

**University of Bridgeport**

**Senior Design Project Fall 2024- Spring 2025**

**Group 1:** Patricia Bermeo  
 [Jordan Berry](https://bridgeport.instructure.com/groups/733956/users/5719835)

[Felix Bonilla](https://bridgeport.instructure.com/groups/733956/users/5553811)

[Thomas Schoenfelder](https://bridgeport.instructure.com/groups/733956/users/5912613)

**Professor:** [Ahmed El Sayed](https://bridgeport.instructure.com/courses/2046850/users/4084146) Ph.D.

**Advisor:** Ausif Mahmood Ph.D.

**Table of Contents**

1. Project Title
2. Team Members’ Names
3. Advisor’s Name
4. Detailed Explanation of Our Final Goals  
   4.1 Description  
   4.2 Objective
5. Expected Timeline
6. Marketplace or Community Value  
   6.1 Unique Features  
   6.2 Competitor Analysis
7. Cost of the Project
8. Selling Price & Licensing Method
9. Frame Infrastructure & Components  
   9.1 Hardware Components  
    9.1.1 Hardware Schematics

9.1.2 Hardware Block Diagram

9.2 Software Architecture

9.2.1 Software Diagram

1. Hardware & Software List  
    10.1 Hardware  
    10.2 Software  
    10.2.1 Libraries used   
    10.2.2 Functions used  
    10.2.3 AI Model Training and Fine-Tuning
2. Task Distribution
3. Results
4. Conclusion
5. References
6. Appendix

**Group 1 Report**

**1. Project Title**

3BT SmartAI Home Surveillance SystemThe name “3BT” comes from our team. Three of us have last names that start with “B”, and the “T” stands for Thomas, representing one of our teammates whose last name stands with “S”. We chose this name because we wanted something more personalized and unique that would reflect our collaboration and give a special identity to our project.  
  
**2. Team Members’ Names**

* Patricia Bermeo
* Jordan Berry
* Felix Bonilla
* Thomas Schoenfelder

**3. Advisors Name**

* Ausif Mahmood, Ph.D

**4. A Detailed Explanation of Our Final Goals**

**4.1 Description:**

The 3BT SmartAI Home Surveillance System[1] is a modern security solution that uses advanced Machine Learning[2] for Facial Recognition [3] to make homes safer. Unlike regular systems like Ring[4], our system can automatically tell the difference between family members and intruders, which helps reduce false alarms and improve overall security.

Additionally, the system includes behavior analysis[5], which adds a more advanced feature beyond just detecting motion or faces. This means the system can understand and react to different behaviors, making it smarter and more adaptable to various situations. This increases the system’s efficiency and effectiveness.

**4.2 Objective:**

Our goal is to develop a Smart Surveillance System[1] that not only accurately recognizes people, but also studies Behavior Patterns[5]. With this advanced feature, the system will automatically spot any unusual or suspicious activity, reducing the need for manual alerts and providing a more proactive way to protect your home. This system overcomes the weakness of current security options and gives users stronger protection and a greater sense of safety.

**5.Timeline**

* **October-November 2024:** Research, planning, and hardware procurement.
* **December 2024-January 2025:** Hardware setup, system integration, and AI model development.
* **February 2025:** Continue withHardware andSoftware development and Behavioral analysis integration.
* **March 2025:** System testing, Debugging, and optimization
* **April 2025:** Final presentation and documentation

**6. Marketplace or Community Value**

The 3BT SmartAI Home Surveillance[1] System is more advanced and affordable than other home security products like Ring[4], which does not have AI features like behavior detection. Our system does more than just Detect Motion[6] or recognize faces[3], it analyzes behavior[5], allowing it to spot suspicious actions and report quickly.

**6.1 Compared to competitors, 3BT SmartAI provides:**

* **Advanced Recognition:** Can tell the difference between family members and intruders.
* **Behavior Based Alerts**: Sends smarter alerts based on what someone is doing, not just movement.
* **Affordability**: Costs under $250 with Facial Recognition and Behavior Analysis[5] for smarter security.

**6.2 Competitor Analysis**

A black and white table with black text

Description automatically generated

**7.Cost of the Project**

* **Hardware Costs:** approximately $212
* **Software Costs:**  Open-source[7] software under MIT license[8].
* **Total Estimated Budget:** $212

**8.Selling Price & Licensing Method**

The 3BT Smart AI Home Surveillance System[1] will cost less than $250 (exact price to be decided). Customers will make a one-time payment for the hardware, and since it does not require Cloud storage[9], it eliminates ongoing fees. With Open-source[7] components, it provides flexibility and easy use, along with smart behavior detection that makes it better than basic systems.

**9.Frame Infrastructure & Components**

**9.1 Hardware Components**

* **Frame:** Protective cover plate/ Camera mounting plate
* **Power Supply:** Direct connection to power
* **Connectivity:** Wi-Fi[11] enabled for remote access and mobile/web notifications

**9.1.1 Hardware Schematics**

**A diagram of a computer

AI-generated content may be incorrect.**

**9.1.2 Hardware Block Diagram**

**A diagram of a computer system

AI-generated content may be incorrect.**

**9.2 Software Architecture**

* **Camera Feed Management:** Capture and process live video from one camera for now, with plans to support several cameras in real-time in the future.
* **Facial Recognition:** Utilize libraries such as OpenCV[12], Deepface[13], to recognize family members and unauthorized individuals, minimizing false alarms.
* **Behavioral Analysis:** Train AI models to detect people wandering around, restricted area entry, and other suspicious behaviors.
* **Database and Storage:** Store profiles of family members and authorized individuals for Facial Recognition[3] and save surveillance footage locally.
* **Alert and Notification System:** Push notifications and real-time alerts will be sent via mobile through telegram bot, and users can customize notifications based on the type of detection (person, intruder).

