WATCHBOT: AI Powered Camera

Why am I making it

While there are many cameras that can detect human presence or objects, most of these devices offer limited customisation and low-level intelligence. They typically provide generic labels like “person detected,” without any deeper context or understanding of who the person is. Furthermore, commercial options are expensive and are not flexible for specific needs like home security, personalised authentication, or custom object detection. The most affordable solutions are often limited in functionality and require cloud-based processing, which raises concerns about data privacy and latency.

This project provides an affordable, local AI-powered object and person recognition system using a Raspberry Pi 5 and custom face recognition. The key innovations:

- local processing for real-time identification of people or objects, removing the need for internet connectivity or reliance on cloud services
- Instead of just detecting a “human” or “object,” the system can detect specific individuals by name and also classify non-human objects, giving more detailed and relevant insights.
- Unlike expensive alternatives, this solution provides customisability (users can upload images and teach the system to recognise different people), while maintaining privacy and being self-contained.

The system is ideal for applications like personalised access control (e.g., in smart homes), personal security, or even creating a custom smart camera system that fits specific needs without hefty costs.

Analysis of the Similar System

Before developing WATCHBot, I researched three existing security camera solutions to understand what is currently available and identify gaps that my project could fill.

1. Amazon Ring Doorbell Camera
Amazon Ring is one of the most popular consumer security cameras. It provides motion-triggered alerts and live video streaming through a mobile app. The device relies entirely on cloud-based processing, meaning all video data is sent to Amazon's servers.

Strengths:
- Wide availability and strong brand recognition
- Mobile app with push notifications
- Integration with Alexa smart home ecosystem
- Good video quality (1080p)

Weaknesses:
- Requires a Ring Protect subscription (from £3.49/month) for video history and person detection
- Person detection only provides generic "person detected" alerts and cannot identify who the person actually is
- Completely dependent on internet connection, making it non-functional without Wi-Fi
- Privacy concerns as all video data is stored on Amazon's cloud servers
- No ability for users to customise the detection to recognise specific individuals

2. Google Nest Cam
Google Nest Cam is a competitor to Amazon Ring, offering indoor and outdoor camera options. It uses Google's machine learning to distinguish between people, animals, and vehicles.

Strengths:

- Can differentiate between people, animals, and vehicles with a subscription
- Good integration with Google Home ecosystem
- 3 hours of free cloud event recording

Weaknesses:

- Nest Aware subscription required for full features (from £5.99/month), costing over £70 per year
- Facial recognition ("Familiar Face detection") only available on the most expensive subscription tier
- Entirely cloud-dependent with video processed on Google's servers
- Users cannot manually add faces; the system learns on its own over time, which is slow and often inaccurate
- Significant privacy concerns as Google processes and stores facial data externally

3. Frigate NVR (Open-Source Solution)
Frigate is an open-source Network Video Recorder that runs locally and uses AI-based object detection. It is designed for users who are comfortable with technical setup.

Strengths:

- Runs entirely locally with no cloud dependency
- Free and open-source with no subscription fees
- Supports hardware acceleration using Google Coral TPU
- Can detect people, cars, animals, and other objects

Weaknesses:

- Requires significant technical expertise to set up (Docker, YAML configuration, MQTT)
- No built-in facial recognition - it can detect a person is present but cannot identify who they are
- No graphical user interface for non-technical users

- Requires a separate home server or NAS to run, adding to hardware costs

How My Project Solves These Issues

- **Not having Internet Dependency:** Unlike Ring and Nest, WATCHBot will operate entirely offline, processing data locally on the Raspberry Pi 5. This means it remains functional even when Wi-Fi is unavailable.
- **Affordable:** There are no subscription fees. The total hardware cost is approximately £80 for the Raspberry Pi and camera, compared to Ring and Nest which require a device purchase plus ongoing monthly payments.
- **Personalised Detection:** Users can upload images and teach the system to recognise specific individuals, providing alerts like "Rishi Detected" rather than just "Person Detected." This is something Ring cannot do at all, Nest only offers on expensive tiers, and Frigate does not support.
- **Simple Interface:** Unlike Frigate which requires command-line setup, WATCHBot provides a graphical interface with clear buttons, making it accessible to non-technical users as my stakeholder interview identified.

