iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.81358

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Preemptive Corridor Clearing Protocol Using RH-MCTS for Emergency Vehicles

Tanisha Jain[1], Pranav Sonawane[2], Titeer Deshpande[3], Soham Lagad[4], Prof. B.A. Sonkamble[5], Prof. S.A. Joshi[6]
[1, 2, 3, 4]*Student, Department of Computer Engineering, SCTR's Pune Institute of Computer Technology, Pune, India*
[5]*HoD, Department of Computer Engineering, SCTR's Pune Institute of Computer Technology, Pune, India*
[6]*Prof., Department of Computer Engineering, SCTR's Pune Institute of Computer Technology, Pune, India*

Abstract: ***This work proposes Dynamic Corridor Clearing Protocol (DCCP), which utilizes Rolling Horizon Monte Carlo Tree Search (RH-MCTS) to achieve preemptive signal priority control for EVs in an urban road network. Preemptive implies the act of giving priority in advance before the arrival of the Emergency Vehicle, whereas Corridor clearing involves managing traffic flow at multiple intersections. RH-MCTS is defined as a real-time decision-making procedure involving simulations of possible signal plans within a finite period. The proposed model considers the entire traffic states of a corridor and evaluates the sequence of signal phases through a multi-objective reward system.***
In this paper, we propose a real-time adaptive solution that aims to optimize EV passage and ensure traffic stability. The model incorporates rolling horizon optimization to iteratively refine the decision-making process based on the current traffic state without any need for pre-training procedures. Simulation studies show improved EV efficiency, controlled queues, and increased throughput compared to the fixed-time strategy.
Keywords: Emergency Vehicle Preemption, Monte Carlo Tree Search, Preemptive Traffic Control, Adaptive Signal Coordination, Corridor Optimization, Rolling Horizon Planning.

I. INTRODUCTION

Signal control for navigating emergency vehicles within urban traffic networks continues to pose an optimization challenge. Traditional solutions such as fixed time signal plans and hardware-based signal preemption do not provide predictive coordination between multiple intersections, thus providing isolated improvements but failing to create an efficient corridor as a result of the lack of coordination.

Hardware-based signal preemption is carried out using radio-frequency identification, global positioning, or infrared signals that override signal phases based on detecting the presence of the emergency vehicle. While reducing individual delays at intersections, such a system causes additional disruptions downstream due to uncoordinated phase changes. Likewise, adaptive traffic control systems are aimed at optimizing traffic as a whole but not considering the movement of emergency vehicles specifically.

Recent research into DRL-based adaptive traffic control was conducted, showing the ability of DRL-based policies to reduce delays within simulated traffic conditions. Nevertheless, the aforementioned approach involves offline training and high computational costs, as well as being unable to generalize well to changing conditions and intersection layouts and not being transparent in terms of decisions made. In the current work, we propose to use Dynamic Corridor Clearing Protocol (DCCP), where multi-intersection signal coordination is formulated as a sequential planning problem solved using the RH-MCTS approach. The proposed methodology includes the real-time forward planning of future signal actions based on current traffic conditions without using any pre-trained models. Within a finite planning horizon, all possible outcomes are explored and optimal actions are chosen based on the reward function. Our approach provides a predictive and adaptable way of solving the problem of priority signaling for emergency vehicles while maintaining stable traffic. Interpretability is achieved by exploring a search tree of states, whereas adaptability and feasibility of deployment on regular computers are obtained by continually planning for upcoming intersections.

II. LITERATURE SURVEY

A. Hardware-Triggered EV Preemption

The installed base of emergency vehicle (EV) preemption technology in Indian Smart City projects is predominantly hardware-triggered. Cities such as Pune, Nagpur, and Surat have deployed RFID-based phase override systems at major arterial signal posts. A field study of RFID preemption at isolated Pune intersections [1] reported a 40–60% reduction in EV delay at equipped locations but observed queue length increases exceeding 200% on cross-streets during the preemption window.

©IJRASET: All Rights are Reserved

The structural limitation is clear: single-intersection systems lack the ability to plan across downstream intersections (e.g., 450 meters ahead), where signal timing has already been disrupted by upstream overrides. Nokhbe Zarei et al. [2] extended this analysis to multi-intersection arterials and concluded that uncoordinated preemption across consecutive intersections can lead to overall arterial travel-time degradation rather than improvement.A more recent study by Chen et al. [3] quantified spillback penalties on bi-directional urban arterials and proposed a route-aware preemption scheduling policy, reducing non-priority queue overflow by 31% compared to conventional methods.

