### **Reinforcement learning model in Netlogo to simulate traffic**

### **1. Introduction**

Traffic congestion and vehicular emissions are critical challenges in urban planning and environmental sustainability. This report presents a reinforcement learning (RL)-based traffic simulation model developed in NetLogo, extending a simpler model found online. The model incorporates three distinct vehicle types (sedans, trucks, and sports cars) with unique acceleration, deceleration, and emission profiles, operating on a three-lane circular highway. A deep Q-network (DQN) architecture, integrated via Python’s Keras library, enables agents to learn lane-changing and speed adjustment strategies while minimizing collisions and maximizing traffic flow efficiency. Emissions are dynamically calculated based on instantaneous speed, acceleration patterns, and vehicle-specific multipliers. Experimental results demonstrate how vehicle heterogeneity and RL hyperparameters influence traffic congestion patterns and cumulative emissions.

#### **Objectives**

1. Extend the original single-lane model to a multi-lane traffic simulation where agents autonomously learn lane-changing and speed control strategies.
2. Introduce vehicle heterogeneity in the model, with distinct acceleration/deceleration profiles and emission rates.
3. Quantify real-time emissions based on speed fluctuations and vehicle type.
4. Analyze how RL hyperparameters (exploration rate, batch size) affect traffic dynamics.

### **2. Background**

#### **Traffic Flow Theory**

The Nagel-Schreckenberg model (1992) established that traffic jams emerge spontaneously from minor perturbations in vehicle density and speed—a phenomenon termed “phantom traffic jams.”

#### **Reinforcement Learning in Traffic**

RL agents learn policies by maximizing cumulative rewards through trial and error. In this model:

* State space: Includes self speed, distance to next car, and neighbor speed.
* Action space: Discrete choices to accelerate, decelerate, or maintain speed.
* Reward function: Logarithmic reward for speed incentivizes smooth flow while penalizing stops.

The DQN algorithm, using experience replay and target networks, stabilizes training.

#### **Emission Modeling**

Vehicle emissions depend non-linearly on speed and acceleration. This is a simple formula that captures this relation, used to calculate the emission of each vehicle:



where v is speed and a is acceleration.

### **3. Model Design**

### **3.1 System Overview**

The simulation environment models a three-lane circular highway in NetLogo, so vehicles exiting the right edge reappear on the left, maintaining constant traffic density. Vehicles (agents) operate with different dynamics and emissions profiles. Key components include:

#### **3.1.1 World Geometry**

There are three horizontal lanes centered at pycor = -3, 0, 3 (NetLogo’s patch coordinates), each represented by white patches.

#### **3.1.2 Vehicle Heterogeneity**

Three vehicle types are modeled, each with unique kinematic and environmental profiles. In particular:

* Trucks mimic real-world heavy vehicles with reduced acceleration capacity
* Sports cars exhibit "impatient driver" behavior, prioritizing speed over fuel efficiency.
* Vehicle shapes provide visual differentiation during simulation.

3.2 Reinforcement Learning Architecture

### **Theoretical Framework**

Reinforcement Learning (RL) is a machine learning paradigm where agents learn optimal policies by interacting with an environment to maximize cumulative rewards. In this project, the Deep Q-Network (DQN) algorithm is employed, which combines Q-learning with deep neural networks to handle high-dimensional state spaces. Below are the key theoretical components and their implementation-specific adaptations:

#### **3.2.1. Markov Decision Process (MDP) Formulation**

The traffic simulation is modeled as an MDP defined by:

* State (s): A vector describing the agent’s current situation (e.g., speed, proximity to other cars).
* Action (a): Discrete choices available to the agent (decelerate, maintain speed, accelerate).
* Reward (r): A scalar feedback signal incentivizing desirable behavior.
* Transition Dynamics: The probability P(s′∣s,a) of moving to state s′ after taking action a in state s.

Implementation:

* The state vector is constructed dynamically using NetLogo’s inputs list (e.g., [-> speed], [-> distance next-car]).
* Actions are selected via an ε-greedy policy, balancing exploration (random actions) and exploitation (actions predicted by the DQN).

#### **3.2.2. Q-Learning and Deep Q-Networks**

#### **Q-Learning**: An agent learns a policy by updating a Q-table (state-action values) using the Bellman equation:



#### balancing exploration and exploitation to maximize discounted future rewards.

#### **Deep Q-Learning (DQN)**: Replaces the Q-table with a neural network to approximate Q-values, enabling handling of large/continuous state spaces.

