SUMO

REPORT OF PROJECT

Reinforcement Learning for Lane-Changing Decisions on highways.

ESTiN

Our Team

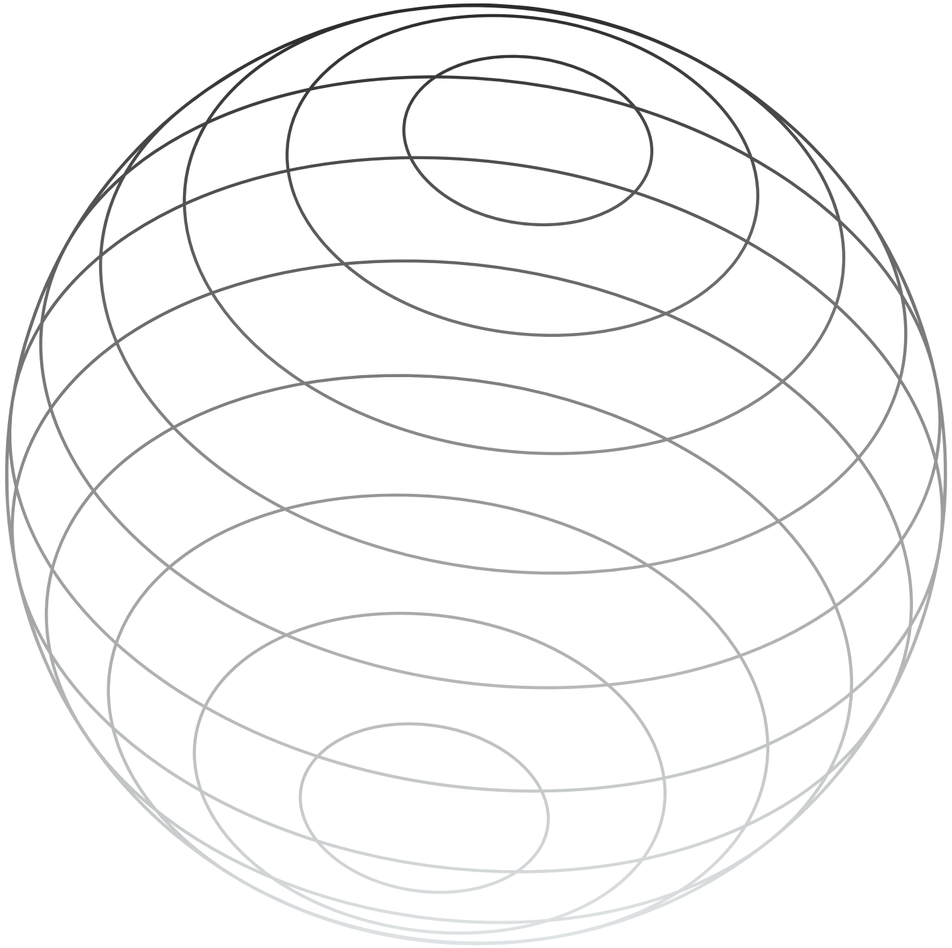
- Benoudjit Lyne

- Chikhi Mey Aya

- Djouzi Nour El Houda

- Mahdaoui Malak Ines

- Sadok Feriel



Agenda

01 Overview

02 Study of Q-Learning and PPO

03 Detailed RL problem formulation

04 Hyperparameters and ANN Architecture

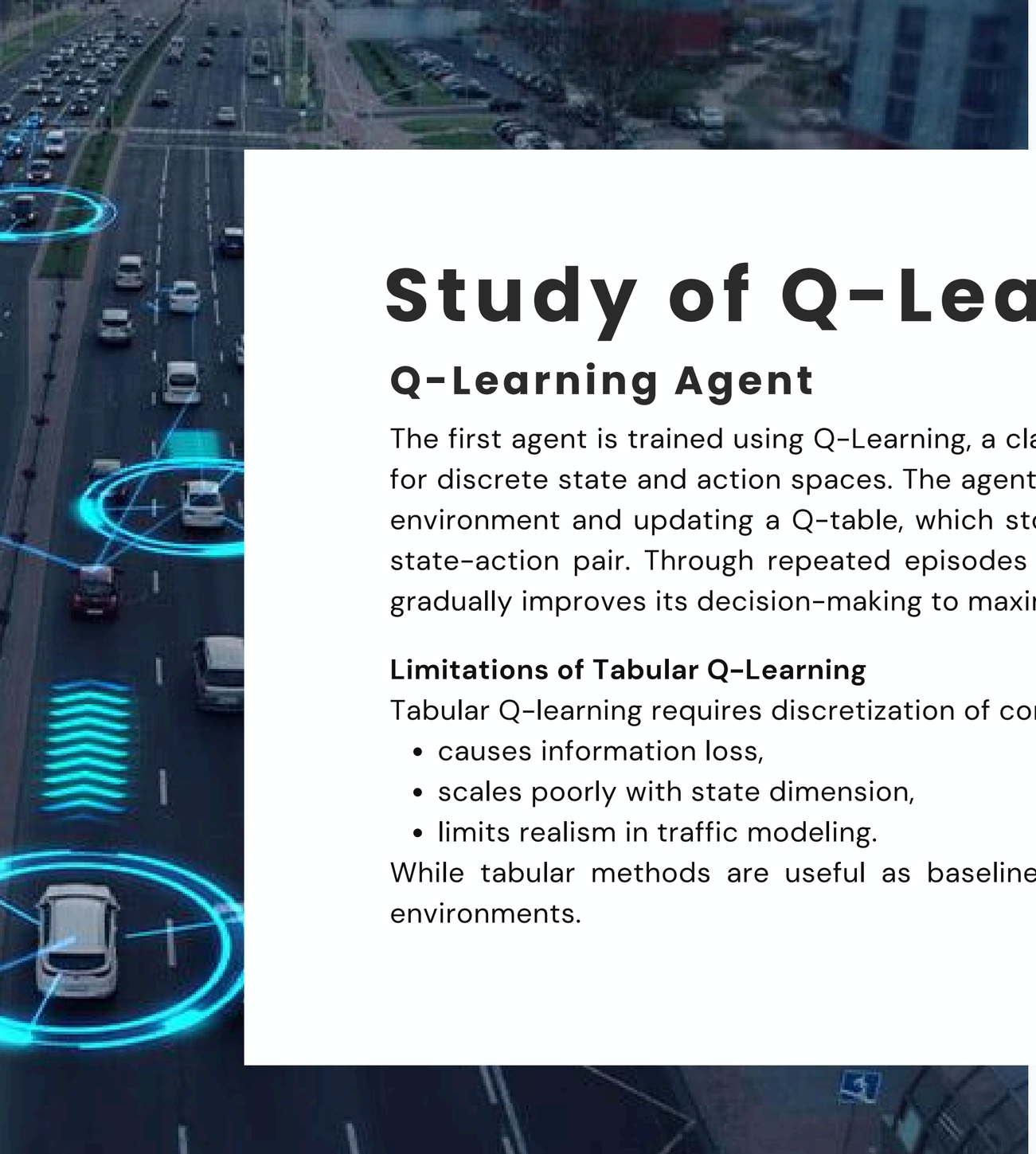
05 Brief description of code

06 Interpretation and discussion of different plots

1.Overview

- **Goal:** Train an autonomous vehicle (AV) to make safe and efficient lane-changing decisions on a two-lane highway using Reinforcement Learning.
- **Environment:** SUMO simulator with static vehicles as obstacles; AV speed is controlled by SUMO, while RL agents handle lateral maneuvers: left lane change, right lane change, or lane keeping.
- **Agents:**
  a. **Q-Learning:** Classical tabular approach, learns optimal lane-changing policies by updating a Q-table based on rewards.
  b. **Deep RL (PPO):** Uses a neural network to approximate the policy in continuous state space, enabling faster and more robust learning.
- **Objective:** Maximize AV travel efficiency while avoiding collisions with obstacles and road boundaries.
- **Evaluation:** Compare learning performance and lane-changing behavior of Q-Learning vs PPO agents, and benchmark against SUMO’s default lane-changing model.
- **Outcome:** Demonstrates the effectiveness of both classical and deep RL approaches for autonomous driving decisions in a constrained highway environment, highlighting trade-offs between simplicity and learning efficiency.