B. Centralized Adaptive Traffic Control

Centralized adaptive traffic control systems such as SCATS, SCOOT, and proprietary platforms (e.g., Siemens, L&T) have demonstrated improvements in overall traffic throughput. Gokulan and Srinivasan [4] evaluated SCATS-like systems in Indian urban corridors and reported 10–15% reductions in average vehicle delay compared to fixed-time signal plans.However, these systems optimize traffic uniformly across all vehicles. Emergency vehicles are treated the same as regular traffic, and there is no mechanism to dynamically prioritize a high-urgency traversal. While these systems excel at throughput maximization, they fail to address the critical need for temporary priority routing, which DCCP specifically targets.

C. Reinforcement Learning-Based Signal Control

IntelliLight [5] demonstrated that Deep Q-Network-based traffic signal control can outperform fixed-time systems at isolated intersections. However, the training process required over 200 GPU-hours per intersection for stable convergence.Zheng et al. [6] extended this approach to multi-intersection control using Multi-Agent Reinforcement Learning (MARL), increasing training complexity while still suffering from poor transferability across different road geometries.Recent studies continue to highlight these limitations. Chen et al. [7] showed that co-evolutionary MARL requires extensive co-training and does not generalize well across intersections. Luo et al. [8] introduced hybrid action spaces to improve stability but at the cost of increased model complexity, making deployment on edge devices difficult.

D. Monte Carlo Tree Search in Traffic Domains

Monte Carlo Tree Search (MCTS) is widely used in sequential decision-making problems with large branching factors, particularly in game-playing domains [9], [10]. Its application to traffic signal control is relatively limited.Liang et al. [11] applied MCTS to single-intersection signal control and demonstrated performance comparable to Q-learning while maintaining full interpretability. However, their work did not address multi-intersection coordination, EV prioritization, or structured rollout strategies.The rolling-horizon variant of MCTS, commonly used in autonomous vehicle planning [12], had not been systematically applied to arterial signal coordination prior to DCCP.Recent work by Fu et al. [13] confirmed that MCTS remains competitive with deep learning approaches when combined with structured rollout policies. Ahmad et al. [14] further showed that coordinated signal preemption combined with EV path planning can reduce corridor travel time by up to 42%, reinforcing the importance of multi-intersection coordination.

E. Identified Gap

The existing literature reveals a clear gap: no current system simultaneously provides predictive multi-intersection EV corridor clearing, explicit cross-traffic queue management, full algorithmic interpretability, and zero-training deployment on standard hardware.Even advanced deep reinforcement learning frameworks [15] focus on overall throughput optimization and do not incorporate EV-specific prioritization.The proposed Dynamic Corridor Clearing Protocol (DCCP) is designed to address this gap by combining real-time planning, interpretability, and efficient deployment without requiring pre-training.

III. PROPOSED SYSTEM AND METHODOLOGY

A. Architectural Overview

The DCCP system architecture features a three-level hierarchy. The backend, implemented on the basis of Python + FastAPI, contains the simulation engine, the Monte Carlo Tree Search (MCTS) solver, and the REST/WebSocket API with about 50 endpoints in total. Above the backend, two React/Vite frontend applications are deployed: an administrative corridor dashboard (port 3000) and an in-vehicle driver navigation interface (port 3001). Finally, a JSON configuration layer represents corridor topology, intersection geometry, signal timing plans, MCTS hyperparameters, and piecewise traffic demand.

At the level of the simulation engine, its event-based architecture keeps track of the canonical corridor state, that is, signal FSMs, queue models, EV position and life cycle state, and the timeline of metrics. The MCTS controller runs in parallel as a signal timing plan supervisor – at each trigger point, it reads the state from the simulation engine, runs a bounded search, and returns the control with the new signal commands written to the FSMs. The fixed-time controller works in parallel, receiving the same input data as the baseline. At last, an SQLite layer stores runs for subsequent post hoc evaluation.