These key points make my security system a better alternative for users who value privacy, affordability, and personalised detection.

What Technology to Use

To develop my security detector which could be place to view a person object or a relative whose data have been stored I need specific tools and language to use to make this project or product:

Materials (Hardware):

1. Raspberry Pi 5 (16GB)
   - This will run my AI models and handle HDMI display.

2. MicroSD Card (32GB or higher)
- To store my OS and AI model.
- Class 10 or UHS-I recommended for speed.

3. Raspberry Pi Camera Module or USB Webcam
- Use official Raspberry Pi Camera v2 or a decent USB webcam.
- Make sure it supports at least 720p resolution.

4. HDMI Cable + Monitor/TV
- To display detection results directly.
- Your Pi will output the detections live to this screen.

5. Power Supply for Raspberry Pi
- Official 5V 3A USB-C power supply is recommended.

6. Keyboard & Mouse
- Useful for setup if you're not SSH-ing remotely.

7. Case with Fan or Heatsinks
- Keeps the Pi cool when running AI inference.

8. Raspberry Pi AI HAT+(26 tops)
- This will increase the runtime of algorithm and program a lot faster

System Software & Tools:

A. Operating System
- Raspberry Pi OS (32-bit), accessed via VNC to cast to my laptop instead of connecting an HDMI cable to a monitor each time
- PostgreSQL: To store user credentials, face embeddings, and detection history

B. Programming Language
- Python 3 - chosen because it has the strongest ecosystem for machine learning and computer vision, and is the language I am most proficient in.

C. AI & Computer Vision Libraries
For face detection:

- facenet-pytorch(MTCNN) - Multi-task Cascaded Convolutional Networks, a deep learning model specifically designed for detecting faces in images. I initially considered using YOLO for detection, but MTCNN is purpose-built for faces and comes packaged with FaceNet in a single library, making it more efficient for my use case.

For face recognition:

- facenet-pytorch(InceptionResnetV1) - The FaceNet model, pre-trained on the VGGFace2 dataset containing 3.3 million images of over 9,000 individuals. It converts a face image into a 512-dimensional embedding vector, which can then be compared against stored embeddings to identify a person. I chose this over the face_recognition library (which uses HOG + dlib with 128-dimensional encodings) because the 512-dimensional vector captures more facial detail and performs better in testing.

For face matching:

- **Cosine Similarity** (via PyTorch) - To compare a live face embedding against stored embeddings, I use cosine similarity rather than Euclidean distance. Cosine similarity measures the angle between two vectors, making it more robust to differences in lighting and contrast. A threshold of 0.75 is used - if the similarity score exceeds this, the face is classified as a match.

Supporting libraries:

- PyTorch- The deep learning framework required to run both MTCNN and FaceNet models
- OpenCV (cv2)- For capturing video frames from the camera, converting colour spaces, and drawing bounding boxes on detected faces

- Pillow (PIL)- For opening and processing uploaded face images and rendering images in the GUI
- Numpy- For underlying array and tensor operations required by PyTorch and OpenCV

D. Database

PostgreSQL was chosen over SQLite because it handles concurrent connections better, which is important when the live feed is logging detections to the database at the same time the user may be browsing their detection history. It also supports array data types natively, which is useful for storing the 512-dimensional face embedding vectors.

E. GUI Framework

Tkinter - Built into Python's standard library, requiring no additional installation. It is suitable for building the multi-window desktop application my user needs, with simple buttons and navigation as identified in the stakeholder interview.

F. Security
bcrypt- Used to hash user passwords before storing them in the database. Bcrypt automatically applies a random salt and is computationally expensive by design, making brute-force attacks impractical. No plaintext passwords are ever stored.

Computational Method

Before starting development, I considered how to break down the problem using key computational thinking concepts.

Decomposition

The overall problem of "build a smart security camera" is too large to tackle at once, so I will break it down into smaller sub-problems:

- Capturing live video from the camera
- Detecting faces within each video frame
- Comparing detected faces against stored data to identify them
- Storing and managing face data in a database
- Logging detection events with timestamps
- Building a user interface for non-technical users
- Handling user accounts and security

Each of these can be worked on separately and then brought together into the final system.