**9.2.1 Software Diagram**

**A diagram of a computer hardware system

Description automatically generated**

**10.Expected Hardware & Software List**

**10.1 Hardware**

**A screenshot of a computer

AI-generated content may be incorrect.**

|  |
| --- |
|  |
| **10.2 Software:**   * **Operating System:** Raspberry Pi Linux [15] * **Programming Languages:** Python[16] * **AI Libraries:** OpenCV[12] Deepface[13]   **10.2.1Libraries used in our project**  This is how these libraries where used in our project  **1. tkinter[22]**     tkinter is the backbone of the user interface; everything starts here. It triggers file uploads, opens the camera, and links to all other functions like open\_file\_explorer(), show\_camera\_screen(), and show\_delete\_screen(). Functions such as show\_selected\_images() and ask\_for\_name\_and\_create\_file() are only executed when tkinter events are triggered.    **2. PIL (Pillow)[23]**     PIL works hand-in-hand with tkinter to display images. It is called by show\_selected\_images() to preview uploaded training photos, and it formats OpenCV images into a format tkinter can display. It depends on OpenCV (cv2) for the raw images and supports tkinter by wrapping them as ImageTk.PhotoImage.    **3. os [18]**     The os module is used by nearly every file-based function: add\_person\_to\_system(), delete\_folder(), and load\_phone\_number\_and\_chat\_id(). It provides path validation and manipulation to let shutil and pickle[28] work correctly. It relies on the directory structure being initialized by functions like add\_person\_to\_system().    **4. shutil[24]**     Relies on os for file paths and is used for tasks like deleting directories in delete\_folder() and moving files if needed. It's critical for managing the structure of the stored face data and removing a person’s training data entirely.    **5. numpy[17]**     Used inside train\_face\_recognition\_iteratively() and update\_camera\_feed() to compare face embeddings. It relies on deepface[13] to first generate those embeddings, and is critical in computing the similarity between vectors when identifying known or unknown faces.    **6. deepface[13]**     Central to facial recognition — it's used by train\_face\_recognition\_iteratively(), crop\_face(), and update\_camera\_feed(). It feeds numpy the face embeddings and handles all face detection and alignment. Without deepface[13], the whole recognition pipeline collapses, as it is responsible for both extracting features and determining identity.    **7. cv2 (OpenCV)[12]**     Required by take\_picture() and update\_camera\_feed() for image capture and frame processing. It captures raw frames which are then converted for deepface[13] or displayed in tkinter. It relies on picamera2 to provide camera input on the Raspberry Pi.    **8. pickle[20]**     Used by train\_face\_recognition\_iteratively() to save face embeddings and by start\_camera\_feed() to load them. It depends on numpy[17] arrays generated by deepface[13] and is essential for persistent recognition without retraining.    **9. requests[25]**     Sends HTTP POST messages in send\_telegram\_message() and uses phone-to-chat ID mapping from JSON files. It relies on json-loaded data and integrates with Telegram’s API to send alerts when handle\_unknown\_face() triggers it.    **10. json [26]**     Used in load\_phone\_number\_and\_chat\_id() and save\_phone\_number\_and\_chat\_id() to store chat ID mappings. It connects to the requests system and relies on tkinter to prompt the user for a phone number.    **11. picamera2[27]**     Feeds real-time frames into start\_camera\_feed() and take\_picture(). It works with cv2 to process and deliver images. It’s the camera source for the entire recognition process, especially on Raspberry Pi platforms.  **13. telegram API[21]**  The Telegram API enables communication between the system and users via Telegram messages. It is used for sending alerts, receiving commands, and providing real-time notifications. The API allows users to receive updates about unknown faces detected by the system. By integrating with Telegram, the system ensures remote  monitoring and instant feedback. The Telegram bot functionality makes the system more  interactive and user-friendly.  **10.2.2 Functions used in our project**  **1. send\_telegram\_message()**  Relies on requests to communicate with Telegram[21] and on load\_phone\_number\_and\_chat\_id() for destination info. It's called by handle\_unknown\_face() and sends alerts after deepface[13] has flagged someone as unknown.    **2. get\_telegram\_updates()**     Uses requests to fetch the latest Telegram[21] messages. It is invoked by wait\_for\_telegram\_response() and relies on a running Telegram[21] bot to provide chat feedback.    **3. get\_chat\_id\_from\_updates()**  Parses the JSON response from get\_telegram\_updates() to find the chat ID. It’s essential for connecting a Telegram[21] user to the Python project and is used during registration of new users.    **4. save\_phone\_number\_and\_chat\_id()**  Depends on tkinter input and writes data via json. It allows send\_telegram\_message() to function in the future without manual input, making the whole alert system reusable.    **5. load\_phone\_number\_and\_chat\_id()**     Reads the saved mappings via json and is called by send\_telegram\_message() to route alerts. It must be called early to ensure proper linkage between the user and Telegram[21]**.**    **6. handle\_unknown\_face()**  Calls send\_telegram\_message() and wait\_for\_telegram\_response() to manage the entire "unknown face" flow. It depends on deepface[13] to detect the unknown and on cv2 to grab the corresponding image.    **7. wait\_for\_telegram\_response()**     Waits until get\_telegram\_updates() returns a matching reply. It allows handle\_unknown\_face() to adapt based on human feedback, forming a feedback loop for false positives.    **8. add\_person\_to\_system()**  Uses os and shutil to create or update the training folder for a new person. It's called by ask\_for\_name\_and\_create\_file() and prepares the structure used by train\_face\_recognition\_iteratively().    **9. train\_face\_recognition\_iteratively()**  Uses deepface[13] to generate face embeddings and pickle to save them. It relies on images prepared by add\_person\_to\_system() and is necessary for recognition to work later.    **10. crop\_face()**     Uses deepface[13] internally to extract a consistent face region from an image. It's called during both training (train\_face\_recognition\_iteratively()) and recognition to standardize inputs.      **11. show\_add\_photos\_screen()**  Depends on tkinter and opens the options for adding a new face. It leads to open\_file\_explorer() or take\_picture(), and eventually to ask\_for\_name\_and\_create\_file().    **12. open\_file\_explorer()**  Called by show\_add\_photos\_screen(), lets users choose images. It passes those images to show\_selected\_images() and eventually into the training pipeline.    **13. show\_selected\_images()**  Uses PIL to display thumbnails of training images in tkinter. It’s a visual helper that works with images selected via open\_file\_explorer().    **14. ask\_for\_name\_and\_create\_file()**  Collects the name from the user and calls add\_person\_to\_system(). It’s a bridge between GUI input and backend training setup.    **15. take\_picture()**  Uses cv2 and picamera2 to capture images live. These are saved and passed to the training function (train\_face\_recognition\_iteratively()).    **16. show\_delete\_screen()**  GUI function that lists all registered users using os. It leads to confirm\_delete() and eventually delete\_folder().  **17. confirm\_delete()**  Pops up a warning dialog before deletion. If confirmed, it passes control to delete\_folder().    **18. delete\_folder()**     Uses shutil and os to delete a face’s data. After deletion, it triggers refresh\_camera\_feed() to update live recognition.    **19. refresh\_camera\_feed()**     Reloads pickled face data via pickle and updates recognition. It's used after training or deleting faces to refresh the model.    **20. show\_camera\_screen()**     Initializes the camera interface with tkinter and starts the recognition loop. It calls start\_camera\_feed() to begin streaming.    **21. start\_camera\_feed()**     Loads embeddings with pickle, then starts the camera using cv2 and picamera2. It relies on update\_camera\_feed() to keep the stream going.    **22. update\_camera\_feed()**  The live recognition engine — it captures a frame, uses deepface[13] for detection, compares vectors with numpy[17], and updates the tkinter GUI. It calls handle\_unknown\_face() if the person isn’t recognized.    **23. show\_enter\_phone\_number\_screen()**  tkinter window that collects the user’s number. It uses save\_phone\_number\_and\_chat\_id() to register the Telegram[21] ID and connects messaging with recognition.    **24. show\_main\_menu()**  The first screen shown. From here, all GUI options are presented: add, delete, recognize, or register a phone. It relies on tkinter and routes to all major functions. |

