

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



Vehicle Counting AI

Nguyễn Minh Anh
Nguyễn Trung Kiên
Nguyễn Thế Quang
Trần Đăng Tài
Nguyễn Doãn Toàn

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 (Đại học Quốc gia Hà Nội – Trường Đại học Khoa học Tự nhiên)

Tên dự án: Vehicle Counting AI

Ngày nộp: 30/11/2025

Báo cáo PDF: Liên kết tới báo cáo PDF trên GitHub

Slide thuyết trình: Liên kết tới slide thuyết trình

Kho GitHub: https://github.com/dekii2275/CV_trafficdetection

Thành viên nhóm

Họ tên	Mã sinh viên	Tên GitHub	Đóng góp
Nguyễn Minh Anh	23001495	Dekii2275	Counting Vehicle
Nguyễn Trung Kiên	23001530	nguyenkien0912	Analysis
Nguyễn Thế Quang	23001549	thequang05	Model
Trần Đăng Tài	23001558	TaiTranDang145	Backend
Nguyễn Doãn Toàn	23001564	DTOan010605	Frontend

Mục lục

1 Giới thiệu	7
1.1 Tóm tắt	7
1.2 Bài toán đặt ra	7
2 Phương pháp & Triển khai	8
2.1 Phương pháp	8
2.1.1 Lý thuyết về mạng nơ-ron tích chập	8
2.1.2 Kiến trúc YOLOv8m	8
2.1.3 Phương pháp Fine-tuning	9
2.1.4 Các chỉ số đánh giá hiệu quả	9
2.1.5 Dữ liệu và tiền xử lý	10
2.2 Triển khai	12
2.2.1 Môi trường và công cụ phát triển	12
2.2.2 Cấu trúc và tổ chức mã nguồn	12
2.2.3 Xây dựng mô hình	14
2.2.4 Đếm phương tiện và xử lý dữ liệu trả về	15
2.2.5 Phân tích dữ liệu	16
2.2.6 Xây dựng ứng dụng web	17
2.2.7 Quy trình triển khai tổng thể	19
3 Kết quả & Phân tích	21
3.1 Kết quả huấn luyện mô hình	21
3.2 Phân tích hiệu năng theo từng Lớp	23
3.3 Hiệu năng xử lý realtime	23
3.4 Kết quả hệ thống đếm phương tiện	23
3.5 Kết quả phân tích dữ liệu	24
3.5.1 Phân tích xu hướng và dự đoán lưu lượng tổng thể	24
3.5.2 Phát hiện và đánh giá đỉnh Lưu lượng	24
3.5.3 Phân tích cơ cấu thành phần và tỷ trọng	25
3.5.4 So sánh chi tiết lưu lượng theo loại xe	25
3.6 Kết quả Web Application	26
3.7 Thảo luận	28
4 Kết luận	29
4.1 Kết luận	29
4.2 Hạn chế	29
4.3 Hướng phát triển	29
Tài liệu tham khảo	31
5 Phụ lục	32
5.1 Yêu cầu Hệ thống	32
5.2 Cài đặt Backend	32
5.3 Cài đặt Frontend	32
5.4 Chạy Hệ thống	33
5.4.1 Chạy Backend	33
5.4.2 Chạy Frontend	33
5.5 Lưu ý Quan trọng	33
5.6 Troubleshooting	33

5.6.1	Lỗi Import Module	33
5.6.2	Lỗi Kết nối API	34
5.6.3	Lỗi Camera	34
5.7	Cấu hình Hệ thống	34

Danh sách hình vẽ

2.1	Mẫu dữ liệu đại diện cho 4 lớp phương tiện trong bộ dữ liệu.	11
2.2	Sơ đồ kiến trúc tổng quan của hệ thống	17
3.1	Biểu đồ tiến trình huấn luyện mô hình qua các epochs	22
3.2	Ma trận nhầm lẫn trên tập Test	22
3.3	Giao diện hệ thống đểm phương tiện thực tế với vùng quan tâm (ROI) và các ID theo dõi.	24
3.4	Biểu đồ đường tổng hợp lưu lượng lũy kế theo thời gian	24
3.5	Biểu đồ trung bình động của tổng lưu lượng	24
3.6	Biểu đồ phát hiện các khoảng thời gian đạt đỉnh lưu lượng	25
3.7	Tỷ lệ phần trăm các loại xe.	25
3.8	Phân bố tích lũy lưu lượng theo loại xe.	25
3.9	So sánh số lượng từng loại xe theo từng mốc thời gian.	26
3.10	Giao diện thực tế của ứng dụng web trên môi trường production.	27

Danh sách bảng

2.1	Mô tả cấu trúc bảng cơ sở dữ liệu giám sát	19
2.2	Mô tả cấu trúc bảng lưu trữ lịch sử hội thoại	20
3.1	Kết quả đánh giá mô hình trên tập Test	21
3.2	Kết quả đánh giá mô hình theo từng lớp phương tiện trên tập Test	23

Chương 1

Giới thiệu

1.1 Tóm tắt

Nghiên cứu này đề xuất một hệ thống giám sát và phân tích lưu lượng giao thông tự động dựa trên thị giác máy tính và học sâu, được tối ưu hóa cho điều kiện thực tế tại Việt Nam. Hệ thống sử dụng kiến trúc YOLOv8m, được tinh chỉnh trên tập dữ liệu đặc thù gồm 1547 hình ảnh, tập trung vào 4 lớp đối tượng: ô tô, xe máy, xe tải và xe buýt.

Kết quả thực nghiệm cho thấy mô hình đạt hiệu suất cao với độ chính xác mAP@0.5 là 92.49%, Precision 85.62% và Recall 87.95%. Bên cạnh mô hình nhận diện, nghiên cứu phát triển quy trình xử lý thời gian thực đa luồng camera, tích hợp thuật toán tracking và vùng quan tâm (ROI) để loại bỏ sai số đếm trùng lặp.

Hệ thống hoàn chỉnh được xây dựng trên nền tảng FastAPI và Next.js 14, không chỉ cung cấp Dashboard giám sát trực quan mà còn tích hợp module phân tích dữ liệu nâng cao, giúp biểu diễn xu hướng lưu lượng và tỷ lệ phương tiện theo chuỗi thời gian. Đặc biệt, hệ thống tích hợp một trợ lý ảo (Chatbot) sử dụng kỹ thuật RAG (Retrieval-Augmented Generation), cho phép người dùng tra cứu và giải đáp các thắc mắc về Luật Giao thông đường bộ Việt Nam một cách chính xác và tự động. Với tốc độ xử lý ổn định từ 10-20 FPS, hệ thống chứng minh tính khả thi và hiệu quả trong triển khai thực tiễn.

1.2 Bài toán đặt ra

Quản lý giao thông đô thị là thách thức cấp thiết tại các siêu đô thị như Hà Nội và Thành phố Hồ Chí Minh. Việc thu thập dữ liệu lưu lượng chính xác là nền tảng cho quy hoạch hạ tầng và cảnh báo ùn tắc. Tuy nhiên, các phương pháp truyền thống như đếm thủ công hoặc dùng cảm biến vật lý còn tồn tại nhiều hạn chế về chi phí, nhân lực và khả năng phân loại.

Nghiên cứu này tập trung giải quyết các thách thức kỹ thuật chính nhằm xây dựng giải pháp giao thông thông minh toàn diện:

- **Nhận diện và phân loại:** Đảm bảo độ chính xác cao cho 4 loại phương tiện phổ biến trong điều kiện môi trường đa dạng.
- **Xử lý và Phân tích dữ liệu:** Khắc phục hiện tượng đếm trùng lặp và chuyển đổi dữ liệu thành các **biểu đồ phân tích chuyên sâu**, hỗ trợ nhận diện giờ cao điểm và đặc điểm dòng giao thông.
- **Hỗ trợ pháp lý:** Giải quyết vấn đề thiếu hụt thông tin pháp luật của người tham gia giao thông bằng cách tích hợp Chatbot thông minh, giúp tra cứu nhanh các quy định và mức phạt dựa trên dữ liệu văn bản luật chính thống.
- **Hiệu năng thực:** Đảm bảo khả năng xử lý thời gian thực (*realtime*) trên nhiều luồng camera đồng thời với kiến trúc phần mềm có khả năng mở rộng.

Chương 2

Phương pháp & Triển khai

2.1 Phương pháp

Phần này trình bày các nền tảng lý thuyết cốt lõi được sử dụng trong nghiên cứu, bao gồm bài toán nhận diện vật thể, kiến trúc mạng nơ-ron tích chập, thuật toán YOLOv8 và phương pháp Fine-tuning

2.1.1 Lý thuyết về mạng nơ-ron tích chập

YOLOv8m về bản chất là một mạng Deep CNN, được cấu thành từ các khối xây dựng cơ bản nhằm chuyển đổi dữ liệu hình ảnh thành các đặc trưng có ý nghĩa:

- **Convolutional Layer:** Đây là thành phần cốt lõi đóng vai trò là bộ trích xuất đặc trưng. Thông qua cơ chế trượt các kernels trên toàn bộ ảnh đầu vào, lớp này học cách phát hiện các mẫu hình ảnh theo cấp độ trừu tượng tăng dần: từ các đặc trưng đơn giản như cạnh, góc, màu sắc đến các đặc trưng phức tạp như mắt, bánh xe hay khuôn mặt .
- **Pooling Layer:** Mô hình thường sử dụng kỹ thuật Max Pooling để giảm kích thước của dữ liệu . Quá trình này không chỉ giúp giảm lượng tham số tính toán và nguy cơ overfitting mà còn giữ lại các đặc trưng nổi bật nhất, đồng thời tạo ra tính bất biến nhẹ đối với các dịch chuyển vị trí nhỏ của vật thể.
- **Activation Function :** Để đảm bảo tính phi tuyến tính, YOLOv8 sử dụng hàm kích hoạt SiLU (Sigmoid Linear Unit), được định nghĩa bởi công thức:

$$f(x) = x \cdot \sigma(x) \quad (2.1)$$

Tính chất đặc biệt của SiLU cho phép mạng nơ-ron học và mô hình hóa các dữ liệu phức tạp mà các phép biến đổi tuyến tính đơn thuần không thể thực hiện được .

2.1.2 Kiến trúc YOLOv8m

Trong nghiên cứu này, phiên bản YOLOv8m (Medium) được lựa chọn nhờ sự cân bằng tối ưu giữa tốc độ và độ chính xác, sở hữu khoảng 25.9 triệu tham số. Mô hình hoạt động dựa trên nguyên lý chia hình ảnh đầu vào thành một lưới kích thước $S \times S$ (grid cells). Về mặt lý thuyết, trách nhiệm phát hiện vật thể được gán cho ô lưới cụ thể chứa tâm của vật thể đó. Cơ chế này cho phép mô hình xử lý song song toàn bộ bức ảnh trong một lần chạy duy nhất.

Về mặt cấu trúc, kiến trúc mạng của YOLOv8m được chia thành ba khối chức năng rõ rệt. Đầu tiên, khối Backbone chịu trách nhiệm trích xuất đặc trưng, sử dụng kiến trúc CSPDarknet cải tiến tích hợp module C2f (Cross-Stage Partial with 2 Convolutions). Module C2f hoạt động bằng cách chia luồng tín hiệu thành hai nhánh: một nhánh đi qua các lớp tính toán sâu và một nhánh đi tắt, sau đó hợp nhất lại ở đầu ra. Cơ chế này giúp làm giàu luồng gradient, cho phép mạng học sâu hơn mà không bị mất mát thông tin. Cuối cùng, lớp SPPF được sử dụng để mở rộng trường tiếp nhận, giúp mô hình nắm bắt được ngữ cảnh toàn cảnh của vật thể

Tiếp theo, khối Neck thực hiện nhiệm vụ hợp nhất đặc trưng thông qua cấu trúc PANet (Path Aggregation Network) để giải quyết bài toán đa tỉ lệ. Nhiệm vụ chính của nó là trộn lẫn các đặc trưng ngữ nghĩa (từ tầng sâu) với các đặc trưng vị trí (từ tầng nông) thông qua các đường dẫn từ trên xuống và từ dưới lên . Kỹ thuật này tăng cường đáng kể khả năng nhận diện các vật thể ở nhiều kích thước khác nhau .

Phần Head của YOLOv8m mang hai cải tiến mang tính đột phá so với các thế hệ trước . Cụ thể, mô hình sử dụng kiến trúc Decoupled Head để tách riêng nhánh Phân loại và nhánh Hồi quy hợp, giúp tăng độ chính xác

hội tụ do hai nhiệm vụ này yêu cầu các loại đặc trưng khác nhau. Bên cạnh đó, cơ chế *Anchor-free Detection* được áp dụng để loại bỏ hoàn toàn các Anchor Box định sẵn, thay vào đó dự đoán trực tiếp tâm vật thể. Điều này giúp mô hình linh hoạt hơn khi xử lý các vật thể có hình dạng bất thường và giảm thiểu các tính toán dư thừa.

