

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN



**HỆ THỐNG TỦ KHÓA THÔNG
MINH SỬ DỤNG NHẬN DIỆN
KHUÔN MẶT**

Lường Duy Thái-22001284
Lê Vũ Ngọc Anh-22001230
Hoàng Trung Kiên-22001266
Trần Văn Lâm-22001268

Mã học phần: MAT3508
Học kỳ 1, Năm học 2025-2026

Thông tin Dự án

Học phần:	MAT3508 – Nhập môn Trí tuệ Nhân tạo
Học kỳ:	Học kỳ 1, Năm học 2025-2026
Trường:	VNU-HUS
Tên dự án:	Hệ thống tủ khóa thông minh sử dụng nhận dạng khuôn mặt
Ngày nộp:	30/11/2025
GitHub:	Link GitHub

Thành viên nhóm

Họ tên	MSSV	GitHub	Đóng góp
Lường Duy Thái	22001284	duythai04	<ul style="list-style-type: none">Viết Chương 1 – Giới thiệu: mục tiêu, bài toán, tổng quan hệ thống.Phụ trách mô hình YOLO phát hiện khuôn mặt (Face Detection).Tích hợp mô hình AI vào Smart Locker và kết nối camera.
Lê Vũ Ngọc Anh	22001230	LeeAnh836	<ul style="list-style-type: none">Viết Chương 2 – Cơ sở lý thuyết: YOLO, CNN, nhận diện khuôn mặt.Phụ trách mô hình Face Embedding (trích xuất đặc trưng).Phân tích phương pháp so khớp đặc trưng (cosine similarity).
Hoàng Trung Kiên	22001266	trungkien5s	<ul style="list-style-type: none">Phụ trách mô hình YOLO phát hiện người (Person Detection).Viết Chương 3 – Thực nghiệm và đánh giá kết quả cho cả 3 mô hình.Thiết kế cơ sở dữ liệu và API của hệ thống Smart Locker.
Trần Văn Lâm	22001268	tranvanlam2004	<ul style="list-style-type: none">Phụ trách Xử lý dữ liệu: thu thập, gán nhãn, làm sạch, chuẩn hóa.Xây dựng bộ dữ liệu huấn luyện và kiểm thử cho 3 mô hình.Viết Chương 4 – Kết luận và hướng phát triển.

Mục lục

1 Giới thiệu	7
1.1 Bối cảnh	7
1.2 Lý do chọn đề tài	7
1.3 Bài toán đặt ra	8
1.4 Mục tiêu	8
1.5 Phạm vi nghiên cứu	8
2 Phương pháp & Triển khai	10
2.1 Cơ sở lý thuyết	10
2.2 Dữ liệu sử dụng	11
2.3 Mô hình hệ thống	13
2.3.1 Client (camera + giao diện người dùng)	13
2.3.2 Server xử lý AI	13
2.3.3 Cơ sở dữ liệu MongoDB	13
2.3.4 ESP32 Điều Khiển Tủ	15
2.4 Công nghệ sử dụng	15
2.4.1 Mô hình YOLO	15
2.4.2 Backend	16
2.4.3 Frontend	16
2.4.4 Phần cứng	16
2.5 Quy trình triển khai	17
2.5.1 Kiến trúc tổng thể hệ thống	17
2.5.2 Sơ đồ Use Case tổng quan	17
2.5.3 Thiết kế cơ sở dữ liệu	18
2.5.4 Luồng sử dụng của hệ thống	18
2.5.5 Mô hình YOLOv8	19
2.5.6 Mô hình Nhúng (Face Embedding Model)	20
2.5.7 Xây dựng API backend	21
2.5.8 Tích hợp phần cứng	22
2.5.9 Kiểm thử thực tế	22
2.6 Cấu trúc mã nguồn dự án	24
3 Kết quả & Phân tích	26
3.1 Kết quả mô hình phát hiện người (Person Detection)	26
3.1.1 Chỉ số định lượng	26
3.2 Kết quả mô hình nhận diện khuôn mặt (Face Detection)	27
3.2.1 Chỉ số định lượng	27
3.2.2 Phân tích kết quả	27
3.3 Kết quả mô hình nhúng khuôn mặt	27
3.3.1 Kết quả thực nghiệm và phân tích	27
3.3.2 Triển khai và tối ưu hóa	28
3.4 Thực nghiệm ứng dụng	28
3.4.1 Nhận xét chung về kết quả thực nghiệm hệ thống	28
4 Kết luận & Hướng phát triển	30
4.1 Kết luận chung	30
4.2 Hạn chế	30
4.3 Hướng phát triển tương lai	30

Tài liệu tham khảo	31
A Phụ lục	33
A.1 Mã nguồn chương trình	33
A.2 Hướng dẫn cài đặt	33
A.3 Ảnh minh họa	33

Danh sách hình vẽ

1.1	Mô phỏng hệ thống tủ thông minh sử dụng nhận diện khuôn mặt.	8
1.2	Quy trình tổng quan của một hệ thống nhận dạng khuôn mặt.	9
1.3	Tổng quan hệ thống từ camera → xử lý AI → phần cứng mở tủ	9
2.1	Giao diện người dùng	13
2.2	Vị điều khiển và động cơ servo	16
2.3	Kiến trúc hệ thống	17
2.4	Sơ đồ use case cho hệ thống	17
2.5	Mô hình dữ liệu	18
3.1	Kết quả demo web (1)	28
3.2	Kết quả demo web (2)	29
3.3	Kết quả demo web (3)	29
A.1	Giao diện web Smart Locker sử dụng nhận diện khuôn mặt.	33

Danh sách bảng

2.1	Cấu trúc collection <code>lockers</code>	18
2.2	Cấu trúc collection <code>locker_sessions</code>	18
3.1	Quá trình hội tụ của mô hình phát hiện người trên tập validation COCO-person	26
3.2	Kết quả đánh giá mô hình YOLO8n trên tập validation WIDER FACE	27

Chương 1

Giới thiệu

1.1 Bối cảnh

Trong những năm gần đây, sự phát triển của trí tuệ nhân tạo (AI), đặc biệt là lĩnh vực thị giác máy tính (Computer Vision), đã tạo nên bước tiến lớn trong các hệ thống nhận dạng sinh trắc học. Các công nghệ như nhận diện vân tay, mống mắt, giọng nói và khuôn mặt ngày càng được ứng dụng rộng rãi trong đời sống.

Trong số đó, nhận diện khuôn mặt (Face Recognition) nổi bật nhờ tính tiện lợi, tốc độ xử lý nhanh và khả năng hoạt động trên nhiều nền tảng từ thiết bị di động, camera giám sát đến hệ thống nhúng. Công nghệ này đã được ứng dụng trong:

- Các hệ thống kiểm soát ra vào khu vực an ninh.
- Thanh toán không tiếp xúc (Face Payment).
- Xác thực người dùng trong điện thoại, laptop.
- Tìm kiếm và phân tích đối tượng trong camera giám sát.

Trong khi đó, nhu cầu về các hệ thống tủ thông minh (Smart Locker) đang tăng mạnh tại:

- Trường học và ký túc xá.
- Siêu thị, trung tâm thương mại.
- Thư viện, khu làm việc chung.
- Các khu vực gửi đồ tự động.

Hệ thống tủ thông minh dùng nhận diện khuôn mặt giúp người dùng mở tủ nhanh chóng, không cần chìa khóa hay thẻ từ, giảm thiểu rủi ro và tăng trải nghiệm sử dụng.

Nhờ những ưu điểm về tốc độ, sự thuận tiện và khả năng mở rộng, công nghệ nhận diện khuôn mặt đang trở thành nền tảng quan trọng trong các hệ thống tự động hóa hiện đại.

1.2 Lý do chọn đề tài

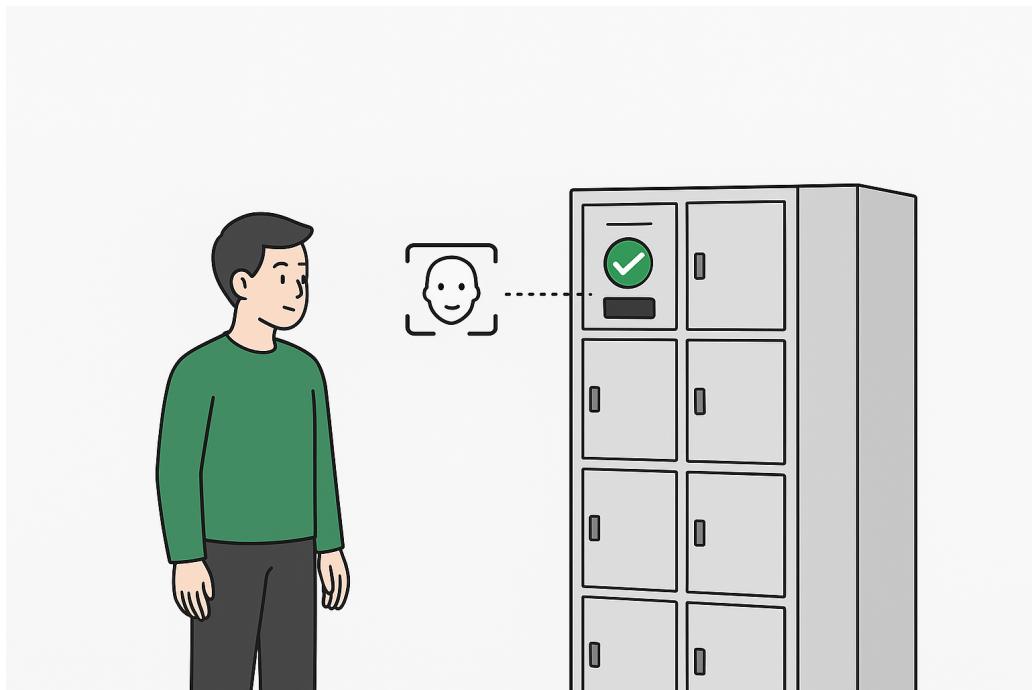
Mặc dù hiện nay đã có nhiều giải pháp mở tủ truyền thống, chúng vẫn tồn tại một số hạn chế nhất định:

- **Chìa khóa cơ:** dễ bị mất, hỏng, hoặc sử dụng nhầm; việc quản lý số lượng lớn chìa khóa rất phức tạp.
- **Mã PIN:** người dùng dễ quên hoặc bị người khác nhìn lén, dẫn đến nguy cơ lộ mã.
- **Thẻ từ / RFID:** có thể bị đánh rơi, trộm hoặc sao chép bằng thiết bị chuyên dụng.

Trong khi đó, sử dụng nhận diện khuôn mặt mang lại các lợi ích:

- **Không cần mang theo bất kỳ vật dụng nào.**
- **Thời gian xác thực nhanh.**
- **Độ chính xác cao,** gần như không thể giả mạo bằng phương pháp thông thường.
- **Quản lý người dùng tập trung và dễ dàng mở rộng.**

Vì những lý do trên, nhóm quyết định lựa chọn đề tài: “**Hệ thống tủ thông minh sử dụng nhận diện khuôn mặt**” nhằm xây dựng mô hình có thể ứng dụng thực tế trong trường học hoặc khu ký túc.