**10.2.3 AI Model Training and Fine-Tuning**

* **Dataset Collection:** Gather data to train AI models for accurate recognition of household members.
* **Training Models:** Fine-tine pre-trained models for Facial Recognition [3] under various lighting conditions and train models to distinguish between family members, and intruders.
* **Performance Optimization:** Regularly refine the models to reduce false positives and false negatives, and periodically retrain the models using real-world data to improve performance.

**11. Task Distribution**

* **Electrical Engineer:**

-Power Supply and Sensor integration.

-Ensuring reliable connections and functionality between devices.

* **Hardware Engineer:**

-Assembling hardware components.

-Designing enclosures and handling wiring.

-Documentation and Presentation

* **Software Engineer 1(AI/Backend):**

-Developing AI models for Facial Recognition [3]and behavior detection.

-Implementing Machine Learning [2] using Deepface[13]

* **Software Engineer 2 (Frontend/Integration):**

-Developing the user interface for control and monitoring.

-Integrating notification systems and real-time alerts

**12.Results:**

* SmartAI Accuracy: successfully detects and classifies individuals with an accuracy of 100%

A black background with white text

AI-generated content may be incorrect.

* Detection Efficiency: The AI model correctly identifies authorized users in 100% of cases and flags intruders in 100% of cases.

A black background with white text

AI-generated content may be incorrect.

* Unauthorized Access: Unauthorized users were successfully flagged, preventing false access.

A screenshot of a chat

AI-generated content may be incorrect.

* Mobile Alert Feature: A mobile alert system has been implemented to confirm new individuals and create a profile if necessary.

A person wearing headphones

AI-generated content may be incorrect.

**13. Conclusion:**

The 3BT SmartAI Home Surveillance System offers a smarter way of improving home security with more efficiency and at an affordable cost. It includes advanced AI features like facial recognition[3] using DeepFace[13] and behavior analysis[5], making it better than other systems like Ring [4]. By analyzing behavior like loitering [5] and sending real-time alerts, the 3BT will make a surveillance system more accurate by reducing false alarms in a more active way of keeping homes safe. It is estimated that it will cost less than $250, which makes the surveillance system both affordable and high-tech for the average homeowner. This is a high-level improvement in home security, giving users more protection and safety.

**14.References:**

1. *Intelligent video surveillance systems*. (n.d.). Google Books. https://books.google.com/books?id=tqEOLOKT\_fAC&lpg=PT9&ots=oFqn5Pl2Av&dq=video%20surveillance%20systems&lr&pg=PT48#v=onepage&q&f=false

2. Yang, Q., Liu, Y., Chen, T., & Tong, Y. (2019). Federated Machine Learning. *ACM Transactions on Intelligent Systems and Technology*, *10*(2), 1–19. https://doi.org/10.1145/3298981

3. *What is Facial Recognition? - Facial Recognition Technology Explained - AWS*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/facial-recognition/

4. Ring Europe. (n.d.). *Get smart security with ring doorbells, cams & security systems*. Ring. https://ring.com/

5. *About behavior analysis*. (n.d.). College of Liberal Arts at Auburn University. https://cla.auburn.edu/psychological-sciences/applied-behavior-analysis/about-behavior-analysis/

6. *Introduction to motion detection*. (n.d.). https://support.reolink.com/hc/en-us/articles/900000623266-Introduction-to-Motion-Detection/

7. *Home page*. (n.d.). Open Source Initiative. https://opensource.org/

8. *The MIT license*. (n.d.). Open Source Initiative. https://opensource.org/license/mit

9. *What is Cloud Storage? - Cloud Storage Explained - AWS*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/cloud-storage/

10. *3D printing: what it is, how it works and examples*. (n.d.). Built In. https://builtin.com/3d-printing

11. *WiFi definition and meaning | WaTech*. (n.d.). https://watech.wa.gov/wifi-definition-and-meaning

12. OpenCV. (2024, December 18). *OpenCV - Open Computer Vision Library*. https://opencv.org/

13. *Deepface. (2024, August 17). PyPI. https://pypi.org/project/deepface*

14. *What is PyTorch?* (n.d.). NVIDIA Data Science Glossary. https://www.nvidia.com/en-us/glossary/pytorch/

15. Nuttall, B. (n.d.). *What is a Raspberry Pi?* Opensource.com. https://opensource.com/resources/raspberry-pi

16. *Welcome to Python.org*. (2024, December 18). Python.org. https://www.python.org/