Bên cạnh kiến trúc mạng, quá trình tối ưu hóa trọng số được điều hướng bởi Loss Function, được định nghĩa bởi công thức:

$$L_{total} = \lambda_{box} L_{box} + \lambda_{cls} L_{cls} + \lambda_{dfi} L_{dfi} \quad (2.2)$$

Trong đó, thành phần Box Regression Loss (L_{box}) sử dụng CIoU Loss (Complete IoU) để tính toán sai số dựa trên độ chồng lấn, khoảng cách tâm và tỷ lệ khung hình, giúp hộp dự đoán nhanh về hộp thực tế. Thành phần Classification Loss (L_{cls}) sử dụng BCEWithLogitsLoss (Binary Cross Entropy) nhằm đo lường sai số xác suất khi phân loại vật thể. Cuối cùng, Distribution Focal Loss (L_{dfi}) được tích hợp để tinh chỉnh biên của hộp giới hạn, đặc biệt phát huy hiệu quả trong các trường hợp biên vật thể bị mờ hoặc che khuất.

2.1.3 Phương pháp Fine-tuning

Để giải quyết thách thức về hạn chế dữ liệu huấn luyện, báo cáo áp dụng phương pháp Fine-tuning thay vì huấn luyện mô hình từ đầu. Fine-tuning là một kỹ thuật của transfer learning, trong đó mô hình đã được huấn luyện trước trên một tập dữ liệu lớn (COCO) được tiếp tục huấn luyện trên tập dữ liệu mới (phương tiện giao thông Việt Nam). Trong quá trình này, tất cả các layers của mô hình được cập nhật, thường với learning rate nhỏ hơn so với training from scratch để tinh chỉnh các trọng số một cách nhẹ nhàng, phù hợp với bài toán cụ thể mà không phá vỡ các đặc trưng hữu ích đã học từ dữ liệu lớn.

Quá trình Fine-tuning mô hình YOLOv8 được thực hiện bằng cách train lại toàn bộ mô hình (không freeze layers nào) từ weights pre-trained. Mô hình được khởi tạo với trọng số từ `yolov8m.pt` đã được huấn luyện trên COCO dataset, sau đó tất cả các layers (bao gồm cả Backbone và Head) đều được cập nhật thông qua quá trình backpropagation trên tập dữ liệu phương tiện giao thông Việt Nam. Chiến lược này cho phép mô hình điều chỉnh cả các đặc trưng cấp thấp trong Backbone và các đặc trưng cấp cao trong Head để phù hợp với đặc thù của bài toán mới.

Việc áp dụng Fine-tuning toàn bộ mô hình mang lại các lợi ích sau:

- Mô hình có thể tận dụng tối đa kiến thức đã học từ COCO dataset, không chỉ ở mức đặc trưng cơ bản mà cả ở mức phân loại
- Với learning rate được điều chỉnh phù hợp (0.01 ban đầu, giảm dần theo cosine annealing), mô hình có thể tinh chỉnh các trọng số một cách nhẹ nhàng, tránh phá vỡ các đặc trưng hữu ích đã học
- Quá trình hội tụ nhanh hơn so với training from scratch vì mô hình bắt đầu từ một điểm khởi đầu tốt thay vì khởi tạo ngẫu nhiên
- Với tập dữ liệu 1547 ảnh, fine-tuning toàn bộ mô hình có thể đạt được khả năng tổng quát hóa tốt khi được kết hợp với các kỹ thuật regularization như data augmentation và early stopping, giúp giảm thiểu nguy cơ overfitting

2.1.4 Các chỉ số đánh giá hiệu quả

Hiệu quả của mô hình sau huấn luyện được định lượng thông qua các chỉ số sau:

- Intersection over Union (IoU):** Đo độ chồng lấn giữa khung dự đoán (B_p) và khung thực tế (B_{gt}).

$$\text{IoU} = \frac{\text{Diện tích phần giao (Area of Overlap)}}{\text{Diện tích phần hợp (Area of Union)}}$$

- Precision (Độ chính xác):** Trong số các vật thể mô hình phát hiện, bao nhiêu % là đúng?

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall (Độ phủ/Độ nhạy):** Mô hình tìm được bao nhiêu % vật thể thực sự có trong ảnh?

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:** Chỉ số tổng hợp cân bằng giữa Precision và Recall, đặc biệt hữu ích khi cần đánh giá sự cân bằng giữa độ chính xác và độ phủ.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Mean Average Precision (mAP):** Là chỉ số quan trọng nhất, là trung bình của AP trên tất cả các lớp.
 - *mAP@0.5*: Độ chính xác trung bình khi ngưỡng IoU chấp nhận là 0.5.
 - *mAP@0.5:0.95*: Trung bình cộng của mAP tại các ngưỡng IoU tăng dần từ 0.5 đến 0.95 (bước nhảy 0.05). Chỉ số này đánh giá khắt khe về độ khít của hộp dự đoán.

2.1.5 Dữ liệu và tiền xử lý

Dữ liệu đóng vai trò là nguyên liệu đầu vào quyết định trực tiếp đến khả năng học và tổng quát hóa của mô hình. Nghiên cứu sử dụng bộ dữ liệu phương tiện giao thông Việt Nam được lấy từ Roboflow. Tổng quy mô bao gồm 1547 hình ảnh, tập trung vào 4 lớp đối tượng chính: *car* (ô tô), *motor* (xe máy), *truck* (xe tải), *bus* (xe buýt). Các mẫu dữ liệu đảm bảo tính đa dạng về điều kiện môi trường và ánh sáng thực tế tại Việt Nam.

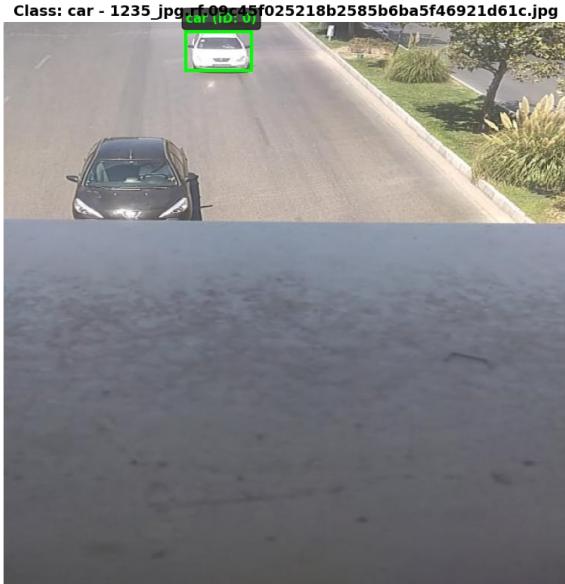
Tiền xử lý

Để phù hợp với kiến trúc đầu vào của YOLOv8m và tối ưu hóa cho bài toán nhận diện phương tiện, dữ liệu đã trải qua các bước xử lý:

- **Lọc và gán lại nhãn:** Dữ liệu gốc ban đầu bao gồm nhiều lớp đối tượng khác nhau. Quá trình tiền xử lý đã thực hiện ánh xạ lại các nhãn để quy về 4 nhóm phương tiện mục tiêu nêu trên, giúp mô hình tập trung học các đặc trưng quan trọng nhất.
- **Định dạng ảnh:** Mọi hình ảnh đầu vào được thay đổi kích thước về độ phân giải cố định 640×640 pixels. Kỹ thuật này giúp cân bằng giữa chi phí tính toán và khả năng giữ lại chi tiết của các vật thể.
- **Phân chia dữ liệu:** Bộ dữ liệu được chia ngẫu nhiên thành ba tập con theo tỷ lệ **80:10:10**, cụ thể bao gồm: 1235 ảnh cho Training, 156 ảnh cho Validation và 156 ảnh cho Testing. Việc phân chia này đảm bảo mô hình được đánh giá khách quan trên những dữ liệu chưa từng "nhìn thấy" trong quá trình học.

Tăng cường dữ liệu

Nhằm khắc phục hạn chế về số lượng mẫu và nâng cao độ bền vững, kỹ thuật tăng cường dữ liệu Mosaic được áp dụng trong quá trình huấn luyện. Bằng cách ghép 4 hình ảnh ngẫu nhiên thành một lưới 2×2 , mô hình bị buộc phải học cách nhận diện vật thể trong các bối cảnh phức tạp và ở các tỷ lệ kích thước khác nhau. Ngoài ra, các phép biến đổi như lật ngang và chỉnh độ bão hòa màu cũng được tích hợp.



(a) Car (Ô tô)



(b) Motor (Xe máy)



(c) Truck (Xe tải)



(d) Bus (Xe buýt)

Hình 2.1: Mẫu dữ liệu đại diện cho 4 lớp phương tiện trong bộ dữ liệu.

2.2 Triển khai

2.2.1 Môi trường và công cụ phát triển

Hệ thống được phát triển trên nền tảng **Python** (phiên bản 3.10 trở lên), lựa chọn dựa trên sự hỗ trợ mạnh mẽ của cộng đồng mã nguồn mở trong lĩnh vực trí tuệ nhân tạo. Kiến trúc công nghệ được chia thành các phân hệ chính như sau:

- **Xử lý thị giác máy tính và học sâu:**
 - **PyTorch:** Framework nền tảng cho các tính toán tensor và xây dựng mô hình học sâu.
 - **Ultralytics YOLO (v8.0+):** Triển khai kiến trúc mạng nơ-ron tích chập (CNN) hiện đại cho bài toán phát hiện và theo dõi đối tượng.
 - **OpenCV:** Thư viện xử lý ảnh cơ bản, chịu trách nhiệm đọc luồng video, tiền xử lý khung hình và hiển thị kết quả trực quan (bounding boxes).
- **Phân tích và trực quan hóa dữ liệu:**
 - **NumPy & Pandas:** Cung cấp cấu trúc dữ liệu hiệu năng cao để thao tác trên mảng đa chiều và xử lý chuỗi thời gian của lưu lượng giao thông.
 - **Matplotlib & Seaborn:** Công cụ trực quan hóa dùng để phân tích phân phối dữ liệu và vẽ biểu đồ đánh giá hiệu suất mô hình.
- **Phát triển hệ thống (Backend & Database):**
 - **FastAPI:** Framework hiệu năng cao hỗ trợ xử lý bất đồng bộ (asynchronous), tối ưu cho việc xây dựng RESTful API và streaming dữ liệu qua WebSocket.
 - **SQLAlchemy & PostgreSQL:** Hệ quản trị cơ sở dữ liệu quan hệ và ORM hỗ trợ async/await để lưu trữ bền vững dữ liệu phân tích giao thông.
- **Giao diện người dùng (Frontend):**
 - **Next.js 14 (React 18):** Framework frontend hiện đại giúp tối ưu hóa hiệu năng render và trải nghiệm người dùng.
 - **TypeScript & Tailwind CSS:** Đảm bảo tính an toàn kiểu dữ liệu (type safety) và xây dựng giao diện đáp ứng (responsive) nhanh chóng.
- **Module Trợ lý ảo (AI Chatbot):**
 - Ứng dụng kiến trúc RAG (Retrieval-Augmented Generation) kết hợp LangChain để điều phối luồng dữ liệu, ChromaDB làm cơ sở dữ liệu vector và Google Gemini làm mô hình ngôn ngữ lớn (LLM) để tư vấn luật giao thông.

2.2.2 Cấu trúc và tổ chức mã nguồn

Mã nguồn dự án được tổ chức theo nguyên tắc Modular Architecture, nhằm tách biệt rõ ràng giữa các thành phần cấu hình, xử lý dữ liệu và logic mô hình. Cách tiếp cận này không chỉ tăng tính bảo trì mà còn tạo điều kiện cho việc mở rộng và tái sử dụng mã nguồn. Cấu trúc thư mục được phân chia thành các phân hệ chức năng như sau:

1. Phân hệ Cấu hình

Phân hệ này đóng vai trò xương sống trong việc quản lý toàn bộ tham số vận hành, đảm bảo nguyên tắc tách biệt giữa logic mã nguồn và dữ liệu cấu hình môi trường (Environment Parity). Hệ thống được thiết kế để hoạt động linh hoạt trên đa nền tảng (Local, Staging, Production) thông qua cơ chế quản lý tập trung:

- **Quản lý biến môi trường (.env):** Lưu trữ các thông tin nhạy cảm và đặc thù cho từng môi trường triển khai nhằm đảm bảo tính bảo mật và dễ dàng thay đổi mà không cần can thiệp mã nguồn.
 - *Database Config:* Chuỗi kết nối PostgreSQL (`DATABASE_URL`), thông tin xác thực user/password.
 - *AI Model Config:* Đường dẫn tới mô hình ngôn ngữ (`CHATBOT_MODEL`) và cơ sở dữ liệu vector (`VECTOR_DB_PATH`).
 - *System Flags:* Chế độ gõ lỗi (`DEBUG`), các khóa API và thiết lập bảo mật.

