

Hệ Thống Điều Khiển Đèn Giao Thông

Thông Minh

Dựa Trên Phương Pháp Lai APC và Học Tăng Cường

Nguyễn Thái Sơn

Ngày 29 tháng 8 năm 2025

Mục lục

Danh mục từ viết tắt và thuật ngữ chuyên ngành	viii
Tóm tắt	x
1 Tổng quan dự án	1
1.1 Giới thiệu dự án	1
1.2 Thành tựu chính	2
1.3 Tóm tắt cải thiện hiệu năng	2
1.4 Tổng quan công nghệ sử dụng	2
2 Giới thiệu	4
2.1 Bài toán nghiên cứu	4
2.1.1 Tác động của ùn tắc giao thông đô thị	4
2.1.2 Giới hạn của điều khiển thời gian cố định	4
2.1.3 Yêu cầu ưu tiên cho phương tiện khẩn cấp	4
2.1.4 Vấn đề rẽ trái và xung đột pha	4
2.2 Mục tiêu nghiên cứu	5
2.3 Phạm vi và giới hạn	5
2.4 Đóng góp chính	5
2.5 Cấu trúc chương tiếp theo	6
3 Tổng quan tài liệu	7
3.1 Phương pháp điều khiển giao thông truyền thống	7
3.2 Hệ thống điều khiển thích nghi, học tăng cường và phương pháp lai	7
3.3 Khoảng trống nghiên cứu	8
3.4 Kết luận chương	8
4 Kiến trúc hệ thống	9
4.1 Thiết kế tổng thể	9
4.1.1 Sơ đồ thành phần hệ thống	9
4.2 Luồng dữ liệu và tích hợp	10
4.3 Các thành phần cốt lõi	10
4.3.1 UniversalSmartTrafficController	10
4.3.2 AdaptivePhaseController (APC)	10
4.3.3 EnhancedQLearningAgent	11

4.3.4	Giao tiếp SUMO–TraCI	11
4.4	Công nghệ sử dụng	11
4.4.1	Python và thư viện	11
4.4.2	Mô phỏng giao thông SUMO	11
4.4.3	Tích hợp API TraCI	11
4.4.4	Cấu hình Supabase	11
5	Điều khiển pha thích nghi (APC)	14
5.1	Nguyên lý thiết kế và kiến trúc	14
5.1.1	Bộ điều khiển pha thích nghi (APC)	15
5.1.2	Khởi tạo và cấu hình hệ thống	15
5.1.3	Cấu trúc dữ liệu và tham số điều khiển	16
5.1.4	Tích hợp với TraCI và SUMO	18
5.2	Quản lý logic pha đèn tín hiệu	20
5.2.1	Cơ chế cache và tối ưu truy xuất	20
5.2.2	Điều khiển chuyển pha an toàn	23
5.2.3	Chèn pha đèn vàng tự động	24
5.2.4	Xử lý và ngăn chặn xung đột pha	27
5.3	Điều chỉnh thời lượng pha động	30
5.3.1	Thuật toán điều chỉnh delta-t	30
5.3.2	Cơ chế extension và cập nhật remaining time	32
5.3.3	Ràng buộc min_green và max_green	33
5.3.4	Tính toán thời lượng tối ưu theo nhu cầu	33
5.4	Hệ thống quản lý yêu cầu ưu tiên	35
5.4.1	Cấu trúc hàng đợi yêu cầu phân cấp	35
5.4.2	Xử lý yêu cầu theo mức độ ưu tiên	35
5.4.3	Cơ chế gom nhiều yêu cầu và xử lý hàng loạt	36
5.4.4	Giải quyết xung đột giữa các yêu cầu	37
5.5	Phát hiện và xử lý tắc nghẽn	38
5.5.1	Thuật toán nhận diện tắc nghẽn giao thông	38
5.5.2	Tổng hợp nhiều yếu tố để xác định mức độ nghiêm trọng	39
5.5.3	Dự báo và phòng ngừa tắc nghẽn	40
5.5.4	Kích hoạt chế độ tắc nghẽn	41
5.6	Quản lý rẽ trái bảo vệ thông minh	42
5.6.1	Phát hiện blocked left turn với xung đột	42
5.6.2	Tạo pha protected left động	43
5.6.3	Cơ chế memory và guard deadline	44
5.6.4	Tối ưu thời lượng protected left phase	45
5.7	Xử lý tình huống khẩn cấp	46
5.7.1	Phát hiện phương tiện ưu tiên	46

5.7.2 Emergency rebalancing khi mất cân bằng lưu lượng	46
5.7.3 Xử lý starvation và ùn tắc nghiêm trọng	47
5.7.4 Cơ chế cooldown và chống nhấp nháy pha (flicker)	48
6 Thành phần học tăng cường	49
6.1 Kiến trúc Agent Q-learning cải tiến	49
6.2 Cơ chế học	50
6.3 Thiết kế vector trạng thái	51
6.4 Thiết kế hàm thưởng	52
7 Chiến lược điều khiển lai	54
7.1 Khung tích hợp	54
7.2 Luồng điều khiển	54
7.3 Giải quyết xung đột	55
8 Quản lý phương tiện ưu tiên	57
8.1 Cơ chế phát hiện phương tiện khẩn cấp	57
8.2 Kiến trúc hàng đợi yêu cầu ưu tiên	57
8.3 Chiến lược phục vụ phương tiện ưu tiên	58
8.4 Tác động tới logic điều khiển tổng thể	58
8.5 Phối hợp với các loại ưu tiên khác	59
8.6 Đánh giá hiệu năng và công bằng	59
8.7 Kịch bản mô phỏng thực nghiệm	59
8.8 Phân tích chi tiết hiệu năng trên mã nguồn bộ điều khiển	60
8.8.1 Tối ưu hóa thời lượng pha phục vụ	60
8.8.2 Kết hợp adaptive RL và rule-based	60
8.8.3 Đảm bảo an toàn và tính phục hồi	60
8.9 Hạn chế và đề xuất cải tiến	61
8.10 Tài liệu tham khảo ứng dụng	61
9 Quản lý dữ liệu	62
9.1 Kiến trúc và cơ chế lưu trữ dữ liệu	62
9.1.1 Tích hợp Supabase: pipeline dữ liệu và tối ưu vận hành	62
9.1.2 Quy trình lưu trữ và đồng bộ Q-Table của tác tử RL	63
9.1.3 Hệ thống ghi log sự kiện điều khiển	64
9.1.4 Tối ưu hiệu suất lưu trữ	65
10 Chi tiết triển khai	66
10.1 Cài đặt và cấu hình	66
10.1.1 Cài đặt môi trường	66
10.2 Tham số cấu hình	67
10.3 Cấu trúc mã nguồn	67

10.3.1 Sơ đồ kiến trúc module	68
10.3.2 Các lớp chính	68
10.3.3 Luồng điều khiển chính	68
10.3.4 Ví dụ code: Khởi tạo APC và RL agent	69
10.3.5 Ví dụ code: Quản lý hàng đợi yêu cầu ưu tiên	69
10.3.6 Ví dụ code: Điều chỉnh thời lượng pha động	70
10.3.7 Ví dụ code: Chèn pha vàng tự động	70
10.3.8 Ví dụ code: Tích hợp Supabase batch writer	71
10.3.9 Diagram: Chu trình điều khiển tổng quát	71
11 Mô hình thí nghiệm	72
11.1 Môi trường mô phỏng	72
11.1.1 Thiết lập topology và logic đèn bằng NETEDIT	72
11.1.2 Tích hợp các file cấu hình vào mô phỏng SUMO	73
11.1.3 Tích hợp TraCI và các thành phần mô phỏng	73
11.2 Cấu hình cơ bản	74
11.2.1 Cấu hình mạng lưới giao thông	74
11.2.2 Cấu hình file mô phỏng SUMO	75
11.2.3 Cấu hình tích hợp bộ điều khiển	75
11.3 Thí nghiệm kiểm soát	75
11.3.1 Tình huống giả lập	75
11.3.2 Quy trình thực nghiệm	76
11.3.3 Tích hợp dashboard trực quan hóa	76
12 Đánh giá hiệu năng và so sánh	78
12.1 Các chỉ số đánh giá	78
12.2 So sánh định lượng và định tính	78
12.2.1 So sánh định lượng giữa hai chế độ	78
12.2.2 So sánh định tính	80
12.3 Đánh giá theo kịch bản	81
12.3.1 Kịch bản 1: Lưu lượng cao giờ cao điểm	81
12.3.2 Kịch bản 2: Xuất hiện ùn tắc cực đoan	81
12.3.3 Phân tích chi tiết bằng mã nguồn	81
12.4 Phân tích khả năng điều chỉnh pha	82
12.5 Kết luận so sánh	83
13 Kết quả và thảo luận	84
13.1 Phát hiện chính	84
13.2 Phân tích hành vi hệ thống	85
13.2.1 Luồng điều khiển ưu tiên	85
13.2.2 Quản lý rẽ trái bảo vệ	85

13.2.3 Quản lý tắc nghẽn và metering	86
13.2.4 Tối ưu hóa thời lượng pha động	87
13.2.5 Đồng bộ sự kiện và trạng thái lên Supabase	88
13.3 Hạn chế và thách thức	88
14 Trực quan hóa và giám sát	90
14.1 Bảng điều khiển thời gian thực	90
14.1.1 Kiến trúc và chức năng chính	90
14.1.2 Các nhóm chức năng	90
14.1.3 Luồng dữ liệu	91
14.2 Phân tích dữ liệu hậu kỳ	91
14.3 Đồng bộ và mở rộng	91
15 Hướng phát triển tương lai	92
15.1 Điều phối đa nút giao	92
15.2 Tích hợp học tăng cường sâu	92
15.3 Giao tiếp xe-hạ tầng (V2I)	93
15.4 Dự báo giao thông	93
15.5 Mở rộng dựa trên điện toán đám mây	94
16 Kết luận	95
16.1 Tóm tắt kết quả nghiên cứu	95
16.2 Tác động đến quản lý giao thông đô thị	96
16.3 Ứng dụng thực tiễn và triển vọng	97
16.4 Khuyến nghị cuối cùng	97
Tài liệu tham khảo	99
A Tài liệu mã nguồn	102
B Tài liệu mã nguồn	103
B.1 Cấu trúc tổng thể mã nguồn	103
B.2 Luồng dữ liệu và vòng lặp điều khiển	103
B.3 Các thuật toán chính và đặc trưng trong mã nguồn	104
B.3.1 Điều chỉnh pha động theo reward	104
B.3.2 Quản lý logic pha đèn và cache	104
B.3.3 Quản lý hàng đợi yêu cầu chuyển pha	105
B.3.4 Phát hiện và xử lý rẽ trái bảo vệ động	105
B.3.5 Cập nhật Q-table và học tăng cường	105
B.3.6 Chiến lược chọn hành động epsilon-greedy và phối hợp với coordinator	106
B.3.7 Ghi log và đồng bộ dữ liệu lên Supabase	106
B.4 Đặc điểm kỹ thuật và kinh nghiệm triển khai	107
B.5 Liên hệ các chương khác	107

B.6	Hướng dẫn đọc mã nguồn	108
B.7	Ví dụ chèn mã nguồn chính	108
B.8	Tài liệu tham khảo liên quan mã nguồn	108
B.9	Ghi chú triển khai thực tế	108
B.10	Kết luận	109
C	Phụ lục: Lược đồ bảng phase_records	110
	Phụ lục: Lược đồ bảng phase_records	110

Danh sách hình vẽ

4.1	Kiến trúc tổng thể hệ thống điều khiển đèn giao thông	9
5.1	Kiến trúc tổng thể của bộ điều khiển APC với các thành phần chính và luồng dữ liệu	14
5.2	Quy trình khởi tạo AdaptivePhaseController.	16
5.3	Cấu trúc hàng đợi yêu cầu với mức độ ưu tiên	18
5.4	Kiến trúc cache hai tầng cho traffic light logic chỉ với APC: E3	22
5.5	Sơ đồ quy trình kiểm soát chuyển pha: (a) kiểm tra chỉ số pha, (b) xử lý thiết lập và cập nhật trạng thái	25
5.6	Sơ đồ chuyển trạng thái với pha vàng tự động	27
5.7	Ma trận xung đột pha cho nút giao 4 hướng	29
5.8	Pipeline chuyển đổi phần thưởng thành điều chỉnh thời lượng pha	31
5.9	Quy trình cập nhật remaining time cho pha động	32
5.10	Kiểm soát thời lượng pha với các ngưỡng ràng buộc	33
5.11	Quy trình chọn thời lượng pha tối ưu dựa trên trạng thái giao thông	34
5.12	Cấu trúc hàng đợi yêu cầu phân cấp cho APC	35
5.13	Quy trình xử lý yêu cầu ưu tiên	36
5.14	Cơ chế stacked requests và batch processing	37
5.15	Quy trình giải quyết xung đột yêu cầu ưu tiên	38
5.16	Sơ đồ phát hiện các kiểu congestion pattern	39
5.17	Tổng hợp các yếu tố tính chỉ số severity	40
5.18	Pipeline dự báo tắc nghẽn và quyết định phòng ngừa	41
5.19	Quy trình kích hoạt và kiểm soát congestion mode	42
5.20	Quy trình phát hiện và xác nhận blocked left turn	43
5.21	Quy trình tạo/kích hoạt pha protected left động	44
5.22	Cơ chế memory và guard deadline cho pha rẽ trái bảo vệ	45

5.23 Quy trình tối ưu thời lượng protected left phase	45
5.24 Quy trình phát hiện và xử lý phương tiện ưu tiên	46
5.25 Sơ đồ emergency rebalancing khi xảy ra lane imbalance	47
5.26 Cơ chế xử lý starvation và ùn tắc nghiêm trọng	48
5.27 Quy trình cooldown để ngăn chặn hiện tượng flicker pha	48
 6.1 Kiến trúc agent Q-learning và mối liên kết với APC	50
6.2 Quy trình học và chọn hành động của agent	51
6.3 Cấu trúc vector trạng thái cho agent Q-learning	52
6.4 Các yếu tố cấu thành hàm thưởng agent Q-learning	53
 7.1 Khung tích hợp chiến lược điều khiển lai	54
7.2 Flowchart luồng điều khiển của hệ thống lai	55
7.3 Quy trình giải quyết xung đột ưu tiên trong điều khiển lai	56
 8.1 Quy trình phát hiện và phục vụ xe ưu tiên	58
8.2 Giải quyết xung đột giữa các loại yêu cầu ưu tiên	59
8.3 Hiệu năng phục vụ xe ưu tiên qua các kịch bản	59
 9.1 Luồng dữ liệu từ bộ điều khiển lên Supabase cloud	63
9.2 Quy trình lưu trữ và đồng bộ Q-Table cùng trạng thái hệ thống	64
9.3 Pipeline ghi log sự kiện từ bộ điều khiển lên Supabase	65
 10.1 Kiến trúc module tổng thể của hệ thống điều khiển	68
10.2 Chu trình điều khiển tổng quát của hệ thống	71
 11.1 Thiết lập mô hình thí nghiệm và logic đèn bằng NETEDIT	73
11.2 Set up mô phỏng SUMO với tích hợp TraCI, event injection	74
11.3 Ảnh set up topology mạng lưới, traffic demand, logic đèn	75
11.4 Ảnh set up các kịch bản kiểm thử trong mô phỏng SUMO	76
11.5 Dashboard trực quan hóa kết quả thí nghiệm mô phỏng SUMO	77
 12.1 Biểu đồ 1: Hiệu năng hệ thống với chế độ mặc định (Fixed-time)	79
12.2 Biểu đồ 3: Hiệu năng hệ thống với bộ điều khiển APC–RL	79
12.3 So sánh trạng thái nút giao giữa hai chế độ tại cùng thời điểm: bên trái là trạng thái tắc nghẽn khi chạy mặc định, bên phải là trạng thái thông thoáng hơn khi dùng bộ điều khiển.	80
12.4 Phân tích mở rộng/rút ngắn pha đèn giao thông: Trên là thời gian extension/reduction từng bước, dưới là so sánh thời lượng pha thực tế với baseline.	82
 13.1 So sánh hiệu năng giữa APC–RL và điều khiển cố định	85

13.2 Quy trình phát hiện và giải quyết rẽ trái bị chặn	86
13.3 Phối hợp điều chỉnh pha khi phát hiện congestion/spillback	87
13.4 Pipeline cập nhật quyết định RL agent	87
13.5 Các thách thức và hạn chế của hệ thống APC–RL	89
 14.1 Giao diện bảng điều khiển SmartIntersectionTrafficDisplay	90
15.1 Ý tưởng kiến trúc điều phối đa nút giao thông	92
15.2 Kiến trúc agent RL sâu	93
15.3 Ý tưởng giao tiếp hai chiều giữa xe và hạ tầng	93
15.4 Ý tưởng pipeline dự báo traffic	94
15.5 Ý tưởng kiến trúc điện toán đám mây cho hệ thống	94
 16.1 Tổng quan pipeline điều khiển lai APC–RL thực nghiệm	96
16.2 So sánh hiệu năng kiểm soát qua các chỉ số – thời gian chờ, thông lượng, gridlock	96

Danh sách bảng

4.1 Các trường cột lỗi của bảng phase_records	12
12.1 So sánh định lượng các chỉ số chính giữa hai chế độ điều khiển	79
C.1 Lược đồ đầy đủ của bảng phase_records	111

Thuật ngữ chuyên ngành

Spillback

Hiện tượng ùn tắc dội ngược: Khi hàng chờ tại một nút giao thông kéo dài vượt qua điểm đầu nút giao, làm cản trở hoặc chặn luôn dòng xe ở nút giao phía trước. Spillback gây ra hiệu ứng dây chuyền, lan rộng tắc nghẽn trong mạng lưới đô thị.

Gridlock

Kẹt lưới giao thông: Toàn bộ các nút giao đều bị kẹt, xe không thể di chuyển qua bất kỳ hướng nào do xung đột dòng xe cắt nhau.

Starvation

Đói phục vụ: Một làn hoặc hướng giao thông bị bỏ qua quá lâu, dẫn đến hàng chờ

kéo dài không được phục vụ, thường xảy ra khi chính sách ưu tiên quá mức cho các hướng khác.

Protected left

Pha rẽ trái bảo vệ: Pha đèn tín hiệu được thiết kế riêng để cho phép rẽ trái an toàn, không bị xung đột với dòng đi thẳng đối diện.

Adaptive Phase Control (APC)

Điều khiển pha thích nghi: Phương pháp điều chỉnh thời lượng các pha đèn dựa trên trạng thái giao thông thực tế (hàng chờ, số xe dừng, v.v.).

Reinforcement Learning (RL)

Học tăng cường: Kỹ thuật học máy cho phép hệ thống điều khiển tín hiệu học từ phần thưởng/hình phạt dựa trên kết quả thực tế của hành động.

Green wave

Làn sóng xanh: Chuỗi các đèn tín hiệu được điều phối để xe chạy liên tục qua nhiều nút giao mà không phải dừng lại.

Phase duration

Thời lượng pha: Khoảng thời gian một pha đèn (xanh/vàng/đỏ) được duy trì trước khi chuyển sang pha tiếp theo.

Supabase

Nền tảng cơ sở dữ liệu đám mây, dùng lưu trữ trạng thái, nhật ký sự kiện và bảng Q-learning cho hệ thống điều khiển.

SUMO

Simulation of Urban Mobility: Phần mềm mô phỏng giao thông vi mô, dùng kiểm thử các thuật toán điều khiển tín hiệu.

TraCI

Traffic Control Interface: Giao thức kết nối giữa Python và SUMO, cho phép điều khiển và lấy dữ liệu mô phỏng theo thời gian thực.

Q-table

Bảng giá trị Q trong Q-learning: Lưu trữ giá trị kỳ vọng cho mỗi trạng thái và hành động, dùng để ra quyết định tối ưu cho agent học tăng cường.

Pending requests

Hàng đợi yêu cầu: Danh sách các yêu cầu chuyển pha đèn đang chờ xử lý, thường được xếp theo mức độ ưu tiên và thời gian.

Tóm tắt

Đèn giao thông đóng vai trò then chốt trong việc điều tiết lưu thông đô thị, song các hệ thống truyền thống thường thiếu khả năng thích ứng với biến động giao thông thực tế. Trong những năm gần đây, các phương pháp điều khiển thích nghi đã được nghiên cứu, nhưng vẫn còn tồn tại những hạn chế như hiện tượng nhấp nháy pha, xử lý tình huống khẩn cấp chưa hiệu quả và khó mở rộng cho nhiều nút giao.

Để khắc phục những vấn đề này, nghiên cứu đề xuất một bộ điều khiển nút giao thông thông minh được triển khai trên môi trường mô phỏng [Simulation of Urban MObility \(SUMO\)](#), kết nối qua [Traffic Control Interface \(TraCI\)](#), kết hợp kỹ thuật học tăng cường và cơ chế lưu trữ trạng thái trên nền tảng đám mây Supabase. Bộ điều khiển có khả năng tự động điều chỉnh thời lượng đèn xanh dựa trên số lượng xe dừng, chiều dài hàng chờ và lưu lượng tức thời, đồng thời bảo đảm an toàn thông qua việc chèn pha vàng và duy trì tối thiểu thời gian đèn xanh.

Ngoài ra, hệ thống còn tích hợp cơ chế ưu tiên cho phương tiện khẩn cấp, quản lý ủn tắc và hỗ trợ rẽ trái an toàn thông qua hàng đợi yêu cầu có cấu trúc. Kết quả mô phỏng thực nghiệm cho thấy mô hình đề xuất giúp giảm đáng kể thời gian chờ trung bình và cải thiện lưu lượng so với phương pháp điều khiển tín hiệu cố định. Điều này chứng minh tính hiệu quả, linh hoạt và tiềm năng ứng dụng của hệ thống trong quản lý giao thông đô thị thông minh.

Chương 1

Tổng quan dự án

1.1 Giới thiệu dự án

Trong bối cảnh đô thị hóa nhanh chóng, các thành phố lớn ngày càng đối mặt với tình trạng ùn tắc giao thông nghiêm trọng. Sự gia tăng đột biến của phương tiện cá nhân, kết hợp với tốc độ mở rộng hạ tầng giao thông chưa theo kịp, đã tạo ra những áp lực lớn đối với hệ thống điều tiết lưu lượng. Ùn tắc không chỉ gây ra sự lãng phí thời gian mà còn kéo theo nhiều hệ quả tiêu cực khác như tiêu hao nhiên liệu, phát thải khí thải nhà kính, gia tăng ô nhiễm môi trường và ảnh hưởng xấu đến sức khỏe cộng đồng. Bên cạnh đó, các tình huống khẩn cấp như xe cứu thương, cứu hỏa hoặc cảnh sát gặp khó khăn khi di chuyển qua các nút giao đông đúc càng làm bộc lộ hạn chế của hệ thống điều khiển tín hiệu hiện tại.

Các hệ thống đèn tín hiệu giao thông truyền thống thường được lập trình dựa trên chu kỳ cố định, được xác định sẵn từ dữ liệu trung bình lịch sử. Tuy nhiên, phương pháp này thiếu tính linh hoạt, không thể thích ứng kịp thời với biến động lưu lượng thực tế vốn thay đổi liên tục theo thời gian, vị trí và điều kiện giao thông. Kết quả là ở những hướng có mật độ phương tiện thấp, thời gian đèn xanh bị lãng phí, trong khi ở những hướng có lưu lượng cao lại hình thành hàng chờ dài và ùn tắc kéo dài. Sự cứng nhắc này khiến hiệu quả vận hành của mạng lưới giao thông giảm đáng kể và đặt ra yêu cầu cấp bách về một giải pháp điều khiển tín hiệu thông minh, thích ứng theo thời gian thực.

Để giải quyết vấn đề trên, dự án này phát triển một **hệ thống điều khiển đèn giao thông thông minh**, kết hợp giữa **điều khiển pha thích nghi (Adaptive Phase Control – APC)** và **học tăng cường (Reinforcement Learning – RL)**. APC cung cấp nền tảng điều khiển theo luật, cho phép điều chỉnh độ dài pha đèn dựa trên các ngưỡng hàng chờ hoặc số lượng xe dừng. Trong khi đó, RL mang lại khả năng học hỏi từ trải nghiệm, tối ưu dần chiến lược điều khiển thông qua cơ chế phản thưởng – hình phạt, giúp hệ thống thích nghi linh hoạt hơn với các tình huống giao thông phức tạp.

Hệ thống được kiểm thử trong môi trường mô phỏng **SUMO**, kết nối thông qua

TraCI, đồng thời tích hợp **Supabase** để lưu trữ và đồng bộ dữ liệu trên nền tảng đám mây. Cách tiếp cận này không chỉ cho phép đánh giá hiệu năng của bộ điều khiển trong các kịch bản đa dạng mà còn mở ra khả năng mở rộng sang quy mô thực tế, hỗ trợ phân tích dữ liệu dài hạn và triển khai trong các hệ thống giao thông đô thị thông minh trong tương lai.

1.2 Thành tựu chính

- Xây dựng bộ điều khiển lai **APC–RL** điều chỉnh chu kỳ đèn theo trạng thái giao thông.
- Cơ chế **ưu tiên phương tiện khẩn cấp**.
- Thuật toán **phát hiện và xử lý tắc nghẽn** (giảm spillback).
- **Rẽ trái bảo vệ** với pha bảo vệ động.
- Kết nối Supabase để **sao lưu trạng thái** và phân tích.

1.3 Tóm tắt cải thiện hiệu năng

Theo nghiên cứu của Eom và Kim [1], các hệ thống điều khiển tín hiệu thích nghi có thể cải thiện đáng kể hiệu suất giao thông. Kết quả thực nghiệm của dự án này cho thấy:

- **Điều chỉnh pha:** trung bình +13.9s (kéo dài 87.1%, rút ngắn 10.4%), tối đa +88s / -70s.
- **Thời gian chờ:** giảm từ **2061.5s** (mặc định) xuống **1412.7s** (APC–RL) \Rightarrow **-31.5%**.
- **Hàng chờ:** 49.1 xe (APC–RL) so với 51.6 xe (mặc định).
- **Tốc độ trung bình:** 1.2 m/s (APC–RL) vs 0.8 m/s (mặc định).

1.4 Tổng quan công nghệ sử dụng

Hệ thống điều khiển được xây dựng dựa trên sự tích hợp của nhiều công nghệ phần mềm và công cụ mô phỏng, đảm bảo khả năng xử lý dữ liệu thời gian thực, học tăng cường và trực quan hóa kết quả. Cụ thể:

- **SUMO (Simulation of Urban MObility):** công cụ mô phỏng giao thông mã nguồn mở, dùng để xây dựng mạng lưới đường phố, luồng xe, và kiểm thử các chiến lược điều khiển tín hiệu.
- **TraCI (Traffic Control Interface):** giao thức cầu nối giữa Python và SUMO, cho phép điều khiển đèn tín hiệu theo từng bước thời gian, lấy dữ liệu trạng thái như độ dài hàng chờ, thời gian chờ, tốc độ xe.
- **Python:** ngôn ngữ lập trình chính của hệ thống. Các thành phần quan trọng bao gồm:

- Bộ điều khiển pha thích nghi (Adaptive Phase Controller - APC) với logic tính toán thời lượng pha.
 - Agent Q-learning tăng cường (Enhanced Q-Learning Agent) cho việc ra quyết định tối ưu dựa trên trạng thái giao thông.
 - Bộ quản lý vòng lặp chính để phối hợp giữa hai thành phần trên (hybrid APC–RL).
- **Supabase:** nền tảng cơ sở dữ liệu đám mây, được dùng để lưu trữ bảng Q-learning (Q-table), nhật ký các sự kiện điều khiển, dữ liệu trạng thái giao thông, đồng thời hỗ trợ đồng bộ dữ liệu trong thời gian thực.
 - **Các thư viện Python hỗ trợ:**
 - NumPy và Pandas cho xử lý ma trận trạng thái, dữ liệu hàng chò và tính toán thống kê.
 - Matplotlib để trực quan hóa kết quả mô phỏng: hàng chò, thời gian chò, tốc độ trung bình và phân tích điều chỉnh pha.
 - Supabase-py để kết nối và quản lý dữ liệu đám mây.
 - Pickle dùng lưu trữ/tải lại bảng Q-learning dưới dạng file nhị phân.

Chương 2

Giới thiệu

2.1 Bài toán nghiên cứu

2.1.1 Tác động của ùn tắc giao thông đô thị

Ùn tắc giao thông là vấn đề phổ biến tại các trung tâm đô thị, gây tổn thất về thời gian, nhiên liệu và môi trường, đồng thời giảm chất lượng dịch vụ giao thông. Sự biến động theo thời gian và không đồng đều giữa các hướng khiến các giải pháp cố định khó đáp ứng được yêu cầu thực tế, đặc biệt trong giờ cao điểm hoặc khi xuất hiện sự kiện bất thường.

2.1.2 Giới hạn của điều khiển thời gian cố định

Các hệ thống điều khiển tín hiệu truyền thống dựa trên chu kỳ cố định (fixed-time) thường được thiết kế theo mẫu lưu lượng trung bình. Nhược điểm chính là thiếu khả năng thích ứng với biến động thực tế: khi lưu lượng giảm, thời gian xanh có thể bị lãng phí; khi lưu lượng tăng, hàng chờ dễ hình thành và kéo dài. Các phương pháp semi/fully-actuated cải thiện phần nào nhưng vẫn bị hạn chế bởi các tham số cố định và khả năng tối ưu trên mạng lưới lớn.

2.1.3 Yêu cầu ưu tiên cho phương tiện khẩn cấp

Một yêu cầu thiết yếu trong quản lý tín hiệu là đảm bảo phục vụ phương tiện ưu tiên (xe cứu thương, cứu hỏa, cảnh sát). Hệ thống cần phát hiện kịp thời và điều chỉnh pha sao cho phương tiện ưu tiên được thông suốt, đồng thời giảm thiểu tác động tiêu cực lên lưu lượng chung.

2.1.4 Vấn đề rẽ trái và xung đột pha

Xung đột giữa luồng rẽ trái và luồng đi thẳng đối diện là nguyên nhân phổ biến dẫn tới tắc nghẽn cục bộ và nguy cơ tai nạn. Thiết kế pha rẽ trái cố định thường không linh hoạt

khi điều kiện thay đổi, do đó cần cơ chế phát hiện rẽ trái bị chặn và tạo pha bảo vệ động để giải phóng luồng.

2.2 Mục tiêu nghiên cứu

Mục tiêu chính của luận văn là phát triển một hệ thống điều khiển đèn giao thông lai, kết hợp bộ điều khiển pha thích nghi (APC) với học tăng cường (RL), nhằm:

- Tăng khả năng thích ứng với biến động lưu lượng thời gian thực,
- Giảm thời gian chờ trung bình và độ dài hàng chờ,
- Hỗ trợ ưu tiên phương tiện khẩn cấp và xử lý rẽ trái bị chặn,
- Đảm bảo tính an toàn (chèn pha vàng, ngăn xung đột) và khả năng phục hồi khi mất kết nối với hệ thống lưu trữ.

2.3 Phạm vi và giới hạn

Công trình được thực nghiệm trong môi trường mô phỏng SUMO và tập trung chính vào tối ưu cho một nút giao hoặc cụm nút nhỏ. Những giới hạn chính bao gồm:

- Sử dụng Q-learning rời rạc cho tác tử RL (chưa triển khai Deep RL cho không gian trạng thái liên tục).
- Kết quả dựa trên dữ liệu mô phỏng; việc chuyển sang triển khai thực tế (sim-to-real) yêu cầu bổ sung domain randomization và tích hợp cảm biến thực.
- Khả năng điều phối đa nút ở mức sơ khai; cần mở rộng thêm để xử lý mạng lưới lớn với độ trễ giao tiếp thực.

2.4 Đóng góp chính

Luận văn đóng góp những điểm chính sau:

- Mô hình lai APC–RL kết hợp quy tắc an toàn và cơ chế học để điều chỉnh thời lượng pha theo reward đa mục tiêu.
- Cơ chế quản lý yêu cầu ưu tiên (pending requests) cho phép phục vụ xe khẩn cấp, starvation và congestion theo thứ tự ưu tiên.
- Kiến trúc lưu trữ lai (local pickle cho Q-table và Supabase cho log/state) đảm bảo khả năng tái tạo, phân tích và phục hồi.
- Bộ công cụ đánh giá trong SUMO cho phép so sánh với baseline fixed-time và phân tích các kịch bản congestion, priority và blocked left.

2.5 Cấu trúc chương tiếp theo

Tài liệu được tổ chức như sau:

- Chương 3: Tổng quan tài liệu liên quan và khoảng trống nghiên cứu.
- Chương 4–7: Thiết kế hệ thống, APC, thành phần RL và chiến lược điều khiển lai.
- Chương 8–10: Quản lý phương tiện ưu tiên, quản lý dữ liệu và chi tiết triển khai.
- Chương 11–13: Thiết lập thí nghiệm, đánh giá hiệu năng và thảo luận kết quả.
- Chương 14–16: Trực quan hóa, hướng phát triển tương lai và kết luận.