2.STUDY OF Q-LEARNING AND PPO



Study of Q-Learning

Q-Learning Agent

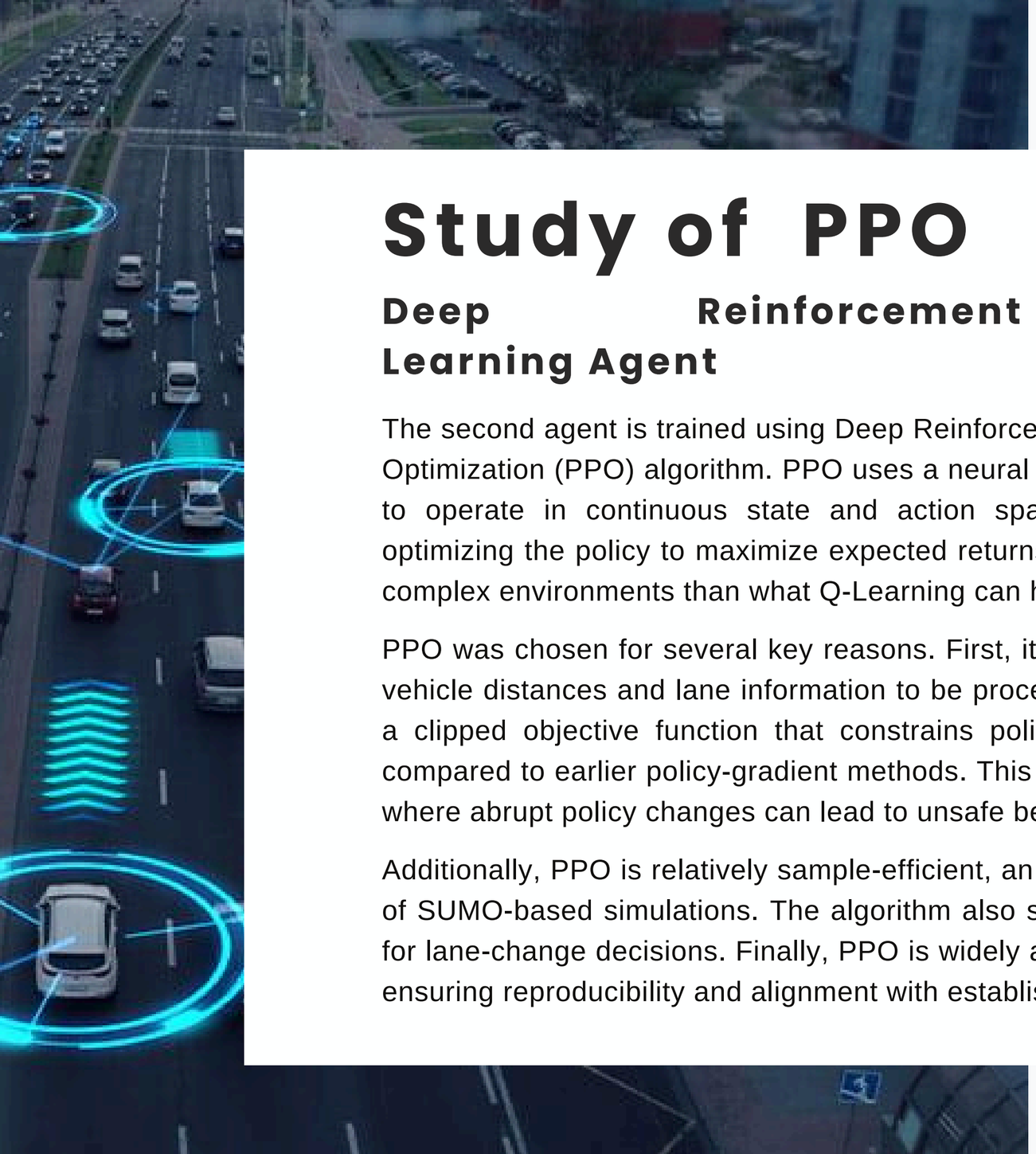
The first agent is trained using Q-Learning, a classical reinforcement learning algorithm suitable for discrete state and action spaces. The agent learns an optimal policy by interacting with the environment and updating a Q-table, which stores the expected cumulative rewards for each state-action pair. Through repeated episodes and exploration of the environment, the agent gradually improves its decision-making to maximize total rewards.

