From Static Plans to Self-Evolving Policies: Migrating Legacy Traffic-Control Software to Adaptive Reinforcement Learning

Jeferson Silva
Federal Institute of Bahia
Vitória da Conquista, Brasil
jeferson.caio17@gmail.com

Isaque Andrade
Federal Institute of Bahia
Vitória da Conquista, Brasil
andradeisq1.7@gmail.com

Cleia Libarino
Federal Institute of Bahia
Vitória da Conquista, Brasil
cleialibarino@ifba.edu.br

João Erivando Soares Marques
Federal Institute of Bahia
Vitória da Conquista, Brasil
joaoerivandosm@gmail.com

Luis A. Correia Filho
Federal Institute of Bahia
Vitória da Conquista, Brasil
luis.correia@ifba.edu.br

Kenedy M. Geraldo dos Santos
Federal Institute of Bahia
Vitória da Conquista, Brasil
kenedymarconi@ifba.edu.br

José Alberto Diaz Amado
Federal Institute of Bahia
Vitória da Conquista, Brasil
jose_diaz@ifba.edu.br

Crescencio Lima
Federal Institute of Bahia
Vitória da Conquista, Brasil
crescencio@gmail.com

Abstract

Traffic signal control is, in practice, a software maintenance problem: each signalized intersection is governed by a static, manually maintained control artifact—a fixed-time signal plan hard-coded into the controller's configuration. We frame the modernization of such systems as a problem of software migration and renovation, replacing the static artifact with a self-evolving control policy learned by a Reinforcement Learning (RL) agent, while preserving the system's external interfaces. The legacy artifact is reconstructed rigorously from the Brazilian DENATRAN standard via Webster's method, so the migration is judged against current engineering practice rather than a strawman. Two renovated variants are evaluated—tabular Q-Learning and a Deep Q-Network (DQN)—in scenarios of increasing complexity (an isolated intersection and a coordinated arterial corridor) using the SUMO simulator. Because a learned policy exposes no inspectable source, we treat the visualization of its convergence and runtime behavior as a first-class instrument for comprehending and validating the evolved software. The renovated DQN artifact reduced waiting time by 58.8% and stops by 24.8% at the isolated intersection, and reduced waiting time by 18.9% on the corridor, while behavioral visualizations exposed a fairness/throughput trade-off that aggregate metrics alone would have misread. The results support the migration of static control logic to maintainable, self-evolving artifacts.

Keywords

Software migration, Software renovation, traffic-control, reinforcement learning

1 Introduction

Urban traffic signal control is, fundamentally, a software maintenance problem. Behind every signalized intersection lies a software artifact—a signal plan—that encodes control logic, must be configured, validated, deployed, and periodically revised as the surrounding environment changes. In Brazil, this artifact is implemented as a fixed-time controller: a static configuration of phases and durations, pre-calculated from historical demand and hard-coded into the controller's configuration files [4]. From a software engineering standpoint, this is legacy software in the most literal sense. It operates in an open loop, it embeds assumptions about demand directly in its source, and adapting it to new conditions requires a manual re-engineering cycle: re-counting traffic volumes, re-running analytical dimensioning, and re-writing the configuration by hand.

The maintenance burden of this approach is well documented in economic terms. According to the Institute for Applied Economic Research (IPEA), the costs of congestion in Brazil—aggregating lost productivity, fuel waste, and premature fleet wear—reach figures on the order of 40 billion reais annually [5]. A substantial fraction of this cost is attributable not to physical road capacity but to the inability of static control software to evolve in response to stochastic, time-varying demand. The frequent symptom is the inefficient allocation of green time to idle approaches while queues accumulate elsewhere, a phenomenon known as "green starvation," followed by intersection blockage ("spillback").

This paper frames the modernization of traffic control software as a problem of software migration and renovation: the systematic replacement of a static, manually maintained control artifact with an adaptive, self-evolving one, while preserving the system's external interfaces and operational guarantees. The legacy artifact is the Webster-dimensioned fixed-time plan, expressed declaratively in the simulator's network configuration. The target artifact is a control policy learned by a Reinforcement Learning (RL) agent [10], which adapts its behavior at runtime without source-level modification. The migration is non-trivial because the two artifacts have different maintenance models: the legacy artifact is changed by re-writing code, whereas the renovated artifact changes by re-training, shifting maintenance effort from manual reconfiguration to data-driven supervision.