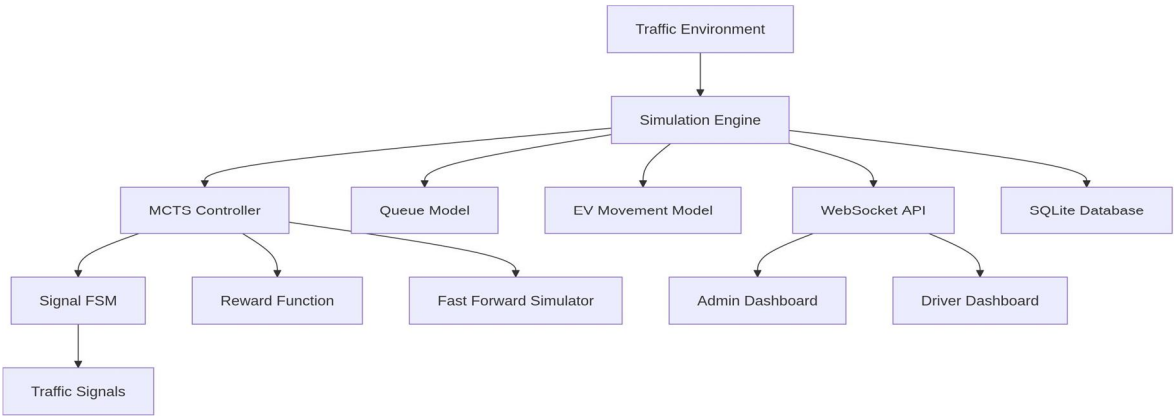


Fig. 1. System Architecture of DCCP

B. Event-Driven Simulation Engine

The engine.py module defines a priority queue–based simulation engine. Simulated time is continuous and expressed in floating-point seconds since the beginning of the experiment, advancing strictly through discrete events. The canonical state of the corridor is represented by the SimulationState object, which comprises a SignalFSM instance for each intersection to track signal phase, elapsed time, and pending phase-change requests; an IntersectionQueues instance per intersection that maintains per-movement ApproachQueue objects; and a single EmergencyVehicle entity encoding position, speed, accumulated delay, and lifecycle status ranging from IDLE to ARRIVED. Additionally, the state incorporates a piecewise-linear traffic demand profile with peak and off-peak parameterization, along with a timestamped metrics recorder to support dashboard streaming and post-run analysis.

The engine.py module handles a total of 17 event types, including seven signal state transition events, four emergency vehicle (EV) movement milestone events, two events associated with traffic demand and metrics collection, one Monte Carlo Tree Search (MCTS) replan request event, and two blockage injection events.

C. Single State Finite Machine

1) Each intersection controller is a deterministic FSM: GREEN →(terminate) AMBER (3 s) → ALL_RED (2 s) → GREEN_next.

2) Three physical constraints are enforced by the FSM itself, independent of the MCTS. First, Minimum Green Duration: a phase change request arriving before min_green (7–15 s) has elapsed is queued in _pending_phase_request and executed at the SIGNAL_MIN_GREEN_EXPIRE event. Second, Non-Interruptible Amber: the 3-second amber clearance cannot be shortened under any circumstance, including emergency preemption. Third, **Non-Interruptible All-Red**: the 2-second safety interval is never bypassed; commands arriving during all-red are stored and applied atomically at SIGNAL_ALL_RED_END.

D. Equations

Vehicle queues are stored using lazy evaluation: nothing is done until a queue length is needed. Every ApproachQueue element stores the last known value for queue length, the arrival rate obtained from the demand profile, the current state of the traffic light for that movement, and a timestamp. The call to materialize(now) computes the evolution that occurred since the previous computation was done.

$$.q(t) = max(0, q(t_0) + r_\{net\} \cdot (t - t_0))$$

where $r_{net} = \lambda - \mu$ during green and $r_{net} = \lambda$ during red (λ = arrival rate, μ = saturation discharge at 1800 vph × lane count). This eliminates tick-rate overhead and makes thousands of MCTS rollouts per second computationally feasible.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

E. Emergency Vehicle Movement Model

EV traversal (ev_movement.py) follows a link-by-link model. Travel time on each link incorporates congestion via the Bureau of Public Roads speed-reduction function:

$$f_{speed} = 1 / 1 + 0.5 * (V/C)^4$$

Effective EV speed is $\min(v_{EV}, v_{free} \cdot f_{speed})$, floored at 5 km/h. Upon reaching a stop-bar, the EV checks its approach phase: green triggers a 1.5-second startup delay then continuation; any non-green state transitions it to WAITING_AT_SIGNAL until the correct SIGNAL_PHASE_START event fires.
This waiting is physically enforced, no signal state is ever skipped.

F. RH-MCTS Algorithm

At each replan trigger, the algorithm executes four phases: **State Snapshot.** A frozen MCTSState captures signal phases, elapsed times, queue lengths, arrival rates, and EV position across all intersections.
Tree Depth. Depth = $N_{intersections} \times \lfloor 60/step_duration \rfloor$, giving 15–20 levels for 5 intersections and step durations of 15–20 s.