Limitations of Tabular Q-Learning

Tabular Q-learning requires discretization of continuous variables, which:

- causes information loss,
- scales poorly with state dimension,
- limits realism in traffic modeling.

While tabular methods are useful as baselines, they are insufficient for high-fidelity traffic environments.



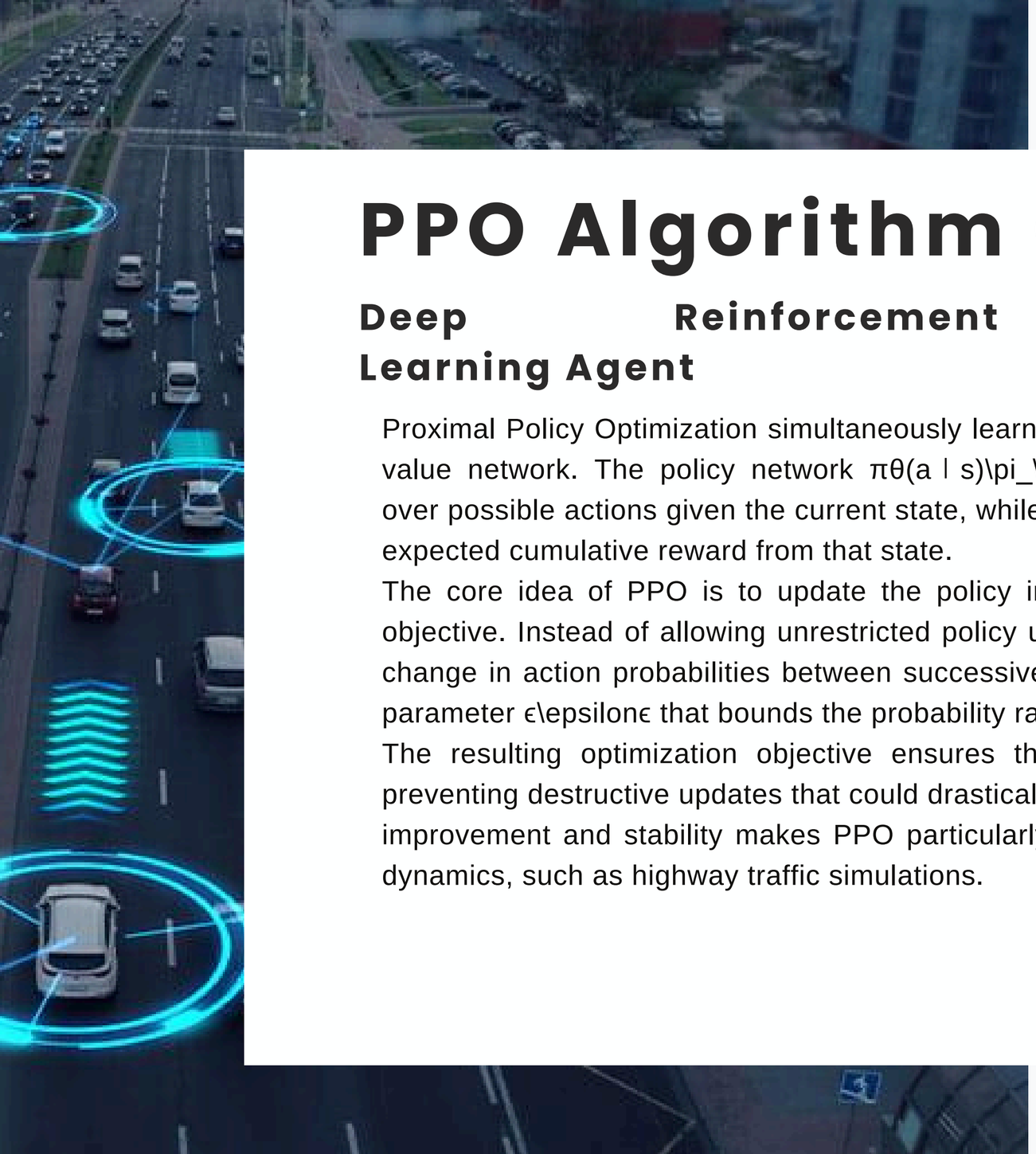
Study of PPO

Deep Reinforcement Learning Agent

The second agent is trained using Deep Reinforcement Learning (DRL), specifically the Proximal Policy Optimization (PPO) algorithm. PPO uses a neural network to approximate the policy, allowing the agent to operate in continuous state and action spaces. The agent learns by collecting experiences, optimizing the policy to maximize expected returns, and balancing exploration and exploitation in more complex environments than what Q-Learning can handle.

PPO was chosen for several key reasons. First, it naturally supports continuous state spaces, allowing vehicle distances and lane information to be processed as real-valued inputs. Second, PPO introduces a clipped objective function that constrains policy updates, significantly improving training stability compared to earlier policy-gradient methods. This stability is particularly important in traffic simulations, where abrupt policy changes can lead to unsafe behaviors or simulation crashes.

Additionally, PPO is relatively sample-efficient, an important consideration given the computational cost of SUMO-based simulations. The algorithm also supports discrete action spaces, making it well-suited for lane-change decisions. Finally, PPO is widely adopted in autonomous driving and robotics research, ensuring reproducibility and alignment with established scientific practices.



PPO Algorithm Overview

Deep Reinforcement Learning Agent

Proximal Policy Optimization simultaneously learns two function approximators: a policy network and a value network. The policy network πθ(a ∣ s)\pi_\theta(a|s)πθ(a ∣ s) outputs a probability distribution over possible actions given the current state, while the value network Vϕ(s)V_\phi(s)Vϕ(s) estimates the expected cumulative reward from that state.

