Neural Network Application in Traffic Management

Syed Ahmad Shah
Masters of Applied Machine Learning
Stevens Institute of Technology
Hoboken, USA
ashah@stevens.edu

Christina Cuneo
Masters of Applied Machine Learning
Stevens Institute of Technology
Hoboken, USA
ccuneo@stevens.edu

***Abstract*—In Urban Transportation, adaptive traffic signal control is an ongoing challenge, in order to effectively respond to constantly changing demands. This paper presents an approach to using reinforcement learning to select traffic signal phases at a simulated four-way intersection using a Deep-Q Network. The simulation models a four direction intersection, each with three lanes for left-turning, straight, and right-turning, and signal phases. The agent receives a twenty seven-dimensional state vector encoding per-approach queue lengths, cumulative wait times, the active phase, and time elapsed in that phase. The reward function rewards vehicle throughput while penalizing total wait time and queue length. The model was trained over 1,000 episodes with epsilon-greedy exploration. In early training the average episode reward was -3000 and later -900, showing measurable improvement.**

I. Background

Traffic management encompasses monitoring, coordinating, and controlling the movement of vehicles (including private, public transit, and utility vehicles), pedestrians, cyclists, and other road users so that transportation networks operate safely, efficiently, and reliably. In urban environments, this is especially challenging because traffic patterns are continuously subject to change due to congestion, peak hour demand fluctuations, turning movements, pedestrian crossings, transit and associated wait times, roadway incidents, and the presence of high-occupancy and emergency vehicles.

One of the most important tools in traffic management is traffic signal control. The timing and coordination of traffic lights directly govern vehicle flow, intersection delay, pedestrian safety, queue lengths, and the overall performance of the roadway network. Signal control elements such as protected turn signals, pedestrian crossing intervals, and emergency vehicle preemption directly play a critical role in reducing conflict points and mitigating bottleneck formation. Suboptimal signal timing results in increased vehicle delay, network congestion, excess fuel consumption, and widespread gridlock conditions.

Traditional traffic control methods have historically relied on fixed timing schedules or rule-based actuation strategies. While these approaches perform adequately under stable and predictable demand conditions, they are ill-suited to environments characterized by rapid, unanticipated demand shifts or substantial inter-intersection variability. For example, a corridor of intersections may exhibit stable behavior during peak commuting periods, but becomes significantly less so during special events, construction-related detours, or unanticipated shifts in pedestrian volumes. These limitations motivate the development of more intelligent and adaptive traffic control methodologies.

Modern machine learning techniques, and neural networks in particular, have gained substantial traction in this domain because they can learn from large amounts of traffic data and identify complex, non-linear patterns that can resist characterization by conventional analytical methods. These models can leverage the learned representations to make predictions, support decision-making, and facilitate real-time adjustments to traffic signal operations. Applied to traffic management, neural networks are capable of continuously analyzing changing conditions and creating more adaptive, data-driven signal control strategies.

This capacity is especially valuable during disruptions to nominal traffic behavior, such as incidents, active construction zones, special events, or sudden surges in demand. In these scenarios, adaptive control models may respond more effectively than fixed signal timing systems by dynamically adapting to real-time conditions. As a result, neural-network-based approaches have considerable promise for improving network efficiency, reducing congestion, and enabling smarter traffic signal control in complex urban environments.

II. System Architecture

The project has been organized as an adaptive traffic signal controller that utilizes a reinforcement learning pipeline. Within the repository:

- `code/` Contains the main implementation
- `doc/` Intended for reports and project progress
- `references/` Supporting material
- `assets/` For miscellaneous items, images, plots, etc.

