HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

ĐỒ ÁN
TỐT NGHIỆP ĐẠI HỌC

ĐỀ TÀI:
PHÁT TRIỂN ỨNG DỤNG WEB TÍCH HỢP DỰ BÁO GIAO THÔNG SỬ DỤNG MACHINE LEARNING

Giảng viên hướng dẫn : **TS. NGUYỄN TẤT THẮNG**

Sinh viên thực hiện : **Nguyễn Thị Thêu**

Lớp : **D20HTTT2**

Mã sinh viên : **B20DCCN665**

Hệ : **ĐẠI HỌC CHÍNH QUY**

Hà Nội, tháng 12 năm 2024

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn chân thành và sâu sắc tới thầy TS. Nguyễn Tất Thắng, người đã trực tiếp hướng dẫn tận tình, chu đáo, chia sẻ những ý kiến và kinh nghiệm quý báu trong suốt quá trình em thực tập và thực hiện đồ án tốt.

Sau đó, em xin gửi lời cảm ơn các thầy, cô trong Học Viện nói chung và khoa CNTT1 nói riêng đã luôn nhiệt huyết, tận tình trong từng bài giảng và tạo điều kiện thuận lợi nhất cho em trong thời gian học tập và nghiên cứu tại trường Học Viện Công Nghệ Bưu Chính Viễn Thông.

Con xin được gửi lời cảm ơn tới bố mẹ và những người thân yêu đã luôn lo lắng, động viên, ủng hộ và tạo điều kiện cho con được học tập tốt. Là chỗ dựa tinh thân và những người tiếp sức cho con có được thành công trong cuộc sống.

Cuối cùng, tôi xin gửi lời chúc tốt đẹp nhất đến những người bạn của tôi và các thầy cô tham gia đợt bảo vệ tốt nghiệp trong khóa này. Chúc cho mọi người luôn vui vẻ và thành công trong cuộc sống.

Em xin chân thành cảm ơn!

Hà Nội, tháng 12 năm 2024

Sinh viên thực hiện

Nguyễn Thị Thêu

NHẬN XÉT, ĐÁNH GIÁ, CHO ĐIỂM

(Của người hướng dẫn)

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Điểm:…… **(bằng chữ:** ……**)**

Đồng ý/Không đồng ý cho sinh viên bảo vệ trước hội đồng chấm đồ án tốt nghiệp?

……**, ngày**…..... **tháng 12 năm 2024**

CÁN BỘ GIẢNG VIÊN HƯỚNG DẪN

NHẬN XÉT, ĐÁNH GIÁ, CHO ĐIỂM

(Của giáo viên phản biện)

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Điểm:…… **(bằng chữ:** ……**)**

Đồng ý/Không đồng ý cho sinh viên bảo vệ trước hội đồng chấm đồ án tốt nghiệp?

……**, ngày**…..... **tháng 12 năm 2024**

CÁN BỘ GIẢNG VIÊN PHẢN BIỆN

MỤC LỤC

BẢNG VIẾT TẮT VÀ THUẬT NGỮ

TỪ VIẾT TẮT VÀ THUẬT NGỮ	Ý NGHĨA
ML	Machine Learning (Học máy)
AI	Artificial Intelligence (Trí tuệ nhân tạo)
API	Application Programming Interface (Giao diện lập trình ứng dụng)
JSON	JavaScript Object Notation (Chuỗi ký tự đối tượng JavaScript)
LSTM	Long Short-Term Memory (Mô hình học sâu với bộ nhớ ngắn hạn và dài hạn)
CNN	Convolutional Neural Network (Mạng nơ-ron tích chập)
RNN	Recurrent Neural Network (Mạng nơ-ron hồi tiếp)
API Key	Khoá API (Khoá bảo mật dùng để xác thực truy cập API)
GPU	Graphics Processing Unit (Đơn vị xử lý đồ họa, thường dùng trong huấn luyện mô hình học máy)
CPU	Central Processing Unit (Đơn vị xử lý trung tâm)
URL	Uniform Resource Locator (Địa chỉ tài nguyên đồng nhất)
HTTP	Hypertext Transfer Protocol (Giao thức truyền tải siêu văn bản)
HTTPS	Hypertext Transfer Protocol Secure (Giao thức truyền tải siêu văn bản bảo mật)
URL	Uniform Resource Locator (Địa chỉ tài nguyên đồng nhất)
IoT	Internet of Things (Mạng lưới vạn vật kết nối)
CSV	Comma-Separated Values (Dữ liệu phân tách bằng dấu phẩy)
XML	eXtensible Markup Language (Ngôn ngữ đánh dấu mở rộng)

DANH SÁCH HÌNH VẼ

DANH SÁCH BẢNG

MỞ ĐẦU

Trong bối cảnh giao thông đô thị ngày càng phát triển, việc dự báo lưu lượng giao thông trước khi di chuyển đã trở thành nhu cầu thiết yếu. Những ùn tắc giao thông không chỉ gây lãng phí thời gian, mà còn ảnh hưởng nghiêm trọng đến chất lượng sống và môi trường.

Song hành cùng sự phát triển nhanh chóng của công nghệ thông tin, việc ứng dụng Machine Learning vào các lĩnh vực đời sống thực tiễn ngày càng trở nên phổ biến và cần thiết. Kết hợp Machine Learning trong xử lý dữ liệu giao thông giúp các ứng dụng có khả năng phân tích dữ liệu tự động và dự báo chính xác hơn.

Đề tài "Phát triển ứng dụng web tích hợp dự báo giao thông sử dụng Machine Learning" nhấn mạnh việc kết hợp giữa các kỹ thuật xử lý Machine Learning và công nghệ phát triển web hiện đại, nhằm tạo nên một hệ thống hỗ trợ quyết định hiệu quả dành cho người dùng trong việc lên kế hoạch di chuyển.

Nội dung của đồ án bao gồm các phần sau:

Chương I: Giới thiệu

1. Lý do chọn đề tài: Dự báo giao thông là nhu cầu thiết yếu trong các đô thị, giúp giảm tắc nghẽn và tiết kiệm thời gian di chuyển.
2. Mục tiêu nghiên cứu: Phát triển hệ thống dự báo giao thông sử dụng Machine Learning, giúp người dùng đưa ra quyết định di chuyển tối ưu.

Chương II: Phương pháp và công nghệ sử dụng

1. Phương pháp: Thu thập và xử lý dữ liệu, huấn luyện mô hình ML, phát triển ứng dụng web tích hợp các API.
2. Công nghệ: LSTM (Machine Learning), React.js, Vite, Node.js, HERE MAP API, OpenWeatherMap API, Google Maps API.
3. Công cụ: Visual Studio Code, GitHub, và Postman

Chương III: Thiết kế hệ thống

1. Kiến trúc hệ thống: Mô tả các thành phần hệ thống (frontend, backend, ML model).
2. Quy trình hoạt động của hệ thống.
3. Các thành phần hệ thống: Cơ sở dữ liệu, mô-đun xử lý, dịch vụ API.

Chương IV: Huấn luyện và đánh giá mô hình dự báo

1. Thu thập và tiền xử lý dữ liệu.
2. Huấn luyện mô hình: Sử dụng các thuật toán đã chọn để huấn luyện mô hình, điều chỉnh tham số để đạt được độ chính xác cao nhất.
3. Đánh giá mô hình: Sử dụng các chỉ số đánh để đo lường hiệu quả của mô hình và so sánh giữa các mô hình khác nhau.
4. Tối ưu hóa mô hình: Nêu ra các phương pháp cải thiện hiệu suất của mô hình, như kỹ thuật cross-validation và tuning

Chương V: Phát triển và triển khai ứng dụng web

1. Phát triển frontend: Lập trình và tích hợp các tính năng của giao diện người dùng với React.js và Vite.
2. Phát triển backend: Xây dựng API và xử lý yêu cầu từ frontend bằng Node.js và Flask/Django.
3. Tích hợp mô hình Machine Learning: Kết nối mô hình dự báo với ứng dụng để cung cấp dữ liệu theo thời gian thực.
4. Tối ưu hóa ứng dụng: Thực hiện các bước cải thiện hiệu suất và tốc độ của ứng dụng, đảm bảo đáp ứng nhanh và hiệu quả.

Kết luận

1. Tổng hợp kết quả: Tóm tắt các chức năng của ứng dụng và hiệu quả dự báo lưu lượng giao thông.
2. Đánh giá: Phân tích những khó khăn và thách thức trong quá trình thực hiện, cũng như những bài học rút ra.
3. Hướng phát triển tương lai: Đề xuất các hướng nghiên cứu tiếp theo và cách mở rộng ứng dụng.

CHƯƠNG I. GIỚI THIỆU

Giao thông đô thị là một trong những vấn đề phức tạp nhất mà các thành phố lớn trên toàn cầu phải đối mặt. Tình trạng tắc nghẽn giao thông không chỉ ảnh hưởng trực tiếp đến thời gian di chuyển mà còn tác động sâu rộng đến nền kinh tế, môi trường và chất lượng sống của người dân.

Theo INRIX Global Traffic Scorecard 2022, tại các thành phố lớn như London (Anh), một người dân trung bình lãng phí 156 giờ mỗi năm trong các tình trạng giao thông ùn tắc, tương đương chi phí lên đến 1.377 USD mỗi người/năm. Tại Việt Nam, các thành phố như Hà Nội và TP.HCM cũng đối mặt với tình trạng nghiêm trọng. Theo báo cáo của Ngân hàng Thế giới (World Bank) năm 2019, tắc nghẽn giao thông ở Hà Nội và TP.HCM gây tổn thất kinh tế khoảng 1,5-2% GDP hàng năm.

Tình trạng này không chỉ dừng lại ở các con số kinh tế. Một nghiên cứu của Cục Bảo vệ Môi trường Hoa Kỳ (EPA) cho thấy, việc ùn tắc giao thông kéo dài dẫn đến việc phát thải 300 triệu tấn CO2 mỗi năm, đóng góp đáng kể vào ô nhiễm không khí toàn cầu.

Ngoài ra, giao thông phi tuyến tính ở các khu vực đô thị còn chịu ảnh hưởng bởi các yếu tố khó dự đoán như:

- Sự kiện bất ngờ: Tai nạn giao thông, sửa chữa đường.
- Thời tiết cực đoan: Mưa lớn gây ngập úng, nắng nóng ảnh hưởng đến hạ tầng đường bộ.

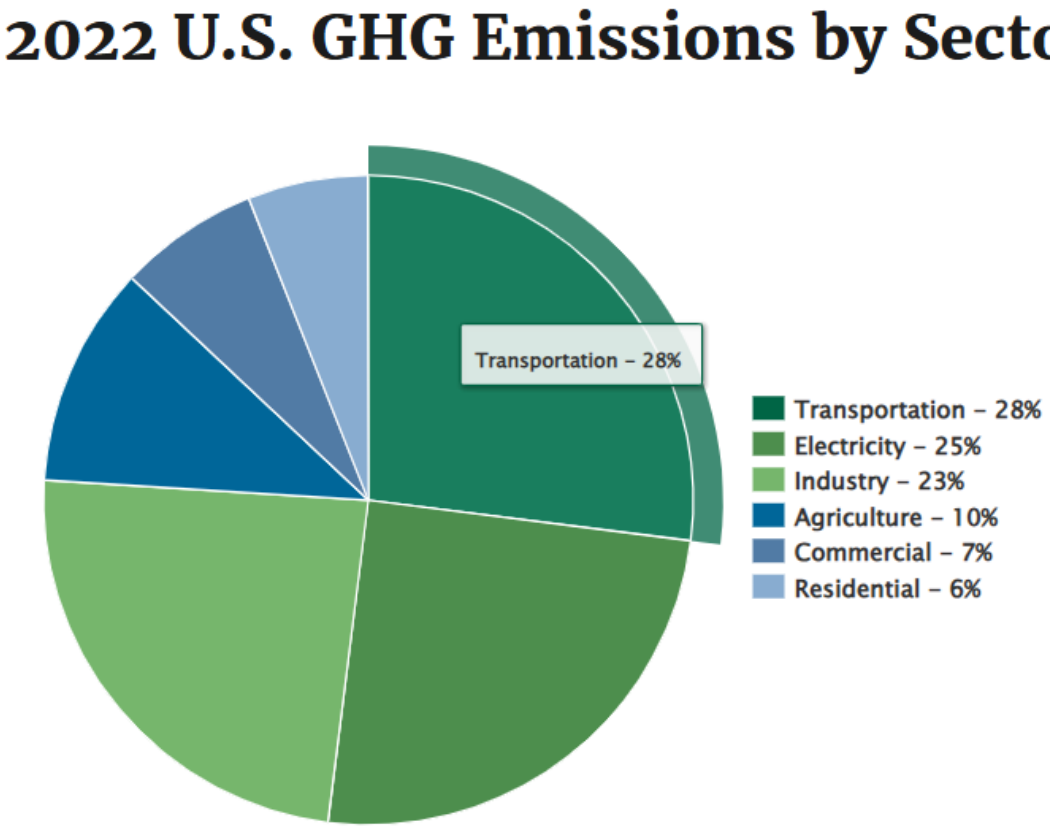
Trước thực trạng này, các giải pháp công nghệ hiện đại đang dần được triển khai để giảm thiểu vấn đề. Một trong những hướng đi hiệu quả nhất là sử dụng Machine Learning (ML), tận dụng khả năng phân tích và dự đoán dựa trên lượng lớn dữ liệu phức tạp.

1.1 Lý do chọn đề tài

Tắc nghẽn giao thông không chỉ ảnh hưởng tiêu cực đến chất lượng sống mà còn gây ra những thiệt hại lớn về mặt kinh tế và môi trường. Các ví dụ điển hình như:

- Lãng phí thời gian: Tài xế ở TP.HCM trung bình dành hơn 1 giờ mỗi ngày trong các tình trạng giao thông ùn tắc (theo Sở Giao thông Vận tải TP.HCM, 2023).

- Chi phí nhiên liệu: Mỗi năm, người dân Việt Nam tiêu tốn hơn 15 triệu lít xăng do thời gian chờ đợi trong kẹt xe (theo báo cáo của Viện Nghiên cứu Kinh tế Phát triển, 2022).
- Phát thải khí CO2: Ô nhiễm từ khí thải xe máy tại Hà Nội chiếm tới 70% lượng khí CO2 phát thải toàn thành phố (Ngân hàng Thế giới, 2019).
- Theo báo cáo của Cục Bảo vệ Môi trường Hoa Kỳ (EPA), ngành giao thông chiếm 28% tổng lượng phát thải khí nhà kính (GHG) tại Hoa Kỳ trong năm 2022, trở thành nguồn phát thải lớn nhất trong các lĩnh vực (EPA, 2022). Lượng khí thải này đến từ các phương tiện như ô tô, xe tải, máy bay thương mại và đường sắt.



Hình 1.1 Biểu đồ thể hiện ngành giao thông chiếm 28% tổng lượng phát thải khí nhà kính (GHG) tại Hoa Kỳ trong năm 2022

Những con số này cho thấy tầm quan trọng của việc tìm ra các giải pháp hiệu quả để giải quyết bài toán giao thông. Trong đó, việc ứng dụng Machine Learning trong dự báo giao thông hứa hẹn sẽ tạo ra đột phá nhờ:

- Dự báo chính xác dựa trên dữ liệu thời gian thực.

- Xử lý dữ liệu phi tuyến tính phức tạp như thời tiết, sự kiện và mật độ giao thông.

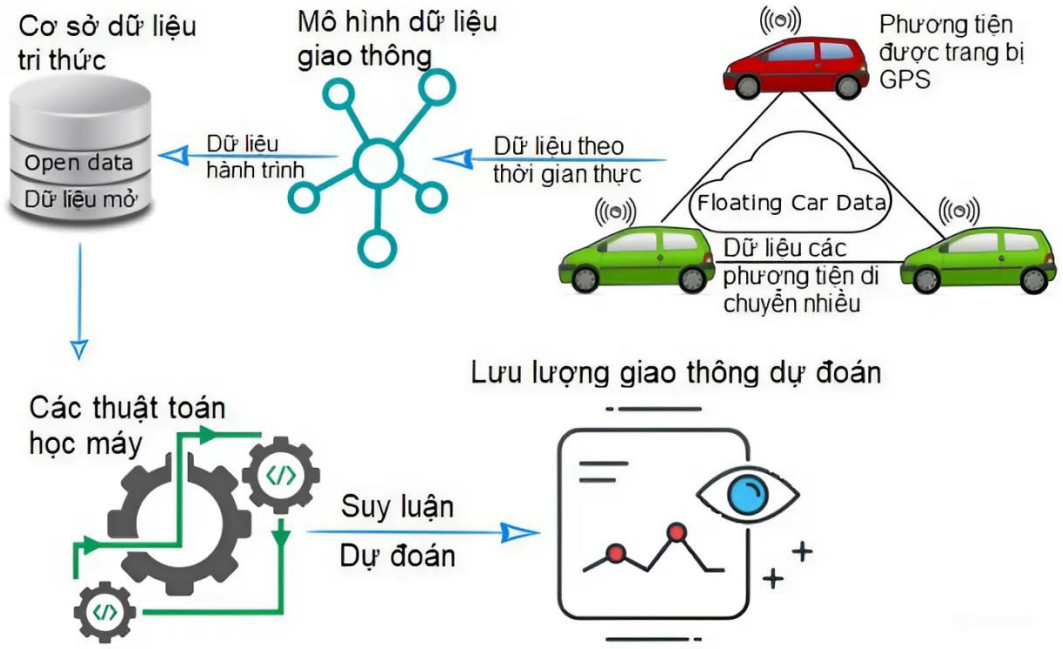
Hệ thống này không chỉ giúp người dân đưa ra quyết định di chuyển tối ưu mà còn góp phần vào việc giảm thiểu thiệt hại kinh tế và bảo vệ môi trường.