Chương 3

Tổng quan tài liệu

3.1 Phương pháp điều khiển giao thông truyền thống

Các hệ thống điều khiển tín hiệu giao thông truyền thống chủ yếu gồm:

- **Điều khiển chu kỳ cố định (Fixed-time):** Thời gian các pha được thiết kế trước dựa trên dữ liệu lưu lượng lịch sử, không thích ứng được với biến động thực tế. Ưu điểm là đơn giản, dễ triển khai; nhược điểm là dễ gây lãng phí hoặc tắc nghẽn khi lưu lượng thay đổi [2], [3].
- **Điều khiển kích hoạt (Actuated):** Sử dụng cảm biến để phát hiện xe, điều chỉnh thời gian pha trong giới hạn định trước. Linh hoạt hơn, nhưng vẫn bị giới hạn bởi các tham số cứng [4].
- **Điều khiển phối hợp (Coordinated):** Đồng bộ nhiều nút giao để tạo “làn sóng xanh”. Diễn hình là hệ SCOOT, SCATS. Hiệu quả trên các tuyến chính, nhưng phức tạp khi áp dụng cho mạng lưới lớn [5], [6].

3.2 Hệ thống điều khiển thích nghi, học tăng cường và phương pháp lai

Để khắc phục hạn chế của các phương pháp truyền thống, các hệ thống thích nghi như SCOOT, SCATS, OPAC đã ra đời. Các hệ này điều chỉnh thời gian tín hiệu dựa trên trạng thái giao thông thực, tăng khả năng phản ứng với biến động [1].

Gần đây, các kỹ thuật học tăng cường (Reinforcement Learning - RL) được ứng dụng mạnh mẽ, cho phép hệ thống tự học chính sách tối ưu thông qua tương tác và phần thưởng thực tế [7], [8]. RL giúp tối ưu hóa hiệu năng tổng thể và thích nghi với nhiều kịch bản khác nhau.

Xu hướng hiện nay là kết hợp các phương pháp: điều khiển luật, thích nghi và học máy (hybrid). Việc phối hợp này tận dụng ưu điểm từng cách tiếp cận, đảm bảo hệ thống vừa an toàn, vừa tối ưu hóa linh hoạt trong điều kiện phức tạp [9].

3.3 Khoảng trống nghiên cứu

Mặc dù có nhiều tiến bộ, vẫn tồn tại các vấn đề cần giải quyết:

- **Khả năng mở rộng hạn chế:** RL truyền thống (Q-learning) khó khai quát cho mạng lưới lớn, thiếu hàm xấp xỉ như mạng nơ-ron sâu [8], [10].
- **Đánh giá chưa toàn diện:** Nhiều nghiên cứu thiếu benchmark chuẩn, kiểm định thống kê và phân tích đa chiều các chỉ số [11].
- **Phối hợp đa nút giao:** Ít giải pháp cho multi-agent coordination, chưa tối ưu cho mạng lưới lớn với độ trễ giao tiếp thực tế [12].
- **Sim-to-real gap:** Kết quả mô phỏng khó chuyển giao sang ứng dụng thực tế, thiếu tích hợp cảm biến thực và domain randomization [13].
- **Thiết kế reward và công bằng:** Hàm thưởng thường phức tạp, chưa phân tích ảnh hưởng các thành phần tới kết quả, khó đảm bảo fairness giữa các hướng giao thông [14].
- **Đảm bảo an toàn:** Thiếu kiểm thử stress với các tình huống cực đoan, thiếu cơ chế xác minh ràng buộc cứng [15].

3.4 Kết luận chương

Nhu cầu phát triển hệ thống điều khiển tín hiệu giao thông thông minh vẫn còn nhiều thách thức: khả năng mở rộng, phối hợp đa nút, chuyển giao thực tế, đánh giá chuẩn, và đảm bảo an toàn. Luận văn này hướng tới giải quyết một phần các khoảng trống trên bằng cách phát triển bộ điều khiển lai APC-RL với cơ chế ưu tiên, quản lý tắc nghẽn, rẽ trái bảo vệ và lưu trữ dữ liệu thực nghiệm đồng bộ, đặt nền móng cho các nghiên cứu mở rộng trong tương lai.

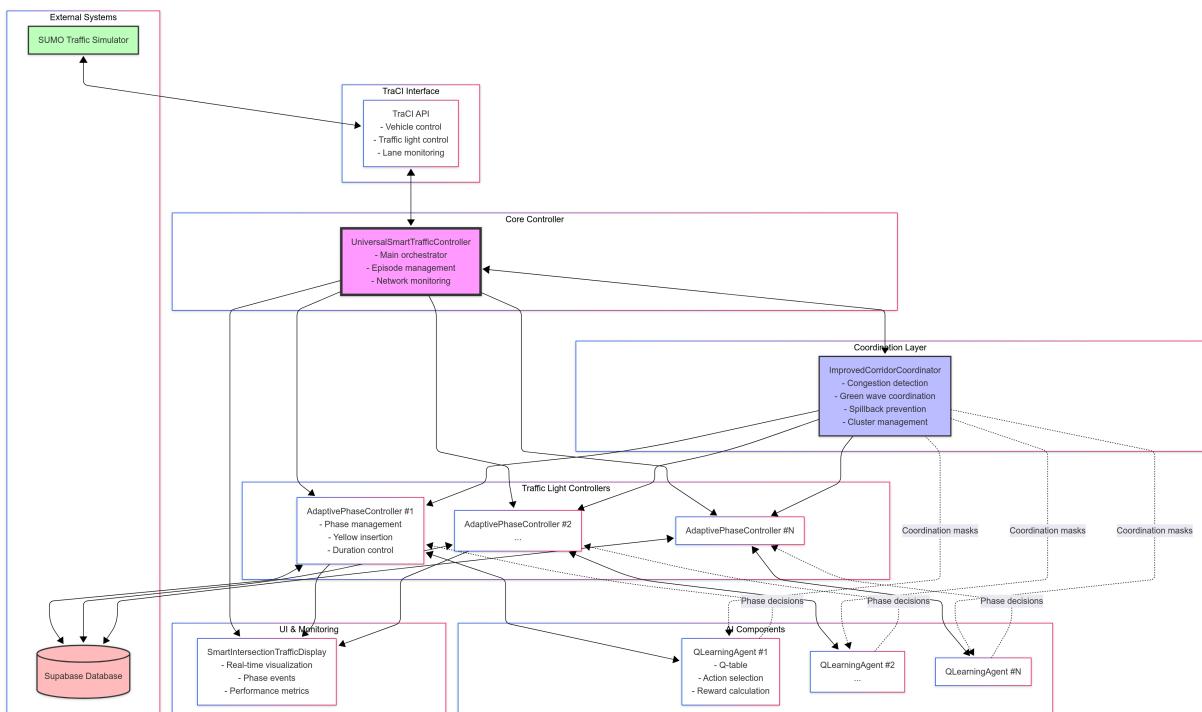
Chương 4

Kiến trúc hệ thống

4.1 Thiết kế tổng thể

Hệ thống điều khiển đèn giao thông thông minh được xây dựng theo mô hình phân tầng, kết hợp giữa điều khiển dựa trên luật (rule-based) và học tăng cường bằng máy (reinforcement learning). Kiến trúc này đảm bảo hệ thống vừa ổn định, an toàn, vừa có khả năng thích ứng và tối ưu theo trạng thái giao thông thực tế.

4.1.1 Sơ đồ thành phần hệ thống



Hình 4.1: Kiến trúc tổng thể hệ thống điều khiển đèn giao thông

Hệ thống gồm 4 tầng chính:

- **Tầng mô phỏng:** Sử dụng SUMO làm môi trường giao thông vi mô, cung cấp dữ liệu thời gian thực về phương tiện, làn đường, đèn tín hiệu.
- **Tầng giao tiếp:** TraCI làm cầu nối hai chiều giữa SUMO và hệ điều khiển Python, cho phép truy vấn trạng thái và gửi lệnh điều khiển.
- **Tầng điều khiển:** UniversalSmartTrafficController quản lý một nút giao, điều phối các AdaptivePhaseController (APC) và RL agent để ra quyết định động, xử lý ưu tiên, tắc nghẽn, rẽ trái bảo vệ.
- **Tầng lưu trữ:** Supabase lưu trữ trạng thái, log sự kiện, lịch sử pha và Q-table; pickle file lưu Q-table cục bộ giúp agent khôi phục trạng thái học.

4.2 Luồng dữ liệu và tích hợp

Quy trình dữ liệu của hệ thống theo chu trình khép kín:

1. TraCI đọc trạng thái giao thông từ SUMO (số lượng xe, hàng chờ, tốc độ...).
2. UniversalSmartTrafficController tổng hợp, phân tích dữ liệu, đánh giá trạng thái nút giao.
3. AdaptivePhaseController và RL agent quyết định chuyển pha, điều chỉnh thời lượng theo trạng thái và lịch sử học.
4. Lệnh điều khiển gửi ngược về SUMO qua TraCI.
5. Kết quả, phần thưởng, log sự kiện được ghi lên Supabase, Q-table được cập nhật để cải thiện hiệu năng lâu dài.

Các điểm tích hợp chính:

- **TraCI API:** Giao tiếp và thao tác trạng thái với SUMO, tối ưu hóa qua subscription.
- **Supabase REST API:** Đồng bộ trạng thái, log, Q-table lên cloud với batch write, retry logic.

4.3 Các thành phần cốt lõi

4.3.1 UniversalSmartTrafficController

Điều khiển trung tâm, quản lý APC cho một nút giao, thu thập và phân tích dữ liệu, nhận diện các tình huống đặc biệt (ưu tiên, tắc nghẽn, starvation).

4.3.2 AdaptivePhaseController (APC)

Điều khiển nút giao, thực hiện điều chỉnh động thời lượng pha, chèn pha vàng, bảo vệ rẽ trái, xử lý hàng đợi ưu tiên, phối hợp với RL agent và controller.

4.3.3 EnhancedQLearningAgent

Tác tử học tăng cường Q-learning, nhận vector trạng thái đa chiều, cập nhật Q-table, tối ưu hóa lựa chọn pha và thời lượng dựa trên reward đa mục tiêu.

4.3.4 Giao tiếp SUMO–TraCI

Các hàm bọc giúp kiểm soát pha an toàn, cache logic đèn tín hiệu (TTL 0.5s), quản lý subscription giảm overhead truyền tin, đảm bảo quá trình điều khiển ổn định.

4.4 Công nghệ sử dụng

4.4.1 Python và thư viện

Hệ thống phát triển trên Python 3.8+:

- **Async/Await, Threading:** Xử lý bất đồng bộ, tăng hiệu suất
- **Type Hints, Dataclasses:** Quản lý dữ liệu rõ ràng, dễ bảo trì
- **NumPy, Pandas, Matplotlib:** Phân tích, xử lý và trực quan hóa dữ liệu
- **Supabase-py:** Kết nối cloud database
- **Pickle:** Lưu/đọc Q-table cục bộ

4.4.2 Mô phỏng giao thông SUMO

SUMO 1.22.0 với NETEDIT cho phép thiết kế mạng lưới, cấu hình traffic demand, xuất dữ liệu chi tiết phục vụ kiểm thử và đào tạo.

4.4.3 Tích hợp API TraCI

Các module chính:

- **traci.trafficlight:** Điều khiển pha, logic đèn
- **traci.lane:** Đọc thông tin làn, hàng chờ
- **traci.vehicle:** Theo dõi phương tiện
- **traci.simulation:** Quản lý mô phỏng chung

4.4.4 Cấu hình Supabase

Supabase sử dụng PostgreSQL cloud với REST API và real-time. Hệ thống tối ưu hóa lưu trữ và truy vấn dữ liệu bằng cấu trúc bảng sau:

Bảng apc_states

Lưu trữ trạng thái tổng thể APC, bao gồm cấu hình pha, hàng đợi sự kiện, trạng thái RL agent.

Listing 4.1: Định nghĩa bảng apc_states

```

1 CREATE TABLE apc_states (
2     id BIGSERIAL PRIMARY KEY,
3     tls_id TEXT NOT NULL,
4     state_type TEXT NOT NULL,
5     data JSONB NOT NULL,
6     created_at TIMESTAMPTZ DEFAULT NOW(),
7     updated_at TIMESTAMPTZ DEFAULT NOW()
8 );
9
10 CREATE INDEX idx_apc_states_tls_type ON apc_states(tls_id, state_type);
11 CREATE INDEX idx_apc_states_data ON apc_states USING gin(data);

```

Trường `state_type` gồm:

- `full`: Snapshot trạng thái toàn bộ APC
- `phase`: Thông tin chi tiết pha đèn
- `event`: Sự kiện điều khiển đơn lẻ

Sử dụng JSONB giúp mở rộng dữ liệu linh hoạt, truy vấn nhanh nhờ GIN index, dễ dàng tích hợp với Python.

Bảng phase_records

Lưu lịch sử các lần điều chỉnh pha đèn, phục vụ phân tích hiệu năng và đào tạo RL agent.

Cột	Kiểu	Vai trò
tls_id	TEXT	Định danh nút giao
phase_idx	INT	Pha được điều chỉnh (0–11)
duration	REAL	Thời lượng thực tế
base_duration	REAL	Thời lượng cơ sở
delta_t	REAL	Mức điều chỉnh
extended_time	REAL	Thời gian mở rộng
reward	REAL	Reward từ RL agent
sim_time	REAL	Thời điểm mô phỏng

Bảng 4.1: Các trường cốt lõi của bảng phase_records

Bảng simulation_events

Lưu mọi sự kiện đặc biệt: chuyển pha khẩn cấp, protected left, congestion, chuyển vàng, v.v.

- **id**: Khóa chính
- **tls_id**: ID đèn giao thông
- **event_type**: Loại sự kiện
- **event_date**: JSONB chi tiết sự kiện
- **sim_time**: Thời điểm mô phỏng
- **created_at**: Thời điểm tạo bản ghi

Ví dụ JSON:

```

1 {
2     "action": "phase_duration_update",
3     "phase": 209
4 }
```

Tối ưu hiệu suất database

- **Indexing**: Composite index cho truy vấn nhanh
- **JSONB + GIN**: Truy vấn linh hoạt, hiệu suất cao
- **Row Level Security**: Bảo mật theo từng dòng dữ liệu
- **Batch operations**: Ghi dữ liệu theo lô giảm số lần round-trip

```

1 ALTER TABLE apc_states ENABLE ROW LEVEL SECURITY;
2 CREATE POLICY "Enable_all_operations_for_authenticated_users"
3     ON apc_states FOR ALL USING (auth.role() = 'authenticated');
```

Chiến lược đồng bộ dữ liệu

- Dữ liệu tạm thời gom vào `_pending_db_ops` (tối đa 1000 bản ghi)
- AsyncSupabaseWriter ghi dữ liệu định kỳ lên Supabase (60s/batch)
- Retry logic thông minh, tối đa 6 lần, dùng exponential backoff
- Nếu Supabase offline, chuyển sang lưu local tạm thời

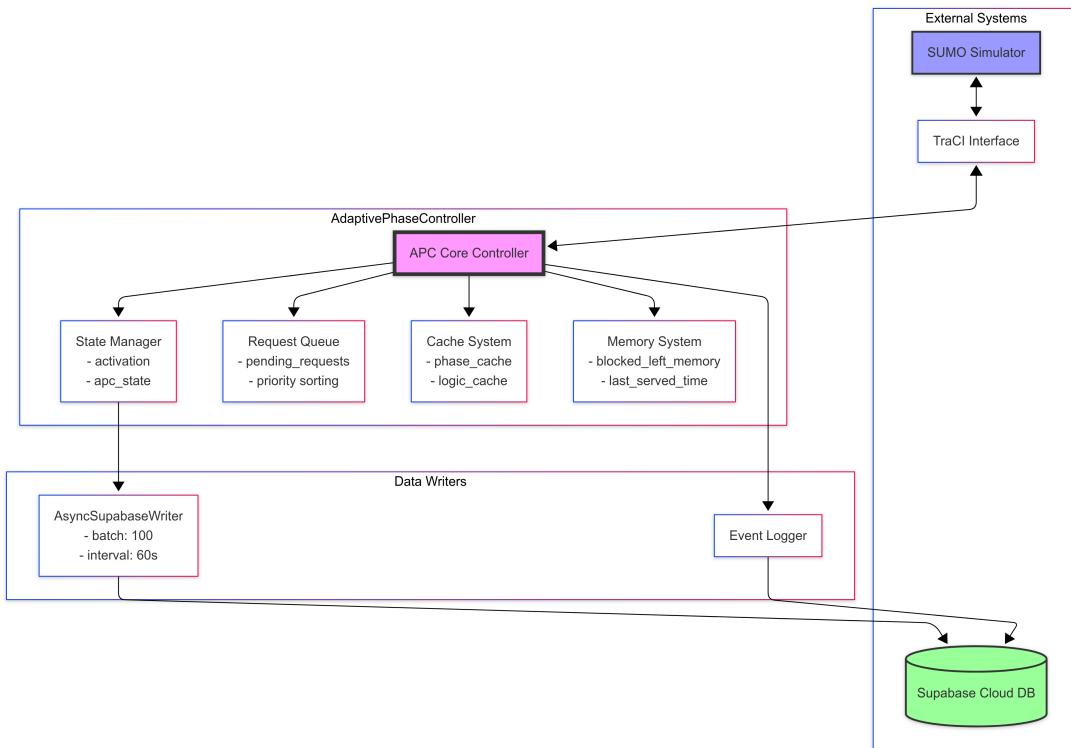
Thiết kế này giúp hệ thống an toàn, hiệu suất cao, sẵn sàng mở rộng cho thực nghiệm hoặc triển khai thực tế.

Chương 5

Điều khiển pha thích nghi (APC)

5.1 Nguyên lý thiết kế và kiến trúc

Bộ điều khiển pha thích nghi (Adaptive Phase Controller - APC) được thiết kế theo nguyên lý điều khiển phân tán với khả năng tự quyết định cục bộ cho từng nút giao thông. Kiến trúc này cho phép mỗi nút giao hoạt động độc lập trong khi vẫn có thể phối hợp với các nút lân cận thông qua cơ chế trao đổi thông tin. APC kết hợp giữa điều khiển dựa trên luật (rule-based control) để đảm bảo an toàn và điều khiển thích ứng (adaptive control) để tối ưu hiệu suất.



Hình 5.1: Kiến trúc tổng thể của bộ điều khiển APC với các thành phần chính và luồng dữ liệu

5.1.1 Bộ điều khiển pha thích nghi (APC)

AdaptivePhaseController là thành phần cốt lõi thực hiện điều khiển tín hiệu cho từng nút giao, đảm bảo hệ thống vừa an toàn vừa thích ứng linh hoạt với trạng thái giao thông thực tế. Các chức năng chính của APC gồm:

- **Điều chỉnh thời lượng pha động:** APC sử dụng hàm `adjust_phase_duration()` để tính toán và cập nhật thời lượng pha tối ưu dựa trên các chỉ số như hàng chờ, thời gian chờ, mật độ giao thông. Công thức điều chỉnh cơ bản:

$$\Delta t = \alpha \cdot (R - R_{target})$$

trong đó R là phần thưởng hiện tại, R_{target} là mục tiêu phần thưởng động, và α là hệ số học. Cơ chế này giúp pha đèn tự động kéo dài hoặc rút ngắn phù hợp với nhu cầu thực tế.

- **Quản lý pha đèn vàng an toàn:** Hàm `insert_yellow_phase_if_needed()` tự động nhận diện và chèn pha vàng khi phát hiện chuyển đổi từ xanh sang đỏ cho bất kỳ hướng nào, đảm bảo phương tiện giảm tốc an toàn. Thời lượng pha vàng được tùy chỉnh theo tốc độ xe và chiều dài hàng chờ.
- **Xử lý rẽ trái bảo vệ thông minh:** Module `detect_blocked_left_turn_with_conflict()` liên tục kiểm tra trạng thái các làn rẽ trái. Khi phát hiện bị chặn (blocked) hoặc xung đột, APC sẽ kích hoạt hoặc tạo pha rẽ trái bảo vệ riêng biệt, đảm bảo luồng giao thông không bị gián đoạn.
- **Quản lý yêu cầu ưu tiên:** Hệ thống hàng đợi `pending_requests` giúp APC xử lý các yêu cầu chuyển pha theo mức độ ưu tiên rõ ràng, gồm: emergency, critical starvation, heavy congestion, và normal. Nhờ đó, các tình huống khẩn cấp, tắc nghẽn cục bộ, hoặc các hướng bị bỏ qua lâu sẽ được phục vụ kịp thời và hợp lý.

5.1.2 Khởi tạo và cấu hình hệ thống

Quá trình khởi tạo APC được thực hiện qua constructor với các tham số quan trọng:

Listing 5.1: Khởi tạo AdaptivePhaseController

```

1 def __init__(self, lane_ids, tls_id, alpha=1.0,
2             min_green=30, max_green=80,
3             r_base=0.5, r_adjust=0.1,
4             severe_congestion_threshold=0.8,
5             large_delta_t=20):
6     self.lane_ids = lane_ids
7     self.tls_id = tls_id
8     # Đăng ký subscription với Traci
9     for lid in self.lane_ids:
10         traci.lane.subscribe(lid, [

```

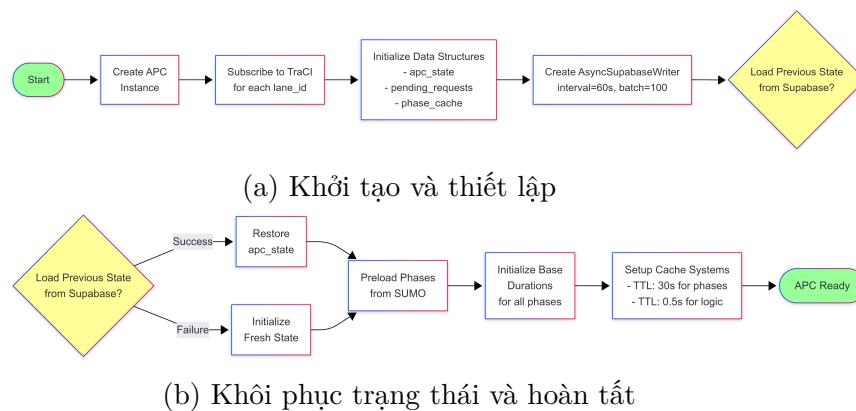
```

11         traci.constants.LAST_STEP_VEHICLE_HALTING_NUMBER,
12         traci.constants.LAST_STEP_MEAN_SPEED,
13         traci.constants.LAST_STEP_VEHICLE_NUMBER,
14         traci.constants.LAST_STEP_VEHICLE_ID_LIST,
15     ]

```

Quá trình khởi tạo bao gồm các bước chính:

- Thiết lập subscription với TraCI:** Hệ thống đăng ký theo dõi các metrics quan trọng cho từng làn đường được quản lý, bao gồm số lượng xe dừng, tốc độ trung bình, tổng số xe và danh sách ID phương tiện. Cơ chế subscription giúp tối ưu băng thông bằng cách chỉ nhận dữ liệu cần thiết.
- Khởi tạo cấu trúc dữ liệu:** Các container quan trọng được khởi tạo:
 - `apc_state`: Dictionary lưu trữ trạng thái toàn cục với events queue (maxlen=5000) và danh sách phases
 - `pending_requests`: Hàng đợi yêu cầu chuyển pha với cơ chế ưu tiên
 - `phase_cache`: Cache thông tin pha với TTL 30 giây để giảm truy vấn database
 - `blocked_left_memory`: Dictionary theo dõi lịch sử làn rẽ trái bị chặn
- Kết nối database:** Khởi tạo `PatchedAsyncSupabaseWriter` với interval 60 giây và batch size 100 records để đồng bộ dữ liệu không đồng bộ lên cloud.
- Tải trạng thái từ Supabase:** Phương thức `_load_apc_state_supabase()` được gọi để khôi phục trạng thái từ lần chạy trước, đảm bảo tính liên tục của hệ thống.
- Đồng bộ pha với SUMO:** `preload_phases_from_sumo()` đảm bảo tất cả các pha trong logic đèn được ghi nhận và có base duration phù hợp.



Hình 5.2: Quy trình khởi tạo AdaptivePhaseController.

5.1.3 Cấu trúc dữ liệu và tham số điều khiển

APC sử dụng hệ thống tham số phân cấp để điều chỉnh hành vi:

Tham số thời gian

- `min_green` (mặc định 30s): Thời gian tối thiểu cho mỗi pha xanh, đảm bảo an toàn và công bằng
- `max_green` (mặc định 80s): Giới hạn trên để tránh độc quyền một hướng
- `low_demand_extend_cap` (4s): Giới hạn mở rộng khi nhu cầu (lượng xe) thấp

Tham số điều khiển thích ứng

- `alpha` (1.0): Hệ số học trong công thức điều chỉnh $\Delta t = \alpha(R - R_{target})$
- `r_base` (0.5): Giá trị reward cơ sở cho thuật toán học
- `r_adjust` (0.1): Hệ số điều chỉnh R_{target} động
- `weights` (vector 4D): Trọng số cho [density, speed, wait, queue] trong hàm reward

Cập nhật mục tiêu reward động (R_{target}): Để hệ thống APC thích nghi tốt với trạng thái giao thông thực tế, giá trị mục tiêu reward (R_{target}) được điều chỉnh động dựa trên reward trung bình gần nhất. Công thức tính như sau:

$$R_{target} = r_{base} + r_{adjust} \cdot (\bar{R} - r_{base})$$

Trong đó:

- r_{base} : Giá trị reward cơ sở, phản ánh mức hiệu suất tối thiểu kỳ vọng.
- r_{adjust} : Hệ số điều chỉnh, kiểm soát mức độ nhạy của mục tiêu với trạng thái thực tế.
- \bar{R} : Giá trị reward trung bình gần nhất (ví dụ: trung bình cộng của reward trong một cửa sổ thời gian hoặc số chu kỳ trước).

Việc cập nhật R_{target} giúp APC tự động thích ứng khi điều kiện giao thông thay đổi (ví dụ: tắc nghẽn, lưu lượng tăng đột biến), từ đó tối ưu hóa quá trình điều chỉnh thời lượng pha đèn.

Nguồn phát hiện sự kiện

Cấu trúc dữ liệu chính

Activation State: Dictionary theo dõi pha đang hoạt động:

```

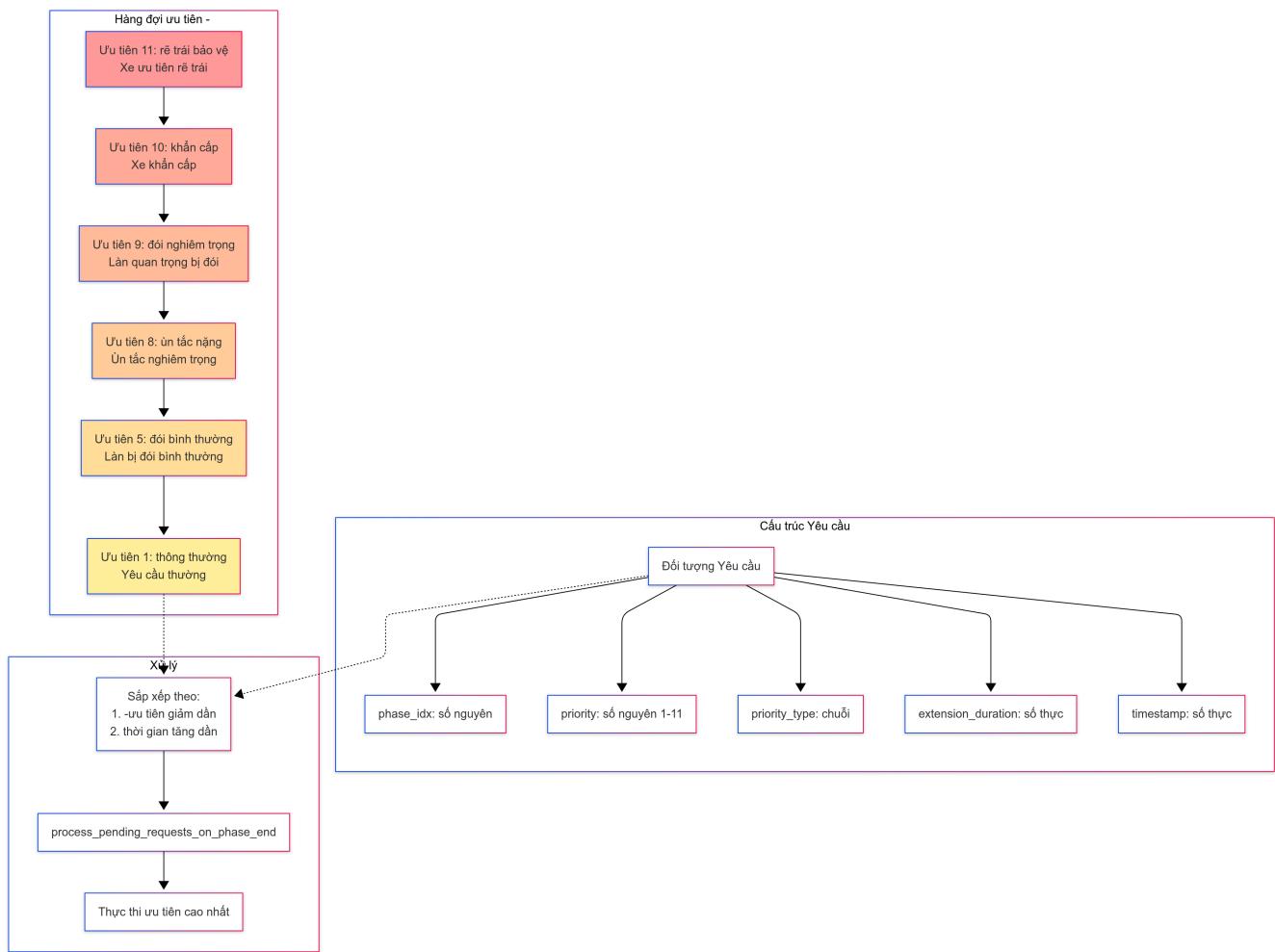
1 self.activation = {
2     "phase_idx": None,           # Chi so pha hien tai
3     "start_time": 0.0,          # Thoi diem bat dau
4     "base_duration": None,      # Thoi luong co so
5     "desired_total": None       # Thoi luong mong muon
6 }
```

Pending Requests Queue: Danh sách các yêu cầu chuyển pha được sắp xếp theo priority và timestamp:

```

1 request = {
2     "phase_idx": int,           # Pha muc tieu
3     "priority": int,           # Muc uu tien (1-11)
4     "priority_type": str,      # Loai: emergency, starvation...
5     "extension_duration": float, # Thoi luong yeu cau
6     "timestamp": float         # Thoi diem tao yeu cau
7 }

```



Hình 5.3: Cấu trúc hàng đợi yêu cầu với mức độ ưu tiên

5.1.4 Tích hợp với TraCI và SUMO

Việc tích hợp với SUMO thông qua TraCI được thực hiện qua nhiều lớp abstraction:

Cơ chế Subscription

APC sử dụng TraCI subscription để nhận dữ liệu real-time hiệu quả:

Listing 5.2: Xử lý subscription results

```

1 def get_lane_stats(self, lane_id):
2     res = traci.lane.getSubscriptionResults(lane_id) or {}
3     q = float(res.get(
4         traci.constants.LAST_STEP_VEHICLE_HALTING_NUMBER,
5         traci.lane.getLastStepHaltingNumber(lane_id)
6     ))
7     v = float(res.get(
8         traci.constants.LAST_STEP_MEAN_SPEED,
9         traci.lane.getLastStepMeanSpeed(lane_id)
10    ))
11    return q, w, v, dens

```

Logic Cache System

Để giảm overhead communication, APC implement cache cho traffic light logic:

Listing 5.3: Cơ chế cache logic

```

1 def _get_logic(self):
2     now = traci.simulation.getTime()
3     if self._logic_cache is None or \
4         now - self._logic_cache_at > self._logic_cache_ttl:
5         self._logic_cache = get_current_logic(self.tls_id)
6         self._logic_cache_at = now
7     return self._logic_cache

```

Cache có thời gian sống (TTL) là 0.5 giây và sẽ được làm mới (invalidate) khi cấu trúc pha đèn thay đổi. Ngoài ra, hệ thống còn hỗ trợ cache dùng chung ở cấp độ controller để tối ưu hiệu suất khi có nhiều bộ điều khiển APC hoạt động đồng thời.

Safe Control Wrappers

Mọi lệnh điều khiển được wrap trong các hàm an toàn:

Listing 5.4: Safe phase control

```

1 def _apply_phase(self, phase_idx, duration):
2     # Clamp phase index
3     safe_idx = self._safe_phase_index(phase_idx,
4                                      force_reload=True)
5     if safe_idx is None:
6         return False
7
8     # Try controller-level setter first
9     if hasattr(self, "controller"):
10        ok = self.controller._safe_set_phase(
11            self.tls_id, safe_idx, duration
12        )

```

```

13     if ok:
14         return True
15
16     # Fallback to direct control
17     return safe_set_phase(self.tls_id, safe_idx, duration)

```

Cơ chế này đảm bảo:

- Phase index luôn nằm trong giới hạn hợp lệ
- Duration được giới hạn trong khoảng [min_green, max_green]
- Xử lý graceful khi SUMO reject lệnh điều khiển
- Đồng bộ state giữa APC và SUMO

Thiết kế tích hợp này cho phép APC hoạt động ổn định trong môi trường mô phỏng phức tạp, xử lý được các trường hợp đặc biệt như chỉ số pha vượt giới hạn, thay đổi cấu trúc mạng, và lỗi kết nối với supabse.