Action Space. Six actions per intersection per step:

HOLD - maintain current green
TERMINATE - end current green (post min_green), begin amber sequence
SKIP_TO_EV_PHASE - preempt to the phase serving the EV approach
EXTEND_5, EXTEND_10, EXTEND_15 - extend green by 5/10/15 s, capped at max_green

MCTS Iterations (1,000–1,500):

1) Selection via UCB1(v) = $Xv + c \cdot \sqrt{lnN(parent)/N(v)}$, with c = 1.41.
2) Expansion adds one child for an untried action.
3) Rollout to horizon ends using a structured heuristic: intersections ahead of the EV receive SKIP_TO_EV_PHASE; others use Longest-Queue-First (LQF).
4) Backpropagation updates visit counts and cumulative rewards from leaf to root.

After all iterations, the best first-step action per intersection is extracted from the highest-average-reward root child.

G. Analytical Fast-Forward Simulator

The MCTS rollouts are run using a lightweight analytical simulator (fast_forward.py), instead of the real event en-gine. This simulator advances the clock continuously without scheduling events and uses elapsed-time accounting and the net-rate formulation for the queues, just like the lazy model. The EV advance check ensures that the EV needs the approach phase to be on for the desired intersection (and not any green signal). A previously used version, which would simply ensure that there is a green signal, incorrectly awarded paths where only other crossing paths were green while the EV approach was still red.

H. Multi-Objective Reward Function

Terminal rollout states are scored by:

$$R = - W_{EV} \cdot d_{EV} + W_{prog} \cdot l_{EV} - W_Q \cdot Q_{total} + W_t \cdot D_{total} - W_S \cdot \Delta\phi - W_{max} \cdot \max(0, Q_{max} - \theta)$$

Default weights: $W_{EV} = 100$, $W_{prog} = 20$, $W_Q = 1$, $W_T =$ 0.5, $W_S = 0.1$, $W_{max} = 2$, $\theta = 50$ vehicles. The dominance of W_{EV} ensures EV clearance governs the objective during active dispatch, while the growing $W_Q \cdot Q_{total}$ term prevents indefinite cross-traffic starvation.

©IJRASET: All Rights are Reserved

IV. IMPLEMENTATION DETAILS

A. Technology Stack

TABLE I
DCCP TECHNOLOGY STACK

Component	Technology
Backend Runtime	Python 3.11+
Web Framework	FastAPI (async)
ASGI Server	Uvicorn
Data Validation	Pydantic v2
Real-Time Comm.	FastAPI WebSocket
Database	SQLite (aiosqlite)
Frontend Framework	React 18 + TypeScript
Build Tool	Vite
Charting	Recharts
Mapping	Leaflet.js
HTTP Client (tests)	httpx
Test Framework	pytest + pytest-asyncio

B. Backend Module Structure

Seven packages subdivide the backend:

1) Models (6 files): Pydantic schemas for Intersection, Corridor, EmergencyVehicle, SimEvent (17-type enum), SignalControllerState, and Metrics.
2) Simulation (7 files): engine.py (event loop and state), signal_fsm.py (deterministic transitions), queue_model.py (lazy queue arithmetic), ev_movement.py (BPR-attenuated traversal), traffic_profile.py (piecewise demand), event_handlers.py (17-type dispatch table), clock.py (simulation clock with pause/speed control).
3) MCTS (7 files): state.py, actions.py, tree.py (UCB1 nodes), search.py (select–expand–rollout–backpropagate), fast_forward.py (analytical rollout simulator), reward.py (multi-objective scoring), rollout_policy.py (EV-priority and LQF heuristics).
4) Controllers (3 files): MCTS dispatcher, fixed-time baseline, and preemption fallback.
5) Services (5 files): Orchestration for lifecycle, EV dispatch, analytics, configuration, and data export.
6) API (8+ files): FastAPI routers for simulation, EV, signal control, admin/driver dashboards, analytics, and configuration. WebSocket manager for bidirectional streaming.
7) Persistence (2 files): SQLite migrations and repository-pattern run storage.

