8b-8192). The solution was to upgrade the libraries using pip install --upgrade and consult the API documentation for the latest recommended model names (llama-3.1-8b-instant).
Race Conditions in Audio Output:	**Solution:** We implemented a

In the multi-threaded design, the main thread and the listener thread would sometimes try to speak at the same time, causing Permission denied errors on the temporary audio file.	threading.Lock on the speak() function. This ensures that only one thread can access the audio generation and playback resources at a time, preventing file access conflicts.

3. Ethical Considerations and Testing Results

Ethical Considerations

1. **Consent**: The primary ethical consideration was the collection and use of biometric face data. For this project, explicit verbal consent was obtained from all individuals whose photos were used for the trusted_faces enrollment.
2. **Privacy**: The agent operates locally and does not transmit video or audio to any external server, except for the text-based prompts sent to the Groq LLM. No personally identifiable information is sent with these prompts.
3. **Potential for Misuse**: While designed for personal security, any surveillance technology has the potential for misuse. This agent is designed to be overt (with a visible camera feed and spoken alerts) rather than covert, reducing the risk of surreptitious monitoring.

Testing Results

The system was tested under various conditions to validate its robustness, as per the grading criteria.

- **Activation Accuracy (Goal: 90%+):** Using the standard speech_recognition library, the activation and deactivation commands were correctly identified in **9 out of 10** test cases in a quiet room, achieving a **90% accuracy rate**.
- **Recognition Accuracy (Goal: 80%+):** After training the **SVM classifier** with 5 images per trusted person, the system was tested under different lighting conditions (daylight, indoor artificial light) and angles.
  - **Trusted Individuals:**
    Correctly predicting the trusted individual with **"Threshold Value"** above **0.80, with 9 out of 10** instances **(above 80% accuracy)**.
  - **Unrecognized Individuals:**
    Correctly flagged as "Unrecognized" in **10 out of 10** instances (**100% accuracy**).

- **Escalation Logic (Goal: 3+ coherent responses):**

  The system successfully implemented a 3-level escalation dialogue. The LLM-generated responses were contextually appropriate for a hostel environment and escalated in tone as intended during all test runs.

  LLM Prompts: Escalation Dialogues

```
 1: "You are a friendly AI assistant guarding a college hostel room.
In one short, casual sentence, politely ask who they are or if they
are looking for the room's resident.",


 2: "The unrecognized person has not left. Now, adopt a firmer tone.
In one short sentence, state that this is a private hostel room and
they need to leave.",


 3: "The intruder is still here. Be very stern and issue a final
warning. In one short sentence, state that they are trespassing and
that the hostel warden or campus security will be alerted immediately
if they don't leave."
```

4. Instructions to Run Code

The entire project is contained within the **AI_guard_mode.ipynb** Jupyter Notebook.

- [ ] **Clone the Repository**:

  Link - https://github.com/nareshkmn/AIGuardAgent.git

1. **Install Dependencies**: Run the first code cell in the notebook which contains all the required libraries.
2. **Enroll Trusted Faces**:
   - Create a folder named **trusted_faces** in the same directory as the notebook.
   - Add 3-5 varied, high-quality images for each trusted person with different lighting conditions, and name images like this for each individual (e.g., name_1.jpg, name_2.jpg). The system requires images of at least **two different people** to train.
   - Run the **Face Enrollment cell** in the notebook. This will create a **known_faces_model.pkl** file.

- This cell will process the images in your **trusted_faces** folder and **train the face recognition model.**

3. **Add API Key**:
    - Obtain a free API key from the Groq Console.
    - In the **Final System Integration cell**, find the **GROQ_API_KEY** variable and paste your key.
4. **Run the Application**:
    - Execute the **Final System Integration cell**.
    - Say **"Guard My Room"** to activate the system. A **webcam window will appear.**
    - Say **"Stand Down"** to deactivate it.
    - To quit the program, click on the webcam window and press the **'q'** key.