The core idea of PPO is to update the policy in a conservative manner using a clipped surrogate objective. Instead of allowing unrestricted policy updates that may destabilize learning, PPO limits the change in action probabilities between successive updates. This is achieved by introducing a clipping parameter ϵ\epsilonϵ that bounds the probability ratio between the new and old policies.

The resulting optimization objective ensures that policy updates remain within a trusted region, preventing destructive updates that could drastically degrade performance. This balance between policy improvement and stability makes PPO particularly effective for environments with complex and noisy dynamics, such as highway traffic simulations.

3.DETAILED PROBLEM FORMULATION

Detailed RL problem formulation

1. Action Space (Discrete)

`0`: Keep Lane

`1`: Change Left

`2`: Change Right

- *Note:* Invalid lane changes (e.g., Left from Leftmost) will be ignored/penalized.

2. State Space (Observation)

`my_lane_id` (Discrete: 0, 1)

`dist_to_leader_current_lane` (Discretized: e.g., Close, Medium, Far)

`dist_to_leader_other_lane` (Discretized: e.g., Close, Medium, Far)

- *Simplification:* Speed is omitted. High distance implies high speed potential. Low distance implies low speed (braking).
- Discrete for Q-learning (e.g., lane 0/1, distance Close/Medium/Far)
- Continuous for DRL (meters, float values)

3. Reward Function We interpret "Maximize travel time" as **Maximize Speed / Efficiency** (i.e., minimize time to destination). "Maximize travel time" literally would mean driving slowly or stopping, which is likely not the goal.

`R_step` = `(current_speed / max_speed)` (Rewards moving fast. 0 if stopped behind obstacle).

`R_lane_change` = `-0.1` (Small cost to discourage weaving).

`R_collision` = `-10` (If virtually too close, though actual collision is prevented by SUMO).