- **Quản lý cấu hình ứng dụng (core/config.py):** Sử dụng thư viện Pydantic (BaseSettings) để nạp, kiểm tra kiểu dữ liệu (validation) và cung cấp các giá trị cấu hình toàn cục cho ứng dụng.
 - Định nghĩa các đường dẫn hệ thống quan trọng: BASE_DIR, DATA_DIR, LOG_DIR.
 - Thiết lập các thông số API như tiền tố endpoint (API_V1_STR) và tên dự án.
 - Tự động hóa việc kiểm tra tính hợp lệ của cấu hình ngay khi khởi động ứng dụng.
- **Cấu hình tham số mô hình (configs/*.yaml):** Các tệp YAML được sử dụng để quản lý các siêu tham số chuyên biệt cho thuật toán thị giác máy tính, cho phép tinh chỉnh mô hình mà không cần biên dịch lại:
 - Các ngưỡng phát hiện đối tượng: Confidence Threshold, IOU Threshold.
 - Cấu hình không gian: Danh sách vùng nhận diện phương tiện (roi) cho từng camera được định nghĩa dưới dạng mảng tọa độ NumPy, hỗ trợ giám sát đa luồng.

2. Phân hệ mô hình

Phân hệ mô hình là thành phần cốt lõi của hệ thống nhận diện, chứa các module quản lý quy trình huấn luyện và đánh giá mô hình. Module `model_trainer.py` đảm nhiệm toàn bộ quy trình huấn luyện thông qua hàm `train_model`, thực hiện fine-tuning mô hình YOLOv8 trên tập dữ liệu phương tiện giao thông. Module này cũng cung cấp hàm `valid_model` để đánh giá mô hình trên tập validation/test và tính toán các chỉ số đánh giá như Precision, Recall và mAP.

3. Phân hệ Backend

Backend được phát triển trên nền tảng FastAPI, đóng vai trò là bộ não điều phối trung tâm, chịu trách nhiệm xử lý các luồng video và quản lý dữ liệu. Kiến trúc được chia thành 2 module chính bao gồm:

- **Service Layer:** Tầng này đóng vai trò là cầu nối để giao diện Web có thể lấy dữ liệu từ hệ thống xử lý bên dưới. Cụ thể, nó chịu trách nhiệm truyền hình ảnh camera thời gian thực (qua WebSocket) và gửi lại câu trả lời từ Chatbot cho người dùng.
 - **Road Services:**
 - * `AnalyzeOnRoadBase.py`: Module nòng cốt tích hợp mô hình YOLOv8m để phát hiện và theo dõi (tracking) 4 loại phương tiện. Hệ thống áp dụng chiến lược tối ưu hiệu năng bao gồm giảm độ phân giải xử lý xuống chuẩn SD (854×480) và kỹ thuật *Frame Skipping* ($k = 3$) để giảm tải CPU xuống 70%.
 - * `AnalyzeOnRoad.py`: Dóng vai trò là module bao bọc (Wrapper) cho các tác vụ đa tiến trình. Nó cung cấp hàm `run_analyzer` hoạt động như một điểm nhập (entry point) để khởi chạy độc lập từng instance phân tích video trên một Process riêng biệt, đồng thời xử lý ngoại lệ và gửi tín hiệu trạng thái về tiến trình cha.
 - **RAG Services:**
 - * `ChatBotAgent.py`: Điều phối luồng hội thoại, quản lý lịch sử chat và ngữ cảnh (Context Window) để duy trì hội thoại tự nhiên.
 - * `vector_store.py`: Quản lý kết nối với ChromaDB, thực hiện truy vấn vector (similarity search) để tìm kiếm các đoạn văn bản luật liên quan nhất.
 - * `document_process.py & traffic_law_processor.py`: Module tiền xử lý văn bản luật, thực hiện chia nhỏ văn bản (chunking) và tạo embeddings sử dụng mô hình ngôn ngữ chuyên biệt.
- **API Layer:**
 - `api_vehicles.py`: Cung cấp các WebSocket endpoints để streaming dữ liệu thời gian thực và RESTful API tích hợp Pandas để phân tích dữ liệu lịch sử.
 - `api_chatbot.py`: Tiếp nhận câu hỏi từ người dùng và trả về câu trả lời pháp lý chính xác thông qua RAG Services.

- **Background Workers:** Triển khai chiến lược Hybrid Data Storage với chu kỳ đồng bộ hóa 10 giây/lần, chuyển dữ liệu từ bộ nhớ đệm RAM xuống cơ sở dữ liệu PostgreSQL để đảm bảo hiệu năng cao và độ bền dữ liệu.

4. Phân hệ Phân tích (analysis/)

Phân hệ phân tích cung cấp các công cụ xử lý và phân tích dữ liệu thống kê giao thông theo thời gian thực, bao gồm:

- **Module analyze.py:** Triển khai pipeline xử lý dữ liệu bao gồm resampling theo chu kỳ thời gian, tính toán tỷ lệ phần trăm theo từng loại phương tiện, phát hiện đỉnh lưu lượng (peak detection) và xuất kết quả ra các định dạng CSV/JSON
- **Module load_data.py:** Thực hiện đọc và chuẩn hóa dữ liệu từ file log JSON, sử dụng kỹ thuật binary-safe tail reading để chỉ đọc phần cuối của file, giảm thiểu chi phí I/O và tối ưu hiệu năng khi xử lý file lớn
- **Module visualize.py:** Cung cấp khả năng trực quan hóa dữ liệu với Matplotlib, hỗ trợ cập nhật realtime thông qua cơ chế in-place updates để tránh việc vẽ lại toàn bộ biểu đồ, giảm thiểu hiện tượng nhấp nháy
- **Module realtime_loop.py:** Chứa vòng lặp chạy liên tục để cập nhật phân tích và visualization theo thời gian thực

5. Phân hệ Frontend (frontend/app/)

Ứng dụng web được xây dựng với Next.js 14 sử dụng App Router, một kiến trúc mới cho phép tối ưu hóa hiệu năng và trải nghiệm người dùng. Cấu trúc phân hệ bao gồm:

- **Thư mục components/:** Chứa thư viện component tái sử dụng: `stream/VideoPlayer.tsx` (hiển thị video stream từ WebSocket), `stats/RealtimeStats.tsx` (thống kê realtime), `charts/VehicleLineChart` (biểu đồ đường xu hướng), `charts/RealtimeCounterChart` (biểu đồ cột phân bố)
- **Thư mục api/:** Chứa Next.js API routes hoạt động như proxy layer để chuyển tiếp requests đến backend FastAPI
- **Thư mục lib/:** Chứa utilities và type definitions cho TypeScript, đảm bảo tính an toàn kiểu dữ liệu trong toàn bộ ứng dụng

2.2.3 Xây dựng mô hình

Quá trình xây dựng mô hình nhận diện phương tiện được thực hiện theo một quy trình có hệ thống, từ việc chuẩn bị dữ liệu đến đánh giá và tối ưu hóa mô hình. Mỗi bước được thiết kế cẩn thận để đảm bảo chất lượng và hiệu quả của mô hình cuối cùng.

Bước 1: Chuẩn bị và tiền xử lý dữ liệu

Tập dữ liệu ban đầu bao gồm 1547 hình ảnh được thu thập từ môi trường giao thông thực tế tại Việt Nam, được gán nhãn thủ công cho 4 lớp phương tiện chính: Car (ô tô), Motor (xe máy), Truck (xe tải) và Bus (xe buýt). Để đảm bảo tính toàn vẹn của dữ liệu, hệ thống tự động quét và xác minh sự tương ứng một-một giữa file ảnh và file nhãn theo định dạng YOLO (định dạng `.txt` với tọa độ được chuẩn hóa). Tất cả tọa độ của bounding boxes được chuẩn hóa về khoảng $[0, 1]$ và trải qua quá trình kiểm tra tính hợp lệ, bao gồm việc đảm bảo tọa độ không vượt quá biên ảnh và diện tích hộp đủ lớn để tránh các đối tượng quá nhỏ không có giá trị huấn luyện. Tập dữ liệu được chia ngẫu nhiên theo tỷ lệ 80:10:10 thành ba tập con: Training (1235 ảnh), Validation (156 ảnh) và Testing (156 ảnh), đảm bảo tính đại diện và khả năng đánh giá khách quan. File cấu hình `data.yaml` được tạo với định nghĩa đầy đủ đường dẫn đến các tập dữ liệu và danh sách tên lớp, phục vụ cho quá trình huấn luyện.

Bước 2: Khởi tạo mô hình Pre-trained

Hệ thống áp dụng phương pháp Fine-tuning, một kỹ thuật hiệu quả khi làm việc với tập dữ liệu có quy mô hạn chế. Quá trình bắt đầu từ mô hình YOLOv8m đã được huấn luyện trước trên tập dữ liệu COCO, một bộ dữ liệu lớn và đa dạng với hơn 80 lớp đối tượng. Trọng số pre-trained được tải từ kho lưu trữ chính thức của Ultralytics (`yolov8m.pt` với khoảng 25.9 triệu tham số), cho phép mô hình kế thừa các đặc trưng cơ bản đã được học từ dữ liệu lớn. Mô hình được khởi tạo thông qua lớp YOLO từ thư viện Ultralytics với cú pháp `YOLO('yolov8m.pt')`, tự động tải kiến trúc mạng và trọng số đã được tối ưu. Tất cả các layers của mô hình (Backbone và Head) sẽ được fine-tune trên tập dữ liệu mới, không có layers nào bị freeze. Cấu hình thiết bị tính toán (GPU/CPU) được xác định tự động dựa trên phần cứng có sẵn hoặc có thể chỉ định thủ công trong file `training_config.yaml` (qua tham số `device`) để tối ưu hóa hiệu năng.

Bước 3: Cấu hình tham số huấn luyện

Các siêu tham số (hyperparameters) được định nghĩa tập trung trong file `training_config.yaml` để dễ dàng điều chỉnh và thử nghiệm. Các tham số chính được cấu hình như sau:

- **Epochs và Early Stopping:** 100 epochs với cơ chế Early Stopping (patience: 30), tự động dừng nếu không có cải thiện về độ chính xác trên tập validation trong 30 epochs liên tiếp, nhằm tránh overfitting và tiết kiệm thời gian tính toán

- **Batch size:** 16 mẫu mỗi batch, giá trị cân bằng giữa tốc độ xử lý và khả năng sử dụng bộ nhớ, phù hợp với phần cứng phổ biến
- **Image size:** 640x640 pixels, kích thước chuẩn được khuyến nghị cho YOLOv8, đảm bảo sự cân bằng giữa độ chính xác và hiệu năng
- **Learning rate:** Giá trị ban đầu 0.01, giảm dần theo chiến lược cosine annealing, cho phép mô hình học nhanh ở giai đoạn đầu và tinh chỉnh dần ở giai đoạn sau
- **Data Augmentation:** Framework YOLO tự động áp dụng các kỹ thuật augmentation mặc định bao gồm Mosaic augmentation (tỷ lệ 1.0), horizontal flip (xác suất 0.5) và các biến đổi HSV (hue: 0.015, saturation: 0.7, value: 0.4) để mô phỏng các điều kiện ánh sáng khác nhau

Bước 4: Quá trình Fine-tuning

Module `model_trainer.py` điều phối toàn bộ quá trình fine-tuning thông qua hàm `train_model()`, nhận vào đường dẫn đến weights pre-trained và file cấu hình. Framework Ultralytics YOLO tự động thực hiện một chuỗi các bước được tối ưu hóa theo thứ tự sau:

- Tải dữ liệu từ file `data.yaml` và áp dụng các kỹ thuật augmentation đã được cấu hình
- Thực thi vòng lặp huấn luyện với các phép tính forward propagation để dự đoán và backward propagation để cập nhật trọng số thông qua gradient descent
- Dánh giá mô hình trên tập validation sau mỗi epoch để theo dõi tiến trình học
- Lưu checkpoint tốt nhất tự động dựa trên chỉ số mAP@0.5:0.95, một chỉ số tổng hợp đánh giá độ chính xác ở nhiều ngưỡng IoU khác nhau

Tất cả kết quả được lưu tự động vào thư mục `runs/detect/train/`, bao gồm `weights/best.pt` (trọng số tốt nhất cho inference), `weights/last.pt` (checkpoint cuối cùng), các biểu đồ đánh giá trực quan (confusion matrix, precision-recall curves, F1 curves) và file `results.csv` chứa log chi tiết các chỉ số đánh giá sau mỗi epoch.