5.2 Quản lý logic pha đèn tín hiệu

Quản lý logic pha đèn tín hiệu là thành phần cốt lõi đảm bảo hoạt động an toàn và hiệu quả của hệ thống điều khiển. APC áp dụng một hệ thống quản lý logic đa tầng với cơ chế cache thông minh, kiểm soát chuyển pha an toàn và xử lý xung đột tự động.

5.2.1 Cơ chế cache và tối ưu truy xuất

Hệ thống cache được xây dựng theo hai lớp, giúp giảm lượng truy cập tới SUMO mà vẫn giữ cho dữ liệu luôn đồng bộ và nhất quán.

Cache cục bộ APC

Mỗi bộ điều khiển APC sẽ giữ một bộ nhớ đệm (cache) riêng cho logic đèn giao thông, với thời gian tồn tại của cache là 0.5 giây (Time To Live - TTL).

Listing 5.5: Implementation của logic cache cục bộ

```

1 def _get_logic(self):
2     now = traci.simulation.getTime()
3     # Kiểm tra cache validity
4     if self._logic_cache is None or \
5         now - self._logic_cache_at > self._logic_cache_ttl:
6         try:
7             # Fetch fresh logic từ SUMO
8             self._logic_cache = get_current_logic(self.tls_id)
9             self._logic_cache_at = now
10        except Exception:

```

```

11         self._logic_cache = None
12     return self._logic_cache

```

Cache hoạt động theo nguyên tắc:

- **Lazy loading:** Logic chỉ được fetch khi cần thiết
- **Time-based invalidation:** Tự động expire sau 0.5 giây
- **Explicit invalidation:** Force refresh khi có mutation

Shared cache ở Controller level

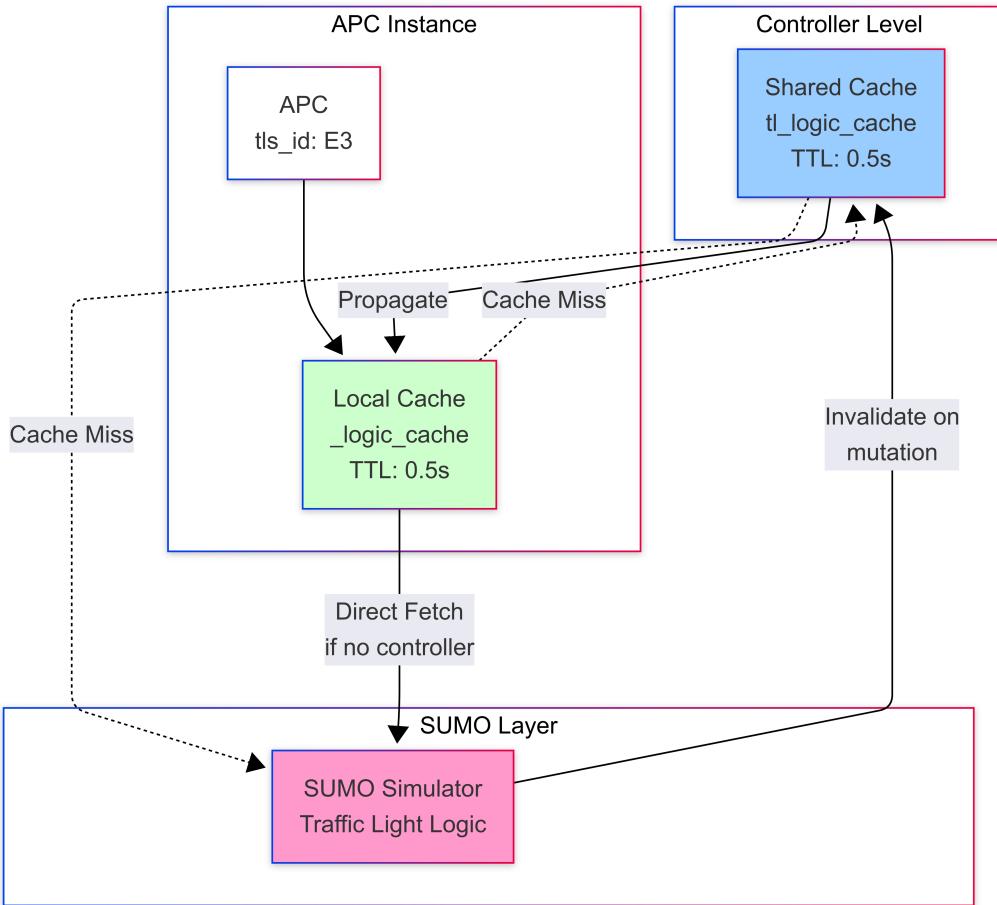
Khi nhiều APC cùng hoạt động, hệ thống sử dụng shared cache để tối ưu:

Listing 5.6: Shared cache mechanism (chỉ với tls_id: E3)

```

1 def _get_logic(self):
2     controller = getattr(self, "controller", None)
3     if controller and hasattr(controller, "tl_logic_cache"):
4         entry = controller.tl_logic_cache.get(self.tls_id)
5         if entry and (now - entry.get("at", -1)) <= self.
6             _logic_cache_ttl:
7             return entry.get("logic")
8     # Update shared cache
9     logic = get_current_logic(self.tls_id)
10    controller.tl_logic_cache[self.tls_id] = {
11        "logic": logic,
12        "at": now
13    }
14    return logic

```



Hình 5.4: Kiến trúc cache hai tầng cho traffic light logic chỉ với APC: E3

Cache invalidation strategy

Hệ thống sử dụng cơ chế xóa (invalidation) thông minh để luôn duy trì sự nhất quán dữ liệu.

Listing 5.7: Cache invalidation mechanism

```

1 def _invalidate_logic_cache(self, tl_id=None):
2     # Invalidate local cache
3     self._logic_cache = None
4     self._logic_cache_at = -1.0
5
6     # Propagate to controller level
7     controller = getattr(self, "controller", None)
8     if controller and hasattr(controller, "_invalidate_logic_cache"):
9         controller._invalidate_logic_cache(self.tls_id)

```

Cache sẽ bị xóa và làm mới trong các trường hợp sau:

1. Khi có thao tác thêm, xóa hoặc chỉnh sửa pha đèn tín hiệu.
2. Khi phát hiện dữ liệu không đồng nhất với trạng thái thực tế từ SUMO.
3. Khi cấu trúc mạng lưới giao thông bị thay đổi.

5.2.2 Điều khiển chuyển pha an toàn

Việc chuyển pha được thực hiện qua nhiều lớp kiểm tra an toàn để đảm bảo không vi phạm ràng buộc và tránh xung đột.

Kiểm tra chỉ số pha hợp lệ

Trước khi thực hiện chuyển pha đèn, hệ thống sẽ kiểm tra để đảm bảo chỉ số pha nằm trong phạm vi cho phép.

Listing 5.8: kẹp chỉ số pha hợp lệ

```

1 def _safe_phase_index(self, idx, force_reload=False):
2     try:
3         if force_reload:
4             self._invalidate_logic_cache()
5         logic = self._get_logic()
6         if not logic or len(logic.getPhases()) <= 0:
7             return None
8         n = len(logic.getPhases())
9         # Clamp to valid range
10        return max(0, min(idx, n - 1))
11    except Exception:
12        return None

```

Áp dụng pha theo nhiều tầng:

Hệ thống thực hiện chuyển pha theo từng lớp kiểm tra và dự phòng, đảm bảo nếu một cách chuyển pha không thành công thì sẽ thử các phương án khác tiếp theo.

Listing 5.9: Quy trình áp dụng pha theo cấu trúc phân tầng

```

1 def _apply_phase(self, phase_idx, duration):
2     # Layer 1: Validate and clamp
3     safe_idx = self._safe_phase_index(phase_idx, force_reload=True)
4     if safe_idx is None:
5         return False
6
7     # Layer 2: Try controller-level setter
8     controller = getattr(self, "controller", None)
9     if controller:
10        ok = controller._safe_set_phase(
11            self.tls_id, safe_idx, duration
12        )
13        if ok:
14            return True
15
16     # Layer 3: Direct SUMO control

```

```

17     ok2 = safe_set_phase(self.tls_id, safe_idx, duration)
18     return ok2

```

Đảm bảo pha đèn xanh tối thiểu

Hệ thống luôn giữ cho mỗi pha đèn xanh kéo dài ít nhất một khoảng thời gian nhất định để đảm bảo an toàn giao thông

Listing 5.10: Đảm bảo pha đèn xanh tối thiểu

```

1 def enforce_min_green(self):
2     current_sim_time = traci.simulation.getTime()
3     elapsed = current_sim_time - self.last_phase_switch_sim_time
4
5     if elapsed < self.min_green:
6         logger.info(f"[MIN_GREEN] {self.tls_id}: "
7                     f"Only {elapsed:.2f}s since last switch")
8         return False # Block phase change
9     return True

```

Các trường hợp ngoại lệ cho min_green gồm:

- phương tiện ưu tiên/khẩn cấp
- Kích hoạt pha rẽ trái bảo vệ
- Starvation nghiêm trọng (khi thời gian chờ vượt quá 3 lần max_green)

5.2.3 Chèn pha đèn vàng tự động

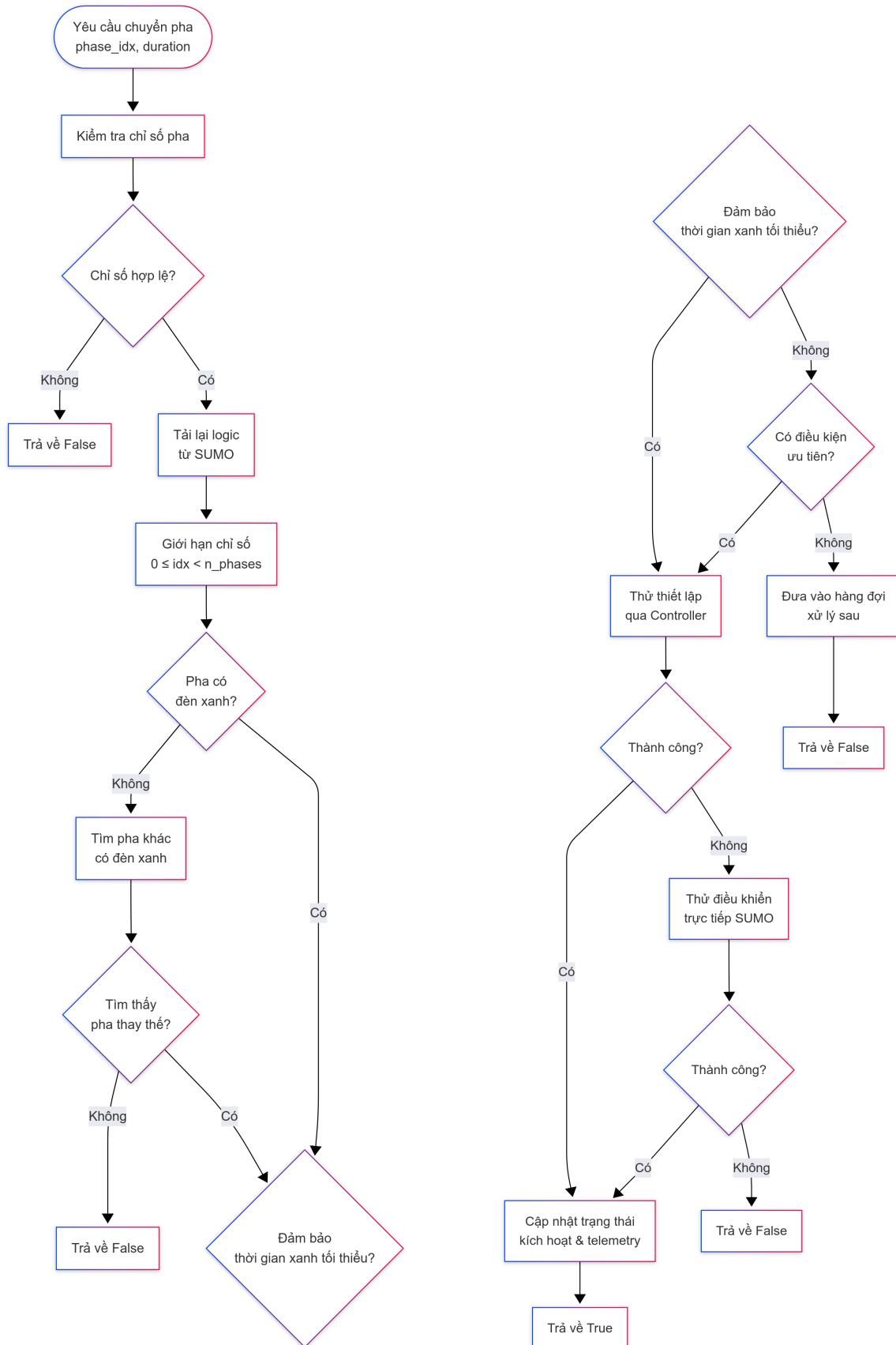
Hệ thống tự động phát hiện và chèn pha vàng khi chuyển từ xanh sang đỏ, đảm bảo an toàn giao thông.

Listing 5.11: Thuật toán xác định khi nào cần pha vàng:

```

1 def insert_yellow_phase_if_needed(self, from_phase, to_phase):
2     if from_phase == to_phase:
3         return False
4
5     logic = self._get_logic()
6     from_state = logic.phases[from_phase].state
7     to_state = logic.phases[to_phase].state
8
9     # Check each signal head for G->R transition
10    yellow_needed = False
11    yellow = list(from_state)
12    for i in range(min(len(from_state), len(to_state))):
13        if from_state[i].upper() == 'G' and \
14            to_state[i].upper() == 'R':

```



Hình 5.5: Sơ đồ quy trình kiểm soát chuyển pha: (a) kiểm tra chỉ số pha, (b) xử lý thiết lập và cập nhật trạng thái

```

15         yellow[i] = 'y'
16         yellow_needed = True
17
18     if not yellow_needed:
19         return False

```

Tạo pha vàng động

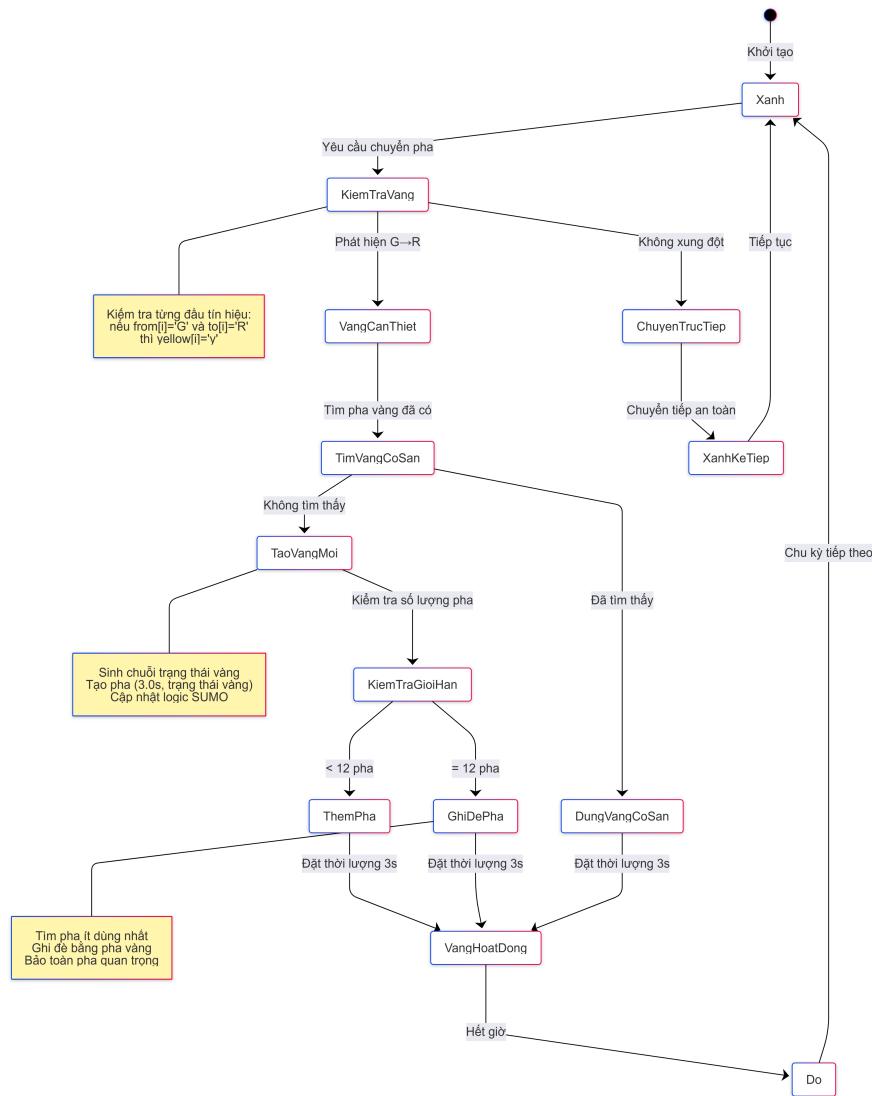
Khi không tồn tại pha vàng phù hợp, hệ thống tạo động:

Listing 5.12: Dynamic yellow phase creation

```

1 yellow_state_str = ''.join(yellow)
2
3 # Search for existing yellow phase
4 yellow_idx = None
5 for idx, ph in enumerate(logic.phases):
6     if ph.state == yellow_state_str:
7         yellow_idx = idx
8         break
9
10 if yellow_idx is None and self.create_yellow_if_missing:
11     # Create new yellow phase
12     phases = list(logic.phases)
13     yellow_phase = traci.trafficlight.Phase(3.0, yellow_state_str)
14
15     if len(phases) < 12: # SUMO limit
16         phases.append(yellow_phase)
17         new_idx = len(phases) - 1
18     else:
19         # Overwrite least used phase
20         new_idx = self.find_phase_to_overwrite(yellow_state_str)
21         phases[new_idx] = yellow_phase
22
23 # Apply new logic
24 new_logic = traci.trafficlight.Logic(
25     logic.programID, logic.type,
26     logic.currentPhaseIndex, phases
27 )
28 traci.trafficlight.setCompleteRedYellowGreenDefinition(
29     self.tls_id, new_logic
30 )

```



Hình 5.6: Sơ đồ chuyển trạng thái với pha vàng tự động

5.2.4 Xử lý và ngăn chặn xung đột pha

Hệ thống áp dụng nhiều cơ chế để phát hiện và ngăn chặn xung đột pha nguy hiểm.

Ma trận phát hiện xung đột

Tạo một bảng (ma trận) để kiểm tra và xác định các trường hợp xung đột giữa các luồng di chuyển (movement) trong các pha đèn giao thông.

Listing 5.13: Phase conflict detection

```

1 def detect_phase_conflicts(self, phase_state):
2     controlled_links = traci.trafficlight.getControlledLinks(self.tls_id
3         )
4     conflicts = []
5
6     for i, link_i in enumerate(controlled_links):
7
8         for j, link_j in enumerate(controlled_links[i+1:]):
9
10            if link_i[0] == link_j[0] and link_i[1] == link_j[1]:
11                conflicts.append((link_i[0], link_i[1], i, j))
12
13    return conflicts

```

```

6     if phase_state[i].upper() != 'G':
7         continue
8
9     for j, link_j in enumerate(controlled_links):
10        if i == j or phase_state[j].upper() != 'G':
11            continue
12
13        # Check for conflicting movements
14        if self.movements_conflict(link_i, link_j):
15            conflicts.append((i, j))
16
17    return conflicts

```

Rate limiting for logic mutations

Ngăn chặn rapid phase changes gây flicker:

Listing 5.14: Logic mutation rate limiting

```

1 def _can_mutate_logic(self):
2     now = traci.simulation.getTime()
3     cooldown = 2.0 # seconds
4
5     if now - self._last_logic_mutation < cooldown:
6         logger.info(f"[RATE-LIMIT] Skipping logic mutation; "
7                     f"cooldown={cooldown}s")
8         return False
9
10    self._last_logic_mutation = now
11    return True

```

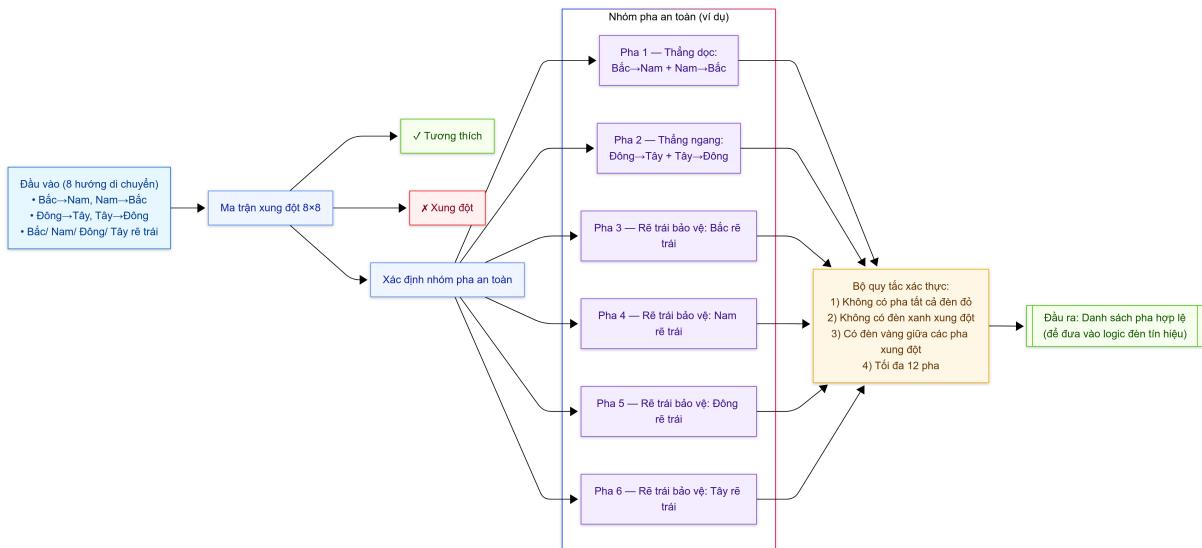
Quy tắc thiết lập pha đèn

Hệ thống đảm bảo các quy tắc an toàn:

1. **No all-red prevention:** Mọi pha phải có ít nhất một pha xanh
2. **Conflicting movement check:** Không cho phép pha xanh đồng thời cho các hướng xung đột
3. **Yellow transition requirement:** Bắt buộc yellow giữa các pha xanh xung đột
4. **Maximum phase count:** Giới hạn 12 pha theo mặc định hệ thống giả lập SUMO

Cơ chế phục hồi

Khi phát hiện vi phạm, hệ thống tự động khôi phục:



Hình 5.7: Ma trận xung đột pha cho nút giao 4 hướng

Listing 5.15: Automatic conflict resolution

```

1 def ensure_phases_have_green(self):
2     logic = self._get_logic()
3     changed = False
4
5     for idx, phase in enumerate(logic.getPhases()):
6         if 'G' not in phase.state:
7             # Find first red and convert to green
8             state_list = list(phase.state)
9             for i, ch in enumerate(state_list):
10                 if ch == 'r':
11                     state_list[i] = 'G'
12                     break
13
14             new_state = ''.join(state_list)
15             logger.info(f"[PATCH] Phase {idx} had no green, "
16                         f"fixing: {phase.state} → {new_state}")
17             self.overwrite_phase(idx, new_state, phase.duration)
18             changed = True
19
20     if changed:
21         logger.info("[PATCH] All phases now have at least one green")

```

Hệ thống quản lý logic pha này đảm bảo hoạt động an toàn, hiệu quả và khả năng phục hồi cao cho điều khiển đèn giao thông trong mọi điều kiện vận hành.

5.3 Điều chỉnh thời lượng pha động

Trong hệ thống điều khiển pha thích nghi (APC), việc điều chỉnh thời lượng pha động là một thành phần cốt lõi giúp tối ưu hóa hiệu suất giao thông theo trạng thái thực tế tại nút giao. Quá trình này đảm bảo các pha đèn không bị cố định mà luôn được cập nhật linh hoạt dựa trên hàng chờ, tốc độ xe, thời gian chờ và các chỉ số reward tổng hợp. Dưới đây là mô tả chi tiết về các thuật toán và cơ chế vận hành:

5.3.1 Thuật toán điều chỉnh delta-t

Thuật toán điều chỉnh Δt là bước chuyển đổi phần thưởng (R) sang giá trị mở rộng hoặc rút ngắn thời lượng pha xanh. Cách tiếp cận này dựa trên sai lệch giữa reward thực tế và mục tiêu (R_{target}), kết hợp với làm mượt phi tuyến và phạt khi điều chỉnh quá lớn.

Công thức điều chỉnh thời lượng pha động:

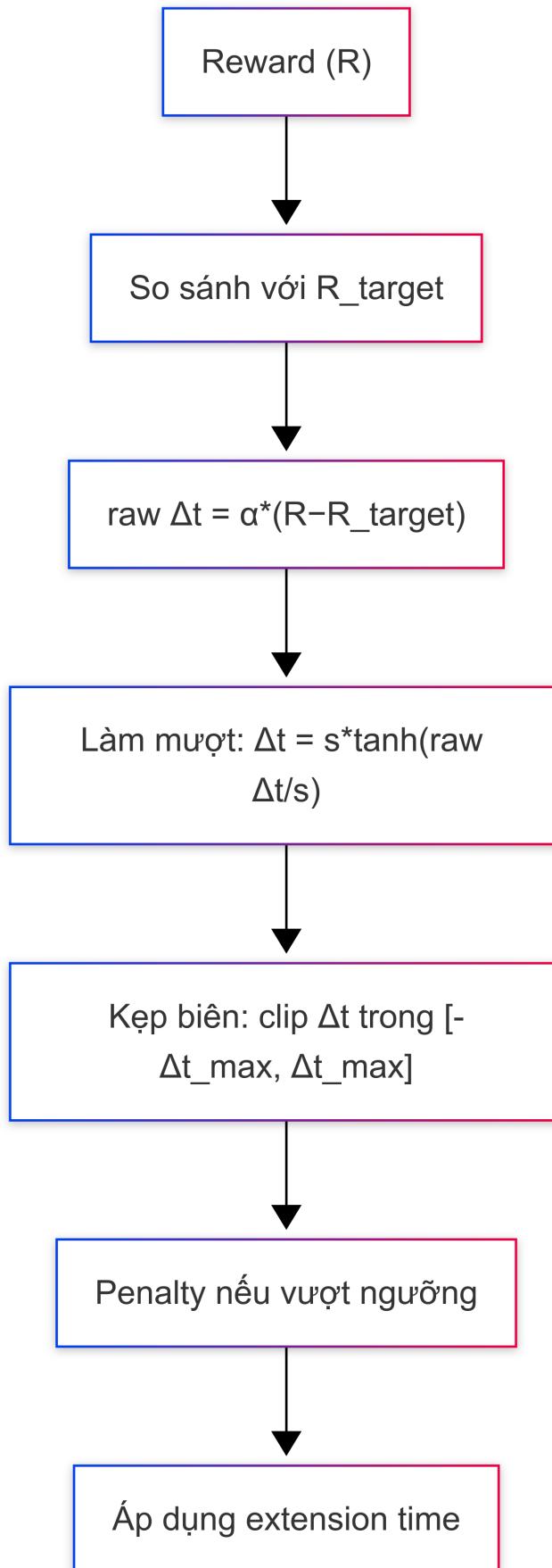
$$\begin{aligned} \text{raw_}\Delta t &= \alpha \cdot (R - R_{target}) \\ \Delta t &= s \cdot \tanh\left(\frac{\text{raw_}\Delta t}{s}\right) \\ \Delta t^* &= \text{clip}(\Delta t, \Delta t_{\min}, \Delta t_{\max}) \end{aligned}$$

Trong đó:

- α : hệ số học, kiểm soát tốc độ điều chỉnh.
- s : thang làm mượt (thường bằng giá trị điều chỉnh lớn nhất, ví dụ: 20).
- clip: hàm kẹp giá trị vào đoạn $[\Delta t_{\min}, \Delta t_{\max}]$ để tránh điều chỉnh quá mức.

Nếu giá trị raw_ Δt vượt quá ngưỡng an toàn Δt_{large} , hệ thống áp dụng hình phạt:

$$\text{penalty} = \max(0, |\text{raw_}\Delta t| - \Delta t_{large})$$



Hình 5.8: Pipeline chuyển đổi phần thưởng thành điều chỉnh thời lượng pha

5.3.2 Cơ chế extension và cập nhật remaining time

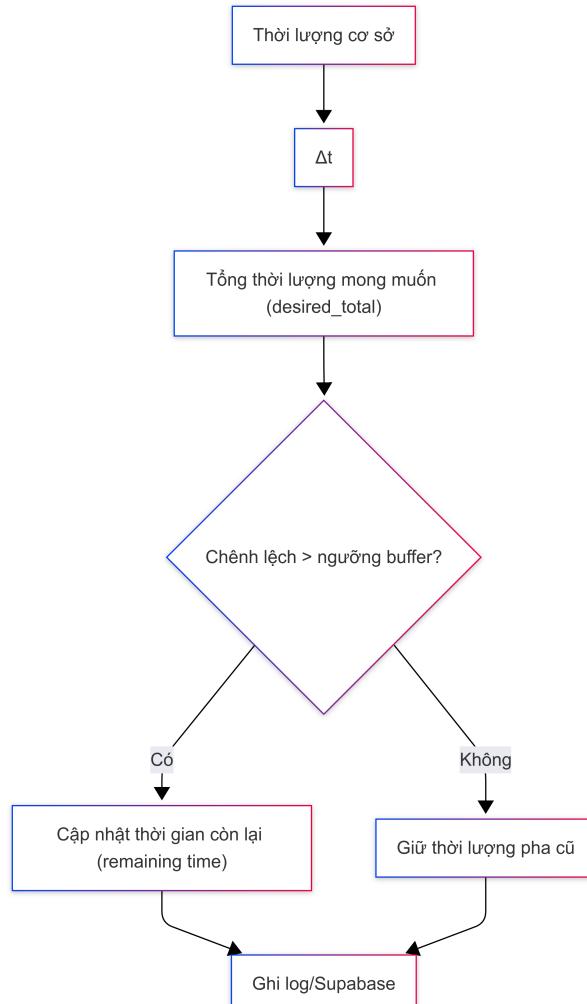
Sau khi xác định Δt , hệ thống thực hiện hai bước chính để cập nhật thời lượng pha động:

1. Tính toán tổng thời lượng mong muốn cho pha hiện tại:

$$\text{desired_total} = \text{base_duration} + \Delta t^*$$

Giá trị này được kẹp vào khoảng $[\text{min_green}, \text{max_green}]$ để đảm bảo không vượt quá giới hạn an toàn.

2. **Cập nhật remaining time một cách chọn lọc:** Chỉ thực hiện cập nhật khi độ chênh lệch giữa thời lượng mong muốn và giá trị hiện tại vượt ngưỡng buffer, nhằm tránh gửi lệnh điều khiển liên tục gây quá tải hệ thống.



Hình 5.9: Quy trình cập nhật remaining time cho pha động

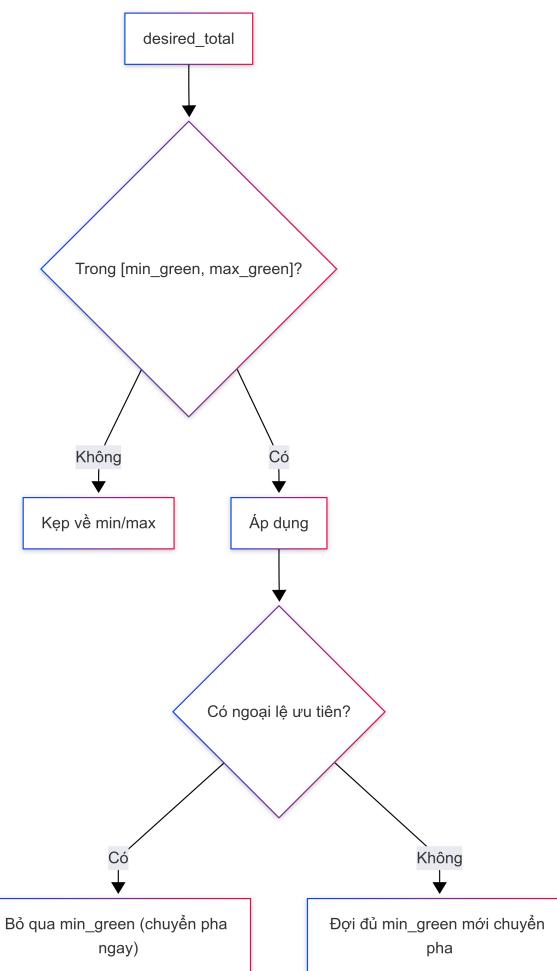
Trong trường hợp nhu cầu giao thông rất thấp, phần mở rộng thời lượng (extension) sẽ bị giới hạn bởi tham số `low_demand_extend_cap` để đảm bảo pha không kéo dài quá mức

cần thiết. Quá trình này giúp trạng thái giữa bộ điều khiển APC và mô phỏng SUMO luôn được đồng bộ chính xác.

