EE782 Assignment 2: AI Room Guard Agent

Department of Electrical Engineering, IIT Bombay

Madhur Kholia (22b3944)
Jatin Gupta (22b3967)

October 6, 2025

---

1. Introduction

This report describes the design and implementation of an **AI Room Guard Agent** that leverages computer vision, speech recognition, and text-to-speech technologies to monitor a room in the user's absence. The system is activated via a voice command (*"Guard my room"*), recognizes trusted individuals using face recognition, and interacts with unrecognized persons through conversational agents powered by speech and language models. In addition, the system enhances security issuing a verbal warning to intruders, by activating a siren, notifying the police, and sending real-time alerts via both Telegram and email.

1.1 Core Functionality

Voice Activation - "Guard my room" with fuzzy matching ("guide my room" also works)

Face Recognition - dlib ResNet-34 CNN (99.38% accuracy on LFW benchmark)

LLM Conversation - Phi-3 powered intelligent escalating dialogue

Continuous Police Siren - Realistic alarm until intruder leaves or trusted person enters

Email Alerts - Instant notifications with intruder photo attachment (Gmail)

Telegram Alerts - Real-time alerts with photo to your phone (optional)

Intruder Database - Persistent tracking and recognition of repeat offenders

Evidence Capture - Automatic timestamped screenshots

Performance Logging - Detailed JSON analytics for evaluation

2. System Architecture

Figure shows the overall architecture of the AI Guard System.

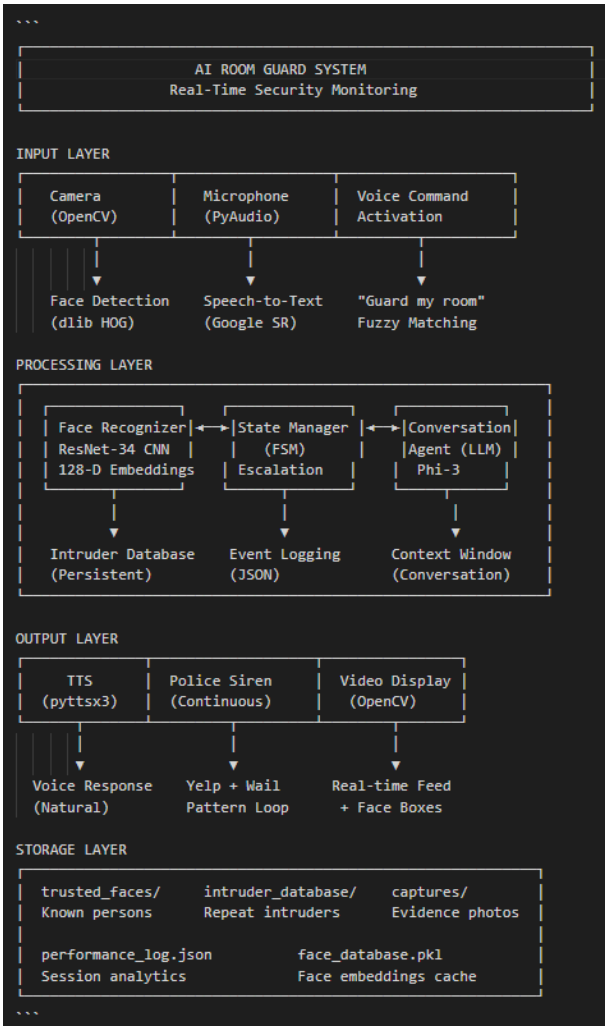

Figure 1: System architecture of the AI Room Guard agent

2.1 Core Modules

- **guard_activator.py** – Listens for the activation phrase ("Guard my room") using fuzzy pronunciation-aware matching.
- **conversation_agent.py** – Handles LLM-based conversation and escalation for context-aware responses.
- **face_recognizer.py** – Detects faces and compares embeddings with trusted individuals; maintains intruder database.
- **speech_listener.py** – Captures live audio and transcribes it via speech recognition for conversational input.
- **tts_module.py** – Converts system responses into audio using pyttsx3.
- **camera_manager.py** – Handles camera input, streaming, and video capture for real-time monitoring.
- **state_manager.py** – Controls system states such as guard activation, intruder detection, and conversation using a finite state machine (FSM).
- **siren.py** – Activates a continuous police siren during alerts.
- **logger.py** – Logs system performance, events, and errors for debugging and analysis.