1.2 Mục tiêu nghiên cứu

Mục tiêu chính của đồ án này là phát triển một hệ thống dự báo giao thông sử dụng các mô hình Machine Learning, cung cấp các dự báo về lưu lượng giao thông cho người dùng theo thời gian thực. Hệ thống này giúp người tham gia giao thông có thể lên kế hoạch di chuyển hiệu quả hơn, tránh tắc nghẽn và tiết kiệm thời gian.

Các mục tiêu cụ thể bao gồm:

1. Xây dựng mô hình dự báo giao thông: Sử dụng các thuật toán ML để dự đoán lưu lượng giao thông dựa trên các yếu tố như thời gian, thời tiết và các điều kiện giao thông khác. Sử dụng các mô hình Machine Learning tiên tiến (như LSTM) để phân tích và dự báo tình trạng giao thông theo thời gian thực.
2. Phát triển ứng dụng web: Xây dựng giao diện người dùng thân thiện, dễ sử dụng để hiển thị kết quả dự báo và giúp người dùng dễ dàng tương tác với hệ thống.
3. Tích hợp dữ liệu thời gian thực: Kết nối với các API công khai như Google Map API, MapQuest API, HERE Map API và OpenWeatherMap API để thu thập dữ liệu giao thông và thời tiết, từ đó cung cấp thông tin chính xác về tình trạng giao thông.



Hình 1.2 Một mô hình dự đoán lưu lượng giao thông sử dụng ML

Mục tiêu lâu dài là tạo ra một nền tảng mở, có thể mở rộng và tích hợp thêm nhiều tính năng nâng cao, như phân tích các tuyến đường tối ưu hoặc dự đoán sự cố giao thông.

1.3 Phạm vi nghiên cứu

Đồ án này tập trung vào việc sử dụng dữ liệu giao thông và thời tiết từ các nguồn mở và API công khai để xây dựng hệ thống dự báo giao thông. Chỉ sử dụng dữ liệu giao thông, ngày giờ và thời tiết, chưa tính đến các yếu tố khác như sự kiện xã hội hay công trình xây dựng.

Cụ thể, dữ liệu giao thông sẽ được thu thập từ Google Map API, MapQuest API, HERE Map API, trong khi dữ liệu thời tiết sẽ được lấy từ OpenWeatherMap API. Các yếu tố này sẽ được sử dụng làm đầu vào cho các mô hình Machine Learning để dự đoán lưu lượng giao thông.

1.4 Kết luận Chương I

Chương I đã giới thiệu bối cảnh và tầm quan trọng của vấn đề giao thông đô thị, nhấn mạnh các tác động tiêu cực của tình trạng ùn tắc giao thông đối với kinh tế, môi trường, và chất lượng cuộc sống. Các số liệu và nghiên cứu cụ thể từ các nguồn uy tín đã minh họa rõ ràng sự cấp thiết của việc áp dụng các giải pháp công nghệ hiện đại để giải quyết bài toán giao thông.

Việc lựa chọn đề tài "Phát triển ứng dụng web tích hợp dự báo giao thông sử dụng Machine Learning" được lý giải từ nhu cầu cấp bách trong việc cải thiện lưu thông đô thị và giảm thiểu thiệt hại kinh tế lẫn môi trường. Đề tài hứa hẹn khai thác tối ưu tiềm năng của Machine Learning, đặc biệt là các mô hình tiên tiến như LSTM, để dự báo chính xác lưu lượng giao thông dựa trên dữ liệu thời gian thực.

Mục tiêu của nghiên cứu không chỉ dừng lại ở việc xây dựng mô hình dự báo mà còn hướng đến phát triển một ứng dụng web thân thiện, tích hợp dữ liệu thời gian thực từ các API công khai. Hệ thống này kỳ vọng sẽ hỗ trợ người dùng lên kế hoạch di chuyển hiệu quả, giảm thiểu tình trạng tắc nghẽn giao thông và đóng góp tích cực vào sự phát triển bền vững của xã hội.

Chương 1 đặt nền móng lý thuyết và phương hướng nghiên cứu rõ ràng, từ đó làm cơ sở để triển khai các nội dung cụ thể trong các chương tiếp theo.

CHƯƠNG II. PHƯƠNG PHÁP VÀ CÔNG NGHỆ SỬ DỤNG

Việc dự báo giao thông hiệu quả đòi hỏi sự kết hợp giữa các phương pháp khoa học và công nghệ hiện đại. Chương này trình bày chi tiết các phương pháp thu thập, xử lý dữ liệu và xây dựng mô hình Machine Learning, cùng với các công nghệ tiên tiến được sử dụng trong quá trình phát triển hệ thống và ứng dụng web.

2.1 Phương pháp nghiên cứu

Trong đồ án này, các phương pháp nghiên cứu chủ yếu bao gồm ba giai đoạn: thu thập và xử lý dữ liệu, huấn luyện mô hình Machine Learning (ML), và phát triển ứng dụng web. Mỗi giai đoạn sử dụng các phương pháp và kỹ thuật cụ thể như sau:

- **Thu thập và xử lý dữ liệu**:
    - **Thu thập dữ liệu**: Dữ liệu về giao thông được thu thập từ Google Map API, MapQuest API, HERE Map API, trong khi dữ liệu thời tiết được lấy từ OpenWeatherMap API. Các dữ liệu này sẽ được thu thập theo thời gian thực hoặc từ các nguồn lịch sử, tùy theo yêu cầu của hệ thống.
    - **Tiền xử lý dữ liệu**: Dữ liệu thu thập được sẽ phải được làm sạch và chuẩn hóa để loại bỏ các nhiễu và đảm bảo tính nhất quán. Các bước xử lý bao gồm loại bỏ các giá trị thiếu, chuẩn hóa các đơn vị (như nhiệt độ từ Celsius sang Fahrenheit), và xử lý các dữ liệu ngoài phạm vi.
- **Huấn luyện mô hình Machine Learning**:
    - Các mô hình Machine Learning sẽ được huấn luyện dựa trên dữ liệu giao thông và thời tiết. Các thuật toán học máy như LSTM sẽ được sử dụng để tạo ra các mô hình dự báo lưu lượng giao thông. Việc huấn luyện mô hình bao gồm việc chọn lựa tham số, tối ưu hóa mô hình và đánh giá độ chính xác của mô hình.
- **Phát triển ứng dụng web**:
    - **Frontend**: Xây dựng giao diện người dùng bằng React.js và Vite để hiển thị kết quả dự báo giao thông và cung cấp các chức năng tương tác với người dùng. Các giao diện sẽ hiển thị bản đồ, chỉ đường và kết quả dự báo lưu lượng giao thông theo thời gian thực.

- **Backend**: Phát triển API backend với Flask, chịu trách nhiệm xử lý yêu cầu từ frontend và kết nối với các mô hình ML để đưa ra dự báo giao thông.

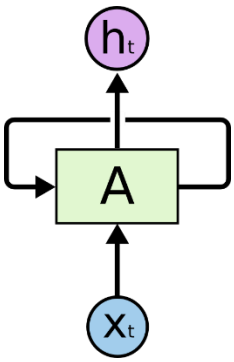
2.2 Công nghệ sử dụng

Đồ án sử dụng một số công nghệ hiện đại để triển khai hệ thống dự báo giao thông, bao gồm các công nghệ sau:

- **Machine Learning**:

LSTM (Long Short-Term Memory): LSTM (Long Short-Term Memory) là một kiến trúc mạng nơ-ron hồi tiếp (RNN - Recurrent Neural Network) đặc biệt, được thiết kế để xử lý và phân tích dữ liệu có mối quan hệ tuần tự (thời gian hoặc chuỗi) và khắc phục những hạn chế của RNN truyền thống. LSTM được giới thiệu lần đầu tiên bởi Sepp Hochreiter và Jürgen Schmidhuber vào năm 1997.

Mạng nơ-ron truy hồi (RNN - Recurrent Neural Network): Mạng nơ-ron truy hồi (RNN) được thiết kế đặc biệt để xử lý dữ liệu tuần tự, chẳng hạn như chuỗi văn bản hoặc âm thanh. Đặc điểm chính của RNN là nó có một vòng lặp, cho phép thông tin từ các bước trước đó ảnh hưởng đến kết quả tại bước hiện tại.

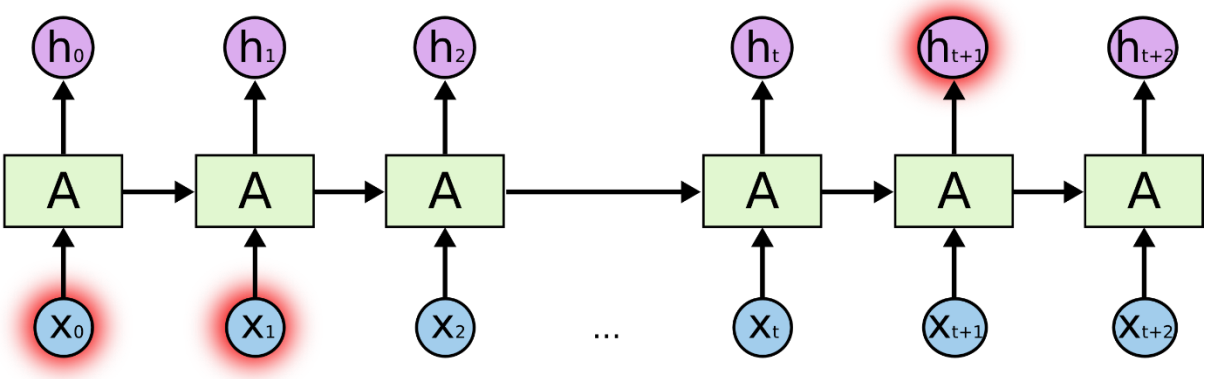


Hình 2.1 Kiến trúc của một mạng nơ ron truy hồi

Ví dụ, khi đọc một câu văn, chúng ta cần biết các từ trước đó để hiểu ý nghĩa của từ hiện tại. RNN thực hiện điều này bằng cách lưu trữ một "trạng thái ẩn" (hidden state) chứa thông tin của các bước trước và sử dụng nó để dự đoán bước tiếp theo.

Tuy nhiên, nếu dữ liệu quá dài (ví dụ như một chuỗi văn bản rất dài), RNN gặp khó khăn trong việc ghi nhớ các thông tin từ xa. Điều này được gọi là vấn đề phụ thuộc dài hạn (long-term dependencies).

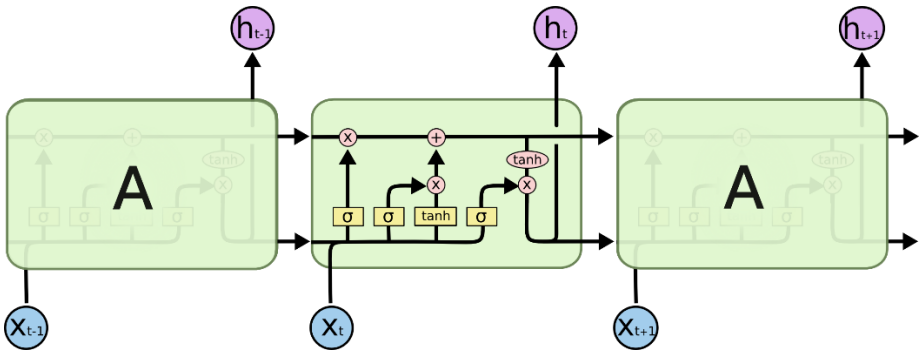
Hạn chế của mạng RNN: RNN có thể ghi nhớ các thông tin ngắn hạn tốt nhưng gặp khó khăn khi xử lý thông tin dài hạn. Điều này xảy ra vì trong quá trình học, các thông tin từ xa có xu hướng bị "mất mát" hoặc bị "triệt tiêu". Vấn đề này được gọi là vanishing gradient (độ dốc biến mất) trong quá trình tối ưu hóa.



Hình 2.2 RNN và vấn đề phụ thuộc xa

Ví dụ: Khi đọc câu "Bống đi học nhưng không mang áo mưa. Trên đường đi học trời mưa. Cặp sách của Bống bị ướt", để suy ra "cặp sách bị ướt", chúng ta cần nhớ các chi tiết trước đó như "trời mưa" và "không mang áo mưa". RNN gặp khó khăn trong việc lưu giữ những thông tin quan trọng này.

Mạng LSTM (Long Short-Term Memory): Để khắc phục hạn chế của RNN, mạng LSTM được ra đời. LSTM là một phiên bản cải tiến của RNN, được thiết kế để ghi nhớ cả thông tin ngắn hạn và dài hạn tốt hơn. LSTM có thể chọn lọc thông tin nào cần lưu giữ hoặc bỏ qua, nhờ vào các cổng đặc biệt trong cấu trúc của nó.

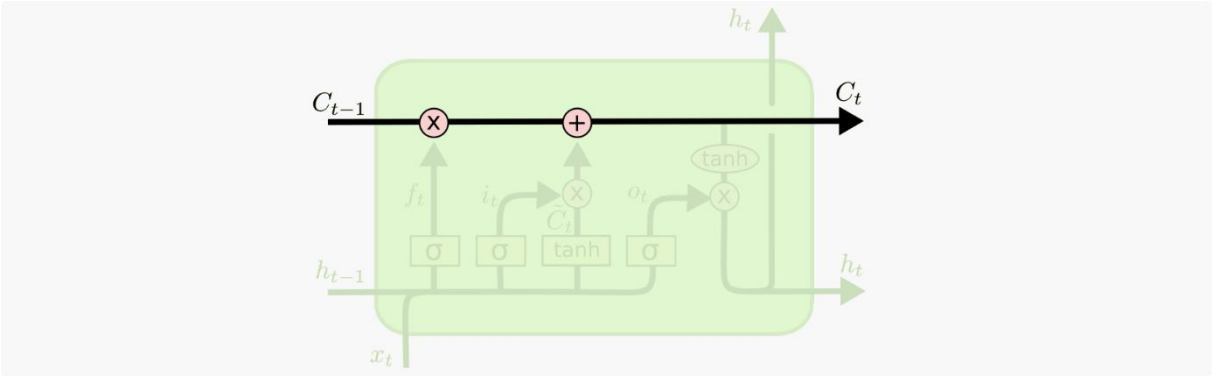


Hình 2.3 Kiến trúc dạng chuỗi LSTM với 4 tầng mạng nơ ron tương tác với nhau một cách rất đặc biệt.

Ý tưởng chính của LSTM: Thành phần quan trọng trong LSTM là ô trạng thái (cell state). Hãy tưởng tượng ô trạng thái giống như một băng chuyền mang thông tin đi qua toàn bộ chuỗi dữ liệu. LSTM sử dụng các cổng để điều chỉnh thông tin trên băng chuyền này:

- Cổng quên (Forget Gate): Quyết định thông tin nào cần loại bỏ.
- Cổng vào (Input Gate): Quyết định thông tin nào cần thêm vào.
- Cổng đầu ra (Output Gate): Quyết định thông tin nào sẽ được sử dụng để tạo đầu ra.

Các cổng này được điều khiển bởi các phép toán dựa trên hàm sigmoid (cho giá trị từ 0 đến 1) và hàm tanh (giới hạn giá trị từ -1 đến 1).

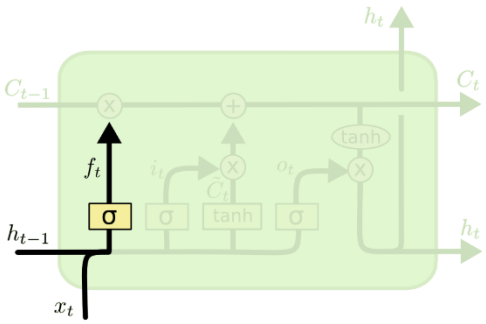


Hình 2.4 Đường đi của ô trạng thái (cell state) trong mạng LSTM

Cách hoạt động của LSTM:

Mỗi bước trong LSTM diễn ra như sau:

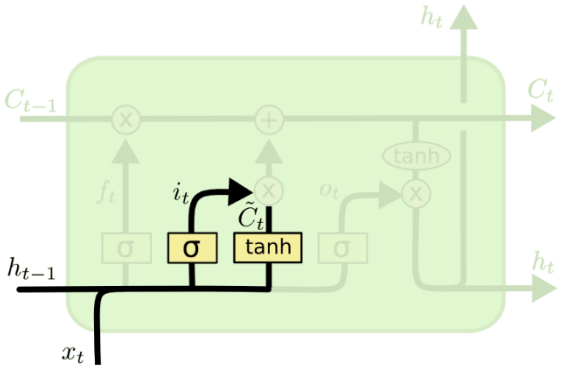
1. Quyết định thông tin cần quên: Cổng quên sẽ xác định thông tin nào từ ô trạng thái cũ cần loại bỏ. Ví dụ: Khi có một chủ ngữ mới xuất hiện, LSTM sẽ "quên" chủ ngữ cũ.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Hình 2.5 Tầng cổng quên (forget gate layer)

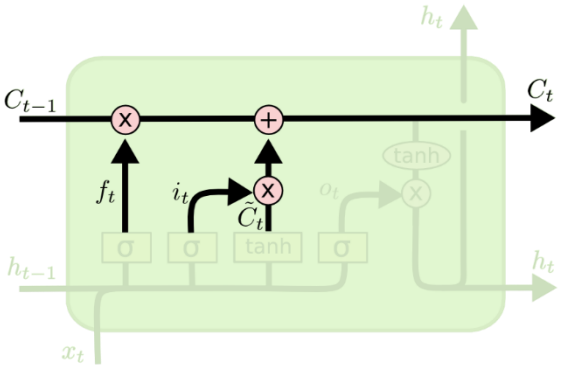
2. Quyết định thông tin cần thêm vào: Cổng vào sẽ xác định thông tin mới nào cần được thêm vào ô trạng thái. Một hàm tanh sẽ tạo ra giá trị mới, và giá trị này được cập nhật vào ô trạng thái.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Hình 2.6 Cập nhật giá trị cho ô trạng thái bằng cách kết hợp 2 kết quả từ tầng cổng vào và tầng ẩn hàm tanh

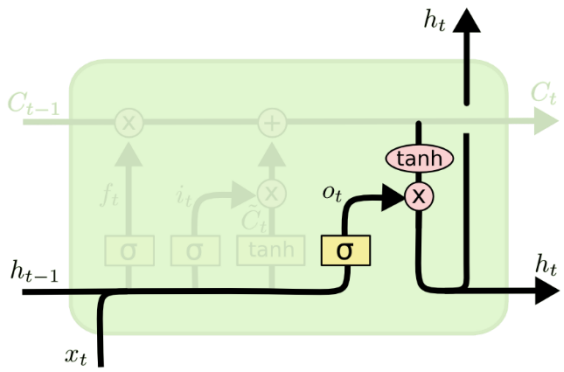
3. Cập nhật ô trạng thái: Ô trạng thái được làm mới bằng cách kết hợp thông tin cũ (sau khi quên) và thông tin mới (sau khi thêm vào).