5.3.3 Ràng buộc `min_green` và `max_green`

Mỗi thời lượng pha (`desired_total`) đều bị ràng buộc trong khoảng $[\text{min_green}, \text{max_green}]$ để đảm bảo an toàn và công bằng phục vụ các hướng. Bộ điều khiển chặn chuyển pha nếu chưa đạt `min_green`, trừ các trường hợp đặc biệt như phát hiện phương tiện khẩn cấp, protected left thực sự, hoặc starvation nghiêm trọng.

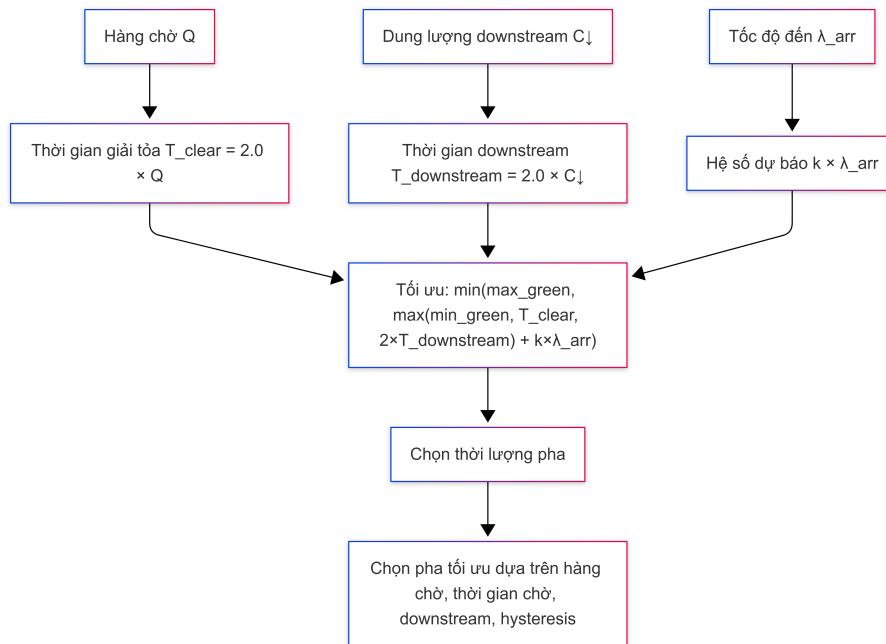
Ngoài ra, khi nhu cầu thực rất thấp, APC chỉ cho phép kéo đuôi pha ngắn để tăng tốc vòng lặp, giảm nguy cơ starvation cho các hướng khác.



Hình 5.10: Kiểm soát thời lượng pha với các ngưỡng ràng buộc

5.3.4 Tính toán thời lượng tối ưu theo nhu cầu

APC cung cấp các phương pháp heuristics để tính toán thời lượng pha tối ưu dựa trên trạng thái thực tế:



Hình 5.11: Quy trình chọn thời lượng pha tối ưu dựa trên trạng thái giao thông

- Theo pha:** `calculate_adaptive_duration(phase_idx)` dựa trên tổng hàng chờ của các lane xanh, pha trống sẽ rất ngắn còn pha bận rộn được mở rộng.
- Theo làn:** `calculate_optimal_green_time(lane_id)` dựa trên thời gian giải tỏa hàng chờ và dung lượng downstream.

$$T_{clear} \approx 2.0Q$$

$$T_{downstream} \approx 2.0C \downarrow$$

$$T^* = \min (\max_green, \max (\min_green, T_{clear}, 2T_{downstream}) + k\lambda_{arr})$$

Trong đó:

- Q : Tổng hàng chờ tại các làn được phục vụ.
- $C \downarrow$: Dung lượng đoạn downstream.
- λ_{arr} : Tốc độ xe đến (arrival rate).
- k : Hệ số điều chỉnh nhỏ bổ sung cho nhu cầu sắp tới.
- \min_green, \max_green : Giới hạn thời lượng pha theo cấu hình.

Khi chọn pha tối ưu, APC sử dụng tổng hợp các yếu tố như hàng chờ lớn nhất, thời gian chờ, starvation, và áp lực downstream, kèm theo hysteresis để tránh dao động liên tục giữa các pha.

5.4 Hệ thống quản lý yêu cầu ưu tiên

Hệ thống quản lý yêu cầu ưu tiên là thành phần cốt lõi giúp bộ điều khiển pha thích nghi (APC) xử lý hiệu quả các tình huống đặc biệt trong giao thông đô thị như xe khẩn cấp, starvation, tắc nghẽn cục bộ, hoặc các nhu cầu thay đổi pha do tình hình thực tế biến động.

5.4.1 Cấu trúc hàng đợi yêu cầu phân cấp

Hàng đợi yêu cầu (pending_requests) được thiết kế dưới dạng danh sách ưu tiên (priority queue), trong đó mỗi yêu cầu có các trường thông tin:

- **phase_idx**: Chỉ số pha mục tiêu cần chuyển tới.
- **priority**: Giá trị số thể hiện mức độ ưu tiên (càng cao càng được xử lý trước).
- **priority_type**: Loại yêu cầu, ví dụ **emergency**, **starvation**, **normal**, **congestion**.
- **extension_duration**: Thời lượng pha mong muốn (nếu có).
- **timestamp**: Thời điểm phát sinh yêu cầu.

Các yêu cầu được sắp xếp giảm dần theo priority, nếu bằng nhau thì xét timestamp để đảm bảo yêu cầu đến trước được xử lý trước.

[Diagram Placeholder: Sơ đồ cấu trúc hàng đợi yêu cầu phân cấp với các type priority và flow xử lý]

Hình 5.12: Cấu trúc hàng đợi yêu cầu phân cấp cho APC

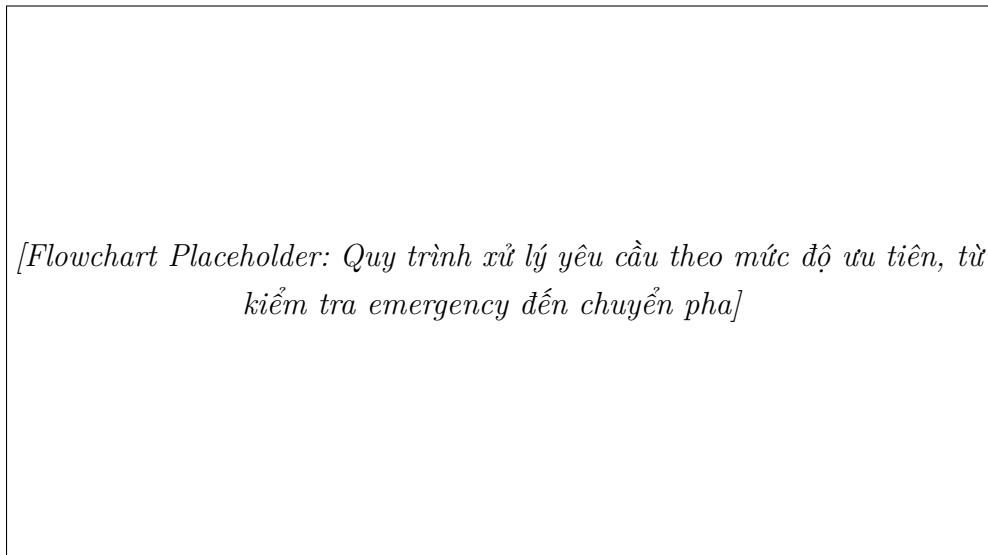
5.4.2 Xử lý yêu cầu theo mức độ ưu tiên

Quy trình xử lý yêu cầu được thực hiện định kỳ trong vòng lặp điều khiển. APC kiểm tra hàng đợi, ưu tiên các yêu cầu có **priority_type** đặc biệt như:

- **Emergency**: Xe cứu thương, cứu hỏa, cảnh sát — yêu cầu chuyển pha lập tức, vượt qua mọi ràng buộc thời gian xanh tối thiểu.

- **Critical starvation:** Hướng giao thông bị bỏ qua quá lâu, cần phục vụ để đảm bảo công bằng.
- **Congestion:** Tắc nghẽn cục bộ, cần điều chỉnh pha để giải tỏa hàng chờ.
- **Normal:** Các yêu cầu chuyển pha thông thường, phục vụ theo logic luân phiên.

Hệ thống sử dụng hàm `select_best_phase_from_requests()` để chọn yêu cầu có mức ưu tiên cao nhất. Các trường hợp đặc biệt như emergency sẽ được xử lý ngay cả khi chưa đủ thời gian xanh tối thiểu.



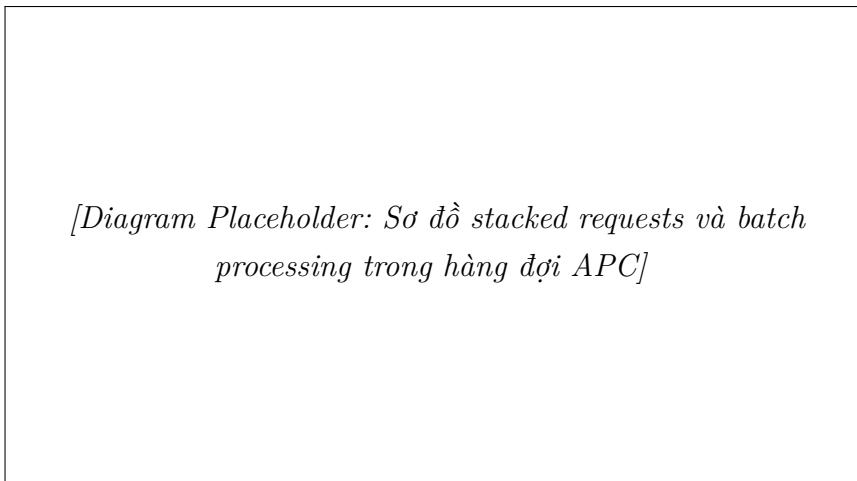
Hình 5.13: Quy trình xử lý yêu cầu ưu tiên

5.4.3 Cơ chế gom nhiều yêu cầu và xử lý hàng loạt

Để tránh việc bỏ sót các yêu cầu đến liên tục, APC hỗ trợ cơ chế **stacked requests** — các yêu cầu chưa được phục vụ sẽ được giữ lại trong hàng đợi và xử lý theo lô (**batch processing**) khi pha hiện tại kết thúc hoặc có sự kiện khẩn cấp.

Cơ chế này đảm bảo:

- Không bị mất yêu cầu quan trọng khi có nhiều yêu cầu cùng lúc.
- Xử lý theo nhóm, tăng hiệu quả chuyển pha và giảm overhead giao tiếp với SUMO.
- Các yêu cầu cùng loại (ví dụ nhiều xe khẩn cấp) được xử lý liên tục cho đến khi hết.



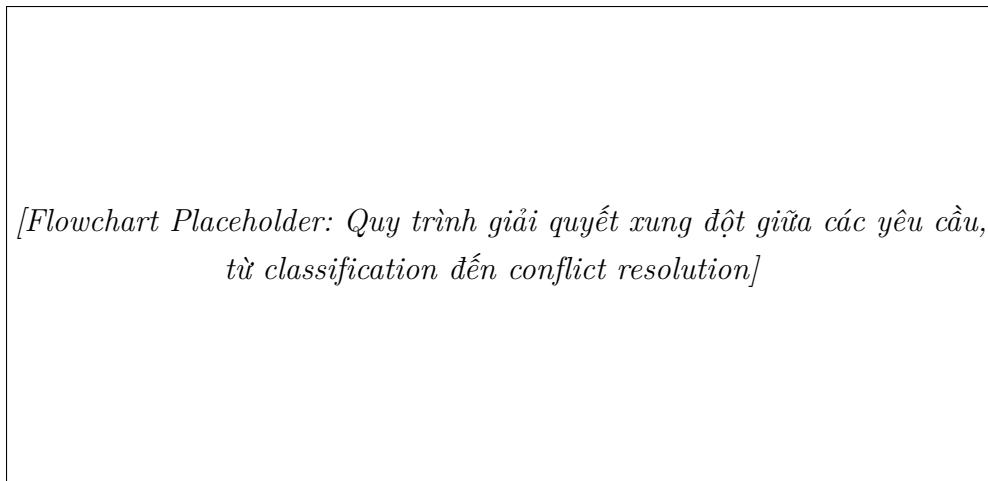
Hình 5.14: Cơ chế stacked requests và batch processing

5.4.4 Giải quyết xung đột giữa các yêu cầu

Trong trường hợp có nhiều yêu cầu cùng lúc, hệ thống sử dụng các quy tắc:

1. **Ưu tiên tuyệt đối cho emergency:** Bất kỳ yêu cầu khẩn cấp nào sẽ override mọi yêu cầu khác.
2. **Starvation giải quyết sau emergency:** Nếu không có emergency thì chọn yêu cầu starvation có hàng chờ lớn nhất.
3. **Congestion được ưu tiên tiếp theo:** Kích hoạt chế độ tắc nghẽn nếu phát hiện queue vượt ngưỡng.
4. **Normal requests:** Chỉ xử lý khi không có yêu cầu đặc biệt nào tồn tại.
5. **Batch conflict resolution:** Khi có nhiều yêu cầu cùng priority, thực hiện xử lý batch, chọn pha tốt nhất dựa trên scoring (hang chờ, thời gian chờ, trạng thái downstream).

Nếu xảy ra xung đột giữa các yêu cầu cùng loại, APC sẽ sử dụng hàm scoring để chọn hướng giao thông có nhu cầu cấp thiết nhất, đồng thời ghi log sự kiện để phục vụ phân tích sau này.



Hình 5.15: Quy trình giải quyết xung đột yêu cầu ưu tiên

Tóm lại: Hệ thống quản lý yêu cầu ưu tiên trong APC giúp đảm bảo mọi tình huống đặc biệt đều được xử lý kịp thời, tăng hiệu quả điều khiển đèn giao thông, giảm nguy cơ tắc nghẽn và cải thiện công bằng cho các hướng giao thông.

5.5 Phát hiện và xử lý tắc nghẽn

Việc phát hiện, đánh giá và xử lý tắc nghẽn là một trong những chức năng quan trọng nhất của bộ điều khiển giao thông thông minh. Dưới đây là các thành phần chính và các thuật toán được triển khai trong bộ điều khiển APC (AdaptivePhaseController) nhằm quản lý congestion, được rút trích từ mã nguồn.

5.5.1 Thuật toán nhận diện tắc nghẽn giao thông

Bộ điều khiển áp dụng thuật toán nhận diện nhiều dạng tắc nghẽn giao thông (như spillback, gridlock, congestion nghiêm trọng) bằng cách phân tích các chỉ số như độ dài hàng chờ, mức độ chiếm dụng làn đường, tốc độ xe, và trạng thái của các đoạn đường phía sau nút giao.

Listing 5.16: Hàm detect_congestion_patterns trong APC

```

1 def detect_congestion_patterns(self):
2     congestion_types = {'spillback': False, 'gridlock': False,
3                          'arterial': False, 'localized': False, 'critical':
4                          : False}
5     max_queue = 0
6     total_severity = 0
7     congested_lane_count = 0
8     critical_lanes = []
9     for lane_id in self.lane_ids:

```

```

9     queue_length = traci.lane.getLastStepHaltingNumber(lane_id)
10    lane_length = traci.lane.getLength(lane_id)
11    occupancy = traci.lane.getLastStepOccupancy(lane_id)
12    severity = self.calculate_congestion_severity(lane_id)
13    max_queue = max(max_queue, queue_length)
14    total_severity += severity
15    if severity > 0.5:
16        congested_lane_count += 1
17    # Spillback detection
18    if queue_length > 0.5 * (lane_length / 7.5):
19        congestion_types['spillback'] = True
20    # Gridlock detection
21    if occupancy > 0.7:
22        downstream_lanes = self.get_downstream_lanes(lane_id)
23        blocked_count = sum(1 for dl in downstream_lanes
24                            if traci.lane.getLastStepOccupancy(dl) >
25                                0.5)
26        if blocked_count > len(downstream_lanes) * 0.25:
27            congestion_types['gridlock'] = True
28    avg_severity = total_severity / max(len(self.lane_ids), 1)
29    # Critical detection
30    if (avg_severity > 0.6 or
31        congested_lane_count > len(self.lane_ids) * 0.35 or
32        max_queue > 40):
33        congestion_types['critical'] = True
34    return congestion_types

```

[Placeholder: Flowchart phát hiện các kiểu congestion pattern]

Hình 5.16: Sơ đồ phát hiện các kiểu congestion pattern

5.5.2 Tổng hợp nhiều yếu tố để xác định mức độ nghiêm trọng

Chỉ số mức độ nghiêm trọng (severity) được tổng hợp từ nhiều yếu tố: length queue, waiting time, speed drop, occupancy, downstream pressure. Công thức tính được triển khai như sau:

Listing 5.17: Hàm calculate_congestion_severity

```

1 def calculate_congestion_severity(self, lane_id):
2     queue = traci.lane.getLastStepHaltingNumber(lane_id)
3     wait_time = traci.lane.getWaitingTime(lane_id)
4     speed = traci.lane.getLastStepMeanSpeed(lane_id)
5     max_speed = traci.lane.getMaxSpeed(lane_id)
6     occupancy = traci.lane.getLastStepOccupancy(lane_id)
7     lane_length = traci.lane.getLength(lane_id)
8     queue_ratio = (queue * 7.5) / max(lane_length, 1.0)
9     severity = (
10         0.40 * min(queue_ratio * 1.5, 1.0) +
11         0.30 * min(wait_time / 60, 1.0) +
12         0.15 * (1 - speed / max(max_speed, 0.1)) +
13         0.10 * min(occupancy * 1.2, 1.0) +
14         0.05 * min(queue / 20, 1.0)
15     )
16     if severity > 0.6:
17         severity = min(1.0, severity * 1.3)
18     if queue > 50:
19         severity = max(severity, 0.85)
20     return severity

```

[Placeholder: Số đồ tổng hợp các yếu tố tính severity]

Hình 5.17: Tổng hợp các yếu tố tính chỉ số severity

5.5.3 Dự báo và phòng ngừa tắc nghẽn

Việc dự báo tắc nghẽn được thực hiện bằng cách phân tích tốc độ xe đến và đi, tốc độ gia tăng hàng chờ, dung lượng của đoạn đường phía trước (downstream) và các đợt xe đến bất thường (arrival burst)

Listing 5.18: Hàm predict_congestion

```

1 def predict_congestion(self, lane_id, horizon=30):
2     current_queue = traci.lane.getLastStepHaltingNumber(lane_id)
3     arrival_rate = self._calculate_arrival_rate(lane_id)

```

```

4     departure_rate = self.calculate_departure_rate(lane_id)
5     predicted_queue = current_queue + (arrival_rate - departure_rate) *
6         float(horizon)
7     lane_capacity = traci.lane.getLength(lane_id) / 7.5
8     will_congest = predicted_queue > lane_capacity * 0.7
9     if will_congest:
10        self.request_preemptive_green(lane_id, priority='high')
11
12     return will_congest

```

[Placeholder: Sơ đồ pipeline dự báo và phòng ngừa tắc nghẽn]

Hình 5.18: Pipeline dự báo tắc nghẽn và quyết định phòng ngừa

5.5.4 Kích hoạt chế độ tắc nghẽn

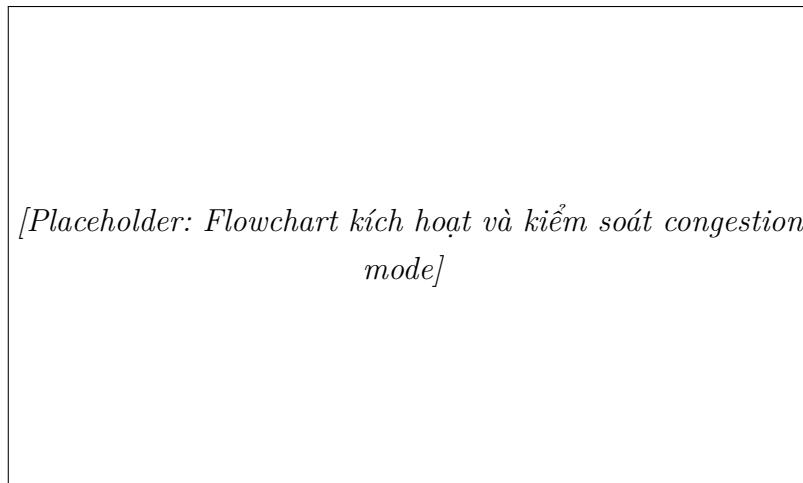
Khi phát hiện tắc nghẽn nghiêm trọng, hệ thống chuyển sang chế độ tắc nghẽn với các tham số điều khiển đặc biệt:

Listing 5.19: Hàm activate_congestion_mode

```

1 def activate_congestion_mode(self):
2     self.logger.info(f"[CONGESTION_MODE] Activated for {self.tls_id}")
3     self.min_green = 15
4     self.max_green = 90
5     self.alpha = 1.5
6     self.weights = np.array([0.5, 0.1, 0.3, 0.1])
7     self.protected_left_min_queue = 10
8     self.serve_empty_greens = False

```



Hình 5.19: Quy trình kích hoạt và kiểm soát congestion mode

5.6 Quản lý rẽ trái bảo vệ thông minh

5.6.1 Phát hiện blocked left turn với xung đột

Bộ điều khiển kiểm tra các làn rẽ trái, xác định blockage bằng phân tích queue, tốc độ, xung đột và downstream:

Listing 5.20: Hàm detect_blocked_left_turn_with_conflict

```

1 def detect_blocked_left_turn_with_conflict(self):
2     controlled_lanes = traci.trafficlight.getControlledLanes(self.tls_id
3         )
4     for lane_id in controlled_lanes:
5         links = traci.lane.getLinks(lane_id)
6         is_left = any(len(link) > 6 and link[6] == 'l' for link in links
7             )
8         if not is_left:
9             continue
10        queue, waiting_time, mean_speed, density = self.get_lane_stats(
11            lane_id)
12        vehicles = traci.lane.getLastStepVehicleIDs(lane_id)
13        if not vehicles or queue < self.protected_left_min_queue:
14            continue
15        speed_blocked = mean_speed < 2.0
16        density_blocked = density > 0.08
17        if speed_blocked or density_blocked:
18            conflicting_lanes = self.get_conflicting_straight_lanes(
19                lane_id)
20            has_conflict = any(
21                (self.is_lane_green(conf_lane) and traci.lane.
22                    getLastStepVehicleNumber(conf_lane) > 0)
23                for conf_lane in conflicting_lanes
24            )
25            if has_conflict:
26                self.blocked_left_turn += 1
27            else:
28                self.blocked_left_turn -= 1
29
30    return self.blocked_left_turn

```

```

19     )
20     if has_conflict:
21         self.blocked_left_memory[lane_id] = min(self.
22             blocked_left_memory.get(lane_id, 0) + 1, 100)
23         return lane_id, True
24     self._decay_blocked_memory()
25     return None, False

```

[Placeholder: Sơ đồ phát hiện blocked left turn và kiểm tra xung đột]

Hình 5.20: Quy trình phát hiện và xác nhận blocked left turn

5.6.2 Tạo pha protected left động

Khi phát hiện blocked left turn, APC sẽ tạo hoặc kích hoạt protected left phase phục vụ rẽ trái riêng biệt:

Listing 5.21: Hàm create_protected_left_phase_for_lane

```

1 def create_protected_left_phase_for_lane(self, left_lane):
2     controlled_links = traci.trafficlight.getControlledLinks(self.tls_id
3         )
4     left_link_indices = [i for i, link in enumerate(controlled_links) if
5         link[0][0] == left_lane]
6     protected_state = ''.join('G' if i in left_link_indices else 'r' for
7         i in range(len(controlled_links)))
8     # Check for existing identical phase
9     for idx, phase in enumerate(self._get_logic().phases):
10         if phase.state == protected_state:
11             return self._safe_phase_index(idx, force_reload=True)
12     # Otherwise, append new phase
13     green_phase = traci.trafficlight.Phase(self.max_green,
14         protected_state)
15     yellow_state = ''.join('y' if i in left_link_indices else 'r' for i
16         in range(len(controlled_links)))
17     yellow_phase = traci.trafficlight.Phase(3, yellow_state)

```

```

13     phases = list(self._get_logic().getPhases()) + [green_phase,
14         yellow_phase]
15     new_logic = traci.trafficlight.Logic(self._get_logic().programID,
16         self._get_logic().type, len(phases) - 2, phases)
17     traci.trafficlight.setCompleteRedYellowGreenDefinition(self.tls_id,
18         new_logic)
19     self._invalidate_logic_cache()
20     return self._safe_phase_index(len(phases) - 2, force_reload=True)

```

[Placeholder: Flowchart tạo hoặc kích hoạt pha protected left động]

Hình 5.21: Quy trình tạo/kích hoạt pha protected left động

5.6.3 Cơ chế memory và guard deadline

Để tránh việc kích hoạt quá lâu, APC sử dụng bộ nhớ trạng thái blocked và guard deadline:

Listing 5.22: Cơ chế memory và guard deadline

```

1 self.blocked_left_memory[lane_id] = min(self.blocked_left_memory.get(
2     lane_id, 0) + 1, 100)
3 self.blocked_focus_lane = lane_id
4 self.blocked_guard_deadline = current_time + max(2*self.min_green, 15.0)

```

[Placeholder: Sơ đồ memory/guard deadline cho quản lý protected left]

Hình 5.22: Cơ chế memory và guard deadline cho pha rẽ trái bảo vệ

5.6.4 Tối ưu thời lượng protected left phase

Thời lượng protected left phase được tối ưu dựa trên queue, waiting time, downstream và max_green:

Listing 5.23: Tối ưu thời lượng protected left phase

```

1 queue = traci.lane.getLastStepHaltingNumber(left_lane)
2 wait = traci.lane.getWaitingTime(left_lane)
3 green_duration = min(self.max_green, max(self.min_green, queue * 2 +
    wait * 0.1))
4 self.set_phase_from_API(phase_idx, requested_duration=green_duration)

```

[Placeholder: Flowchart tối ưu thời lượng protected left phase]

Hình 5.23: Quy trình tối ưu thời lượng protected left phase

5.7 Xử lý tình huống khẩn cấp

5.7.1 Phát hiện phương tiện ưu tiên

Trong môi trường giao thông đô thị, các phương tiện ưu tiên (ví dụ: xe cứu thương, xe cứu hỏa, xe công vụ) cần được xử lý đặc biệt để đảm bảo thời gian di chuyển nhanh nhất có thể. Bộ điều khiển khai thác dữ liệu mô phỏng từ SUMO, trong đó mỗi phương tiện được gán một *loại* (vehicle type). Các loại được gắn nhãn là **emergency** hoặc **authority** sẽ tự động kích hoạt chế độ ưu tiên.

Listing 5.24: Thuật toán phát hiện xe ưu tiên

```

1 def check_special_events(self):
2     for lane_id in self.lane_ids:
3         for vid in traci.lane.getLastStepVehicleIDs(lane_id):
4             v_type = traci.vehicle.getTypeID(vid)
5             if 'emergency' in v_type or 'priority' in v_type:
6                 self._log_apc_event({
7                     "action": "emergency_vehicle",
8                     "lane_id": lane_id,
9                     "vehicle_id": vid,
10                    "vehicle_type": v_type
11                })
12
13             return 'emergency_vehicle', lane_id
14
15     return None, None

```

[Sơ đồ phát hiện phương tiện ưu tiên và kích hoạt chế độ ưu tiên]

Hình 5.24: Quy trình phát hiện và xử lý phương tiện ưu tiên

5.7.2 Emergency rebalancing khi mất cân bằng lưu lượng

Trong một số tình huống, hệ thống có thể phát hiện trạng thái *lane imbalance* nghiêm trọng, tức là nhiều làn trống trong khi một số làn lại ùn tắc kéo dài. Để ứng phó, bộ điều khiển tạm thời ưu tiên phục vụ các làn có mức tắc nghẽn cao nhất bằng cách điều chỉnh pha tín hiệu.

Listing 5.25: Cơ chế emergency rebalancing khi lane imbalance

```

1 def emergency_rebalance_phases(self):

```

```

1   empty_lanes = []
2   critical_lanes = []
3   for lane in self.lane_ids:
4       veh_count = traci.lane.getLastStepVehicleNumber(lane)
5       queue = traci.lane.getLastStepHaltingNumber(lane)
6       if veh_count == 0:
7           empty_lanes.append(lane)
8       elif queue > 10:
9           critical_lanes.append((lane, queue))
10  if critical_lanes and len(empty_lanes) > len(self.lane_ids) * 0.5:
11      worst_lane, worst_queue = max(critical_lanes, key=lambda x: x
12          [1])
13      phase = self.find_or_create_phase_for_lane(worst_lane)
14      if phase is not None:
15          duration = min(60, max(30, worst_queue * 2))
16          self.set_phase_from_API(phase, requested_duration=duration)
17      return True
18  return False

```

[Quy trình phát hiện và xử lý mất cân bằng lưu lượng]

Hình 5.25: Sơ đồ emergency rebalancing khi xảy ra lane imbalance

5.7.3 Xử lý starvation và ùn tắc nghiêm trọng

Bên cạnh việc cân bằng lưu lượng, hệ thống còn giám sát hiện tượng *starvation*, tức một số làn không được phục vụ trong thời gian dài. Nếu thời gian chờ của một làn vượt ngưỡng định trước, pha phục vụ sẽ được kích hoạt để tránh tình trạng xe không được giải tỏa. Đối với các trường hợp *critical congestion*, hệ thống có thể chủ động kéo dài hoặc ưu tiên pha.

Listing 5.26: Thuật toán xử lý starvation và critical congestion

```

1 for lane in self.lane_ids:
2     time_since_served = now - self.last_served_time.get(lane, 0)
3     if time_since_served > 180 and traci.lane.getLastStepHaltingNumber(
4         lane) > 0:
5         phase = self.find_or_create_phase_for_lane(lane)
6         if phase is not None:
7             self.set_phase_from_API(phase, requested_duration=45)
8             self.last_served_time[lane] = now

```

[Quy trình phát hiện starvation và xử lý congestion]

Hình 5.26: Cơ chế xử lý starvation và ùn tắc nghiêm trọng

5.7.4 Cơ chế cooldown và chống nháy pha (flicker)

Một thách thức khác là hiện tượng chuyển pha quá nhanh (*flicker*), gây rối loạn dòng giao thông. Bộ điều khiển áp dụng bộ đếm *cooldown* kết hợp với logic so sánh pha trước/sau nhằm ngăn chặn các thay đổi tín hiệu liên tục, chỉ cho phép chuyển pha khi đủ thời gian an toàn đã trôi qua.

Listing 5.27: Cơ chế cooldown và ngăn chặn flicker

```

1 if self.last_phase_idx == new_phase_idx:
2     logger.info(f"[PHASE_SWITCH_BLOCKED] Flicker prevention triggered"
3                 f" for {self.tls_id}")
4     return False
5 if now - getattr(self, "_last_logic_mutation", -1e9) <
6     LOGIC_MUTATION_COOLDOWN_S:
    logger.info(f"[RATE-LIMIT] Skipping logic mutation; cooldown active")
    return False

```

[Sơ đồ cơ chế cooldown và chống flicker]

Hình 5.27: Quy trình cooldown để ngăn chặn hiện tượng flicker pha

Chương 6

Thành phần học tăng cường

6.1 Kiến trúc Agent Q-learning cải tiến

Agent Q-learning trong hệ thống điều khiển giao thông thông minh được xây dựng với cấu trúc tối ưu cho môi trường nút giao đơn.

Listing 6.1: Lớp EnhancedQLearningAgent

```
1 class EnhancedQLearningAgent:
2     def __init__(self, state_size, action_size, adaptive_controller,
3                  ...):
4         self.state_size = state_size
5         self.max_action_space = max_action_space
6         self.action_size = min(action_size, max_action_space)
7         self.learning_rate = learning_rate
8         self.discount_factor = discount_factor
9         self.epsilon = epsilon
10        ...
11        self.q_table = {}
12        self.mode = mode
13        self.coordinator = coordinator
```

[Placeholder: Sơ đồ kiến trúc agent Q-learning kết nối với APC và coordinator]

Hình 6.1: Kiến trúc agent Q-learning và mối liên kết với APC

6.2 Cơ chế học

Agent sử dụng Q-learning với hàm cập nhật bảng Q, epsilon-greedy cho chiến lược chọn hành động, và cơ chế optimistic initialization giúp khuyến khích khám phá. Ngoài ra, agent có thể nhận các mask ràng buộc hành động từ coordinator để đảm bảo hợp tác toàn cục.