A recurring difficulty in such migrations is establishing that the renovated software improves on the legacy baseline rather than on a strawman. Much of the literature on learning-based control

compares against ad-hoc or non-optimized fixed-time configurations, which inflate the apparent gains and weaken the case for adoption. We therefore reconstruct the legacy artifact, following the DENATRAN national standard [4] and Webster's method, so that the comparison reflects a real advantage over current engineering practice. Two control strategies are evaluated as migration targets: a tabular Q-Learning policy [12], and a Deep Q-Network (DQN) policy [8] that approximates the value function with a neural network to overcome the state-space explosion that makes tabular methods unmaintainable in realistic settings.

Because the renovated artifact evolves its behavior over time rather than being written once, understanding how it changes becomes a software comprehension task in its own right. A learned policy offers no source code to read; its behavior is legible only through its runtime traces. We therefore treat the visualization of the agent's learning and operational behavior—convergence curves, temporal evolution of waiting time and speed, and stop counts—as a first-class instrument for comprehending and validating the evolved software, not merely as a presentation of results. These visualizations make the otherwise opaque evolution of the control logic observable, supporting the kind of behavioral comprehension that source-level inspection provides for conventional software.

The contributions of this work are:

- A formulation of adaptive traffic control as a **software migration and renovation** problem, in which a static, manually maintained control artifact is replaced by a self-evolving learned policy under preserved interfaces and a rigorously reconstructed legacy baseline.
- An **evolution-aware evaluation** that contrasts the maintainability and runtime behavior of the legacy artifact against two renovated variants (tabular and deep), in scenarios of increasing complexity (an isolated intersection and a coordinated arterial corridor).
- A set of **behavioral visualizations** of the learned software's evolution—convergence, delay, speed, and stops over time—used as comprehension instruments to validate that the renovated artifact improves on the legacy one and to expose the trade-offs it introduces.

The remainder of this paper is organized as follows: Section 2 presents the related work. Section 3 describes the research method. Section 4 presents the results and discussion. Section 5 concludes the paper.

2 Related Work

We organize related work around the lens that frames this paper: the migration of control software from static, manually maintained artifacts toward adaptive ones, and the role of behavioral visualization in comprehending the resulting systems.

2.1 Legacy Control Artifacts and Their Maintenance

Fixed-time signal plans, dimensioned by Webster's method, remain the standard control artifact in Brazilian traffic engineering, valued for their implementation simplicity and robustness under saturation [4]. As software, however, they are static configurations whose evolution requires a full manual re-engineering cycle. Classical adaptive systems such as the British SCOOT and the Australian SCATS represent an intermediate generation: they adjust cycles and offsets incrementally from sensor data, reducing—but not eliminating—the manual maintenance burden, while introducing complex calibration and costly infrastructure dependencies. Kartikasari et al. [6] encoded control knowledge as Fuzzy Logic rules (Sugeno inference) to handle demand uncertainty, improving over fixed time; yet the rules remain hand-authored ("if queue is long, then extend green"), so the maintenance model is still source-level editing that scales poorly to networks of intersections. Across this generation, evolving the controller means rewriting it.

2.2 Migrating Control Logic to Learned Policies

Reinforcement Learning reframes maintenance: control logic is no longer written but learned, shifting effort from manual reconfiguration to data-driven training. Watkins and Dayan [12] established the convergence of tabular Q-Learning for finite Markovian environments, and early work applied it to isolated intersections. The tabular representation, however, does not scale—the combinatorial growth of the state space makes the artifact effectively unmaintainable in realistic networks, a software-evolution dead end analogous to an architecture that cannot accommodate new requirements. Deep RL removes this barrier by approximating the value function with neural networks. Muresan et al. [9] showed that deep policies can ingest high-dimensional inputs such as occupancy matrices or camera images while reducing delay, and Chen et al. [3] proposed the Adaptive Weighted Averaged Double DQN (AWA-DDQN) to stabilize training by mitigating Q-value overestimation. These advances make the learned artifact a viable migration target where the tabular one is not.