Within the `code/` directory, there exist three directories that are separated by task/purpose:

- `State/` Defines the traffic environment, legal moves, state representation
- `Neural_Networks/` Contains the Double DQN implementation and supporting analysis notebooks for training curve visualization, and baseline controller comparison against Fixed-Time Control, Max-Pressure Control, and Webster's Method
- Contains trained models separated by [model, epoch iterations]

- `Simulation/` Evaluations of the trained model in different situations and environments
- Visual representation of a 4-way intersection, to provide a better view of the environment and state

The configuration layer in "traffic_config.py" defines the legal intersection phases, reward weights, min/max green light durations, and normalization constants that persists across all experiments, ensuring standarization and equivalent evaluation. The environment layer in "traffic_env.py" simulates a four approach intersection with left, straight, and right turn lanes on each approach. We discretely identify these as separate states to allow the model to reason about directional demand and lane queues.

The current implementation includes five valid phases: 'all_right', 'AC_forward', 'BD_forward', 'AC_left', and 'BD_left'. Each phase can be paired with one of nine green durations from 10 to 90 seconds in 10 second increments. The environment exposes a 27 dimensional vector, that is composed of 12 lane queue values [4 approaches x 3 lanes], and 12 lane wait-time values. The input to the model is shown below (TQ refers to Total-Queue):

TABLE I
HEAVY A/C STRAIGHT AND RIGHT-DEMAND SCENARIO

App.	LQ	SQ	RQ	TQ	Waits (L,S,R)	Phase / Time
A	1	9	3	13	(18,18,18)	AC_forward / 20
B	0	2	1	3	(4,4,4)	AC_forward / 20
C	2	8	2	12	(16,16,16)	AC_forward / 20
D	1	1	1	3	(5,5,5)	AC_forward / 20

Where we then normalize the input to a 27-value state vector:

TABLE II
TRAFFIC PHASE AND QUEUE DATA

Parameter	Value
A_left_queue	0.050000
A_straight_queue	0.450000
A_right_queue	0.150000
B_left_queue	0.000000
B_straight_queue	0.100000
B_right_queue	0.050000
C_left_queue	0.100000
...	...
C_right_queue	0.100000
D_left_queue	0.050000
B_right_wait	0.033333
C_left_wait	0.133333
C_straight_wait	0.133333
C_right_wait	0.133333
D_left_wait	0.041667
D_straight_wait	0.041667
D_right_wait	0.041667
current_phase	0.200000
time_in_phase	0.222222
last_duration	0.111111

Finally, the trained model then outputs the selected phase to perform:

	action_id	phase	duration_s	q_value	selected
40	40	BD_left	50	-7694.384766	True
13	13	AC_forward	50	-7723.174316	False
41	41	BD_left	60	-7761.450684	False
39	39	BD_left	40	-7767.203613	False
11	11	AC_forward	30	-7772.420898	False
12	12	AC_forward	40	-7780.854492	False
42	42	BD_left	70	-7788.079102	False
33	33	AC_left	70	-7799.320312	False
32	32	AC_left	60	-7813.319824	False
14	14	AC_forward	60	-7822.116699	False

Fig. 1. Model output: selected signal phase for the current step.

The architecture keeps a focus on keeping the environment, learning model, and simulation implementations separate and consistent, to allow us to easily iterate the focus of our solution if required.

III. DESIGN AND ALGORITHM

Our model design choice is the Deep Q-Network, but the decision problem is now richer than in our earlier implementation. Instead of choosing only the next phase, we have the model select both the phase and the duration of the green light for that phase. This adds an extra layer of decision for the model, as it now has to balance the phase, the effect the duration would have on it, and the queue length increase that may occur in other lanes, which may lead to a higher negative penalty on the next iteration.

The DQN itself is relatively lightweight, as a fully connected feedforward Neural Network, with two hidden layers of 64 units each and ReLU activations. The optimizer is Adam with a learning rate of 0.001, and epsilon greedy exploration decay from 1.00 to 0.05. During training, the model would likely choose the best known action from its memory as the next move; however, with our probability “epsilon” we force it to explore several possibilities to expand its search, then as training progresses, we decrease the learning rate to prioritize minimizing the error. The model makes informed decisions based on its previous training data, by utilizing a "Experience Replay Buffer", a dynamic memory bank that stores the model’s past interactions. This is implemented efficiently by utilizing a "deque" with a capacity of 20,000 consisting of tuples. The tuples consist of the following: current state, action, reward, next state, terminated. We utilize a deque, as when new training samples are recorded by the model, we have an O(1) deletion and insertion for the new element.