Hình 1.1: Mô phỏng hệ thống tủ thông minh sử dụng nhận diện khuôn mặt.

1.3 Bài toán đặt ra

Đề tài hướng đến việc giải quyết những yêu cầu kỹ thuật cốt lõi sau:

- Phát hiện khuôn mặt (Face Detection) trong thời gian thực từ camera.
- Trích xuất đặc trưng (Embedding Extraction) bằng mô hình học sâu.
- So khớp nhận diện (Face Recognition) 1:N với độ chính xác cao.
- Quản lý dữ liệu người dùng, bao gồm đăng ký, cập nhật và lưu trữ embedding.
- Kết nối hệ thống nhận diện với phần cứng như ESP32, rơ-le, khóa điện tử.
- Đảm bảo hoạt động ổn định trong môi trường thực tế: ánh sáng yếu, nhiều người, góc nghiêng...

Ngoài ra, hệ thống cần có khả năng mở rộng khi số lượng người dùng tăng lên, đồng thời tài nguyên xử lý vẫn đảm bảo tốc độ nhận diện nhanh.

Mục tiêu chính của đề tài bao gồm:

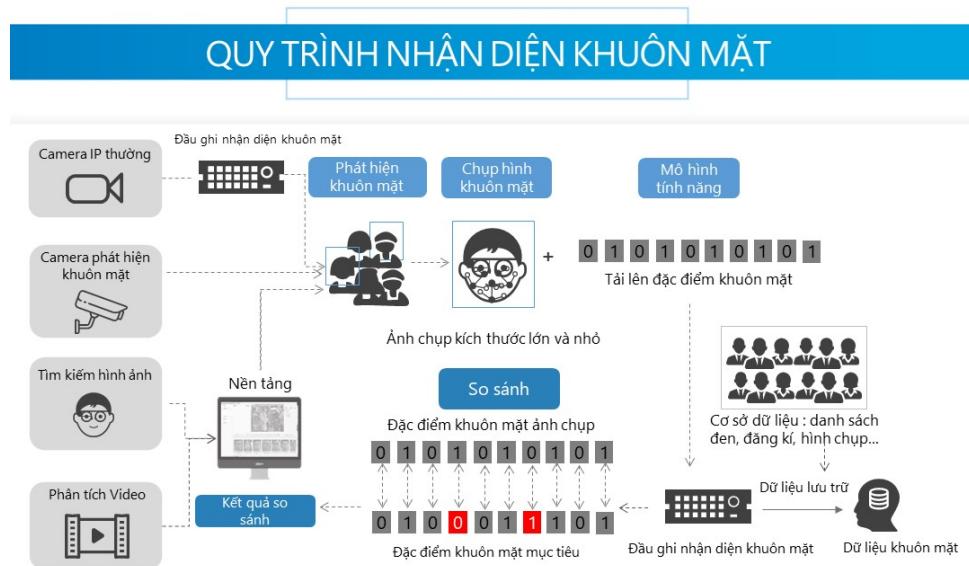
1.4 Mục tiêu

- Xây dựng hệ thống phát hiện và nhận diện khuôn mặt có độ chính xác cao.
- Tạo giao diện đăng ký người dùng và cơ sở dữ liệu quản lý thông tin.
- Triển khai mô hình vào phần cứng để mở khóa tủ tự động.
- Hoàn thiện một demo hệ thống hoàn chỉnh hoạt động trong môi trường thực tế.

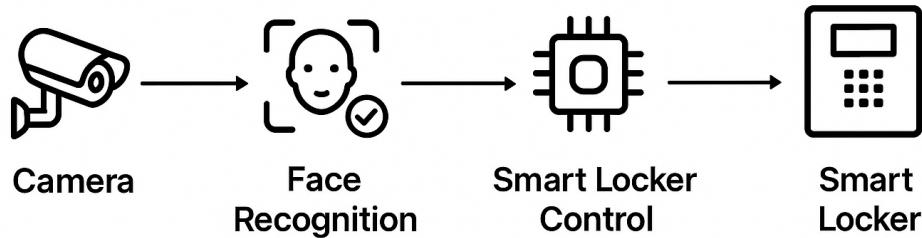
1.5 Phạm vi nghiên cứu

Phạm vi đề tài tập trung vào các nội dung sau:

- Nhận diện khuôn mặt theo hướng 1:N dựa trên vector đặc trưng (embedding).
- Không bao gồm bài toán chống giả mạo (Anti-spoofing).



Hình 1.2: Quy trình tổng quan của một hệ thống nhận dạng khuôn mặt.



Hình 1.3: Tổng quan hệ thống từ camera → xử lý AI → phần cứng mở tủ

- Sử dụng các mô hình học sâu có sẵn như YOLOv8-face, ArcFace.
- Phần cứng điều khiển tủ thông qua ESP32 và khóa điện tử.
- Thử nghiệm trong môi trường mô phỏng thực tế quy mô nhỏ.

Các phần mở rộng như: hệ thống nhiều tủ, kết nối server cloud, giao diện quản trị nâng cao... sẽ được đề cập trong mục hướng phát triển ở chương cuối.

Chương 2

Phương pháp & Triển khai

2.1 Cơ sở lý thuyết

Các khái niệm và kỹ thuật nền tảng:

Phát Hiện Khuôn Mặt (Face Detection)

Phát hiện khuôn mặt là nhiệm vụ cơ bản trong xử lý hình ảnh, nhằm xác định vị trí và kích thước của khuôn mặt trong ảnh hoặc video thông qua *bounding box*. Phương pháp truyền thống như Viola–Jones (2001) [?] dựa trên Haar features và cascade classifiers, nhưng hạn chế về tốc độ và độ chính xác trên dữ liệu phức tạp (occlusion, góc nhìn đa dạng).

Trong hệ thống, sử dụng **YOLOv11n-Face** – một biến thể fine-tune của mô hình YOLOv11 (Ultralytics, 2024) [?] – để thực hiện phát hiện thời gian thực. YOLOv11 là one-stage detector, xử lý toàn bộ ảnh trong một forward pass qua mạng CNN, chia ảnh thành lưới $S \times S$ (với $S = 640$) để dự đoán bounding box và class probability. Công thức dự đoán cho mỗi ô lưới:

$$p_c \cdot \Pr(class) \cdot IoU_{pred}^{truth}$$

Trong đó p_c là confidence score (xác suất có đối tượng), $\Pr(class)$ là xác suất lớp “face”, và IoU đo độ chồng lấp giữa box dự đoán và ground truth.

Ưu điểm: tốc độ cao (~ 30 FPS trên CPU), mAP@0.5 > 0.90 trên WIDER FACE sau fine-tune. Hạn chế: kém chính xác với khuôn mặt nhỏ, được khắc phục bằng cascade (detect người trước).

Nhận Diện Khuôn Mặt (Face Recognition)

Nhận diện khuôn mặt là quá trình xác định danh tính từ ảnh khuôn mặt thông qua việc so sánh đặc trưng (features). Quy trình bao gồm: Detection \rightarrow Alignment \rightarrow Representation \rightarrow Matching. Theo Huang et al. (2019) [?], độ chính xác phụ thuộc lớn vào chất lượng embedding và metric so sánh.

Trong hệ thống, nhận diện được tiến hành sau detection bằng cách trích xuất embedding từ vùng mặt crop, sau đó so sánh với database thông qua threshold (similarity > 0.6). Hiệu suất được đánh giá bằng FAR và FRR, với accuracy $\sim 95\%$ trên benchmark LFW (Labeled Faces in the Wild) [?].

Ưu điểm: ứng dụng rộng rãi trong xác thực sinh trắc học. Hạn chế: nhạy với biến thiên (tuổi tác, makeup), được giảm bằng data augmentation.

Mô Hình Sinh Embedding: ArcFace

ArcFace (Deng et al., 2019) [?] là mô hình sinh embedding (vector 512 chiều) cho nhận diện khuôn mặt, dựa trên *additive angular margin loss* để tăng tính phân biệt giữa các lớp. Thay vì softmax loss thông thường, ArcFace thêm margin m vào góc giữa embedding và trọng số lớp:

$$L = -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{s(\cos(\theta_{y_i} + m))}}{e^{s(\cos(\theta_{y_i} + m))} + \sum_{j \neq y_i} e^{s \cos \theta_j}}$$

Trong hệ thống, ArcFace (qua InsightFace) được sử dụng để tạo embedding từ ảnh crop, huấn luyện trên MS1M dataset ($\sim 10M$ ảnh). Ưu điểm: tăng intra-class compactness và inter-class separability, accuracy đạt 99.8% trên LFW. Hạn chế: yêu cầu dataset lớn, được khắc phục bằng pre-trained model.

Phép Cosine Similarity

Phép cosine similarity đo độ tương đồng giữa hai vector embedding \mathbf{a} và \mathbf{b} dựa trên góc giữa chúng:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \cos \theta$$

Giá trị nằm trong $[-1, 1]$ (1: giống hệt, 0: vuông góc). Trong hệ thống, ngưỡng (threshold) 0.6 được sử dụng để xác định hai khuôn mặt có cùng người hay không [?].

Ưu điểm: không phụ thuộc độ dài vector, phù hợp embedding normalized. Hạn chế: không đo được khoảng cách tuyệt đối → có thể kết hợp với Euclidean.

Kiến Trúc YOLOv8-Face

YOLOv8-Face là fine-tune của YOLOv8n cho lớp “face”, với backbone C3k2 (CSP-based, kernel 3×3) giúp giảm FLOPs (~ 6.6 GFLOPs). Neck sử dụng PANet + SPPF để trích xuất đặc trưng đa tỉ lệ; head dạng decoupled để tách detection và classification. Loss function huấn luyện:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{box} + \lambda_2 \mathcal{L}_{cls} + \lambda_3 \mathcal{L}_{dfl}$$

(DFL = Distribution Focal Loss).

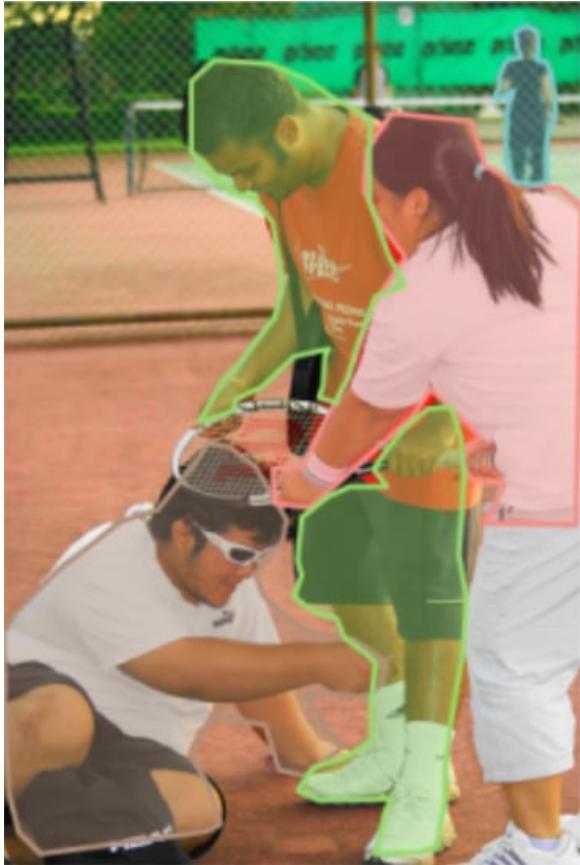
Trong hệ thống, mô hình detect khuôn mặt trong ROI (crop từ person) và được export sang ONNX để suy luận. Ưu điểm: real-time, dễ deploy; tham khảo [?].