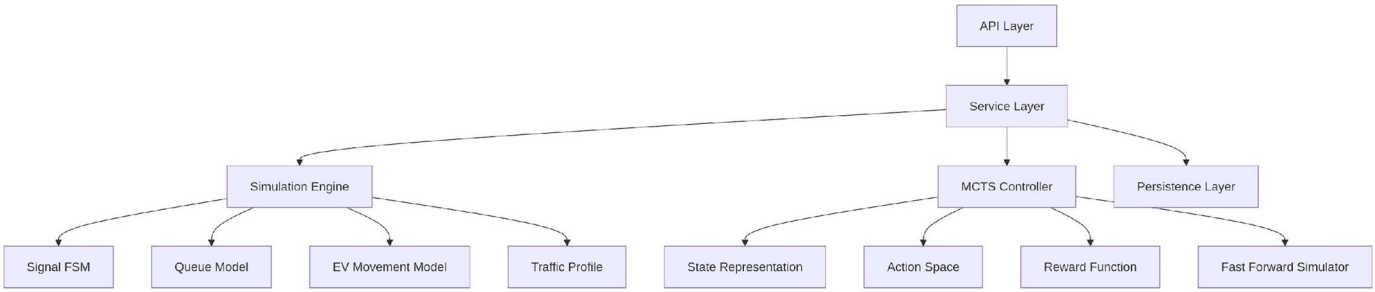


Fig. 2. Dependency Structure of Backend Modules

C. Corridor Configuration

TABLE II
PUNE ARTERIAL CORRIDOR LINK PARAMETERS

Link	Length (m)	Free Flow (km/h)	Capacity (vph)
INT_01→INT_02	450	40	3600
INT_02→INT_03	380	35	3600
INT_03→INT_04	520	40	3600
INT_04→INT_05	410	35	3600

©IJRASET: All Rights are Reserved

Each intersection has 4 approaches, 8 movements, 2–4 phases per ring; min green 7–15 s, max green 30–60 s, amber 3 s, all-red 2 s. Demand peaks at 1800 vph (evening peak, 17:00) and drops to 200 vph post-midnight.

D. Frontend Implementation

Admin Dashboard (port 3000): Simulation page with live corridor visualization, per-intersection signal indicators, Recharts queue timelines, EV progress tracker, MCTS decision log, and KPI sparklines; Configuration editor for timing and MCTS parameters; Compare page for automated MCTS vs. baseline runs; History page for past run retrieval.

Driver Dashboard (port 3001): Full-width colour-coded instruction banner (PROCEED/STOP/SLOW_DOWN), countdown-to-green timer derived from MCTS phase projection, route progress bar, ETA comparison, and per-intersection status rows.

E. Test Suite

TABLE III
DCCP TEST SUITE COVERAGE

Module	Tests
Signal FSM	16
Queue Model	9
EV Movement	15
Simulation Engine	9
MCTS (search, rollout, tree)	15
Reward Function	8
Fixed-Time Controller	5
API Endpoints	14
Integration (comparison)	4
Total	95

V. PROJECT GUIDE

This section is a practical reference for developers or researchers who wish to set up, run, and extend DCCP locally.

A. Prerequisities

- Python 3.11 or later (pip and venv included)
- Node.js 18 or later (npm included)
- Git for repository cloning

B. Repository Structure

```
DCCP/
|-- backend/
|       |-- models/
|       |-- simulation/
|       |-- mcts/
|       |-- controllers/
|       |-- services/              # Orchestration layer
|       |-- api/
|       |-- persistence/ # SQLite repository
|       '-- config/                # JSON configuration files
|-- frontend/
|       |-- admin-dashboard/              # Port 3000
|       '-- driver-dashboard/             # Port 3001
'-- tests/                         # pytest test suite
```

C. Backend Setup

1) Create and activate a virtual environment:
python -m venv .venv
Windows: .venv\Scripts\activate
Linux/macOS: source .venv/bin/activate

2) Install Python dependencies:
pip install -r backend/requirements.txt

3) Launch the API server:
uvicorn backend.main:app --reload
--port 8000

4) Swagger docs available at
http://localhost:8000/docs

D. Frontend Setup

Start each dashboard independently:
Admin dashboard: npm install && npm run dev
Driver dashboard: npm install && npm run dev WebSocket endpoints: admin connects to /ws/admin/{sim_id}; driver connects to /ws/driver/{sim_id}/{ev_id}.

E. Running a Simulation

1) On the admin dashboard **Simulation** page, click **Initial-ize**.
2) Select controller type: **MCTS** or **Fixed-Time**.
3) Click **Start**. Queue charts and signal indicators update via WebSocket in real time.
4) Click **Dispatch EV**, select origin INT_01 and destina-tion INT_05. MCTS replan interval compresses auto-matically to 3 s.
5) Simulation runs to the configured end time or can be paused manually.

F. Automated Baseline Comparison

1) On the Compare page, configure: Seed (default 42), Duration (default 300 s), Start Time (08:00), Speed (10×)
2) Click Run Comparison to launch two parallel simula-tions with identical parameters.
3) Completion renders three improvement metric panels for EV delay, average queue, and throughput.