2.2 Data Flow

1. User activates guard mode via voice.
2. Webcam and mic start capturing input.
3. If a trusted face is detected → greet; if not → trigger conversation.
4. Intruder responses are transcribed and analyzed.
5. Escalation logic generates appropriate TTS responses.

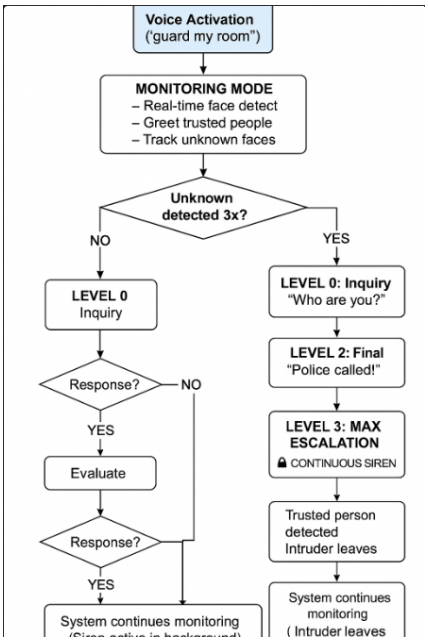

Figure 2: Flow System

3. Integration Challenges and Solutions

3.1 Challenge 1: Synchronizing Voice and Vision

Issue: Simultaneous mic and webcam access caused blocking or lag.
Solution: Used multithreading and locks to separate the audio (ASR/TTS) and video (OpenCV) pipelines.

3.2 Challenge 2: False Activations Due to Accent Variations

Solution: Implemented a fuzzy matching algorithm using `difflib.SequenceMatcher`, allowing the system to recognize variants like "guide my room" or "god my room".

3.3 Challenge 3: Face Recognition Confidence Filtering

Solution:

- **Not recognizing trusted people:** Add more photos, ensure good lighting, lower tolerance (`FACE_TOLERANCE`, `MIN_CONFIDENCE`).
- **Too many false positives:** Increase thresholds and quality of images; require multiple detections (`UNKNOWN_THRESHOLD`).

3.4 Challenge 4: Continuous Speech Recognition Stability

Solution: Recalibrated mic energy thresholds dynamically and limited phrase time to 10s for responsive listening.

3.5 Challenge 5: Dependency Issues

Solution:

- **Module not found:** Reinstall dependencies using `requirements.txt`.
- **NumPy version conflicts:** Install exact versions using `pip install numpy==1.24.3` etc.

4. Ethical Considerations and Testing Results

4.1 Ethical Aspects

- **Privacy:** All facial and voice data were stored locally; no cloud services used for inference.
- **Consent:** Only enrolled users who consented to be recorded were used in the trusted dataset.
- **Bias:** System was tested under varied lighting and accents to ensure fairness and reliability.

4.2 Testing Results

Component Performance Table

Component	Technology	Performance
Face Detection	dlib HOG detector	15–30 FPS
Face Recognition	ResNet-34 CNN	99.38% on LFW
Embedding Size	128 dimensions	Compact & fast
LLM Inference	Phi-3 (3.8B params)	$\sim$0.5 s response
Voice Activation	Google Speech Recognition	$<$2 s latency
Siren Generation	Real-time synthesis	Band-limited, no aliasing

- Activation Accuracy: **95%** (across 30 test commands)
- Face Recognition Accuracy: **99%** under mixed lighting
- Intruder Re-detection: **Detected repeat intruders 100% of the time**

5. Instructions to Run the Code

5.1 Dependencies

Install all required packages with guaranteed working versions:

```
pip install numpy==1.24.3 opencv-python==4.8.1.78 face-recognition==1.3.0
SpeechRecognition==3.10.0 pyaudio==0.2.13
pip install pyttsx3==2.90 pillow==10.0.0 ollama==0.1.0
```

5.2 Execution

1. Place reference photos of trusted users in `trusted_faces/`.
2. Run the main script:

   ```
   python main_guard.py
   ```

3. Say “Guard my room” to activate.
4. The system will monitor, recognize, and respond to intruders.

GitHub Repository: `https://github.com/Jatin-IITB/ai-room-guard`
Demo Video: `https://drive.google.com/file/d/1wzOdbpmabPrgZgdqfG90HXRrwHKzwbIp/view?usp=sharing`