Abstraction

Rather than comparing raw images pixel by pixel (which would be slow and inaccurate), I will reduce each face down to a vector of 512 numbers using the FaceNet model. This removes unnecessary detail like background, lighting, and image size, keeping only the key features that make a face unique. This makes comparison much faster and more reliable.

Algorithmic Thinking

To match a live face against stored faces, I will use cosine similarity to compare their vectors. The system will loop through all known faces, calculate a similarity score for each, and pick the best match. If the best score is above 0.75, the person is identified - otherwise they are

labelled as "Unknown." I will also need a debounce mechanism so that the same person is not logged to the database repeatedly every frame - a 30-second cooldown window will prevent this.

Pattern Recognition

The FaceNet model has already been trained on millions of face images, meaning it has learned the common patterns that distinguish one face from another. By using this pre-trained model rather than training my own, I can take advantage of these learned patterns without needing a massive dataset of my own.

Identification Of User's Need

To identify the specific requirements for my system, I interviewed my primary end user - my father, who would be using WATCHBot as a home security device. He represents the target audience for this product: someone who wants reliable home security with facial recognition but finds commercial options either too expensive, too limited, or too dependent on cloud services. The following are the questions and responses I gathered from the interview:

Q1: How important is it to know who specifically is at the door rather than just that someone is there?

"Very important. If it's a family member or a neighbour I don't need to worry, but if it's someone I don't recognise then I'd want to know about it. Just saying 'person detected' isn't very helpful."

Q2: Would you prefer the system to work offline or are you comfortable with cloud-based processing?

"I'd prefer it to work without needing the internet. Our Wi-Fi goes down sometimes and I wouldn't want the camera to stop working

because of that. Also I'm not keen on our family's faces being stored on some company's server.”

Q3: How comfortable are you with facial data being stored on the device locally?

"If it's just on the device in our home and not being sent anywhere, I'm fine with that. As long as it's password protected so not just anyone can access it.”

Q4: What features would matter most to you?

"Being able to see a live feed is the main one. But I'd also want to look back and see who was detected and when - like a log. And it needs to be easy to add new people, like if a relative visits regularly.”

Identified Needs

From this interview and my own research into the limitations of existing products, I identified the following user needs:

1. **Personalised face recognition** - The system must identify specific individuals by name, not just detect generic human presence. This was the user's primary frustration with commercial cameras that only display "Person Detected.”

2. **Offline and local operation** - The system must function without an internet connection and process all data locally, both for reliability when Wi-Fi is unavailable and for privacy by keeping facial data off external servers.

3. **Detection history logging** - The user needs to review a log of who was detected and when, so they can check past activity when they were not watching the live feed.

4. **Password-protected access** - Only authorised users should be able to access the system and the stored facial data, ensuring household members' biometric information is secure.

5. **Easy face enrolment** - The user must be able to upload images of new people such as family members and regular visitors without requiring any technical knowledge.

These needs confirm that there is a genuine gap between what commercial solutions currently offer and what this user requires. My project will directly address each of these needs, and they will form the basis of my measurable success criteria.

Acceptable limitations

The device won't be a perfect replica of the real products which are in the market and the main limitation for the project are:

- No notification and alerts: This allows the user to get the status with the help of email and messages. This all goes to a cloud and then to the user

- Limited Data logging: There would be limited data logging which store the past information and allows the user to access it

- No Real-Multi Person Tracking: The system my sometimes struggle to recognise multiple people but that can be resolved.

- Recognition Accuracy in Challenging Conditions: While the camera would be night-visioned it will still might struggle with the detection at night and because of this it could affect the detection accuracy

Proposed Solution For The Limitations

To overcome this limitations, the best way is to add new features which would help solve this problem:

- Even thought it won't have cloud storage where it stores the data into the cloud for future reference it would still have SQL database where it would contain information on what time does a person or an object appears in front of the camera

- The condition for real-multi person Tracking can be solved by adding Raspberry Pi AI HAT+ (Hardware Attached on Top). The Raspberry Pi AI HAT+ add-on board has a built-in Hailo AI accelerator compatible with Raspberry Pi 5. The one to use is the The 26 TOPS variant which can run larger networks, can run networks faster, and can more effectively run multiple networks simultaneously. This can help increase the program much faster and should affect the limitation by a lot.

Privacy and Data Security

Since my project handles facial data, I need to consider how the system will protect personal information.