#### **Experience Replay**: Stores past experiences (state, action, reward, next state) in a buffer and samples them to train the network, improving stability.

#### **Goal**: Learn an optimal policy by iteratively improving Q-value estimates, enabling the agent to make better decisions over time.

All these concepts are used and implemented in the model.

#### **3.2.3. State Space Design**

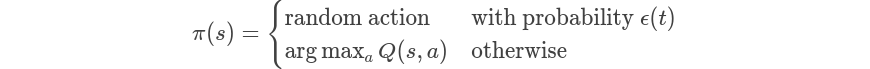
The state vector includes:

1. Self Speed: Current speed of the agent.
2. Distance to Next Car: Distance to the nearest vehicle ahead.
3. Next Car’s Speed: Speed of the leading vehicle.
4. Exploration Rate (Optional): Time-decaying exploration rate ϵ, modeling time-dependent traffic patterns.
5. Simulation Tick: Current step counter, helping agents learn temporal patterns.

#### **3.2.4. Action Space and Policy**

* Actions:
  + 0: Decelerate
  + 1: Maintain speed
  + 2: Accelerate
* ε-Greedy Policy:

The ε-Greedy Policy balances exploration (trying random actions to discover new strategies) and exploitation (using learned knowledge to maximize rewards). Here’s the formula:

* Decay Formula:



* + ϵ0: Initial exploration rate (e.g., 0.3 = 30% random actions).
  + decay\_rate: How fast exploration decreases over time t (ticks).

#### **3.2.5. Reward Function Design**

The reward function incentivizes smooth traffic flow while penalizing collisions and excessive emissions:

r(s,a)=log(speed+ϵ)

where ϵ prevents numerical instability.

Rationale:

* The logarithmic function encourages agents to maintain non-zero speeds without promoting dangerous acceleration.
* Emissions are tracked separately (via total-emissions) but not directly penalized in rewards, allowing post-hoc analysis of environmental impact.

#### **3.2.6. Experience Replay and Training**

Experience Replay breaks temporal correlations by sampling random mini-batches from a buffer of past experiences.

Training Steps:

1. Store Transitions:

to remember

ask turtles [ set next-state map runresult inputs ]

let data [ (list state action reward next-state) ] of turtles

py:set "new\_exp" data

(py:run "memory.extend(new\_exp)")

1. Sample Batch:

sample\_ix = np.random.randint(len(memory), size=batch\_size)

inputs = np.array([memory[i][0] for i in sample\_ix])

1. Compute Target Q-Values:

next\_state\_rewards = model.predict(next\_states)

next\_state\_qs = np.max(next\_state\_rewards, axis=1)

targets[np.arange(targets.shape[0]), actions] = rewards + discount \* next\_state\_qs

1. Update Network:

model.train\_on\_batch(inputs, targets)

#### **3.2.7. Hyperparameters and Stability**

The neural network used in the model to approximate the Q-values has various hyperparameters that can be tuned for the training process:

| Hyperparameter | Value(default) | Role in Training |
| --- | --- | --- |
| Learning Rate (lr) | 0.001 | Small steps prevent Q-value divergence. |
| Batch Size | 32 | Balances noise reduction and computational efficiency. |
| Discount Factor (γ) | 0.95 | Emphasizes near-term rewards while considering future states. |
| Memory Size | 10,000 | Large buffers improve sample diversity. |

### **3.3 Lane-Changing Algorithm**

Lane changes are autonomously initiated by agents to bypass slower traffic, balancing flow efficiency and collision avoidance.

#### **3.3.1 Theoretical Basis**

Lane-changing decisions in real traffic depend on:

1. Incentive: Faster speed in adjacent lanes.
2. Safety: Sufficient gap to avoid side/rear collisions.

#### **3.3.2 Implementation Logic**

1. Lane Identification:
   * Current lane: pycor of the agent’s patch.
   * Target lanes: current\_lane ± 3 (e.g., lane 0 → lanes -3 or +3).
2. Safety Check:
   * Buffer zone: 5-patch (unit) margin ahead/behind in target lane.
   * Collision avoidance: No vehicles in buffer zone (NetLogo code):
3. Post-Change Acceleration:
   * Successful lane changes trigger acceleration to maximize flow

### **3.4 Emission Calculation**

#### **Theoretical Basis**

Vehicle emissions are modeled using a physics-inspired formula that accounts for three key factors:

1. Base Emissions: Proportional to instantaneous speed, reflecting fuel combustion rates at steady-state operation.
2. Acceleration Penalty: Quadratic dependence on acceleration/deceleration magnitude, capturing increased fuel consumption during rapid speed changes.
3. Speed Inefficiency: Penalizes deviations from an optimal speed where emissions are minimized.

