EE782: Advanced Topics in Machine Learning

Programming Assignment 2: AI Guard Agent

Objective: Design and implement an AI guard agent that uses a laptop's webcam, microphone, screen, and speakers to monitor your hostel room in your absence. The agent activates on a spoken command (e.g., "Guard my room"), recognizes trusted individuals (you, your roommate, your friends who frequent your room) via face recognition, and engages in an escalating spoken conversation with unrecognized individuals to deter intrusion. This project emphasizes integration of pre-trained AI models (speech, vision, language) into a cohesive system, requiring thoughtful system design and robust logic. You will work in groups of 2 and use free resources like Google Colab or Kaggle, or personal laptops with Anaconda.

Assignment Scope: Integration vs. Training

This is primarily an **integration challenge**. You will use pre-trained models and focus on combining them into a working system. No heavy neural network training is required, but "enrollment" (e.g., storing reference photos and voice for face and voice recognition) is expected.

- **Face Recognition**:
  - Use existing libraries (e.g., face_recognition, DeepFace, MediaPipe).
  - "Training" means enrolling 1–2 trusted individuals by storing their face embeddings from reference photos (no neural net training). Seek permission to use photos.
- **Speech Recognition (ASR)**:
  - Use pre-trained models like Whisper or Google Speech Recognition.
  - No training required.
- **Text-to-Speech (TTS)**:
  - Use pre-trained models (e.g., gTTS, pyttsx3, Coqui TTS).
  - No training required.
- **Conversational Agent (LLM)**:
  - Use free-tier LLMs (e.g., Google Gemini, OpenAI GPT-4o mini, or open-source models like Llama-3 via Hugging Face).
  - Design the escalation logic and integration flow.

Optional Stretch Goals

For bonus credit, consider:

- Training a simple keyword spotter (e.g., for "guard mode") using the Google Speech Commands dataset.
- Fine-tuning a face classifier on embeddings of trusted vs. “untrusted” individuals.
- Collecting a small dataset of speech/face samples to personalize the system.

Technical Requirements

- **Programming Language**: Python 3.8+ (compatible with Colab/Kaggle).
- **Dependencies**: Use free, open-source libraries (e.g., OpenCV, face_recognition, Whisper). Install via !pip install in Colab or locally. Common libraries include:

    - mediapipe, face_recognition, deepface, opencv-python (vision).
    - openai-whisper, SpeechRecognition, pyaudio (ASR).
    - gtts, pyttsx3, TTS (TTS).
    - transformers, langchain, google-generativeai (LLM/agent).
    - pygame, playsound (audio playback).

- **Hardware**: Laptop webcam and microphone for real-time input. Test locally or use Colab with workarounds (e.g., ngrok for webcam access). Or make it work on pre-recorded videos.
- **Platforms**: Google Colab (free GPU/TPU) or Kaggle notebooks. Local testing is recommended for hardware access. Or make it work at least on pre-recorded videos.

Deliverables

- **Code**: A well-documented Python codebase in a GitHub repository.
- **Demo Video**: A 2-3 minute video showing the agent activating, recognizing trusted/untrusted individuals, and escalating with an intruder, followed by a 5 minute code walkthrough
- **Report**: A 2–3 page PDF (or notebook integrated markdown) explaining:

    - System architecture (diagram recommended).
    - Integration challenges and solutions.
    - Ethical considerations and testing results.
    - Instructions to run your code.

Suggested Milestones

To pace your work and enable systematic grading, follow these milestones (assuming a 3 week assignment):

Milestone 1: Activation and Basic Input

- Goal: Agent activates via speech command (e.g., "Guard my room").
- Tasks: Implement ASR for command detection; set up webcam/mic access; establish basic state management (guard mode on/off).
- Deliverable: Code snippet + short video of activation.
- Grading: Command recognition accuracy (90%+ on clear audio).

Milestone 2: Face Recognition and Trusted User Enrollment

- Goal: Detect faces and verify trusted users.
- Tasks: Integrate face detection/recognition; enroll 1–2 trusted faces via photos; implement logic to welcome trusted users or flag others.
- Deliverable: Demo script + explanation of embedding comparison.
- Grading: Accuracy on 5+ test cases (including lighting variations).