- All data will be processed and stored locally on the Raspberry Pi. No images or face data will be sent to any external server or cloud service, unlike commercial products like Ring and Nest.
- The system will not store raw face images. Instead, each face will be converted into a numerical vector (embedding) and only that will be stored in the database.
- User passwords will be hashed using bcrypt before being stored, meaning no plaintext passwords are ever saved.
- Before creating an account, the user will be shown a Terms and Agreement dialog explaining how the system uses camera data and how their information is protected. The user must agree before they can proceed.

- Each user account will only have access to their own stored faces and detection history, so one user cannot see another user's data.

Final Proposed Objectives

Based on the user needs identified from the stakeholder interview and my research into existing solutions, the following success criteria define what WATCHBot must achieve. Each criterion is specific and measurable so that it can be tested and evaluated once the system is built.

1. The system shall detect faces in a live video feed and draw a bounding box around each detected face in real time.
2. The system shall identify a known person by name with a confidence score above 0.75, displaying their name and confidence score on screen.
3. The system shall label any unrecognised face as "Unknown" and display a red bounding box, while known faces are shown with a green bounding box.
4. The system shall allow users to upload images of a new person and store their face embedding in the database without requiring any technical knowledge.
5. The system shall store all detection events (person name, known/unknown status, confidence score, and timestamp) in a PostgreSQL database automatically.
6. The system shall not log duplicate detections of the same person within a 30-second window to prevent the database from being flooded with repeated entries.
7. The system shall provide a detection history view where the user can see all past detections in a table sorted by date and time, with the option to delete the history.
8. The system shall require users to create an account with a username and password, where the password is securely hashed using bcrypt before being stored.

9. The system shall display an ethical terms and agreement dialog before account creation, requiring the user to consent before proceeding.
10. The system shall operate entirely offline with no internet connection or cloud services required, processing all data locally on the Raspberry Pi.
11. The system shall provide a simple graphical user interface built with Tkinter, navigable through buttons with no command-line interaction required by the user.
12. The system shall allow users to view and remove stored faces from the database through the GUI.

Database Design (ER Diagrams)

The system will use a PostgreSQL database with three tables to store user accounts, face embeddings, and detection history. The structure is shown in the Entity-Relationship diagram below.

login_details

- user_id (Primary Key, auto-generated)
- username (Text)
- hash_values (Text - stores the bcrypt hashed password)

embedding_table

- user_id (Foreign Key → login_details.user_id)
- person_name (Text)
- embedding (Array of 512 floating point numbers)

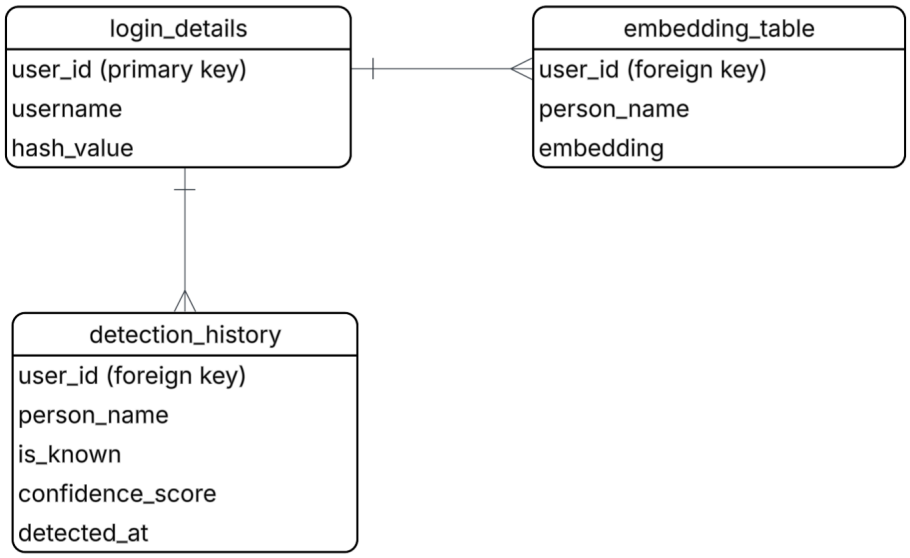
detection_history

- user_id (Foreign Key → login_details.user_id)
- person_name (Text)
- is_known (Boolean)
- confidence_score (Float)
- detected_at (Timestamp, auto-generated)

Relationships:

- One user can have **many** face embeddings (one-to-many) - because a user can add multiple people to their database
- One user can have **many** detection history records (one-to-many) - because the system logs every detection event for that user

The user_id field links all three tables together, ensuring that each user only sees their own stored faces and their own detection history.