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Hình 2.7 Ô trạng thái mới

4. Quyết định đầu ra: Đầu ra sẽ dựa trên ô trạng thái, nhưng sẽ được lọc qua một cổng sigmoid và hàm tanh để đảm bảo thông tin trả về phù hợp với mục tiêu



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Hình 2.8 Điều chỉnh thông tin đầu ra hàm tanh

Hỗ trợ xây dựng triển khai LSTM: TensorFlow và Keras đều hỗ trợ mạnh mẽ việc triển khai LSTM, một dạng mạng nơ-ron hồi tiếp (RNN) đặc biệt, giúp xử lý chuỗi dữ liệu và dự đoán các giá trị trong tương lai từ dữ liệu quá khứ. Việc sử dụng TensorFlow cho LSTM nhằm:

- LSTM hiệu quả: TensorFlow cung cấp lớp LSTM và Bidirectional LSTM, cho phép người dùng dễ dàng triển khai mô hình LSTM để học các phụ thuộc theo thời gian trong dữ liệu chuỗi. TensorFlow đã tối ưu hóa các lớp này để xử lý nhanh và hiệu quả.
- Khả năng tùy chỉnh cao: Người dùng có thể tùy chỉnh cấu trúc LSTM như thêm số lượng lớp, số lượng đơn vị LSTM, điều chỉnh cách sử dụng Bidirectional LSTM, và các tham số khác để tối ưu hóa mô hình dựa trên bài toán cụ thể.
- Xử lý các vấn đề phức tạp: Mô hình LSTM có thể được sử dụng để giải quyết nhiều loại bài toán chuỗi thời gian phức tạp, từ dự báo giao thông đến nhận dạng giọng nói, dự báo tài chính, và nhiều ứng dụng khác. TensorFlow giúp triển khai các mô hình này một cách đơn giản mà không cần lo lắng về các vấn đề kỹ thuật chi tiết.
- TensorFlow và Keras cung cấp các kỹ thuật tối ưu hóa và phòng tránh overfitting, như:
    - Early Stopping: Đây là kỹ thuật giúp dừng huấn luyện sớm nếu mô hình không còn cải thiện trên dữ liệu kiểm tra, tránh tình trạng overfitting và tiết kiệm tài nguyên.

- Dropout: Kỹ thuật này giúp loại bỏ ngẫu nhiên một phần của các nơ-ron trong quá trình huấn luyện, giúp mô hình trở nên tổng quát hơn và giảm nguy cơ overfitting.
- Optimizers: TensorFlow hỗ trợ các thuật toán tối ưu hóa như Adam, giúp cập nhật trọng số của mô hình một cách hiệu quả và nhanh chóng, đảm bảo mô hình không bị mắc kẹt trong các cực tiểu địa phương.

TensorFlow là một trong những thư viện học sâu phổ biến nhất hiện nay, được phát triển bởi Google. Đây là một nền tảng mã nguồn mở hỗ trợ việc xây dựng và triển khai các mô hình học máy, học sâu và học củng cố. Những lý do chính để chọn TensorFlow trong việc xây dựng mô hình LSTM là:

- Khả năng mở rộng: TensorFlow được tối ưu hóa để xử lý khối lượng dữ liệu lớn và có thể chạy trên nhiều thiết bị khác nhau, bao gồm CPU, GPU, và TPU, giúp tăng tốc quá trình huấn luyện mô hình, đặc biệt là với các mô hình như LSTM đòi hỏi tính toán cao.
- Hỗ trợ đa nền tảng: TensorFlow hỗ trợ triển khai mô hình trên các nền tảng khác nhau từ máy tính cá nhân đến các hệ thống phân tán trên đám mây (cloud). Điều này giúp mô hình LSTM có thể mở rộng dễ dàng và ứng dụng trong các trường hợp thực tế với dữ liệu lớn.
- Tối ưu hóa mô hình: TensorFlow cung cấp các công cụ và kỹ thuật tối ưu hóa mạnh mẽ như Gradient Tape, Automatic Differentiation, giúp việc tối ưu hóa các tham số mô hình (như trong trường hợp LSTM) trở nên hiệu quả và dễ dàng hơn.

Keras là một API cấp cao (high-level API) của TensorFlow, giúp người dùng dễ dàng xây dựng các mô hình học sâu mà không phải viết mã phức tạp. Keras giúp đơn giản hóa việc triển khai các lý thuyết phức tạp về mạng nơ-ron, bao gồm cả LSTM.

- Cấu trúc dễ hiểu: Keras cung cấp một cú pháp dễ sử dụng và dễ hiểu, giúp người dùng dễ dàng xây dựng, thử nghiệm và triển khai các mô hình học sâu. Chỉ cần một vài dòng mã là có thể xây dựng một mô hình LSTM mạnh mẽ.

- Khả năng linh hoạt cao: Mặc dù Keras đơn giản, nhưng nó vẫn cung cấp khả năng điều chỉnh các tham số sâu (hyperparameter tuning), cho phép tối ưu hóa mô hình và nâng cao hiệu suất khi làm việc với LSTM hoặc các mô hình khác.
- Tích hợp tốt với TensorFlow: Keras được tích hợp trực tiếp trong TensorFlow, giúp sử dụng các công cụ mạnh mẽ của TensorFlow mà vẫn giữ được sự đơn giản trong việc xây dựng mô hình. Điều này tạo ra một môi trường thống nhất dễ dàng sử dụng các công cụ, chiến lược huấn luyện và triển khai mạnh mẽ mà TensorFlow cung cấp.

❖ **Frontend**:

1. **React.js**: React.js là thư viện JavaScript phổ biến dùng để xây dựng giao diện người dùng động. Nó sẽ được sử dụng để xây dựng các trang web tương tác, giúp người dùng dễ dàng xem và tương tác với các dự báo giao thông. Đây là Là framework chính để xây dựng giao diện.
2. **Vite**: Vite là công cụ phát triển frontend hiện đại với tính năng "hot module replacement" (HMR), giúp tăng tốc quá trình phát triển ứng dụng bằng cách cập nhật nhanh chóng các thay đổi trên giao diện người dùng.
3. **React-datepicker**: Để chọn ngày và giờ một cách thân thiện với người dùng.
4. **Axios**: Gửi yêu cầu HTTP đến backend và các API bên ngoài (như OpenWeatherMap, Google Maps API).
5. **React Google Maps API**: Hiển thị bản đồ, thêm layer giao thông và render chỉ dẫn đường đi.

❖ **Backend**:

1. **Flask – Web Framework**: Flask là một micro web framework được viết bằng Python.
    - Flask là một micro web framework mã nguồn mở được viết bằng Python, giúp xây dựng các ứng dụng web dễ dàng và linh hoạt. Flask rất phù hợp cho việc xây dựng các ứng dụng web nhẹ, không yêu cầu cấu trúc phức tạp. Flask cung cấp các công cụ cơ bản để xử lý các yêu cầu HTTP, routing, và tạo các response cho client.

- Flask được sử dụng để xây dựng ứng dụng web, bao gồm việc định nghĩa các route (đường dẫn) và xử lý các yêu cầu HTTP (như POST). Ở hệ thống này, Flask giúp xây dựng API RESTful để giao tiếp với frontend.

2. **Tích hợp CORS** (Cross-Origin Resource Sharing) với thư viện flask_cors để cho phép frontend từ các domain khác gửi yêu cầu.
    - CORS là cơ chế bảo mật trong trình duyệt web, cho phép hoặc từ chối các yêu cầu HTTP từ các nguồn (domain) khác nhau. Đây là một biện pháp bảo vệ các trang web khỏi các cuộc tấn công Cross-Site Request Forgery (CSRF).
    - Để cho phép frontend (nằm trên một domain khác) có thể gửi yêu cầu tới backend Flask mà không gặp phải vấn đề CORS, mã nguồn sử dụng thư viện flask_cors để cấu hình cho phép CORS. Điều này giúp các yêu cầu từ frontend (ví dụ: từ một ứng dụng React) có thể dễ dàng giao tiếp với backend mà không bị chặn bởi trình duyệt.
3. **TensorFlow/Keras**: TensorFlow là một thư viện mã nguồn mở cho các tác vụ học sâu, và Keras là một API cấp cao cho TensorFlow.
    - TensorFlow và Keras được sử dụng để xây dựng và triển khai mô hình học sâu (ở đây là mô hình LSTM). Mô hình LSTM này dùng để dự đoán dữ liệu giao thông. Mô hình được tải lại từ tệp đã huấn luyện và sử dụng để dự đoán kết quả.
4. **Scikit-learn:** Scikit-learn là một thư viện Python phổ biến được sử dụng cho các tác vụ học máy như phân loại, hồi quy, và tiền xử lý dữ liệu.
    - Trong ứng dụng này, scikit-learn được sử dụng để chuẩn hóa dữ liệu (với StandardScaler) và mã hóa dữ liệu phân loại (với LabelEncoder và ColumnTransformer). Các mô hình học máy và phương pháp tiền xử lý dữ liệu đã được huấn luyện và lưu trữ.
5. **Joblib:** Joblib là một thư viện Python được sử dụng để lưu và tải các đối tượng lớn, chẳng hạn như mô hình học máy.
    - Joblib được sử dụng để lưu và tải các mô hình học máy đã huấn luyện và các đối tượng tiền xử lý dữ liệu (như LabelEncoder và

ColumnTransformer). Điều này giúp giảm thời gian tải lại mô hình và cho phép tái sử dụng chúng trong các lần chạy sau.

6. **NumPy:** NumPy là một thư viện Python chuyên về tính toán số học và xử lý mảng (arrays).
    - NumPy được sử dụng để xử lý và chuyển đổi dữ liệu đầu vào thành mảng numpy, giúp chuẩn hóa và biến đổi dữ liệu trước khi đưa vào mô hình học máy.
7. **HTTP Methods (POST):** HTTP là giao thức chính để truyền tải thông tin giữa client và server trên nền tảng web. HTTP hỗ trợ nhiều phương thức khác nhau để thực hiện các thao tác như GET, POST, PUT, DELETE, v.v.
    - Trong hệ thống, phương thức HTTP POST được sử dụng để nhận dữ liệu từ client. Phương thức POST cho phép frontend gửi dữ liệu (dưới dạng JSON) đến backend, nơi dữ liệu được xử lý để thực hiện dự đoán
8. **Python**: Python là ngôn ngữ lập trình chính được sử dụng trong backend này.
    - Python được sử dụng để xây dựng toàn bộ logic backend, bao gồm việc xử lý các yêu cầu HTTP từ frontend, triển khai mô hình học máy, và thực hiện các phép toán với dữ liệu.
9. **Multithreading (Đa luồng)** : Threading là kỹ thuật trong lập trình cho phép thực hiện đồng thời nhiều tác vụ.
    - Mặc dù không sử dụng trực tiếp threading trong mã, nhưng trong cách thiết kế của backend, việc xử lý các dự đoán (dự đoán dữ liệu giao thông) có thể được thực hiện trong nền, giúp giảm độ trễ và không làm gián đoạn phản hồi cho người dùng.
10. **RESTful API**: REST (Representational State Transfer) là một kiểu kiến trúc phần mềm cho các dịch vụ web.
    - Flask cung cấp một API RESTful cho phép frontend gửi yêu cầu POST với dữ liệu JSON và nhận về kết quả dự đoán từ mô hình học máy. Dữ liệu được truyền qua lại giữa frontend và backend theo chuẩn REST

11. **JSON (JavaScript Object Notation)**:

- JSON là một định dạng dữ liệu phổ biến, dễ đọc và dễ viết cho người và máy tính. JSON là định dạng chính để trao đổi dữ liệu giữa frontend và backend, đặc biệt trong các ứng dụng web.
- Mã nguồn nhận và gửi dữ liệu dưới dạng JSON. Flask sử dụng request.json để lấy dữ liệu JSON từ client, và sử dụng jsonify để trả kết quả dưới dạng JSON. Điều này giúp frontend có thể dễ dàng xử lý và hiển thị kết quả dự đoán từ server.

❖ **API**:

1. **HERE Map API Google Map API, MapQuest API:** Đây là API cung cấp dữ liệu giao thông và bản đồ chi tiết, được sử dụng để thu thập thông tin về tình trạng giao thông hiện tại và dự báo giao thông trong tương lai.
2. **OpenWeatherMap API**: API này cung cấp dữ liệu về thời tiết, bao gồm các yếu tố như nhiệt độ, độ ẩm, và điều kiện thời tiết. Thông tin này sẽ được tích hợp vào mô hình dự báo để giúp cải thiện độ chính xác của các dự báo giao thông.

2.3 Công cụ sử dụng

Đồ án sử dụng một số công cụ lập trình thân thiện với người dùng, miễn phí và hiện đại để triển khai hệ thống dự báo giao thông, bao gồm:

❖ **Công cụ lập trình và phát triển**:

1. **Visual Studio Code (VSCode)**: Sử dụng để viết mã nguồn, chỉnh sửa, và quản lý các tệp trong dự án. VSCode hỗ trợ rất nhiều extension hữu ích như:
   - **ESLint**: Đảm bảo chất lượng mã nguồn.
   - **Prettier**: Định dạng mã tự động.
   - **GitLens**: Theo dõi lịch sử thay đổi mã nguồn.

❖ **Công cụ kiểm tra và debug**:

1. **Browser DevTools**: Sử dụng trong trình duyệt (Chrome, Edge) để:
   - Debug giao diện người dùng.
   - Kiểm tra và theo dõi các yêu cầu API (request/response).

2. **Postman**:

   Công cụ kiểm tra và gửi các yêu cầu API, kiểm tra phản hồi từ backend.

❖ **Công cụ quản lý mã nguồn và theo dõi tiến độ:**:

1. **GitHub hoặc GitLab**:

   - Quản lý mã nguồn dự án.
   - Theo dõi lịch sử commit, pull request, và tiến độ phát triển.
   - Lưu trữ mã nguồn an toàn trên kho lưu trữ trực tuyến.

2.4 Kết luận Chương II

Chương II đã trình bày chi tiết các phương pháp nghiên cứu và công nghệ sử dụng trong đồ án. Quá trình nghiên cứu bao gồm ba giai đoạn chính: thu thập và xử lý dữ liệu, huấn luyện mô hình Machine Learning, và phát triển ứng dụng web. Cụ thể, việc thu thập dữ liệu giao thông và thời tiết từ các API như Google Maps, HERE, OpenWeatherMap và MapQuest đã cung cấp nguồn thông tin quan trọng. Dữ liệu này sau đó được tiền xử lý để đảm bảo tính chính xác và đồng nhất, sẵn sàng cho quá trình huấn luyện mô hình dự báo.

Về phần học máy, các mô hình LSTM được lựa chọn để dự đoán lưu lượng giao thông, với việc tối ưu hóa tham số và đánh giá độ chính xác nhằm đạt hiệu quả dự báo cao nhất. Phần phát triển ứng dụng web sử dụng các công nghệ hiện đại như React.js và Vite cho frontend, giúp tạo ra giao diện người dùng tương tác và mượt mà, trong khi Flask và Node.js được sử dụng để xây dựng backend API, kết nối với mô hình ML và xử lý yêu cầu từ frontend.

Các công cụ lập trình như Visual Studio Code, GitHub, và Postman hỗ trợ quá trình phát triển và quản lý mã nguồn hiệu quả, giúp theo dõi tiến độ và đảm bảo chất lượng mã. Tổng thể, các phương pháp và công nghệ đã được áp dụng hợp lý và tối ưu, nhằm xây dựng hệ thống dự báo giao thông chính xác và dễ sử dụng.

CHƯƠNG III: THIẾT KẾ HỆ THỐNG

Chương này mô tả chi tiết về kiến trúc hệ thống, giao diện người dùng, và các thành phần trong hệ thống dự báo giao thông sử dụng Machine Learning. Các thành phần hệ thống sẽ được thiết kế sao cho tích hợp tốt với nhau, từ thu thập dữ liệu, xử lý đến cung cấp thông tin dự báo giao thông cho người dùng.

3.1 Tổng quan hệ thống

Hệ thống dự đoán lưu lượng giao thông sử dụng Machine Learning (ML) gồm hai thành phần chính: Frontend và Backend. Frontend chịu trách nhiệm giao tiếp với người dùng, thu thập dữ liệu và hiển thị kết quả dự đoán, trong khi Backend xử lý các yêu cầu từ phía người dùng, thực hiện tiền xử lý dữ liệu và trả về kết quả dự đoán từ mô hình LSTM.

Frontend là giao diện người dùng được xây dựng bằng React & Vite, cho phép người dùng nhập các thông tin như ngày, múi giờ, thời tiết và nhiệt độ. Dữ liệu này sau đó được gửi qua API HTTP POST đến Backend để xử lý và trả về kết quả dự đoán lưu lượng giao thông.

Backend sử dụng Flask để cung cấp các API cho hệ thống. Các API này nhận dữ liệu từ Frontend, thực hiện tiền xử lý, chuẩn hóa, và cuối cùng sử dụng mô hình LSTM để đưa ra dự đoán.

3.2 Kiến trúc hệ thống

Hình 3.1 Mô hình ứng dụng hệ thống

Kiến trúc hệ thống sẽ được thiết kế theo mô hình ba lớp bao gồm:

- **Frontend (Giao diện người dùng)**:
  - Được xây dựng bằng React, giúp người dùng nhập các thông tin cần thiết cho việc dự đoán lưu lượng giao thông.
  - Sau khi người dùng nhập dữ liệu, hệ thống gửi một yêu cầu POST đến Backend với dữ liệu đã nhập.