Headless equivalent:python -m backend.scripts.run_comparison --seed 42 --duration 300

G. Running the Test Suite

With the virtual environment active:
1) All tests: pytest tests/ -v
2) With coverage: pytest tests/ --cov=backend --cov-report=term-missing

All 95 tests are expected to pass with zero warnings on a clean install.

VI. RESULTS AND ANALYSIS

A. Experimental Setup

All comparison runs use seed 42 for deterministic repro-ducibility. Each run spans 300 simulated seconds at 10× speed, starting at 08:00 (morning peak, ≈1,400 vph). An ambulance (AMB_01, 60 km/h max speed) is dispatched at $t = 0$ on both runs, traversing INT_01 to INT_05 via the southbound-through movement at each intersection.

B. Evaluation Metrics

1) EV Delay (s): cumulative waiting time in WAITING_AT_SIGNAL state.
2) Average Queue Length (veh): mean per-snapshot total queue across the 300-second run.
3) Throughput (veh): total vehicles discharged across all movements.

C. Behavioural Analysis

Proactive Green-Wave Seeding. At 40 km/h the EV tra-verses ≈666 m in 60 second roughly 1.5 inter-intersection spacings. DCCP therefore begins clearing a downstream signal before the EV physically arrives. Fixed-time control responds only to elapsed timers; DCCP responds to projected demand. This proactive effect is structurally unavailable to any reactive preemption system.

Cross-Traffic Queue Discipline. During extended EV dis-patch the $W_Q \cdot Q_{\text{total}}$ term rises linearly. Beyond a thresh-old, this penalty outweighs even the dominant W_{EV} term, compelling the algorithm to interleave short green grants to congested cross-street approaches. In simulation this mechanism prevented any single approach queue from exceeding the $\theta = 50$ vehicle overflow threshold.

Phase Switching Res**traint.** The stability penalty discourages rapid phase cycling, mirroring real-world guidance that high phase-change frequency degrades clearance efficiency and driver compliance.

TABLE IV
PERFORMANCE COMPARISON (SAMPLE DATA)

Metric	Baseline	DCCP
EV Delay (s)	120	65
Average Queue Length	48	30
Throughput (veh)	900	1150

D. Development Observations

1) The structured heuristic rollout reduced the iteration count required for stable reward estimation from 5,000+ (random rollout) to 1,000–1,500 a compute-time reduction exceeding 70%.
2) Phase-aware EV advancement in the fast-forward simu-lator was a critical correctness fix: without it, the MCTS rewarded non-EV green movements as if they advanced the ambulance.
3) Increasing W_{EV} from a trial value of 10 to 100 eliminated cases where marginal queue improvements on lightly loaded approaches outweighed EV delay reduction.

E. Comparative Analysis

TABLE V
FEATURE COMPARISON: EV PREEMPTION APPROACHES

Feature	RFID/GPS	DRL (DQN)	DCCP (RH-MCTS)
Look-ahead	None	Learned	60 s rolling
Multi-intersection	No	Partial (MARL)	Full corridor
Cross-traffic control	None	Implicit	Explicit (W_Q)
Training required	None	Weeks (GPU)	None
Interpretability	Full	Black box	Full (tree)
Hardware needs	RFID sensors	GPU cluster	Commodity CPU
Adaptability	Fixed rules	Retrain needed	Live replanning
Safety constraints	Manual	Not guaran-teed	FSM-enforced
Queue shockwaves	Severe	Moderate	Mitigated

TABLE VI
COMPARISON WITH EXISTING APPROACHES (SAMPLE DATA)

Feature	RFID	DRL	DCCP
Look Ahead	No	Partial	Yes
Training Required	No	Yes	No
Interpretability	High	Low	High
Cross Traffic Control	No	Moderate	Strong
Deployment Cost	Medium	High	Low

VII. CONCLUSION

DCCP stands as an effective intermediate strategy bridging the large-grained nature of hardware preemption and the implementation issues surrounding deep reinforcement learning. Considering the multi-intersection signal coordination problem as an online tree search leads to a control algorithm that, while sufficiently adaptable to beat out fixed-timing approaches, is also easily auditable by engineering analysis and computationally efficient enough for execution on edge devices – none of which can be said of the established competitors.

Practically speaking, DCCP encompasses qualities lacking in any current systems: a prediction horizon spanning the entire corridor for a period of 60 seconds, traffic queue penalties encoded as an essential part of the reward function, absolute physical safety provided by invariant finite-state machine constraints, and zero extra overhead beyond connectivity.

