Assignment-2: Final Code

This snippet imports all the libraries needed for the AI Guard Agent's core functions, which are computer vision (cv2, face_recognition), speech input/output (speech_recognition, gTTS, pygame), and AI dialogue (google.generativeai). It also sets up utilities (os, numpy, logging, time) and initializes the audio system with pygame.mixer.init() for playing voice responses.

```
import cv2
import face_recognition
import speech_recognition as sr
from gtts import gTTS
from playsound import playsound
import google.generativeai as genai
import os
import numpy as np
import logging
import time
import pygame
import threading

pygame.mixer.init()
```

This snippet imports modules for sending email alerts, allowing the system to compose messages (MIMEText, MIMEImage, MIMEMultipart) and send them through SMTP when suspicious activity (like an unknown person detected) occurs.

```
import json
import smtplib
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart
```

This is to set up the default logging configuration for the Python logging module. After calling it, any logging calls made (logging.info(), logging.warning(), etc.) will follow the configuration defined here and all logging calls are also stored in the file guard_log.txt. This basically helps to track the code smoothly. Also, this snippet configures access to Google's Gemini API using an API key and initializes the Gemini 2.5 Flash generative model. It enables the AI guard agent to generate natural, context-aware spoken responses or dialogue during interactions with people detected in the room.

```
# Setup logging for robustness (stretch goal)
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %
(levelname)s - %(message)s',
                    handlers=[logging.FileHandler("guard_log.txt"),
logging.StreamHandler()])

# Configure Gemini API (replace with your key)
genai.configure(api_key="AIzaSyDhBUtOZ40t0FdwPOzG0XsUgDVC9PbLSQU")  #
```

```
Get from makersuite.google.com
model = genai.GenerativeModel('gemini-2.5-flash')  # Free, fast model
```

This function continuously listens to the microphone for speech, attempting up to three times for robustness. It uses Google's Speech Recognition to convert spoken audio into lowercase text and logs results or errors. If speech isn't detected or understood after all retries, it returns an empty string.

```
# Listen for speech (ASR with retry for robustness)
def listen_for_speech(timeout=5, retries=3):
    recognizer = sr.Recognizer()
    for attempt in range(retries):
        try:
            with sr.Microphone() as source:
                logging.info("Listening...")
                audio = recognizer.listen(source, timeout=timeout)
            text = recognizer.recognize_google(audio).lower()
            logging.info(f"Recognized: {text}")
            return text
        except sr.WaitTimeoutError:
            logging.warning("No speech detected.")
        except sr.UnknownValueError:
            logging.warning("Could not understand audio.")
        except Exception as e:
            logging.error(f"ASR error: {e}")
    return ""
```

This function converts a given text message into speech using Google Text-to-Speech (gTTS), plays the audio aloud via pygame, waits until playback finishes, then deletes the temporary audio file. It also logs the spoken message for tracking system interactions.

```
def speak(text):
    try:
        tts = gTTS(text=text, lang='en')
        tts.save('response.mp3')
        pygame.mixer.music.load('response.mp3')
        pygame.mixer.music.play()
        while pygame.mixer.music.get_busy(): # Wait until playback
finishes
            pygame.time.Clock().tick(50)
        pygame.mixer.music.unload() # Control frame rate
        time.sleep(1)  # Extra buffer to ensure playback ends
        os.remove('response.mp3')
        logging.info(f"Spoke: {text}")
    except Exception as e:
        pass
```

This function runs the speak() function in a separate background thread, allowing the AI guard to speak while continuing other tasks (like camera or mic monitoring) without freezing the main program.

```
def speak_async(text):
    threading.Thread(target=speak, args=(text,), daemon=True).start()
```

This function scans a given folder named 'trusted_faces', containing subfolders for each trusted person, extracts facial embeddings from their images using face_recognition, and stores them in a dictionary. It then saves these embeddings to a .npy file for later use and logs the process, enabling the agent to recognize trusted individuals.

```
# Enroll trusted faces from folder
def enroll_trusted_faces(folder_path):
    trusted_embeddings = {}
    try:
        for person in os.listdir(folder_path):
            person_path = os.path.join(folder_path, person)
            if os.path.isdir(person_path):
                trusted_embeddings[person] = []
                for img_file in os.listdir(person_path):
                    img_path = os.path.join(person_path, img_file)
                    image = face_recognition.load_image_file(img_path)
                    encodings = face_recognition.face_encodings(image)
                    if encodings:

trusted_embeddings[person].append(encodings[0])
                    else:
                        logging.warning(f"No face found in
{img_path}")
        np.save('trusted_embeddings.npy', trusted_embeddings)  # Save
for reuse
        logging.info("Enrollment complete.")
        return trusted_embeddings
    except Exception as e:
        logging.error(f"Enrollment error: {e}")
        return {}
```