Assuming a car going through a 30 m wide intersection accelerates from rest to a constant 2 m/s2 it would take approximately 5.48 seconds. Assuming a user delay in recognizing light changes and lag between subsequent cars progressing, we assume it takes an car 5 seconds on average to cross the intersection. Vehicle flow is now tied to the selected green

duration, where straight and right-turn movements release two vehicles every 10-second interval, and left turns release one vehicle per 10 second interval.

The reward function is designed to encourage flow and penalize congestion. We combine the positive reward for cars that pass, with penalties for the total queue length, wait time, and excessive green durations that go beyond 60 seconds. We also add a pressure penalty, where we penalize the model if it favors any one approach neglecting other phases from activating. This allows the model to be more sensitive towards lane imbalances, prolonged wait durations, and inefficient phase timings.

$$R = 5P - 0.03W - 0.5Q - 4O - X + \Pi \tag{1}$$

where

$$W = \text{total wait time across all lanes} \tag{2}$$
$$Q = \text{total queue length across all lanes} \tag{3}$$
$$P = \text{number of cars passed during the selected action} \tag{4}$$
$$O = \text{overtime beyond the 60-second target green} \tag{5}$$
$$X = \text{maximum lane wait time} \tag{6}$$
$$\Pi = \text{pressure-based reward and penalty term} \tag{7}$$
$$\tag{8}$$

IV. Analysis and Discussion

The model was trained over 1000 episodes against the simulated environment. As shown in Fig. 2, episode reward improves substantially from approximately -3000 in early training to under -900 by episode 1000, demonstrating that the agent learns to reduce congestion and wait times over time. Per-episode variance remains high due to the stochastic arrival of vehicles, but the upward trend in the smoothed average confirms consistent policy improvement.

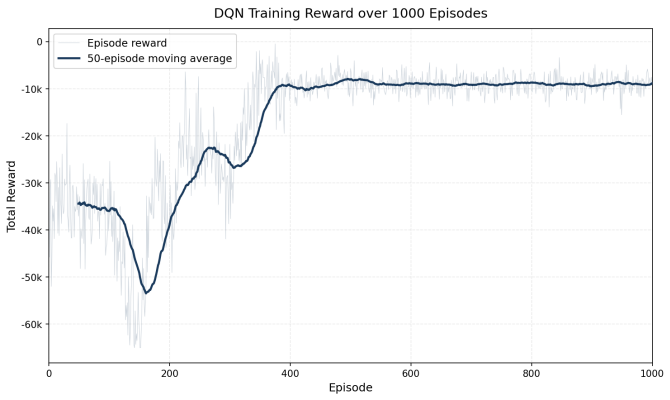

Fig. 2. DQN training reward over 1000 episodes. The 50-episode moving average (dark line) shows steady improvement from approximately -3000 in early episodes to under -900 by episode 1000, despite high per-episode variance.

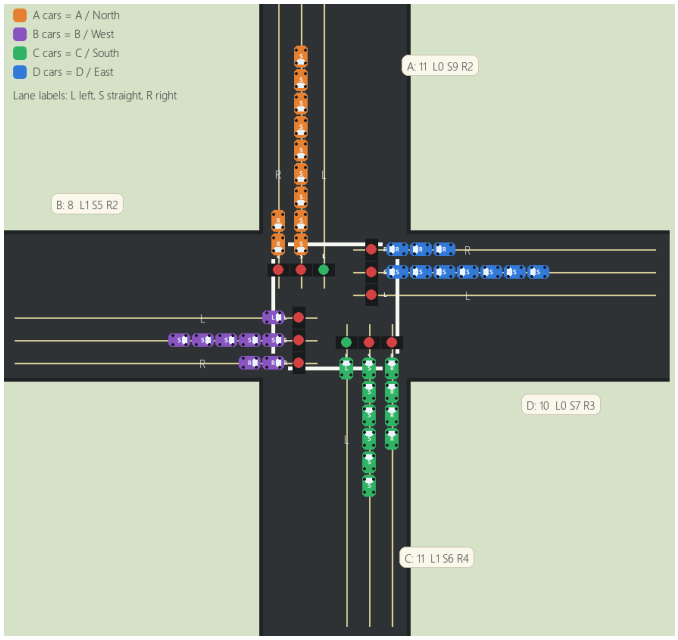

Fig. 3. Pygame visualizer showing intersection state, per-lane queue counts, and active signal phases

Above depicts our live traffic viewer that allows us to step through the decision making process.