4. Termination

- Reach end of highway.
- Max steps exceeded.(this is the one the teacher defined)

we initialize the MDP by defining

- State space (what variables describe the state)
- Action space (what the agent can do)
- Reward function (feedback to encourage correct behavior)
- Episode termination (when the environment ends)

4. HYPERPARAMETERS AND ANN ARCHITECTURE

Q-LEARNING

Q-LEARNING HYPERPARAMETERS

Learning Rate (alpha)
0.1

Discount Factor(gamma)
0.95

Initial Epsilon
1.0

Epsilon decay
0.998

Min Epsilon
0.05

Trainning episodes
500

PPO

PPO HYPERPARAMETERS

Learning Rate (alpha)
3e-4

gamma
0.99

rollout steps
2048

batch size
64

time steps
100-000

entropy coefficiennt
0.01

PPO
ANN ARCHITECTURE

PPO uses a shared **multilayer perceptron (MLP)** for feature extraction, with separate output heads for the policy and value function.

The selected ANN architecture balances expressiveness and computational efficiency. A relatively shallow network is sufficient due to the low-dimensional state space, while two hidden layers provide enough capacity to model nonlinear relationships between traffic variables. Sharing the feature extraction layers between the policy and value networks improves learning efficiency and reduces the total number of parameters.

Network Structure

Layer	Size	Activation
Input	3	
Hidden	64	ReLU
Hidden	64	ReLU
Policy output	3	Softmax
Value output	1	Linear

5. BRIEF DESCRIPTION OF CODE

5.1 REQUIRED PACKAGES

- **TRACI & SUMOLIB**: For SUMO interaction.
- **NUMPY**: For numerical computations and Q-table handling.
- **MATPLOTLIB**: For generating plots.
- **STABLE-BASELINE3**: For DRL algorithms (DQN, PPO).For visualizing training progress.
- .

5.2 PROJECT STRUCTURE

src	Main source code folder (all Python logic lives here).
__pycache__	Auto-generated Python cache files
data	SUMO simulation configuration files (road, vehicles, routes, GUI).
logs/ppo_lane_change/PPO_1	TensorBoard training logs for the PPO agent (used to visualize rewards, losses).
models	Saved trained RL models (DQN / PPO checkpoints).
results	Final evaluation outputs (plots, metrics, performance summaries).
deep_rl_train.py	Trains a Deep Reinforcement Learning agent (PPO) using Stable-Baselines.
demo.py	Runs PPO trained agent in SUMO to visually demonstrate lane-changing behavior.
env.py	Discrete SUMO environment used for classic Q-Learning.
env_continuous.py	Gym-compatible continuous environment used for Deep RL (DQN / PPO).
evaluation_result2.py	Evaluates trained agents and computes performance metrics (rewards, success rate).
q_learning.py	Implements tabular Q-Learning (baseline approach).
q_table_highway.npy	Saved Q-table learned from classic Q-Learning training.
rewards_history.npy	Stored reward values per episode for plotting learning curves.
test_env.py	Tests the discrete SUMO environment step-by-step (sanity check).
test_env_continuous.py	Tests the continuous Gym-style environment with random actions.
utils.py	Helper functions (plotting, logging, model saving/loading).
log	General logging folder (keeps repo structure clean even if empty).
requirements.txt	List of required Python packages to run the project.
run.py	Basic SUMO + TraCI script (manual simulation, no learning).
run2.py	TraCI control script showing lane changes and vehicle state logging.

5.2 PROJECT STRUCTURE

data	
obstacles.net.xml	Defines the road network (lanes, edges, junctions).
obstacles.rou.xml	Defines vehicles, routes, and departures.
obstacles.sumocfg	Main SUMO configuration file (links everything together)..
obstacles.view.xml	Controls visual appearance (colors, camera, zoom).
logs/ppo_lane_change/PPO_1	
events.out.tfevents	TensorBoard training logs (rewards, losses, entropy).
models	
ppo_lane_change.zip	Saved trained PPO model (neural network weights).
results	
training_plots_tensorboard_ppo/	Exported training curves from TensorBoard.
Evaluation_Ql (5eps).png	Q-Learning evaluation results over 5 episodes.
evaluation_qL.png	Visual summary of Q-Learning performance.
final1.png → final8.png	Snapshots / plots of agent behavior & performance for ppo.
evaluate_Qlearning.py	Script to evaluate Q-Learning agent.
evaluation_ppo.py	Script to evaluate PPO agent.
evaluation_results.csv	Numerical evaluation results (reward, ep length,etc.).
q_table_results.csv	Saved statistics from Q-Learning Q-table.
rewards_history.csv	Reward history exported to CSV.