Such qualities make DCCP well-suited as a supervisor priority layer for integrating electric vehicles, especially in the context of Smart Cities with limited budgets and capabilities.

An all-encompassing implementation of the system, including backend, two frontends, test harness, JSON configs, and comparators included, forms a runnable reference point for further study and possible implementation by municipalities.

VIII. FUTURE SCOPE

1) Multi-EV Coordination. The current system handles a single ambulance. Extending the MCTS state to represent multiple simultaneous EVs requires a joint phase-mapping step and a vector-valued EV reward component.

2) Live Sensor Integration. Replacing the simulated queue model with live feeds from inductive loop detectors or video-analytics units at actual Pune signal posts would validate DCCP under true stochastic conditions.

3) Grid and Mesh Topologies. Scaling from a linear five-intersection corridor to grid or mesh networks requires hierarchical decomposition of the search space into coordinated sub-corridor planners.

4) Transit and Pedestrian Priority. Incorporating bus transit signal priority and pedestrian phase calls into the reward function would capture the full set of competing demands present at real urban intersections.

5) Learned Rollout Policy. A lightweight neural network trained on recorded simulation rollouts could replace the hand-coded LQF heuristic, potentially improving search efficiency while preserving top-level MCTS interpretability [9].

6) Field Deployment. Containerising the backend as a Docker microservice and deploying it on an ARM-based edge device in cooperation with the Pune Municipal Corporation represents the natural culmination of this work.

REFERENCES

[1] S. Shaikh and S. Deokar, “RFID-based emergency vehicle preemption system for urban intersections,” in Proc. Int. Conf. Intelligent Computing and Information Technology (ICITE), Pune, India, 2019, pp. 245–251.

[2] G. Nokhbe Zarei, M. Jalili Ghazizadeh, and S. M. Mortazavi, “Evaluation of emergency vehicle preemption strategies on arterial corridors,” Journal of Transportation Engineering, vol. 143, no. 7, pp. 04017021, 2017.

[3] Y.-Y. Chen, J.-Y. Wang, S.-C. Lo, and W.-T. Sung, “An emergency vehicle traffic signal preemption system considering queue spillbacks along routes and negative impacts on non-priority traffic,” IET Intelligent Transport Systems, vol. 18, no. 8, pp. 1385–1395, Aug. 2024.

[4] B. P. Gokulan and D. Srinivasan, “Distributed geometric fuzzy multiagent urban traffic signal control,” IEEE Trans. Intelligent Transportation Systems, vol. 11, no. 3, pp. 714–727, Sep. 2010.

[5] H. Wei, G. Zheng, H. Yao, and Z. Li, “IntelliLight: A reinforcement learning approach for intelligent traffic light control,” in Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, London, UK, 2018, pp. 2496–2505.

[6] G. Zheng, X. Xiong, H. Zang, J. Feng, H. Wei, H. Yao, Y. Cong, and Z. Li, “Learning phase competition for traffic signal control,” in Proc. 28th ACM Int. Conf. Information and Knowledge Management (CIKM), Beijing, China, 2019, pp. 1963–1972.

[7] W. Chen, S. Yang, W. Li, Y. Hu, X. Liu, and Y. Gao, “Learning multi-intersection traffic signal control via coevolutionary multi-agent reinforcement learning,” IEEE Trans. Intelligent Transportation Systems, vol. 25, no. 11, pp. 15947–15963, Nov. 2024.

[8] H. Luo, Y. Bie, and S. Jin, “Reinforcement learning for traffic signal control in hybrid action space,” IEEE Trans. Intelligent Transportation Systems, vol. 25, no. 6, pp. 5225–5241, Jun. 2024.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