Milestone 3: Escalation Dialogue and Full Integration

- Goal: Handle intruders with escalating conversation.
- Tasks: Integrate LLM for dialogue (e.g., Level 1: "Who are you?"; Level 2: "Please leave"; Level 3: Alarm or stern warning); add TTS for spoken responses; test full flow.
- Deliverable: Complete code + architecture diagram + test video.
- Grading: Coherent integration + creative escalation (3+ response levels).

Optional Milestone 4: Polish and Stretch Goals

- Goal: Enhance robustness or add stretch features.
- Tasks: Optimize real-time performance; add error handling/logging; implement one stretch goal (e.g., keyword spotter).
- Deliverable: Updated repo + reflection in report.
- Grading: Bonus for extras + documentation clarity.

Grading Criteria (15 points; both partners will be graded the same way)

- **System Design & Integration (5 points)**: Does the agent seamlessly combine vision, speech, and language modalities?
- **Robustness of Logic (4 points)**: Does it correctly handle activation (90%+ accuracy), recognition (80%+ accuracy), and escalation (3+ coherent responses)?
- **Creativity in Interaction Design (3 points)**: Is the guard's behavior natural, engaging, or clever (e.g., polite but firm escalation)?
- **Documentation Clarity (3 points)**: Can others run your code and understand your architecture from the report?
- **Bonus (up to 1.5 points)**: Stretch goals or exceptional creativity/robustness.

Suggested Free Resources

- **Face Detection/Recognition**:
  - MediaPipe (!pip install mediapipe): Face detection/landmarks.
  - face_recognition (!pip install face_recognition): Embedding-based recognition.
  - DeepFace (!pip install deepface): Verification and emotion detection.
  - OpenCV (opencv-python): Webcam access and Haar cascades.
  - Dataset: Kaggle's "Labeled Faces in the Wild" or own photos (with consent).
- **Speech Recognition (ASR)**:
  - Whisper (!pip install -U openai-whisper): Accurate, open-source transcription.
  - SpeechRecognition (!pip install SpeechRecognition pyaudio): Google Web Speech API.
  - Vosk (!pip install vosk): Offline keyword spotting.
- **Text-to-Speech (TTS)**:
  - gTTS (!pip install gtts): Google's TTS for audio files.
  - pyttsx3 (!pip install pyttsx3): Offline system voices.
  - Coqui TTS (!pip install TTS): Natural speech via Hugging Face models.

- **Agentic AI/LLM**:
    - LangChain (!pip install langchain): For agent workflows.
    - Google Gemini (!pip install google-generativeai): Free API key via makersuite.google.com.
    - OpenAI GPT-4o mini (!pip install openai): Free credits for new users.
    - Hugging Face Transformers (!pip install transformers accelerate): Llama-3 or Mistral.
    - Groq API (groq.com): Fast inference for open models.
- **General Tools**:
    - PyAudio/SoundDevice (!pip install pyaudio): Mic input.
    - Pygame/Playsound (!pip install pygame playsound): Audio playback.
    - Streamlit/Gradio (!pip install streamlit gradio): Optional UI for demos.
    - Datasets: Kaggle's "Common Voice" for speech testing.

Notes

- **Hardware Challenges**: Webcam/mic access in Colab may require local runs or ngrok for tunneling. Test early on laptops.
- **Dependencies**: Check library compatibility (e.g., face_recognition needs cmake, dlib). Use Colab's pre-installed OpenCV.
- **Collaboration**: Divide tasks (e.g., one student on vision/speech, one on LLM/logic) but integrate together.
- **Support**: Use Stack Overflow, GitHub issues, or Hugging Face forums for troubleshooting.

Submission

Submit via Moodle with links to GitHub repo, a previously ran .ipynb and a link to a video demo by Oct 9:

- GitHub repo link with code and README.
- Demo video (2–3 min + 5 min, uploaded or linked).
- Report (PDF/Markdown within ipynb).

Good luck building your AI guard agent! Be creative, test thoroughly, and document clearly.