At the network scale, the migrated software must coordinate across intersections. Li et al. [7] studied distributed control in arterial corridors via multi-agent systems (MARL), with local agents cooperating toward a global objective, while Zhu et al. [13] used Transfer Learning to port a model trained on one intersection geometry to another—a learned-software analogue of reuse and adaptation across deployments. Exploration strategy also shapes how the policy evolves during training: Tokic [11] and Chen et al. [2] adapt the ϵ-greedy parameter to the agent's uncertainty or to action semantics.

2.3 Visualization for Comprehending Evolving Control Software

A learned policy exposes no readable source, so its evolution and runtime behavior can only be comprehended through visualization of its traces. While the works above report aggregate metrics, the comprehension of *how* a policy converges and how its behavior differs from the legacy artifact over time is, in software terms, a dynamic-behavior visualization problem. We adopt this stance explicitly, using convergence curves and temporal behavioral plots as comprehension instruments for the evolved artifact rather than as mere result displays.

2.4 Positioning

Most cited works compare learning-based controllers against generic or non-optimized fixed-time baselines, which obscures whether the migration yields a real advantage over current practice. This study differs in two respects aligned with software evolution and maintenance concerns. First, it reconstructs the legacy artifact strictly per the DENATRAN Manual [4], so the reported gains reflect a genuine improvement over the best engineering practice in force, strengthening the case for adoption by public managers. Second, it treats the renovated artifact's behavioral evolution as something to be *comprehended and validated through visualization*, not assumed, which is essential when the software in question has no inspectable source.

3 Methodology

The methodology is multidisciplinary, integrating classical traffic engineering to reconstruct the *legacy control artifact* and machine learning to build the *renovated, self-evolving artifact*. We frame the work as a controlled migration: the two artifacts must expose the same external interface to the simulator (the same phases, the same network) so that the comparison isolates the change in control logic. The approach has three stages: (1) modeling the execution environment in which both artifacts run, (2) analytical reconstruction of the legacy fixed-time artifact, and (3) construction of the learned replacement.

3.1 Execution Environment and Migration Scenarios

Both the legacy and renovated artifacts execute in the SUMO microscopic simulator (*Simulation of Urban MObility*) [1], an open-source platform. The renovated artifact interacts with the environment through the TraCI interface (*Traffic Control Interface*) in Python, wrapped with the *Gymnasium* library to standardize the observation and action spaces. To keep the migration interface explicit and reusable, the environment was encapsulated in a custom class (Code 1) that fixes the contract—a discrete action space of phases and a normalized observation vector—shared by every control strategy under test.

```python
class SumoGymEnvironment(gym.Env):
    def __init__(self, net_file, route_file, ...):
        super(SumoGymEnvironment, self).__init__()
        # Actions: Discrete traffic phases
        self.action_space = spaces.Discrete(3)
        # Observation: Vector of 10 variables
        # (Phase, Density, Queues) normalized
        self.observation_space = spaces.Box(
            low=np.zeros(10),
            high=np.ones(10),
            dtype=np.float32
        )
```

Code 1: Environment contract shared by legacy and renovated artifacts (Python/Gymnasium)

We evaluate the migration under two scenarios of increasing complexity, so that maintainability and behavior can be observed both for an isolated artifact and for a set of artifacts that must coordinate.

3.1.1 Scenario 1: Simple Intersection. The orthogonal crossing of two two-way arterial roads, four lanes each (two per direction), with exclusive left-turn lanes and shared lanes for through movements, following the topology in Fig. 1. Virtual inductive sensors of 5 meters were installed in all approach lanes.

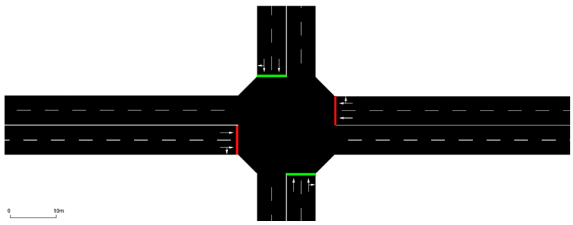

Figure 1: Topology of the simple intersection modeled in SUMO, illustrating approach lanes and detection zones.