- Sau khi nhận được kết quả dự đoán từ Backend, hệ thống hiển thị kết quả cho người dùng.

- **Backend (Máy chủ và API)**:
  - Sử dụng Flask để xây dựng các API, nơi người dùng gửi yêu cầu và nhận phản hồi.
  - API /predict nhận yêu cầu POST với các tham số như ngày, múi giờ, thời tiết và nhiệt độ. Backend tiến hành:
    - Mã hóa dữ liệu bằng ColumnTransformer để xử lý các giá trị phân loại.
    - Chuẩn hóa dữ liệu bằng StandardScaler.
    - Định dạng lại dữ liệu đầu vào để phù hợp với mô hình LSTM.
    - Dự đoán lưu lượng giao thông bằng mô hình LSTM.
    - Giải mã kết quả dự đoán và trả về nhãn cho người dùng.
  - Mô hình LSTM đã được huấn luyện trước và lưu trữ trong tệp .keras. Các bộ công cụ như joblib được sử dụng để tải các mô hình học máy và các đối tượng như LabelEncoder, ColumnTransformer, và StandardScaler.

- **Machine Learning Model (Mô hình học máy)**:
  - Mô hình LSTM đã được huấn luyện từ trước với dữ liệu lịch sử và sử dụng để dự đoán lưu lượng giao thông dựa trên các tham số đầu vào (ngày, múi giờ, thời tiết, và nhiệt độ).
  - Dữ liệu được xử lý và chuẩn hóa trước khi đưa vào mô hình LSTM để đảm bảo đầu vào tương thích với cấu trúc mạng

3.3 Quy trình hoạt động của hệ thống

Quy trình hoạt động của hệ thống có thể được mô tả như sau:

1. **Người dùng nhập dữ liệu:**
   - Giao diện người dùng frontend yêu cầu người dùng nhập các tham số như ngày, múi giờ, thời tiết và nhiệt độ sẽ tự động cập nhật theo ngày và giờ đã nhập.
   - Sau khi người dùng nhập dữ liệu, hệ thống gửi một yêu cầu POST đến API /predict trên server backend.

2. **Backend xử lý yêu cầu:**
    - Backend nhận yêu cầu và kiểm tra tính hợp lệ của dữ liệu.
    - Dữ liệu được mã hóa (dùng ColumnTransformer) và chuẩn hóa (dùng StandardScaler).
    - Dữ liệu được định dạng lại để phù hợp với mô hình LSTM.
    - Backend gọi mô hình LSTM để dự đoán lưu lượng giao thông dựa trên dữ liệu đã được xử lý.
3. **Dự đoán và trả kết quả:**
    - Mô hình LSTM trả về một dự đoán về lưu lượng giao thông.
    - Backend giải mã dự đoán và trả kết quả cho frontend.
4. **Frontend hiển thị kết quả:**
    - Sau khi nhận được kết quả từ Backend, giao diện người dùng hiển thị kết quả dự đoán lưu lượng giao thông cho người dùng.

3.4 Các thành phần hệ thống

Các thành phần trong hệ thống sẽ được chia thành các mô-đun chức năng rõ ràng, giúp quản lý và triển khai dễ dàng hơn:

1. **Frontend (React)**
    - React Components: Các thành phần giao diện người dùng như form nhập liệu, nút gửi yêu cầu, khu vực hiển thị kết quả dự đoán.
    - State Management: Sử dụng state để lưu trữ dữ liệu người dùng nhập và kết quả dự đoán.
    - HTTP Requests: Dùng axios hoặc fetch để gửi yêu cầu POST tới API backend và nhận kết quả.
2. **Backend (Flask)**
    - Flask Routes: Định nghĩa các route như /predict để xử lý yêu cầu và trả về kết quả dự đoán.
    - Machine Learning Integration: Sử dụng các mô hình LSTM, ColumnTransformer, StandardScaler, và LabelEncoder đã huấn luyện để xử lý dữ liệu và thực hiện dự đoán.

- CORS: Sử dụng Flask-CORS để cho phép frontend giao tiếp với backend mà không bị lỗi liên quan đến cùng nguồn (CORS).

3. **Machine Learning (LSTM Model)**
   - Mô hình LSTM: Mô hình học sâu dùng để dự đoán lưu lượng giao thông dựa trên các tham số đầu vào.
   - Preprocessing: Sử dụng ColumnTransformer để xử lý các đặc trưng phân loại và StandardScaler để chuẩn hóa dữ liệu.
   - Prediction: Sử dụng mô hình LSTM để dự đoán kết quả.

3.3 Use case

Use-case: Dự báo giao thông cho lộ trình cố định

Actor: Người dùng

Mô tả: Trong trường hợp này, người dùng nhập ngày và giờ xuất phát, hệ thống sẽ tự động lấy thông tin thời tiết, nhiệt độ tương ứng với thời điểm trên từ API của OpenWeatherMap, kết hợp 4 tham số để dự báo tình trạng giao thông cho lộ trình cố định từ điểm A đến điểm B.

Tiền điều kiện: Người dùng phải có kết nối internet và truy cập vào giao diện web của hệ thống.

Luồng hoạt động:

1. Người dùng nhập ngày và giờ xuất phát trên giao diện web.
2. Hệ thống tự động cập nhật thông tin thời tiết, nhiệt độ từ OpenWeatherMap API.
3. Lộ trình cố định từ điểm A đến điểm B đã được cấu hình sẵn trong hệ thống.
4. Hệ thống sử dụng Google Map API để lấy dữ liệu giao thông theo thời gian thực.
5. Dựa trên dữ liệu giao thông trong lịch sử, dữ liệu thời tiết, nhiệt độ và thông tin thời gian thời điểm xuất phát và mô hình dự báo Machine Learning (LSTM) hệ thống tính toán tình trạng giao thông
6. Kết quả dự báo bao gồm mức độ tắc nghẽn, và các khuyến nghị (nếu có) được hiển thị cho người dùng.
7. Hệ thống cung cấp chỉ dẫn đường đi chi tiết từ điểm A đến điểm B qua Google Maps API.

Kịch bản chính:

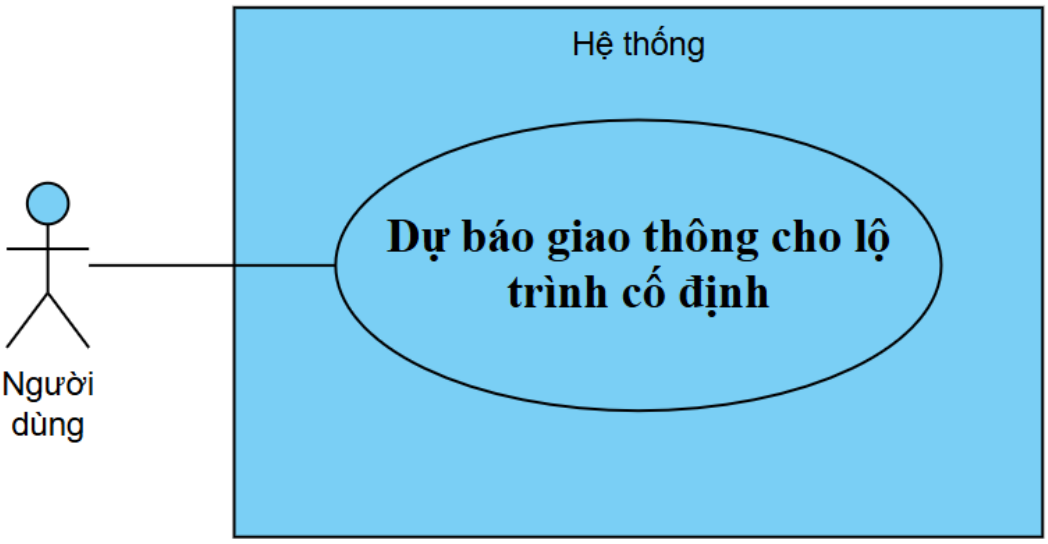
- Người dùng truy cập giao diện web và nhìn thấy ô nhập liệu yêu cầu chọn ngày và giờ xuất phát.
- Người dùng chọn ngày và giờ dự định xuất phát từ điểm A.
- Hệ thống tự động lấy thông tin thời tiết từ OpenWeatherMap API dựa trên thời gian người dùng đã chọn, bao gồm nhiệt độ, điều kiện thời tiết như mưa, nắng, gió, v.v.
- Hệ thống tự động xác định lộ trình cố định từ điểm A đến điểm B mà người dùng sẽ di chuyển.
- Hệ thống sử dụng Google Map API, MapQuest API, HERE Map API và dữ liệu thời tiết từ OpenWeatherMap API. để lấy thông tin theo thời gian thực cho lộ trình đã xác định.
- Mô hình Machine Learning (LSTM) sẽ nhận và sử dụng dữ liệu đầu vào để dự báo tình trạng giao thông tại thời gian xuất phát đã chọn.
- Sau khi tính toán, hệ thống hiển thị kết quả cho người dùng, bao gồm:
    - Mức độ tắc nghẽn trên lộ trình.
    - Các khuyến nghị như chọn thời điểm xuất phát khác.
- Nếu người dùng yêu cầu, hệ thống có thể cung cấp các chỉ dẫn chi tiết từ điểm A đến điểm B qua Google Maps API, bao gồm thông tin về tình trạng giao thông hiện tại.

Ngoại lệ:

- Nếu không thể kết nối với OpenWeatherMap API hoặc Google Map API, hệ thống sẽ thông báo lỗi và yêu cầu người dùng thử lại sau.
- Nếu dữ liệu từ các API không đầy đủ hoặc không hợp lệ, hệ thống có thể yêu cầu người dùng nhập lại thông tin hoặc hiển thị thông báo lỗi.

Kết quả mong đợi:

- Hệ thống dự báo chính xác tình trạng giao thông cho lộ trình cố định từ điểm A đến điểm B dựa trên ngày và giờ xuất phát của người dùng.
- Người dùng có thể đưa ra quyết định di chuyển hợp lý, tiết kiệm thời gian.



Hình 3.2 Sơ đồ usecase dự báo giao thông cho lộ trình cố định

3.4 Kết luận Chương III

Trong chương này, chúng ta đã trình bày chi tiết về thiết kế hệ thống dự báo giao thông sử dụng Machine Learning, bao gồm các thành phần cấu trúc của hệ thống, từ Frontend, Backend đến mô hình học máy LSTM. Hệ thống được xây dựng theo kiến trúc ba lớp, giúp các thành phần giao tiếp hiệu quả và dễ dàng mở rộng trong tương lai.

Frontend, được phát triển bằng React và Vite, đảm nhận việc tương tác trực tiếp với người dùng, thu thập thông tin cần thiết và hiển thị kết quả dự báo. Backend sử dụng Flask để cung cấp các API, nhận và xử lý yêu cầu từ người dùng, và trả về kết quả dự đoán lưu lượng giao thông. Mô hình LSTM đã được huấn luyện từ dữ liệu lịch sử giao thông và thời tiết để đưa ra dự đoán chính xác nhất.

Các thành phần của hệ thống được chia thành các mô-đun chức năng rõ ràng, giúp việc quản lý và triển khai trở nên dễ dàng và hiệu quả. Quy trình hoạt động của hệ thống từ việc thu thập dữ liệu, tiền xử lý, cho đến việc trả về kết quả dự báo đều được thực hiện một cách mượt mà và nhanh chóng, đáp ứng được nhu cầu của người dùng.

Một trong những tính năng quan trọng của hệ thống là khả năng dự báo giao thông cho các lộ trình cố định, giúp người dùng đưa ra quyết định di chuyển hợp lý. Các API như OpenWeatherMap và Google Maps được tích hợp để cung cấp thông tin thời tiết và giao thông thời gian thực, từ đó tạo ra một công cụ hữu ích cho người dùng trong việc lập kế hoạch di chuyển.

CHƯƠNG IV. HUẤN LUYỆN VÀ ĐÁNH GIÁ MÔ HÌNH DỰ BÁO

Chương này trình bày quá trình xử lý dữ liệu, huấn luyện và tối ưu hóa mô hình dự báo lưu lượng giao thông, bao gồm các bước từ thu thập dữ liệu đến đánh giá và cải thiện mô hình.

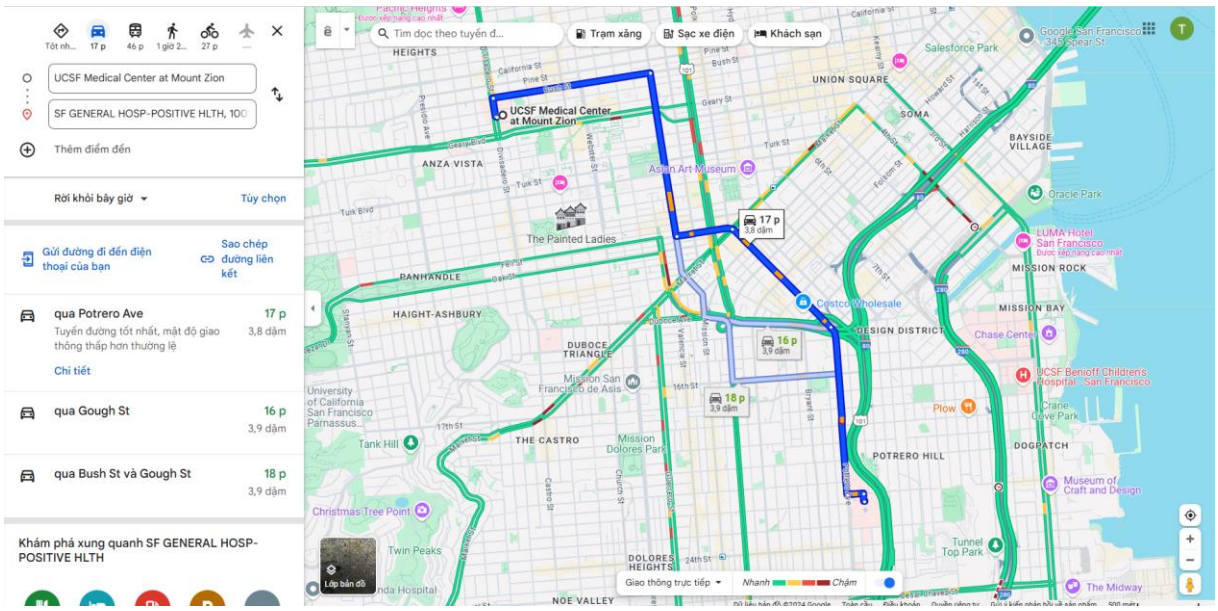
Tôi sẽ trình bày về mô hình dự đoán lưu lượng giao thông sử dụng mạng nơ-ron hồi tiếp dài ngắn (LSTM). Mô hình được xây dựng với mục tiêu dự đoán mức độ giao thông trong ngày dựa trên các yếu tố như thời gian trong ngày, ngày trong tuần, điều kiện thời tiết và nhiệt độ. Dữ liệu huấn luyện được thu thập trong năm 2017 và dữ liệu kiểm tra là 6 tháng đầu năm 2018.

Mô hình LSTM được chọn vì khả năng xử lý các chuỗi thời gian và dự đoán các yếu tố liên quan trong các bước tiếp theo của quá trình thời gian.

4.1 Thu thập và tiền xử lý dữ liệu

4.1.1 Thu thập dữ liệu

Dữ liệu được thu thập từ các nguồn như Google Map API và OpenWeatherMap API, bao gồm các tham số liên quan đến thời gian di chuyển và các yếu tố ảnh hưởng như thời tiết và nhiệt độ.



Hình 4.1 Dữ liệu dựa trên thời gian thực giữa 2 bệnh viện tại San Fracisco là UCSF Medical Center at Mount Zion và SF GENERAL HOSP-POSITIVE HLTH

Tập dữ liệu lịch sử được thu thập từ ngày 1 tháng 1 năm 2017 đến ngày 31 tháng 12 năm 2017. Các tham số chính trong dữ liệu bao gồm:

- Thời gian trong ngày (Zone): Một mã số đại diện cho khoảng thời gian 10 phút.
- Ngày trong tuần (CodedDay): Số mã hóa đại diện cho các ngày trong tuần.
- Điều kiện thời tiết (CodedWeather): Mã số đại diện cho điều kiện thời tiết.
- Nhiệt độ (Temperature): Nhiệt độ trung bình trong ngày.
- Mức độ giao thông (Traffic): Lưu lượng giao thông cần dự đoán trên đường đi

Tập huấn luyện

Tập huấn luyện (train_set.csv) chứa 52417 dòng dữ liệu trong năm 2017.