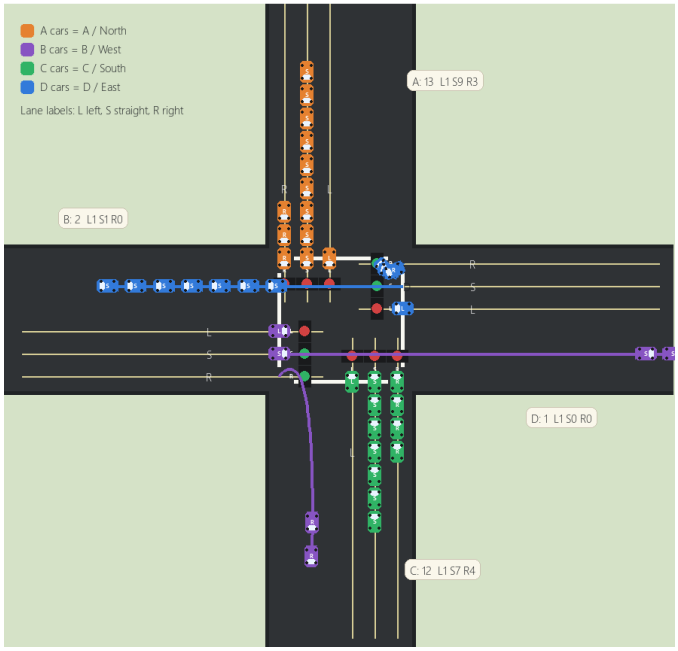

Fig. 4. Decision State

In this image, the model chose to keep the straight/right light open on the BD approach for 50 seconds, allowing 5 cars forward and 2 cars right from the B approach through, and 7 cars forward and 3 cars right from the A approach.

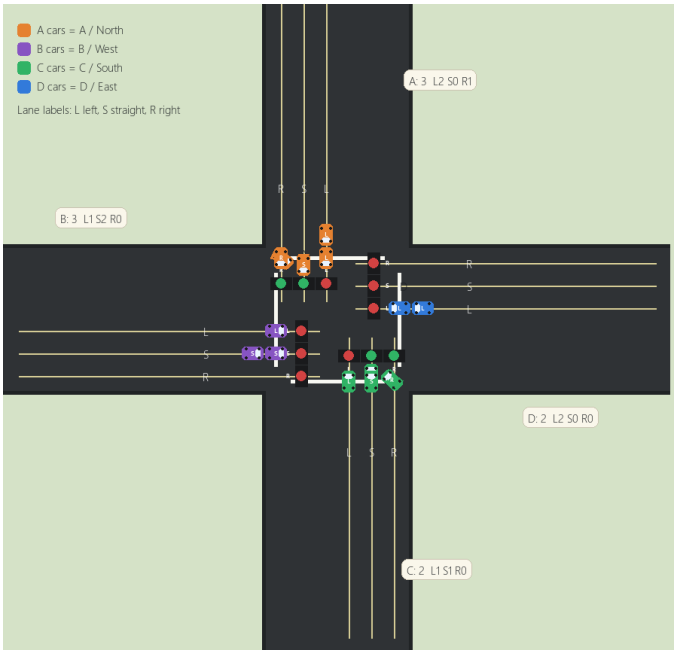

Fig. 5. Result

A. Baseline Controllers

In order to have a meaningful comparison for the trained DQN, three other controllers, all non-learning, were implemented. Each controller is an example of a distinct class of control strategy. The controllers are Fixed-Time Control, which is fully fixed, Max-Pressure which is a mix of fixed and derived, and Webster's, which is fully derived.