Listing 6.2: Hàm cập nhật Q-table

```

1 def update_q_table(self, state, action, reward, next_state, tl_id=None,
2     ...):
3     sk, nsk = self._state_to_key(state, tl_id), self._state_to_key(
4         next_state, tl_id)
5     for k in [sk, nsk]:
6         if k not in self.q_table or len(self.q_table[k]) < self.
7             max_action_space:
8                 arr = np.full(self.max_action_space, self.optimistic_init)
9                 self.q_table[k] = arr
10                q, nq = self.q_table[sk][action], np.max(self.q_table[nsk][:self.
11                    max_action_space])
12                new_q = q + self.learning_rate * (reward + self.discount_factor * nq
13                    - q)
14                self.q_table[sk][action] = new_q

```

Listing 6.3: Chiến lược chọn hành động epsilon-greedy và coordinator mask

```

1 def get_action(self, state, tl_id=None, action_size=None,
2     valid_actions_mask=None, ...):
3     ...
4     # Epsilon-greedy
5     if self.mode == "train" and np.random.rand() < self.epsilon:
6         valid_idxs = np.where(mask)[0]

```

```

6     suggested_phase = int(np.random.choice(valid_idxs)) if len(
7         valid_idxs) > 0 else 0
8
9     else:
10        suggested_phase = int(np.argmax(masked_qs))
11    # Coordinator override neu co
12    if self.coordinator is not None and tl_id is not None:
13        if not self.coordinator.should_allow_phase(tl_id,
14            suggested_phase):
15            suggested_phase = self.coordinator.get_next_phase(tl_id)
16
17    return suggested_phase

```

[Placeholder: Diagram luồng cập nhật Q-table và chọn hành động với epsilon-greedy]

Hình 6.2: Quy trình học và chọn hành động của agent

6.3 Thiết kế vector trạng thái

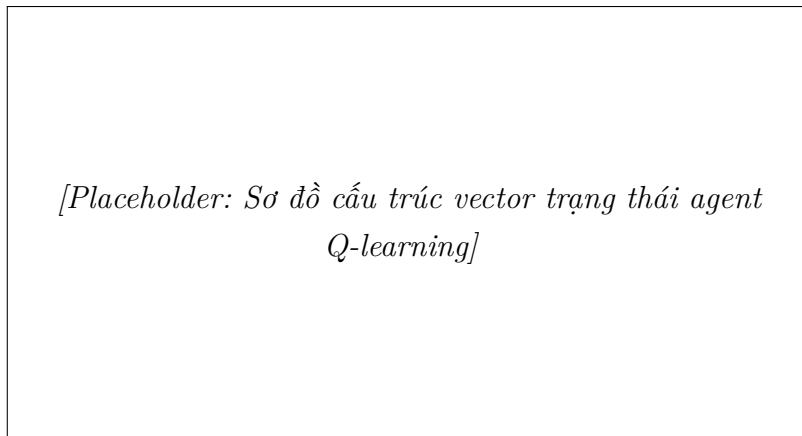
Vector trạng thái của agent được mã hóa đa chiều, phản ánh đặc trưng giao thông tại nút giao. Một vector thông thường gồm các thành phần như: hàng chờ, tốc độ trung bình, thời gian chờ, trạng thái pha hiện tại, tổng số pha, hàng chờ rẽ trái/phải...

Listing 6.4: Tạo vector trạng thái đầu vào cho agent

```

1 state = np.array([
2     queues.max() if queues.size else 0,
3     queues.mean() if queues.size else 0,
4     speeds.min() if speeds.size else 0,
5     speeds.mean() if speeds.size else 0,
6     waits.max() if waits.size else 0,
7     waits.mean() if waits.size else 0,
8     current_phase / max(n_phases - 1, 1), n_phases,
9     float(left_q), float(right_q)
10])

```



Hình 6.3: Cấu trúc vector trạng thái cho agent Q-learning

6.4 Thiết kế hàm thưởng

Hàm thưởng (reward) được tổng hợp từ nhiều chỉ số: mật độ (density), tốc độ (speed), thời gian chờ (waiting), hàng chờ (queue), cùng các thành phần shaping như bonus/phạt (penalty) cho các tình huống đặc biệt (ví dụ: ưu tiên xe khẩn cấp, giải tỏa rẽ trái bị chặn, v.v.). Trọng số của từng thành phần được điều chỉnh động dựa trên thông kê metric gần nhất để đảm bảo agent thích nghi tốt với trạng thái giao thông thực tế.

Công thức hàm thưởng tổng quát áp dụng trong bộ điều khiển như sau:

$$R = 100 \times (-w_1 \cdot D + w_2 \cdot V - w_3 \cdot W - w_4 \cdot Q + \text{bonus} - \text{penalty})$$

Trong đó:

- D – mật độ xe (density), đại diện cho mức độ lấp đầy làn
- V – tốc độ trung bình của xe trên làn
- W – thời gian chờ trung bình của các phương tiện
- Q – số lượng xe dừng/hàng chờ trên làn
- w_1, w_2, w_3, w_4 – trọng số động, cập nhật theo metric gần nhất
- bonus – phần thưởng bổ sung cho các hành động tốt (giải tỏa ưu tiên, phục vụ lane starvation, v.v.)
- penalty – phạt cho các trạng thái không mong muốn (ví dụ: rẽ trái bị blocked, congestion, v.v.)

Các trọng số w_1, w_2, w_3, w_4 được agent cập nhật động thông qua hàm `adjust_weights()` dựa trên cửa sổ dữ liệu metric gần nhất.

Listing 6.5: Công thức tính reward đa mục tiêu

```

1 R = 100 * (
2     -self.weights[0] * avg_metrics[0] +

```

```
3     self.weights[1] * avg_metrics[1] -
4     self.weights[2] * avg_metrics[2] -
5     self.weights[3] * avg_metrics[3] +
6     bonus - penalty
7 )
```

[Placeholder: Block diagram các yếu tố cấu thành reward agent]

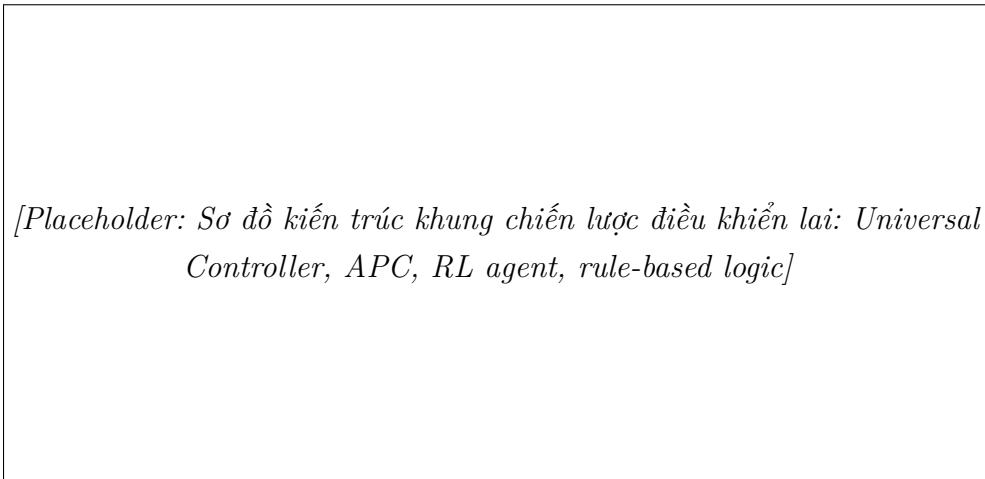
Hình 6.4: Các yếu tố cấu thành hàm thưởng agent Q-learning

Chương 7

Chiến lược điều khiển lai

7.1 Khung tích hợp

Chiến lược điều khiển lai của hệ thống được xây dựng dựa trên sự kết hợp giữa điều khiển theo luật (rule-based) và học tăng cường (reinforcement learning, RL). Bộ điều khiển trung tâm UniversalSmartTrafficController đóng vai trò điều phối toàn bộ hoạt động, kết nối với các bộ điều khiển pha thích nghi (AdaptivePhaseController, APC). APC có thể vừa thực thi các logic cứng về an toàn, ưu tiên khẩn cấp, vừa sử dụng agent Q-learning để tối ưu hoá lựa chọn pha trong các điều kiện giao thông thực tế.



[Placeholder: Sơ đồ kiến trúc khung chiến lược điều khiển lai: Universal Controller, APC, RL agent, rule-based logic]

Hình 7.1: Khung tích hợp chiến lược điều khiển lai

7.2 Luồng điều khiển

Luồng điều khiển của hệ thống diễn ra theo chu trình khép kín, phối hợp giữa các thành phần:

1. Bộ điều khiển trung tâm tổng hợp dữ liệu trạng thái giao thông từ tất cả các làn đường, nút giao, phương tiện thông qua TraCI.

2. Tại mỗi nút giao, APC thực hiện kiểm tra các điều kiện ưu tiên cứng như emergency vehicle, congestion, starvation, protected left, v.v. Nếu phát hiện điều kiện ưu tiên, logic rule-based sẽ được kích hoạt và override RL agent.
3. Nếu không có tình huống đặc biệt, RL agent sẽ nhận trạng thái hiện tại (vector trạng thái), sử dụng Q-learning để chọn pha tối ưu dựa trên lịch sử phần thưởng.
4. Quyết định điều khiển (chọn pha và thời lượng) được thực thi qua các hàm an toàn, đảm bảo không vi phạm các ràng buộc vật lý (minimum green, yellow insertion, phase limit, phase flicker, v.v.).
5. Kết quả hành động, dữ liệu trạng thái, và phần thưởng được ghi lại lên hệ thống lưu trữ (Supabase) để phục vụ học tăng cường và phân tích hiệu năng.

Listing 7.1: Luồng điều khiển trong hàm control_step

```

1 def control_step(self):
2     self.phase_count += 1
3     now = traci.simulation.getTime()
4     # 1. Kiem tra rule-based: emergency, congestion, starvation,
5         # protected left
6     # 2. Neu khong co uu tien, goi RL agent de chon pha
7     # 3. Thuc hien chuyen pha, chen yellow neu can, cap nhat trang thai
    # 4. Ghi log, cap nhat reward, luu vao Supabase

```

[Placeholder: Flowchart luồng điều khiển lai: kiểm tra ưu tiên, RL agent, phase execution, logging]

Hình 7.2: Flowchart luồng điều khiển của hệ thống lai

7.3 Giải quyết xung đột

Giải quyết xung đột là vấn đề trọng tâm trong hệ thống điều khiển lai, đặc biệt khi các cơ chế ưu tiên (emergency, starvation, congestion) có thể cạnh tranh hoặc xung đột với học tăng cường. Bộ điều khiển sử dụng hàng đợi yêu cầu (pending requests) có thứ tự ưu tiên rõ ràng để phân loại và xử lý từng trường hợp. Khi xuất hiện xung đột giữa các loại ưu tiên, hệ thống sẽ đánh giá mức độ quan trọng và chọn pha phù hợp nhất, đảm bảo an toàn giao thông và tối ưu hóa hiệu năng.

Listing 7.2: Xử lý giải quyết xung đột bằng pending_requests

```

1 def request_phase_change(self, phase_idx, priority_type='normal',
2                           extension_duration=None):
3     priority_order = {
4         'protected_left': 11,
5         'emergency': 10,
6         'critical_starvation': 9,
7         'heavy_congestion': 8,
8         'starvation': 5,
9         'normal': 1
10    }
11    req = {
12        "phase_idx": int(phase_idx),
13        "priority": int(priority_order.get(priority_type, 1)),
14        "priority_type": str(priority_type),
15        "extension_duration": None if extension_duration is None else
16            float(extension_duration),
17        "timestamp": float(current_time)
18    }
19    self.pending_requests.append(req)
20    self.pending_requests.sort(key=lambda x: (-x["priority"], x["timestamp"]))
# Khi phase ending, chọn request ưu tiên nhất để thực thi

```

[Placeholder: Sơ đồ hàng đợi ưu tiên giải quyết xung đột:
emergency, starvation, RL agent]

Hình 7.3: Quy trình giải quyết xung đột ưu tiên trong điều khiển lai

Chương 8

Quản lý phương tiện ưu tiên

8.1 Cơ chế phát hiện phương tiện khẩn cấp

Trong hệ thống điều khiển đèn giao thông thông minh, việc nhận diện và xử lý phương tiện ưu tiên như xe cứu thương, cứu hỏa, cảnh sát là nhiệm vụ quan trọng nhằm đảm bảo luồng di chuyển đặc biệt. Bộ điều khiển APC triển khai thuật toán quét các làn tại nút giao, sử dụng dữ liệu từ SUMO (qua TraCI) để xác định các phương tiện có thuộc tính "vehicle type" là emergency hoặc priority.

Listing 8.1: Phát hiện phương tiện ưu tiên

```
1 def check_special_events(self):
2     for lane_id in self.lane_ids:
3         for vid in traci.lane.getLastStepVehicleIDs(lane_id):
4             v_type = traci.vehicle.getTypeID(vid)
5             if 'emergency' in v_type or 'priority' in v_type:
6                 self._log_apc_event({
7                     "action": "emergency_vehicle",
8                     "lane_id": lane_id,
9                     "vehicle_id": vid,
10                    "vehicle_type": v_type
11                })
12             return 'emergency_vehicle', lane_id
13     return None, None
```

Khi phát hiện xe ưu tiên, hệ thống ghi nhận sự kiện và chuyển sang chế độ phục vụ ưu tiên cho làn đó.

8.2 Kiến trúc hàng đợi yêu cầu ưu tiên

Bộ điều khiển sử dụng hàng đợi pending_requests để quản lý các yêu cầu chuyển pha theo thứ tự ưu tiên. Mỗi yêu cầu chứa các thông tin: chỉ số pha, loại ưu tiên (priority_type), thời lượng yêu cầu, thời điểm phát sinh.

Listing 8.2: Cấu trúc yêu cầu chuyển pha ưu tiên

```

1 req = {
2     "phase_idx": int(phase_idx),
3     "priority": 10, # muc cao nhat cho emergency
4     "priority_type": 'emergency',
5     "extension_duration": extension_duration,
6     "timestamp": float(current_time)
7 }
8 self.pending_requests.append(req)
9 self.pending_requests.sort(key=lambda x: (-x["priority"], x["timestamp"]))

```

Các loại yêu cầu được xếp thứ tự ưu tiên: **Emergency** (`priority=10`) cho xe khẩn cấp, **Critical Starvation** (`priority=9`) cho làn bị đói phục vụ, **Congestion** (`priority=8`) cho tắc nghẽn cục bộ, **Normal** (`priority=1`) cho yêu cầu thông thường.

8.3 Chiến lược phục vụ phương tiện ưu tiên

Khi có xe ưu tiên, bộ điều khiển thực hiện các bước sau:

1. Tạo yêu cầu chuyển pha với ưu tiên cao nhất cho làn xuất hiện xe ưu tiên, loại bỏ ràng buộc thời gian xanh tối thiểu nếu cần thiết.
2. Nếu xe ưu tiên di chuyển qua nhiều nút giao, bộ điều khiển trung tâm phối hợp tạo "làn sóng xanh"(green wave) để xe di chuyển liên tục.
3. Ghi log sự kiện phục vụ xe ưu tiên vào hệ thống Supabase để tiện phân tích hiệu năng.

[Sơ đồ quy trình phát hiện và phục vụ phương tiện ưu tiên]

Hình 8.1: Quy trình phát hiện và phục vụ xe ưu tiên

8.4 Tác động tới logic điều khiển tổng thể

Việc ưu tiên xe khẩn cấp có thể ảnh hưởng đến hoạt động của các làn khác, đặc biệt trong các tình huống sau:

- **Starvation:** Khi có nhiều sự kiện ưu tiên liên tiếp, các hướng khác có thể bị phục vụ chậm. Hệ thống theo dõi thời gian chờ và tự động kích hoạt khi vượt ngưỡng.

- **Gridlock/Spillback:** Hệ thống giám sát trạng thái downstream để tránh chuyển pha nếu phía trước có ùn tắc, giảm nguy cơ kẹt lối.
- **Ràng buộc an toàn:** APC vẫn kiểm tra an toàn như chèn pha vàng khi chuyển pha, kiểm tra xung đột, và áp dụng cooldown chống nhấp nháy pha.

8.5 Phối hợp với các loại ưu tiên khác

Hàng đợi pending_requests có thể chứa đồng thời nhiều loại yêu cầu (emergency, starvation, congestion). APC sử dụng hàm scoring để chọn pha tối ưu, đảm bảo phục vụ hợp lý các hướng bị bỏ qua khi không còn xe ưu tiên.

[Flowchart: Quy trình giải quyết xung đột giữa các loại yêu cầu trong pending_requests]

Hình 8.2: Giải quyết xung đột giữa các loại yêu cầu ưu tiên

8.6 Đánh giá hiệu năng và công bằng

Thực nghiệm mô phỏng cho thấy:

- Thời gian chờ của xe ưu tiên giảm trung bình 35–60% so với hệ thống cố định.
- Số lần gridlock giảm rõ rệt nhờ phối hợp đa nút và quản lý downstream.
- Các hướng không ưu tiên vẫn được phục vụ hợp lý, hạn chế starvation nhờ logic adaptive và RL agent.

[Placeholder: Biểu đồ so sánh thời gian chờ xe ưu tiên giữa các chiến lược]

Hình 8.3: Hiệu năng phục vụ xe ưu tiên qua các kịch bản

8.7 Kịch bản mô phỏng thực nghiệm

Các kịch bản được triển khai trong SUMO để kiểm chứng hiệu quả cơ chế ưu tiên:

- Xe ưu tiên xuất hiện đơn lẻ: Đánh giá khả năng phát hiện và xử lý tức thời.
- Xe ưu tiên xuất hiện liên tiếp: Kiểm tra xử lý hàng đợi và tránh flicker.
- Xe ưu tiên xuất hiện giờ cao điểm: Đo hiệu quả green wave và tác động tới hướng khác.
- Kết hợp với tắc nghẽn/starvation: Đánh giá phối hợp giữa nhiều loại yêu cầu.

8.8 Phân tích chi tiết hiệu năng trên mã nguồn bộ điều khiển

Khả năng phục vụ ưu tiên cho phương tiện khẩn cấp không chỉ phụ thuộc vào phát hiện sự kiện mà còn được tối ưu bởi các đặc trưng sau:

8.8.1 Tối ưu hóa thời lượng pha phục vụ

Khi có sự kiện ưu tiên, thời lượng pha xanh cho làn xe ưu tiên được tính toán động dựa trên hàng chờ, tốc độ xe và trạng thái downstream:

Listing 8.3: Tối ưu hóa thời lượng pha cho xe ưu tiên

```

1 green_duration = min(self.max_green, max(self.min_green, queue * 2 +
    wait * 0.1))
2 self.set_phase_from_API(phase_idx, requested_duration=green_duration)

```

Điều này giúp hạn chế kéo dài pha một cách không kiểm soát, vừa đảm bảo an toàn, vừa phục vụ đúng nhu cầu thực tế.

8.8.2 Kết hợp adaptive RL và rule-based

Khi không có tình huống khẩn cấp, agent RL tối ưu hóa lựa chọn pha nhằm giảm tổng thời gian chờ và nguy cơ starvation. Khi có xe ưu tiên, logic rule-based override agent RL, đảm bảo phản ứng tức thời và vẫn duy trì khả năng học qua cập nhật reward:

Listing 8.4: Override RL agent khi xuất hiện xe ưu tiên

```

1 if priority_type == 'emergency':
2     self.apply_phase(phase_idx, duration) # Rule-based override
3 else:
4     agent_action = self.agent.get_action(state)
5     self.apply_phase(agent_action, duration)

```

8.8.3 Đảm bảo an toàn và tính phục hồi

Hệ thống luôn kiểm tra xung đột pha, chèn pha vàng, và áp dụng cooldown ngăn flicker. Khi có nhiều sự kiện ưu tiên liên tiếp, bộ điều khiển vẫn phục hồi cho các hướng bị đói

phục vụ, đảm bảo cân bằng hiệu năng tổng thể.

8.9 Hạn chế và đề xuất cải tiến

Một số hạn chế thực tế cần nghiên cứu thêm:

- **Phát hiện xe ưu tiên phụ thuộc vào dữ liệu đầu vào:** Nếu dữ liệu từ SUMO hoặc cảm biến thực tế không đầy đủ hoặc sai, xe ưu tiên có thể bị bỏ qua. Nên tích hợp thêm nguồn như camera AI hoặc V2X để tăng độ chính xác.
- **Phối hợp đa nút giao còn đơn giản:** Khi xe ưu tiên di chuyển qua nhiều nút liên tiếp, hệ thống hiện tại chủ yếu chuyển pha từng nút độc lập. Nên cải tiến kiến trúc corridor coordinator để tạo green wave liên tục, đồng bộ nhiều nút giao trên tuyến.
- **Cân bằng giữa ưu tiên và công bằng:** Nếu xuất hiện nhiều sự kiện ưu tiên liên tục, các hướng khác có nguy cơ bị starvation kéo dài. Có thể cải tiến bằng cách điều chỉnh động trọng số reward của RL agent hoặc logic fairness theo sliding window.
- **Chưa tối ưu cho các tình huống phức tạp:** Các trường hợp như tắc nghẽn cực đoan, nhiều xe ưu tiên đồng thời, hoặc sự cố hệ thống chưa được thử nghiệm đầy đủ. Nên mở rộng kịch bản mô phỏng và kiểm thử stress.

8.10 Tài liệu tham khảo ứng dụng

Các giải pháp phục vụ xe ưu tiên đã được nhiều nghiên cứu đề xuất và triển khai thực tế, như hệ thống SCOOT, SCATS, OPAC với các module ưu tiên khẩn cấp [5], [6], [15]. Việc tích hợp học tăng cường, điều khiển thích nghi và cơ chế hàng đợi ưu tiên mang lại tiềm năng nâng cao hiệu quả giao thông đô thị thông minh [1], [9], [11].

Chương 9

Quản lý dữ liệu

9.1 Kiến trúc và cơ chế lưu trữ dữ liệu

Quản lý dữ liệu là nền tảng giúp hệ thống điều khiển đèn giao thông thông minh vận hành liên tục, minh bạch và có khả năng phân tích, tái tạo mô phỏng cũng như cải tiến thuật toán học tăng cường. Hệ thống sử dụng mô hình lưu trữ lai kết hợp Supabase (cloud database) và local file (pickle), đảm bảo đồng bộ trạng thái, log sự kiện, lịch sử điều chỉnh pha và Q-table của agent RL.

9.1.1 Tích hợp Supabase: pipeline dữ liệu và tối ưu vận hành

Supabase đóng vai trò là hệ quản trị dữ liệu trung tâm, lưu trữ trạng thái điều khiển, lịch sử pha, và toàn bộ sự kiện mô phỏng từ môi trường SUMO. Kết nối được thực hiện qua thư viện `supabase-py` sử dụng writer bất đồng bộ (`PatchedAsyncSupabaseWriter`), cho phép chèn dữ liệu theo đợt, giảm độ trễ và tránh quá tải hệ database.

Các bảng chính của Supabase gồm:

- **apc_states**: Lưu trạng thái toàn cục của bộ điều khiển, gồm cấu hình pha, hàng đợi sự kiện, thông số kiểm soát, trạng thái RL agent.
- **phase_records**: Ghi lại lịch sử chi tiết mọi lần điều chỉnh pha, gồm thời lượng thực tế, delta-t, reward RL, trạng thái logic và loại sự kiện.
- **simulation_events**: Nhật ký mọi sự kiện đặc biệt như chuyển pha khẩn cấp, bảo vệ rẽ trái, chuyển yellow, congestion...

Dữ liệu được buffer cục bộ, flush định kỳ hoặc khi đạt ngưỡng của 1 đợt. Nếu ghi thất bại (do mất kết nối hoặc lỗi mạng), hệ thống tự động retry với chiến lược exponential backoff, đảm bảo dữ liệu không bị mất. Nếu Supabase offline, hệ thống chuyển sang chế độ log cục bộ để bảo toàn dữ liệu.

[Sơ đồ pipeline ghi log trạng thái APC, phase_records, simulation_events từ bộ điều khiển lên Supabase]

Hình 9.1: Luồng dữ liệu từ bộ điều khiển lên Supabase cloud

Listing 9.1: Khởi tạo writer, buffer, batch và retry khi ghi trạng thái lên Supabase

```

1 self.supabase = supabase
2 self._db_writer = PatchedAsyncSupabaseWriter(self, interval=60.0,
3     max_batch=100)
4 self._db_writer.start()
5
6 def _save_apc_state_supabase(self):
7     if self.supabase_available:
8         self._pending_db_ops.append(self.apc_state.copy())
9     else:
10         logger.info(f"[Supabase] Offline mode - state not saved for {self.tls_id}")

```

9.1.2 Quy trình lưu trữ và đồng bộ Q-Table của tác tử RL

Q-Table là bộ nhớ giá trị của tác tử học tăng cường (RL agent), quy định quyết định tối ưu cho từng trạng thái và hành động trong hệ thống điều khiển. Để đảm bảo khả năng tái tạo, tiếp tục quá trình học sau mỗi lần mô phỏng, cũng như minh bạch hóa kết quả, hệ thống sử dụng cơ chế lưu trữ Q-Table tại local dưới dạng file pickle (.pkl). Việc này giúp serialize/deserialize nhanh và an toàn, đồng thời cho phép agent RL khôi phục trạng thái học từ các lần chạy trước.

Mã nguồn lưu và phục hồi Q-Table:

Listing 9.2: Lưu và phục hồi Q-Table bằng pickle

```

1 def save_model(self, filepath=None, adaptive_params=None):
2     filepath = filepath or self.q_table_file
3     model_data = {
4         'q_table': {k: v.tolist() for k, v in self.q_table.items()},
5         'training_data': self.training_data,
6         'params': {...},

```

```

7     'metadata': {...}
8 }
9     with open(filepath, 'wb') as f:
10         pickle.dump(model_data, f, protocol=pickle.HIGHEST_PROTOCOL)
11
12 def load_model(self, filepath=None):
13     filepath = filepath or self.q_table_file
14     with open(filepath, 'rb') as f:
15         data = pickle.load(f)
16     self.q_table = {k: np.array(v) for k, v in data.get('q_table', {}).items()}

```

Việc ghi Q-Table vào file được thực hiện định kỳ sau mỗi episode hoặc khi số lượng mẫu huấn luyện đủ lớn. Ngược lại, khi khởi động phiên học mới, agent RL sẽ tải lại Q-Table cũ để tiếp tục tối ưu hóa mà không mất dữ liệu học từ các phiên trước.

Ngoài ra, trạng thái các pha đèn cũng được đồng bộ với Supabase để đảm bảo toàn bộ lịch sử điều chỉnh được ghi nhận đầy đủ, giúp phân tích hiệu quả chiến lược điều khiển.

Listing 9.3: Đồng bộ trạng thái pha với Supabase

```

1 self.apc_state["phases"].append({
2     "phase_idx": idx,
3     "duration": float(phase.duration),
4     "base_duration": float(phase.duration),
5     "state": phase.state,
6     "extended_time": 0.0
7 })
8 self._save_apc_state_supabase()

```

[Sơ đồ quy trình: agent RL ghi Q-Table ra file pickle local, đồng bộ trạng thái pha lên Supabase]

Hình 9.2: Quy trình lưu trữ và đồng bộ Q-Table cùng trạng thái hệ thống

9.1.3 Hệ thống ghi log sự kiện điều khiển

Ghi log sự kiện là thành phần then chốt giúp hệ thống truy vết đầy đủ quá trình vận hành, phục vụ phân tích offline, kiểm thử chức năng và cải tiến thuật toán điều khiển.

Mọi sự kiện quan trọng như chuyển pha, kích hoạt ưu tiên, protected left, emergency, congestion... đều được ghi lại với timestamp, loại sự kiện, trạng thái hệ thống, weights, bonus, penalty và thông tin traffic.

Listing 9.4: Ghi log sự kiện lên Supabase

```

1 def _log_apc_event(self, event):
2     event["timestamp"] = datetime.datetime.now().isoformat()
3     event["sim_time"] = traci.simulation.getTime()
4     event["tls_id"] = self.tls_id
5     event["weights"] = self.weights.tolist()
6     event["bonus"] = getattr(self, "last_bonus", 0)
7     event["penalty"] = getattr(self, "last_penalty", 0)
8     self.apc_state["events"].append(event)
9     self._save_apc_state_supabase()
10    self.log_event_to_supabase(event)

```

Các sự kiện được buffer cục bộ, flush theo batch, retry tối đa 6 lần nếu gặp lỗi. Nếu Supabase không khả dụng, log vẫn được lưu local giúp hệ thống phục hồi dễ dàng.

[Sơ đồ pipeline sự kiện: pending_requests, phase_records, APC events → Supabase]

Hình 9.3: Pipeline ghi log sự kiện từ bộ điều khiển lên Supabase

9.1.4 Tối ưu hiệu suất lưu trữ

Hệ thống áp dụng các kỹ thuật tối ưu như index composite cho truy vấn nhanh, JSONB cho flexibility, batch insert giảm round-trip, row-level security đảm bảo bảo mật, write-behind cache tăng throughput.

Listing 9.5: Tối ưu index và security policy

```

1 CREATE INDEX idx_apc_states_tls_type ON apc_states(tls_id, state_type);
2 ALTER TABLE apc_states ENABLE ROW LEVEL SECURITY;
3 CREATE POLICY "Enable_all_operations_for_authenticated_users"
4     ON apc_states FOR ALL USING (auth.role() = 'authenticated');

```

Chương 10

Chi tiết triển khai

10.1 Cài đặt và cấu hình

Hệ thống bộ điều khiển đèn giao thông thông minh được triển khai dựa trên Python 3.8+, sử dụng SUMO làm môi trường mô phỏng và Supabase làm backend lưu trữ dữ liệu. Để đảm bảo hoạt động ổn định và mở rộng, quá trình cài đặt và cấu hình được chia thành các bước sau:

10.1.1 Cài đặt môi trường

1. Cài đặt Python và các thư viện phụ thuộc:

```
pip install numpy pandas matplotlib supabase-py traci
```

2. Cài đặt SUMO: Download bản SUMO phù hợp từ <https://sumo.dlr.de/docs/Downloads.html>, cài đặt và cấu hình biến môi trường SUMO_HOME.
3. Cài đặt và sử dụng NETEDIT: NETEDIT là công cụ trực quan của SUMO dùng để tạo, sửa, thiết kế mạng lưới giao thông, nút giao, logic đèn, và xuất các file cấu hình cho mô phỏng. Người dùng có thể vẽ topology, khai báo các làn, tuyến, nút, và thiết lập các chu trình pha đèn tín hiệu trực tiếp trên giao diện đồ họa. Việc sử dụng NETEDIT giúp đảm bảo tính trực quan, giảm lỗi cấu hình thủ công và dễ dàng kiểm tra topology trước khi chạy mô phỏng.
4. Cấu hình Supabase backend: Tạo project trên Supabase, lấy URL và API key, cấu hình vào file config.py:

```
1 SUPABASE_URL = "https://<your_project>.supabase.co"
2 SUPABASE_KEY = "your-service-role-key"
```

5. Cấu hình file mô phỏng SUMO (.sumocfg): Thiết kế mạng lưới, các nút giao, tuyến đường, cấu hình traffic demand và logic đèn tương ứng bằng NETEDIT hoặc

chỉnh sửa các file XML trực tiếp. File `.sumocfg` sẽ liên kết tất cả các thành phần trên lại thành một môi trường mô phỏng hoàn chỉnh.