3.1.2 Scenario 2: Coordinated Arterial Corridor. Three horizontally aligned intersections, uniformly spaced by 150 meters (Fig. 2). This scenario stresses whether the renovated artifacts can maintain coordination ("green wave") and manage platoon dispersion—an evolution concern, since it tests how well the migrated logic accommodates inter-component dependencies that the legacy artifact handled only through static offsets.

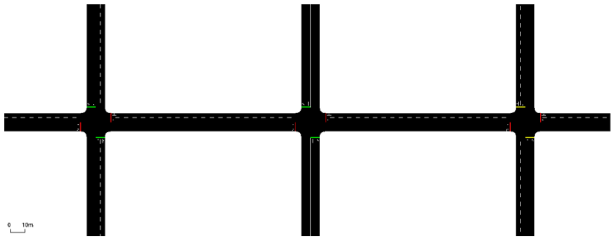

Figure 2: Arterial network with three coordinated traffic lights.

3.2 Reconstructing the Legacy Artifact (Baseline)

To ensure the migration is judged against a faithful baseline and not a strawman, the legacy artifact was dimensioned strictly per the DENATRAN Signaling Manual and Webster's method.

3.2.1 Inter-green Times. The inter-green time (I), composed of yellow time (t_{am}) and all-red time (t_{vg}), was computed from road kinematics. Adopting regulatory speed $v = 50$ km/h (13.88 m/s), perception-reaction time $t_{pr} = 2.0$ s, comfortable deceleration $a = 3.0$ m/s^2, and vehicle length $c = 5.0$ m:

$$t_{am} = t_{pr} + \frac{v}{2a} \approx 2.0 + \frac{13.88}{6.0} \approx 4.3s \rightarrow \text{Adopted } 4s \quad (1)$$

$$t_{vg} = \frac{L_{\text{crossing}} + c}{v} \approx \frac{15.0 + 5.0}{13.88} \approx 1.4s \rightarrow \text{Adopted } 1s \quad (2)$$

This yields a total lost time per cycle of $L = \sum(t_{am} + t_{vg}) = 5s$ per stage.

3.2.2 Saturation Flow and Optimal Cycle. The saturation flow rate (S) was estimated at 3050 vehicles/hour/lane. From volumetric counts in simulations, where critical flow (q_{crit}) reached 1140 veh/h, the occupancy rate (Y) was computed as the sum of critical flow ratios per stage ($y_i = q_i/S$). The optimal cycle time (C_{opt}) follows Webster's formula:

$$C_{opt} = \frac{1.5L + 5}{1 - Y} \tag{3}$$

For the simple intersection this gives $C = 120$ seconds. In the arterial corridor, an *offset* from the distance-speed relationship ($T = D/v$) produced progressive offsets of 34s and 40s for green-wave synchronization. Critically, the legacy control logic is declarative and static: it lives in the SUMO network configuration (`.net.xml`), shown in Code 2. Any change to demand requires editing this artifact by hand—the precise maintenance limitation the migration aims to remove.

```
<tlLogic id="0" type="static" programID="0" offset="0">
    <phase duration="53" state="GGGrrrGGGrrr"/>
    <phase duration="4"  state="yyyrrryyyrrr"/>
    <phase duration="1"  state="rrrrrrrrrrrr"/>
    <phase duration="57" state="rrrGGGrrrGGG"/>
    <phase duration="4"  state="rrryyyrrryyy"/>
    <phase duration="1"  state="rrrrrrrrrrrr"/>
</tlLogic>
```

Code 2: Legacy artifact: static phase logic hard-coded in SUMO (XML)

3.3 Building the Renovated Artifact

The replacement control logic was modeled as a Markov Decision Process (MDP) $\langle S, A, P, R, \gamma \rangle$. Where the legacy artifact encodes behavior in static phases, the renovated artifact encodes it in a reward function and a learned value approximation, so that behavior evolves through training rather than through source edits.

3.3.1 Reward Function (R). Learning is guided by minimizing intersection "pressure" (the difference between inflow and outflow), which indirectly minimizes queues and delay. Code 3 shows the implementation.

```
def calculate_reward(self):
    # Get controlled lanes
    incoming = self.tls.getControlledLanes()
    outgoing = self.tls.getControlledLinks()

    # Calculate Pressure = Veh. Entering - Veh. Exiting
    pressure = sum(getVehicles(l) for l in incoming) - \
               sum(getVehicles(l) for l in outgoing)

    # Objective: Maximize reward (Minimize pressure)
    return -pressure
```

Code 3: Reward calculation defining the renovated artifact's objective (Python)

This formulation penalizes accumulation in entry lanes, driving the agent to clear the most saturated approaches.