Bước 5: Đánh giá

Sau khi hoàn tất quá trình huấn luyện, mô hình được đánh giá toàn diện trên tập test độc lập để đảm bảo khả năng tổng quát hóa. Quá trình đánh giá bao gồm:

- **Đánh giá chỉ số:** Hàm `valid_model()` thực hiện inference trên tập validation/test và tính toán các chỉ số đánh giá: Precision (tỷ lệ dự đoán đúng), Recall (tỷ lệ phát hiện được), F1-Score (chỉ số tổng hợp cân bằng giữa Precision và Recall), mAP@0.5 (mean Average Precision tại ngưỡng IoU 0.5) và mAP@0.5:0.95 (mean Average Precision trung bình tại các ngưỡng IoU từ 0.5 đến 0.95)

2.2.4 Đếm phương tiện và xử lý dữ liệu trả về

Hệ thống đếm xe được thiết kế để hoạt động ổn định trong môi trường thời gian thực (Real-time), xử lý đồng thời nhiều luồng video với độ trễ thấp. Giải pháp tích hợp các kỹ thuật thị giác máy tính tiên tiến và kiến trúc phần mềm hướng dữ liệu (Data-driven Architecture) để giải quyết bài toán đếm trùng lặp và tắc nghẽn dữ liệu.

1. Kiến trúc xử lý đa tiến trình (Multiprocessing Architecture)

Để khắc phục giới hạn Global Interpreter Lock (GIL) của Python khi xử lý các tác vụ nặng về tính toán (CPU-bound), hệ thống áp dụng kỹ thuật Multiprocessing. Mỗi Camera được khởi chạy trong một tiến trình (Process) độc lập, sở hữu không bộ nhớ riêng biệt. Điều này đảm bảo:

- **Tính cách ly (Isolation):** Lỗi tại một luồng camera (ví dụ: mất kết nối mạng) không gây ảnh hưởng đến toàn bộ hệ thống.
- **Tận dụng đa nhân:** Khai thác tối đa sức mạnh của CPU đa nhân để chạy song song các mô hình AI.
- **Giao tiếp liên tiến trình (IPC):** Sử dụng cơ chế bộ nhớ chia sẻ (`Manager.dict`) để truyền tải dữ liệu đếm xe và khung hình (đã nén JPEG) từ các tiến trình con về tiến trình chính (API Server) với độ trễ gần như bằng 0.

2. Quy trình phát hiện và theo dõi (Detection & Tracking Pipeline)

Module `AnalyzeOnRoadBase` triển khai quy trình xử lý logic nhận diện và theo dõi phương tiện với các tham số tối ưu hóa:

- **Mô hình:** Sử dụng YOLOv8m (đã fine-tuning) để phát hiện đối tượng. Chế độ **track** được kích hoạt (**persist=True**) để gán ID duy nhất cho phương tiện qua các khung hình liên tiếp.
- **Chiến lược đầu vào:** Luồng video được lấy ở độ phân giải HD (720p) để đảm bảo độ nét, sau đó được **Resize** xuống chuẩn SD (854×480) cho quá trình suy luận (Inference). Kỹ thuật này giúp cân bằng giữa tốc độ xử lý và khả năng nhận diện các vật thể nhỏ/xa.
- **Frame Skipping:** Áp dụng kỹ thuật nhảy khung hình với tham số $k = 3$ (xử lý 1 frame, bỏ qua 3 frame). Vị trí của phương tiện trong các frame bị bỏ qua được nội suy bởi thuật toán tracking, giúp giảm tải CPU xuống 70% mà không làm giảm đáng kể độ chính xác đếm.

3. Thuật toán đếm số phương tiện dựa trên vùng roi (ROI-based Counting)

Hệ thống sử dụng vùng quan sát (**roi**) để lọc các đối tượng không liên quan. Logic đếm hoạt động dựa trên máy trạng thái (State Machine):

- Mỗi phương tiện (Track ID) được theo dõi trạng thái vị trí so với vùng ROI bằng thuật toán Ray Casting (thông qua hàm `cv2.pointPolygonTest`).
- Sự kiện đếm (Counting Event) chỉ được kích hoạt khi trọng tâm (centroid) của phương tiện chuyển trạng thái từ ngoài → trong vùng ROI.
- Cơ chế này loại bỏ hoàn toàn việc đếm trùng lặp khi xe di chuyển chậm hoặc dừng lại trong vùng quan sát.

4. Chiến lược lưu trữ dữ liệu Lai (Hybrid Data Strategy)

Khác với các hệ thống truyền thống ghi log ra file (gây nghẽn ổ cứng), nghiên cứu này đề xuất mô hình lưu trữ hai lớp:

- **Lớp dữ liệu nóng (Hot Data - RAM):** Dữ liệu đếm xe và FPS tức thời được lưu trên RAM để phục vụ việc hiển thị Dashboard thời gian thực thông qua WebSocket (độ trễ < 50ms).
- **Lớp dữ liệu lạnh (Cold Data - PostgreSQL):** Một tác vụ nền (Background Worker) chạy định kỳ mỗi 10 giây, thực hiện đồng bộ dữ liệu từ RAM xuống cơ sở dữ liệu quan hệ PostgreSQL.

Cách tiếp cận này giúp loại bỏ nút thắt cổ chai I/O (I/O Bottleneck), cho phép hệ thống vận hành mượt mà ngay cả khi lưu lượng ghi dữ liệu lớn.

5. Tối ưu hóa độ trễ (Latency Optimization)

Để đảm bảo trải nghiệm "Real-time" trên giao diện người dùng, hệ thống áp dụng các tinh chỉnh mức thấp:

- Thiết lập `cv2.CAP_PROP_BUFFERSIZE = 1` để ép OpenCV luôn lấy khung hình mới nhất từ camera, loại bỏ hiện tượng trễ hình tích lũy.
- Nén ảnh JPEG với chất lượng 65% trước khi gửi qua mạng nội bộ, giảm băng thông truyền tải nhưng vẫn giữ được độ chi tiết cần thiết cho người quan sát.

2.2.5 Phân tích dữ liệu

1. Nguồn dữ liệu (thực tế):

Dự án đọc dữ liệu từ cơ sở dữ liệu qua bảng `traffic_logs`. Bản ghi được tạo bởi hai nguồn chính:

- **AnalyzeOnRoadBase** (video processing): mỗi `save_interval` hàm `_check_and_save()` ghi một `TrafficLog` (các cột: `count_car, count_motor, count_bus, count_truck, total_vehicles, timestamp`).
- **Background worker** (`save_stats_to_db_worker` trong `api_vehicles.py`): có thể ghi snapshot từ shared-state vào DB.

2. Quy trình phân tích trên backend:

- Các endpoint trong `backend/app/api/api_vehicles.py` (`load_traffic_df`, `time-series`, `peaks`, `rolling-avg`) đọc trực tiếp từ DB, chuyển `timestamp` sang timezone local (UTC+7), `resample` (thường "1min" hoặc "5min") và tính toán.
- Resampling thực hiện bằng `pandas.resample` (lấy `max` rồi `diff()` để có lưu lượng theo khoảng); backend tính histogram, boxplot summaries, rolling mean và đánh dấu peaks (có thể bằng quantile hoặc 3-sigma).

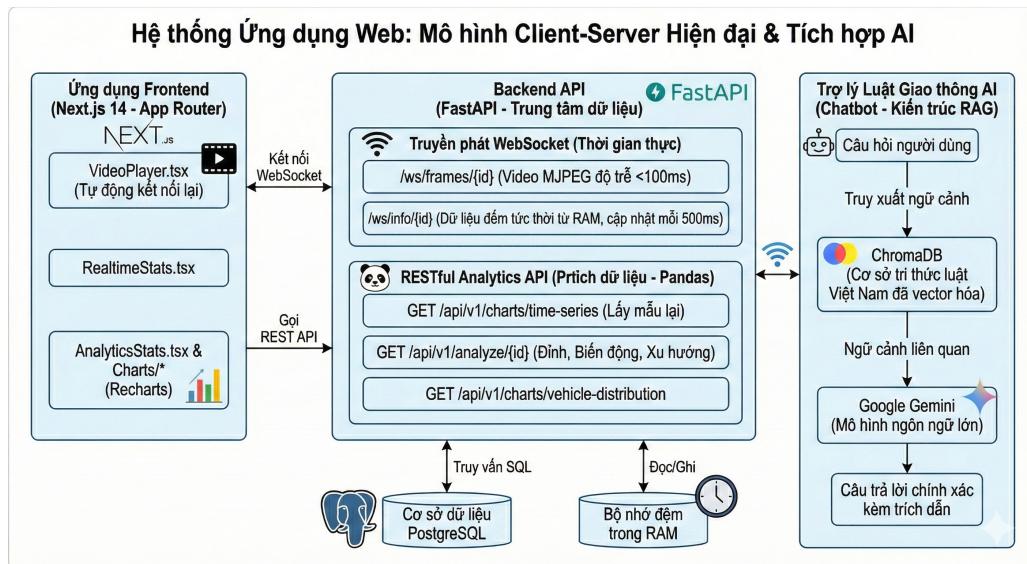
3. Vai trò của thư mục `analysis/` (mô-đun hỗ trợ):

- **analysis/analyze.py**: thư viện helper phục vụ phân tích (aggregate_timeseries, compute_percentages, detect_peaks) và tiện ích đọc thử từ DB; dùng cho phân tích offline và kiểm thử.
- **analysis/visualize.py** và **analysis/realtimeloop.py**: công cụ trực quan hóa và vòng lặp demo (ảnh/PNG hoặc interactive plots) phục vụ khảo sát dữ liệu, không phải phần bắt buộc của luồng server production.

4. **File export (dùng cho debug, không phải luồng sản xuất)**: Backend và frontend giao tiếp qua DB/API; các endpoint trả JSON cho frontend. Hàm export_for_backend tồn tại để sinh CSV/JSON phục vụ báo cáo hoặc debug, nhưng project không phụ thuộc vào việc xuất file này trong luồng vận hành chính.
5. **Phát hiện đỉnh và thông kê chính**: Backend tính vehicles_per_min bằng diff trên total_vehicles, rolling mean/std để phát hiện peaks, và tỷ lệ từng loại để phục vụ frontend.
6. **Những cột/định dạng mong đợi**: timestamp (lưu UTC), count_car, count_motor, count_bus, count_truck, total_vehicles. Nếu DB dùng tên khác, cần điều chỉnh ánh xạ trong code (AnalyzeOnRoadBase hoặc loader).

2.2.6 Xây dựng ứng dụng web

Hệ thống được phát triển theo mô hình Client-Server hiện đại, tách biệt hoàn toàn giữa tầng xử lý logic (FastAPI) và tầng giao diện người dùng (Next.js), đảm bảo khả năng mở rộng độc lập và hiệu năng cao.



Hình 2.2: Sơ đồ kiến trúc tổng quan của hệ thống

1. Backend API (FastAPI)

Backend đóng vai trò là trung tâm dữ liệu, cung cấp hai phương thức giao tiếp chính:

- **WebSocket Streaming (Thời gian thực):**
 - **/ws/frames/{id}**: Truyền tải luồng video MJPEG đã qua xử lý (vẽ bounding box) với độ trễ cực thấp (<100ms).
 - **/ws/info/{id}**: Đẩy dữ liệu đếm xe tức thời từ bộ nhớ đệm (RAM) xuống client mỗi 500ms, giúp giao diện cập nhật mượt mà không cần tải lại trang.
- **RESTful Analytics API (Phân tích dữ liệu):** Sử dụng thư viện Pandas để thực hiện các truy vấn phức tạp trực tiếp trên cơ sở dữ liệu PostgreSQL, thay thế hoàn toàn phương pháp đọc file log truyền thống:
 - **GET /api/v1/charts/time-series**: Trích xuất chuỗi thời gian lưu lượng xe, hỗ trợ các thuật toán Resampling (gom nhóm dữ liệu theo phút/giờ).
 - **GET /api/v1/analyze/{id}**: Cung cấp các chỉ số phân tích nâng cao như: *Phát hiện đỉnh lưu lượng (Peak Detection)*, *Dộ biến động (Volatility)*, và *Xu hướng tăng/giảm (Trend Analysis)*.

- GET /api/v1/charts/vehicle-distribution: Tổng hợp tỷ lệ phần trăm các loại phương tiện trong ngày.

2. Frontend Application (Next.js 14)

Giao diện người dùng được xây dựng trên Next.js 14 với kiến trúc App Router, tận dụng khả năng Server-Side Rendering (SSR) để tối ưu hóa SEO và tốc độ tải trang ban đầu.

- **Kiến trúc Component thông minh:** Các thành phần giao diện (UI) được thiết kế theo hướng dữ liệu (Data-driven):
 - `VideoPlayer.tsx`: Tự động xử lý kết nối và tái kết nối (Auto-reconnect) WebSocket khi mạng không ổn định.
 - `RealtimeStats.tsx`: Hiển thị các chỉ số nhảy số thời gian thực, đồng bộ hoàn toàn với video.
 - `AnalyticsStats.tsx & Charts/*`: Sử dụng thư viện Recharts để vẽ các biểu đồ trực quan (Line Chart, Donut Chart) dựa trên dữ liệu lịch sử từ API.
- **Quản lý trạng thái:** Sử dụng React Hooks (`useEffect`, `useState`) để quản lý vòng đời của các kết nối thời gian thực, đảm bảo không rò rỉ bộ nhớ (memory leak) khi người dùng chuyển trang.