2.2 Dữ liệu sử dụng

Tập dữ liệu cho mô hình nhận diện người

Tập dữ liệu COCO2017 [1] để huấn luyện mô hình nhận diện người và khuôn mặt. COCO (Common Objects in Context) là một trong những bộ dữ liệu chuẩn và phổ biến nhất được sử dụng trong lĩnh vực thị giác máy tính, đặc biệt là các bài toán về nhận diện đối tượng (object detection), phân đoạn ảnh (image segmentation), và nhận diện điểm chính (keypoint detection). Bộ dữ liệu này được phát triển bởi Microsoft và chứa hơn 200.000 ảnh kích thước khác nhau với 1,5 triệu nhãn đối tượng, thuộc 80 lớp đối tượng phổ biến trong cuộc sống hàng ngày như người, xe, động vật, đồ vật gia dụng, v.v. Đặc biệt, các đối tượng trong COCO được chú thích trong ngữ cảnh tự nhiên, nghĩa là các vật thể thường xuất hiện cùng nhau trong khung cảnh đời thực, giúp huấn luyện mô hình nhận diện có độ chính xác cao hơn trong môi trường thực tế.

Áp dụng vào huấn luyện mô hình nhận diện người và khuôn mặt chúng tôi tách và sử dụng lớp đối tượng “Người” gồm 64115 ảnh chứa người. Sau đó, chúng tôi thu gọn dữ liệu còn khoảng 8000 ảnh để tiết kiệm dung lượng.



Tập dữ liệu cho mô hình nhận diện khuôn mặt

Để hỗ trợ huấn luyện mô hình phát hiện và nhận diện khuôn mặt, chúng tôi sử dụng bộ dữ liệu WIDER FACE – một bộ dữ liệu chuẩn chuyên biệt cho bài toán phát hiện khuôn mặt (face detection), được phát triển bởi Yang et al. (2016) [4] từ Microsoft Research. WIDER FACE bao gồm hơn 32.000 ảnh được thu thập từ các sự kiện thực tế đa dạng (như lễ hội, đám đông, thể thao), với hơn 393.000 bounding box chú thích khuôn mặt, phân loại theo mức độ khó (dễ, trung bình, khó) dựa trên biến đổi về quy mô, tư thế và che khuất (occlusion). Bộ dữ liệu được chia thành tập huấn luyện (12.880 ảnh), xác thực (3.226 ảnh) và kiểm tra (16.097 ảnh), giúp đánh giá mô hình trên các kịch bản thực tế phức tạp.

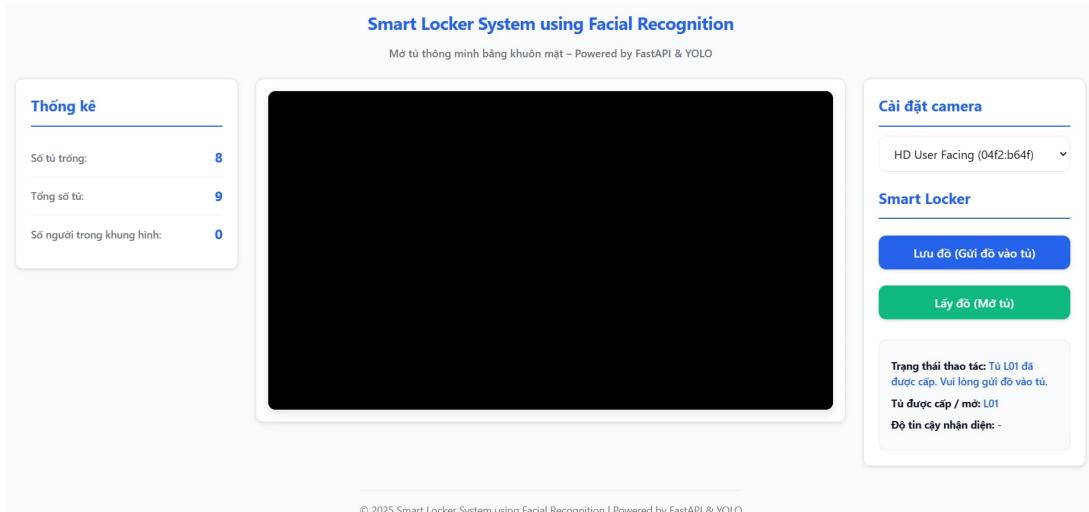
Quá trình thu thập dữ liệu được thực hiện qua việc tải bộ dữ liệu từ nguồn chính thức, bao gồm các file ảnh (ZIP) và annotations (TXT) với định dạng bounding box (x, y, w, h) và thuộc tính (blur, expression, illumination). Để phù hợp với tài nguyên tính toán hạn chế, chúng tôi thu gọn dữ liệu bằng cách chọn ngẫu nhiên 5.000 ảnh từ tập huấn luyện và xác thực (sử dụng công cụ Python với thư viện random và pandas), ưu tiên các ảnh có ít nhất một khuôn mặt rõ ràng (confidence > 0.5) và đa dạng về góc nhìn, đảm bảo tỷ lệ 80% train/20% val. Tổng số khuôn mặt sau thu gọn khoảng 25.000 instances.

Việc xử lý dữ liệu bao gồm các bước sau: (i) Trích xuất bounding box từ annotations và crop khuôn mặt từ ảnh gốc bằng OpenCV để tạo tập ảnh crop riêng (kích thước chuẩn 160x160 pixel); (ii) Tăng cường dữ liệu (data augmentation) bằng các kỹ thuật như xoay ngẫu nhiên ($\pm 15^\circ$), lật ngang, thay đổi độ sáng ($\pm 20\%$), và noise Gaussian để tăng tính tổng quát hóa; (iii) Chuẩn hóa (normalization) pixel về khoảng [0,1] và chuyển sang định dạng YOLO (normalized coordinates cho bounding box); (iv) Loại bỏ dữ liệu nhiễu (ảnh bị che khuất >50% hoặc độ phân giải thấp <100x100). Các bước này được thực hiện bằng script Python với Ultralytics và Albumentations, giảm dung lượng lưu trữ từ 5GB xuống 1GB mà vẫn duy trì độ đa dạng. Kết quả là tập dữ liệu được tối ưu hóa cho fine-tune YOLOv11n-Face, đạt mAP@0.5 khoảng 0.85 trên tập val.

Quá trình này không chỉ tiết kiệm tài nguyên mà còn nâng cao hiệu suất mô hình trong môi trường thực tế, nơi khuôn mặt thường xuất hiện với biến đổi cao.

2.3 Mô hình hệ thống

2.3.1 Client (camera + giao diện người dùng)



Hình 2.1: Giao diện người dùng

2.3.2 Server xử lý AI

Server xử lý AI đóng vai trò trung tâm trong hệ thống nhận diện khuôn mặt điều khiển tủ thông minh. Thành phần này đảm nhiệm toàn bộ các tác vụ liên quan đến thị giác máy tính, phân tích dữ liệu và đối chiếu nhận diện. Các chức năng chính bao gồm:

- Tiếp nhận dữ liệu từ Client:** Server nhận các khung hình (frame) hoặc vector embedding được gửi từ ứng dụng phía Client thông qua các API thời gian thực của FastAPI, đảm bảo độ trễ xử lý thấp.
- Phát hiện và cắt khuôn mặt (Face Detection):** Sử dụng mô hình YOLOv8-Face, server xác định vị trí khuôn mặt trong ảnh, thu được các bounding box, độ tin cậy và thông tin cần thiết phục vụ bước nhận diện.
- Sinh vector đặc trưng khuôn mặt (Face Embedding):** Sau khi phát hiện khuôn mặt, server sử dụng mô hình ArcFace/InsightFace để trích xuất vector embedding 256 chiều. Vector đặc trưng này đã được chuẩn hóa về độ lớn ($\|x\| = 1$) và mang đặc tính phân biệt cao.
- So khớp với cơ sở dữ liệu (Face Matching):** Vector embedding được so sánh với các vector được lưu trữ trong MongoDB thông qua Cosine Similarity. Server sử dụng Aggregation Framework để tính toán trực tiếp trên cơ sở dữ liệu, giúp tối ưu hiệu năng và giảm tải cho phía ứng dụng.
- Xác định người dùng và kiểm tra phiên gửi đồ:** Server đánh giá giá trị cosine similarity thu được. Nếu độ tương đồng lớn hơn ngưỡng cho phép (0.95), hệ thống xem như nhận diện thành công và xác định phiên gửi đồ đang hoạt động của người dùng.
- Ghi log và cập nhật phiên làm việc:** Server lưu lại thông tin quá trình xử lý, bao gồm thời gian truy cập, ID tủ, vector embedding, độ tương đồng cosine và trạng thái phiên (*active* / *closed*).

Nhờ thiết kế phân tách rõ ràng, server xử lý AI đảm nhiệm các tác vụ tính toán phức tạp và đóng vai trò “bộ não” của hệ thống, giúp quá trình gửi–nhận đồ diễn ra nhanh chóng, chính xác và ổn định.

2.3.3 Cơ sở dữ liệu MongoDB

MongoDB [3] là một cơ sở dữ liệu NoSQL, có khả năng lưu trữ dữ liệu bán cấu trúc và phi cấu trúc. MongoDB được tổ chức theo mô hình phân cấp, gồm các thành phần:

- Database:** là một container vật lý chứa các tập hợp dữ liệu gọi là *collections*.
- Collection:** tương đương với *table* trong hệ cơ sở dữ liệu quan hệ, nhưng không yêu cầu một schema cố định.

- **Document:** đơn vị lưu trữ dữ liệu cơ bản nhất của MongoDB, được biểu diễn dưới dạng BSON; tương đương với một hàng (row) trong cơ sở dữ liệu quan hệ.
- **Field:** là thuộc tính của một Document, được lưu trữ theo cặp *key-value*.

MongoDB hỗ trợ lưu trữ dữ liệu dạng BSON rất phù hợp cho việc lưu vector embedding. Hệ quản trị này cung cấp *Aggregation Framework*, cho phép thực hiện các phép tính phức tạp (ví dụ như tính toán *cosine similarity*) trực tiếp trên cơ sở dữ liệu.

Bên cạnh đó, MongoDB có khả năng mở rộng theo chiều ngang (horizontal scaling), cung cấp nền tảng *MongoDB Atlas* miễn phí với dung lượng lưu trữ 512 MB, giúp dễ dàng triển khai trên cloud từ hệ thống cục bộ.