3.3.2 Deep Q-Network (DQN) Architecture. Tabular Q-Learning does not scale: its memory grows with the state space, making it an unmaintainable target for realistic deployments. The DQN variant approximates $Q(s, a)$ with a neural network (MlpPolicy), detailed in Fig. 3:

- **Input Layer:** sized to the state vector (10 normalized variables: phase, density, queues).
- **Hidden Layers:** two dense (*fully connected*) layers of 64 neurons each, with ReLU activation ($f(x) = \max(0, x)$) for non-linearity and to mitigate gradient vanishing.
- **Output Layer:** linear, providing estimated Q-values for each of the 3 actions.

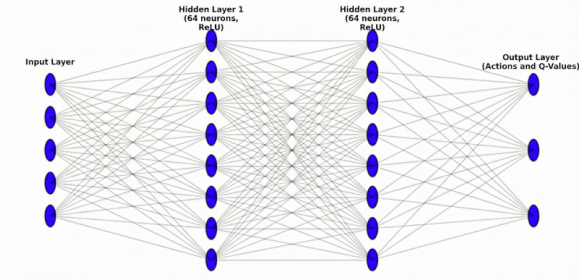

Figure 3: MlpPolicy neural network architecture of the renovated DQN artifact: sensory input, hidden layers with ReLU, and Q-value output.

Training used the *Stable Baselines3* library. The training loop (Code 4) employs an *Experience Replay* buffer to break temporal correlation among samples and stabilize convergence.

```
model = DQN("MlpPolicy", env, verbose=1,
            learning_rate=0.001,
            buffer_size=50000,
            learning_starts=1000,
            target_update_interval=500,
            exploration_fraction=0.1,
            exploration_final_eps=0.02)

# Training for 100,000 time steps
model.learn(total_timesteps=100000)
model.save("dqn_single_intersection")
```

Code 4: DQN training loop—how the renovated artifact is produced and persisted

4 Results and Discussion

Experiments ran on a workstation with an Intel Core i5 processor and 6GB of RAM. Each renovated artifact was trained over 500 simulation episodes, each corresponding to 30,000 time steps, exposing the agent to diverse traffic regimes. Throughout, we read the figures as *behavioral visualizations of the evolving software*: because a learned policy has no inspectable source, these traces are the primary means of comprehending how the renovated artifact behaves and of validating that it improves on the legacy one.

4.1 Visualizing Convergence and Stability

The learning curves expose how each renovated artifact evolves toward a stable policy. Tabular Q-Learning converged slowly and noisily, stabilizing only after episode 19 in the complex scenario, because it must visit each state-action pair repeatedly to update its table—an evolution path that does not scale. The DQN artifact generalized across unvisited states and converged to a stable policy around episode 11. Made observable through visualization, this difference is the comprehension evidence that the deep variant is the more maintainable migration target: it reaches stable behavior faster and degrades more gracefully as state complexity grows.

4.2 Scenario 1: Simple Intersection

Table 1 consolidates the results (mean ± standard deviation). The legacy fixed-time artifact, though Webster-optimized, showed the worst global behavior, with an average waiting time of 95.29s; qualitative inspection of the simulation showed its static 120s cycle repeatedly assigning green to empty approaches. The renovated DQN artifact reduced waiting time by **58.8%** (39.24s) relative to the legacy baseline and outperformed Q-Learning by 41%. It also sharply reduced the standard deviation of waiting time, indicating more equitable service. Fig. 4 visualizes this: the DQN trace (bottom line) holds minimal delay across the run, making the behavioral improvement legible at a glance.