This function checks whether a detected face matches any trusted person by comparing its encoding with stored embeddings using Euclidean distance. If the minimum distance is below the set tolerance (0.4), it returns True along with the person's name; otherwise, it returns False.

```
# Check if trusted
def is_trusted(face_encoding, trusted_embeddings, tolerance=0.4):
    for person, embeds in trusted_embeddings.items():
        if embeds:
            distances = face_recognition.face_distance(embeds,
face_encoding)
            if np.min(distances) < tolerance:
```

```
                return True, person
    return False, None
```

This function uses the Gemini LLM to generate a short, polite-but-firm response to a potential intruder. It includes the current escalation level in the prompt, sends it to the model, and returns the generated text; if an error occurs, it returns a default warning message.

```
# Generate LLM response
def generate_response(prompt, level):
    try:
        full_prompt = f"Act as a polite but firm AI room guard.
Escalation level {level}/3: Respond to potential intruder. Keep short,
natural, engaging. Base: {prompt}."
        response = model.generate_content(full_prompt)
        return response.text.strip()
    except Exception as e:
        logging.error(f"LLM error: {e}")
        return f"Default level {level} warning."
```

This snippet loads email alert settings from a JSON file and defines a function to send an email with a subject, message, and attached image (like a captured intruder photo). It uses Gmail's SMTP server for sending alerts when unauthorized activity is detected.

```
# Load alert settings
with open('alert_settings.json', 'r') as f:
    alert_settings = json.load(f)

def send_alert(subject, body, image_path):
    if not alert_settings.get('enabled', False):
        return
    msg = MIMEMultipart()
    msg['From'] = "sfyashops1978@gmail.com"  # Replace with your Gmail
    msg['To'] = alert_settings['recipient']
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))
    with open(image_path, 'rb') as img:
        msg.attach(MIMEImage(img.read(),
name=os.path.basename(image_path)))

    with smtplib.SMTP('smtp.gmail.com', 587) as server:
        server.starttls()
        server.login("sfyashops1978@gmail.com", "gszs ctmq veef vqiz")
# Use App Password for Gmail
        server.send_message(msg)
```

This function runs a 3-level verbal escalation: it generates and speaks increasingly firm prompts (via generate_response + speak), listens for a reply (listen_for_speech), and accepts simple keyword de-escalation (e.g., "friend"/"owner"). If nobody verifies by level 3 it captures the

current camera frame, saves and emails an intruder image via send_alert, logs/announces the intrusion, and cleans up the temp file.

```
# Escalation logic (3 levels, creative and coherent)
def escalate_conversation():
    escalation_level = 1
    while escalation_level <= 3:
        if escalation_level == 1:
            prompt = "Politely ask who they are."
        elif escalation_level == 2:
            prompt = "Firmly request they leave."
        else:
            prompt = "Issue a stern warning or alarm."

        response = generate_response(prompt, escalation_level)
        speak(response)

        # Listen for reply
        reply = listen_for_speech()
        if "friend" in reply or "owner" in reply:  # Simple de-
escalation logic (enhance with LLM if needed)
            speak("Verified. Welcome.")
            break

        escalation_level += 1
        time.sleep(1)  # Pause between levels

    if escalation_level > 3:
        speak("Intruder alert! Alerting authorities.")
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        image_path = f"intruder_{timestamp}.jpg"
        cv2.imwrite(image_path, frame)  # frame from outer loop
        send_alert(f"Intruder Alert - {timestamp}", "Unrecognized
person detected.", image_path)
        print("Alert sent.")
        os.remove(image_path)
```

This snippet attempts to enroll trusted faces from the 'trusted_faces/' folder. If no embeddings are found or enrollment fails, it logs an error and stops, ensuring the agent only runs with known trusted individuals enrolled.

```
# Load or enroll embeddings
trusted_embeddings = enroll_trusted_faces('trusted_faces/')
if not trusted_embeddings:
    logging.error("No trusted faces enrolled. Exiting.")
```