The emission formula is defined as:



Where:

* v: Current speed
* a: Instantaneous acceleration (Δv/Δt, approximated as vcurrent−vprevious)
* vopt: Optimal speed
* α,β,γ: Calibration coefficients
* Mtype: Vehicle-type multiplier (sedan=1.0, truck=1.5, sports=2.5)

#### **Emission Tracking**

* Per-Tick Emissions: Aggregated in current-tick-emissions for real-time monitoring.
* Cumulative Emissions: Stored in total-emissions for environmental impact analysis.

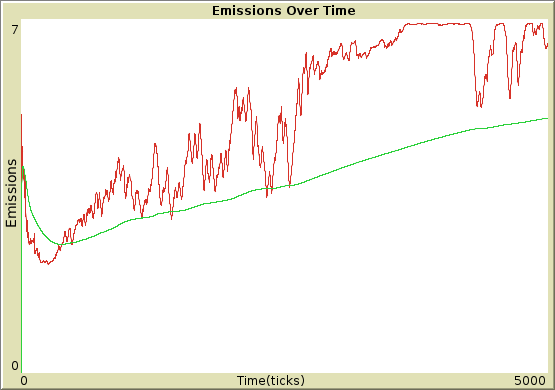
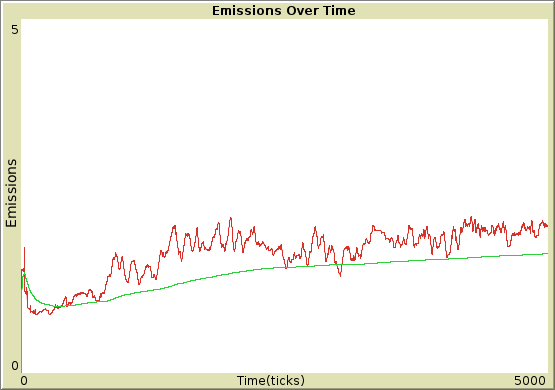
### **3.5 Integration of Components**

The synergy between RL-driven speed control, lane-changing logic, and emission tracking enables multi-objective analysis:

1. Traffic Flow: Measured via average speed and throughput (vehicles/tick).
2. Environment: Cumulative emissions across vehicle types.

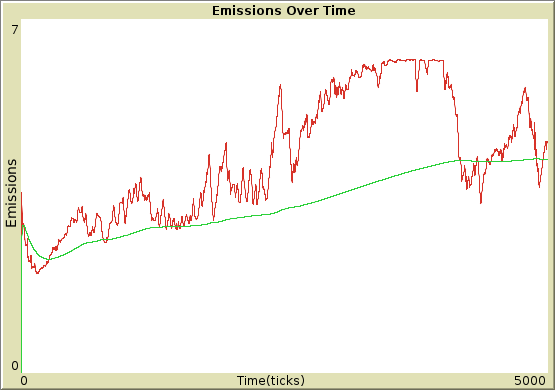
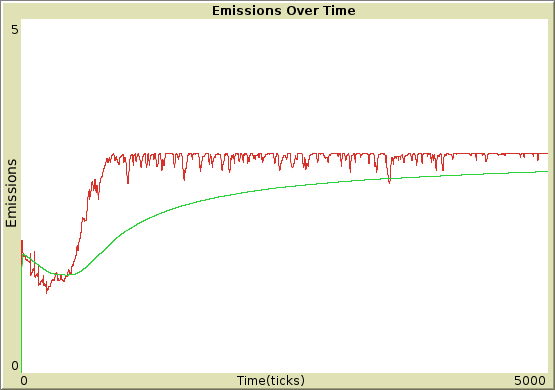
### 4 Simulation results:

In this section are presented the result of various simulations with different hyperparameters and design choices, the focus is on the emissions registered in these experiments and how the various choices affected these. Low and high traffic conditions are modeled in Netlogo playing with the number of cars on the road, in low traffic conditions there are up to 11 vehicles on the lanes and in high traffic conditions these go up to 25 vehicles.