3. Triển khai và cấu hình (Local Deployment)

Hệ thống được thiết kế để triển khai trực tiếp trên môi trường máy tính cục bộ (Local Host), tối ưu hóa cho quá trình phát triển và kiểm thử nhanh chóng mà không phụ thuộc vào các nền tảng ảo hóa phức tạp. Quy trình triển khai tuân thủ các chuẩn mực về quản lý môi trường và bảo mật:

- **Quản lý môi trường Backend:** Sử dụng Python Virtual Environment (venv) để cài đặt các thư viện phụ thuộc, đảm bảo tính nhất quán của các gói thư viện (như PyTorch, Ultralytics, SQLAlchemy) và tránh xung đột với hệ thống gốc.
- **Quản lý môi trường Frontend:** Frontend được khởi chạy trên nền tảng Node.js Runtime, sử dụng npm để quản lý các gói thư viện giao diện và build source code tối ưu.
- **Cơ sở dữ liệu cục bộ:** PostgreSQL được cài đặt và vận hành trực tiếp trên máy chủ (Native Service), kết nối với ứng dụng thông qua driver `psycopg2` và `asyncpg`.
- **Quản lý Cấu hình:** Toàn bộ các tham số nhạy cảm (Database Credentials, API Keys của Gemini, Cổng kết nối) được tách biệt hoàn toàn khỏi mã nguồn và quản lý thông qua tệp biến môi trường `.env`, giúp dễ dàng tùy chỉnh cấu hình mà không cần can thiệp vào logic code.

4. Tính năng mở rộng: trợ lý Luật giao thông (AI Chatbot)

Nhằm giải quyết vấn đề thiếu hụt kiến thức pháp lý của người tham gia giao thông, hệ thống tích hợp một module Chatbot thông minh, hoạt động như một chuyên gia tư vấn luật tự động 24/7. Module này được xây dựng dựa trên kỹ thuật **RAG (Retrieval-Augmented Generation)**, cho phép mô hình AI truy cập và "hiểu" các văn bản pháp luật Việt Nam mới nhất mà không cần huấn luyện lại (fine-tuning).

Cơ chế hoạt động chi tiết của Chatbot bao gồm ba giai đoạn chính:

- **Giai đoạn 1: Số hóa và Vector hóa dữ liệu (Indexing):** Các văn bản Luật Giao thông đường bộ và Nghị định xử phạt được tiền xử lý và chia nhỏ thành các đoạn văn bản (Chunks) có ngữ nghĩa. Sau đó, mô hình nhúng ngôn ngữ (Embedding Model) chuyên biệt cho tiếng Việt (`keepitreal/vietnamese-sbert`) sẽ chuyển đổi các đoạn văn này thành các vector đa chiều và lưu trữ vào Cơ sở dữ liệu Vector ChromaDB. Quá trình này giúp hệ thống có khả năng tìm kiếm dựa trên ý nghĩa (Semantic Search) thay vì chỉ tìm kiếm từ khóa đơn thuần.
- **Giai đoạn 2: Truy xuất ngữ cảnh (Retrieval):** Khi người dùng đặt câu hỏi (ví dụ: "Vượt đèn đỏ phạt bao nhiêu?"), hệ thống sẽ chuyển câu hỏi thành vector và thực hiện thuật toán tìm kiếm tương đồng (Cosine Similarity) trong ChromaDB để trích xuất ra K đoạn văn bản luật có liên quan nhất.
- **Giai đoạn 3: Sinh câu trả lời (Generation):** Hệ thống xây dựng một Prompt (câu lệnh) chứa câu hỏi của người dùng kèm theo các đoạn luật vừa tìm được, sau đó gửi đến Mô hình Ngôn ngữ Lớn Google Gemini. AI sẽ tổng hợp thông tin và sinh ra câu trả lời chính xác, mạch lạc, đồng thời trích dẫn rõ ràng nguồn gốc pháp lý (Điều khoản, Chương, Mục), giúp loại bỏ hoàn toàn hiện tượng "ảo giác" (AI tự bịa thông tin) thường gặp.

2.2.7 Quy trình triển khai tổng thể

Quá trình triển khai hệ thống từ đầu đến cuối được thực hiện theo một quy trình có hệ thống, được chia thành sáu giai đoạn chính, mỗi giai đoạn có các mục tiêu và đầu ra cụ thể, đảm bảo hệ thống được triển khai một cách chính xác và hiệu quả.

Giai đoạn 1: Chuẩn bị môi trường và dữ liệu

Giai đoạn đầu tiên tập trung vào việc thiết lập môi trường phát triển và chuẩn bị dữ liệu huấn luyện. Môi trường phát triển được cài đặt với Python phiên bản 3.10 trở lên và Node.js phiên bản 18 trở lên, cùng với tất cả các dependencies được liệt kê trong `requirements.txt` và `package.json`.

Tiếp theo, tập dữ liệu ảnh với nhãn theo định dạng YOLO được chuẩn bị kỹ lưỡng, đảm bảo có đủ mẫu đại diện cho 4 lớp phương tiện và chất lượng nhãn đạt yêu cầu. Script tiền xử lý được thực thi để kiểm tra tính toàn vẹn của dữ liệu (đảm bảo mỗi ảnh có nhãn tương ứng), chuẩn hóa tọa độ bounding boxes và tạo file cấu hình `data.yaml` với định nghĩa đầy đủ về đường dẫn và tên lớp. Cuối cùng, tập dữ liệu được chia ngẫu nhiên theo tỷ lệ 80:10:10 thành ba tập con: training, validation và testing, đảm bảo tính đại diện và khả năng đánh giá khách quan.

Giai đoạn 2: Huấn luyện mô hình

Giai đoạn huấn luyện bắt đầu với việc cấu hình các siêu tham số thông qua chỉnh sửa file `training_config.yaml`, bao gồm số lượng epochs, batch size, learning rate và các tham số khác phù hợp với đặc thù của tập dữ liệu và phần cứng có sẵn. Mô hình YOLOv8m pre-trained được tải từ kho lưu trữ chính thức của Ultralytics, cung cấp điểm khởi đầu tốt cho quá trình fine-tuning.

Quá trình training được thực thi thông qua hàm `model_trainer.train_model()` và được theo dõi liên tục qua TensorBoard (một công cụ trực quan hóa) hoặc console logs để phát hiện sớm các vấn đề như overfitting hoặc underfitting. Sau khi training hoàn tất, validation được chạy trên tập test độc lập để đánh giá toàn diện các chỉ số như mAP, Precision, Recall và các chỉ số khác. Trọng số tốt nhất (`best.pt`) được lưu vào thư mục `models/` để sử dụng cho inference trong môi trường production.

Giai đoạn 3: Thiết lập Backend và cơ sở dữ liệu

Giai đoạn này tập trung vào việc cấu hình môi trường chạy và khởi tạo kho dữ liệu.

- Cấu hình môi trường:** Tệp `.env` được thiết lập với các tham số quan trọng như `DATABASE_URL` (kết nối PostgreSQL), `GEMINI_API_KEY` và cấu hình cổng mạng.
- Khởi tạo Database:** Sử dụng SQLAlchemy để ánh xạ các lớp mô hình thành bảng trong PostgreSQL. Lược đồ cơ sở dữ liệu (Database Schema) được thiết kế tối ưu cho hai tác vụ chính: lưu trữ chuỗi thời gian (Time-series) và quản lý hội thoại thông minh. Cụ thể:

a. *Bảng traffic_logs (Lưu trữ dữ liệu giao thông):*

Được thiết kế để ghi nhận dữ liệu với tần suất cao (High-frequency write), phục vụ cho việc vẽ biểu đồ và phân tích xu hướng.

Bảng 2.1: Mô tả cấu trúc bảng cơ sở dữ liệu giám sát

Trường (Field)	Kiểu dữ liệu	Mô tả
<code>id</code>	Integer	Khóa chính tự tăng (Primary Key).
<code>camera_id</code>	Integer	Định danh camera giám sát (được đánh Index để truy vấn nhanh).
<code>timestamp</code>	DateTime	Thời điểm ghi nhận dữ liệu.
<code>count_*</code>	Integer	Các cột riêng biệt (<code>car</code> , <code>motor</code> , <code>bus</code> , <code>truck</code>) lưu số lượng từng loại xe.
<code>total_vehicles</code>	Integer	Tổng lưu lượng phương tiện tại thời điểm đó.
<code>fps</code>	Float	Chỉ số hiệu năng xử lý của hệ thống tại thời điểm ghi.

b. *Bảng chat_messages (Lưu trữ lịch sử Chatbot):*

Được thiết kế để hỗ trợ kiến trúc RAG, cho phép AI ghi nhớ ngữ cảnh và trích dẫn nguồn.

- Cấu hình Camera:** Tệp `core/config.py` được cập nhật với danh sách đường dẫn video (RTSP/YouTube) và tọa độ vùng quan tâm (ROI) tương ứng cho từng camera, đảm bảo tính chính xác của thuật toán đếm.

Bảng 2.2: Mô tả cấu trúc bảng lưu trữ lịch sử hội thoại

Trường (Field)	Kiểu dữ liệu	Mô tả
session_id	String	Định danh phiên làm việc, dùng để gom nhóm hội thoại và truy xuất ngữ cảnh (Context Window).
role	String	Vai trò người gửi: user (người dùng) hoặc assistant (AI).
content	Text	Nội dung tin nhắn (Dùng kiểu Text để lưu câu trả lời dài).
sources	JSON	Lưu danh sách các văn bản luật được trích dẫn (Metadata cho RAG).
created_at	DateTime	Thời gian tạo tin nhắn.

Giai đoạn 4: Khởi chạy và Đồng bộ Hệ thống (System Startup)

Khi Backend được khởi động (qua Uvicorn), quy trình khởi tạo diễn ra theo trình tự nghiêm ngặt để đảm bảo tính toàn vẹn của đa tiến trình:

- Khởi tạo Bộ nhớ chia sẻ (Shared Memory Init):** Hệ thống khởi tạo Multiprocessing Manager và các cấu trúc dữ liệu dùng chung (`info_dict`, `frame_dict`) trên RAM. Cơ chế kiểm tra (Guard Clause) được áp dụng để ngăn chặn việc khởi tạo trùng lặp tiến trình (Double Start).
- Phân luồng Xử lý (Process Spawning):** Dựa trên số lượng camera cấu hình, hệ thống sinh ra các tiến trình con (Child Processes) độc lập. Mỗi tiến trình nạp riêng một instance của mô hình YOLov8 vào GPU/CPU để đảm bảo không tranh chấp tài nguyên.
- Kích hoạt Đồng bộ hóa (Worker Activation):** Tác vụ nền `save_stats_to_db_worker` được kích hoạt song song. Tác vụ này hoạt động như một cầu nối, định kỳ mỗi 10 giây sẽ "chụp" (snapshot) dữ liệu từ bộ nhớ RAM và ghi bền vững xuống bảng `traffic_logs` trong PostgreSQL.

Giai đoạn 5: Tích hợp và Triển khai Frontend

Giao diện người dùng (Next.js) được khởi chạy trên môi trường Node.js cục bộ, kết nối với Backend theo mô hình luồng dữ liệu kép:

- Luồng Thời gian thực (Hot Stream):** Các component `VideoPlayer` và `RealtimeStats` thiết lập kết nối WebSocket tới Backend. Dữ liệu được lấy trực tiếp từ RAM (Shared Memory) giúp độ trễ hiển thị giảm xuống dưới 100ms, đảm bảo hình ảnh và số đếm khớp nhau từng khung hình.
- Luồng Phân tích (Cold Stream):** Các biểu đồ `VehicleLineChart` và `VehicleDistributionChart` gọi RESTful API (`/analytics/trend`). Backend sử dụng Pandas để truy vấn và tổng hợp dữ liệu lịch sử từ PostgreSQL, giúp giảm tải cho Frontend.
- Cơ chế Tự phục hồi (Auto-reconnect):** Frontend được tích hợp logic tự động kết nối lại sau 3 giây nếu WebSocket bị ngắt, đảm bảo Dashboard luôn hoạt động ổn định mà không cần tải lại trang (F5).

Giai đoạn 6: Vận hành và Bảo trì (Operations & Maintenance)

Trong quá trình vận hành thực tế, hệ thống tập trung vào tính ổn định và độ chính xác của dữ liệu:

- Tối ưu hóa I/O:** Nhờ kiến trúc lưu trữ lai (Hybrid Storage), ổ cứng không bị quá tải bởi việc ghi log liên tục. Hệ thống chỉ thực hiện thao tác ghi đĩa (Disk Write) theo chu kỳ của Worker, giúp tăng tuổi thọ phần cứng và duy trì FPS ổn định cho AI.
- Bảo trì Cơ sở dữ liệu:** Dữ liệu giao thông được lưu trữ có cấu trúc trong PostgreSQL, cho phép dễ dàng thực hiện sao lưu (Backup) hoặc trích xuất báo cáo (Export) bằng các công cụ quản trị tiêu chuẩn (như pgAdmin) mà không cần dừng hệ thống.
- Tinh chỉnh tham số:** Dựa trên quan sát thực tế tại Dashboard, các tham số như vùng quan tâm (ROI) hay ngưỡng nhận diện (Confidence Threshold) có thể được tinh chỉnh nóng trong file cấu hình để thích nghi với các góc quay hoặc điều kiện ánh sáng mới.