Date	Day	CodedDay	Zone	Weather	Temperature	Traffic
01-01-2017	Sunday	7	1	21	17	1
01-01-2017	Sunday	7	2	12	34	2
01-01-2017	Sunday	7	3	25	24	2
01-01-2017	Sunday	7	4	46	41	4
01-01-2017	Sunday	7	5	33	19	3

Bảng 4.1 Ví dụ về 5 dòng đầu tiên trong tập huấn luyện

Tập kiểm tra

Tập kiểm tra (test_set.csv) chứa 1440 dòng dữ liệu từ ngày 1 đến ngày 10 của tháng 6 năm 2018. Dưới đây là vài dòng đầu tiên trong tập kiểm tra:

Date	Day	CodedDay	Zone	Weather	Temperature	Traffic
01-06-2018	Wednesday	3	2	35	17	2
01-06-2018	Wednesday	3	3	36	16	3
01-06-2018	Wednesday	3	4	27	25	5
01-06-2018	Wednesday	3	5	23	23	3
01-06-2018	Wednesday	3	6	18	42	2

Bảng 4.2 Ví dụ về 5 dòng đầu tiên trong tập kiểm tra

Điều kiện thời tiết

Dữ liệu điều kiện thời tiết được thu thập từ OpenWeatherMap và sau đó được mã hóa lại bằng các mã định nghĩa riêng để thuận tiện cho việc sử dụng trong mô hình. Các mã thời tiết sau khi định nghĩa lại được mô tả như sau:

STT	Mã OpenWeatherMap	Mô tả (OpenWeatherMap)	Mã Định Nghĩa Lại	Mô tả Định Nghĩa Lại

1	200	Thunderstorm with light rain	4	Thunderstorm
2	201	Thunderstorm with rain	4	Thunderstorm
3	202	Thunderstorm with heavy rain	4	Thunderstorm
4	210	Light thunderstorm	4	Thunderstorm
5	211	Thunderstorm	4	Thunderstorm
6	212	Heavy thunderstorm	4	Thunderstorm
7	221	Ragged thunderstorm	4	Thunderstorm
8	230	Thunderstorm with light drizzle	4	Thunderstorm
9	231	Thunderstorm with drizzle	4	Thunderstorm
10	232	Thunderstorm with heavy drizzle	4	Thunderstorm
11	300	Light drizzle	9	Drizzle
12	301	Drizzle	9	Drizzle
13	302	Heavy drizzle	9	Drizzle
14	310	Light rain with drizzle	9	Drizzle
15	311	Rain with drizzle	9	Drizzle
16	312	Heavy rain with drizzle	9	Drizzle
17	500	Light rain	11	Light rain
18	501	Moderate rain	11	Moderate rain
19	502	Heavy rain	11	Heavy rain
20	503	Very heavy rain	11	Extreme rain
21	504	Extreme rain	11	Extreme rain
22	511	Freezing rain	10	Freezing rain
23	520	Light shower rain	11	Light shower rain
24	521	Shower rain	11	Shower rain
25	522	Heavy shower rain	11	Heavy shower rain
26	531	Ragged shower rain	11	Ragged shower rain
27	600	Light snow	16	Snow

28	601	Snow	16	Snow
29	602	Heavy snow	16	Heavy snow
30	611	Sleet	18	Sleet
31	612	Light shower sleet	18	Sleet
32	613	Shower sleet	18	Sleet
33	615	Light rain and snow	16	Snow
34	616	Rain and snow	16	Snow
35	620	Light shower snow	16	Snow
36	621	Shower snow	16	Snow
37	622	Heavy shower snow	16	Heavy snow
38	701	Mist	20	Foggy
39	711	Smoke	22	Smoky
40	721	Haze	21	Haze
41	731	Dust	19	Dust
42	741	Fog	20	Foggy
43	751	Sand	19	Dust
44	761	Dust in suspension	19	Dust
45	762	Volcanic ash	19	Dust
46	771	Squalls	3	Severe thunderstorms
47	781	Tornado	0	Tornado
48	800	Clear sky	31	Clear (night)
49	801	Few clouds	32	Sunny
50	802	Scattered clouds	34	Scattered clouds
51	803	Broken clouds	26	Cloudy
52	804	Overcast clouds	26	Cloudy
53	900	Tornado	0	Tornado
54	901	Tropical storm	1	Tropical storm
55	902	Hurricane	2	Hurricane
56	903	Cold	25	Cold
57	904	Hot	36	Hot
58	905	Windy	24	Windy
59	906	Hail	17	Hail
60	907	Blizzards	43	Heavy snow

61	908	Dust	19	Dust
62	909	Fog	20	Foggy
63	910	Sand	19	Dust
64	911	Tornado	0	Tornado
65	912	Sleet	18	Sleet
66	913	Duststorm	19	Dust
67	914	Hurricane	2	Hurricane
68	915	Tornado	0	Tornado
69	916	Thunderstorm	4	Thunderstorm
70	917	Light rain	11	Light rain
71	918	Heavy snow	41	Heavy snow
72	919	Snow flurries	13	Snow flurries
73	920	Showers	40	Showers
74	921	Snow showers	42	Snow showers
75	922	Storm	3	Severe thunderstorms

Bảng 4.3 Bảng ánh xạ mã số thời tiết

4.1.2. Tiền xử lý dữ liệu

Khi làm việc với mô hình học máy, chúng ta thường phân chia dữ liệu thành đặc trưng (features) và nhãn (labels). Các đặc trưng sẽ được sử dụng để huấn luyện mô hình, trong khi nhãn là kết quả mà mô hình cần dự đoán.

- Đặc trưng (X): Các thông tin như CodedDay, Zone, Weather, Temperature.
- Nhãn (y): Thông tin về lưu lượng giao thông (Traffic).

```
# Nhập bộ dữ liệu huấn luyện
train_dataset = pd.read_csv('/content/drive/My Drive/DATAX/train_set.csv')

# Lấy các đặc trưng và nhãn từ bộ dữ liệu huấn luyện
X_train = train_dataset.iloc[:, [2, 3, 4, 5]].values
y_train = train_dataset.iloc[:, 6].values

# Nhập bộ dữ liệu kiểm tra
test_dataset = pd.read_csv('/content/drive/My Drive/DATAX/test_set.csv')
X_test = test_dataset.iloc[:, [2, 3, 4, 5]].values
y_test = test_dataset.iloc[:, 6].values
```

- X_train và X_test chứa các đặc trưng từ các cột: CodedDay, Zone, Weather, Temperature.
- y_train và y_test chứa nhãn từ cột Traffic

Trong dữ liệu, có nhiều cột chứa các giá trị phân loại, như Weather (thời tiết) và Zone (múi giờ). Những dữ liệu phân loại này cần phải được mã hóa thành dạng số để mô hình học máy có thể hiểu được. Quá trình này được thực hiện bằng hai phương pháp chính:

Mã hóa nhãn (y_train, y_test)

Chúng ta sử dụng LabelEncoder để chuyển các nhãn (mức độ giao thông) từ dạng chuỗi thành dạng số.

Ví dụ, nếu nhãn Traffic có các giá trị như 1, 2, 3, 4, 5 chúng ta chuyển chúng thành các giá trị số tương ứng.

```
# Mã hóa dữ liệu phân loại (categorical data) bằng LabelEncoder
labelencoder_Y = LabelEncoder()
y_train = labelencoder_Y.fit_transform(y_train)  # Chuyển y_train thành dạng số
y_test = labelencoder_Y.transform(y_test)  # Chuyển y_test thành dạng số
```

Mã hóa các cột phân loại (X_train, X_test)

Các cột Zone, Weather, Temperature, và CodedDay là các đặc trưng phân loại, cần mã hóa bằng **One-Hot Encoding**. Quá trình này tạo ra một vector nhị phân cho mỗi giá trị trong cột. Cột này sẽ được mã hóa thông qua OneHotEncoder từ thư viện sklearn.

```
# Mã hóa các cột phân loại
ct = ColumnTransformer(
    transformers=[('encoder', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), [0, 1, 2, 3])],
    remainder='passthrough'
)  # Mã hóa các cột phân loại
```

- OneHotEncoder chuyển đổi các cột phân loại thành các vector nhị phân.
- handle_unknown='ignore' đảm bảo rằng nếu có giá trị chưa từng xuất hiện trong tập huấn luyện, chúng sẽ được bỏ qua.

Các đặc trưng trong dữ liệu có thể có các phạm vi giá trị khác nhau, ví dụ như Temperature có thể dao động từ -10 đến 40 độ, trong khi Zone có thể chỉ là các giá trị 1, 2, 3. Để mô hình học máy hoạt động hiệu quả hơn, chúng ta cần chuẩn hóa các giá trị này sao cho chúng có cùng phạm vi.

```
# Chuẩn hóa các đặc trưng để có phạm vi giá trị đồng nhất
sc = StandardScaler()
X_train = sc.fit_transform(X_train)  # Chuẩn hóa X_train
X_test = sc.transform(X_test)  # Chuẩn hóa X_test
```

StandardScaler chuẩn hóa các giá trị sao cho trung bình là 0 và độ lệch chuẩn là 1. Điều này giúp mô hình học nhanh hơn và chính xác hơn.

Để mô hình phân loại có thể xử lý các nhãn phân loại đa lớp, chúng ta chuyển nhãn thành dạng one-hot encoding. Mỗi nhãn sẽ được chuyển thành một vector nhị phân với một giá trị 1 tại vị trí của lớp tương ứng.

```
# Chuyển đổi y_train và y_test thành mã hóa one-hot
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Mô hình LSTM yêu cầu dữ liệu đầu vào có dạng 3D: (samples, timesteps, features). Do đó, chúng ta cần **định dạng lại dữ liệu** sao cho phù hợp với yêu cầu của mô hình LSTM.

```
# Định dạng lại dữ liệu để sử dụng với LSTM
X_train_lstm = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))  # Định dạng lại X_train cho LSTM
X_test_lstm = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))  # Định dạng lại X_test cho LSTM
```

- Samples: Số lượng dòng trong dữ liệu.
- Timesteps: Số lượng bước thời gian (ở đây là 1 vì mỗi mẫu là một điểm dữ liệu duy nhất).
- Features: Số lượng đặc trưng của mỗi mẫu (số cột trong X_train)

4.2 Huấn luyện mô hình

Google Colab cung cấp môi trường Python miễn phí với GPU/TPU, rất phù hợp cho việc huấn luyện các mô hình học sâu như LSTM. Sử dụng Google Colab để huấn luyện mô hình LSTM và xuất mô hình.

Tải dữ liệu

Tải bộ dữ liệu huấn luyện và kiểm tra từ các tệp CSV đã lưu trên Google Drive. Import các thư viện cần thiết để xây dựng, huấn luyện mô hình

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Nhập bộ dữ liệu huấn luyện
train_dataset = pd.read_csv('/content/drive/My Drive/DATAX/train_set.csv')

# Lấy các đặc trưng và nhãn từ bộ dữ liệu huấn luyện
X_train = train_dataset.iloc[:, [2, 3, 4, 5]].values
y_train = train_dataset.iloc[:, 6].values

# Nhập bộ dữ liệu kiểm tra
test_dataset = pd.read_csv('/content/drive/My Drive/DATAX/test_set.csv')
X_test = test_dataset.iloc[:, [2, 3, 4, 5]].values
y_test = test_dataset.iloc[:, 6].values
```

Khởi tạo mô hình

Sử dụng mô hình **Bidirectional LSTM** vì nó giúp mô hình học được thông tin từ cả phía trước và phía sau của chuỗi dữ liệu, làm tăng khả năng dự đoán.

```
# Khởi tạo mô hình LSTM
lstm_model = Sequential()

# Thêm lớp Bi-directional LSTM với 100 đơn vị
lstm_model.add(Bidirectional(LSTM(units=100, return_sequences=True), input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
lstm_model.add(Dropout(0.3))  # Dropout 30%

lstm_model.add(Bidirectional(LSTM(units=100, return_sequences=False)))
lstm_model.add(Dropout(0.3))  # Dropout 30%

lstm_model.add(Dense(units=5, activation='softmax'))  # Lớp đầu ra sử dụng softmax cho phân loại nhiều lớp
```

Biên dịch mô hình

Sau khi xây dựng xong, biên dịch mô hình bằng cách sử dụng hàm mất mát categorical_crossentropy và tối ưu hóa bằng Adam.

```
# Biên dịch mô hình
optimizer = Adam(learning_rate=0.001)
lstm_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

Thiết lập callbacks

Để cải thiện quá trình huấn luyện, thiết lập EarlyStopping và ReduceLROnPlateau để dừng huấn luyện nếu mô hình không cải thiện và giảm learning rate nếu cần.

```
# Thiết lập callbacks để dừng sớm và giảm learning rate khi không có sự cải thiện
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.0001)
```

Huấn luyện mô hình

Huấn luyện mô hình với dữ liệu huấn luyện (X_train_lstm, y_train) và dữ liệu kiểm tra (X_test_lstm, y_test) để theo dõi quá trình học.

```
# Huấn luyện mô hình
history = lstm_model.fit(X_train_lstm, y_train, epochs=50, batch_size=32, validation_data=(X_test_lstm, y_test), callbacks=[early_stopping, lr_scheduler])
```

Lưu mô hình và các đối tượng tiền xử lý

Lưu mô hình đã huấn luyện và các đối tượng tiền xử lý vào các tệp để có thể tái sử dụng sau này.

Lưu mô hình để sử dụng tại backend

```
[ ]  # Đảm bảo rằng thư mục DATAX đã tồn tại trên Google Drive
     model_path = '/content/drive/My Drive/DATAX/lstm_model1.keras'

     # Lưu mô hình vào thư mục
     lstm_model.save(model_path)
```

Lưu các khâu tiền xử lý dữ liệu

```
[ ]  import joblib

     # Lưu LabelEncoder (y_train và y_test đã được mã hóa)
     joblib.dump(labelencoder_Y, '/content/drive/My Drive/DATAX/labelencoder_Y1.pkl')

     # Lưu ColumnTransformer (đã mã hóa các đặc trưng phân loại)
     joblib.dump(ct, '/content/drive/My Drive/DATAX/column_transformer1.pkl')

     # Lưu StandardScaler (đã chuẩn hóa các đặc trưng)
     joblib.dump(sc, '/content/drive/My Drive/DATAX/scaler1.pkl')

     print("Các đối tượng tiền xử lý đã được lưu thành công!")
```

4.3 Đánh giá mô hình

Đánh giá mô hình trên bộ dữ liệu kiểm tra để biết được độ chính xác và loss khi mô hình dự đoán dữ liệu chưa thấy.

```
Epoch 45/50
1638/1638 ━━━━━━━━━━━━━━━━━━━━ 40s 13ms/step - accuracy: 0.8121 - loss: 0.5066 - val_accuracy: 0.9854 - val_loss: 0.1371 - learning_rate: 0.0010
Epoch 46/50
1638/1638 ━━━━━━━━━━━━━━━━━━━━ 41s 13ms/step - accuracy: 0.8168 - loss: 0.4978 - val_accuracy: 0.9854 - val_loss: 0.1297 - learning_rate: 0.0010
Epoch 47/50
1638/1638 ━━━━━━━━━━━━━━━━━━━━ 40s 13ms/step - accuracy: 0.8170 - loss: 0.4907 - val_accuracy: 0.9840 - val_loss: 0.1287 - learning_rate: 0.0010
Epoch 48/50
1638/1638 ━━━━━━━━━━━━━━━━━━━━ 23s 14ms/step - accuracy: 0.8200 - loss: 0.4849 - val_accuracy: 0.9847 - val_loss: 0.1221 - learning_rate: 0.0010
Epoch 49/50
1638/1638 ━━━━━━━━━━━━━━━━━━━━ 39s 13ms/step - accuracy: 0.8164 - loss: 0.4898 - val_accuracy: 0.9854 - val_loss: 0.1206 - learning_rate: 0.0010
Epoch 50/50
1638/1638 ━━━━━━━━━━━━━━━━━━━━ 41s 14ms/step - accuracy: 0.8215 - loss: 0.4827 - val_accuracy: 0.9868 - val_loss: 0.1200 - learning_rate: 0.0010
```

Hình 4.2 Kết quả chạy mô hình sau 50 epochs

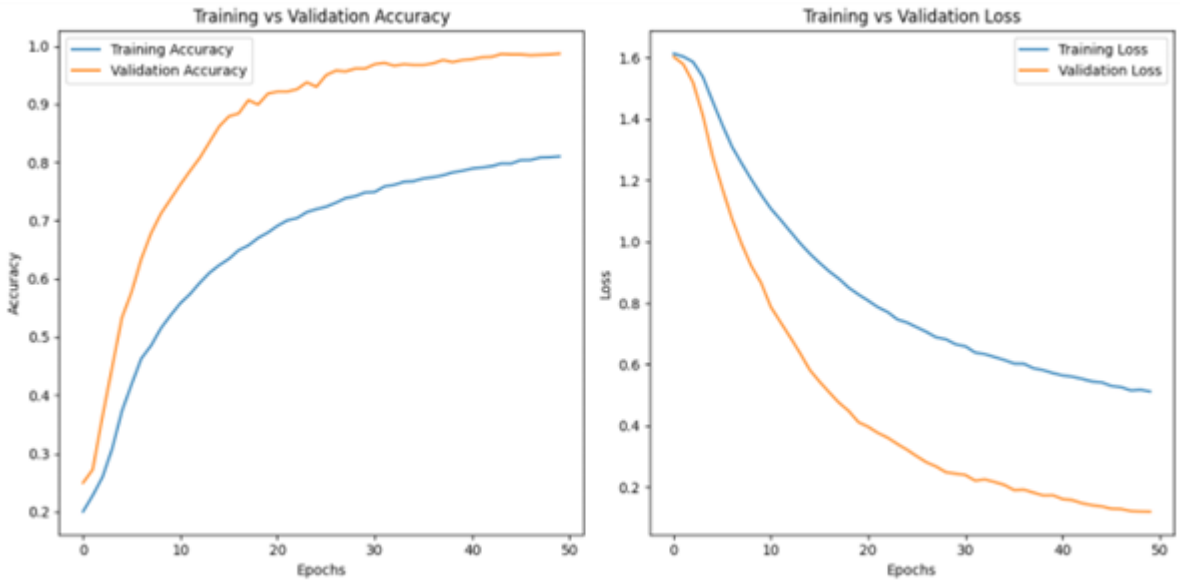
Sau khi huấn luyện xong, vẽ đồ thị để theo dõi độ chính xác và loss trong suốt quá trình huấn luyện và kiểm tra.

```
# Vẽ đồ thị độ chính xác và loss
plt.figure(figsize=(12, 6))

# Đồ thị độ chính xác
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Đồ thị loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Hình 4.3 Đồ thị độ chính xác và loss

Kết quả huấn luyện mô hình

- Độ chính xác (Accuracy): Độ chính xác trên tập huấn luyện và tập kiểm tra đã cải thiện đáng kể qua các epoch. Ban đầu, độ chính xác chỉ khoảng 20% ở Epoch 1, nhưng đến Epoch 40+, độ chính xác đã đạt trên 79%. Đây là một sự cải thiện rõ rệt, chứng tỏ rằng mô hình đang học tốt dần qua mỗi epoch.
- Độ mất mát (Loss): Mức độ mất mát (loss) giảm mạnh từ 1.6140 ở Epoch 1 xuống còn 0.6510 ở Epoch 28, cho thấy mô hình đang học tốt hơn qua từng epoch. Điều này cho thấy quá trình huấn luyện đang đi đúng hướng và mô hình đang tối ưu hóa hiệu quả.
- Val_loss (Mất mát trên Tập Kiểm Tra): Mất mát trên tập kiểm tra (val_loss) giảm đều đặn từ 1.6033 ở Epoch 1 xuống còn 0.2670 ở Epoch 28. Sự giảm dần này, cùng với việc val_loss sát với training loss, chứng minh rằng mô hình không gặp phải vấn đề overfitting nghiêm trọng.