1) Fixed-Time Control: Fixed-Time Control (FTC) is the simplest possible signal controller. It uses a preset schedule with no awareness of current traffic conditions. It will cycle through all phases in a fixed order, with each one lasting a consistent 30 seconds regardless of queue lengths or wait times.

The implementation uses an ordered list of phase configurations to define the phase sequence and an integer counter to track elapsed time in the current phase. When the counter reaches the fixed green duration, it resets and the next phase in the list becomes active. There is no sensing, no feedback, and no logic. The controller never reads the traffic state, so it serves as a lower-bound baseline. In theory, any adaptive controller should be able to perform better when demand is uneven.

Per step, this algorithm's computational complexity is $O(1)$ because all it does is increment a counter and move to the next phase in the cycle.

2) Max-Pressure Control: Max-Pressure Control (MPC) [6] is a greedy, reactive controller. At each decision step it reads the current traffic state and selects the phase that would relieve the most cumulative queue pressure.

The pressure score for each candidate phase is computed by summing the current queue lengths of all lanes that phase could potentially serve. The phase with the highest score is selected. There is no lookahead or memory of prior decisions. The green duration is set proportionally to that phase's pressure share relative to the total pressure across all phases, giving higher-pressure phases longer green time. The key data structures are per-lane queue arrays and a per-phase pressure score computed fresh each step from a dictionary mapping phases to the set of lanes they activate.

MPC responds immediately to congestion, but does not take into account fairness or state history. If there is an approach with consistently low pressure, it has potential to be skipped for multiple consecutive steps. For this reason, the starvation scenario is part of the evaluation.

In practice, per step, the computational complexity is $O(P \times L)$, where P is the number of phases and L is the number of lanes. Because P and L are fixed values, at $P = 5$ and $L \leq 6$ in this environment, the complexity is effectively constant.

3) Webster's Method: Webster's Method [7] is an analytically derived controller based on a classical traffic engineering formula for optimal cycle length:

$$C^* = \frac{1.5L + 5}{1 - Y} \tag{9}$$

where L is the total lost time across all phases (seconds of green wasted at the start and end of each phase) and Y is the sum of the critical flow ratios. The resulting cycle length C^* is split across phases proportionally by their demand share to determine individual green durations. Webster's Method is also $O(P)$ per step. It calculates a single value for each phase, evaluates the formula one time, then allocates proportionally. Although each step requires more arithmetic than Max-Pressure, the overall complexity remains the same.

Because the simulation does not track arrival rates explicitly, observed queue pressure is used to represent the flow ratio Y. The implementation stores one scalar pressure value per phase, computes C^* each step from the current queue state, and allocates green time accordingly. Webster's formula was designed for steady-state demand; under rapidly shifting or unbalanced conditions it can produce suboptimal splits, but it remains a standard analytical benchmark in traffic signal research.

B. Training Evaluation

Beyond single-regime evaluation, the controllers were compared across three traffic intensity levels to test how each responds as demand increases. Arrival rates were varied to produce light (0–1 vehicles per step per lane), medium (0–2), and heavy (0–4) traffic regimes. The DQN was trained only on the medium regime, so the light and heavy results measure out-of-distribution performance.

Fig. 6 shows mean average maximum wait time and mean final queue length across all four controllers and all three regimes over 200 episodes each. The DQN maintains a consistent advantage in wait time and queue length across light and medium demand. Under heavy load—arrival rates the model never saw during training—the gap narrows, as the rule-based controllers recompute directly from current conditions at every

step while the DQN relies on patterns learned from lower-demand episodes.

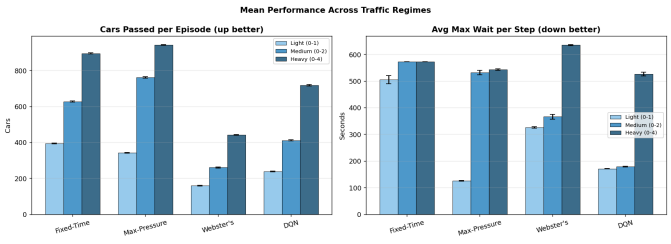

Fig. 6. Mean average maximum wait time and final queue length across light, medium, and heavy traffic regimes. Each bar is the mean over 200 episodes; error bars show 95% confidence intervals.