17. *NumPy*. (n.d.). https://numpy.org/

18.*os — Miscellaneous operating system interfaces*. (n.d.). Python Documentation. https://docs.python.org/3/library/os.html

19. *time — Time access and conversions*. (n.d.). Python Documentation. https://docs.python.org/3/library/time.html

20. *pickle — Python object serialization*. (n.d.). Python Documentation. https://docs.python.org/3/library/pickle.html

21. *python-telegram-bot*. (n.d.). https://python-telegram-bot.org/

22. *tkinter — Python interface to Tcl/Tk*. (n.d.). Python Documentation. https://docs.python.org/3/library/tkinter.html

23. *pillow*. (2025, April 12). PyPI. https://pypi.org/project/pillow/

24. *shutil — High-level file operations*. (n.d.). Python Documentation. https://docs.python.org/3/library/shutil.html

25. *requests*. (2024, May 29). PyPI. https://pypi.org/project/requests/

26. *json — JSON encoder and decoder*. (n.d.). Python Documentation. https://docs.python.org/3/library/json.html

27. *picamera — Picamera 1.13 Documentation*. (n.d.). https://picamera.readthedocs.io/en/release-1.13/

**15.Appendix**

**3BT Smart Surveillance System Full Code Implementation:**

# Import necessary libraries

import tkinter as tk # GUI library

from tkinter import filedialog, messagebox # For file dialogs and message boxes

from PIL import Image, ImageTk # For image processing

import os # For operating system interactions

import shutil # For file operations

import numpy as np # For numerical operations

from deepface import DeepFace # For face recognition

import cv2 # OpenCV for image processing

import pickle # For serializing/deserializing Python objects

import requests # For HTTP requests (Telegram API)

import time # For time-related functions

import threading # For multi-threading (not used in current code)

import json # For JSON data handling

from picamera2 import Picamera2 # For Raspberry Pi camera control

# Telegram credentials (Note: These should not be hardcoded in production)

TELEGRAM\_BOT\_TOKEN = "7231732441:AAH3dKfLkKOFQJ0Bmc5anOA-on-T5Nxl1Fk" # Bot token

TELEGRAM\_CHAT\_ID = None # Will be set dynamically

# Global variables

last\_update\_id = None # For Telegram update tracking

camera\_frame = None # For displaying camera feed

selected\_images = [] # Stores selected image paths

face\_data = [] # Stores face recognition data

picam2 = None # For Raspberry Pi camera instance

# ----------------- Telegram & Face Recognition Functions -----------------

# Function to send a message (and optionally an image) via Telegram

def send\_telegram\_message(chat\_id, image\_path=None, message=None):

try:

if image\_path: # If sending an image

url = f"https://api.telegram.org/bot{TELEGRAM\_BOT\_TOKEN}/sendPhoto"

files = {"photo": open(image\_path, "rb")} # Open image file

payload = {"chat\_id": chat\_id, "caption": message} # Message caption

response = requests.post(url, files=files, data=payload) # Send request

else: # If sending only text

url = f"https://api.telegram.org/bot{TELEGRAM\_BOT\_TOKEN}/sendMessage"

payload = {"chat\_id": chat\_id, "text": message} # Text message

response = requests.post(url, data=payload) # Send request

if response.status\_code != 200: # Check for errors

print(f"Failed to send Telegram message. Error: {response.text}")

except Exception as e: # Handle exceptions

print(f"Error sending Telegram message: {e}")

# Function to get updates from Telegram

def get\_telegram\_updates():

global last\_update\_id # Access global variable

url = f"https://api.telegram.org/bot{TELEGRAM\_BOT\_TOKEN}/getUpdates"

params = {"offset": last\_update\_id + 1 if last\_update\_id else None, "timeout": 10} # Parameters for updates

response = requests.get(url, params=params).json() # Get updates

if response.get("ok"): # If successful

updates = response.get("result", []) # Get updates list

if updates: # If there are updates

last\_update\_id = updates[-1]["update\_id"] # Update last\_update\_id

return updates # Return updates

return [] # Return empty list if no updates

# Function to get chat ID from Telegram updates

def get\_chat\_id\_from\_updates():

updates = get\_telegram\_updates() # Get updates

for update in updates: # Iterate through updates

if "message" in update: # If update contains a message

return update["message"]["chat"]["id"] # Return chat ID

return None # Return None if no chat ID found

# Function to save phone number and chat ID to a JSON file

def save\_phone\_number\_and\_chat\_id(phone\_number, chat\_id):

data = {"phone\_number": phone\_number, "chat\_id": chat\_id} # Create data dictionary

with open("phone\_number\_chat\_id.json", "w") as f: # Open file for writing

json.dump(data, f) # Write data to file

# Function to load phone number and chat ID from JSON file

def load\_phone\_number\_and\_chat\_id():

if os.path.exists("phone\_number\_chat\_id.json"): # Check if file exists

with open("phone\_number\_chat\_id.json", "r") as f: # Open file for reading

return json.load(f) # Return loaded data

return None # Return None if file doesn't exist

# Function to handle unknown faces (send notification via Telegram)

def handle\_unknown\_face(image\_path):

data = load\_phone\_number\_and\_chat\_id() # Load saved data

if not data or "chat\_id" not in data: # Check if chat ID exists

print("No Chat ID found.")

return

chat\_id = data["chat\_id"] # Get chat ID

send\_telegram\_message(chat\_id, image\_path, "🚨 DO YOU KNOW THE PERSON? Reply with /yes or /no.") # Send message

response = wait\_for\_telegram\_response() # Wait for response

if response == "/yes": # If user knows the person

send\_telegram\_message(chat\_id, None, "PLEASE ENTER THE PERSON'S FIRST NAME AND LAST NAME.") # Ask for name

person\_name = wait\_for\_telegram\_response() # Wait for name

if person\_name: # If name provided

add\_person\_to\_system(image\_path, person\_name) # Add person to system

refresh\_camera\_feed() # Refresh camera feed

elif response == "/no": # If user doesn't know the person

location = "Office Camera" # Default location

send\_telegram\_message(chat\_id, None, f"THE INFO WILL BE SENT TO ALT + {location}.") # Send notification