Chương 3

Kết quả & Phân tích

Chương này trình bày các kết quả thực nghiệm đạt được từ quá trình huấn luyện và đánh giá mô hình YOLOv8m trên tập dữ liệu phương tiện giao thông Việt Nam, cùng với các kết quả triển khai hệ thống đếm xe và ứng dụng web. Các chỉ số đánh giá được phân tích chi tiết để đánh giá hiệu quả và khả năng ứng dụng thực tế của hệ thống.

3.1 Kết quả huấn luyện mô hình

Quá trình huấn luyện mô hình YOLOv8m được thực hiện trong 87 epochs với các siêu tham số đã được cấu hình: batch size 16, learning rate ban đầu 0.01 với cosine annealing, và kích thước ảnh đầu vào 640x640 pixels. Mô hình được fine-tuning từ weights pre-trained trên tập dữ liệu COCO, áp dụng phương pháp Fine-tuning để tận dụng các đặc trưng đã được học từ dữ liệu lớn.

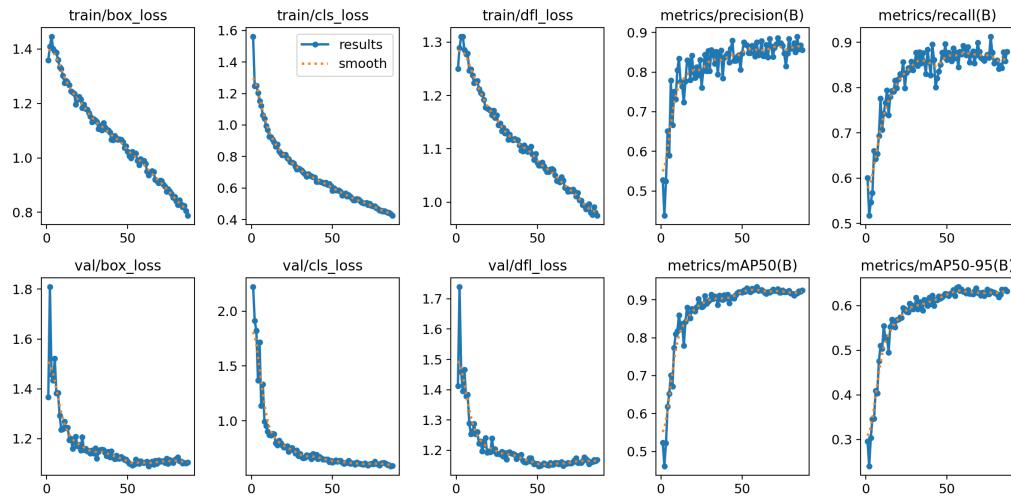
Bảng 3.1: Kết quả đánh giá mô hình trên tập Test

Chỉ số đánh giá	Giá trị
Precision	0.8562 (85.62%)
Recall	0.8795 (87.95%)
F1-Score	0.8677 (86.77%)
mAP@0.5	0.9249 (92.49%)
mAP@0.5:0.95	0.6321 (63.21%)

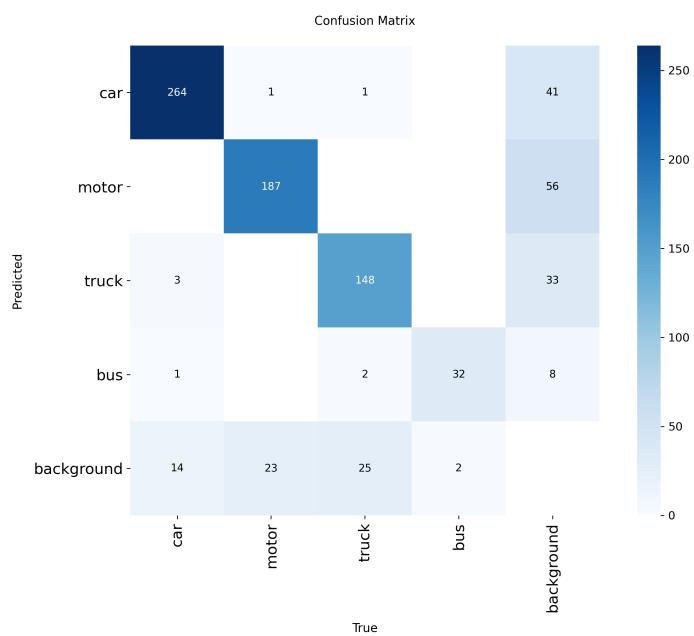
Kết quả đánh giá trên tập test độc lập cho thấy mô hình đạt được hiệu suất tốt với các chỉ số chính như sau: Precision đạt 85.62%, cho thấy trong số các phương tiện được phát hiện, hơn 85% là chính xác. Recall đạt 87.95%, nghĩa là mô hình có khả năng phát hiện được gần 88% tổng số phương tiện thực tế có trong ảnh. F1-Score đạt 86.77%, một chỉ số tổng hợp phản ánh sự cân bằng tốt giữa Precision và Recall, cho thấy mô hình vừa có độ chính xác cao vừa có khả năng phát hiện tốt. Chỉ số mAP@0.5 đạt 92.49%, một kết quả xuất sắc cho thấy độ chính xác trung bình của mô hình tại ngưỡng IoU 0.5 là rất cao. Chỉ số mAP@0.5:0.95 đạt 63.21%, phản ánh khả năng của mô hình trong việc dự đoán các bounding box chính xác ở nhiều ngưỡng IoU khác nhau, từ 0.5 đến 0.95.

Hình 3.1 thể hiện quá trình hội tụ của mô hình qua các epochs. Có thể quan sát thấy các đường cong loss (box loss, classification loss, DFL loss) giảm dần và ổn định sau khoảng 30-40 epochs, cho thấy mô hình đã học được các đặc trưng quan trọng. Các chỉ số Precision, Recall và mAP tăng dần và đạt giá trị cao nhất vào các epochs cuối, chứng tỏ quá trình fine-tuning đã thành công.

Ma trận nhầm lẫn (Hình 3.2) cho thấy mô hình phân loại tốt cả 4 lớp phương tiện. Tỷ lệ nhầm lẫn giữa các lớp là thấp, với hầu hết các dự đoán nằm trên đường chéo chính. Một số nhầm lẫn nhỏ có thể xảy ra giữa các lớp có hình dạng tương tự, chẳng hạn như giữa Car và Truck trong một số trường hợp góc nhìn hoặc điều kiện ánh sáng đặc biệt.



Hình 3.1: Biểu đồ tiến trình huấn luyện mô hình qua các epochs



Hình 3.2: Ma trận nhầm lẫn trên tập Test

3.2 Phân tích hiệu năng theo từng lớp

Để đánh giá chi tiết hơn, hệ thống tính toán các chỉ số đánh giá riêng cho từng lớp phương tiện trên tập test. Bảng 3.2 trình bày kết quả đầy đủ các chỉ số đánh giá cho từng lớp, bao gồm Precision, Recall, F1-Score, mAP@0.5 và mAP@0.5:0.95.

Bảng 3.2: Kết quả đánh giá mô hình theo từng lớp phương tiện trên tập Test

Lớp phương tiện	Precision	Recall	F1-Score	mAP@0.5	mAP@0.5:0.95
Bus (Xe buýt)	0.9091 (90.91%)	0.8824 (88.24%)	0.8955 (89.55%)	0.9331 (93.31%)	0.7250 (72.50%)
Car (Ô tô)	0.9084 (90.84%)	0.8788 (87.88%)	0.8934 (89.34%)	0.9206 (92.06%)	0.7560 (75.60%)
Truck (Xe tải)	0.8331 (83.31%)	0.8693 (86.93%)	0.8508 (85.08%)	0.8826 (88.26%)	0.6631 (66.31%)
Motor (Xe máy)	0.8335 (83.35%)	0.8064 (80.64%)	0.8197 (81.97%)	0.8548 (85.48%)	0.5108 (51.08%)

Kết quả cho thấy mô hình đạt hiệu suất tốt trên cả 4 lớp phương tiện, với các chỉ số đánh giá đều ở mức cao. Lớp Bus đạt kết quả tốt nhất với mAP@0.5 = 93.31%, Precision = 90.91% và F1-Score = 89.55% (cao nhất trong 4 lớp), có thể do đặc trưng hình dạng lớn và dễ phân biệt của xe buýt so với các phương tiện khác. Lớp Car cũng đạt kết quả xuất sắc với mAP@0.5 = 92.06%, Precision = 90.84%, F1-Score = 89.34% và mAP@0.5:0.95 = 75.60% (cao nhất trong 4 lớp), cho thấy mô hình nhận diện tốt loại phương tiện phổ biến này với độ chính xác cao về vị trí bounding box.

Lớp Truck đạt mAP@0.5 = 88.26%, Recall = 86.93% (cao nhất trong 4 lớp) và F1-Score = 85.08%, thể hiện khả năng phát hiện tốt các xe tải, mặc dù Precision = 83.31% thấp hơn một chút so với Bus và Car, cho thấy có một số false positives. Lớp Motor đạt mAP@0.5 = 85.48%, Precision = 83.35%, F1-Score = 81.97% (thấp nhất trong 4 lớp), và mAP@0.5:0.95 = 51.08% cũng thấp hơn đáng kể so với các lớp khác. Điều này có thể do kích thước nhỏ của xe máy, dễ bị che khuất bởi các phương tiện lớn hơn, và đặc trưng hình dạng đôi khi tương tự với một số đối tượng khác trong môi trường giao thông, dẫn đến việc mô hình gặp khó khăn trong việc dự đoán chính xác vị trí bounding box (thể hiện qua mAP@0.5:0.95 thấp).

Nhìn chung, mô hình thể hiện khả năng phân loại tốt với Precision, Recall và F1-Score đều trên 80% cho tất cả các lớp, và mAP@0.5 đều trên 85%, đáp ứng yêu cầu triển khai trong môi trường thực tế.

3.3 Hiệu năng xử lý realtime

Để đánh giá khả năng triển khai thực tế, hệ thống được kiểm tra hiệu năng xử lý trên video thời gian thực. Với các tối ưu hóa đã được áp dụng (resize frame xuống 480x270 pixels, frame skipping với skip_frames=3, và confidence threshold 0.4), hệ thống đạt được tốc độ xử lý trung bình khoảng 10-20 FPS trên phần cứng phổ biến (GPU NVIDIA hoặc CPU đa lõi). Tốc độ này đảm bảo hệ thống có thể xử lý video realtime một cách mượt mà, đáp ứng yêu cầu của ứng dụng giám sát giao thông.

Khi xử lý nhiều camera đồng thời, hệ thống sử dụng multiprocessing để phân tán tải xử lý lên các CPU cores khác nhau. Với 2 camera, mỗi camera vẫn duy trì được tốc độ xử lý khoảng 10-20 FPS, đảm bảo độ trễ thấp và cập nhật dữ liệu realtime kịp thời. Việc sử dụng Manager.dict() để chia sẻ dữ liệu giữa các process đảm bảo các API endpoint có thể truy xuất thông tin đếm xe ngay lập tức mà không bị block.

3.4 Kết quả hệ thống đếm phương tiện

Hệ thống đếm xe được đánh giá trên các video thực tế từ camera giao thông. Kết quả cho thấy thuật toán đếm với ROI và tracking đạt độ chính xác cao, với tỷ lệ đếm đúng cao khi so sánh với đếm thủ công. Hệ thống thành công trong việc tránh đếm trùng lặp nhờ cơ chế tracking ID, đảm bảo mỗi phương tiện chỉ được đếm một lần khi đi qua vùng quan tâm (như được minh họa trong Hình 3.3).

Một số thách thức được ghi nhận trong quá trình đếm bao gồm:

- Phương tiện di chuyển quá nhanh có thể bị mất track trong một số frame, dẫn đến việc gán ID mới và đếm trùng.
- Phương tiện bị che khuất một phần bởi các vật thể khác có thể gây khó khăn cho việc tracking liên tục.
- Điều kiện ánh sáng thay đổi (ngày/đêm, mưa) có thể ảnh hưởng đến độ chính xác phát hiện.
- Góc camera và độ sáng khác với dữ liệu training gây ra thiếu sót, đặc biệt là khi camera được đặt ở góc nhìn khác hoặc trong điều kiện ánh sáng khác biệt so với các mẫu trong tập huấn luyện, dẫn đến một số phương tiện không được phát hiện.