#### **Shallow Model(Low vs. High Traffic)**

Description:  
Emission profiles for the shallow neural network model (3×36-unit hidden layers, Adam optimizer, lr=0.001, ε₀=0.3, decay=0.005, batch=32, γ=0.95) under low traffic (on the left) and high traffic (on the right) conditions. In low traffic, emissions exhibit erratic fluctuations due to uncoordinated acceleration/deceleration events, reflecting the model’s limited policy learning. High traffic amplifies these inefficiencies, with frequent emission spikes corresponding to phantom traffic jams. These results indicate that the model wasn’t able to learn an effective strategy, it would probably need more time to converge.



#### **Deep Model (Low vs. High Traffic)**

Description:  
Emission trends for the deep neural network model (36→64→64→36-unit hidden layers, same hyperparameters) under low (on the left) and high (on the right) traffic.

In low traffic, emissions stabilize significantly compared to the shallow model, with fewer acceleration-driven spikes, demonstrating improved policy generalization.

High traffic shows reduced emission magnitudes with respect to the shallow model and shorter-lived congestion peaks, though persistent oscillations indicate partial failure to resolve traffic waves. The deeper architecture enables better anticipation of traffic dynamics.

### 

#### **Emission-Aware Deep Model (Low vs. High Traffic)**

Description:  
Emission profiles for the emission-aware deep model with same deep architecture + emission-penalized reward (λ=0.3, ε₀=0.3, decay=0.01, batch=32) under low (on the left) and high (on the right) traffic. Low traffic achieves near-optimal emissions with minimal variability, as agents prioritize steady speeds and avoid unnecessary acceleration. Emissions rise significantly under congestion, with erratic spikes from unavoidable stop-and-go cycles. The reward penalizes acceleration but the penalty is insufficient (λ=0.3), there is a persistent training instability and the emission spikes do not plateau after a certain number of ticks as for the low traffic conditions.

### **Summary of Emissions Analysis Across Models**

This study explored the environmental and behavioral impacts of three neural network architectures shallow, deep, and emission-aware deep in a multi-lane traffic simulation, comparing their performance under low and high traffic densities. By analyzing six distinct emission plots (two per model), key patterns emerged that highlight the relations between traffic dynamics, reinforcement learning (RL) design, and sustainability outcomes.

#### **1. Shallow Model**

* Low Traffic: Emissions were erratic, with frequent minor spikes from unnecessary acceleration/deceleration. Agents failed to learn steady cruising, prioritizing short-term speed gains over efficiency. Emissions fluctuated wildly without reaching stability.
* High Traffic: Emissions showed chaotic peaks during phantom traffic jams. The shallow network’s limited capacity caused agents to overreact to congestion, resulting in stop-and-go cycles and runaway emissions.
* Visual Observations: Vehicles clustered unpredictably, with frequent backward-moving jams. Sports cars exacerbated instability by accelerating aggressively.

#### **2. Deep Model (No Emission Penalty)**

* Low Traffic: Emissions stabilized significantly. Agents learned smoother acceleration and better anticipation of nearby cars, though occasional spikes persisted during merges.
* High Traffic: Emissions plateaued lower than the shallow model. Congestion peaks were shorter-lived.
* Visual Observations: Emergent lane specialization (trucks in one lanes, sports cars in another). Traffic waves still formed but dissolved faster.

#### **3. Emission-Aware Deep Model**

* Low Traffic: Emissions plummeted with near-elimination of spikes. Agents optimized speed to stay near the optimal-speed, minimizing acceleration and inefficiency.
* High Traffic: Emissions do not plateau and there is a very chaotic behavior. This happens probably because the emission penalty is not sufficient to dissuade the cars from trying to reach higher speeds, which are considered positively in the reward.

### **Key Comparisons and Trends**

1. Network Depth Enhances Coordination:
   * Deeper models reduced emissions in high traffic compared to shallow models, demonstrating superior state representation and anticipation.
   * Agents developed emergent strategies like buffer spacing and gradual merging, absent in shallow networks.
2. Traffic Density Amplifies Model Flaws:
   * All models struggled in high traffic, but deeper models generally mitigated congestion better than the others.

### **Conclusions and Implications**

This study demonstrates that RL architecture and reward design are interesting tools for sustainable traffic management. The deep model achieved the best balance of efficiency and environmental impact. Key takeaways:

* Policy Design Matters: Deepening networks improves traffic flow, but explicit emission penalties are essential for sustainability.
* Real-World Applicability: These models mirror real-world phenomena (e.g., phantom jams).

#### **Future Directions**

* Mixed Fleets: Introduce electric/hybrid vehicles with lower emission multipliers.
* Human-in-the-Loop Testing: Integrate human driver behavior to validate RL policies.