# Function to wait for Telegram response

def wait\_for\_telegram\_response():

while True: # Infinite loop

updates = get\_telegram\_updates() # Get updates

for update in updates: # Iterate through updates

if "message" in update and "text" in update["message"]: # If text message exists

return update["message"]["text"].strip().lower() # Return message text

time.sleep(1) # Wait 1 second before checking again

# Function to add a new person to the system

def add\_person\_to\_system(image\_path, person\_name):

folder\_name = f"{person\_name}" # Create folder name

os.makedirs(folder\_name, exist\_ok=True) # Create folder

for i in range(1, 4): # Copy image 3 times (for training)

file\_name = f"{folder\_name}/image\_{i}.jpg" # Create file name

shutil.copy(image\_path, file\_name) # Copy image

train\_face\_recognition\_iteratively(folder\_name, person\_name) # Train model

data = load\_phone\_number\_and\_chat\_id() # Load saved data

if data and "chat\_id" in data: # If chat ID exists

send\_telegram\_message(data["chat\_id"], None,

f"{person\_name} has been added to the system.") # Send confirmation

# Function to train face recognition model iteratively

def train\_face\_recognition\_iteratively(folder\_name, person\_name):

face\_encodings = [] # List to store face encodings

image\_files = [f for f in os.listdir(folder\_name) if f.endswith(".jpg")] # Get all JPG files in folder

max\_iterations = 10 # Maximum training iterations

tolerance = 0.6 # Matching tolerance

for iteration in range(max\_iterations): # Loop for each iteration

print(f"Iteration {iteration + 1} of training...") # Print iteration info

for img\_file in image\_files: # Process each image

img\_path = os.path.join(folder\_name, img\_file) # Get full image path

try:

image = cv2.imread(img\_path) # Read image

if image is None: # If image not loaded

raise ValueError(f"Could not load image: {img\_path}") # Raise error

cropped\_face = crop\_face(image) # Crop face from image

embedding = np.array(DeepFace.represent(cropped\_face, model\_name="ArcFace", enforce\_detection=False)[0][

"embedding"]) # Get face embedding

embedding = embedding / np.linalg.norm(embedding) # Normalize embedding

face\_encodings.append(embedding) # Add to encodings list

except Exception as e: # Handle errors

messagebox.showwarning("Warning", f"Error processing {img\_file}: {str(e)}") # Show warning

continue # Skip to next image

if not face\_encodings: # If no valid faces found

messagebox.showerror("Error", "No valid faces found in the folder.") # Show error

return # Exit function

data = {"name": person\_name, "encodings": face\_encodings} # Create data dictionary

with open(f"{folder\_name}/face\_data.pkl", "wb") as f: # Open file for writing

pickle.dump(data, f) # Save data

confident = True # Confidence flag

for img\_file in image\_files: # Validate each image

img\_path = os.path.join(folder\_name, img\_file) # Get full image path

try:

image = cv2.imread(img\_path) # Read image

if image is None: # If image not loaded

raise ValueError(f"Could not load image: {img\_path}") # Raise error

cropped\_face = crop\_face(image) # Crop face

face\_embedding = np.array(

DeepFace.represent(cropped\_face, model\_name="ArcFace", enforce\_detection=False)[0][

"embedding"]) # Get embedding

face\_embedding = face\_embedding / np.linalg.norm(face\_embedding) # Normalize

for encoding in face\_encodings: # Compare with saved encodings

encoding = np.array(encoding) # Convert to numpy array

encoding = encoding / np.linalg.norm(encoding) # Normalize

distance = np.linalg.norm(encoding - face\_embedding) # Calculate distance

if distance > tolerance: # If distance exceeds tolerance

confident = False # Set flag to False

break # Break loop

if not confident: # If not confident

break # Break loop

except Exception as e: # Handle errors

print(f"Error processing face: {e}") # Print error

if confident: # If model is confident

print("Model is confident. Training complete.") # Print message

break # Break training loop

else: # If model is not confident

print("Model is uncertain. Adding more images and retraining...") # Print message

messagebox.showinfo("Info", "The AI is uncertain. Please add more images of your face.") # Show info

open\_file\_explorer() # Open file explorer to add more images

image\_files = [f for f in os.listdir(folder\_name) if f.endswith(".jpg")] # Update image files list

# Create popup window for training completion

training\_done\_window = tk.Toplevel(root) # Create new window

training\_done\_window.title("Training Complete") # Set title

popup\_width = 300 # Width of popup

popup\_height = 150 # Height of popup

x\_position = root.winfo\_x() + (root.winfo\_width() - popup\_width) // 2 # Calculate x position

y\_position = root.winfo\_y() + (root.winfo\_height() - popup\_height) // 2 # Calculate y position

training\_done\_window.geometry(f"{popup\_width}x{popup\_height}+{x\_position}+{y\_position}") # Set geometry

tk.Label(training\_done\_window, text="AI Training is Done!", font=("Arial", 14)).pack(pady=20) # Add label

tk.Button(training\_done\_window, text="Confirm", font=("Arial", 14), command=training\_done\_window.destroy).pack(

pady=10) # Add button

# Function to crop face from image

def crop\_face(image):

try:

detected\_faces = DeepFace.extract\_faces(image, detector\_backend="mtcnn") # Detect faces

if not detected\_faces: # If no faces detected

raise ValueError("No faces detected in the image.") # Raise error

face = detected\_faces[0] # Get first face

cropped\_face = face["face"] # Get cropped face

if cropped\_face.shape[2] == 4: # If RGBA image

cropped\_face = cv2.cvtColor(cropped\_face, cv2.COLOR\_RGBA2RGB) # Convert to RGB

cropped\_face = cv2.resize(cropped\_face, (160, 160)) # Resize face

return cropped\_face # Return cropped face

except Exception as e: # Handle errors

raise ValueError(f"Error detecting face: {e}") # Raise error

# ----------------- GUI Functions -----------------

# Function to show the "Add Photos" screen

def show\_add\_photos\_screen():