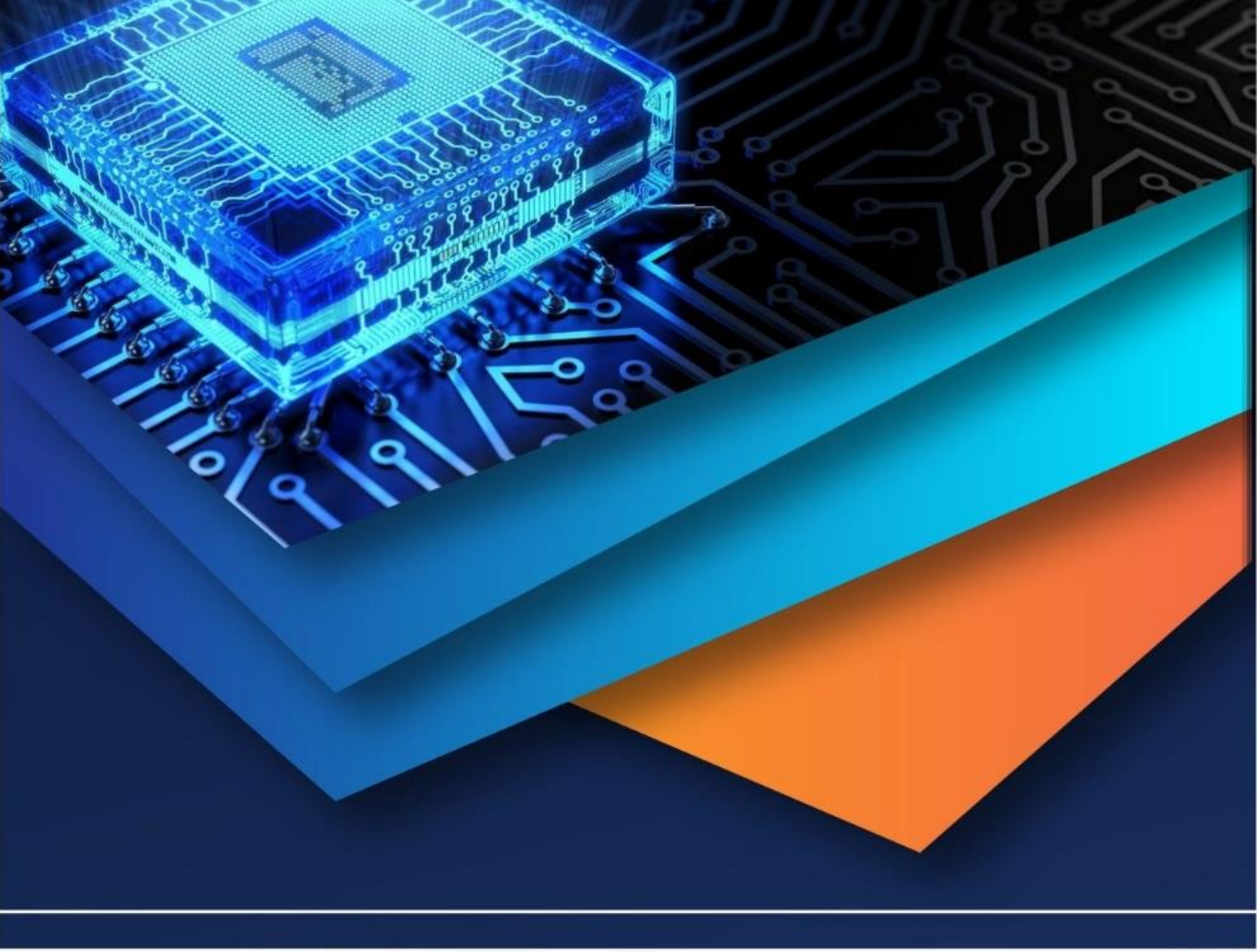
[10] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of Monte Carlo tree search methods,” IEEE Trans. Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[11] X. Liang, X. Du, G. Wang, and Z. Han, “A deep reinforcement learning network for traffic light cycle control,” IEEE Trans. Vehicular Technology, vol. 68, no. 2, pp. 1243–1253, Feb. 2019.

[12] A. Cou¨etoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, “Continuous upper confidence trees,” in Proc. Int. Conf. Learning and Intelligent Optimization (LION), Rome, Italy, 2011, pp. 433–445.

[13] Z. Fu, L. Wen, P. Cai, D. Fu, S. Mao, and B. Shi, "TrafficMCTS: A closed-loop traffic flow generation framework with group-based Monte Carlo tree search," IEEE Trans. Intelligent Transportation Systems, vol. 26, no. 10, pp. 15453–15470, 2025.
[14] A. Ahmad, A. S. Al-Sumaiti, Y.-J. Byon, and K. Alhosani, "Multiple intelligent control strategies for travel-time reduction of connected emergency vehicles," IEEE Trans. Intelligent Transportation Systems, vol. 26, no. 1, pp. 337–353, Jan. 2025.
[15] H. Gu et al., "Large-scale traffic signal control using constrained network partition and adaptive deep reinforcement learning," IEEE Trans. Intelligent Transportation Systems, vol. 25, no. 7, pp. 7619–7632, Jul. 2024.
[16] Highway Capacity Manual 2010, Transportation Research Board, National Research Council, Washington, D.C., 2010.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)