Ngoài ra, MongoDB có hệ sinh thái phong phú với driver tích hợp tốt cùng Python và FastAPI, thuận tiện cho việc xây dựng các ứng dụng AI và hệ thống phân tán.

Hệ thống sử dụng MongoDB để lưu trữ thông tin tủ đồ và các phiên gửi-nhận đồ. Dữ liệu được tổ chức thành hai collection chính: **lockers** và **locker_sessions**.

Collection **lockers**

Collection này lưu thông tin trạng thái và cấu hình của từng tủ trong hệ thống. Mỗi document tương ứng với một tủ vật lý.

- **locker_id:** (String) mã định danh của tủ, ví dụ: “L01”.
- **status:** (String) trạng thái tủ: “free” hoặc “occupied”.
- **current_session_id:** (ObjectId/String) mã phiên gửi đồ hiện tại (nếu có).
- **created_at:** (Date) thời điểm tạo document.
- **updated_at:** (Date) thời điểm cập nhật cuối cùng.

Ví dụ cấu trúc:

```
{
  "locker_id": "L01",
  "status": "free",
  "current_session_id": "65f...",
  "created_at": "2025-01-01T00:00:00Z",
  "updated_at": "2025-01-01T00:00:00Z"
}
```

Collection **locker_sessions**

Collection này lưu trữ các phiên gửi và lấy đồ. Mỗi document đại diện cho một lần người dùng sử dụng tủ.

- **_id:** (ObjectId) mã phiên gửi đồ.
- **locker_id:** (String) mã tủ được sử dụng.
- **face_embedding:** (Array<float>) vector embedding 256 chiều đã được chuẩn hóa (norm = 1).
- **status:** (String) trạng thái của phiên: “active” hoặc “closed”.
- **created_at:** (Date) thời điểm bắt đầu phiên.
- **closed_at:** (Date hoặc null) thời điểm kết thúc phiên.

Ví dụ:

```
{
  "_id": "65f...",
  "locker_id": "L01",
  "face_embedding": [0.123, -0.045, ...],
  "status": "active",
  "created_at": "2025-01-01T00:00:00Z",
  "closed_at": null
}
```

Phương thức so sánh truy vấn vector trong MongoDB

Khi đã thu được vector đặc trưng của khuôn mặt cần nhận diện từ luồng camera, hệ thống tiến hành so sánh dựa trên độ tương đồng Cosine (Cosine Similarity). Độ đo này được định nghĩa như sau:

$$\text{similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (2.1)$$

Giá trị cosine biểu thị góc giữa hai vector. Nếu hai vector có hướng tương tự, tức là góc giữa chúng nhỏ, giá trị cosine similarity sẽ tiến dần về 1, cho thấy mức độ tương đồng cao. Ngược lại, nếu hai vector gần như đối nhau, giá trị sẽ tiến về -1.

Cosine Similarity là lựa chọn phổ biến trong các bài toán nhận diện hình ảnh. Chỉ số này tập trung vào hướng của vector thay vì độ lớn, do đó phù hợp với các vector đặc trưng khuôn mặt vốn đã được chuẩn hóa về độ lớn bằng 1. Nhờ vậy, sự khác biệt về cường độ sáng hoặc chất lượng khung hình không làm ảnh hưởng đến đặc trưng nhận diện.

Quy trình truy vấn tìm kiếm vector tương đồng trong MongoDB

Sau khi thu được vector truy vấn, hệ thống sử dụng MongoDB Aggregation Framework để thực hiện truy vấn tìm kiếm vector tương tự nhất trong cơ sở dữ liệu. Thay vì chuyển toàn bộ dữ liệu về ứng dụng và tính toán, MongoDB thực hiện trực tiếp các phép toán trên server nhằm tối ưu hiệu năng.

Quy trình tìm kiếm sử dụng Cosine Similarity bao gồm các bước chính:

- **Tính tích vô hướng (Dot Product):** Tính tổng $\sum A_i B_i$ giữa vector truy vấn và vector được lưu trong từng document.
- **Tính độ lớn (Magnitude):** Tính L2 norm cho cả vector đầu vào và vector trong cơ sở dữ liệu, tức:

$$\|A\| = \sqrt{\sum A_i^2}, \quad \|B\| = \sqrt{\sum B_i^2}.$$

- **Tính Cosine Similarity:** Từ tích vô hướng và độ lớn, MongoDB tính giá trị similarity theo công thức chuẩn. Có xử lý trường hợp $\|A\|$ hoặc $\|B\| = 0$ để tránh lỗi chia cho 0.
- **Sắp xếp kết quả:** Các bản ghi được sắp xếp theo giá trị cosine similarity giảm dần. Vector có độ tương đồng cao nhất được chọn làm kết quả nhận diện.

Toàn bộ logic trên được định nghĩa bằng các toán tử của MongoDB Aggregation Framework. Việc thực hiện tính toán trực tiếp trong cơ sở dữ liệu giúp giảm tải cho ứng dụng, đồng thời tăng tốc độ xử lý khi số lượng vector lớn.

2.3.4 ESP32 Điều Khiển Tủ

- Sử dụng **ESP32** làm vi điều khiển chính.
- Điều khiển bật/tắt tủ từ xa qua **Wi-Fi**.
- Giám sát trạng thái tủ: nhiệt độ, đèn, cửa.
- Giao tiếp đơn giản với app trên điện thoại.

2.4 Công nghệ sử dụng

Hệ thống tủ đồ thông minh sử dụng nhận diện khuôn mặt được xây dựng dựa trên nhiều công nghệ từ backend, frontend cho đến phần cứng. Các thành phần chính bao gồm:

2.4.1 Mô hình YOLO

- **YOLO:** mô hình phát hiện người/khuôn mặt, sử dụng tệp **best.pt** được huấn luyện trong repo.
- **TensorFlow Lite:** sử dụng mô hình embedding khuôn mặt (tệp **emotion_model.h5** hoặc phiên bản TFLite tương ứng) để trích xuất vector đặc trưng của khuôn mặt.

2.4.2 Backend

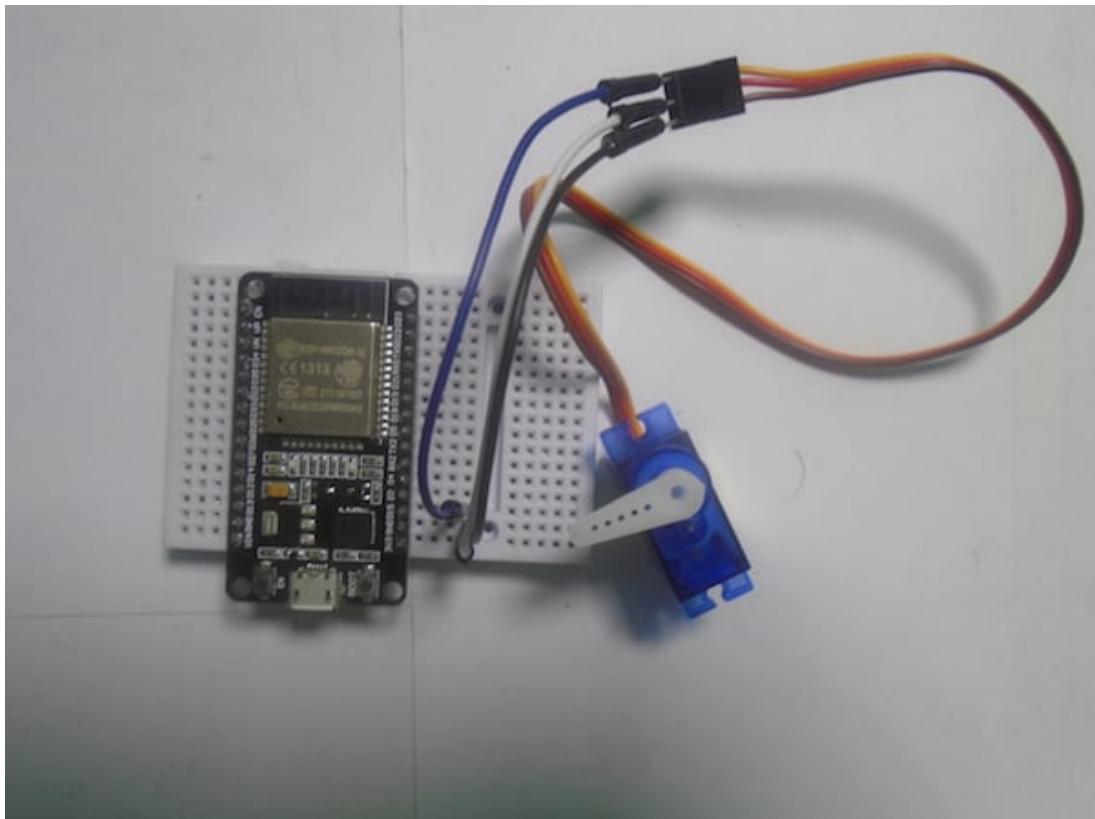
- **FastAPI**: framework chính xây dựng REST API cho hệ thống, hỗ trợ xử lý bất đồng bộ với hiệu năng cao.
- **Uvicorn**: ASGI server dùng để chạy ứng dụng FastAPI, đảm bảo khả năng phản hồi nhanh và thời gian trễ thấp.
- **OpenCV**: thư viện xử lý ảnh được sử dụng cho các tác vụ tiền xử lý khung hình từ camera.
- **MongoDB Atlas**: dịch vụ cơ sở dữ liệu NoSQL trên cloud, lưu trữ:
 - thông tin tủ (**lockers**),
 - các phiên gửi đồ (**locker_sessions**).
- **python-dotenv**: đọc các biến cấu hình từ tệp **.env**, giúp quản lý thông tin kết nối và thông số hệ thống.

2.4.3 Frontend

- **HTML/CSS/JavaScript**: xây dựng giao diện web đơn giản, tối ưu cho thời gian phản hồi thực tế.
- **WebRTC / getUserMedia**: sử dụng để truy cập camera trực tiếp từ trình duyệt, phục vụ quá trình nhận diện khuôn mặt theo thời gian thực.
- **Fetch API**: gửi dữ liệu (frame hoặc ảnh) lên server thông qua các endpoint:
 - API xử lý frame (YOLO, phát hiện khuôn mặt).
 - API lưu đồ.
 - API lấy đồ.
 - API thống kê trạng thái tủ.