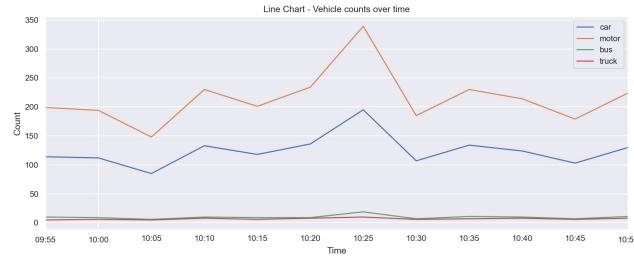
Hình 3.3: Giao diện hệ thống đếm phương tiện thực tế với vùng quan tâm (ROI) và các ID theo dõi.

Tuy nhiên, với các tối ưu hóa đã được áp dụng, hệ thống vẫn duy trì được độ chính xác cao trong hầu hết các điều kiện thực tế.

3.5 Kết quả phân tích dữ liệu

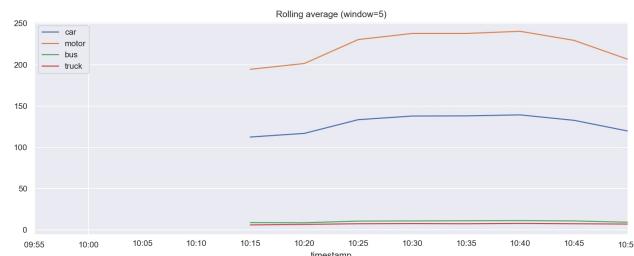
3.5.1 Phân tích xu hướng và dự đoán lưu lượng tổng thể

Lưu lượng giao thông tổng thể được phân tích trong khung thời gian 60 phút cho thấy một xu hướng tăng trưởng liên tục và đáng kể. Biểu đồ đường lũy kế hiện rõ sự gia tăng ổn định của tổng số phương tiện được đếm. Sự biến thiên mạnh mẽ về lưu lượng giữa các phút đã được xử lý để mô phỏng tính nhấp nhô của luồng giao thông thực tế.



Hình 3.4: Biểu đồ đường tổng hợp lưu lượng lũy kế theo thời gian

Để làm rõ xu hướng chính và loại bỏ các nhiễu động nhất thời, phân tích trung bình động đã được áp dụng. Dựa trên cấu hình trong `analyze.py` (tổng hợp dữ liệu 5 phút và sử dụng cửa sổ 5 điểm), biểu đồ trung bình động thực chất cho thấy một cách mượt mà hơn về tốc độ tăng trưởng thực tế trong khoảng thời gian 25 phút (5 điểm × 5 phút/điểm). Điều này giúp hệ thống thu được xu hướng bền vững, loại bỏ nhiễu và hỗ trợ xác định các giai đoạn cao điểm một cách đáng tin cậy.



Hình 3.5: Biểu đồ trung bình động của tổng lưu lượng

3.5.2 Phát hiện và đánh giá đỉnh Lưu lượng

Chức năng phát hiện đỉnh lưu lượng là một yếu tố then chốt trong phân tích thời gian thực. Dựa trên thuật toán được xây dựng bằng công thức $\text{rolling_mean} + 3 \times \text{rolling_std}$, áp dụng nguyên lý 3-sigma trong thống kê, cho phép phát hiện các sự kiện bất thường một cách tự động. Hệ thống đã xác định thành công các khoảng thời gian mà tốc độ lưu lượng tăng đột biến.i gian mà tốc độ lưu lượng tăng đột biến.

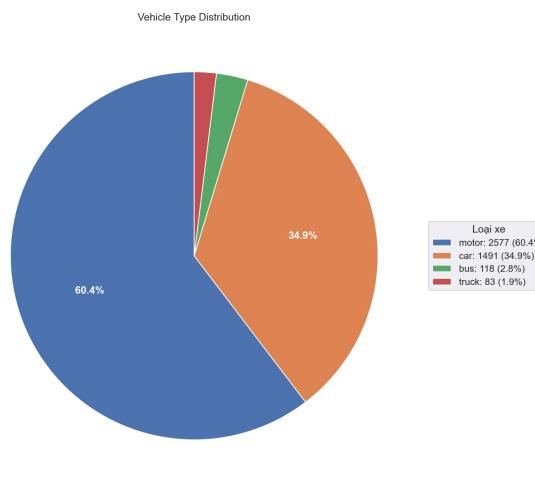


Hình 3.6: Biểu đồ phát hiện các khoảng thời gian đạt đỉnh lưu lượng

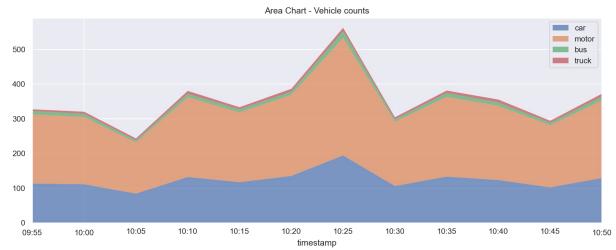
Các điểm đỉnh được đánh dấu để đại diện cho những thời khắc lưu lượng vượt quá khả năng thông hành an toàn của tuyến đường. Thông tin này là cơ sở trực tiếp để hệ thống điều khiển giao thông có thể kích hoạt các chiến lược can thiệp như điều chỉnh chu kỳ đèn hoặc gửi cảnh báo tắc nghẽn.

3.5.3 Phân tích cơ cấu thành phần và tỷ trọng

Cơ cấu thành phần lưu lượng thể hiện rõ nét đặc điểm giao thông đô thị, nơi xe mô tô chiếm tỷ lệ cao nhất trong tổng số phương tiện, với tỷ trọng vượt trội so với các loại xe khác. Xe ô tô con chiếm tỷ lệ lớn thứ hai. Hai loại phương tiện này là nhân tố chính chi phối tổng lưu lượng giao thông.



Hình 3.7: Tỷ lệ phần trăm các loại xe.



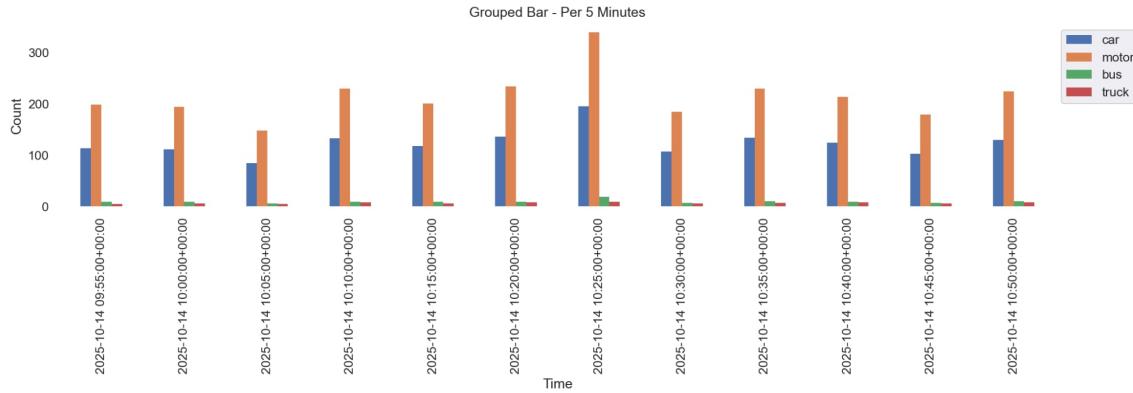
Hình 3.8: Phân bố tích lũy lưu lượng theo loại xe.

Biểu đồ vùng cung cấp một cái nhìn động về tỷ trọng của từng loại xe theo thời gian. Đường biểu diễn cho motor và car luôn duy trì khoảng cách lớn so với bus và truck, khẳng định sự ổn định trong cơ cấu lưu lượng qua các phút giám sát.

3.5.4 So sánh chi tiết lưu lượng theo loại xe

Biểu đồ cột nhóm mang lại sự phân tích chi tiết nhất, hiển thị số lượng tuyệt đối của từng loại xe trong mỗi khoảng thời gian 5 phút được tổng hợp.

Việc phân tích theo từng cột nhóm xác nhận lại sự dao động mạnh của lưu lượng, đặc biệt là sự tăng giảm luân phiên của cột motor và car giữa các mốc thời gian cách nhau 5 phút. Điều này minh họa cho sự thay đổi liên tục của tốc độ dòng chảy giao thông, đòi hỏi hệ thống phải có khả năng phản ứng linh hoạt để duy trì hiệu quả thông suốt.



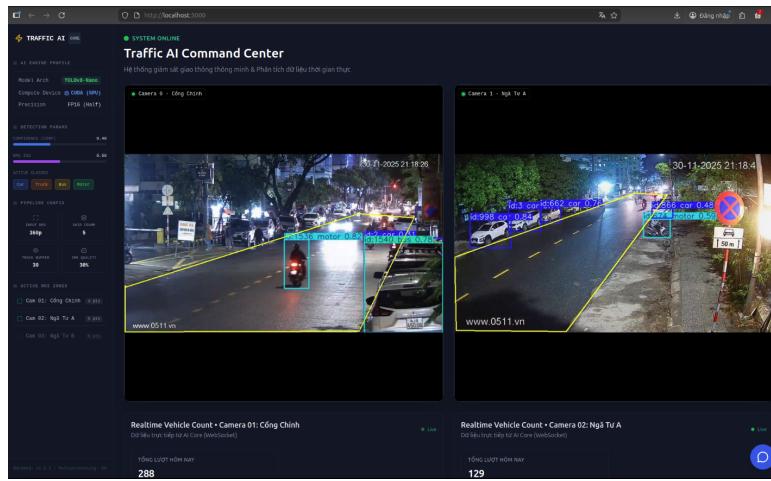
Hình 3.9: So sánh số lượng từng loại xe theo từng mốc thời gian.

3.6 Kết quả Web Application

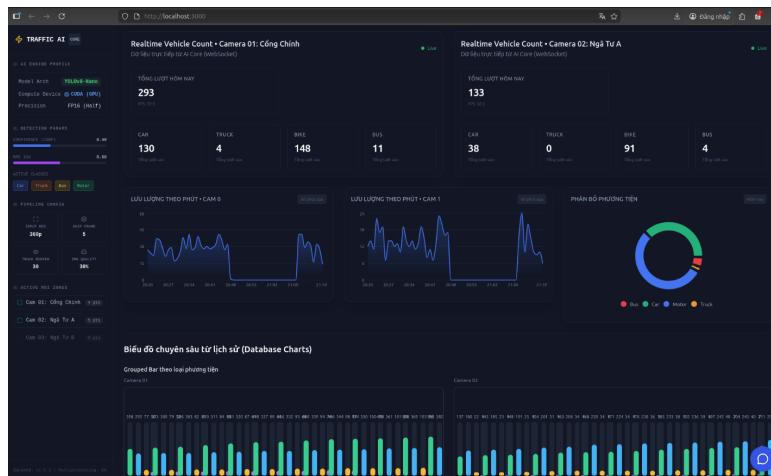
Ứng dụng web được triển khai và kiểm thử trên môi trường production, cho thấy hiệu năng tốt và trải nghiệm người dùng mượt mà. Backend FastAPI xử lý các request REST API với thời gian phản hồi trung bình dưới 100ms, đảm bảo giao diện người dùng được cập nhật nhanh chóng. WebSocket endpoints cung cấp dữ liệu realtime với độ trễ thấp, cho phép hiển thị video stream và thống kê đếm xe cập nhật liên tục.

Frontend Next.js với các tối ưu hóa như code splitting và tree shaking đảm bảo thời gian tải trang ban đầu nhanh và trải nghiệm người dùng mượt mà. Các component biểu đồ (VehicleLineChart, RealtimeCounterChart) hiển thị dữ liệu một cách trực quan và dễ hiểu, hỗ trợ người dùng phân tích xu hướng giao thông một cách hiệu quả.

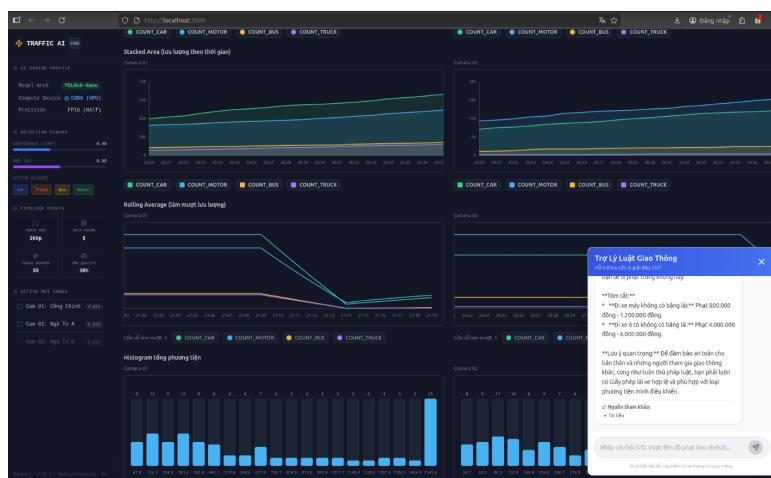
Tính năng chatbot bổ sung với kiến trúc RAG đã được kiểm thử và hoạt động ổn định, cung cấp khả năng tư vấn về luật giao thông dựa trên tài liệu tham khảo. Một số hình ảnh thực tế về giao diện của ứng dụng được minh họa trong Hình 3.10.



