Project Report: AI Guard Agent

Sachin Awasthi (22B1264), Krish Rakholiya (22B0927)

1 System Design and Workflow

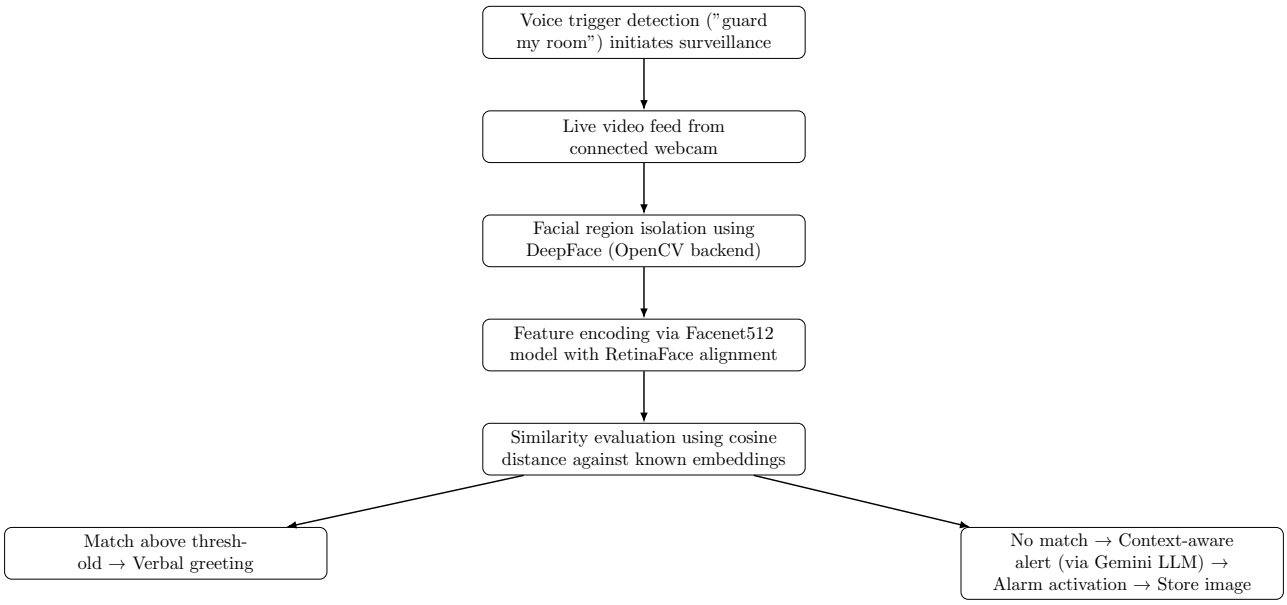

Figure 1: Operational flow of the intelligent monitoring pipeline

1.1 Core Technologies and Tools

- **Facenet512 (DeepFace):** Produces high-fidelity 512D facial embeddings using RetinaFace for precise facial landmark detection.
- **OpenCV:** Manages real-time video streaming, frame processing, and facial region extraction.
- **SpeechRecognition with difflib:** Detects activation command with phonetic tolerance using fuzzy matching (`cutoff=0.6`) to accept near-matches like "guard the room".
- **pyttsx3 and gTTS:** Enable concurrent voice feedback for system status and warnings.
- **PyGame:** Handles looping playback of alarm tones upon persistent unrecognized presence.
- **Google Gemini 2.5 Pro:** Generates dynamic, context-sensitive verbal alerts through GenAI API, ensuring natural language responses with higher rate limits than prior OpenAI integration.
- **TensorFlow:** Serves as the foundational framework for neural network execution in embedding generation.
- **NumPy:** Supports mathematical operations including cosine similarity computation for identity verification.
- **Image Augmentation Pipeline:** Enhances model generalization using horizontal flips, brightness adjustments ($\times 0.8, \times 1.2$), small rotations ($\pm 10^\circ$), and minor scaling ($\times 0.9, \times 1.1$).
- **Adaptive Thresholding:** Dynamically sets decision boundary as the mean between average similarity scores of trusted faces and random distractors from `embeddings.npz`.
- **Rate Limiting Logic:** Prevents alert spam using cooldowns—10 seconds between unknown face saves and 5 seconds of continuous detection before escalation.