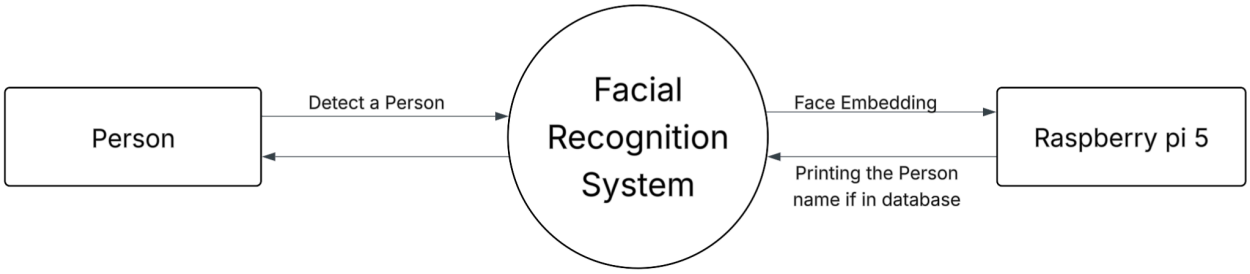
DFD Models

The current system consists of the pi camera interacting or detecting the object or a person in front of the camera and try recognising it and the different scenario which can show up are:

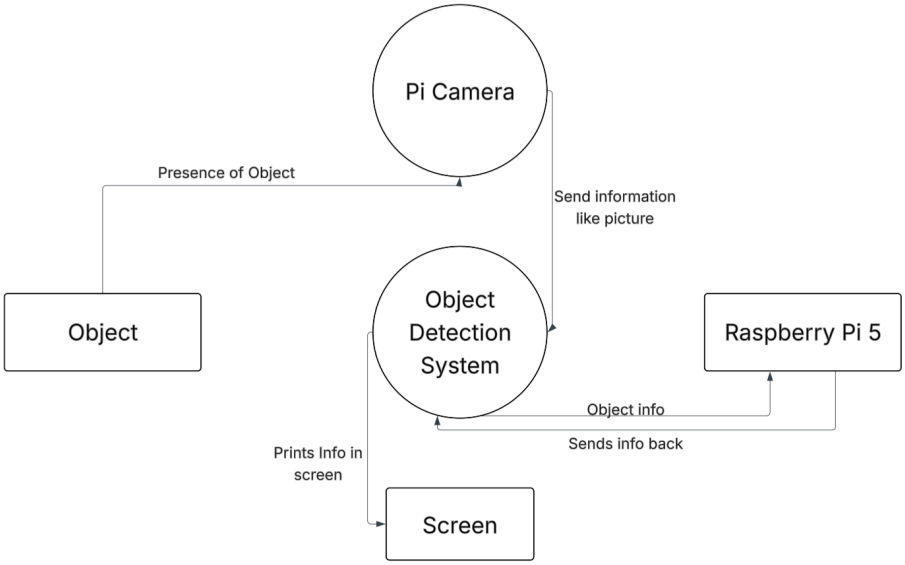
1. Camera detects an object :

- The pi camera first detect the object and then run the detection program to check if there is information in the SQL or database