Đánh giá với các chỉ số khác

- Precision, Recall và F1-score: Đây là những chỉ số quan trọng để đánh giá mô hình phân loại, đặc biệt khi dữ liệu không đồng đều (imbalanced). Sử dụng thư viện sklearn để tính các chỉ số này.

```
from sklearn.metrics import classification_report
y_pred = lstm_model.predict(X_test_lstm)
y_pred = np.argmax(y_pred, axis=1)  # Chuyển dự đoán về dạng chỉ số
y_test_actual = np.argmax(y_test, axis=1)  # Chuyển y_test về dạng chỉ số
print(classification_report(y_test_actual, y_pred))
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       280
           1       0.99      0.99      0.99       294
           2       0.99      0.98      0.99       277
           3       0.98      0.99      0.99       276
           4       0.98      0.99      0.99       312

    accuracy                           0.99      1439
   macro avg       0.99      0.99      0.99      1439
weighted avg       0.99      0.99      0.99      1439
```

Hình 3.1 Đánh giá mô hình với các chỉ số quan trọng

Nhận xét : Kết quả đánh giá mô hình rất ấn tượng! Mô hình có độ chính xác cao với accuracy = 0.99 và các chỉ số precision, recall, và f1-score đều đạt mức cao gần như hoàn hảo cho tất cả các lớp. Đây là một dấu hiệu cho thấy mô hình đã học được tốt và có khả năng phân loại chính xác các lớp.

- Tính ma trận nhầm lẫn

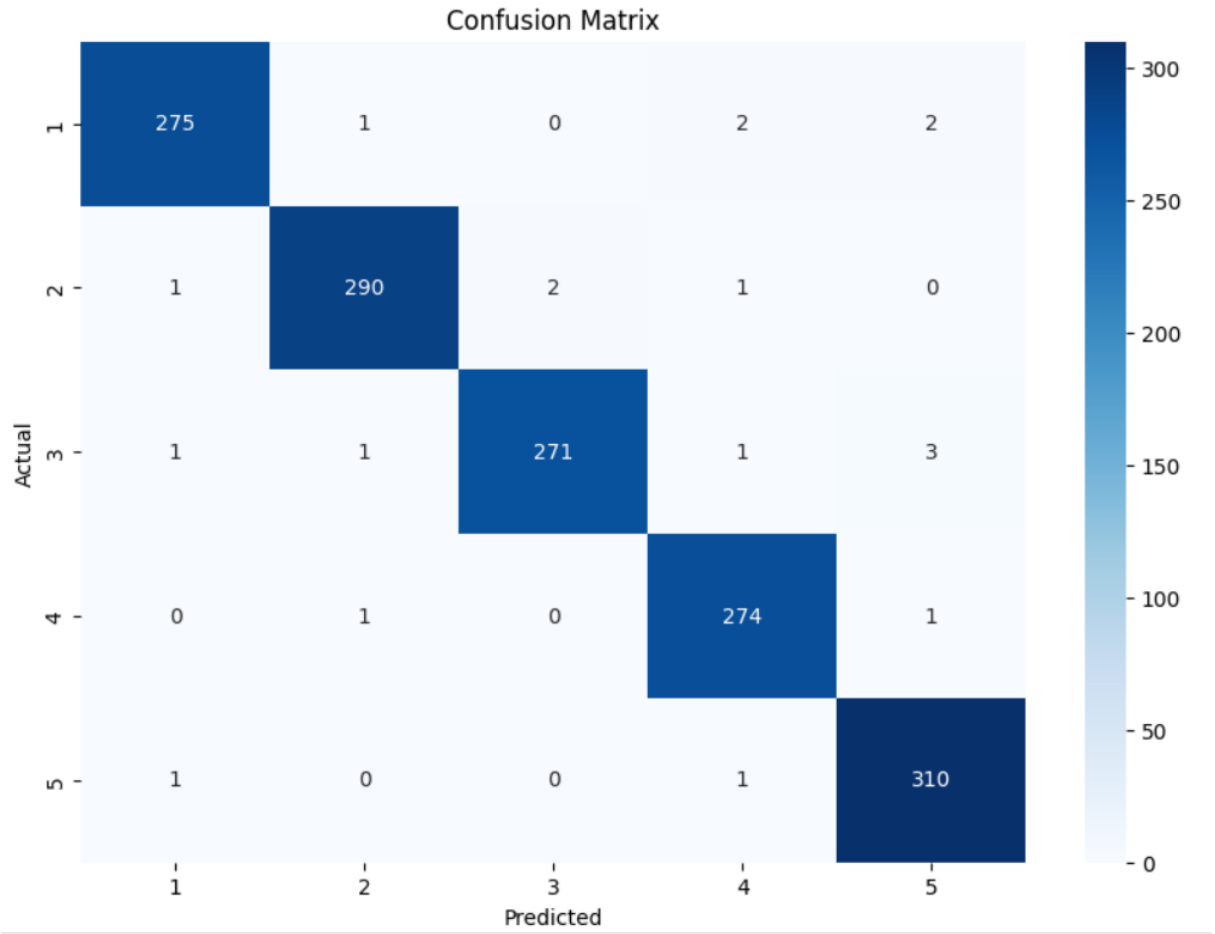
```
from sklearn.metrics import confusion_matrix
import numpy as np

# Dự đoán và kết quả thực tế
y_pred = lstm_model.predict(X_test_lstm)
y_pred = np.argmax(y_pred, axis=1)  # Dự đoán lớp
y_test_actual = np.argmax(y_test, axis=1)  # Kết quả thực tế

# Tính ma trận nhầm lẫn
cm = confusion_matrix(y_test_actual, y_pred)

# In ra ma trận nhầm lẫn dưới dạng văn bản
labels = labelencoder_Y.classes_  # Lớp của dữ liệu
print("Confusion Matrix:")
for i in range(len(labels)):
    row = [str(cm[i][j]) for j in range(len(labels))]
    print(f"{labels[i]}: " + "\t".join(row))
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Confusion Matrix:
1: 275  1       0       2       2
2: 1    290     2       1       0
3: 1    1       271     1       3
4: 0    1       0       274     1
5: 1    0       0       1       310
```



Hình 4.4 Tính ma trận nhầm lẫn

Phân tích ma trận nhầm lẫn

- Lớp 1 (Row 1): Dự đoán khá chính xác, với 275 mẫu đúng, chỉ có 1 mẫu nhầm thành lớp 2, và rất ít nhầm lẫn với các lớp khác.
- Lớp 2 (Row 2): 290 mẫu đúng, nhưng có một số nhầm lẫn với:
    - Lớp 1: 1 mẫu
    - Lớp 3: 2 mẫu

      Nhìn chung, mô hình rất ít nhầm lẫn với các lớp còn lại.
- Lớp 3 (Row 3): Tương tự như lớp 2, với 271 mẫu đúng, nhưng vẫn có một chút nhầm lẫn với các lớp khác:
    - Lớp 1 và Lớp 2: Một số mẫu bị nhầm
    - Lớp 5: Một số mẫu cũng bị nhầm
- Lớp 4 (Row 4): Hầu như không có nhầm lẫn với Lớp 1 (0 mẫu), nhưng vẫn có một số nhầm lẫn nhỏ với:

    - Lớp 2: 1 mẫu
    - Lớp 5: 1 mẫu
- Lớp 5 (Row 5): 310 mẫu đúng, rất ít nhầm lẫn với các lớp khác.

Nhận xét chung: Tổng thể, ma trận nhầm lẫn cho thấy mô hình hoạt động khá tốt, với số lượng mẫu đúng rất cao ở tất cả các lớp. Việc nhầm lẫn chủ yếu xảy ra giữa các lớp gần nhau (như lớp 1 và lớp 2) hoặc các lớp có đặc điểm tương đồng. Tuy nhiên, các lỗi nhầm lẫn vẫn ở mức thấp, chứng tỏ rằng mô hình có khả năng phân loại chính xác trong đa số trường hợp.

4.4 Tối ưu hóa mô hình

Các thay đổi đã thực hiện trong code để cải thiện độ chính xác mô hình:

```
# Thêm lớp Bi-directional LSTM với 100 đơn vị
lstm_model.add(Bidirectional(LSTM(units=100, return_sequences=True), input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
lstm_model.add(Dropout(0.3))  # Dropout 30%

lstm_model.add(Bidirectional(LSTM(units=100, return_sequences=False)))
lstm_model.add(Dropout(0.3))  # Dropout 30%

lstm_model.add(Dense(units=5, activation='softmax'))  # Lớp đầu ra sử dụng softmax cho phân loại nhiều lớp

# Biên dịch mô hình
optimizer = Adam(learning_rate=0.001)
lstm_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Thiết lập callbacks để dừng sớm và giảm learning rate khi không có sự cải thiện
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.0001)
```

- Bi-directional LSTM:
    - Đã thêm lớp Bidirectional với 100 đơn vị để mô hình có thể học từ cả hai chiều của dữ liệu chuỗi, giúp cải thiện khả năng nhận diện các mẫu dữ liệu phức tạp.
- Dropout:
    - Tăng tỷ lệ dropout từ 20% lên 30% để giảm nguy cơ overfitting, giúp mô hình học được các đặc trưng mạnh mẽ hơn từ dữ liệu mà không bị quá khớp.
- Callbacks:
    - EarlyStopping:
        - Dừng quá trình huấn luyện nếu không có sự cải thiện về val_loss trong 10 epoch liên tiếp và phục hồi trọng số tốt nhất để tránh huấn luyện quá mức và tiết kiệm thời gian.
    - ReduceLROnPlateau:

- Giảm learning rate nếu val_loss không cải thiện trong 5 epoch liên tiếp, giúp mô hình học tốt hơn trong các bước cuối của quá trình huấn luyện.

4.5 Kiểm thử đánh giá mô hình trên tập kiểm tra

Tiến hành kiểm thử mô hình LSTM đã huấn luyện trên bộ dữ liệu kiểm tra (test_set.csv).

1. Tải lại các đối tượng tiền xử lý và mô hình đã lưu:
   - Sử dụng joblib.load() để tải lại ColumnTransformer, Scaler và LabelEncoder từ các tệp đã lưu.
   - Tải lại mô hình LSTM đã huấn luyện từ tệp .keras bằng load_model().

```
# Tải lại các đối tượng tiền xử lý và mô hình đã lưu
ct = joblib.load('/content/drive/My Drive/DATAX/column_transformer1.pkl')
sc = joblib.load('/content/drive/My Drive/DATAX/scaler1.pkl')
labelencoder_Y = joblib.load('/content/drive/My Drive/DATAX/labelencoder_Y1.pkl')
lstm_model = load_model('/content/drive/My Drive/DATAX/lstm_model1.keras')
```

2. Chuẩn bị dữ liệu kiểm tra:
   - Đọc dữ liệu từ tệp CSV /content/drive/My Drive/DATAX/test_set.csv và chia thành các biến đầu vào (X_test) và đầu ra (y_test).
   - Áp dụng ColumnTransformer và Scaler đã tải lại để tiền xử lý và chuẩn hóa dữ liệu đầu vào (X_test).
   - Mã hóa nhãn đầu ra (y_test) thành dạng one-hot bằng LabelEncoder.

```
# Nhập bộ dữ liệu kiểm tra
test_dataset = pd.read_csv('/content/drive/My Drive/DATAX/test_set.csv')
X_test = test_dataset.iloc[:, [2, 3, 4, 5]].values
y_test = test_dataset.iloc[:, 6].values

# Mã hóa các cột phân loại
X_test = np.array(ct.transform(X_test))

# Chuẩn hóa các đặc trưng
X_test = sc.transform(X_test)

# Chuyển đổi y_test thành mã hóa one-hot
y_test = to_categorical(labelencoder_Y.transform(y_test))
```

3. Định dạng lại dữ liệu cho LSTM: Định dạng lại dữ liệu đầu vào (X_test) thành dạng (samples, timesteps, features) phù hợp cho mô hình LSTM.

```
# Định dạng lại dữ liệu để sử dụng với LSTM
X_test_lstm = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

4. Đánh giá mô hình trên dữ liệu kiểm tra: Sử dụng lstm_model.evaluate() để tính toán test_loss và test_accuracy trên dữ liệu kiểm tra (X_test_lstm, y_test).

```
# Đánh giá mô hình trên dữ liệu kiểm tra
test_loss, test_accuracy = lstm_model.evaluate(X_test_lstm, y_test)
print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')
```

Đo lường hiệu suất của mô hình trên dữ liệu kiểm tra thông qua hai chỉ số quan trọng: loss (mất mát) và accuracy (độ chính xác).

- test_loss cho biết mức độ sai lệch giữa dự đoán và giá trị thực
- test_accuracy cho biết tỷ lệ dự đoán đúng của mô hình trên toàn bộ bộ dữ liệu kiểm tra.

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9865 - loss: 0.0492
Test Loss: 0.04899534210562706, Test Accuracy: 0.988186240196228
```

5. Dự đoán với mô hình đã huấn luyện:
   - Dự đoán đầu ra (y_pred) từ mô hình LSTM sử dụng lstm_model.predict().
   - Chuyển đổi kết quả dự đoán từ dạng one-hot thành các lớp ban đầu bằng cách sử dụng np.argmax().
   - So sánh và in ra các nhãn thực tế và dự đoán cho một số mẫu (10 mẫu đầu tiên).

```
# Dự đoán với mô hình đã huấn luyện
y_pred = lstm_model.predict(X_test_lstm)
```

Kết quả : Các nhãn thực tế (True label) và nhãn dự đoán (Predicted) được in ra, cho thấy mô hình dự đoán rất chính xác (tất cả các nhãn dự đoán trùng khớp với nhãn thực tế).

Test Loss: 0.04899534210562706

Test Accuracy: 0.988186240196228

Cho thấy mô hình đang hoạt động rất tốt, với độ chính xác lên đến 98.8% trên bộ dữ liệu kiểm tra.

```
True label: 2, Predicted: 2
True label: 3, Predicted: 3
True label: 5, Predicted: 5
True label: 3, Predicted: 3
True label: 2, Predicted: 2
True label: 2, Predicted: 2
True label: 4, Predicted: 4
True label: 5, Predicted: 5
True label: 4, Predicted: 4
True label: 5, Predicted: 5
```

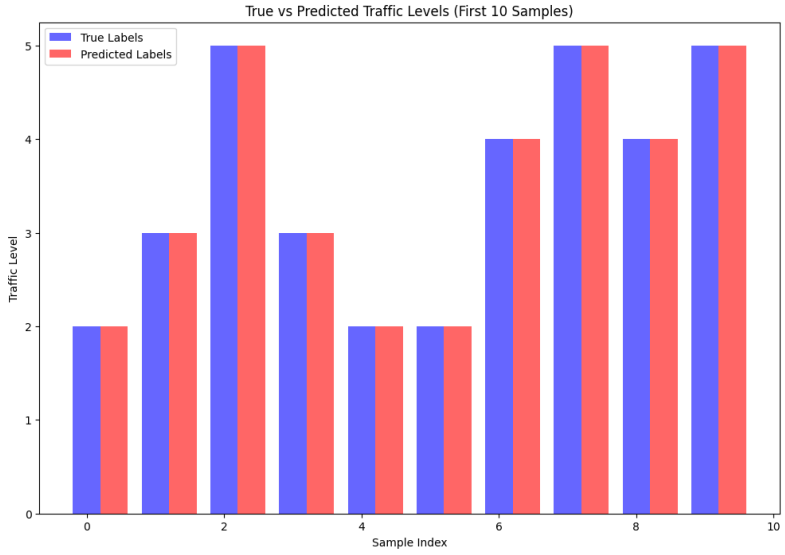
Hình 4.5 Kết quả của 10 mẫu đầu tiên

Các kết quả cho thấy nhãn thực tế và nhãn dự đoán gần như khớp nhau, ví dụ:

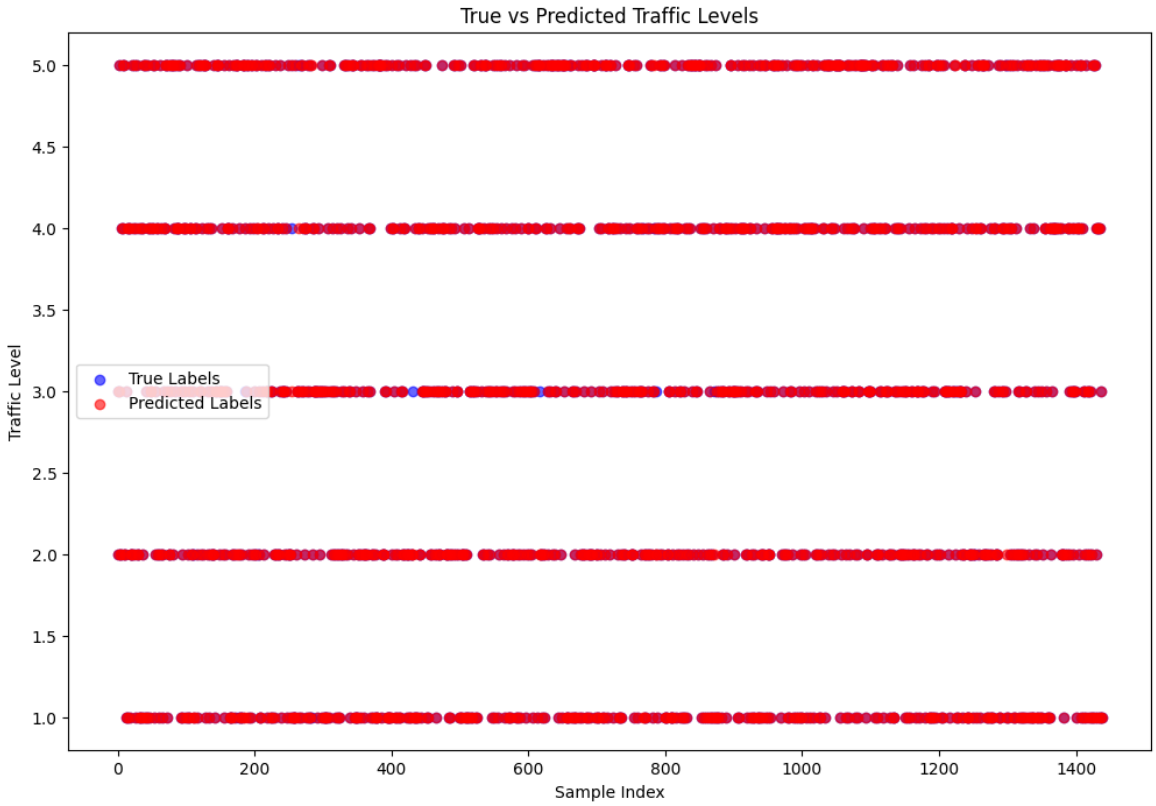
- True label: 2, Predicted: 2
- True label: 3, Predicted: 3
- True label: 5, Predicted: 5

Điều này cho thấy mô hình đã học rất tốt và dự đoán chính xác được mức độ giao thông (Traffic Level).