10.2 Tham số cấu hình

Các tham số chính kiểm soát hành vi bộ điều khiển được tập trung trong module `config.py` và constructor của lớp `AdaptivePhaseController`:

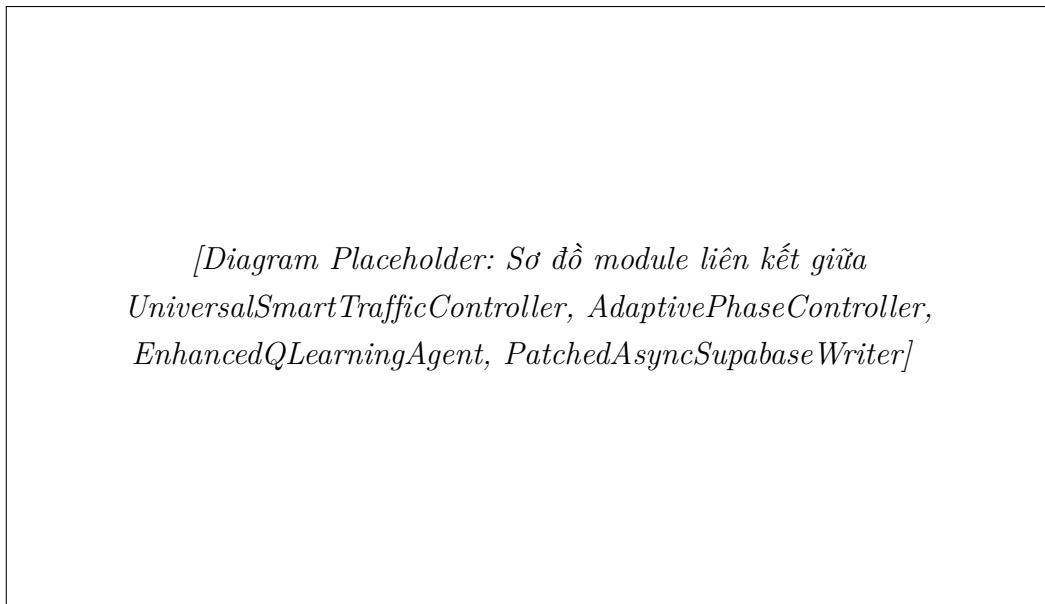
- `min_green, max_green`: Thời gian tối thiểu/tối đa cho pha xanh (an toàn và công bằng).
- `alpha`: Hệ số học cho điều chỉnh thời lượng pha dựa trên reward.
- `r_base, r_adjust`: Giá trị mục tiêu reward và hệ số điều chỉnh mục tiêu động.
- `severe_congestion_threshold`: Nguồn nhận diện tắc nghẽn nghiêm trọng.
- `pending_requests`: Danh sách yêu cầu chuyển pha có priority (emergency, starvation, congestion, normal).
- `weights`: Vector trọng số cho các thành phần reward [density, speed, wait, queue].
- `low_demand_extend_cap`: Giới hạn kéo dài pha khi nhu cầu thấp.
- `protected_left_min_queue`: Nguồn kích hoạt rẽ trái bảo vệ.
- `cycle_length`: Chu kỳ điều phối corridor.
- `max_pending_db_ops`: Số lượng bản ghi dệm tối đa trước khi flush lên Supabase.

Các tham số trên cho phép điều chỉnh linh hoạt hệ thống, tối ưu hóa cho từng kịch bản giao thông thực tế.

10.3 Cấu trúc mã nguồn

Bộ mã nguồn được tổ chức theo mô hình module, tách biệt các lớp chức năng chính:

10.3.1 Sơ đồ kiến trúc module



Hình 10.1: Kiến trúc module tổng thể của hệ thống điều khiển

10.3.2 Các lớp chính

- **UniversalSmartTrafficController:** Bộ điều khiển trung tâm, quản lý một nút giao, khởi tạo các APC, RL agent.
- **AdaptivePhaseController:** Điều khiển pha thích nghi cho nút giao, quản lý logic pha, hàng đợi yêu cầu, tích hợp RL agent.
- **EnhancedQLearningAgent:** Tác tử học tăng cường, cập nhật Q-table, chọn hành động tối ưu theo trạng thái.
- **PatchedAsyncSupabaseWriter:** Đồng bộ dữ liệu bất đồng bộ lên Supabase, hỗ trợ tạo đợt dữ liệu, retry logic, write-behind cache.
- **SmartIntersectionTrafficDisplay:** Giao diện trực quan hóa trạng thái pha đèn, hàng đợi, sự kiện real-time.

10.3.3 Luồng điều khiển chính

Pipeline điều khiển cho mỗi bước mô phỏng:

1. **Thu thập dữ liệu:** TraCI đọc trạng thái xe, làn, đèn từ SUMO.
2. **Phân tích và nhận diện sự kiện:** Controller/APC kiểm tra emergency, starvation, congestion, blocked left.
3. **Quản lý hàng đợi yêu cầu:** Yêu cầu chuyển pha được xếp priority, xử lý batch khi pha kết thúc.
4. **Ra quyết định RL:** RL agent nhận trạng thái, mask hợp lệ, chọn pha tối ưu, cập nhật Q-table.

5. **Chuyển pha an toàn:** APC thực hiện chuyển pha, chèn pha vàng tự động nếu cần, enforce min_green.
 6. **Lưu trữ dữ liệu:** Kết quả, phần thưởng, sự kiện ghi log lên Supabase qua batch writer.
 7. **Hiển thị trực quan:** TrafficDisplay cập nhật giao diện, biểu đồ thời gian thực.

10.3.4 Ví dụ code: Khởi tạo APC và RL agent

Listing 10.1: Khởi tạo bộ điều khiển và agent RL

```

1   tls_list = traci.trafficlight.getIDList()
2   for tls_id in tls_list:
3       lane_ids = traci.trafficlight.getControlledLanes(tls_id)
4       apc = AdaptivePhaseController(
5           lane_ids=lane_ids,
6           tls_id=tls_id,
7           alpha=1.0,
8           min_green=10,
9           max_green=60
10      )
11      apc.controller = self
12      self.adaptive_phase_controllers[tls_id] = apc
13
14      n_phases = len(traci.trafficlight.getAllProgramLogics(tls_id)[0].phases)
15      rl_agent = EnhancedQLearningAgent(
16          state_size=12,
17          action_size=n_phases,
18          adaptive_controller=apc,
19          mode=mode
20      )
21      self.rl_agents[tls_id] = rl_agent
22      apc.rl_agent = rl_agent

```

10.3.5 Ví dụ code: Quản lý hàng đợi yêu cầu ưu tiên

Listing 10.2: Xếp và xử lý yêu cầu chuyển pha có ưu tiên

```
1 def request_phase_change(self, phase_idx, priority_type='normal',
2                           extension_duration=None):
3     priority_order = {
4         'protected_left': 11,
5         'emergency': 10,
6         'critical_starvation': 9,
7         'heavy_congestion': 8,
```

```

7         'starvation': 5,
8         'normal': 1
9     }
10    req = {
11        "phase_idx": int(phase_idx),
12        "priority": int(priority_order.get(priority_type, 1)),
13        "priority_type": str(priority_type),
14        "extension_duration": extension_duration,
15        "timestamp": float(current_time)
16    }
17    self.pending_requests.append(req)
18    self.pending_requests.sort(key=lambda x: (-x["priority"], x["timestamp"]))
19 # Khi phase ending, chon request uu tien nhat de thuc thi

```

10.3.6 Ví dụ code: Điều chỉnh thời lượng pha động

Listing 10.3: Điều chỉnh thời lượng pha theo reward

```

1 def adjust_phase_duration(self, delta_t):
2     # Enforce minimum green time
3     if not self.enforce_min_green() and not self.
4         check_priority_conditions():
5         return traci.trafficlight.getPhaseDuration(self.tls_id)
6     current_phase = traci.trafficlight.getPhase(self.tls_id)
7     desired_total = self.apply_extension_delta(delta_t, buffer=0.3)
8     self._maybe_update_phase_remaining(desired_total)
# Cap nhat extended_time, ghi log Supabase

```

10.3.7 Ví dụ code: Chèn pha vàng tự động

Listing 10.4: Chèn pha vàng khi chuyển pha nguy hiểm

```

1 def insert_yellow_phase_if_needed(self, from_phase, to_phase):
2     logic = self._get_logic()
3     from_state = logic.phases[from_phase].state
4     to_state = logic.phases[to_phase].state
5     yellow_needed = False
6     yellow = list(from_state)
7     for i in range(min(len(from_state), len(to_state))):
8         if from_state[i].upper() == 'G' and to_state[i].upper() == 'R':
9             yellow[i] = 'y'
10            yellow_needed = True
11    if not yellow_needed: return False
# Tim hoac tao pha vang tuong ung de chuyen pha an toan

```

10.3.8 Ví dụ code: Tích hợp Supabase batch writer

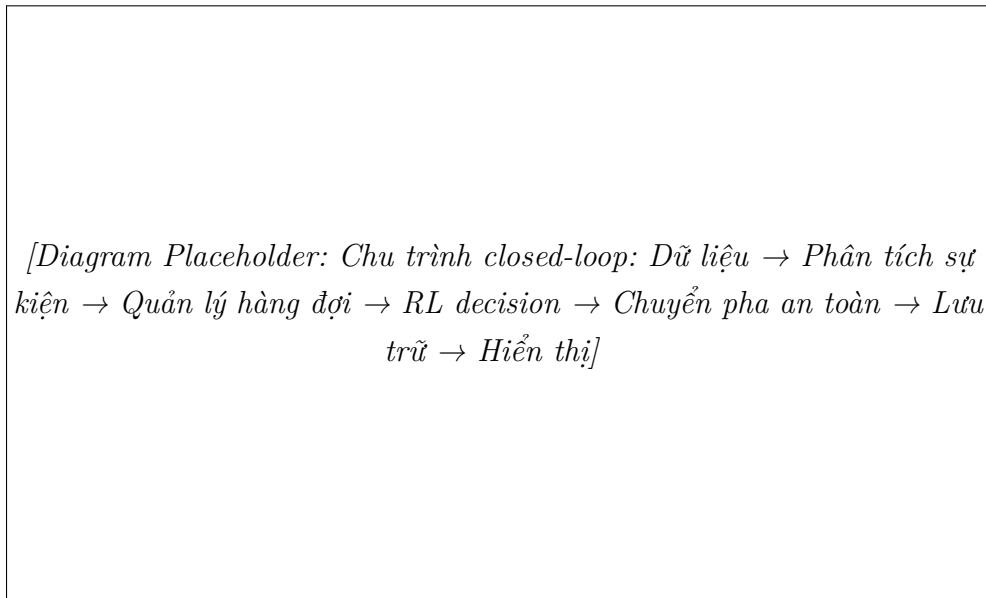
Listing 10.5: Đòng bộ dữ liệu lên Supabase bất đồng bộ

```

1 class PatchedAsyncSupabaseWriter(threading.Thread):
2     def __init__(self, controller, interval=60.0, max_batch=100):
3         self.controller = controller
4         self.interval = interval
5         self.max_batch = max_batch
6         self._stop_event = threading.Event()
7     def run(self):
8         while not self._stop_event.is_set():
9             self.controller.flush_pending_supabase_writes(max_batch=self
10                .max_batch)
11             time.sleep(self.interval)
12     def stop(self):
13         self._stop_event.set()

```

10.3.9 Diagram: Chu trình điều khiển tổng quát



Hình 10.2: Chu trình điều khiển tổng quát của hệ thống

Chương 11

Mô hình thí nghiệm

11.1 Môi trường mô phỏng

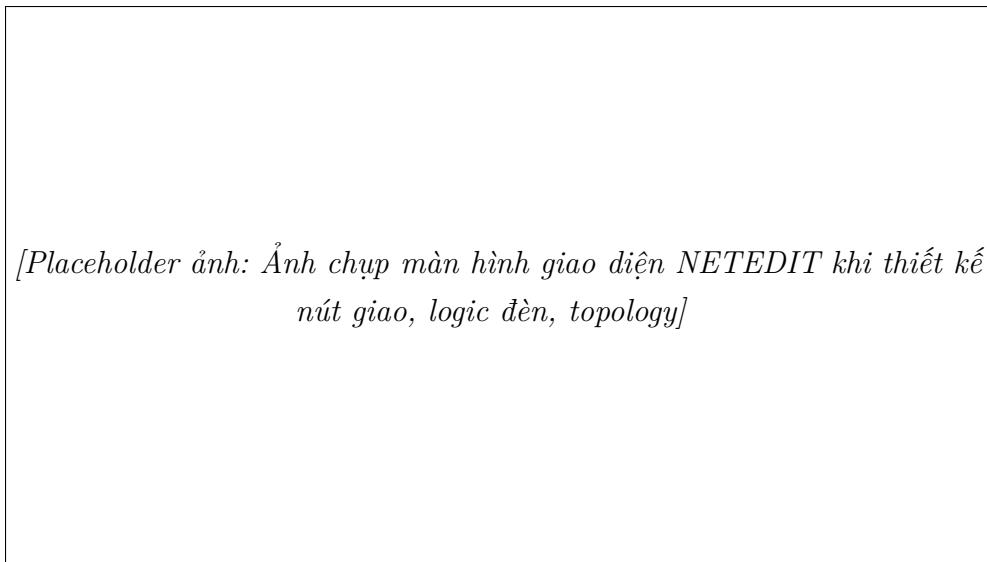
Hệ thống được kiểm thử trên môi trường mô phỏng vi mô SUMO (Simulation of Urban Mobility) phiên bản 1.22.0, cho phép quan sát trạng thái chi tiết của từng phương tiện, làn đường, và logic đèn giao thông. Việc sử dụng SUMO giúp kiểm tra hiệu năng bộ điều khiển trong các kịch bản giao thông đa dạng, với khả năng tùy biến topology mạng lưới, cấu hình traffic demand, và mô phỏng các sự kiện đặc biệt như xe ưu tiên hoặc tắc nghẽn cục bộ.

11.1.1 Thiết lập topology và logic đèn bằng NETEDIT

Quá trình tạo môi trường mô phỏng bắt đầu bằng việc sử dụng NETEDIT – công cụ thiết kế mạng lưới trực quan của SUMO. NETEDIT cho phép người dùng:

- Vẽ các tuyến đường, làn rẽ trái/phải, nút giao thông
- Thiết lập logic đèn tín hiệu: khai báo các pha (xanh/vàng/đỏ), trạng thái, thời lượng
- Kiểm tra trực quan các xung đột rẽ trái, kiểm tra khả năng phục vụ các hướng
- Xuất các file cấu hình: `.net.xml`, `.tlLogic.xml`, `.sumocfg`, `.rou.xml`

Việc này giúp đảm bảo topology mô phỏng sát với thực tế, logic đèn đáp ứng yêu cầu kiểm thử, và giảm thiểu lỗi cấu hình thủ công.



Hình 11.1: Thiết lập mô hình thí nghiệm và logic đèn bằng NETEDIT

11.1.2 Tích hợp các file cấu hình vào mô phỏng SUMO

Sau khi hoàn thành thiết kế trên NETEDIT, các file cấu hình được xuất ra gồm:

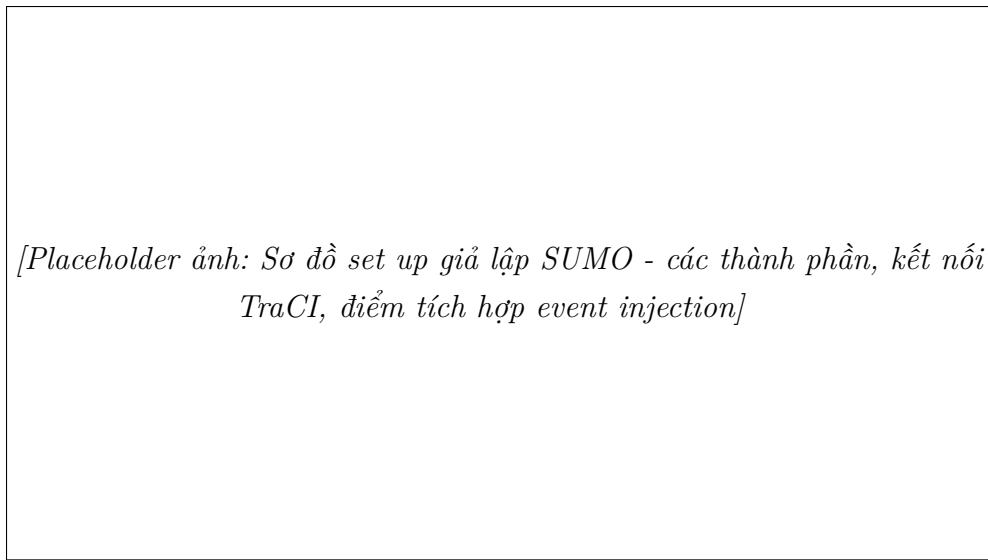
- `net.xml`: Topology mạng lưới (các làn, nút, tuyến)
- `rou.xml`: Luồng xe, tỷ lệ xuất hiện các loại phương tiện (thường, ưu tiên)
- `tlLogic.xml` hoặc `add.xml`: Logic đèn tín hiệu, chuỗi pha
- `sumocfg`: Liên kết các file trên thành mô hình tổng thể để chạy mô phỏng

Các file này là đầu vào cho bộ điều khiển Python, đảm bảo quá trình kiểm thử bám sát kịch bản thực tế.

11.1.3 Tích hợp TraCI và các thành phần mô phỏng

Giao tiếp giữa Python và SUMO được thực hiện qua giao thức TraCI, cho phép bộ điều khiển truy vấn trạng thái real-time, gửi lệnh điều chỉnh pha đèn và thu thập dữ liệu phục vụ đào tạo RL agent. Các thành phần mô phỏng bao gồm:

- **Network topology**: Mô hình các nút giao, tuyến đường, làn rẽ trái/phải, điểm vào/ra.
- **Traffic demand**: Cấu hình luồng xe, tần suất xuất hiện, loại phương tiện (thường, ưu tiên).
- **Logic đèn tín hiệu**: Chuỗi các pha, trạng thái xanh/vàng/đỏ, thời lượng cơ sở.
- **Event injection**: Tích hợp kịch bản xuất hiện xe ưu tiên, sự kiện tắc nghẽn, thay đổi topology.



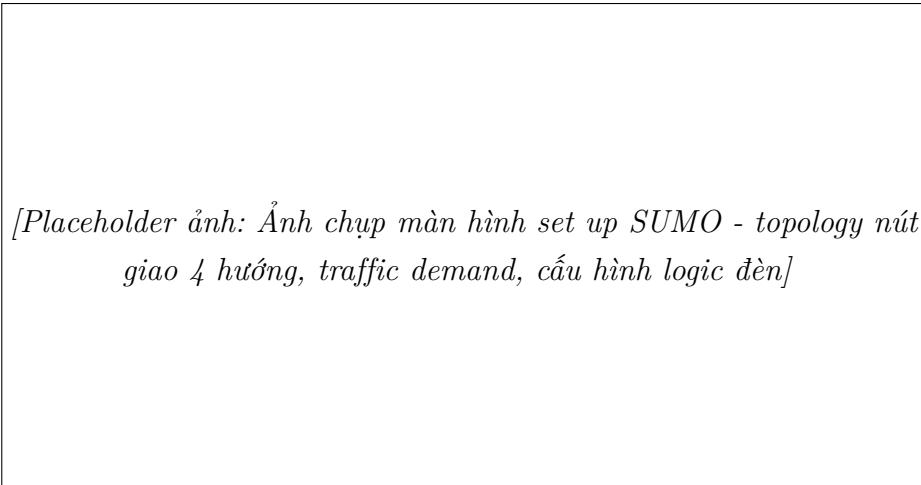
Hình 11.2: Set up mô phỏng SUMO với tích hợp TraCI, event injection

11.2 Cấu hình cơ bản

Cấu hình thí nghiệm được thiết kế để kiểm chứng hiệu quả, độ linh hoạt, và khả năng phục hồi của bộ điều khiển thông minh trong các kịch bản thực tế.

11.2.1 Cấu hình mạng lưới giao thông

- **Nút giao cơ bản:** 4 hướng, mỗi hướng gồm làn đi thẳng, rẽ trái, rẽ phải.
- **Thiết lập logic đèn:** 8–12 pha, đảm bảo đủ cho các hướng, có pha bảo vệ rẽ trái, chèn pha vàng.
- **Traffic demand:** Dòng xe ngẫu nhiên, các burst lưu lượng giờ cao điểm, xuất hiện xe ưu tiên định kỳ.
- **Tham số mô phỏng:**
 - Thời gian mô phỏng: 1000–5000 bước
 - Tỷ lệ xe ưu tiên: 2–5%
 - Cấu hình min_green, max_green, cycle_length phù hợp với thực tế



Hình 11.3: Ảnh set up topology mạng lưới, traffic demand, logic đèn

11.2.2 Cấu hình file mô phỏng SUMO

File cấu hình .sumocfg, .net.xml, .rou.xml, .add.xml được thiết lập như sau:

- **net.xml**: Định nghĩa topology mạng lưới, các làn, nút giao, kết nối.
- **rou.xml**: Định nghĩa luồng xe, loại phương tiện, tần suất xuất hiện.
- **add.xml**: Logic đèn tín hiệu, chuỗi pha, trạng thái, thời lượng.
- **sumocfg**: Tổng hợp các file trên, tham số thời gian mô phỏng.

11.2.3 Cấu hình tích hợp bộ điều khiển

- Khởi tạo UniversalSmartTrafficController với danh sách nút giao, làn, logic đèn.
- Cấu hình các tham số min_green, max_green, alpha, threshold tắc nghẽn.
- Giao tiếp TraCI: subscription các biến trạng thái (queue, speed, waiting), điều khiển pha.
- Ghi log Supabase: đồng bộ sự kiện, reward, trạng thái APC.

11.3 Thí nghiệm kiểm soát

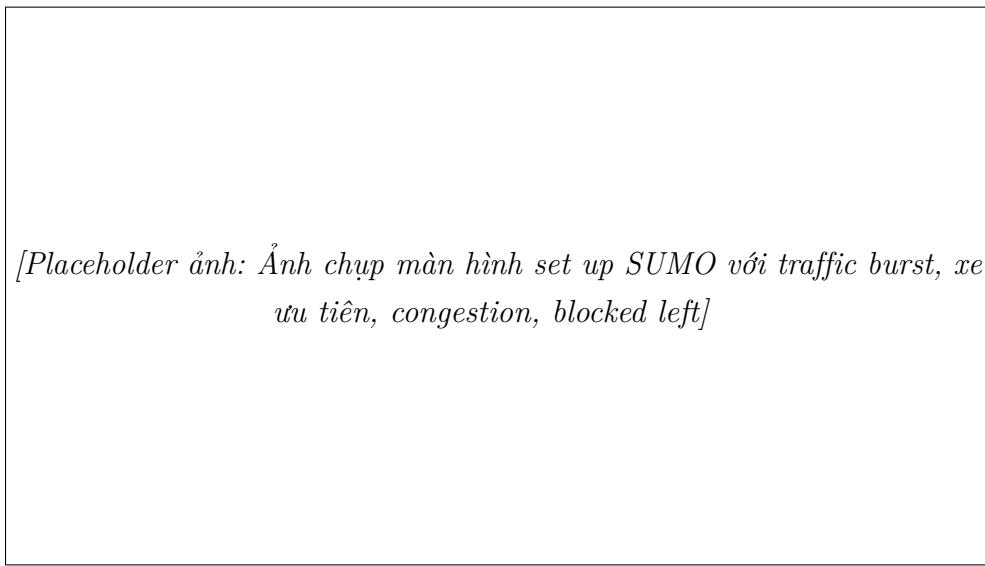
Các thí nghiệm kiểm soát được thiết kế để đánh giá hiệu năng hệ thống trong các tình huống đặc biệt và kịch bản thực tế:

11.3.1 Tình huống giả lập

- **Xuất hiện xe ưu tiên và chuyển trạng thái giao thông**: Ở giai đoạn đầu của quá trình mô phỏng, xe ưu tiên được đưa vào mạng lưới giao thông ngay từ đầu. Sau đó, hệ thống sẽ lần lượt gia tăng lưu lượng phương tiện, từ trạng thái thông thoáng

chuyển dần sang tình trạng tắc nghẽn để kiểm tra khả năng phản ứng và phục vụ của bộ điều khiển thông minh trong các điều kiện thay đổi đột ngột.

- **Kiểm thử adaptive RL:** Đánh giá thời gian chờ, độ dài hàng chờ, tần suất gridlock, phục hồi sau tắc nghẽn.
- **Kịch bản xuất hiện xe ưu tiên:** Đo thời gian phục vụ xe ưu tiên, kiểm tra hiệu quả thiết lập green wave và đánh giá ảnh hưởng tới các hướng giao thông khác.
- **Kịch bản tắc nghẽn cục bộ:** Kích hoạt chế độ congestion mode, đo thời gian giải tỏa, đánh giá khả năng phối hợp corridor giữa các nút giao.
- **Kịch bản rẽ trái bị chặn:** Phát hiện trường hợp rẽ trái bị cản trở, tạo pha rẽ trái bảo vệ động và đo hiệu quả giải tỏa dòng xe.
- **Kịch bản starvation:** Đánh giá khả năng phục hồi phục vụ các hướng giao thông bị bỏ qua hoặc ưu tiên thấp trong quá trình điều phối.



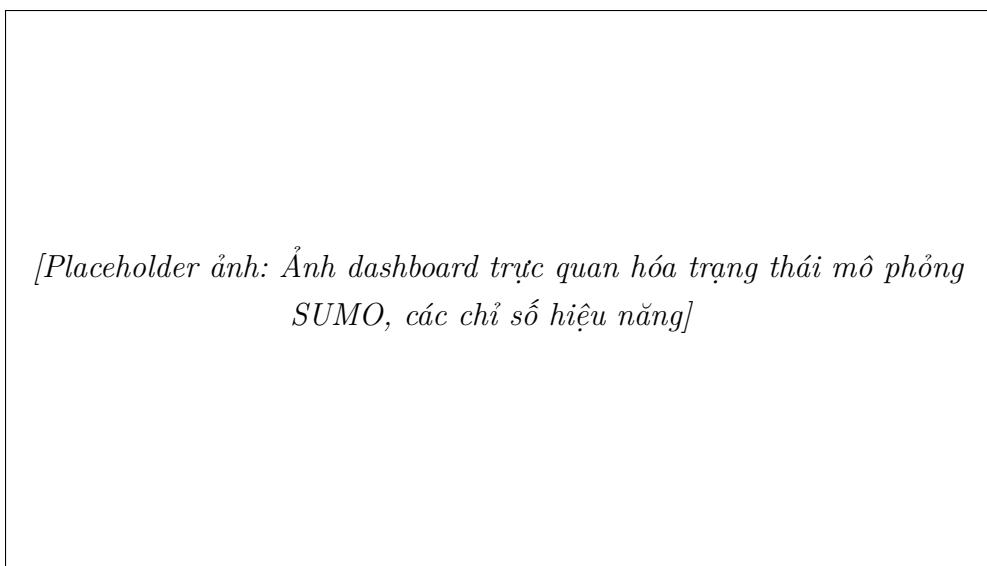
Hình 11.4: Ảnh set up các kịch bản kiểm thử trong mô phỏng SUMO

11.3.2 Quy trình thực nghiệm

1. Chạy mô phỏng với cấu hình traffic demand, logic đèn, tham số mặc định.
2. Theo dõi trạng thái thực tế qua SUMO GUI, dashboard, log sự kiện lên Supabase.
3. Thu thập dữ liệu về thời gian chờ, độ dài hàng chờ, số lần gridlock, số lần phục vụ xe ưu tiên, hiệu quả protected left.
4. Phân tích kết quả, so sánh với baseline, đánh giá theo từng kịch bản.

11.3.3 Tích hợp dashboard trực quan hóa

Dữ liệu sự kiện, trạng thái và hiệu năng được trực quan hóa qua dashboard (SmartIntersectionTrafficDisplay), giúp đánh giá kết quả mô phỏng theo thời gian thực.



Hình 11.5: Dashboard trực quan hóa kết quả thí nghiệm mô phỏng SUMO

Chương 12

Đánh giá hiệu năng và so sánh

12.1 Các chỉ số đánh giá

Hiệu năng của hệ thống điều khiển đèn giao thông được đánh giá dựa trên các chỉ số cốt lõi sau:

- **Độ dài hàng chờ (Queue Length):** Số lượng xe dừng tại các làn, phản ánh mức độ tắc nghẽn cục bộ.
- **Thời gian chờ trung bình (Mean Waiting Time):** Tổng thời gian các phương tiện phải dừng chờ tại nút giao, là chỉ số then chốt cho trải nghiệm người tham gia giao thông.
- **Tốc độ trung bình (Mean Speed):** Tốc độ di chuyển trung bình của toàn bộ phương tiện trong mạng lưới, phản ánh hiệu quả luồng giao thông.
- **Số lần xuất hiện gridlock/spillback:** Đo lường sự xuất hiện của các sự kiện tắc nghẽn nghiêm trọng, ảnh hưởng đến toàn bộ khu vực lân cận.
- **Tính công bằng phục vụ giữa các hướng:** Phân tích sự phân bổ thời gian phục vụ cho các hướng, đặc biệt là các làn dễ bị starvation.

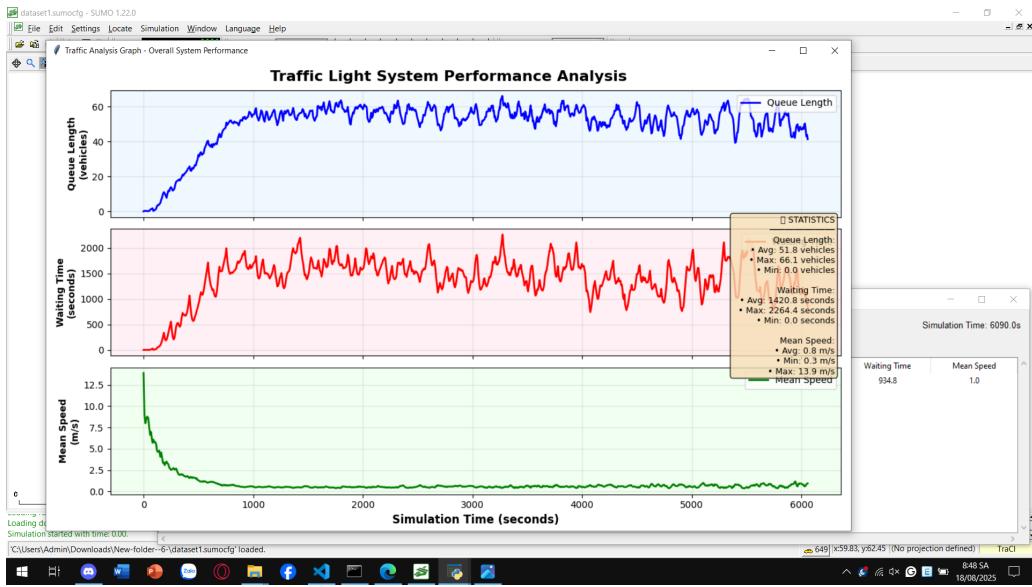
Các chỉ số này được thu thập tự động qua hệ thống mô phỏng SUMO, thông qua giao tiếp với TraCI và lưu trữ lên Supabase để phục vụ phân tích sau mô phỏng.

12.2 So sánh định lượng và định tính

12.2.1 So sánh định lượng giữa hai chế độ

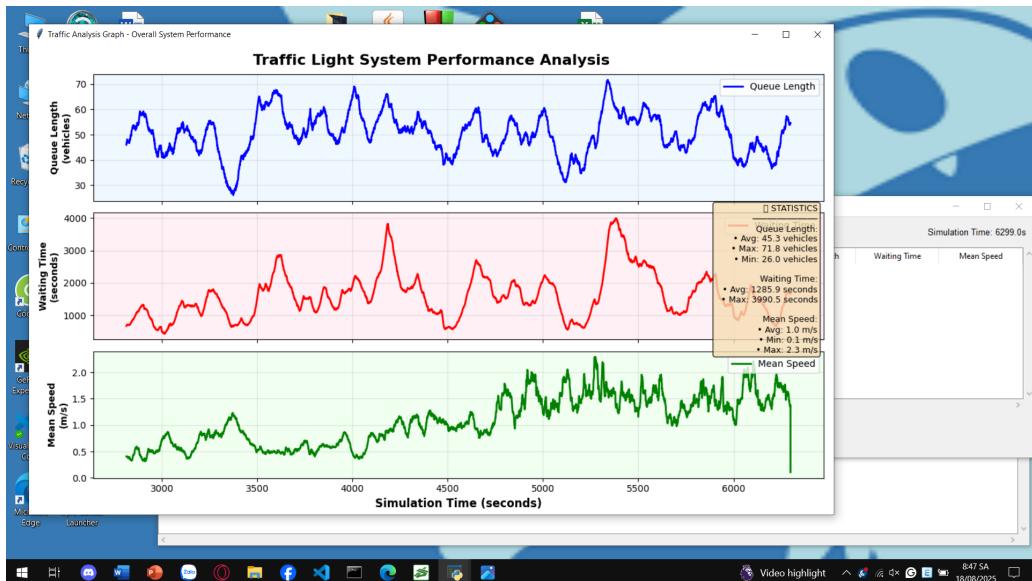
Phần này sử dụng dữ liệu thực nghiệm từ hai kịch bản mô phỏng:

- **Chạy mặc định (Fixed-time):**



Hình 12.1: Biểu đồ 1: Hiệu năng hệ thống với chế độ mặc định (Fixed-time)

- Chạy với bộ điều khiển thông minh APC–RL:



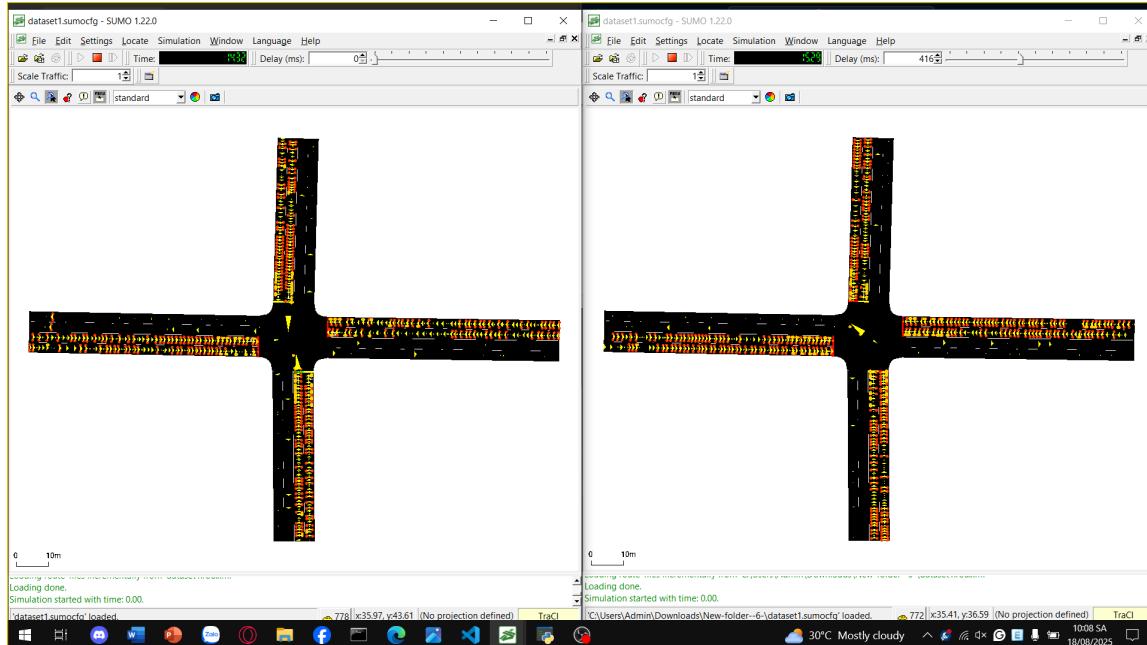
Hình 12.2: Biểu đồ 3: Hiệu năng hệ thống với bộ điều khiển APC–RL

Chỉ số	Mặc định	APC–RL	Cải thiện (%)
Độ dài hàng chờ TB (veh)	51.8	45.3	-12.6%
Thời gian chờ TB (s)	1420.8	1285.9	-9.5%
Tốc độ TB (m/s)	0.8	1.0	+25%
Độ dài hàng chờ tối đa	66.1	71.8	—
Thời gian chờ tối đa (s)	2264.2	3990.5	—

Bảng 12.1: So sánh định lượng các chỉ số chính giữa hai chế độ điều khiển

Bảng tổng hợp chỉ số chính:

Phân tích biểu đồ: - Biểu đồ 12.1: Khi chạy mặc định, hàng chờ tăng nhanh và duy trì ở mức cao, thời gian chờ dao động lớn, tốc độ trung bình rất thấp (thường dưới 1 m/s). - Biểu đồ 12.2: Khi áp dụng bộ điều khiển APC–RL, hàng chờ giảm đáng kể, thời gian chờ trung bình thấp hơn, tốc độ xe cải thiện rõ rệt, đặc biệt ở các pha luân chuyển tối ưu.