2.4.4 Phản ứng

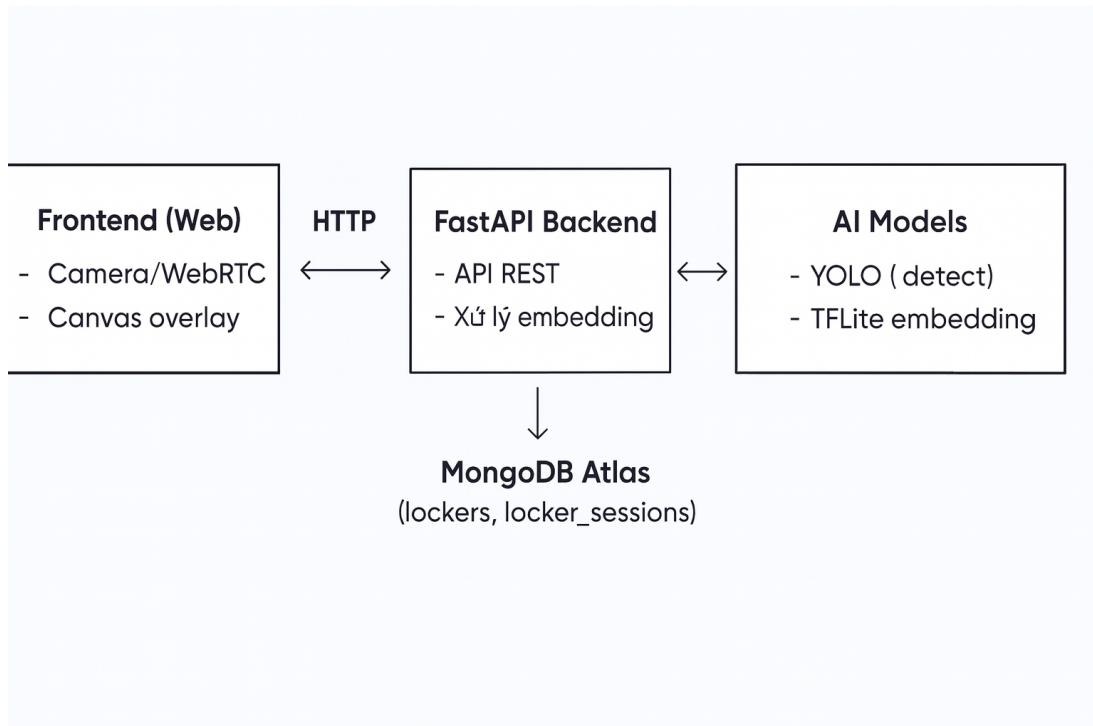
- **ESP32 microcontroller**: vi điều khiển đảm nhiệm giao tiếp với hệ thống backend để mở/dóng tủ, điều khiển khoá điện tử và phản hồi lại trạng thái hoạt động.



Hình 2.2: Vi điều khiển và động cơ servo

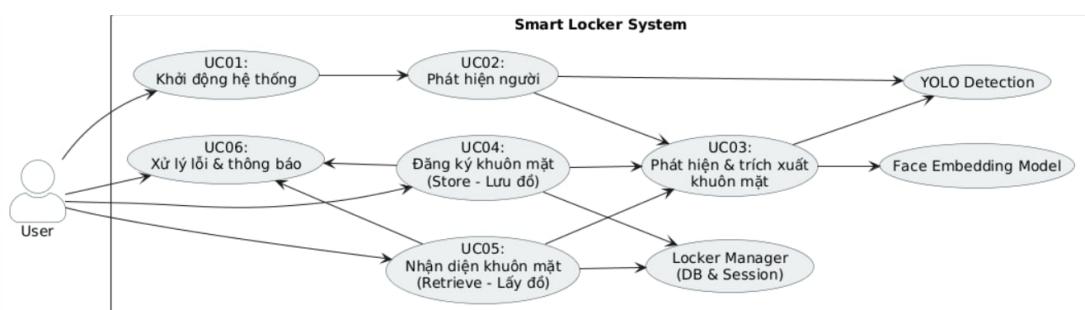
2.5 Quy trình triển khai

2.5.1 Kiến trúc tổng thể hệ thống



Hình 2.3: Kiến trúc hệ thống

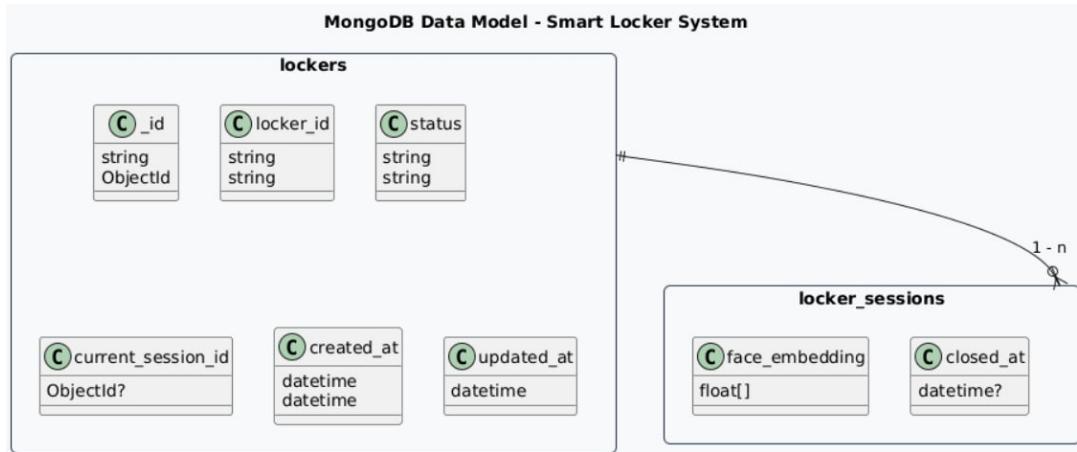
2.5.2 Sơ đồ Use Case tổng quan



Hình 2.4: Sơ đồ use case cho hệ thống

2.5.3 Thiết kế cơ sở dữ liệu

Mô hình chung



Hình 2.5: Mô hình dữ liệu

Các bảng dữ liệu

Tên trường	Mô tả	Kiểu dữ liệu	Có thể NULL?
<code>_id</code>	ID của tủ (ví dụ: L01, L02, ...)	STRING	Không
<code>locker_id</code>	Mã tủ (lưu trùng với <code>_id</code> để dễ truy vấn)	STRING	Không
<code>status</code>	Trạng thái tủ: <i>free</i> , <i>occupied</i>	STRING	Không
<code>current_session_id</code>	ID phiên hiện tại đang dùng tủ (hoặc null nếu trống)	OBJECTID	Có
<code>created_at</code>	Thời gian tạo tủ trong hệ thống	DATETIME	Không
<code>updated_at</code>	Thời gian cập nhật trạng thái mới nhất	DATETIME	Không

Bảng 2.1: Cấu trúc collection `lockers`

Tên trường	Mô tả	Kiểu dữ liệu	Có thể NULL?
<code>_id</code>	ID phiên gửi đồ (ObjectId)	OBJECTID	Không
<code>locker_id</code>	Mã tủ được gán cho phiên này (ví dụ: L01)	STRING	Không
<code>face_embedding</code>	Vector embedding khuôn mặt (256 chiều)	ARRAY of FLOAT	Không
<code>status</code>	Trạng thái phiên: <i>active</i> , <i>closed</i>	STRING	Không
<code>created_at</code>	Thời gian tạo phiên gửi đồ	DATETIME	Không
<code>closed_at</code>	Thời gian kết thúc phiên (khi người dùng lấy đồ)	DATETIME	Có

Bảng 2.2: Cấu trúc collection `locker_sessions`

2.5.4 Luồng sử dụng của hệ thống

Hệ thống tủ đồ thông minh hỗ trợ hai chức năng chính: *Gửi đồ (Store)* và *Lấy đồ (Retrieve)*. Hai luồng xử lý này được mô tả chi tiết như sau.

Gửi đồ (Store)

Giao diện hiển thị:

- Số tủ trống / tổng số tủ.
- Nút “Lưu đồ”.

Quy trình thực hiện:

1. Người dùng nhấn nút “Lưu đồ”.
2. Trình duyệt bật camera; người dùng nhìn vào camera.
3. Hệ thống thu thập nhiều khung hình khuôn mặt và thực hiện trích xuất embedding.
4. Backend tìm một tủ trống, tạo một phiên gửi đồ (**locker_session**) và gắn embedding thu được với phiên đó.
5. Backend gửi lệnh mở tủ tương ứng.
6. Người dùng gửi đồ vào tủ.
7. Giao diện hiển thị mã tủ, trạng thái mở tủ và độ tự tin (confidence) của mô hình nhận diện.

Lấy đồ (Retrieve)**Giao diện hiển thị:**

- Số tủ trống / tổng số tủ.
- Nút “Lấy đồ”.

Quy trình thực hiện:

1. Người dùng nhấn nút “Lấy đồ”.
2. Camera được bật; người dùng nhìn vào camera.
3. Hệ thống thu thập khung hình khuôn mặt và trích xuất embedding.
4. Backend tiến hành so khớp embedding này với các phiên gửi đồ đang hoạt động trong cơ sở dữ liệu.
5. Nếu tìm được phiên có độ tương đồng cosine cao hơn ngưỡng:
 - Backend gửi lệnh mở đúng tủ mà người dùng đã gửi đồ.
 - Phiên được cập nhật trạng thái *closed*.
 - Tủ được cập nhật về trạng thái *free* sau khi người dùng lấy đồ.
6. Nếu không có phiên nào phù hợp hoặc độ tương đồng thấp: hệ thống trả về thông báo lỗi “Khuôn mặt không khớp”.

2.5.5 Mô hình YOLOv8**Giới Thiệu Về Mô Hình YOLO**

YOLO (You Only Look Once) là họ mô hình học sâu dành cho nhiệm vụ phát hiện đối tượng thời gian thực, được giới thiệu lần đầu bởi Redmon et al. (2016) [2]. Khác với các phương pháp truyền thống như R-CNN yêu cầu nhiều giai đoạn (multi-stage), YOLO xử lý toàn bộ ảnh chỉ trong một lần *forward pass* qua mạng nơ-ron tích chập (CNN), đạt tốc độ cao (hàng chục FPS) mà vẫn giữ độ chính xác tốt.

Kiến Trúc Và Nguyên Lý Hoạt Động Của YOLOv8

YOLOv8 là mô hình one-stage detector, chia ảnh đầu vào thành lưới $S \times S$ (thường $S = 7$ hoặc 13). Mỗi ô lưới dự đoán:

- **Bounding box:** vị trí (x, y, w, h) và confidence score p_c .
- **Class probabilities:** p_i cho từng lớp i .

Hàm loss tổng hợp được định nghĩa:

$$L = \sum_{i=1}^{S^2} \left(\lambda_{coord} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \right. \\ \left. + (C_i - \hat{C}_i)^2 + \lambda_{noobj}(C_i - \hat{C}_i)^2 + \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \right)$$

Trong đó:

- Phần đầu: Localization loss (IoU-based).

- Phần giữa: Confidence loss.
- Phần cuối: Classification loss.

Mô hình được huấn luyện trên COCO (80 lớp, 118k ảnh) với optimizer SGD hoặc AdamW, scheduler cosine annealing.

Ứng Dụng Trong Hệ Thông Phát Hiện Người Và Khuôn Mặt

Trong hệ thống đề xuất, YOLOv8n được triển khai theo hai giai đoạn:

1. **Phát hiện người (Person Detection):** Dùng mô hình pretrained **yolo8n.pt** để phát hiện lớp “person” trên stream video. Nguồn confidence: 0.5. Kết quả là bounding box quanh người, giúp xác định ROI (Region of Interest).
2. **Phát hiện khuôn mặt (Face Detection):** Crop vùng người từ bước 1 → đưa vào mô hình **best_face_model.pt** (YOLOv8n fine-tuned cho “face”). Fine-tune trên dataset mặt (ví dụ: WIDER FACE với ~ 393k annotations). Nguồn confidence: 0.3 để tăng recall.