Vẽ biểu đồ cột tương ứng:



Hình 4.6 Biểu đồ cột hiển thị sự so sánh giữa giá trị thực tế và giá trị dự đoán cho mỗi mẫu trong bộ kiểm tra.



Hình 4.7 Biểu đồ scatter (scatter plot) để so sánh giữa nhãn thực tế (y_test_classes) và nhãn dự đoán (y_pred_classes) của mô hình

Quan sát biểu đồ, có thể thấy rằng phần lớn các điểm đỏ (dự đoán) và xanh (thực tế) trùng khớp hoặc khá gần nhau, cho thấy mô hình đã dự đoán đúng trong phần lớn trường hợp. Tuy nhiên, vẫn có một số sự lệch nhẹ giữa các điểm màu xanh và đỏ, điều này cho thấy rằng mô hình không phải lúc nào cũng chính xác hoàn toàn đối với mọi mẫu.

Lệch giữa các màu có thể chỉ ra rằng mô hình có thể dự đoán đúng một số nhãn, nhưng vẫn gặp phải những sai sót nhỏ trong việc phân loại chính xác từng mẫu. Mặc dù có sự lệch nhỏ này, nhưng mô hình vẫn đạt hiệu suất khá tốt và không có dấu hiệu của sự sai lệch nghiêm trọng.

Biểu đồ cho thấy mô hình có hiệu suất dự đoán khá tốt, nhưng vẫn có thể cải thiện hơn nữa trong việc phân loại chính xác tất cả các mẫu.

4.6 Kết luận Chương IV

Chương IV đã trình bày chi tiết quá trình huấn luyện và đánh giá mô hình dự báo lưu lượng giao thông sử dụng mạng nơ-ron hồi tiếp dài ngắn (LSTM). Quá trình này bao gồm các bước thu thập và tiền xử lý dữ liệu, huấn luyện mô hình, và đánh giá kết quả.

1. Thu thập và tiền xử lý dữ liệu: Dữ liệu được thu thập từ các nguồn như Google Map API và OpenWeatherMap API, bao gồm các yếu tố thời gian, thời tiết và nhiệt độ, cùng với mức độ giao thông. Quá trình tiền xử lý bao gồm mã hóa các giá trị phân loại và chuẩn hóa dữ liệu để mô hình học máy có thể hiểu và xử lý hiệu quả.
2. Huấn luyện mô hình: Mô hình LSTM được huấn luyện trên bộ dữ liệu huấn luyện (năm 2017) và kiểm tra trên bộ dữ liệu kiểm tra (6 tháng đầu năm 2018). Quá trình huấn luyện đã sử dụng Google Colab với các thư viện và công cụ như Keras và TensorFlow. Mô hình sử dụng kỹ thuật Bidirectional LSTM để học thông tin từ cả hai hướng thời gian, kết hợp với các kỹ thuật cải thiện quá trình học như EarlyStopping và ReduceLROnPlateau.
3. Đánh giá mô hình: Kết quả huấn luyện cho thấy mô hình cải thiện đáng kể qua các epoch, với độ chính xác đạt 79% sau hơn 40 epoch. Mô hình cho thấy khả năng học và dự đoán lưu lượng giao thông chính xác, mặc dù vẫn còn tiềm năng cải tiến trong các giai đoạn tiếp theo.

CHƯƠNG V. PHÁT TRIỂN VÀ TRIỂN KHAI ỨNG DỤNG WEB

Mục tiêu của ứng dụng web là cung cấp dự đoán lưu lượng giao thông dựa trên các yếu tố như ngày, múi giờ, nhiệt độ và điều kiện thời tiết. Ứng dụng này sẽ giúp người dùng có thể tra cứu và dự báo tình hình giao thông trong thời gian thực, hỗ trợ các quyết định đi lại.

5.1 Giới thiệu

Ứng dụng web này được phát triển để dự đoán giao thông dựa trên các yếu tố như ngày, khu vực, điều kiện thời tiết và nhiệt độ. Phần mềm này bao gồm hai phần chính: frontend (giao diện người dùng) và backend (máy chủ xử lý logic và dữ liệu). Phần frontend cung cấp giao diện người dùng đơn giản, cho phép người dùng nhập các tham số cần thiết để nhận dự đoán, trong khi backend chịu trách nhiệm xử lý dự đoán bằng cách sử dụng mô hình học máy LSTM đã huấn luyện.

Chương này mô tả chi tiết quá trình phát triển và triển khai ứng dụng web dự báo giao thông với các bước cụ thể:

1. Phát triển giao diện người dùng (frontend) bằng React.js, tận dụng Google Maps API để hiển thị dữ liệu bản đồ và thông tin giao thông.
2. Xây dựng backend sử dụng Flask, tích hợp các API từ Google Maps và mô hình Machine Learning. Backend sử dụng TensorFlow và Keras để tải và triển khai mô hình học máy LSTM đã huấn luyện. Flask cũng sử dụng CORS để cho phép frontend gửi yêu cầu từ các nguồn khác nhau.
3. Tích hợp mô hình dự đoán giao thông dựa trên dữ liệu thực và huấn luyện.
4. Tối ưu hóa và triển khai ứng dụng, đảm bảo hiệu suất và khả năng mở rộng cho người dùng thực tế.

5.2 Phát triển Frontend

Frontend của ứng dụng được phát triển sử dụng React và Vite để tạo giao diện người dùng. Giao diện người dùng bao gồm các trường nhập liệu để người dùng cung cấp dữ liệu cần thiết, và sau đó gửi yêu cầu tới backend để nhận dự đoán.

Khả năng của ứng dụng bao gồm:

- Hiển thị bản đồ Google Maps với các tính năng như TrafficLayer và Polyline.
- Lấy dữ liệu thời tiết từ OpenWeatherMap API dựa trên ngày được chọn.

- Dự đoán tình trạng giao thông dựa trên thời gian, ngày, và các yếu tố thời tiết.
- Cung cấp các chỉ số giao thông giúp người dùng quyết định thời gian và lộ trình di chuyển.

Các thành phần và công nghệ sử dụng:

- React: Được sử dụng để xây dựng giao diện người dùng, bao gồm việc quản lý trạng thái với các hook như useState và useEffect.
- Google Maps API: Được sử dụng để hiển thị bản đồ và các lớp giao thông (TrafficLayer), các tuyến đường (Polyline) và các thông tin liên quan đến giao thông.
- OpenWeatherMap API: Dùng để lấy dữ liệu thời tiết cho ngày hiện tại hoặc dự báo cho ngày tương lai.
- Axios: Thư viện dùng để gửi các yêu cầu HTTP để lấy dữ liệu từ các API và gửi dữ liệu đến backend để dự đoán giao thông.

```
import React, { useState, useEffect } from "react"; // Import các hook
useState và useEffect
import axios from "axios"; // Import thư viện axios để gửi yêu cầu HTTP
import DatePicker from "react-datepicker"; // Import thư viện DatePicker để
chọn ngày và giờ
import "react-datepicker/dist/react-datepicker.css"; // Import stylesheet cho
DatePicker
import { GoogleMap, LoadScript, TrafficLayer, Polyline, Marker, InfoWindow }
from "@react-google-maps/api"; // Import các component từ thư viện Google Maps
API
```

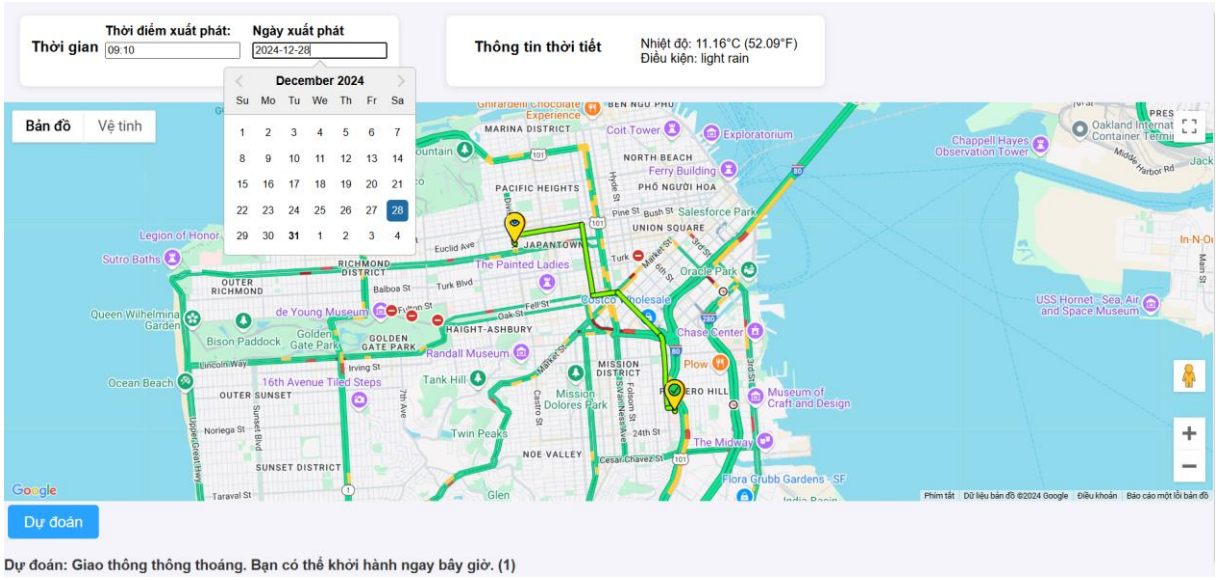
Xây dựng giao diện người dùng:

- Form lựa chọn ngày, giờ và địa điểm: Người dùng có thể chọn ngày, giờ
- Bản đồ Google Maps: Hiển thị bản đồ với các lớp giao thông, và cung cấp các thông tin lộ trình từ Google Directions API.
- Dự đoán giao thông: Kết quả dự đoán giao thông sẽ được hiển thị sau khi người dùng gửi yêu cầu.

```
{/* Form để chọn thời gian và ngày */}
                <form>
                    <h3>Thời gian</h3>
                    <div>
                        <label>Thời điểm xuất phát: </label>
                        <DatePicker
                            selected={formData.time} // Hiển thị thời gian hiện tại
                            onChange={(time) => setFormData({ ...formData, time })} // Cập nhật thời
gian khi thay đổi

                            showTimeSelect // Hiển thị phần chọn thời gian
                            showTimeSelectOnly // Chỉ hiển thị chọn thời gian, không có ngày
```

```
                        timeIntervals={10} // Khoảng cách giữa các mốc thời gian
                        timeCaption="Time" // Tiêu đề cho phần chọn thời gian
                        dateFormat="HH:mm" // Định dạng giờ: phút
                    />
                </div>
                <div />
                <div>
                    <label>Ngày xuất phát </label>
                    <DatePicker
                        selected={formData.date} // Hiển thị ngày hiện tại
                        onChange={(date) => setFormData({ ...formData, date })
                        } // Cập nhật ngày khi thay đổi
                        dateFormat="yyyy-MM-dd" // Định dạng ngày: Năm-Tháng-Ngày
                    />
                </div>
            </form>
```



Hình 5.1 Giao diện người dùng

Lấy dữ liệu thời tiết: Ứng dụng sử dụng OpenWeatherMap API để lấy dữ liệu thời tiết cho khu vực người dùng chọn. Tùy vào ngày chọn, API sẽ trả về thời tiết hiện tại hoặc dự báo cho các ngày tiếp theo. Dữ liệu bao gồm nhiệt độ và mô tả thời tiết, được sử dụng để dự đoán giao thông.

```
// Hàm để lấy dữ liệu thời tiết từ OpenWeatherMap API
const fetchWeather = async () => {
    const apiKey = "511e39ec6ae9e0c06a8b24a62ee17305"; // API Key cho OpenWeatherMap
    const city = "San Francisco"; // Thành phố cần lấy thời tiết
    const isFutureDate = formData.date > new Date(); // Kiểm tra xem ngày có ở tương lai không

    let weatherAPIUrl;

    if (isFutureDate) {
        // Sử dụng API dự báo thời tiết cho các ngày tương lai (tối đa 7 ngày)
```

```
            weatherAPIUrl =
`https://api.openweathermap.org/data/2.5/forecast?q=${city}&appid=${apiKey}&units=metric`;
        } else {
            // Sử dụng API thời tiết hiện tại cho các ngày hiện tại hoặc quá khứ
            weatherAPIUrl =
`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`;
        }

        try {
            const response = await axios.get(weatherAPIUrl);
            const data = response.data;

            if (isFutureDate) {
                // Xử lý dữ liệu dự báo thời tiết
                const selectedDate = formData.date.toISOString().split("T")[0]; // Ngày được chọn
(YYYY-MM-DD)
                const forecast = data.list.find((entry) => entry.dt_txt.startsWith(selectedDate)); //
Tìm dữ liệu dự báo cho ngày được chọn

                if (forecast) {
                    setWeatherData({
                        temp: forecast.main.temp, // Cập nhật nhiệt độ
                        description: forecast.weather[0].description, // Cập nhật mô tả thời tiết
                        weatherId: forecast.weather[0].id, // Lưu mã thời tiết (weather id)
                    });
                } else {
                    setError("Weather forecast data not available for the selected date.");
                }
            } else {
                // Xử lý dữ liệu thời tiết hiện tại
                setWeatherData({
                    temp: data.main.temp, // Cập nhật nhiệt độ
                    description: data.weather[0].description, // Cập nhật mô tả thời tiết
                    weatherId: data.weather[0].id, // Lưu mã thời tiết (weather id)
                });
            }
        } catch (error) {
            console.error("Error fetching weather data:", error); // In lỗi nếu có
            setError("Unable to fetch weather data. Please try again later."); // Hiển thị thông báo
lỗi cho người dùng
        }
    };
```

Tính toán và dự đoán giao thông: Sau khi người dùng nhập thông tin, dữ liệu như ngày, giờ, và thời tiết sẽ được gửi đến backend. Backend sẽ xử lý dữ liệu và trả về dự đoán giao thông. Các yếu tố đầu vào bao gồm:

- Ngày trong tuần (CodedDay)
- Múi giờ (Zone)
- Mã thời tiết (CodedWeather)

- Nhiệt độ (Temperature)

```
// Hàm gửi dữ liệu tới backend để dự đoán giao thông
    const sendDataToBackend = async () => {
        try {
            // Tính Zone từ thời gian chọn
            const totalMinutes = formData.time.getHours() * 60 + formData.time.getMinutes(); // Tổng số
phút từ 00:00
            const zone = Math.floor(totalMinutes / 10); // Chuyển đổi thành zone (mỗi zone = 10 phút)

            // Tính CodedDay từ ngày chọn
            const codedDay = formData.date.getDay() === 0 ? 7 : formData.date.getDay(); // CodedDay
(Chủ nhật = 7, Thứ 2 = 1, ...)

            // Lấy mã thời tiết từ weatherData (dữ liệu thời tiết đã thu thập)
            const codedWeather = weatherData.weatherId || 31; // Nếu không có mã thời tiết, mặc định là
Clear sky (31)
// Hàm chuyển mã thời tiết chỉ lấy nhiệt độ
            function convertWeatherCodeToTemperature(code) {
                return weatherMapping[code] || null;  // Trả về null nếu mã không hợp lệ
            }
            // Gửi dữ liệu lên backend
            const response = await axios.post("http://localhost:5000/predict", {
                day: codedDay, // Ngày trong tuần (CodedDay)
                zone: zone, // Múi giờ khoảng thời gian (Zone)
                weather: convertWeatherCodeToTemperature(codedWeather), // Mã thời tiết (CodedWeather)
                temperatureFahrenheit: parseFloat(((weatherData.temp * 9) / 5 + 32).toFixed(2)), //
Chuyển nhiệt độ sang Fahrenheit
            });
            console.log({
                day: codedDay,
                zone: zone,
                weather: convertWeatherCodeToTemperature(codedWeather),
                temperatureFahrenheit: parseFloat(((weatherData.temp * 9) / 5 + 32).toFixed(2)),
            });
            setTrafficPrediction(response.data.prediction); // Lưu kết quả trả về từ backend
        } catch (error) {
            console.error("Error sending data to backend:", error); // In lỗi nếu có
            setError("Unable to fetch traffic prediction. Please try again later."); // Thông báo lỗi
khi không thể lấy dự đoán giao thông
        }
    };
```

Hiển thị kết quả: Dựa trên dự đoán giao thông nhận được từ backend, ứng dụng sẽ thay đổi màu sắc và độ dày của đường trên bản đồ Google Maps để phản ánh mức độ tắc nghẽn giao thông. Các màu sắc được sử dụng:

- Xanh lá: Giao thông thông thoáng
- Xanh sáng: Giao thông thông thoáng nhẹ
- Vàng: Giao thông trung bình

- Cam: Giao thông có tắc nghẽn nhẹ
- Đỏ: Giao thông đông đúc

Ngoài ra, thông báo chi tiết về tình trạng giao thông sẽ được hiển thị cho người dùng.

```
// Hàm getTrafficColor trả về một đối tượng với màu sắc, độ trong suốt và độ dày đường
    const getTrafficColor = (trafficPrediction) => {
        const prediction = parseFloat(trafficPrediction); // Chuyển sang số thực
        if (prediction <= 1) {
            return { color: '#7FFF00', opacity: 1, weight: 5 }; // Xanh lá - Giao thông thông thoáng
        } else if (prediction <= 2) {
            return { color: '#5CB338', opacity: 1, weight: 6 }; // Xanh sáng - Giao thông thông thoáng
nhẹ
        } else if (prediction <= 3) {
            return { color: '#ECE852', opacity: 1, weight: 7 }; // Vàng - Giao thông trung bình
        } else if (prediction <= 4) {
            return { color: '#FFB546', opacity: 1, weight: 8 }; // Cam - Giao thông có chút tắc nghẽn
        } else {
            return { color: '#FB4141', opacity: 1, weight: 9 }; // Đỏ - Giao thông đông đúc
        }
    };
    // Hàm hiển thị thông báo tùy thuộc vào dự đoán giao thông
    const displayTrafficMessage = () => {
        if (trafficPrediction !== null) {
            const prediction = parseFloat(trafficPrediction);
            if (prediction <= 1) {
                return "Dự đoán: Giao thông thông thoáng. Bạn có thể khởi hành ngay bây giờ. (1)";
            } else if (prediction <= 2) {
                return "Dự đoán: Giao thông thông thoáng nhẹ. Có thể khởi hành nhưng hãy lưu ý thời
gian.(2)";
            } else if (prediction <= 3) {
                return "Dự đoán: Giao thông trung bình. Có thể có chút chậm trễ.(3)";
            } else if (prediction <= 4) {
                return "Dự đoán: Giao thông tắc nghẽn nhẹ. Bạn nên xem xét thời gian khởi hành.(4)";
            } else {
                return "Dự đoán: Giao thông đông đúc. Bạn nên cân nhắc thay đổi thời gian khởi
hành.(5)";
            }
        }
        return null;
    };
```

5.3 Phát triển Backend

Backend của ứng dụng sử dụng **Flask** để xây dựng API RESTful cho phép frontend gửi yêu cầu và nhận kết quả dự đoán. Cụ thể:

- Mã nguồn backend sử dụng Flask và TensorFlow (với Keras) để cung cấp API dự đoán giao thông qua mô hình học sâu LSTM. Với cấu hình môi trường

```
import os

os.environ["TF_ENABLE_ONEDNN_OPTS"] = "0"
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "1"
```

Cấu hình TensorFlow để tối ưu hóa các tùy chọn của OneDNN và giảm mức độ log từ TensorFlow. Điều này giúp giảm sự nhiễu thông tin trong quá trình thực thi.

- Import các thư viện cần thiết

```
import tensorflow as tf
import keras
from flask import Flask, request, jsonify
from flask_cors import CORS
import numpy as np
from datetime import datetime
from sklearn.preprocessing import LabelEncoder, StandardScaler
import joblib
from tensorflow.keras.models import load_model  # type: ignore
import threading
import time
import numpy as np
```

- Flask: Framework để xây dựng API RESTful.
- CORS: Được sử dụng để cho phép frontend từ các domain khác gửi yêu cầu đến backend (giúp tránh lỗi CORS khi frontend và backend chạy trên các máy chủ khác nhau).
- TensorFlow & Keras: Thư viện học sâu (deep learning) để tải và sử dụng mô hình đã huấn luyện.
- Scikit-learn: Dùng để chuẩn hóa dữ liệu và mã hóa các nhãn (LabelEncoder).
- Joblib: Dùng để tải các đối tượng đã được lưu trữ (như bộ chuyển đổi cột, bộ chuẩn hóa).
- Threading và Time: Được sử dụng để xử lý các tác vụ không đồng bộ và đo thời gian thực thi.

- Khởi tạo Flask app: Flask được sử dụng để tạo ứng dụng web với các route xử lý các yêu cầu HTTP. Mỗi yêu cầu từ frontend sẽ được gửi tới một route cụ thể trong ứng dụng Flask để xử lý.

```
# Khởi tạo Flask app
app = Flask(__name__)

# Cấu hình CORS để cho phép frontend gửi yêu cầu từ các domain khác
```

```
CORS(app)
```

- Cài đặt mô hình học máy: Mô hình LSTM được huấn luyện từ trước và lưu trữ dưới dạng file .keras. Các mô hình này được tải vào bộ nhớ khi ứng dụng Flask khởi động để sử dụng cho các dự đoán mà không cần tải lại mỗi lần.

```
# Tải các mô hình đã huấn luyện khi server khởi động
ct = joblib.load("column_transformer1.pkl")  # Đọc ColumnTransformer đã huấn
luyện
sc = joblib.load("scaler1.pkl")  # Đọc Scaler đã huấn luyện
lstm_model = load_model("lstm_model1.keras")  # Tải mô hình LSTM
labelencoder_Y = joblib.load("labelencoder_Y1.pkl")  # Đọc LabelEncoder đã
huấn luyện
```

– Column Transformer và Scaler: Các bộ chuyển đổi (transformer) và chuẩn hóa dữ liệu (scaler) cũng được tải để xử lý các dữ liệu đầu vào từ frontend trước khi đưa vào mô hình học máy. LabelEncoder được sử dụng để giải mã kết quả dự đoán.

- Hàm dự đoán

```
# Hàm xử lý dự đoán (trong môi trường async)
def process_prediction(data):
    try:
        # Trích xuất các tham số từ dữ liệu gửi lên
        day = data["day"]
        zone = data["zone"]
        weather = data["weather"]
        temperature_fahrenheit = data["temperatureFahrenheit"]

        # Đóng gói dữ liệu thành mảng numpy để xử lý
        new_data = np.array([[day, zone, weather, temperature_fahrenheit]])

        # Mã hóa các đặc trưng phân loại của dữ liệu mới
        new_data_encoded = ct.transform(new_data)

        # Chuẩn hóa dữ liệu mới
        new_data_scaled = sc.transform(new_data_encoded)

        # Định dạng lại dữ liệu cho LSTM
        new_data_lstm = np.reshape(
            new_data_scaled, (new_data_scaled.shape[0], 1,
new_data_scaled.shape[1])
        )

        # Dự đoán nhãn cho dữ liệu mới bằng mô hình LSTM
        y_pred_new = lstm_model.predict(new_data_lstm)
```

```
        # Giải mã nhãn dự đoán
        y_pred_new_label = labelencoder_Y.inverse_transform(
            np.argmax(y_pred_new, axis=1)
        )

        return y_pred_new_label[0]
    except Exception as e:
        return str(e)
```

- Hàm process_prediction nhận đầu vào là dữ liệu dự đoán từ frontend (ngày, khu vực, thời tiết, nhiệt độ), xử lý chúng qua các bước:
- Mã hóa các đặc trưng phân loại (sử dụng ct).
- Chuẩn hóa các đặc trưng (sử dụng sc).
- Định dạng lại dữ liệu thành định dạng mà mô hình LSTM cần.
- Dự đoán nhãn giao thông và giải mã nhãn dự đoán (sử dụng labelencoder_Y).

- API Dự đoán: Một API POST được tạo tại route /predict để nhận dữ liệu từ frontend, xử lý dữ liệu, thực hiện dự đoán và trả lại kết quả. Dữ liệu nhận từ frontend bao gồm các tham số như day, zone, weather, và temperatureFahrenheit. Sau khi xử lý và chuẩn hóa dữ liệu, mô hình LSTM sẽ trả về dự đoán mức độ giao thông, kết quả này sẽ được gửi về frontend.

```
@app.route("/predict", methods=["POST"])
def predict():
    try:
        start_time = time.time()  # Bắt đầu đo thời gian
        # Lấy dữ liệu từ request
        data = request.json
        print("Received data:", data)  # Kiểm tra dữ liệu nhận được

        # Kiểm tra các trường bắt buộc
        required_keys = ["day", "zone", "weather", "temperatureFahrenheit"]
        if not all(key in data for key in required_keys):
            print(error_message)
            return jsonify({"error": error_message}), 400

        # Kiểm tra giá trị hợp lệ
        if not isinstance(data["day"], int) or not isinstance(data["zone"],
int):
            return jsonify({"error": "Invalid data type for 'day' or
'zone'"}), 400

        # Tạo một luồng mới để xử lý dự đoán trong nền
        prediction = process_prediction(data)
```

```
        print("Prediction:", prediction)
        end_time = time.time()  # Kết thúc đo thời gian
        print(f"Prediction time: {end_time - start_time:.4f} seconds")
        # Trả về kết quả dự đoán ngay lập tức mà không đợi thread hoàn thành
        return jsonify({"prediction": str(prediction)})

    except Exception as e:
        error_message = str(e)
        print("Error in /predict:", error_message)
        # Xử lý lỗi nếu có
        return jsonify({"error": str(e)}), 400
```

- Định nghĩa một route /predict nhận yêu cầu POST từ frontend.
- Kiểm tra xem dữ liệu gửi lên có hợp lệ hay không (kiểm tra các trường bắt buộc).
- Gọi hàm process_prediction để xử lý dữ liệu và trả về kết quả dự đoán.
- Kết quả dự đoán sẽ được trả về dưới dạng JSON.

- Chạy ứng dụng Flask

```
if __name__ == "__main__":
    # Chạy ứng dụng Flask trong môi trường production với gunicorn
    app.run(debug=True, host="0.0.0.0", port=5000)
```

- Chạy ứng dụng Flask trong môi trường phát triển với debug=True.
- Ứng dụng sẽ lắng nghe trên tất cả các địa chỉ IP (host="0.0.0.0") và cổng 5000.

5.5 Kết luận Chương V

Trong chương này, chúng ta đã trình bày quy trình phát triển và triển khai ứng dụng web dự báo giao thông. Quá trình này bao gồm việc phát triển phần frontend và backend, tích hợp mô hình Machine Learning vào hệ thống, và triển khai ứng dụng lên môi trường production. Ứng dụng web này sẽ cung cấp những dự đoán chính xác và kịp thời về tình hình giao thông, giúp người dùng lên kế hoạch di chuyển hiệu quả hơn.

KẾT LUẬN

Kết quả đạt được

Trong quá trình phát triển và triển khai hệ thống dự báo giao thông, chúng ta đã đạt được một số kết quả quan trọng như sau:

1. Ứng dụng hoàn thiện với đầy đủ tính năng: Hệ thống đã triển khai thành công các tính năng chính bao gồm bản đồ thời gian thực, dự báo lưu lượng giao thông, thông báo cảnh báo và các tính năng liên quan đến việc cập nhật dữ liệu từ các API bên ngoài (Google Maps, OpenWeatherMap,..)
2. Mô hình Machine Learning được tích hợp thành công: Mô hình dự báo lưu lượng giao thông sử dụng dữ liệu lịch sử và thời gian thực đã được huấn luyện và tích hợp vào hệ thống backend. Dữ liệu từ các API được xử lý và cung cấp kết quả dự báo giao thông chính xác.
3. Tối ưu hóa hiệu suất ứng dụng: Ứng dụng đã được tối ưu hóa về tốc độ tải trang, hiệu suất tổng thể, cũng như khả năng chịu tải trong môi trường sử dụng thực tế, đảm bảo hoạt động mượt mà ngay cả khi có nhiều người dùng truy cập đồng thời.
4. Kiểm thử và đánh giá người dùng: Quá trình kiểm thử chức năng và hiệu suất hệ thống đã được thực hiện thành công, xác nhận rằng ứng dụng đáp ứng yêu cầu về tính năng và hiệu suất. Phản hồi từ người dùng thử nghiệm đã giúp cải thiện giao diện và trải nghiệm người dùng.

Hạn chế của hệ thống

Mặc dù hệ thống đã được triển khai thành công và đạt được nhiều kết quả tích cực, nhưng vẫn còn một số hạn chế cần lưu ý:

1. Dữ liệu dự báo chưa hoàn toàn chính xác: Mặc dù mô hình đã được huấn luyện với dữ liệu lịch sử, việc dự báo chính xác lưu lượng giao thông trong thời gian thực vẫn còn gặp một số vấn đề, nhất là đối với các tình huống bất ngờ như tai nạn giao thông, sự kiện đặc biệt hay thời tiết cực đoan.
2. Khả năng mở rộng hệ thống: Hệ thống có thể gặp khó khăn khi số lượng người dùng tăng lên đáng kể trong các khoảng thời gian cao điểm. Việc xử lý đồng thời nhiều yêu cầu có thể gây ra độ trễ hoặc giảm hiệu suất.

3. Chưa có đầy đủ tính năng cảnh báo: Mặc dù hệ thống cung cấp thông báo cảnh báo, nhưng khả năng cảnh báo về tình trạng giao thông chưa đa dạng, như cảnh báo về tắc nghẽn giao thông hoặc tai nạn.
4. Dữ liệu phụ thuộc vào API bên ngoài: Việc sử dụng các API bên ngoài như Google Maps API và OpenWeatherMap có thể gây ra sự phụ thuộc vào tính ổn định và chính xác của các dịch vụ này. Nếu API gặp sự cố hoặc thay đổi về cách thức cung cấp dữ liệu, hệ thống có thể gặp vấn đề.

Định hướng phát triển hệ thống

Dựa trên các kết quả đạt được và những hạn chế đã chỉ ra, hệ thống có thể được phát triển theo những định hướng sau:

1. Cải thiện mô hình dự báo: Nâng cao độ chính xác của mô hình dự báo bằng cách sử dụng các thuật toán machine learning tiên tiến hơn như LSTM, kết hợp thêm dữ liệu thời gian thực từ các cảm biến giao thông và dữ liệu từ mạng xã hội để tạo ra một mô hình dự báo giao thông đa chiều và chính xác hơn.
2. Tăng cường khả năng mở rộng: Cải thiện khả năng mở rộng của hệ thống bằng cách sử dụng các công nghệ mới như Docker và Kubernetes để triển khai hệ thống trong môi trường phân tán, từ đó giảm thiểu độ trễ và tăng khả năng chịu tải.
3. Mở rộng tính năng cảnh báo: Phát triển thêm các tính năng cảnh báo liên quan đến các tình huống bất thường như tai nạn giao thông, đường bị tắc nghẽn, hay thời tiết xấu để người dùng có thể chuẩn bị tốt hơn.
4. Tích hợp thêm dữ liệu và API mới: Khám phá và tích hợp các nguồn dữ liệu khác để làm phong phú thêm các tính năng của ứng dụng. Việc sử dụng các API khác nhau có thể giúp hệ thống cung cấp dữ liệu đa dạng và chính xác hơn.
5. Cải thiện giao diện người dùng: Dựa trên phản hồi từ người dùng thử nghiệm, tiếp tục tối ưu hóa giao diện người dùng để đảm bảo tính trực quan, dễ sử dụng và thân thiện hơn với tất cả người dùng.

Hệ thống đã hoàn thành mục tiêu ban đầu là phát triển một ứng dụng web dự báo giao thông sử dụng machine learning, tuy nhiên, vẫn cần tiếp tục cải tiến và mở rộng tính năng để đáp ứng tốt hơn nhu cầu thực tế của người dùng và đối phó với các thách thức trong tương lai.

DANH MỤC TÀI LIỆU THAM KHẢO

Tài liệu, giáo trình:

[1] PGS.TS Trần Đình Quế, Phân tích và thiết kế hệ thống thông tin, Học viện Công nghệ Bưu chính Viễn thông, 2014. 1.

[2] Giang Thị Thu Huyền và Lê Quý Tài (2022). Ứng dụng học máy trong dự đoán lưu lượng giao thông. Xuất bản ngày 19/05/2022.

[3] Lê Dương Phong (2023). *Nghiên cứu phát triển nền tảng tích hợp phân tích dữ liệu dòng*. Luận văn thạc sĩ kỹ thuật, TP. Hồ Chí Minh.

[4] Mengqi Yang (2020). *Big Data: Issues, Challenges, Tools*. Luận văn tốt nghiệp, Centria University of Applied Sciences, Finland.

[5] CH.Devi, V.V.Sai Lavanya, T.S.Lavanya Pushpa, S.Krishnaveni, S.Kavya, và D.Sai Sri (2023). *Traffic Flow Prediction using Machine Learning*. *Samriddhi Journal*, Tập 15, Số 3, 2023, ISSN: 2229-7111 (Print), 2454-5767 (Online).

[6] Vishwasgowda TM và Chinnaswamy CN (2020). *Traffic Prediction using Machine Learning*. Department of Information Science & Engineering, National Institute of Engineering, Mysuru, Karnataka, India.

[7] INRIX (2022). *2022 Global Traffic Scorecard UK*. Truy cập từ: https://inrix.com/press-releases/2022-global-traffic-scorecard-uk/.

[8] Deekshetha H.R., Shreyas Madhav A.V., Amit Kumar Tyagi (2023). *Traffic Prediction using Machine Learning*. *Mukt Shabd Journal*, ISSN: 2347-3150.

[9] Noor Afiza Mat Razali, Nuraini Shamsaimon, Khairul Khalil Ishak, Suzaimah Ramli, Mohd Fahmi Mohamad Amran và Sazali Sukardi (2021). *Gap, techniques and evaluation: Traffic flow prediction using machine learning and deep learning*. *Journal of Big Data*, 8:152. https://doi.org/10.1186/s40537-021-00542-7.

[10] *Big Data: Issues, Challenges, Tools* - Springer Journal, Truy cập từ: https://link.springer.com/article/10.1186/s40537-021-00542-7.

[11] *Big Data Storage Architecture*, *Journal of Big Data*, Truy cập từ: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00358-x.