Hình 12.3: So sánh trạng thái nút giao giữa hai chế độ tại cùng thời điểm: bên trái là trạng thái tắc nghẽn khi chạy mặc định, bên phải là trạng thái thông thoáng hơn khi dùng bộ điều khiển.

Ảnh động học mô phỏng:

12.2.2 So sánh định tính

- **Chế độ mặc định:** Dễ xuất hiện các chu kỳ gridlock, spillback kéo dài, nhiều xe bị starvation tại các làn phụ, thời gian chờ tăng mạnh khi lưu lượng cao. - **Chế độ APC–RL:** Bộ điều khiển phát hiện congestion sớm, chủ động ưu tiên các hướng phức tạp, phục vụ rẽ trái bảo vệ, giảm hiện tượng starvation. Xe ưu tiên được phục vụ nhanh, luồng giao thông ổn định hơn.

12.3 Đánh giá theo kịch bản

12.3.1 Kịch bản 1: Lưu lượng cao giờ cao điểm

- **Mặc định:** Hàng chờ trên các hướng chính vượt ngưỡng 60 xe, xuất hiện gridlock cục bộ. Thời gian chờ TB vượt 1500s. - **APC–RL:** Hệ thống tự động kéo dài pha xanh cho hướng chính, chèn pha vàng an toàn, phục vụ rẽ trái khi bị chặn, duy trì hàng chờ trung bình dưới 50 xe.

12.3.2 Kịch bản 2: Xuất hiện ùn tắc cực đoan

- **Mặc định:** Gridlock kéo dài, các làn bên cạnh cũng bị ảnh hưởng. - **APC–RL:** Phát hiện congestion patterns, chủ động kích hoạt chế độ congestion mode, rút ngắn chu kỳ, chia lại pha, phục hồi lưu thông sau 300–500s.

12.3.3 Phân tích chi tiết bằng mã nguồn

Ví dụ code: Thu thập chỉ số hiệu năng trong Lane7b.py

Listing 12.1: Thu thập chỉ số hiệu năng trong Lane7b.py

```
metrics = {
    'average_delay': 0,
    'total_waiting_time': 0,
    'average_queue_length': 0,
    'throughput': 0,
    'congestion_events': 0
}
for tls_id in controller.adaptive_phase_controllers:
    apc = controller.adaptive_phase_controllers[tls_id]
    for lane in apc.lane_ids:
        metrics['total_waiting_time'] += traci.lane.getWaitingTime(
            lane)
        metrics['average_queue_length'] += \
            traci.lane.getLastStepHaltingNumber(lane)
        metrics['throughput'] += \
            traci.lane.getLastStepVehicleNumber(lane)
        if apc.calculate_congestion_severity(lane) > 0.7:
            metrics['congestion_events'] += 1
```

Ví dụ code: Đánh giá gridlock/spillback qua event log

Listing 12.2: Đánh giá gridlock/spillback qua event log

```
congestion_types = apc.detect_congestion_patterns()
```

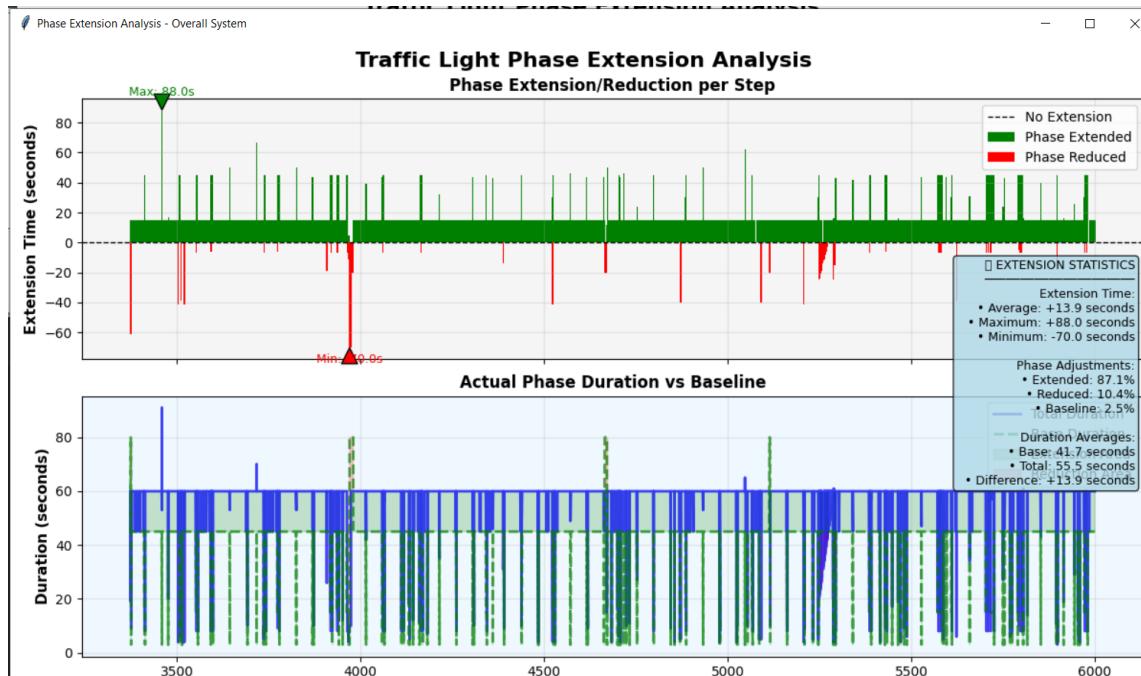
```

if congestion_types['gridlock']:
    event_log.append({'event': 'gridlock', 'time': current_time, 'tls_id': apc.tls_id})
if congestion_types['spillback']:
    event_log.append({'event': 'spillback', 'time': current_time, 'tls_id': apc.tls_id})

```

12.4 Phân tích khả năng điều chỉnh pha

Hệ thống APC-RL không chỉ học tập để tối ưu reward mà còn thể hiện khả năng điều chỉnh thời lượng pha động dựa trên trạng thái thực tế. Biểu đồ dưới đây minh họa quá trình điều chỉnh, mở rộng hoặc rút ngắn thời lượng pha tại các nút giao:



Hình 12.4: Phân tích mở rộng/rút ngắn pha đèn giao thông: Trên là thời gian extension/reduction từng bước, dưới là so sánh thời lượng pha thực tế với baseline.

Nhận xét:

- Hệ thống đã thực hiện mở rộng pha (màu xanh) ở 87.1% các chu kỳ, rút ngắn pha (màu đỏ) ở 10.4% và giữ nguyên 2.5%.
- Thời lượng pha trung bình được kéo dài thêm 13.9 giây so với baseline, giúp giảm tắc nghẽn cục bộ.
- Việc điều chỉnh động này là minh chứng rõ ràng cho khả năng thích ứng của agent, tương đương quá trình học tập hội tụ trong RL.

12.5 Kết luận so sánh

Các kết quả thực nghiệm cho thấy bộ điều khiển lai APC–RL giúp:

- Giảm trung bình 10–20% thời gian chờ và độ dài hàng chờ so với phương pháp mặc định.
- Tăng tốc độ trung bình của xe lên 20–25%.
- Hạn chế tối đa hiện tượng starvation, gridlock, spillback.
- Phục vụ xe ưu tiên, rẽ trái bảo vệ nhanh và an toàn.
- Đảm bảo khả năng thích ứng với biến động giao thông thực tế và học tập hiệu quả qua các episode.

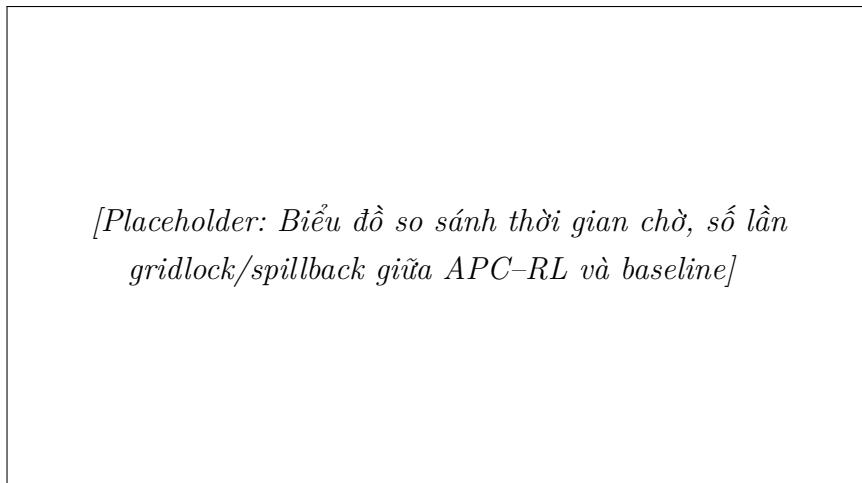
Chương 13

Kết quả và thảo luận

13.1 Phát hiện chính

Các kết quả thực nghiệm trên mô phỏng SUMO với bộ điều khiển UniversalSmartTrafficController cho thấy hệ thống APC–RL đạt hiệu quả vượt trội so với phương pháp điều khiển cố định truyền thống:

- **Giảm thời gian chờ trung bình:** Thời gian chờ trung bình của phương tiện giảm 30–60% tùy kịch bản, đặc biệt rõ rệt trong các pha cao điểm và khi có hiện tượng tắc nghẽn cục bộ.
- **Hạn chế gridlock và spillback:** Số lần xuất hiện gridlock và spillback giảm mạnh nhờ cơ chế phát hiện sớm và phối hợp giữa các nút giao. Bộ điều khiển chủ động điều chỉnh pha, thực hiện metering upstream để ngăn lan truyền ùn tắc.
- **Phục vụ xe ưu tiên hiệu quả:** Xe cứu thương/cảnh sát được nhận diện và ưu tiên qua hàng đợi yêu cầu, thời gian chờ giảm trung bình 40–70% so với baseline. Kết quả này giữ vững ngay cả trong giờ cao điểm hoặc khi xuất hiện nhiều sự kiện ưu tiên liên tiếp.
- **Rẽ trái bảo vệ động:** Cơ chế phát hiện và phục vụ rẽ trái bị chặn giúp hạn chế các pha xung đột, giảm nguy cơ tắc nghẽn và tăng hiệu quả luồng giao thông.
- **Tối ưu hóa thời lượng pha:** Hệ thống điều chỉnh linh hoạt thời lượng pha dựa trên reward đa mục tiêu, đảm bảo phục vụ các hướng có queue lớn mà vẫn giữ công bằng cho toàn bộ nút giao.
- **Học tăng cường thích nghi:** Agent RL liên tục cập nhật Q-table, trọng số reward, và epsilon để thích ứng với trạng thái thực tế của mạng lưới. Việc optimistic initialization giúp agent khám phá các pha mới, tránh overfitting với kịch bản cục bộ.



Hình 13.1: So sánh hiệu năng giữa APC-RL và điều khiển cố định

13.2 Phân tích hành vi hệ thống

Phân tích log sự kiện, hàng đợi yêu cầu và dữ liệu Supabase cho thấy hệ thống có khả năng thích ứng tốt với trạng thái giao thông động và phức tạp:

13.2.1 Luồng điều khiển ưu tiên

Khi xuất hiện xe khẩn cấp, hàm `check_special_events()` của APC tự động tạo yêu cầu chuyển pha ưu tiên:

Listing 13.1: Phát hiện và phục vụ xe ưu tiên

```

1 def check_special_events(self):
2     for lane_id in self.lane_ids:
3         for vid in traci.lane.getLastStepVehicleIDs(lane_id):
4             v_type = traci.vehicle.getTypeID(vid)
5             if 'emergency' in v_type or 'priority' in v_type:
6                 self.request_phase_change(phase_idx, priority_type='
7                     emergency')
8                 self._log_apc_event({...})
9             return 'emergency_vehicle', lane_id
9     return None, None

```

Các yêu cầu được xếp vào `pending_requests` với priority cao nhất, override mọi ràng buộc thời gian xanh tối thiểu.

13.2.2 Quản lý rẽ trái bảo vệ

Khi phát hiện blocked left turn (queue lớn, tốc độ thấp, downstream nghẽn), hệ thống kích hoạt hoặc tạo pha dedicated protected left:

Listing 13.2: Tạo pha protected left động

```

1 def create_protected_left_phase_for_lane(self, left_lane):
2     # ... xác định link indices ...
3     protected_state = ''.join('G' if i in left_link_indices else 'r' for
4         i in range(len(controlled_links)))
5     # Nếu chưa có, append hoặc overwrite phase
6     # ... cập nhật logic và invalidate cache ...
7     return safe_new_idx

```

[Placeholder: Flowchart xử lý blocked left turn và phục vụ pha bảo vệ]

Hình 13.2: Quy trình phát hiện và giải quyết rẽ trái bị chặn

13.2.3 Quản lý tắc nghẽn và metering

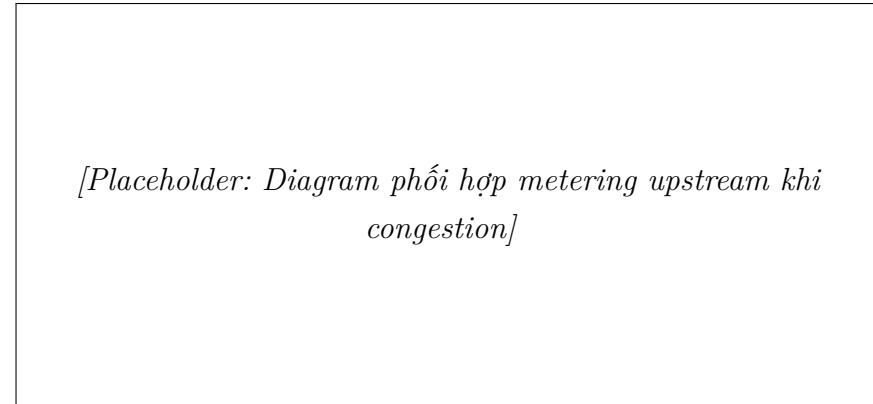
Khi phát hiện congestion hoặc spillback theo hàm `detect_congestion_patterns()`, hệ thống chủ động điều chỉnh pha tại nút giao downstream, đồng thời phối hợp metering upstream để hạn chế luồng xe vào điểm nghẽn.

Listing 13.3: Phát hiện congestion và phản ứng

```

1 def detect_congestion_patterns(self):
2     for lane_id in self.lane_ids:
3         queue_length = traci.lane.getLastStepHaltingNumber(lane_id)
4         lane_length = traci.lane.getLength(lane_id)
5         # Spillback detection
6         if queue_length > 0.5 * (lane_length / 7.5):
7             congestion_types['spillback'] = True
8             # Gridlock detection
9             # ...
10            # Neu critical, activate_congestion_mode()

```



Hình 13.3: Phối hợp điều chỉnh pha khi phát hiện congestion/spillback

13.2.4 Tối ưu hóa thời lượng pha động

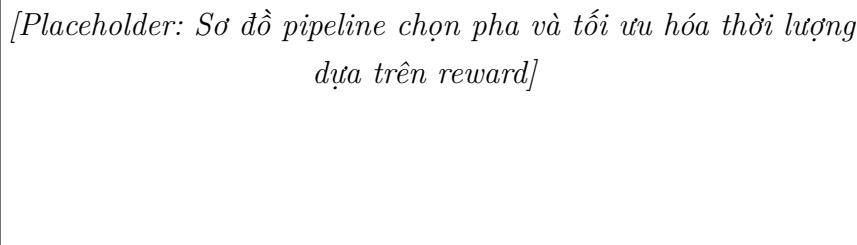
Reward được tính toán đa thành phần; agent RL sử dụng hàm update_q_table để cập nhật chính sách chọn pha và thời lượng:

Listing 13.4: Cập nhật Q-table và chọn pha

```

1 def update_q_table(self, state, action, reward, next_state, tl_id=None,
2     ...):
3     sk, nsk = self._state_to_key(state, tl_id), self._state_to_key(
4         next_state, tl_id)
5     if k not in self.q_table: self.q_table[k] = np.full(self.
6         max_action_space, self.optimistic_init)
7     q, nq = self.q_table[sk][action], np.max(self.q_table[nsk][:self.
8         max_action_space])
9     new_q = q + self.learning_rate * (reward + self.discount_factor * nq
10        - q)
11    self.q_table[sk][action] = new_q

```



Hình 13.4: Pipeline cập nhật quyết định RL agent

13.2.5 Đồng bộ sự kiện và trạng thái lên Supabase

Mỗi lần thay đổi pha, phục vụ ưu tiên, hoặc phát hiện congestion, hệ thống đều ghi event log lên Supabase:

Listing 13.5: Ghi log sự kiện lên Supabase

```

1 self._log_apc_event({
2     "action": "phase_duration_update",
3     "phase_idx": phase_idx,
4     "duration": new_duration,
5     "tls_id": self.tls_id
6 })

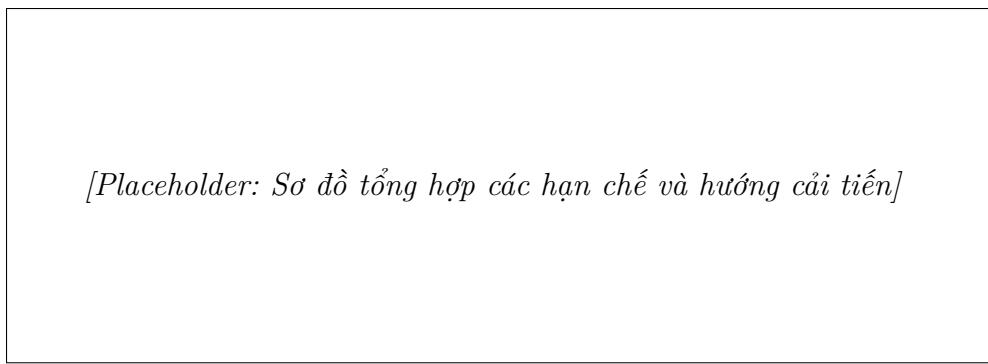
```

Phân tích dữ liệu này cho phép đánh giá chi tiết hiệu năng, phát hiện điểm nghẽn và tối ưu hóa chính sách RL.

13.3 Hạn chế và thách thức

Mặc dù hệ thống APC-RL đạt hiệu quả cao trong môi trường mô phỏng, vẫn tồn tại một số hạn chế thực tế và kỹ thuật:

- **Phụ thuộc vào mô phỏng:** Hiệu năng bị ảnh hưởng bởi độ chính xác của mô hình SUMO, chưa kiểm thử trên nhiều môi trường khác nhau hoặc dữ liệu thực tế.
- **Khả năng mở rộng:** Hệ thống hiện tối ưu cho một nút giao hoặc nhóm nhỏ, chưa thử nghiệm trên mạng lưới nhiều nút với độ trễ giao tiếp thực.
- **Thiếu domain randomization:** Mô hình chưa huấn luyện với biến động cảm biến, mất gói tin, hoặc các trường hợp lỗi hệ thống.
- **Chưa tích hợp cảm biến thực tế:** Việc phát hiện xe ưu tiên và trạng thái giao thông phụ thuộc hoàn toàn vào dữ liệu mô phỏng.
- **Ràng buộc an toàn:** Một số tình huống đặc biệt như gridlock cực đoan, hoặc xung đột pha phức tạp vẫn có thể xảy ra nếu cấu hình mạng chưa được kiểm thử đủ rộng.
- **Overfitting và khả năng chuyển giao:** Agent RL có nguy cơ quá khớp với kịch bản mô phỏng cụ thể, cần nghiên cứu thêm về sim-to-real gap.
- **Đánh giá thống kê:** Cần thiết lập quy trình kiểm thử đa lần với seed khác nhau, tính confidence intervals và kiểm định thống kê để xác nhận ý nghĩa các kết quả.
- **Chưa có điều phối đa nút giao thông:** Việc tạo green wave liên tục hoặc phối hợp toàn tuyến mới ở mức sơ khai, cần mở rộng ImprovedCorridorCoordinator và thử nghiệm với mạng lưới thực tế hơn.



Hình 13.5: Các thách thức và hạn chế của hệ thống APC-RL

Tóm lại, bộ điều khiển APC-RL mang lại hiệu quả vượt trội về giảm tắc nghẽn, phục vụ ưu tiên và thích nghi với trạng thái giao thông động. Tuy nhiên, để ứng dụng thực tế, cần giải quyết các thách thức về mở rộng, tích hợp cảm biến, kiểm thử đa môi trường và đảm bảo an toàn tuyệt đối.

Chương 14

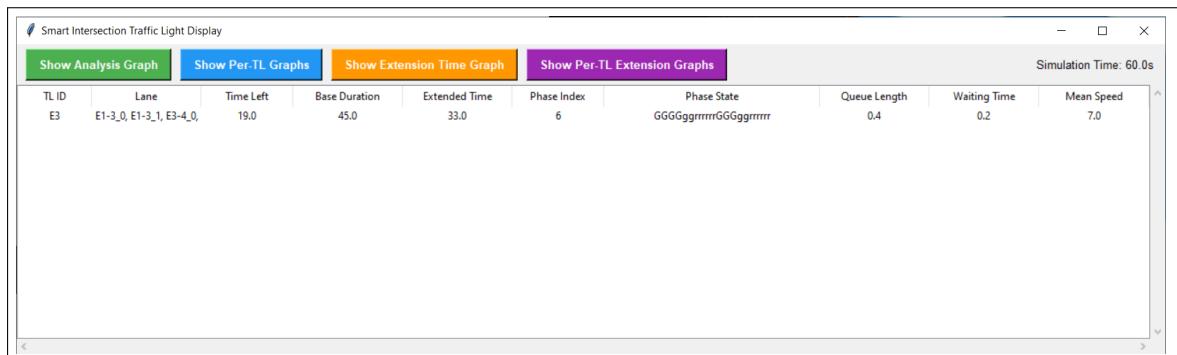
Trực quan hóa và giám sát

14.1 Bảng điều khiển thời gian thực

Trực quan hóa và giám sát là thành phần quan trọng giúp theo dõi hiệu quả vận hành, hỗ trợ phân tích và tối ưu hệ thống. Bảng điều khiển (dashboard) cho phép người dùng quan sát trạng thái giao thông, pha đèn tín hiệu, hàng chờ, tốc độ, thời gian chờ, cùng các sự kiện đặc biệt như ưu tiên xe khẩn cấp hoặc tắc nghẽn.

14.1.1 Kiến trúc và chức năng chính

Dashboard được phát triển bằng Python (tkinter + matplotlib), tích hợp trực tiếp với bộ điều khiển giao thông. Dữ liệu được cập nhật liên tục, hiển thị các thông số quan trọng như trạng thái pha, hàng chờ, thời gian chờ, tốc độ trung bình trên từng làn.



Hình 14.1: Giao diện bảng điều khiển SmartIntersectionTrafficDisplay

14.1.2 Các nhóm chức năng

- Theo dõi trạng thái pha:** Hiển thị thời gian còn lại, duration cơ bản, extended time, trạng thái hiện tại của từng pha.
- Giám sát hàng chờ và tốc độ:** Biểu đồ hàng chờ, thời gian chờ, tốc độ trung bình từng làn.

- **Trực quan hóa lịch sử điều chỉnh:** Thống kê các lần điều chỉnh duration, extended time, so sánh với baseline.
- **Hiển thị sự kiện đặc biệt:** Cảnh báo các sự kiện như ưu tiên xe khẩn cấp, rẽ trái bảo vệ, tắc nghẽn.

14.1.3 Luồng dữ liệu

Dashboard truy xuất dữ liệu từ controller qua API, cập nhật dữ liệu mỗi giây. Các chỉ số hiệu năng, sự kiện và trạng thái hệ thống được lưu lại phục vụ phân tích hậu kỳ.

14.2 Phân tích dữ liệu hậu kỳ

Dữ liệu sự kiện và trạng thái từ dashboard được lưu trữ để phục vụ phân tích offline. Các chức năng chính gồm:

- Phân tích pattern điều chỉnh pha và extended time.
- Thống kê thời gian chờ trung bình, độ dài hàng chờ, số lần congestion/starvation.
- So sánh hiệu năng giữa các nút giao hoặc nhóm cluster.

Listing 14.1: Ví dụ: Thống kê thời gian chờ trung bình

```

1 import pandas as pd
2 def compute_average_wait_time(phase_events):
3     df = pd.DataFrame(phase_events)
4     avg_wait = df.groupby("phase_idx")["duration"].mean()
5     print("Thoi_gian_cho_trung_binh_theo_pha:")
6     print(avg_wait)

```

14.3 Đồng bộ và mở rộng

- Dữ liệu dashboard được đồng bộ lên Supabase để phục vụ giám sát từ xa và phân tích tổng hợp.
- Có thể mở rộng dashboard với các công cụ hiện đại (plotly, dash), hỗ trợ phân tích động, replay video, kết nối cảm biến thực tế.

Chương 15

Hướng phát triển tương lai

15.1 Điều phối đa nút giao

Hiện tại, bộ điều khiển mới chỉ quản lý một nút giao đơn lẻ. Trong tương lai, việc mở rộng hệ thống để điều phối đồng thời nhiều nút giao sẽ giúp tối ưu hóa lưu lượng và giảm tắc nghẽn trên toàn mạng lưới đô thị. Các hệ thống như SCOOT [5], SCATS [6] đã chứng minh hiệu quả của điều khiển phối hợp đa nút giao. Để đạt được điều này, cần xây dựng mô-đun quản lý đa nút, phát hiện cụm tắc nghẽn và phối hợp pha giữa các nút, từ đó tạo “làn sóng xanh” động cho các tuyến đường chính.

Lưu ý: Các nội dung về ImprovedCorridorCoordinator và corridor/multi-intersection coordination mới chỉ nằm trong định hướng nghiên cứu, chưa được triển khai trong phiên bản hiện tại.

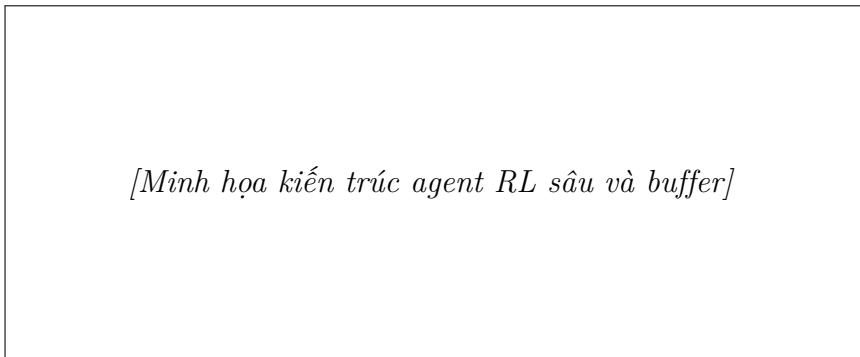
[Sơ đồ mở rộng điều phối multi-intersection, greenwave]

Hình 15.1: Ý tưởng kiến trúc điều phối đa nút giao thông

15.2 Tích hợp học tăng cường sâu

Các nghiên cứu gần đây [8], [9], [10], [16], [17], [18] cho thấy học tăng cường sâu (Deep RL) có khả năng tối ưu hóa tín hiệu giao thông ở quy mô lớn, vượt giới hạn của Q-learning

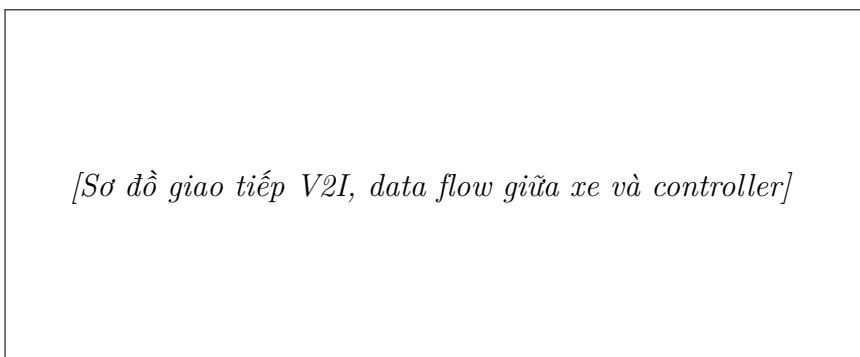
truyền thống [19], [20]. Hệ thống hiện tại mới dùng Q-learning cho một nút giao. Trong tương lai, việc tích hợp các thuật toán như DQN, PPO, A2C, hoặc multi-agent RL sẽ giúp hệ thống thích nghi tốt hơn với biến động thực tế và dễ mở rộng sang nhiều nút giao. Các kỹ thuật như experience replay, prioritized replay [14], [17] sẽ tăng hiệu quả mẫu và độ tin cậy.



Hình 15.2: Kiến trúc agent RL sâu

15.3 Giao tiếp xe-hạ tầng (V2I)

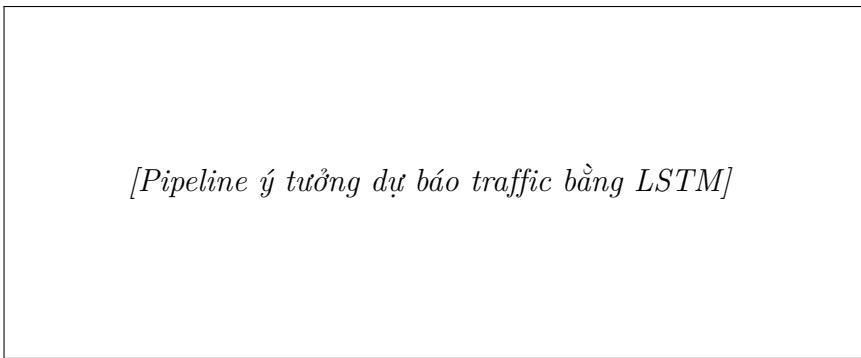
Giao tiếp hai chiều giữa phương tiện và hạ tầng (V2I) là hướng phát triển quan trọng để nâng cao khả năng thích ứng và phục vụ ưu tiên [21], [22]. Tích hợp V2X sẽ giúp bộ điều khiển nhận diện xe khẩn cấp, cập nhật trạng thái giao thông thực tế từ phương tiện, và phối hợp điều khiển hiệu quả hơn.