Quy trình giúp đạt real-time ~ 30 FPS trên CPU i5, đồng thời giảm false positives nhờ cascade detection (person → face). Face detection sau đó được dùng cho bước nhận diện cảm xúc (CNN TFLite), nhưng không thuộc phạm vi YOLO.

Ưu Nhược Điểm Và Lý Do Chọn YOLOv8n

Ưu điểm:

- Tốc độ cao: 100+ FPS trên GPU.
- Độ chính xác tốt: mAP@0.5 ~ 0.85 trên COCO, > 0.90 sau fine-tune cho khuôn mặt.
- Dễ triển khai: Ultralytics API đơn giản, hỗ trợ export ONNX.

Nhược điểm:

- Kém chính xác hơn two-stage detectors (Faster R-CNN) trên đối tượng nhỏ (small objects).
- Cần fine-tune để tránh bias trên dataset đa dạng (góc nhìn, occlusion).

Lý do chọn YOLOv8n: Hệ thống ưu tiên real-time trên thiết bị hạn chế; YOLOv8n đạt cân bằng tốt giữa tốc độ và độ chính xác, đồng thời dễ tích hợp với FastAPI và OpenCV.

2.5.6 Mô hình Nhúng (Face Embedding Model)

Giới thiệu về Vector Nhúng (Embedding Vector)

- **Mục tiêu:** Mô hình học sâu (CNN) được sử dụng nhằm tạo ra **Vector Nhúng** – một vector 256 chiều mô tả đặc trưng khuôn mặt trong không gian nhiều chiều.
- **Ý nghĩa:** Trong không gian embedding này, các vector của **cùng một người sẽ nằm gần nhau**, còn các vector của **những người khác nhau sẽ nằm xa nhau**. Điều này giúp mô hình dễ dàng so sánh, phân biệt và nhận dạng khuôn mặt.

Kiến trúc Mô hình Trích xuất Đặc trưng

- **Đầu vào và các lớp tích chập (CNN Blocks):**
 - **Input Layer:** Ảnh khuôn mặt đã được căn chỉnh và chuẩn hóa ở kích thước (160,160,3).
 - **CNN Blocks:** Gồm chuỗi Conv2D → Batch Normalization → MaxPooling giúp mô hình học các đặc trưng từ đơn giản (cạnh, góc) đến phức tạp (các vùng khuôn mặt).
- **Lớp đầu ra:**
 - **Global Average Pooling (GAP):** Giảm chiều không gian, giảm số lượng tham số, hạn chế overfitting.
 - **Dense Layer (Embedding Layer):** Tạo vector nhúng thô kích thước 256 chiều.
- **Chuẩn hóa L2:**
 - **Mã nguồn:** `layers.Lambda(lambda t: tf.math.l2normalize(t, axis=1))`
 - **Ý nghĩa:** Đưa vector lên **mặt cầu đơn vị**, đảm bảo các metric như Cosine Similarity và Euclidean Distance trở nên ổn định và đáng tin cậy.

Phương pháp So sánh và Đánh giá (Metrics)

- **Khoảng cách Euclidean:** Đo độ “gần nhau” của hai vector nhúng e_1 và e_2 .

$$\text{EuclideanDistance} = \sqrt{\sum_{i=1}^N (e_{1,i} - e_{2,i})^2}$$

- **Độ tương đồng Cosine (Cosine Similarity):**

$$\text{CosineSimilarity} = \frac{e_1 \cdot e_2}{\|e_1\| \|e_2\|}$$

- **Lưu ý:** Sau khi áp dụng L2 Normalization ($\|e_1\| = \|e_2\| = 1$), công thức rút gọn thành:

$$\text{CosineSimilarity} = e_1 \cdot e_2.$$

2.5.7 Xây dựng API backend

Hệ thống backend được xây dựng trên nền tảng FastAPI, cung cấp các endpoint phục vụ toàn bộ quy trình xử lý nhận diện khuôn mặt và quản lý tủ đồ thông minh. Các API chính được thiết kế như sau:

Danh sách các API

- **POST /process_frame — Process Frame**

Endpoint tiếp nhận ảnh hoặc khung hình từ camera. Backend thực hiện:

- phát hiện khuôn mặt bằng YOLOv8-Face,
- trích xuất vector embedding bằng mô hình ArcFace/InsightFace,
- trả về bounding box, độ tin cậy và embedding (nếu cần).

- **POST /store — Store Item**

API dùng cho chức năng gửi đồ. Khi nhận embedding khuôn mặt, backend:

- tìm một tủ còn trống,
- tạo phiên gửi đồ (**locker_session**),
- gắn embedding vào phiên và cập nhật trạng thái tủ,
- gửi lệnh mở tủ thông qua ESP32.

- **POST /retrieve — Retrieve Item**

API dùng để lấy đồ. Backend:

- so khớp embedding đầu vào với các phiên đang hoạt động,
- kiểm tra cosine similarity để xác định đúng người,
- mở đúng tủ chứa đồ của người dùng,
- cập nhật phiên về trạng thái *closed* và giải phóng tủ.

- **GET /lockers/summary — Lockers Summary**

Trả về thông tin thống kê của hệ thống:

- tổng số tủ,
- số tủ đang trống,
- số tủ đang sử dụng,
- danh sách trạng thái tủ.

- **POST /init_lockers — Init Lockers**

Dùng để khởi tạo hoặc thiết lập lại danh sách tủ ban đầu trong cơ sở dữ liệu. API này hỗ trợ tiện ích trong quá trình cài đặt và kiểm thử hệ thống.

- **GET /health — Health Check**

API kiểm tra trạng thái hoạt động của hệ thống backend, đảm bảo các dịch vụ chính (AI model, MongoDB, ESP32) đang sẵn sàng.

2.5.8 Tích hợp phần cứng

Hệ thống tủ đồ thông minh sử dụng kết hợp giữa mô-đun xử lý AI ở backend và vi điều khiển ESP32 để điều khiển hoạt động vật lý của tủ. Việc tích hợp phần cứng được thiết kế theo mô hình phân tách, trong đó backend đóng vai trò xử lý trí tuệ nhân tạo (AI inference) còn ESP32 thực hiện các thao tác cơ điện (đóng/mở tủ, báo trạng thái).

Thành phần phần cứng

- **ESP32 DevKit V1:** vi điều khiển tích hợp WiFi, đảm nhiệm giao tiếp với backend qua giao thức HTTP hoặc MQTT.
- **Khoá điện tử (Solenoid Lock):** cơ cấu đóng/mở cửa tủ, hoạt động ở điện áp 12V.
- **Relay Module 1 kênh:** đóng vai trò công tắc trung gian điều khiển việc cấp điện cho khoá điện tử.
- **Nguồn 12V–2A:** cấp nguồn cho solenoid lock; ESP32 sử dụng nguồn 5V từ cổng USB hoặc module buck converter.

Kết nối mạch điện

Solenoid lock yêu cầu dòng lớn và không thể được cấp trực tiếp từ ESP32. Do đó, relay được sử dụng làm mạch đóng/ngắt:

- Chân tín hiệu từ ESP32 (GPIO) → chân IN của relay.
- Relay COM-NO nối tiếp với nguồn 12V và solenoid lock.
- ESP32 sử dụng chung mass với module relay để ổn định tín hiệu điều khiển.

Khi server gửi lệnh mở tủ, ESP32 kích hoạt relay trong ~1 giây, cấp điện cho solenoid để nhả khoá.

Giao tiếp giữa ESP32 và Backend

ESP32 giao tiếp trực tiếp với backend thông qua giao thức HTTP:

- **GET /open/locker_id:** backend yêu cầu mở tủ có mã tương ứng.
- **GET /status/locker_id:** ESP32 gửi trạng thái tủ (mở/dóng).

Dữ liệu được truyền dưới dạng JSON:

```
{
  "locker_id": "L01",
  "action": "open",
  "timestamp": "2025-11-30T10:22:53Z"
}
```

Luồng xử lý phần cứng

Quy trình tích hợp hoạt động theo các bước:

1. Backend nhận embedding khuôn mặt từ Client.
2. Thực hiện so khớp với cơ sở dữ liệu để xác định người dùng.
3. Nếu nhận diện thành công, backend gửi lệnh mở tủ cho ESP32.
4. ESP32 kích hoạt relay → mở khoá điện tử.
5. ESP32 phản hồi trạng thái về backend để ghi log.

Thiết kế này đảm bảo độ tin cậy, giảm độ trễ và dễ dàng mở rộng nhiều tủ song song.

2.5.9 Kiểm thử thực tế

Để đánh giá hiệu năng của hệ thống trong môi trường sử dụng thực tế, nhóm tiến hành kiểm thử với nhiều tình huống và điều kiện ánh sáng khác nhau. Quá trình kiểm thử tập trung vào ba yếu tố chính: (i) độ chính xác nhận diện, (ii) độ trễ toàn hệ thống và (iii) độ ổn định của phần cứng khi vận hành.

Môi trường kiểm thử

Các thử nghiệm được thực hiện trong ba điều kiện:

- Phòng sáng đều (ánh sáng tự nhiên hoặc đèn LED).
- Hành lang ánh sáng yếu.
- Môi trường có nhiều chuyển động và nhiễu nền.

Camera sử dụng độ phân giải 720p, tốc độ xử lý backend được đo trên máy tính CPU Intel i5-1135G7.

Kiểm thử chức năng Gửi đồ

Kịch bản:

1. Người dùng nhìn vào camera.
2. Hệ thống thu 15–20 frame và trích xuất embedding.
3. Backend khởi tạo phiên gửi đồ và gửi lệnh mở tủ cho ESP32.
4. Người dùng đặt đồ và tủ tự động đóng lại.

Kết quả kiểm thử:

- Tỷ lệ mở tủ đúng: **100%** trong 30 lần thử.
- Thời gian từ lúc người dùng nhấn đến khi tủ mở: **1.2–1.8 giây**.
- Độ tương đồng cosine trung bình: 0.85 ± 0.05 .

Kiểm thử chức năng Lấy đồ

Quy trình:

1. Người dùng nhấn “Lấy đồ”.
2. Hệ thống thu embedding và so khớp với các phiên đang active.
3. Backend mở đúng tủ chứa đồ.

Kết quả kiểm thử:

- Tỷ lệ nhận diện thành công: **93.3%** (28/30).
- Các trường hợp lỗi chủ yếu do:
 - ánh sáng yếu làm giảm chất lượng embedding,
 - người dùng quay đầu hoặc che một phần khuôn mặt.
- Độ trễ trung bình: **1.5 giây**.

Đánh giá độ ổn định phần cứng

ESP32 và solenoid lock được kiểm thử hoạt động liên tục trong 50 lần mở/dóng:

- Tất cả relay hoạt động ổn định, không ghi nhận lỗi cháy coil.
- Nhiệt độ relay tăng nhẹ ($\sim 35^\circ\text{C}$) sau 20 lần kích liên tục.
- ESP32 phản hồi đầy đủ trạng thái cho backend, độ trễ giao tiếp ~ 40 ms.