(a) Giao diện web



(b) Giao diện web



(c) Giao diện web

Hình 3.10: Giao diện thực tế của ứng dụng web trên môi trường production.

3.7 Thảo luận

Kết quả thực nghiệm cho thấy hệ thống đạt được các mục tiêu đề ra với hiệu suất tốt trên nhiều khía cạnh. Mô hình YOLOv8m sau khi fine-tuning đạt độ chính xác cao (mAP@0.5 = 92.49%), đủ để triển khai trong môi trường thực tế. Việc sử dụng Fine-tuning đã chứng minh hiệu quả trong việc tận dụng kiến thức từ dữ liệu lớn (COCO) để cải thiện hiệu suất trên tập dữ liệu nhỏ hơn (1547 ảnh).

Hệ thống đếm xe với tracking và ROI đạt độ chính xác cao, đáp ứng yêu cầu của ứng dụng giám sát giao thông. Các tối ưu hóa hiệu năng (frame skipping, resize, multiprocessing) đảm bảo hệ thống có thể xử lý realtime trên nhiều camera đồng thời mà vẫn duy trì tốc độ xử lý cao.

Tuy nhiên, vẫn còn một số hạn chế cần được cải thiện trong tương lai: Mô hình có thể gặp khó khăn với các phương tiện bị che khuất một phần hoặc trong điều kiện ánh sáng kém; Thuật toán tracking đôi khi mất track với các phương tiện di chuyển quá nhanh; Hệ thống cần được kiểm thử trên nhiều môi trường và điều kiện thời tiết khác nhau để đánh giá khả năng tổng quát hóa.

Nhìn chung, hệ thống đã chứng minh được khả năng ứng dụng thực tế với hiệu suất tốt và kiến trúc mở rộng được, tạo nền tảng vững chắc cho việc phát triển và cải thiện trong tương lai.

Chương 4

Kết luận

4.1 Kết luận

Nghiên cứu đã thành công trong việc xây dựng một hệ thống đếm phương tiện giao thông tự động sử dụng mô hình YOLOv8m được fine-tuning trên tập dữ liệu phương tiện giao thông Việt Nam. Hệ thống đạt được các kết quả đáng khích lệ với độ chính xác mAP@0.5 đạt 92.49%, Precision 85.62% và Recall 87.95%, chứng minh khả năng ứng dụng thực tế của mô hình trong môi trường giao thông Việt Nam.

Hệ thống đã được triển khai thành công với kiến trúc đa tầng bao gồm:

- Mô hình nhận diện sử dụng YOLOv8m với fine-tuning toàn bộ mô hình từ weights pre-trained COCO
- Hệ thống đếm xe realtime với tracking và ROI đạt độ chính xác cao
- Pipeline phân tích dữ liệu với các kỹ thuật tối ưu hóa I/O và phát hiện điểm nóng
- Ứng dụng web với backend FastAPI và frontend Next.js cung cấp giao diện trực quan và tương tác

Hệ thống có khả năng xử lý realtime trên nhiều camera đồng thời với tốc độ 10-20 FPS, đáp ứng yêu cầu của ứng dụng giám sát giao thông thực tế.

Việc áp dụng phương pháp Fine-tuning đã chứng minh hiệu quả trong việc tận dụng kiến thức từ dữ liệu lớn (COCO) để cải thiện hiệu suất trên tập dữ liệu nhỏ hơn (1547 ảnh), cho thấy giá trị của Fine-tuning trong các bài toán thực tế với dữ liệu hạn chế. Các kỹ thuật tối ưu hóa như multiprocessing, frame skipping và binary-safe tail reading đã góp phần quan trọng vào việc đảm bảo hiệu năng và khả năng mở rộng của hệ thống.

4.2 Hạn chế

Mặc dù đạt được các kết quả tích cực, hệ thống vẫn còn một số hạn chế cần được cải thiện:

- **Sự khác biệt giữa dữ liệu training và dữ liệu thực tế:** Dữ liệu huấn luyện và dữ liệu sử dụng để phát hiện trong môi trường thực tế có sự khác biệt đáng kể về góc camera, điều kiện ánh sáng, góc nhìn và bối cảnh môi trường. Sự khác biệt này dẫn đến việc mô hình gặp khó khăn trong việc phát hiện một số phương tiện, gây ra thiếu sót (false negatives) trong quá trình đếm xe. Điều này đặc biệt rõ ràng khi camera được đặt ở các góc nhìn khác hoặc trong điều kiện ánh sáng khác biệt so với các mẫu trong tập huấn luyện.
- **Hiệu suất với phương tiện di chuyển nhanh:** Thuật toán tracking đôi khi mất track với các phương tiện di chuyển quá nhanh, dẫn đến việc gán ID mới và đếm trùng lặp trong một số trường hợp.
- **Xử lý phương tiện bị che khuất:** Mô hình gặp khó khăn khi phát hiện các phương tiện bị che khuất một phần bởi các vật thể khác hoặc các phương tiện khác, ảnh hưởng đến độ chính xác đếm.
- **Điều kiện môi trường đa dạng:** Hệ thống cần được kiểm thử và cải thiện trên nhiều điều kiện môi trường khác nhau như mưa, sương mù, ban đêm để đảm bảo khả năng tổng quát hóa tốt hơn.

4.3 Hướng phát triển

Để cải thiện và mở rộng hệ thống trong tương lai, các hướng phát triển sau đây được đề xuất:

- **Cải thiện chất lượng dữ liệu training:** Thu thập và bổ sung thêm dữ liệu huấn luyện đa dạng hơn, bao gồm các góc camera khác nhau, điều kiện ánh sáng đa dạng (ngày, đêm, mưa, sương mù), và các bối cảnh môi trường khác nhau. Việc này sẽ giúp giảm thiểu sự khác biệt giữa dữ liệu training và dữ liệu thực tế, cải thiện khả năng tổng quát hóa của mô hình.
- **Áp dụng Domain Adaptation:** Nghiên cứu và áp dụng các kỹ thuật Domain Adaptation để giảm thiểu sự khác biệt giữa domain training và domain thực tế, giúp mô hình thích ứng tốt hơn với các điều kiện môi trường mới.
- **Cải thiện thuật toán tracking:** Nâng cấp thuật toán tracking bằng cách sử dụng các phương pháp tiên tiến hơn như DeepSORT hoặc các mô hình tracking dựa trên deep learning để cải thiện độ chính xác tracking với phương tiện di chuyển nhanh.
- **Tối ưu hóa cho điều kiện ánh sáng kém:** Nghiên cứu và tích hợp các kỹ thuật xử lý ảnh như histogram equalization, CLAHE (Contrast Limited Adaptive Histogram Equalization) hoặc các mô hình được huấn luyện riêng cho điều kiện ánh sáng yếu để cải thiện khả năng phát hiện trong môi trường ban đêm.
- **Mở rộng chức năng phân tích:** Phát triển thêm các tính năng phân tích nâng cao như dự đoán lưu lượng giao thông, phát hiện tai nạn, nhận diện vi phạm giao thông (vượt đèn đỏ, đi sai làn đường) để tăng giá trị ứng dụng của hệ thống.
- **Tối ưu hóa hiệu năng:** Nghiên cứu các kỹ thuật quantization, model pruning và knowledge distillation để giảm kích thước mô hình và tăng tốc độ xử lý, cho phép triển khai trên các thiết bị edge computing với tài nguyên hạn chế.
- **Tích hợp Machine Learning Operations (MLOps):** Xây dựng pipeline tự động hóa cho việc thu thập dữ liệu mới, đánh giá mô hình và cập nhật mô hình định kỳ để đảm bảo hệ thống luôn được cải thiện và thích ứng với các thay đổi trong môi trường thực tế.

Với những cải tiến này, hệ thống sẽ có khả năng ứng dụng rộng rãi hơn trong thực tế, góp phần vào việc quản lý và giám sát giao thông hiệu quả tại Việt Nam.

Tài liệu tham khảo

- [1] G. Jocher, A. Chaurasia, và J. Qiu, “Ultralytics YOLO,” phiên bản 8.0.0, 2023. [Online]. Có tại: <https://github.com/ultralytics/ultralytics>.
- [2] J. Redmon, S. Divvala, R. Girshick, và A. Farhadi, “You only look once: Unified, real-time object detection,” trong *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, tr. 779–788.
- [3] P. Lewis và cộng sự, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” trong *Advances in Neural Information Processing Systems (NeurIPS)*, tập 33, 2020, tr. 9459–9474.
- [4] N. Buch, S. A. Velastin, và J. Orwell, “A review of computer vision techniques for the analysis of urban traffic,” *IEEE Transactions on Intelligent Transportation Systems*, tập 12, số 3, 2011.

Chương 5

Phụ lục

5.1 Yêu cầu Hệ thống

Để chạy hệ thống, máy tính cần đáp ứng các yêu cầu tối thiểu sau:

- Python 3.8 trở lên
- Node.js 18 trở lên
- npm hoặc yarn (package manager cho Node.js)
- GPU NVIDIA (khuyến nghị) hoặc CPU đa lõi để xử lý realtime

5.2 Cài đặt Backend

Quá trình cài đặt backend được thực hiện theo các bước sau:

1. Di chuyển vào thư mục backend:

```
cd backend
```

2. Tạo virtual environment (nếu chưa có):

```
python -m venv venv
```

3. Kích hoạt virtual environment:

- **Windows:** venv\Scripts\activate
- **Linux/Mac:** source venv/bin/activate

4. Cài đặt các dependencies:

```
pip install -r ./requirements.txt
```

5.3 Cài đặt Frontend

Quá trình cài đặt frontend được thực hiện theo các bước sau:

1. Di chuyển vào thư mục frontend:

```
cd frontend
```

2. Cài đặt dependencies:

```
npm install
```

5.4 Chạy Hệ thống

5.4.1 Chạy Backend

Mở terminal mới và thực hiện các lệnh sau:

```
# Di chuyển vào thư mục backend
cd backend

# Kích hoạt virtual environment (nếu chưa kích hoạt)
# Windows: venv\Scripts\activate
# Linux/Mac: source venv/bin/activate

# Chạy server
python -m uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

Backend sẽ chạy tại địa chỉ `http://localhost:8000` với các endpoint:

- API Documentation: `http://localhost:8000/docs`
- ReDoc: `http://localhost:8000/redoc`

5.4.2 Chạy Frontend

Mở terminal mới và thực hiện các lệnh sau:

```
# Di chuyển vào thư mục frontend
cd frontend

# Chạy development server
npm run dev
```

Frontend sẽ chạy tại địa chỉ `http://localhost:3000`. Mở trình duyệt và truy cập địa chỉ này để sử dụng ứng dụng.

5.5 Lưu ý Quan trọng

- **Backend cần chạy trước** để frontend có thể kết nối API
- **Camera/Video source:** Hệ thống sẽ tự động khởi động 2 camera processes. Nếu không có camera thật, có thể cần cấu hình video source trong file `backend/app/core/config.py` thông qua biến `settings_metric_transport.PATH_VIDEOS`
- **Database:** Hệ thống sử dụng SQLite mặc định, không cần cấu hình thêm
- **Log files:** Dữ liệu thống kê được lưu tự động trong thư mục `backend/logs/traffic_count/` với định dạng JSON theo ngày

5.6 Troubleshooting

5.6.1 Lỗi Import Module

Nếu gặp lỗi import module:

- Đảm bảo đã cài đặt đầy đủ dependencies từ file `requirements.txt`
- Kiểm tra virtual environment đã được kích hoạt đúng cách
- Thử cài đặt lại dependencies: `pip install -r requirements.txt`

5.6.2 Lỗi Kết nối API

Nếu frontend không kết nối được với backend:

- Kiểm tra backend đã chạy tại port 8000
- Kiểm tra CORS settings trong file `backend/app/main.py`
- Kiểm tra firewall không chặn kết nối giữa frontend và backend

5.6.3 Lỗi Camera

Nếu hệ thống không nhận diện được camera:

- Kiểm tra camera có được kết nối đúng cách không
- Xem log trong terminal backend để biết chi tiết lỗi
- Kiểm tra cấu hình video source trong file `backend/app/core/config.py`
- Đảm bảo đường dẫn video hoặc URL camera là hợp lệ

5.7 Cấu hình Hệ thống

Các tham số quan trọng của hệ thống có thể được cấu hình trong các file sau:

- `backend/app/core/config.py`: Cấu hình camera, ROI, model path, database
- `model_detection/configs/training_config.yaml`: Cấu hình hyperparameters cho training
- `model_detection/configs/model_config.yaml`: Cấu hình inference parameters