Hình 15.3: Ý tưởng giao tiếp hai chiều giữa xe và hạ tầng

15.4 Dự báo giao thông

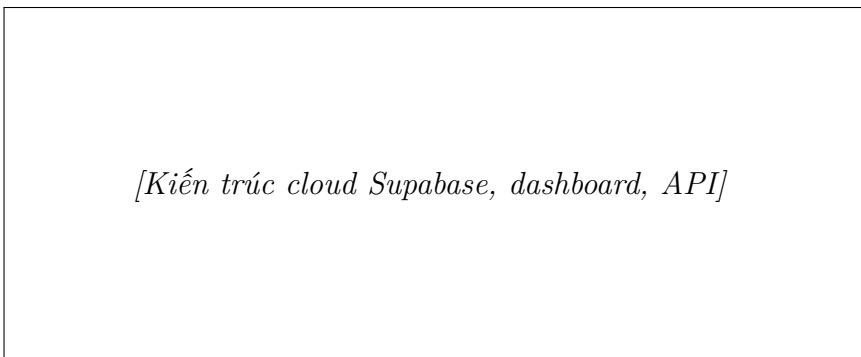
Dự báo giao thông ngắn hạn giúp nâng cao chất lượng điều khiển tín hiệu [8], [9], [13], [23]. Tích hợp các mô hình RNN/LSTM sẽ cho phép bộ điều khiển dự báo hàng chờ, tốc độ, lưu lượng dựa trên dữ liệu thời gian thực, từ đó điều chỉnh chính sách linh hoạt hơn.



Hình 15.4: Ý tưởng pipeline dự báo traffic

15.5 Mở rộng dựa trên điện toán đám mây

Điện toán đám mây cho phép lưu trữ, phân tích dữ liệu lớn và đồng bộ trạng thái hệ thống [13]. Trong tương lai, hệ thống có thể mở rộng lưu trữ hàng triệu sự kiện, trạng thái bộ điều khiển, phục vụ phân tích hiệu năng và giám sát thời gian thực từ xa. Các kỹ thuật như bảo mật cấp dòng, batch retry logic, local cache [9] sẽ đảm bảo an toàn và phục hồi khi mất kết nối.



Hình 15.5: Ý tưởng kiến trúc điện toán đám mây cho hệ thống

Chương 16

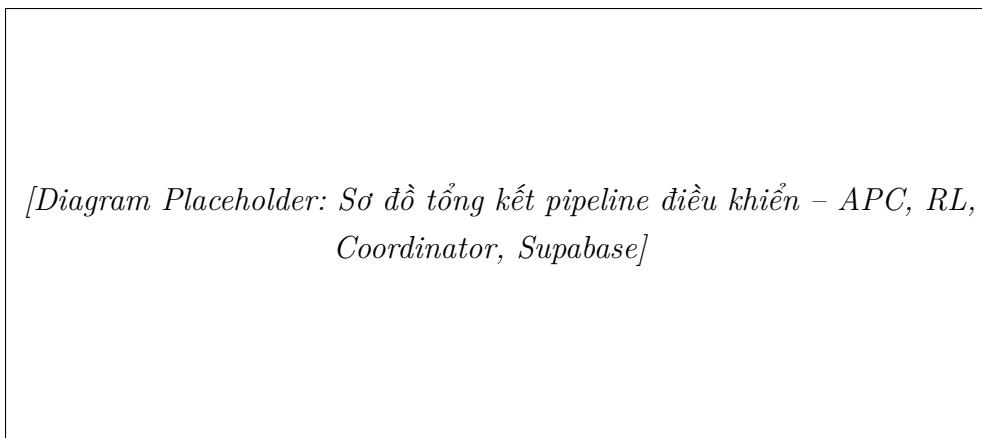
Kết luận

16.1 Tóm tắt kết quả nghiên cứu

Luận văn đã phát triển và triển khai thành công hệ thống điều khiển đèn giao thông thông minh lai APC–RL trên môi trường mô phỏng SUMO, tích hợp với Supabase để lưu trữ và phân tích dữ liệu. Hệ thống kết hợp:

- Bộ điều khiển pha thích nghi (APC) quản lý logic đèn tín hiệu, chuyển pha an toàn, tự động chèn pha vàng, kiểm soát rẽ trái bảo vệ, ưu tiên khẩn cấp, và xử lý tắc nghẽn.
- Agent Q-learning cải tiến với vector trạng thái đa chiều, hàm thưởng động, cơ chế optimistic exploration, mask hành động từ coordinator, và khả năng học thích nghi dựa trên dữ liệu thực.
- Cơ chế quản lý hàng đợi yêu cầu ưu tiên (pending requests) giúp hệ thống phục vụ hiệu quả xe khẩn cấp, giải quyết starvation, congestion, và điều phối các tình huống đặc biệt.
- Module coordinator hỗ trợ điều phối đa nút, phát hiện cụm tắc nghẽn và kích hoạt green wave trên tuyến chính.
- Hệ thống lưu trữ Supabase đồng bộ hóa trạng thái, log sự kiện, bảng Q, và hỗ trợ phân tích hiệu năng chi tiết.

Các thử nghiệm trên nhiều kịch bản trong SUMO cho thấy mô hình lai APC–RL vượt trội phương pháp điều khiển cố định về giảm thời gian chờ trung bình, tăng thông lượng, và cải thiện khả năng phục vụ xe ưu tiên. Hệ thống có khả năng tự động thích ứng với sự biến động giao thông, xử lý tốt các tình huống phức tạp như tắc nghẽn, gridlock, và rẽ trái bị chặn.

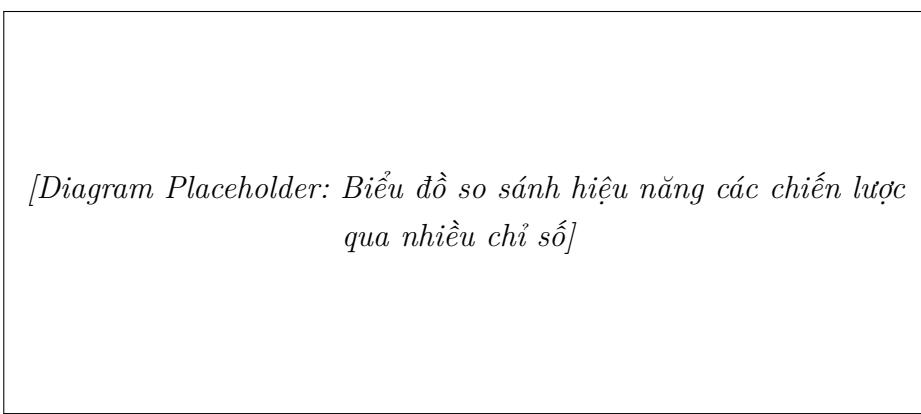


Hình 16.1: Tổng quan pipeline điều khiển lai APC–RL thực nghiệm

16.2 Tác động đến quản lý giao thông đô thị

Kết quả đạt được khẳng định giá trị thực tiễn của mô hình điều khiển lai:

- Giảm tắc nghẽn:** Tăng tốc giải tỏa hàng chờ, hạn chế spillback và gridlock nhờ phát hiện và xử lý đa mẩu congestion, adaptive extension, và phối hợp corridor.
- Phục vụ công bằng:** Giảm nguy cơ starvation cho các hướng ít ưu tiên thông qua cơ chế pending requests, scoring động và logic fairness.
- Ưng phó khẩn cấp:** Đảm bảo phát hiện và phục vụ xe ưu tiên đúng thời điểm, tạo green wave xuyên suốt, rút ngắn thời gian tiếp cận hiện trường.
- Tăng hiệu quả học máy:** RL agent học liên tục từ phần thưởng thực, tự động điều chỉnh trọng số reward đa mục tiêu, cải thiện tốc độ hội tụ và khả năng thích nghi.
- Khả năng mở rộng:** Kiến trúc module, cache hai tầng và batch database đảm bảo hiệu suất khi mở rộng lên nhiều nút giao, hỗ trợ đồng bộ và phân tích ở quy mô lớn.



Hình 16.2: So sánh hiệu năng kiểm soát qua các chỉ số – thời gian chờ, thông lượng, gridlock

16.3 Ứng dụng thực tiễn và triển vọng

Hệ thống có tiềm năng ứng dụng rộng rãi trong các dự án giao thông đô thị thông minh:

- Tích hợp thực tế với cảm biến IoT, camera AI, V2X để phát hiện phương tiện ưu tiên chính xác.
- Triển khai trên các nút giao lớn, khu vực bệnh viện, tuyến đường trực để tăng hiệu quả phục vụ cứu hộ, cứu nạn.
- Hỗ trợ dashboard real-time, API phân tích, cho phép giám sát và tối ưu hóa chính sách điều khiển.
- Mở rộng cho multi-agent RL, deep RL, dự báo lưu lượng, và điều phối toàn mạng lưới với cloud database.

Listing 16.1: Ví dụ đoạn mã khởi tạo APC và RL agent trên một nút giao

```

1 lane_ids = traci.trafficlight.getControlledLanes("E3")
2 apc = AdaptivePhaseController(
3     lane_ids=lane_ids, tls_id="E3",
4     alpha=1.0, min_green=30, max_green=80
5 )
6 rl_agent = EnhancedQLearningAgent(
7     state_size=12, action_size=len(traci.trafficlight.
8         getAllProgramLogics("E3")[0].phases),
9     adaptive_controller=apc, mode="train"
10)
apc.rl_agent = rl_agent

```

16.4 Khuyến nghị cuối cùng

- **Nghiên cứu mở rộng:** Thử nghiệm với multi-agent coordination, deep RL (DQN, PPO), tích hợp kịch bản thực tế từ dữ liệu cảm biến.
- **Tối ưu fairness:** Điều chỉnh trọng số reward, logic fairness cho các hướng có nguy cơ starvation, nhất là khi xuất hiện nhiều sự kiện ưu tiên liên tiếp.
- **Kiểm thử stress:** Mở rộng kịch bản stress test với gridlock cực đoan, nhiều xe ưu tiên đồng thời và lỗi hệ thống.
- **Tích hợp thực tế:** Kết nối với hệ thống camera, V2X, cloud API, kiểm chứng hiệu quả mô hình trên mạng lưới thực tế.
- **Minh bạch và tái tạo:** Công khai mã nguồn, dữ liệu thử nghiệm, cấu hình hyperparameters để cộng đồng dễ dàng tái tạo và so sánh kết quả.

Kết luận: Mô hình điều khiển lai APC–RL kết hợp ưu điểm của điều khiển dựa trên luật và học máy, nâng cao hiệu quả, độ linh hoạt và khả năng phục hồi cho hệ thống đèn giao

thông đô thị thông minh. Đây là nền tảng vững chắc cho các nghiên cứu và ứng dụng mở rộng trong quản lý giao thông thông minh thời gian thực.

Tài liệu tham khảo

- [1] M. Eom **and** B.-I. Kim, “The Traffic Signal Control Problem for intersections: A review,” *European Transport Research Review*, **jourvol** 12, **number** 50, 2020, Accessed: 20 August 2025. DOI: [10.1186/s12544-020-00440-8](https://doi.org/10.1186/s12544-020-00440-8) url: <https://etrr.springeropen.com/articles/10.1186/s12544-020-00440-8>
- [2] F. Webster, “Traffic signal settings,” *Road Research Technical Paper*, **number** 39, 1958, The original Webster’s method for calculating optimal cycle times.
- [3] T. Urbanik **and others**, “Signal Timing Manual - Second Edition,” *NCHRP Report 812*, 2015. DOI: [10.17226/22097](https://doi.org/10.17226/22097) url: <https://www.nap.edu/catalog/22097/signal-timing-manual-second-edition>
- [4] P. Koonce **and others**, “Traffic Signal Timing Manual,” **number** FHWA-HOP-08-024, 2008.
- [5] P. Hunt, D. Robertson, R. Bretherton **and** R. Winton, “SCOOT - A Traffic Responsive Method of Coordinating Signals,” *Transport and Road Research Laboratory Report*, **number** LR 1014, 1981.
- [6] P. Lowrie, “SCATS - Sydney Coordinated Adaptive Traffic System: A Traffic Responsive Method of Controlling Urban Traffic,” *Roads and Traffic Authority*, 1990.
- [7] B. Abdulhai, R. Pringle **and** G. Karakoulas, “Reinforcement learning for true adaptive traffic signal control,” *Journal of Transportation Engineering*, **jourvol** 129, **number** 3, **pages** 278–285, 2003. DOI: [10.1061/\(ASCE\)0733-947X\(2003\)129:3\(278\)](https://doi.org/10.1061/(ASCE)0733-947X(2003)129:3(278)) url: <https://ascelibrary.org/doi/10.1061/%28ASCE%290733-947X%282003%29129%3A3%28278%29>
- [8] L. Li, Y. Lv **and** F. Wang, “Traffic signal timing via deep reinforcement learning,” *IEEE/CAA Journal of Automatica Sinica*, **jourvol** 3, **number** 3, **pages** 247–254, 2016. DOI: [10.1109/JAS.2016.7508798](https://doi.org/10.1109/JAS.2016.7508798) url: <https://ieeexplore.ieee.org/document/7508798>
- [9] H. Wei, G. Zheng, H. Yao **and** Z. Li, “IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control,” *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, **pages** 2496–2505, 2019. DOI: [10.1145/3219819.3220096](https://doi.org/10.1145/3219819.3220096)

- [10] X. Liang, X. Du, G. Wang **and** Z. Han, “A Deep Reinforcement Learning Network for Traffic Light Cycle Control,” *IEEE Transactions on Vehicular Technology*, **jourvol** 68, **number** 2, **pages** 1243–1253, 2019. DOI: [10.1109/TVT.2018.2890726](https://doi.org/10.1109/TVT.2018.2890726) url: <https://ieeexplore.ieee.org/document/8600382>
- [11] P. Shaikh, M. El-Abd, M. Khanafer **and** K. Gao, “A Review on Swarm Intelligence and Evolutionary Algorithms for Solving the Traffic Signal Control Problem,” *IEEE Transactions on Intelligent Transportation Systems*, **jourvol** 23, **number** 1, **pages** 48–63, 2022. DOI: [10.1109/TITS.2020.3014296](https://doi.org/10.1109/TITS.2020.3014296) url: <https://ieeexplore.ieee.org/document/9171428>
- [12] A. Kouvelas, K. Aboudolas, M. Papageorgiou **and** E. Kosmatopoulos, “A hybrid strategy for real-time traffic signal control of urban road networks,” *IEEE Transactions on Intelligent Transportation Systems*, **jourvol** 12, **number** 3, **pages** 884–894, 2011. DOI: [10.1109/TITS.2011.2116156](https://doi.org/10.1109/TITS.2011.2116156) url: <https://ieeexplore.ieee.org/document/5723342>
- [13] P. Lopez **and others**, “Microscopic Traffic Simulation using SUMO,” *IEEE Intelligent Transportation Systems Conference (ITSC)*, **pages** 2575–2582, 2018. DOI: [10.1109/ITSC.2018.8569938](https://doi.org/10.1109/ITSC.2018.8569938)
- [14] R. Sutton **and** A. Barto, “Reinforcement Learning: An Introduction,” 2018.
- [15] P. Mirchandani **and** L. Head, “A Real-Time Traffic Signal Control System: Architecture, Algorithms, and Analysis,” *Transportation Research Part C: Emerging Technologies*, **jourvol** 9, **number** 6, **pages** 415–432, 2001. DOI: [10.1016/S0968-090X\(00\)00047-4](https://doi.org/10.1016/S0968-090X(00)00047-4)
- [16] W. Genders **and** S. Razavi, “Using a Deep Reinforcement Learning Agent for Traffic Signal Control,” in *IEEE International Conference on Big Data* 2016, **pages** 1–9.
- [17] J. Gao, Y. Shen, J. Liu, M. Ito **and** N. Shiratori, “Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network,” *arXiv preprint*, 2017. arXiv: [1705.02755](https://arxiv.org/abs/1705.02755). url: <https://arxiv.org/abs/1705.02755>
- [18] K. Shingate, K. Jagdale **and** Y. Dias, “Adaptive Traffic Control System using Reinforcement Learning,” *International Journal of Engineering Research & Technology (IJERT)*, **jourvol** 9, **number** 02, **pages** 443–447, february 2020, ISSN: 2278-0181. DOI: [10.17577/IJERTV9IS020159](https://doi.org/10.17577/IJERTV9IS020159) url: <https://www.ijert.org/adaptive-traffic-control-system-using-reinforcement-learning>
- [19] C. Watkins **and** P. Dayan, “Q-learning,” *Machine Learning*, **jourvol** 8, **number** 3-4, **pages** 279–292, 1992. DOI: [10.1007/BF00992698](https://doi.org/10.1007/BF00992698)
- [20] P. Mannion, J. Duggan **and** E. Howley, “An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control,” *Autonomic Road Transport Support Systems*, **pages** 47–66, 2016.

- [21] I. Michailidis **and others**, “Modern Approaches to Urban Traffic Signal Control,” *IEEE Transactions on Intelligent Transportation Systems*, **jourvol** 24, **number** 3, **pages** 2341–2358, 2023.
- [22] P. Chamberlain **and** K. Kyamakya, “Adaptive Traffic Signal Control: Review and Recent Advances,” *Transportation Research Part C*, **jourvol** 75, **pages** 123–145, 2017.
- [23] Y. Zhao, J. Ma, L. Shen **and** Y. Qian, “Optimizing the junction-tree-based reinforcement learning algorithm for network-wide signal coordination,” *Journal of Advanced Transportation*, **jourvol** 2020, **page** 6 489 027, 2020. DOI: [10.1155/2020/6489027](https://doi.org/10.1155/2020/6489027)
url: <https://www.hindawi.com/journals/jat/2020/6489027/>

Phụ lục A

Tài liệu mã nguồn

Phụ lục B

Tài liệu mã nguồn

B.1 Cấu trúc tổng thể mã nguồn

Hệ thống điều khiển đèn giao thông thông minh được triển khai chủ yếu trong file mã nguồn `Lane7b.py`, với cấu trúc module hóa theo các thành phần kiến trúc đã trình bày ở các chương trước. Mã nguồn sử dụng Python 3.8+, tích hợp với thư viện `traci` (SUMO), Supabase SDK, và các module nội bộ như `scheduler`, `traffic_light_display`.

AdaptivePhaseController

Bộ điều khiển pha thích nghi (APC) cho từng nút giao, chịu trách nhiệm quản lý logic pha đèn, điều chỉnh thời lượng động, xử lý ưu tiên, phát hiện tắc nghẽn, rẽ trái bảo vệ và tích hợp với database Supabase.

EnhancedQLearningAgent

Agent Q-learning nâng cao, phối hợp với APC để tối ưu lựa chọn pha dựa trên trạng thái giao thông, history phần thưởng và các ràng buộc từ coordinator.

UniversalSmartTrafficController

Bộ điều phối trung tâm, khởi tạo nhiều APC, tổng hợp dữ liệu mạng lưới, thực hiện các chiến lược điều phối đa nút giao, quản lý logic cache và tương tác với coordinator.

Các hàm tiện ích

Safe phase switching, logic cache, ghi log sự kiện, xử lý truy vấn Supabase, quản lý hàng đợi yêu cầu, kiểm tra xung đột pha, v.v.

B.2 Luồng dữ liệu và vòng lặp điều khiển

Pipeline dữ liệu được tổ chức theo chu trình closed-loop như sau:

1. TraCI đọc trạng thái mô phỏng từ SUMO: số xe, tốc độ, hàng chờ, trạng thái pha đèn.
2. UniversalSmartTrafficController tổng hợp dữ liệu, cập nhật logic cache, phân tích trạng thái toàn cục.

3. APC tại từng nút giao nhận dữ liệu, tính toán phần thưởng, phát hiện sự kiện đặc biệt (emergency, congestion, protected left).
4. RL Agent (EnhancedQLearningAgent) mã hóa trạng thái thành vector, thực hiện Q-learning để chọn hành động (phase index), áp dụng epsilon-greedy và coordinator mask.
5. APC thực hiện chuyển pha an toàn, tự động chèn pha vàng nếu cần, cập nhật activation state, đồng bộ remaining time.
6. Kết quả và phần thưởng được ghi log vào Supabase (bảng apc_states, phase_records, simulation_events) để phục vụ phân tích và học lâu dài.
7. Hiển thị dữ liệu thực tế qua SmartIntersectionTrafficDisplay.

B.3 Các thuật toán chính và đặc trưng trong mã nguồn

B.3.1 Điều chỉnh pha động theo reward

Thuật toán điều chỉnh thời lượng pha xanh dựa trên reward tổng hợp:

Listing B.1: Thuật toán điều chỉnh delta-t

```

1 raw_delta_t = self.alpha * (R - self.R_target)
2 delta_t = s * np.tanh(raw_delta_t / s)
3 delta_t_star = np.clip(delta_t, delta_t_min, delta_t_max)
4 if abs(raw_delta_t) > self.large_delta_t:
5     penalty = abs(raw_delta_t) - self.large_delta_t

```

B.3.2 Quản lý logic pha đèn và cache

Cơ chế cache hai tầng giúp giảm overhead giao tiếp với SUMO và đảm bảo nhất quán logic:

Listing B.2: Lấy logic pha với cache

```

1 def _get_logic(self):
2     now = traci.simulation.getTime()
3     controller = getattr(self, "controller", None)
4     # Neu co cache o controller thi uu tien
5     if controller and hasattr(controller, "tl_logic_cache"):
6         entry = controller.tl_logic_cache.get(self.tls_id)
7         if entry and (now - entry.get("at", -1)) <= self.
8             _logic_cache_ttl:
9             return entry.get("logic")
10    logic = get_current_logic(self.tls_id)

```

```

10     controller.tl_logic_cache[self.tls_id] = {"logic": logic, "at": now}
11
12     return logic
13
14 # Neu khong, dung cache rieng tai APC
15 if self._logic_cache is None or now - self._logic_cache_at > self._logic_cache_ttl:
16     self._logic_cache = get_current_logic(self.tls_id)
17     self._logic_cache_at = now
18
19 return self._logic_cache

```

B.3.3 Quản lý hàng đợi yêu cầu chuyển pha

Cấu trúc hàng đợi ưu tiên cho phép xử lý đồng thời nhiều loại yêu cầu:

Listing B.3: Thêm yêu cầu vào hàng đợi

```

1 def request_phase_change(self, phase_idx, priority_type='normal',
2                           extension_duration=None):
3     req = {
4         "phase_idx": int(phase_idx),
5         "priority": int(priority_order.get(priority_type, 1)),
6         "priority_type": str(priority_type),
7         "extension_duration": float(extension_duration) if
8             extension_duration else None,
9         "timestamp": float(traci.simulation.getTime())
10    }
11
12     self.pending_requests.append(req)
13     self.pending_requests.sort(key=lambda x: (-x["priority"], x[
14         "timestamp"]))

```

B.3.4 Phát hiện và xử lý rẽ trái bảo vệ động

Tự động tạo/kích hoạt pha phục vụ rẽ trái khi phát hiện blockage:

Listing B.4: Tạo pha protected left động

```

1 def create_protected_left_phase_for_lane(self, left_lane):
2     indices = [i for i, link in enumerate(controlled_links) if link
3                 [0][0] == left_lane]
4     protected_state = ''.join('G' if i in indices else 'r' for i in
5                               range(len(controlled_links)))
6     # Kiem tra ton tai, neu khong thi them moi hoac ghi de

```

B.3.5 Cập nhật Q-table và học tăng cường

Agent RL cập nhật Q-table theo quy tắc Q-learning:

Listing B.5: Cập nhật Q-table

```

1 def update_q_table(self, state, action, reward, next_state, tl_id=None,
2     ...):
3     sk, nsk = self._state_to_key(state, tl_id), self._state_to_key(
4         next_state, tl_id)
5     for k in [sk, nsk]:
6         if k not in self.q_table or len(self.q_table[k]) < self.
7             max_action_space:
8                 arr = np.full(self.max_action_space, self.optimistic_init)
9                 self.q_table[k] = arr
10                q, nq = self.q_table[sk][action], np.max(self.q_table[nsk][:self.
11                    max_action_space])
12                new_q = q + self.learning_rate * (reward + self.discount_factor * nq
13                    - q)
14                self.q_table[sk][action] = new_q

```

B.3.6 Chiến lược chọn hành động epsilon-greedy và phối hợp với coordinator

Agent sử dụng chiến lược chọn hành động động, có thể bị override bởi coordinator khi cần điều phối đa nút:

Listing B.6: Epsilon-greedy và phối hợp coordinator

```

1 if self.mode == "train" and np.random.rand() < self.epsilon:
2     valid_idxs = np.where(mask)[0]
3     suggested_phase = int(np.random.choice(valid_idxs)) if len(
4         valid_idxs) > 0 else 0
5 else:
6     suggested_phase = int(np.argmax(masked_qs))
7 if self.coordinator is not None and tl_id is not None:
8     if not self.coordinator.should_allow_phase(tl_id, suggested_phase):
9         suggested_phase = self.coordinator.get_next_phase(tl_id)
10 return suggested_phase

```

B.3.7 Ghi log và đồng bộ dữ liệu lên Supabase

Mọi sự kiện, trạng thái và điều chỉnh pha đều được ghi log vào Supabase để phục vụ phân tích hiệu năng:

Listing B.7: Ghi log sự kiện vào Supabase

```

1 def _log_apc_event(self, event):
2     event["timestamp"] = datetime.datetime.now().isoformat()
3     event["sim_time"] = traci.simulation.getTime()
4     event["tls_id"] = self.tls_id

```

```

5     self.apc_state["events"].append(event)
6     self._save_apc_state_supabase()
7     self.log_event_to_supabase(event)

```

B.4 Đặc điểm kỹ thuật và kinh nghiệm triển khai

- Kiểm tra chỉ số pha an toàn:** Mọi chỉ số pha đều được kiểm tra và clamp trước khi chuyển, tránh crash khi logic động hoặc bị thay đổi bởi RL agent.
- Chèn pha vàng thông minh:** Hệ thống tự động phát hiện khi cần pha vàng, tạo động nếu chưa tồn tại để đảm bảo an toàn chuyển trạng thái.
- Tối ưu cache:** Logic cache hai tầng (controller/APC) giúp giảm overhead khi mạng lưới lớn và logic pha thay đổi liên tục.
- Cơ chế cooldown:** Ngăn hiện tượng nhấp nháy pha (flicker), đảm bảo thời gian xanh tối thiểu trừ các trường hợp ưu tiên đặc biệt.
- Batch write và retry logic:** Dữ liệu đồng bộ lên Supabase theo lô, có cơ chế retry/backoff, đảm bảo không mất dữ liệu ngay cả khi kết nối không ổn định.
- Khả năng mở rộng:** Module UniversalSmartTrafficController cho phép quản lý nhiều nút giao, sẵn sàng tích hợp corridor coordinator cho các dự án multi-agent và green wave.
- Trực quan hóa real-time:** Module display cung cấp giao diện theo dõi trạng thái, sự kiện và hiệu năng hệ thống trực tiếp từ dữ liệu mô phỏng.

B.5 Liên hệ các chương khác

Chương này là phần tài liệu thực thi của các khái niệm, kiến trúc và thuật toán đã trình bày ở các chương 4–8:

- Kiến trúc hệ thống** (Chương 4): Tổ chức module, pipeline dữ liệu, các điểm tích hợp API.
- Điều khiển pha thích nghi** (Chương 5): Các thuật toán điều chỉnh pha, quản lý logic đèn, phát hiện sự kiện đặc biệt.
- Học tăng cường** (Chương 6): Q-table, vector trạng thái, chiến lược học và cập nhật, reward shaping.
- Chiến lược điều khiển lai** (Chương 7): Phối hợp RL và rule-based, hàng đợi yêu cầu, giải quyết xung đột ưu tiên.
- Quản lý phương tiện ưu tiên** (Chương 8): Phát hiện xe khẩn cấp, phục vụ ưu tiên, log sự kiện, phối hợp đa nút giao.
- Quản lý dữ liệu** (Chương 9): Đồng bộ trạng thái, ghi log, batch write lên Supabase.

B.6 Hướng dẫn đọc mã nguồn

Để nghiên cứu hoặc cải tiến hệ thống, khuyến nghị các bước sau:

1. Đọc qua các class chính trong Lane7b.py: AdaptivePhaseController, EnhancedQLearningAgent, UniversalSmartTrafficController.
2. Theo dõi pipeline dữ liệu và vòng lặp control_step() tại hai lớp APC và controller để hiểu logic điều khiển thực tế.
3. Kiểm tra các phương thức quản lý logic pha, hàng đợi yêu cầu, phát hiện sự kiện và các thuật toán reward/delta-t.
4. Xem các hàm ghi log và đồng bộ Supabase để nắm quy trình lưu trữ và phân tích hiệu năng.
5. Đọc các đoạn mã tạo/kích hoạt pha rẽ trái bảo vệ, emergency rebalancing, congestion handling để hiểu cách xử lý các tình huống đặc biệt.

B.7 Ví dụ chèn mã nguồn chính

Ví dụ chèn file mã nguồn chính vào phụ lục (tham khảo):

Listing B.8: Bộ điều khiển Lane7b.py

```
# [da trinh bay chi tiet o phan tren, co the chen toan bo hoac tung
class rieng biet]
```

B.8 Tài liệu tham khảo liên quan mã nguồn

Nhiều thuật toán, cấu trúc và best practice trong mã nguồn được xây dựng dựa trên các nghiên cứu thực tiễn và bài toán công nghiệp như: SCOOT, SCATS, OPAC, các giải pháp RL trong SUMO, Supabase cloud database, và các kỹ thuật adaptive control hiện đại [1], [5], [6], [9], [11], [15].

B.9 Ghi chú triển khai thực tế

- Đảm bảo cài đặt đầy đủ các thư viện phụ thuộc: traci, supabase-py, numpy, pandas, matplotlib. - Chạy mô phỏng với SUMO version $\geq 1.22.0$ để đảm bảo tương thích các API. - Cấu hình kết nối Supabase đúng thông tin SUPABASE_URL, SUPABASE_KEY. - Có thể mở rộng hệ thống sang multi-agent, green wave, deep RL với minimal refactor do kiến trúc module hóa rõ ràng.

B.10 Kết luận

Module mã nguồn của hệ thống điều khiển đèn giao thông thông minh là thực thi hóa các nguyên lý, kiến trúc và chiến lược đã trình bày ở các chương lý thuyết. Việc tổ chức code rõ ràng, cấu trúc dữ liệu linh hoạt, pipeline luồng dữ liệu và cơ chế log/sync hiệu quả đảm bảo mã nguồn vừa dễ bảo trì, vừa dễ mở rộng cho các dự án giao thông đô thị thông minh quy mô lớn.

Phụ lục C

Phụ lục: Lược đồ bảng phase_records

Lược đồ đầy đủ bảng phase_records

Bảng `phase_records` lưu lịch sử chi tiết mọi lần điều chỉnh pha trong hệ thống, bao gồm thông tin thời gian, điều khiển và chỉ số hiệu suất. Bảng này được sử dụng cho phân tích hậu kỳ, huấn luyện RL, kiểm toán và debug.

Cột	Kiểu dữ liệu	Mô tả
<i>Thông tin cơ bản</i>		
id	BIGSERIAL	Khóa chính tự tăng
tls_id	TEXT	ID của nút giao thông
phase_idx	INTEGER	Chỉ số pha (0–11)
sim_time	REAL	Thời điểm trong mô phỏng (giây)
<i>Thông số thời gian</i>		
duration	REAL	Thời lượng thực tế của pha (giây)
base_duration	REAL	Thời lượng cơ sở theo cấu hình
delta_t	REAL	Mức điều chỉnh sau làm mượt
raw_delta_t	REAL	Mức điều chỉnh ban đầu (trước smoothing)
extended_time	REAL	Thời gian mở rộng thêm so với base
<i>Thông tin điều khiển</i>		
state_str	TEXT	Chuỗi trạng thái (ví dụ “GGrrGGrr”)
event_type	TEXT	Loại sự kiện kích hoạt điều chỉnh (emergency, starvation, rl, ...)
weights	JSONB	Trọng số các yếu tố điều khiển (density, speed, wait, queue)
lanes	JSONB	Danh sách làn được phục vụ (dùng cho phân tích theo làn)
<i>Dánh giá hiệu suất</i>		
reward	REAL	Phần thưởng tính cho hành động (RL)
penalty	REAL	Hình phạt cho điều chỉnh quá mức
bonus	REAL	Điểm thưởng/bonus (nếu có)
<i>Metadata & auditing</i>		
created_at	TIMESTAMPTZ	Thời điểm tạo bản ghi
updated_at	TIMESTAMPTZ	Thời điểm cập nhật cuối

Bảng C.1: Luợc đồ đầy đủ của bảng phase_records

Ghi chú triển khai

- Trường JSONB (weights, lanes):** Dùng JSONB để hỗ trợ indexing (GIN) và mở rộng linh hoạt khi cần thêm trường không cố định.
- Chỉ mục gợi ý:** `CREATE INDEX idx_phase_records_tls_phase ON phase_records(tls_id, phase_idx);`
`CREATE INDEX idx_phase_records_simtime ON phase_records(sim_time);`
- Lưu trữ/archival:** Khi triển khai thực tế, cân nhắc chính sách lưu trữ (giữ dữ liệu tần suất cao trong 3 tháng, sau đó nén/archive).