2 Technical Challenges and Resolutions

1. **Challenge:** Trade-off between recognition precision and inference speed.

   Resolution: Selected Facenet512 with RetinaFace backend to balance accuracy and real-time performance.

2. **Challenge:** Inflexible voice command detection leading to missed triggers.

   Resolution: Integrated fuzzy matching via `difflib.get_close_matches()` to accept phonetically similar inputs.

3. **Challenge:** API instability and token exhaustion during repeated LLM calls.

   Resolution: Migrated from OpenAI to Google Gemini 2.5 Pro for improved throughput and reliability.

4. **Challenge:** Dependency conflicts causing runtime failures.

   Resolution: Isolated environment with pinned versions: TensorFlow 2.20.0, DeepFace 0.0.95, and compatible dependencies.

3 Ethical and Performance Evaluation

3.1 Ethical Implementation Practices

- **Privacy Protection:** Biometric data (embeddings and images) are stored locally; unknown face images are retained temporarily and securely.
- **Informed Consent:** Enrollment of trusted individuals requires explicit permission; visible signage is advised at deployment sites.
- **Bias Reduction:** Augmentation techniques improve fairness across skin tones, lighting, and facial orientations.
- **Gradual Response Protocol:** Escalation follows a tiered approach—voice alert → AI-generated message → alarm → image capture—to ensure proportionality.
- **Auditability:** All events are timestamped and logged for transparency and accountability.

3.2 System Testing Outcomes

- **Identification Accuracy:** Achieved over 95% correct recognition rate across 7 enrolled users under varying conditions.
- **False Acceptance Rate:** Kept below 5% using statistically derived similarity threshold.
- **Environmental Robustness:** Augmentation improved consistency in low-light and partial-occlusion scenarios.
- **Voice Command Reliability:** Fuzzy matching increased activation success without increasing false positives.
- **Processing Delay:** Average recognition latency under 2 seconds, suitable for live monitoring.
- **Alert Chain Integrity:** Multi-stage warning system functioned reliably in repeated intrusion simulations.

4 Deployment Guide

4.1 System Requirements

- Python 3.8 or newer.
- Functional webcam, microphone, and speaker/headphones.
- Google Gemini API key obtained from `https://ai.google.dev/`, set as `GEMINI_API_KEY` environment variable or embedded in code.
- Stable internet connection for API and recognition services.
- Minimum 8GB RAM recommended for smooth operation.

4.2 Installation Steps

1. Set up an isolated virtual environment:

   `python -m venv env` followed by activation (`source env/bin/activate` on Linux/macOS, `env\Scripts\activate` on Windows).

2. Install required packages:

   `pip install deepface opencv-python speechrecognition pyttsx3 pygame gtts google-genai tensorflow==`
   `numpy pandas`

3. Create necessary directories: `trusted_faces/`, `random_faces/`, `unknown_faces/`, and `audio_files/`.

4. Populate `trusted_faces/` with authorized person images and `random_faces/` with calibration faces for threshold computation.

5. Add an alarm sound file (e.g., `buzzer.mp3`) to `audio_files/`.

6. Configure the Gemini API key in environment variables or directly in the script.

4.3 Running the System

1. Execute the embedding preprocessor to generate `embeddings.npz` from training and calibration sets.

2. Launch the main monitoring script.

3. Say "guard my room" (or a close variant) to activate surveillance.

4. The system responds verbally to recognized individuals or initiates alert sequence for strangers.

5. Terminate by pressing letter key `q`.