This loop continuously listens for the spoken command “guard my room.” Once detected, it activates guard mode, announces it verbally, and logs that the AI agent is now monitoring the room.

```
# Activation: Listen for command
guard_mode = False
while not guard_mode:
    command = listen_for_speech()
    if "guard my room" in command:
        guard_mode = True
        speak("Guard mode activated. Monitoring room.")
        logging.info("Guard mode ON.")
```

This snippet initializes webcam capture using OpenCV to start video monitoring, checks if the webcam is accessible, and exits with an error if not. It also sets up a face tracking system using a dictionary (face_tracker) and a counter (next_face_id) to uniquely identify detected faces.

```
face_tracker = {}
next_face_id = 0
cap = cv2.VideoCapture(0)  # 0 for default webcam
if not cap.isOpened():
    logging.error("Webcam access failed.")
    exit()
```

This loop continuously captures webcam frames while the system is in guard mode. It processes every 5th frame to detect faces, recognizes trusted individuals, greets them, and flags unknown ones. Unrecognized faces trigger escalation dialogue and potential alerts. It also tracks faces over time, cleans inactive ones, and displays live video with labels and color-coded boxes (green = trusted, red = unknown).

```
frame_skip = 5  # Process every 5th frame for optimization
frame_count = 0
unmatched_id = []

while guard_mode:
    ret, frame = cap.read()
    if not ret:
        logging.error("Failed to Capture Frame")
        break
    frame_count += 1
    display_frame = frame.copy()

    if frame_count % frame_skip != 0:
        continue  # Skip frames for speed

    try:
        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
        rgb_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)
        face_locations = face_recognition.face_locations(rgb_frame)
        face_encodings = face_recognition.face_encodings(rgb_frame,
face_locations)

        current_time = time.time()
```

```
        detected = False

        for face_loc, encoding in zip(face_locations, face_encodings):
            detected = True

            # Match existing face or assign new ID (from snippet)
            matched_id = None
            for face_id, data in face_tracker.items():
                if face_recognition.compare_faces([data['encoding']],
encoding, tolerance=0.4)[0]:
                    matched_id = face_id
                    break
            if matched_id is None:
                matched_id = next_face_id
                next_face_id += 1
                trusted, person = is_trusted(encoding,
trusted_embeddings)
                face_tracker[matched_id] = {'encoding': encoding,
'last_seen': current_time, 'trusted': trusted, 'person': person}

            # Update tracking data
            face_tracker[matched_id]['last_seen'] = current_time

            # Scale back face location to original frame size
            top, right, bottom, left = [v * 4 for v in face_loc]
            color = (0, 255, 0) if face_tracker[matched_id]['trusted']
else (0, 0, 255)
            cv2.rectangle(frame, (left, top), (right, bottom), color,
2)
            label = f"ID{matched_id}: {face_tracker[matched_id]
['person'] or 'Unknown'}"
            cv2.putText(frame, label, (left, top - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

            if face_tracker[matched_id]['trusted']:
                speak_async(f"Welcome back, {face_tracker[matched_id]
['person']}.")
                logging.info(f"Trusted user: {face_tracker[matched_id]
['person']}")
            else:
                # Snippet: Untrusted persistence check
                if matched_id not in unmatched_id:
                    unmatched_id.append(matched_id)
                elapsed = current_time - face_tracker[matched_id]
['last_seen']
                cv2.rectangle(frame, (left, top), (right, bottom), (0,
0, 255), 4)  # Thicker red box
                speak_async("Unrecognized person detected")
                logging.warning(f"{len(unmatched_id)} Unknown persons
detected so far")
```

```
                # Original escalation logic (kept intact)
                logging.info("Untrusted detected. Escalating.")
                escalate_conversation()

        # Clean up expired faces (not seen for 5 seconds)
        face_tracker = {id: data for id, data in face_tracker.items()
if current_time - data['last_seen'] < 5}

        if not detected:
            time.sleep(0.1)

    except Exception as e:
        logging.error(f"Face processing error: {e}")

    cv2.imshow('AI Guard', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
logging.info("Guard mode OFF.")
```

Challenges Faced During Implementation

For the EE782 Assignment 2, we encountered several technical obstacles related to dependency installation, hardware integration, model reliability, and API configurations. These were addressed iteratively through debugging, community resources (e.g., Stack Overflow, GitHub issues), and targeted optimizations. Below, we have outlined the key challenges, their impacts, and the solutions implemented.