for widget in root.winfo\_children(): # Clear current widgets

widget.destroy()

tk.Label(root, text="Add Photos (Only JPG)", font=("Arial", 24)).pack(pady=20) # Add title

global preview\_frame # Access global variable

preview\_frame = tk.Frame(root) # Create frame for preview

preview\_frame.pack(pady=20) # Pack frame

# Add buttons for different actions

tk.Button(root, text="Add Pictures", font=("Arial", 18), width=20, height=2,

command=open\_file\_explorer).pack(expand=True, pady=10) # Add button for file explorer

tk.Button(root, text="Take Picture", font=("Arial", 18), width=20, height=2,

command=take\_picture).pack(expand=True, pady=10) # Add button for taking picture

tk.Button(root, text="Back to Main Menu", font=("Arial", 14),

command=show\_main\_menu).pack(side="left", padx=20, pady=20) # Add back button

# Function to open file explorer for selecting images

def open\_file\_explorer():

global selected\_images # Access global variable

if len(selected\_images) < 3: # If less than 3 images selected

file\_path = filedialog.askopenfilename(filetypes=[("JPG Files", "\*.jpg")]) # Open file dialog

if file\_path: # If file selected

selected\_images.append(file\_path) # Add to selected images

show\_selected\_images() # Show selected images

if len(selected\_images) == 3: # If 3 images selected

ask\_for\_name\_and\_create\_file() # Ask for name

else: # If already 3 images selected

messagebox.showinfo("Info", "You have already selected 3 JPGs.") # Show info

# Function to display selected images

def show\_selected\_images():

for widget in preview\_frame.winfo\_children(): # Clear current preview

widget.destroy()

for img\_path in selected\_images: # For each selected image

img = Image.open(img\_path) # Open image

img.thumbnail((150, 150)) # Create thumbnail

photo = ImageTk.PhotoImage(img) # Convert to PhotoImage

label = tk.Label(preview\_frame, image=photo) # Create label

label.image = photo # Keep reference

label.pack(side="left", padx=10, pady=10) # Pack label

# Function to ask for name and create folder

def ask\_for\_name\_and\_create\_file():

def confirm\_and\_save(): # Nested function for confirmation

first\_name = first\_name\_entry.get().strip() # Get first name

last\_name = last\_name\_entry.get().strip() # Get last name

if not first\_name or not last\_name: # If names not provided

messagebox.showerror("Error", "Please enter both first and last names.") # Show error

return

folder\_name = f"{first\_name}\_{last\_name}" # Create folder name

os.makedirs(folder\_name, exist\_ok=True) # Create folder

for idx, img\_path in enumerate(selected\_images): # Copy images

file\_name = f"{folder\_name}/image\_{idx + 1}.jpg" # Create filename

shutil.copy(img\_path, file\_name) # Copy image

train\_face\_recognition\_iteratively(folder\_name, f"{first\_name} {last\_name}") # Train model

name\_window.destroy() # Close window

selected\_images.clear() # Clear selected images

show\_main\_menu() # Show main menu

refresh\_camera\_feed() # Refresh camera feed

# Create popup window for name entry

name\_window = tk.Toplevel(root) # Create new window

name\_window.title("Enter Name") # Set title

popup\_width = 400 # Width of popup

popup\_height = 200 # Height of popup

x\_position = root.winfo\_x() + (root.winfo\_width() - popup\_width) // 2 # Calculate x position

y\_position = root.winfo\_y() + (root.winfo\_height() - popup\_height) // 2 # Calculate y position

name\_window.geometry(f"{popup\_width}x{popup\_height}+{x\_position}+{y\_position}") # Set geometry

# Add widgets for name entry

tk.Label(name\_window, text="First Name:", font=("Arial", 14)).pack(pady=5) # Add label

first\_name\_entry = tk.Entry(name\_window, font=("Arial", 14)) # Add entry

first\_name\_entry.pack(pady=5) # Pack entry

tk.Label(name\_window, text="Last Name:", font=("Arial", 14)).pack(pady=5) # Add label

last\_name\_entry = tk.Entry(name\_window, font=("Arial", 14)) # Add entry

last\_name\_entry.pack(pady=5) # Pack entry

tk.Button(name\_window, text="Confirm", font=("Arial", 14), command=confirm\_and\_save).pack(pady=20) # Add button

# Function to take a picture using camera

def take\_picture():

global selected\_images, picam2 # Access global variables

if picam2 is None: # If camera not initialized

picam2 = Picamera2() # Initialize camera

picam2.configure(picam2.create\_preview\_configuration()) # Configure

picam2.start() # Start camera

def capture\_image(): # Nested function for capturing image

frame = picam2.capture\_array() # Capture frame

if frame is not None: # If frame captured

if frame.shape[2] == 4: # If RGBA

frame = cv2.cvtColor(frame, cv2.COLOR\_RGBA2RGB) # Convert to RGB

image\_path = f"captured\_image\_{len(selected\_images) + 1}.jpg" # Create filename

cv2.imwrite(image\_path, frame) # Save image

selected\_images.append(image\_path) # Add to selected images

show\_selected\_images() # Show selected images

if len(selected\_images) == 3: # If 3 images captured

ask\_for\_name\_and\_create\_file() # Ask for name

# Create camera window

camera\_window = tk.Toplevel(root) # Create new window

camera\_window.title("Take Picture") # Set title

popup\_width = 640 # Width of window

popup\_height = 580 # Height of window

x\_position = root.winfo\_x() + (root.winfo\_width() - popup\_width) // 2 # Calculate x position

y\_position = root.winfo\_y() + (root.winfo\_height() - popup\_height) // 2 # Calculate y position

camera\_window.geometry(f"{popup\_width}x{popup\_height}+{x\_position}+{y\_position}") # Set geometry

camera\_frame = tk.Label(camera\_window) # Create label for camera feed

camera\_frame.pack() # Pack label

tk.Button(camera\_window, text="Take Picture", font=("Arial", 14), command=capture\_image).pack(