Table 1: Average Results and Standard Deviation – Simple Intersection

Metric	Fixed Time	Q-Learning	DQN
Total Stops	9.33 (±4.72)	8.19 (±3.90)	**7.01** (±3.77)
Waiting Time (s)	95.29 (±75.6)	66.61 (±33.1)	**39.24** (±27.9)
Speed (m/s)	7.88 (±1.63)	**7.99** (±0.77)	7.89 (±1.37)

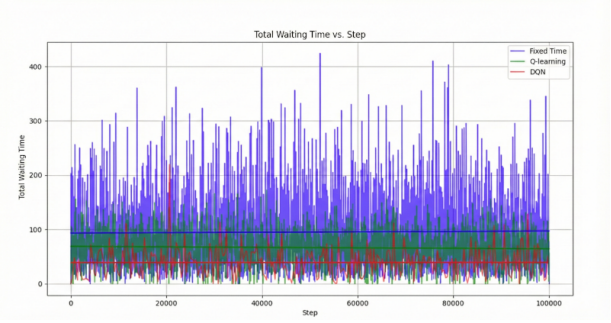

Figure 4: Behavioral visualization of waiting-time evolution at the simple intersection. The renovated DQN artifact stabilizes at minimal delay after the initial exploration phase.

The effect on average speed was also positive. Fig. 5 shows the DQN artifact holding a more constant speed, avoiding the zero-speed valleys that mark saturation under the legacy controller.

Figure 5: System average speed at the simple intersection, visualized over time.

4.3 Scenario 2: Arterial Network

In the corridor, control complexity grows with flow interdependence. The legacy fixed-time artifact, even with a theoretical *offset* for a green wave, suffered from stochastic speed variability, averaging 12.20 stops/vehicle. The renovated DQN artifact achieved implicit coordination: by observing the global state (queues across all approaches), it learned to release flow to avoid upstream blockage (*gridlock*), reducing waiting time by **18.9%** (Table 2) and stops by 20.1%.

Table 2: Average Results – Arterial Network (3 Intersections)

Metric	Fixed Time	Q-Learning	DQN
Total Stops	12.20	10.96	**9.74**
Waiting Time (s)	94.94	84.97	**76.94**
Speed (m/s)	**7.32**	6.93	5.97

Fig. 7 visualizes the consistent reduction in stops delivered by the renovated artifact. Notably, the legacy artifact achieved the highest average speed (7.32 m/s) in this scenario. Visualizing the per-road behavior explains why: the static plan strongly prioritizes the main arterial at the expense of cross streets, so main-road vehicles travel fast (green wave) while cross-street vehicles suffer extreme delay. The DQN artifact, minimizing global delay, balanced service across roads—yielding slightly lower average speed but a globally fairer and more efficient system (lower total waiting time). This trade-off is only apparent through behavioral visualization; the aggregate speed figure alone would misrepresent the migration as a regression.

4.4 Threats to Validity

The study is conducted in the SUMO simulation environment, which limits the extent to which the results can be generalized to real-world deployments. Although the legacy artifact is reconstructed according to the national standard and both control strategies are evaluated under the same simulation interface, real traffic-control systems involve factors that are not represented in the current evaluation, including sensing noise, communication delays, actuation latency, and hardware constraints.

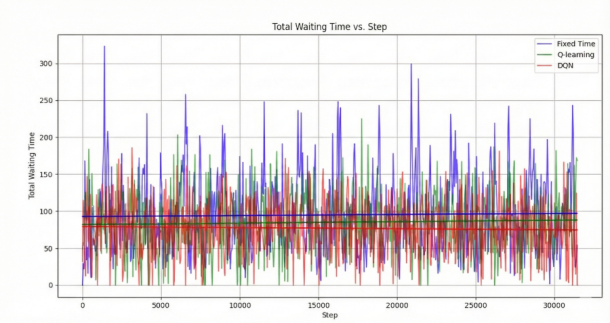

Figure 6: Waiting-time comparison in the arterial network, visualized over time. The renovated DQN artifact stays consistent under high coordination complexity.

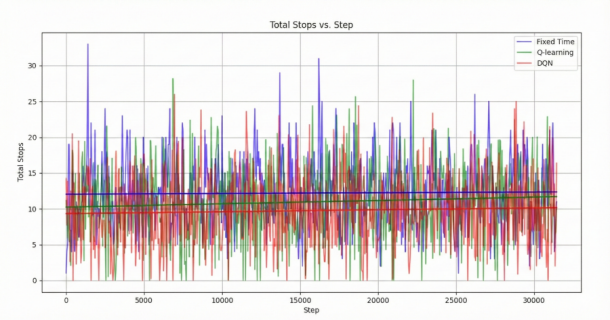

Figure 7: Total stops in the arterial network. Fewer stops indicate smoother operation and lower fuel consumption.