Challenge	Description	Impact	Solution
Dependency Installation Errors (dlib/face_recognition and PyAudio)	On Windows with Python 3.12, `pip install dlib` failed due to missing C++ build tools (CMake/Visual Studio errors), and PyAudio required manual wheel downloads for compatibility. This blocked core libraries for face recognition and audio input.	Prevented initial setup; wasted ~2-3 hours on compilation attempts.	Installed Visual Studio Community (C++ workload) for dlib builds, but switched to pre-compiled wheels from GitHub repos (e.g., z-mahmud22 for dlib-19.24.99-cp312-cp312-win_amd64.whl). For PyAudio, downloaded from Christoph Gohlke's site (PyAudio-0.2.14-cp312-cp312-win_amd64.whl)

Challenge	Description	Impact	Solution
			and installed via `pip install wheel.whl`. Updated README with Windows-specific notes.
Webcam Window Hanging/Freezing	The OpenCV webcam loop froze after a few frames due to high CPU load from `face_recognition.face_encodings()` on every frame, causing unresponsive windows and 100% CPU usage.	Disrupted real-time monitoring; made testing unreliable during Milestone 3.	Implemented frame skipping (process every 5th frame) and downsampling (resize to 1/4 resolution before encoding, scale back locations).
Face Recognition Failure	The system failed to recognise trusted faces under different variations .	Broke Milestone 2 accuracy (target 80%); required re-testing with varied data.	Added logging in `enroll_trusted_faces()` to count encodings per photo/user. Used 10-12 diverse photos per person (frontal, varied lighting/angles). Lowered tolerance to 0.4 for stricter matching. Verified with standalone encoding tests.
TTS File Lock Errors (WinError 32)	During escalation, `os.remove('response.mp3')` failed because `pygame.mixer.music` held a file lock after playback, especially on rapid successive calls.	Interrupted TTS flow; caused crashes in multi-response scenarios.	Explicitly called `pygame.mixer.music.unload()` post-playback. Added retry loop with some (up to 3 attempts).
Gmail SMTP Authentication Failure (535 5.7.8 BadCredentials)	Email alerts at level 3 failed due to deprecated "less secure app access" in Gmail (post-2025 policy), blocking SMTP login with regular password.	Prevented stretch goal completion; no notifications during demos.	Enabled 2FA and generated App Passwords via Google Account Security > App passwords. Updated `server.login()` with the 16-char token. Added SMTP-specific error handling in

Challenge	Description	Impact	Solution
			`send_alert()`. Tested with standalone script; configured via `alert_settings.json`.

Ethical Considerations:

We tested the guard with us as intruders as well as trusted faces alterating the training data among us by taking each others' photos with their consent respectively.

Testing Results

1. Milestone 1: We have uploaded a video in milestone-1 folder on github showing that the audio detection for "guard my room" works with 100% accuracy
2. The face recognition also works with high accuracy distinguishing between the trusted as well as untrusted faces. We have also tested the detection under different light conditions and it recognized the trusted face correctly even in different settings, giving no false negatives.
3. In case the guard encounters an intruder, the escalation works perfectly with an alert mail sent to the recipient email address mentioned in the `alert_settings.json` file in case the escalation goes above level 3.
4. In case the intruder, during escalation mentions he is a friend or an intruder i.e., in case the keywords `friend` or `owner` are used then escaltion stops smoothly. Hence, keyword spotting also works

Instructions to run the code:

1. Run this ipynb file and when the cell containing the code for activation (mentioned above) runs and prints `Listeining...` says `guard my room`. Thw guard will get activated and also say that guard mode is on by running an audio file.
2. After that the cv2 webcam capture window will pop up after the webcam access is opened successfully and will show whatever the webcam captures. It will detect the face in the face captured mark it will green box showing trudted face with its name and a red box otherwise.
3. In case the guard detects and intruder face but you are a friend or owner and want to stop the escaltion please say something such that the keywords `friend` or `owner` is spotted by the guard to stop the escalation

System Architecture