pady=10) # Add capture button

def update\_camera\_feed(): # Function to update camera feed

frame = picam2.capture\_array() # Capture frame

if frame is not None: # If frame captured

if frame.shape[2] == 4: # If RGBA

frame = cv2.cvtColor(frame, cv2.COLOR\_RGBA2RGB) # Convert to RGB

img = Image.fromarray(frame) # Convert to PIL Image

img.thumbnail((640, 480)) # Create thumbnail

photo = ImageTk.PhotoImage(img) # Convert to PhotoImage

camera\_frame.config(image=photo) # Update label

camera\_frame.image = photo # Keep reference

camera\_frame.after(10, update\_camera\_feed) # Schedule next update

update\_camera\_feed() # Start camera feed

# Function to show delete screen

def show\_delete\_screen():

for widget in root.winfo\_children(): # Clear current widgets

widget.destroy()

tk.Label(root, text="Delete Photos", font=("Arial", 24)).pack(pady=20) # Add title

folders = [f for f in os.listdir() if

os.path.isdir(f) and os.path.exists(f"{f}/face\_data.pkl")] # Get folders with face data

global folder\_listbox # Access global variable

folder\_listbox = tk.Listbox(root, font=("Arial", 14), selectmode=tk.SINGLE) # Create listbox

for folder in folders: # Add folders to listbox

folder\_listbox.insert(tk.END, folder) # Insert folder

folder\_listbox.pack(pady=20) # Pack listbox

# Add buttons

tk.Button(root, text="Confirm", font=("Arial", 14), command=confirm\_delete).pack(pady=10) # Confirm button

tk.Button(root, text="Back to Main Menu", font=("Arial", 14), command=show\_main\_menu).pack(side="bottom",

pady=20) # Back button

# Function to confirm deletion

def confirm\_delete():

selected\_folder = folder\_listbox.get(tk.ACTIVE) # Get selected folder

if not selected\_folder: # If no folder selected

messagebox.showinfo("Info", "Please select a folder to delete.") # Show info

return

# Create confirmation window

confirm\_window = tk.Toplevel(root) # Create new window

confirm\_window.title("Confirm Delete") # Set title

popup\_width = 300 # Width of window

popup\_height = 150 # Height of window

x\_position = root.winfo\_x() + (root.winfo\_width() - popup\_width) // 2 # Calculate x position

y\_position = root.winfo\_y() + (root.winfo\_height() - popup\_height) // 2 # Calculate y position

confirm\_window.geometry(f"{popup\_width}x{popup\_height}+{x\_position}+{y\_position}") # Set geometry

# Add widgets

tk.Label(confirm\_window, text=f"Are you sure you want to delete {selected\_folder}?", font=("Arial", 14)).pack(

pady=20) # Add label

tk.Button(confirm\_window, text="Cancel", font=("Arial", 14), command=confirm\_window.destroy).pack(side="left",

padx=20,

pady=10) # Cancel button

tk.Button(confirm\_window, text="Delete", font=("Arial", 14),

command=lambda: delete\_folder(selected\_folder, confirm\_window)).pack(side="right", padx=20,

pady=10) # Delete button

# Function to delete folder

def delete\_folder(folder\_name, confirm\_window):

try:

shutil.rmtree(folder\_name) # Delete folder

messagebox.showinfo("Info", f"Folder {folder\_name} deleted successfully.") # Show success

confirm\_window.destroy() # Close window

show\_delete\_screen() # Refresh delete screen

refresh\_camera\_feed() # Refresh camera feed

except Exception as e: # Handle errors

messagebox.showerror("Error", f"Error deleting folder {folder\_name}: {e}") # Show error

# Function to refresh camera feed

def refresh\_camera\_feed():

show\_camera\_screen() # Show camera screen

# Function to show camera screen

def show\_camera\_screen():

for widget in root.winfo\_children(): # Clear current widgets

widget.destroy()

# Main container with proper weight distribution

main\_container = tk.Frame(root) # Create main container

main\_container.pack(fill=tk.BOTH, expand=True) # Pack container

# Title frame (10% of vertical space)

title\_frame = tk.Frame(main\_container, height=int(root.winfo\_height() \* 0.1)) # Create title frame

title\_frame.pack(fill=tk.X, pady=5) # Pack frame

title\_frame.pack\_propagate(0) # Prevent frame from resizing to contents

tk.Label(title\_frame, text="Camera", font=("Arial", 24)).pack() # Add title

# Camera frame (60% of vertical space - reduced by 20% from original)

camera\_container = tk.Frame(main\_container, height=int(root.winfo\_height() \* 0.6)) # Create camera container

camera\_container.pack(fill=tk.BOTH, expand=True) # Pack container

camera\_container.pack\_propagate(0) # Prevent resizing

global camera\_frame # Access global variable

camera\_frame = tk.Label(camera\_container) # Create label for camera feed

camera\_frame.pack(fill=tk.BOTH, expand=True) # Pack label

# Button frame (30% of vertical space)

button\_frame = tk.Frame(main\_container, height=int(root.winfo\_height() \* 0.3)) # Create button frame

button\_frame.pack(fill=tk.X, pady=10) # Pack frame

button\_frame.pack\_propagate(0) # Prevent resizing

# Buttons with proper spacing

btn\_frame = tk.Frame(button\_frame) # Create frame for buttons

btn\_frame.pack(expand=True) # Pack frame

# Add buttons

tk.Button(btn\_frame, text="Add/Shear", font=("Arial", 14),

command=show\_add\_photos\_screen).pack(side=tk.LEFT, padx=20, pady=10) # Add/Shear button

tk.Button(btn\_frame, text="Back", font=("Arial", 14),

command=show\_main\_menu).pack(side=tk.LEFT, padx=20, pady=10) # Back button

tk.Button(btn\_frame, text="Delete", font=("Arial", 14),

command=show\_delete\_screen).pack(side=tk.LEFT, padx=20, pady=10) # Delete button

start\_camera\_feed() # Start camera feed

# Function to start camera feed with face recognition

def start\_camera\_feed():

global face\_data, picam2 # Access global variables

face\_data = [] # Reset face data

for folder in os.listdir(): # Load face data from all folders

if os.path.isdir(folder) and os.path.exists(f"{folder}/face\_data.pkl"): # If face data exists