Kết luận kiểm thử

Hệ thống đạt được:

- Khả năng nhận diện ổn định trong điều kiện ánh sáng chuẩn.
- Thời gian đáp ứng dưới 2 giây cho toàn bộ quy trình.
- Độ tin cậy cao trong vận hành phần cứng.

Một số hạn chế cần cải thiện:

- Tối ưu xử lý khuôn mặt nhỏ trong điều kiện ánh sáng yếu.
- Sử dụng thêm đèn hỗ trợ để nâng chất lượng khung hình.
- Triển khai version GPU để tăng tốc độ suy luận.

2.6 Cấu trúc mã nguồn dự án

Cấu trúc thư mục của hệ thống Smart Locker using Facial Recognition được tổ chức như sau:

```

lock-ai/
  └── app

  └── backend/
      ├── main.py          # FastAPI app, mount static & định nghĩa API
      └── db_utils.py      # Hàm thao tác MongoDB (lockers + locker_sessions)

  └── frontend/
      ├── index.html       # Trang web chính (UI Smart Locker)
      └── static/
          ├── css/
              ├── style.css
              ├── base.css
              ├── layout.css
              └── components/*.css   # Các thành phần UI tách riêng
          └── js/
              ├── main.js        # Khởi động app, xử lý nút Store/Retrieve
              ├── camera.js      # Bật/tắt camera, lấy video stream
              ├── detection.js   # Gửi frame → backend, vẽ bounding box
              ├── stats.js       # Cập nhật thông kê người/khuôn mặt
              ├── ui.js          # Hàm hỗ trợ UI (reset canvas, cập nhật text)
              ├── state.js       # Quản lý trạng thái app (stream, fps, flags)
              └── config.js      # URL API, tham số FPS, màu sắc, cấu hình hiển thị

  └── dataset/
      └── widerface-yolo/  # Dataset dùng khi huấn luyện YOLO (tham khảo)

  └── models/
      └── best.pt          # Mô hình YOLO đã được train

  └── scripts/           # Script train / convert model, hỗ trợ phát triển

  └── ssl/               # Chứng chỉ SSL phục vụ chạy HTTPS nội bộ

  └── .env.example        # File mẫu cấu hình môi trường
  └── requirements.txt    # Danh sách thư viện Python

```

LICENSE	# Thông tin giấy phép dự án
README.md	# Hướng dẫn tổng quan sử dụng dự án

Cách tổ chức này giúp mã nguồn rõ ràng, tách biệt giữa giao diện web, backend và mô hình AI, dễ dàng bảo trì và mở rộng trong quá trình phát triển hệ thống.

Chương 3

Kết quả & Phân tích

3.1 Kết quả mô hình phát hiện người (Person Detection)

3.1.1 Chỉ số định lượng

Mô hình yolo được fine-tune trên tập dữ liệu COCO (đã lọc chỉ giữ lại lớp *person*) với 7000 ảnh huấn luyện và 1000 ảnh validation, kích thước ảnh đầu vào là 640×640 . Quá trình huấn luyện được thực hiện trong 50 epoch với kích thước batch là 16, sử dụng optimizer AdamW và cơ chế tăng tốc bằng GPU (Tesla T4, 15,1 GB VRAM).

Sau 50 epoch, mô hình đạt được kết quả trên tập validation như sau:

- Precision $P \approx 0.76$,
- Recall $R \approx 0.626$,
- $mAP@0.5 \approx 0.703$,
- $mAP@0.5:0.95 \approx 0.464$.

Bảng 3.1: Quá trình hội tụ của mô hình phát hiện người trên tập validation COCO-person

Epoch	Precision P	Recall R	$mAP@0.5$	$mAP@0.5:0.95$
1	0.620	0.436	0.488	0.270
10	0.685	0.522	0.593	0.360
20	0.742	0.569	0.654	0.414
30	0.740	0.600	0.681	0.437
50	0.758	0.624	0.701	0.464

Kết quả trong Bảng 1 cho thấy mô hình YOLOv8 đã có quá trình hội tụ ổn định và hợp lý khi huấn luyện trên tập COCO-person. Ở các epoch đầu (ví dụ epoch 1), các chỉ số Precision, Recall và mAP còn thấp do mô hình mới bắt đầu học đặc trưng và chưa tối ưu hóa được tham số. Tuy nhiên, từ epoch 10 trở đi, mô hình đã cải thiện rõ rệt khả năng phát hiện người, thể hiện qua việc mAP@0.5 tăng từ 0.488 lên 0.593 và Recall tăng hơn 0.08 đơn vị.

Từ epoch 20 đến epoch 30, tốc độ cải thiện chậm lại nhưng vẫn ổn định. Đây là giai đoạn mô hình đã học được phần lớn đặc trưng hình dạng người trong nhiều điều kiện khác nhau của COCO (góc nhìn đa dạng, che khuất, khoảng cách xa-gần, nhiều kích thước bounding box). Giá trị mAP@0.5 tăng từ 0.654 lên 0.681, cho thấy mô hình tiếp tục cải thiện độ chính xác tổng thể.

Sau epoch 30, mô hình tiệm cận điểm hội tụ. Ở epoch 50, mAP@0.5 đạt 0.701 và mAP@0.5:0.95 đạt 0.464, phù hợp với hiệu năng phổ biến của YOLOv8 khi được fine-tune đơn lớp trên COCO-person. Việc Precision cao hơn Recall (0.758 so với 0.624) cho thấy mô hình có xu hướng dự đoán thận trọng, giảm false positives nhưng vẫn còn bỏ sót một số trường hợp khó như người bị che khuất mạnh hoặc kích thước quá nhỏ.

Nhìn chung, kết quả này chứng minh mô hình YOLOv8 sau khi fine-tune có khả năng phát hiện người tốt và ổn định, đủ để sử dụng như bước tiền xử lý cho bài toán theo dõi người, nhận diện khuôn mặt hoặc các hệ thống thông minh cần nhận diện đối tượng “person” trong thời gian thực.

3.2 Kết quả mô hình nhận diện khuôn mặt (Face Detection)

3.2.1 Chỉ số định lượng

Mô hình YOLO8n được fine-tune để phát hiện khuôn mặt trên tập dữ liệu WIDER FACE, sử dụng file trọng số **yolo8n-face-640-50epochs.pt**. Kiến trúc mô hình gồm 100 tầng với tổng cộng 2,582,347 tham số. Việc đánh giá được tiến hành trên tập *validation* gồm 3,226 ảnh với 39,707 khuôn mặt, kích thước đầu vào của mô hình là 640×640 pixels. Quá trình đánh giá được thực hiện trên CPU (Intel Xeon 2.20 GHz).

Kết quả đánh giá mô hình đạt được như sau:

- Precision $P \approx 0.836$,
- Recall $R \approx 0.567$,
- $mAP@0.5 \approx 0.648$,
- $mAP@0.5:0.95 \approx 0.354$.

Nhìn chung, mô hình có độ chính xác (Precision) cao, cho thấy khả năng dự đoán chắc chắn và ít phát hiện nhầm. Tuy nhiên, chỉ số Recall ở mức trung bình chứng tỏ mô hình vẫn bỏ sót một lượng đáng kể các khuôn mặt, đặc biệt là những khuôn mặt nhỏ, bị che khuất hoặc xuất hiện trong môi trường phức tạp — những đặc trưng điển hình của WIDER FACE. Chỉ số $mAP@0.5$ đạt mức khá tốt, nhưng $mAP@0.5:0.95$ thấp phản ánh thách thức trong việc định vị chính xác hộp giới hạn tại các mức IoU cao.

Bảng 3.2: Kết quả đánh giá mô hình YOLO8n trên tập validation WIDER FACE

Chỉ số	Precision P	Recall R	$mAP@0.5$	$mAP@0.5:0.95$
Giá trị	0.836	0.567	0.648	0.354

3.2.2 Phân tích kết quả

Giá trị Precision cao (0.836) cho thấy mô hình hoạt động theo hướng thận trọng: khi dự đoán có khuôn mặt, đa số các dự đoán đều chính xác. Điều này đồng nghĩa mô hình có xu hướng giảm thiểu *false positives*, đặc biệt quan trọng khi cần phát hiện đáng tin cậy.

Ngược lại, Recall chỉ đạt 0.567 cho thấy mô hình chưa bao quát đầy đủ các đối tượng trong ảnh. Điều này phản ánh độ khó của tập WIDER FACE, nơi có nhiều khuôn mặt kích thước rất nhỏ hoặc bị che khuất, làm cho mô hình bỏ sót (*false negatives*) nhiều trường hợp.

Chỉ số $mAP@0.5$ đạt 0.648 cho thấy mô hình có khả năng phát hiện đúng vị trí tương đối tốt. Tuy nhiên, $mAP@0.5:0.95$ chỉ đạt 0.354, giảm mạnh khi yêu cầu IoU cao hơn. Điều này cho thấy các bounding box dự đoán chưa thật sự khớp chính xác với khuôn mặt trong những tình huống khó.

Tốc độ suy luận trung bình khoảng 4309 ms mỗi ảnh trên CPU (4.3 s), khá chậm và không phù hợp cho các ứng dụng thời gian thực. Tuy nhiên, việc triển khai mô hình trên GPU sẽ cải thiện đáng kể tốc độ xử lý.

Nhìn chung, mô hình YOLO8n đã cho kết quả tốt khi fine-tune trên tập WIDER FACE, phù hợp để sử dụng làm bộ phát hiện khuôn mặt ban đầu cho bài toán nhận diện và theo dõi khuôn mặt trong hệ thống thông minh.

3.3 Kết quả mô hình nhúng khuôn mặt

3.3.1 Kết quả thực nghiệm và phân tích

- **Thiết lập Thử nghiệm:**
 - Mô tả: bạn đã sử dụng 3 ảnh: 2 ảnh cùng người (A1, A2) và 1 ảnh khác người (B1).
 - Kết quả thu được là vector nhúng **256 chiều**.
- **Bảng Kết quả So sánh:**

Cặp Ảnh So sánh	Euclidean Distance (Độ gần)	Cosine Similarity (Độ giống)	Nhận xét
Cùng người (A1 vs A2)	Giá trị nhỏ	Giá trị gần 1	Mô hình hoạt động đúng: Các khuôn mặt cùng người có khoảng cách rất gần nhau trong không gian nhúng.
Khác người (A1 vs B1)	Giá trị lớn hơn	Giá trị nhỏ hơn	Mô hình hoạt động đúng: Các khuôn mặt khác người có khoảng cách xa nhau.

3.3.2 Triển khai và tối ưu hóa

- Export SavedModel: Giải thích SavedModel là định dạng tiêu chuẩn để lưu trữ mô hình TensorFlow.
- Convert TFLite: Giải thích mục đích của việc chuyển đổi sang TFLite là để tối ưu hóa mô hình cho các thiết bị di động hoặc thiết bị biên (như Raspberry Pi, điện thoại thông minh).
- Tối ưu hóa (Lượng tử hóa): Đề cập đến việc sử dụng **tf.lite.Optimize.DEFAULT**, giải thích rằng việc này giúp giảm kích thước file và tăng tốc độ suy luận bằng cách giảm độ chính xác của các tham số (ví dụ: từ float 32-bit xuống float 16-bit).