Fig. 7 shows the phase usage distribution for each controller across regimes. The distribution reveals whether the DQN is appropriately serving left-turn phases (`AC_left`, `BD_left`) or defaulting to straight-and-right phases at their expense.

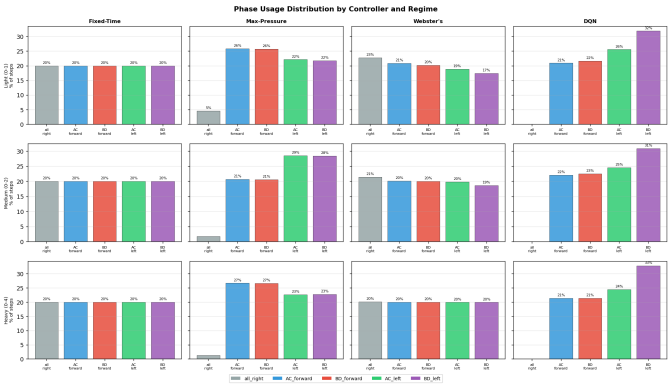

Fig. 7. Phase selection frequency for each controller across light, medium, and heavy regimes. Rows are regimes; columns are controllers.

C. Baseline Comparison

We compared the DQN results to three non-learning controllers, over 50 episodes of 120 steps each. The baselines were (1) Fixed-Time Control (FTC), which cycles through all five phases using a constant 30-second green; (2) Max-Pressure Control (MPC) [6], which selects the phase with the highest weighted queue pressure at each step; (3) Webster's Method [7], which applies the optimal cycle formula $C^* = (1.5L + 5)/(1 - Y)$ using the observed queue pressure as a proxy for demand.

TABLE III
CONTROLLER COMPARISON OVER 50 EVALUATION EPISODES

Controller	Cars Passed	Reward	Avg Max Wait (s)	Final Queue
Fixed-Time	624.5	−40,485	573.0	146.8
Max-Pressure	689.0	−29,869	458.9	82.3
Webster's	257.6	−9,465	363.4	33.6
DQN	408.7	**−5,397**	**180.8**	**32.4**

These results show that DQN is not the best when merely taking a total number of cars passed. Fixed-Time and Max-Pressure both exceeded it, but the DQN still has the lowest average maximum wait time and smallest final queue. This is not unexpected because DQN prioritizes penalizing wait-time and queue buildup more than it rewards pure throughput.

D. Controlled Scenario Probing

We tested the controllers on six different traffic scenarios, too. Each starts from a street designed to isolate a specific traffic condition. By looking at these situations separately, it is easier to compare each controller's response to recognized patterns rather than comparing broadly using purely randomly generated scenarios.

There were six scenarios: (1) balanced light demand; (2) heavy A/C straight and right-turn demand; (3) heavy B/D straight and right-turn demand; (4) A/C left-turn pocket pressure; (5) B/D left-turn pocket pressure; (6) B/D moderate demand but significantly longer wait times,

For each scenario, we recorded the selected scenario, green duration, and Q-value distribution across all controllers.

The left-turn scenarios (4 and 5) are where the DQN's biggest weakness shows up. When a left-turn lane is congested, the model reads that as general congestion for that whole direction. As a result, it picks a through-traffic phase when it really should be giving the left-turn lane its own protected phase. The rule-based controllers don't have this problem because they look at individual lane counts directly, not as an aggregated number.

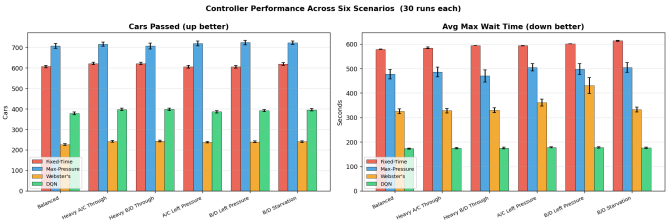

Fig. 8. Controller performance across six controlled scenarios. Each bar is the mean over 30 runs.