In addition, the reward function is based on pressure minimization, which represents one design choice among other possible reinforcement-learning objectives. This formulation encourages the reduction of traffic accumulation, but alternative objectives, such as waiting time, throughput, or fairness, could lead to different learned behaviors and performance trade-offs. Therefore, the observed improvements should be interpreted in the context of the selected simulation environment and reward formulation.

5 Conclusion and Future Work

This paper framed the modernization of urban traffic signal control as a problem of software migration and renovation: replacing a static, manually maintained control artifact—the Webster-dimensioned fixed-time plan hard-coded in the simulator's configuration with a self-evolving learned policy under a preserved environment interface. Evaluated against a rigorously reconstructed legacy baseline, the renovated Deep Q-Network artifact improved behavior across all scenarios, validating the migration rather than a strawman comparison.

The gains were substantial and, importantly, comprehensible through behavioral visualization. At the isolated intersection, the renovated artifact cut waiting time by nearly 60%, exposing the capacity waste baked into the legacy artifact's rigid cycle. In the arterial corridor, the deep variant achieved coordination implicitly, without the explicit green-wave modeling the legacy artifact depended on, evidencing better accommodation of inter-component dependencies—an evolution property. The visualizations were not decorative: they were the instrument that made the evolved, sourceless software legible, including a fairness/speed trade-off that aggregate metrics alone would have misread.

The comparison with tabular Q-Learning reinforced a maintainability argument: the tabular representation is an evolutionary dead end, unable to scale with state complexity, whereas the deep approximation generalizes and converges faster, making it the maintainable migration target.

We plan validation on real hardware (*Hardware-in-the-Loop*), integrating the renovated artifact with commercial controllers; cooperative Multi-Agent (MARL) architectures for explicit city-scale coordination; richer evolution-aware visualizations to support continuous comprehension of policy drift over a deployment's lifetime; and robustness analysis against sensor and communication failures.

Artifact Availability

The simulation models (SUMO networks and routes), training scripts, and analysis code are available at https://github.com/crescenciolima/VEM-2026

Acknowledgments

Generative AI tools were used to assist with text editing during manuscript preparation, including *ChatGPT* and *Claude*. All experimental design, evaluation rubrics, data analysis, result interpretation, and scientific claims are the sole responsibility of the authors.

References

[1] Michael Behrisch et al. 2011. SUMO – Simulation of Urban MObility: An Overview. In *Proceedings of SIMUL 2011*.
[2] Tse-Leung Chen et al. 2022. Exploiting Semantic Epsilon Greedy Exploration Strategy in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2201.10803* (2022).
[3] Youqing Chen et al. 2023. Traffic signal optimization control method based on adaptive weighted averaged double deep q network. *Applied Intelligence* (2023), 1–22.
[4] DENATRAN. 2020. *Manual Brasileiro de Sinalização de Trânsito - Volume V: Sinalização Semafórica.* Departamento Nacional de Trânsito.
[5] Instituto de Pesquisa Econômica Aplicada. 2020. *Mobilidade Urbana no Brasil.* IPEA.
[6] R. Kartikasari, G. Prakarsa, and D. Pradeka. 2020. Optimization of traffic light control using fuzzy logic sugeno method. *International Journal of Global Operations Research* 1 (2020).
[7] J. Li et al. 2023. Distributed signal control of arterial corridors using multi-agent deep reinforcement learning. *Transportation Research Part C: Emerging Technologies* (2023), 146–164.
[8] Volodymyr Mnih et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.
[9] M. Muresan, L. Fu, and G. Pan. 2019. Adaptive traffic signal control with deep reinforcement learning: an exploratory investigation. *arXiv preprint arXiv:1901.00960* (2019).
[10] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
[11] Michel Tokic. 2010. Adaptive ϵ-greedy Exploration in Reinforcement Learning Based on Value Differences. In *KI 2010: Advances in Artificial Intelligence.*
[12] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8 (1992), 279–292.
[13] Z. Zhu et al. 2023. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).