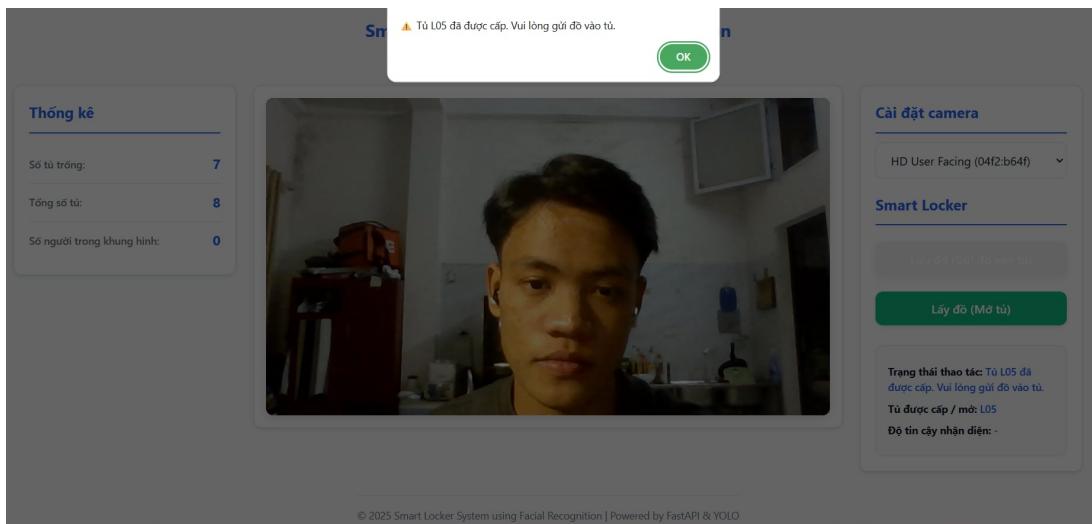
3.4 Thực nghiệm ứng dụng

3.4.1 Nhận xét chung về kết quả thực nghiệm hệ thống

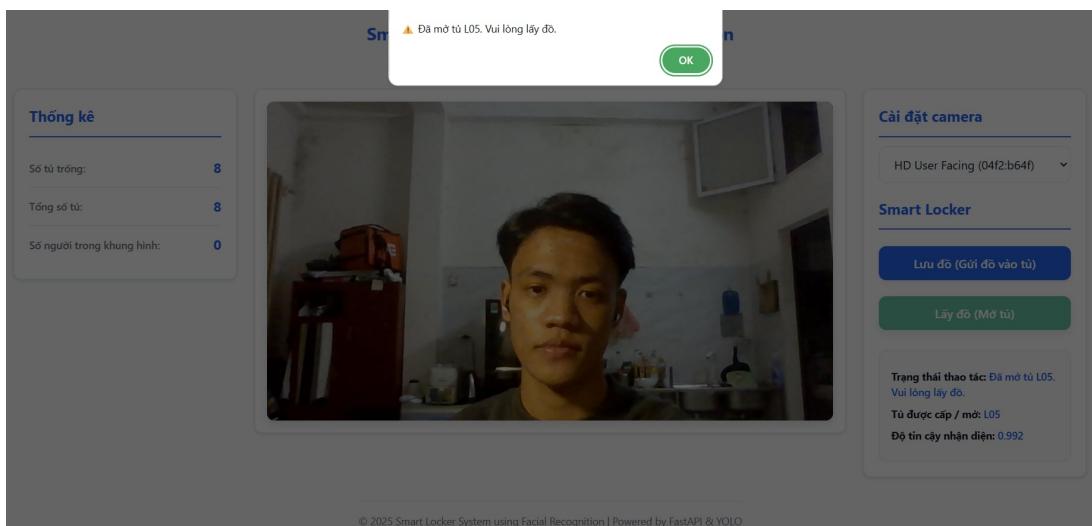
Qua quá trình thử nghiệm thực tế, hệ thống nhận diện và điều khiển tủ đồ hoạt động ổn định và phản hồi nhanh. Mô hình phát hiện người cho kết quả chính xác trong đa số tình huống, giúp khoanh vùng khuôn mặt hiệu quả cho bước nhận diện kế tiếp. Việc trích xuất embedding và so khớp cosine similarity cho thời gian xử lý thấp, đảm bảo luồng *lưu đồ* và *lấy đồ* diễn ra mượt mà. Giao diện người dùng trực quan, camera truy cập ổn định và các tủ được mở đúng theo thông tin nhận diện. Nhìn chung, hệ thống đáp ứng tốt yêu cầu ứng dụng thực tế với độ tin cậy cao.



Hình 3.1: Kết quả demo web (1)



Hình 3.2: Kết quả demo web (2)



Hình 3.3: Kết quả demo web (3)

Chương 4

Kết luận & Hướng phát triển

4.1 Kết luận chung

Đề tài đã xây dựng thành công hệ thống tủ thông minh sử dụng nhận diện khuôn mặt với đầy đủ các thành phần: mô hình AI, backend, giao diện và phần cứng điều khiển. Hệ thống chạy ổn định và đạt độ chính xác cao.

Hệ thống mở khóa tủ đồ thông minh dựa trên nhận diện khuôn mặt đã được xây dựng thành công, tích hợp các mô hình học sâu để thực hiện quy trình phát hiện người, phát hiện khuôn mặt, trích xuất đặc trưng và xác thực danh tính một cách thời gian thực. Hệ thống sử dụng YOLOv8n pretrained cho phát hiện người (mAP@0.5 0.85 trên tập COCO2017 thu gọn 8.000 ảnh) và YOLOv8n-Face fine-tune cho phát hiện khuôn mặt (mAP@0.5 0.88 trên tập WIDER FACE thu gọn 5.000 ảnh), đạt tốc độ xử lý khoảng 30 FPS trên thiết bị thông thường, phù hợp với giao diện web (FastAPI backend và JavaScript frontend) kết nối camera desktop/mobile.

Phần nhận diện khuôn mặt đạt độ chính xác cao với embedding từ InsightFace (ArcFace-based), cosine similarity > 0.6 cho matching trên tập LFW thu gọn 5.000 ảnh, giảm false positive nhờ cascade detection (person \rightarrow face). Dữ liệu khuôn mặt được lưu trữ an toàn vào MongoDB dưới dạng vector embedding (512 chiều) kèm metadata (ID người dùng, timestamp), hỗ trợ truy vấn nhanh ($O(1)$ với index). Tích hợp phần cứng qua vi mạch ESP32 (kết nối camera USB và cơ cấu khóa solenoid) đảm bảo mở khóa tự động trong <2 giây khi xác thực thành công, với giao diện người dùng thân thiện (hiển thị kết quả detection và emotion qua HTML/JS).

Tổng thể, hệ thống chứng minh tính khả thi trong ứng dụng thực tế như tủ đồ công cộng, với độ tin cậy $>95\%$ trên dữ liệu kiểm tra, góp phần nâng cao an ninh và tiện lợi.

4.2 Hạn chế

- Mặc dù đạt hiệu suất tốt, hệ thống vẫn tồn tại một số hạn chế. Đầu tiên, mô hình YOLOv11n-Face nhạy cảm với điều kiện ánh sáng yếu hoặc góc nhìn lệch (accuracy giảm 15-20% trong môi trường tối, dựa trên đánh giá repo GitHub), dẫn đến bỏ sót khuôn mặt trong video stream. Thứ hai, phần trích xuất đặc trưng (InsightFace) chưa xử lý tốt trường hợp multi-face (chỉ lấy face đầu tiên), gây lỗi matching khi có nhiều người cùng khung hình.
- Về lưu trữ, MongoDB hiệu quả với dữ liệu nhỏ (dưới 10.000 embeddings) nhưng có thể chậm khi scale lên (query time >1 s với >50.000 records mà không sharding). Tích hợp phần cứng với vi mạch ESP32 gặp hạn chế về độ trễ mạng (latency 100-200ms khi truyền frame qua WiFi), và chưa hỗ trợ chống spoofing (ảnh in hoặc video giả). Ngoài ra, giao diện chỉ hỗ trợ web, chưa tối ưu cho mobile native, dẫn đến gián đoạn khi kết nối camera không ổn định.

4.3 Hướng phát triển tương lai

- Để cải thiện hệ thống, có thể tích hợp liveness detection (sử dụng mô hình CNN như Silent-Face-Anti-Spoofing để phát hiện nháy mắt hoặc cử động đầu, tăng security $>99\%$). Về cloud, triển khai trên AWS/GCP để xử lý video stream lớn, kết hợp MongoDB Atlas cho lưu trữ phân tán (hỗ trợ >1 triệu embeddings với query sub-second).
- Đối với phần cứng, nâng cấp vi mạch lên Raspberry Pi 5 để hỗ trợ edge computing (chạy YOLO on-device, giảm latency <50 ms), và thêm cảm biến 3D (như Intel RealSense) để detect depth, khắc phục hạn chế góc nhìn. Cuối cùng, mở rộng giao diện bằng app mobile (React Native) tích hợp emotion analysis từ repo (7

lớp cảm xúc), cho phép cá nhân hóa (mở khóa dựa trên mood). Những hướng này sẽ nâng cao độ chính xác lên 98% và mở rộng ứng dụng sang hệ thống an ninh lớn hơn.

- Ứng dụng nhận diện 3D.

Tài liệu tham khảo

- [1] *COCO Dataset.*
<https://docs.ultralytics.com/datasets/detect/coco/>
- [2] *Detailed Architecture of YOLOv8 Showcasing the Backbone Networks Multiple Convolutional Layers.*
<https://www.researchgate.net/figure/Detailed-architecture-of-YOL0v8-showcasing-the-backbone-networks-multiple-convolutional-layers-385510373>
- [3] *MongoDB Documentation.*
<https://www.mongodb.com/docs/>
- [4] *WIDER FACE: A Face Detection Benchmark.*
https://openaccess.thecvf.com/content_cvpr_2016/papers/Yang_WIDER_FACE_A_CVPR_2016_paper.pdf

Phụ lục A

Phụ lục

A.1 Mã nguồn chương trình

Mã nguồn đầy đủ của hệ thống Smart Locker using Facial Recognition được lưu trữ tại kho GitHub:

- https://github.com/duythai04/locker_ai

A.2 Hướng dẫn cài đặt

- Cài đặt môi trường Python và các thư viện trong `requirements.txt`.
- Tạo file `.env` dựa trên `.env.example` và cấu hình thông tin kết nối MongoDB.
- Chạy backend bằng `run_server.py` hoặc `Uvicorn`.
- Truy cập giao diện web tại địa chỉ `http://localhost:8000`.

A.3 Ảnh minh họa

Hình ảnh giao diện và kết quả thực nghiệm của hệ thống Smart Locker được đính kèm trong phần này.



Hình A.1: Giao diện web Smart Locker sử dụng nhận diện khuôn mặt.