E. 2×2 Grid Network

We also ran the DQN on a 2×2 grid of four connected intersections to see how it holds up outside the single-intersection setting it was trained on. Each intersection could feed into the adjacent two. Cars leaving one intersection fed into the next one after each step, so the nodes were actually affecting each other. The same weights were used at all four nodes without any retraining, with each node only seeing its own local state. All four controllers were tested on the grid under the same conditions, which allows us to compare them against each other and against their single-intersection numbers.

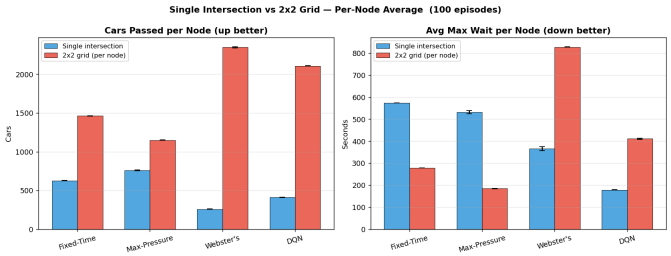

Fig. 9. Per-node average performance on the 2×2 grid compared to single-intersection results.

V. Conclusion

This work demonstrates that a lightweight DQN can learn an adaptive signal control policy for a simulated four-way intersection, reducing average episode reward from approximately -3000 to under -900 over 1000 training episodes. The trained model consistently prioritizes high-pressure approaches and avoids prolonged single-phase repetition. A visual step-through tool was developed to inspect model decisions at each stage, which directly enabled identification of the key limitation described below.

The most impactful avenue for improvement in the neural network lies in expanding the state representation to incorporate lane-level vehicle counts and waits as distinct input rather than aggregating them into a single composite value. This granularity enables the model to better capture directional flow and imbalances across individual lanes. This allows for more precise and context-aware signal time decisions, particularly at high-volume or more complex intersections (additional lanes, more than basic four-way intersections, etc).

VI. Acknowledgments

The authors thank professor Jingxuan Liu for her guidance and support throughout this project.

References

[1] TrafficSimulationOptimizationConsideringDrivingStyles-Sciencedirect, www.sciencedirect.com/science/article/pii/S2772424725000216. Accessed28Apr.2026.

[2] President,KLDAssociates,Inc.300Broadway,HuntingtonStation, NY1174618,www.fhwa.dot.gov/publications/research/operations/tft/ chap10.pdf.Accessed28Apr.2026.

[3] EnergyConsumptionandFaultAnalysisofDataCenterPowerSystem| IEEEConferencePublication|IEEEXplore,ieeexplore.ieee.org/document/ 10245678/.Accessed28Apr.2026.

[4] (PDF)ResearchonOptimizationAlgorithmforUrbanTrafficFlowBasedonComputerSimulation, www.researchgate.net/publication/375849235_Research_on_ Optimization_Algorithm_for_Urban_Traffic_Flow_Based_on_ Computer_Simulation.Accessed28Apr.2026.

[5] Deng,Xuefeng,etal.âĂœTrafficFlowSimulationofModifiedCellularAutomataModelBasedonProducer-ConsumerAlgorithm. âĂİPeerJ.ComputerScience,U.S.NationalLibraryofMedicine,20Sept. 2022,pmc.ncbi.nlm.nih.gov/articles/PMC9575854/.

[6] Varaiya, P. (2013). Max pressure control of a network of signalized intersections. *Transportation Research Part C*, 36, 177–195.

[7] Webster, F.V. (1958). *Traffic Signal Settings*. Road Research Technical Paper No. 39. HMSO, London.

[8] A note on AI use: Claude (Anthropic) was used to edit and draft sentences from outlines by the author, and for final grammar and spell-checking. All changes were reviewed. Anthropic, "Claude," claude.ai (accessed May 2025).