with open(f"{folder}/face\_data.pkl", "rb") as f: # Open file

data = pickle.load(f) # Load data

face\_data.append(data) # Add to face data

if picam2 is None: # If camera not initialized

picam2 = Picamera2() # Initialize camera

picam2.configure(picam2.create\_preview\_configuration()) # Configure

picam2.start() # Start camera

def update\_camera\_feed(): # Function to update camera feed

frame = picam2.capture\_array() # Capture frame

if frame is not None: # If frame captured

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB) # Convert to RGB

try:

detected\_faces = DeepFace.extract\_faces(rgb\_frame, detector\_backend="mtcnn") # Detect faces

for face in detected\_faces: # Process each face

x = face["facial\_area"]["x"] # Get x coordinate

y = face["facial\_area"]["y"] # Get y coordinate

w = face["facial\_area"]["w"] # Get width

h = face["facial\_area"]["h"] # Get height

face\_region = frame[y:y + h, x:x + w] # Extract face region

if face\_region.shape[2] == 4: # If RGBA

face\_region = cv2.cvtColor(face\_region, cv2.COLOR\_RGBA2RGB) # Convert to RGB

try:

face\_embedding = np.array(

DeepFace.represent(face\_region, model\_name="ArcFace", enforce\_detection=False)[0][

"embedding"]) # Get embedding

face\_embedding = face\_embedding / np.linalg.norm(face\_embedding) # Normalize

match = False # Match flag

name = "UNKNOWN" # Default name

for data in face\_data: # Compare with known faces

for encoding in data["encodings"]: # For each encoding

encoding = np.array(encoding) # Convert to numpy array

encoding = encoding / np.linalg.norm(encoding) # Normalize

distance = np.linalg.norm(encoding - face\_embedding) # Calculate distance

if distance < 0.6: # If match found

match = True # Set flag

name = data["name"] # Set name

break # Break loop

if match: # If match found

break # Break loop

color = (0, 255, 0) if match else (255, 0, 0) # Set color (green for known, red for unknown)

cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2) # Draw rectangle

cv2.putText(frame, name, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, color, 2) # Add text

if not match: # If unknown face

unknown\_face\_path = "unknown\_face.jpg" # Create filename

cv2.imwrite(unknown\_face\_path, face\_region) # Save image

handle\_unknown\_face(unknown\_face\_path) # Handle unknown face

except Exception as e: # Handle errors

print(f"Error processing face: {e}") # Print error

except Exception as e: # Handle errors

print(f"Error detecting faces: {e}") # Print error

img = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)) # Convert to PIL Image

img.thumbnail((512, 384)) # Create thumbnail (20% reduction)

photo = ImageTk.PhotoImage(img) # Convert to PhotoImage

camera\_frame.config(image=photo) # Update label

camera\_frame.image = photo # Keep reference

camera\_frame.after(10, update\_camera\_feed) # Schedule next update

update\_camera\_feed() # Start camera feed

# Function to show phone number entry screen

def show\_enter\_phone\_number\_screen():

for widget in root.winfo\_children(): # Clear current widgets

widget.destroy()

tk.Label(root, text="Enter Phone Number", font=("Arial", 24)).pack(pady=20) # Add title

phone\_entry = tk.Entry(root, font=("Arial", 14)) # Add entry

phone\_entry.pack(pady=20) # Pack entry

def confirm\_phone\_number(): # Nested function for confirmation

phone\_number = phone\_entry.get().strip() # Get phone number

if phone\_number: # If phone number provided

messagebox.showinfo("Info",

"Please start a chat with the bot on Telegram to link your phone number.") # Show info

chat\_id = get\_chat\_id\_from\_updates() # Get chat ID

if chat\_id: # If chat ID found

save\_phone\_number\_and\_chat\_id(phone\_number, chat\_id) # Save data

messagebox.showinfo("Success", "Phone number and Chat ID saved successfully!") # Show success

show\_main\_menu() # Show main menu

else: # If no chat ID

messagebox.showerror("Error", "No Chat ID found. Please start a chat with the bot.") # Show error

else: # If no phone number

messagebox.showerror("Error", "Please enter a valid phone number.") # Show error

tk.Button(root, text="Confirm", font=("Arial", 14), command=confirm\_phone\_number).pack(

pady=10) # Add confirm button

tk.Button(root, text="Back to Main Menu", font=("Arial", 14), command=show\_main\_menu).pack(side="bottom",

pady=20) # Add back button

# Function to show main menu

def show\_main\_menu():

for widget in root.winfo\_children(): # Clear current widgets

widget.destroy()

tk.Label(root, text="Main Menu", font=("Arial", 24)).pack(pady=20) # Add title

button\_frame = tk.Frame(root) # Create frame for buttons

button\_frame.pack(expand=True) # Pack frame

# Define buttons

buttons = [

("Add/Shear Photos", show\_add\_photos\_screen), # Add/Shear button

("Delete Photos", show\_delete\_screen), # Delete button

("Main Camera", show\_camera\_screen), # Camera button

("Enter Phone Number", show\_enter\_phone\_number\_screen) # Phone number button

]

# Create buttons

for text, command in buttons: # For each button definition

tk.Button(button\_frame, text=text, font=("Arial", 18), width=20, height=2,

command=command).pack(pady=10) # Create and pack button

# ----------------- Main Program -----------------

root = tk.Tk() # Create main window

root.title("Face Recognition App") # Set title

# Get screen dimensions and set main window size

screen\_width = root.winfo\_screenwidth() # Get screen width

screen\_height = root.winfo\_screenheight() # Get screen height

window\_width = int(screen\_width \* 0.8) # Set window width (80% of screen)

window\_height = int(screen\_height \* 0.8) # Set window height (80% of screen)

x\_position = int((screen\_width - window\_width) / 2) # Calculate x position

y\_position = int((screen\_height - window\_height) / 2) # Calculate y position

root.geometry(f"{window\_width}x{window\_height}+{x\_position}+{y\_position}") # Set window geometry

show\_main\_menu() # Show main menu

root.mainloop() # Start main event loop