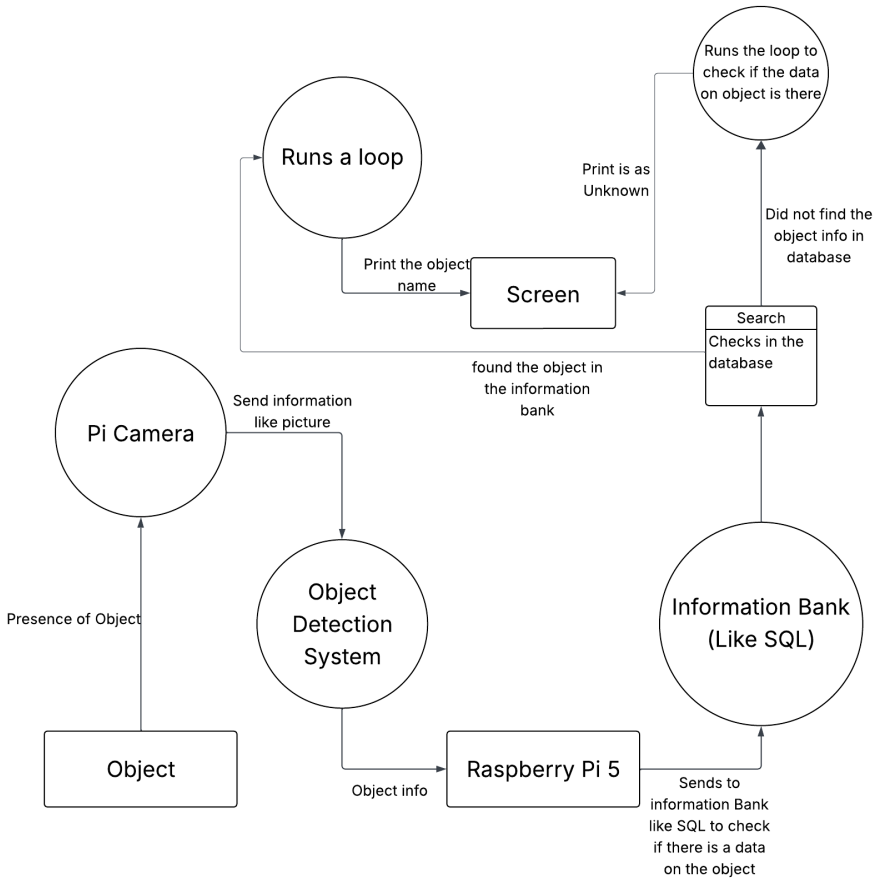
- 0 Level Data Flow Mode:



- 1 Level Data Flow Model

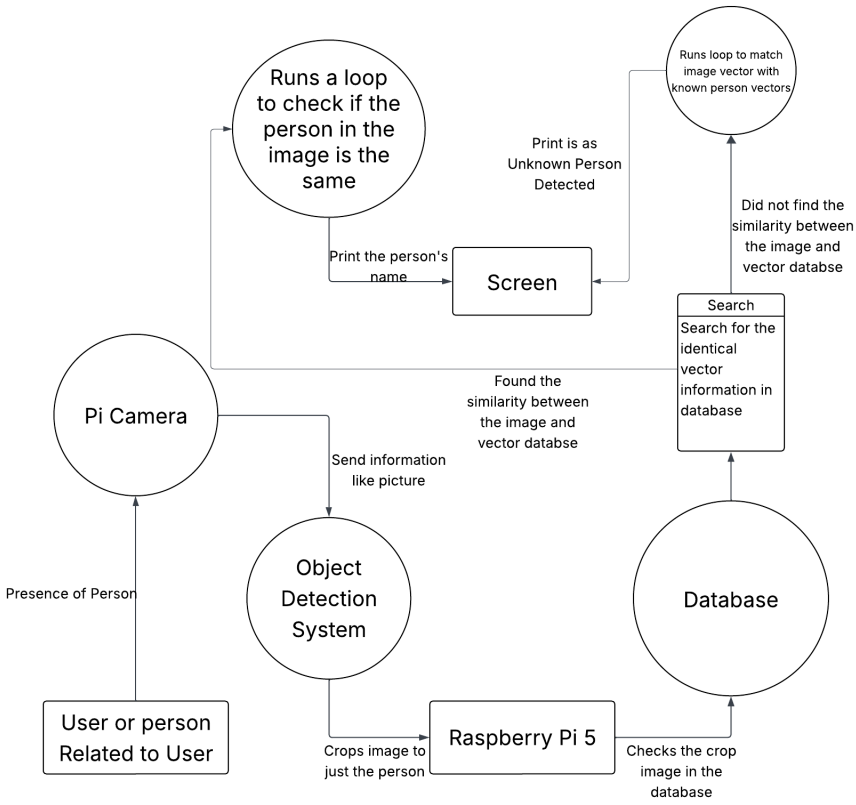


- 2 Level Data Flow Model



2. Camera detects a person

- User and User related and Unknown Person



- Suspicious Person