5.3 HOW TO EXECUTE THE DIFFERENT FILE(S).

python file_name.py

What it does:
This command executes a Python script.

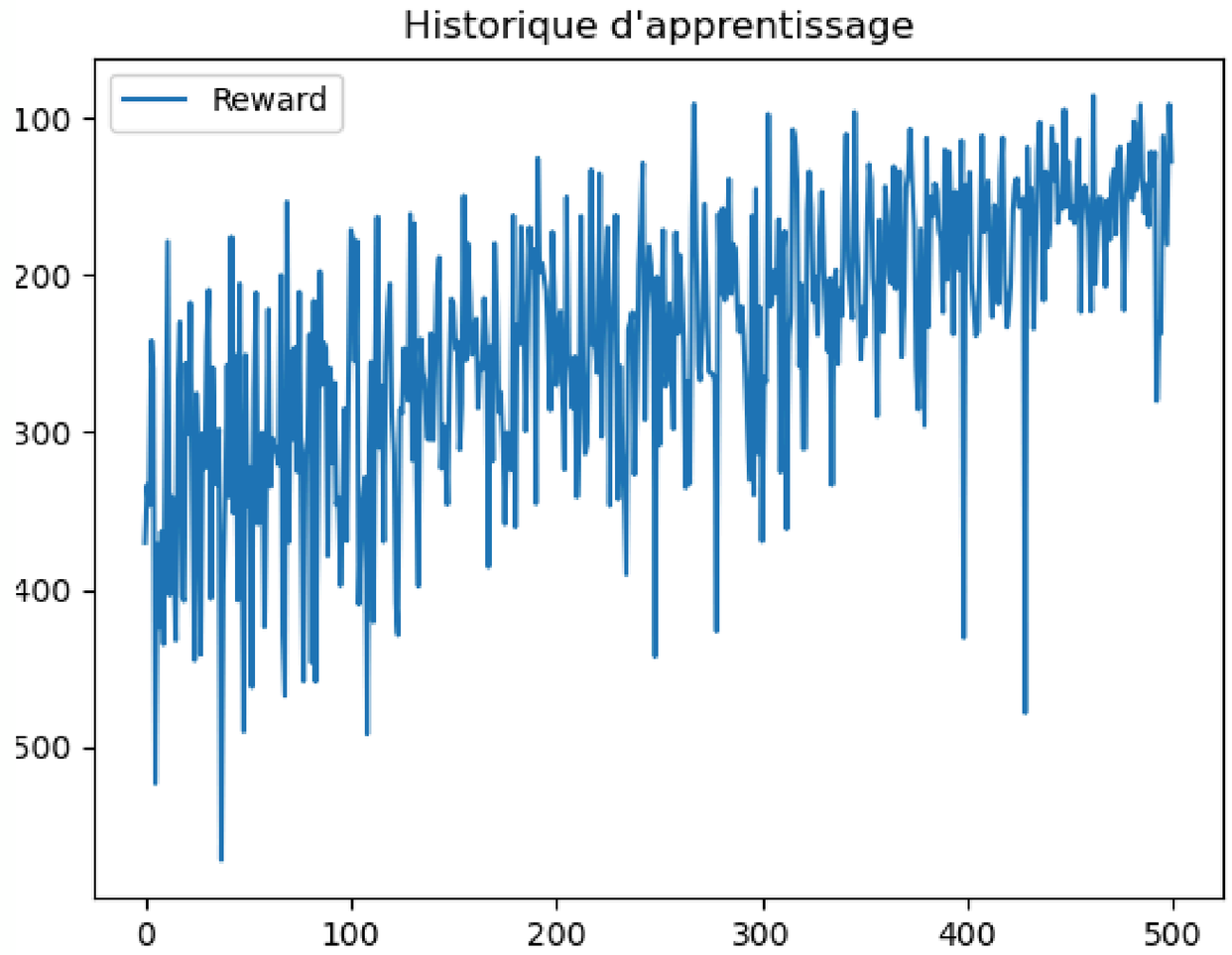
tensorboard --logdir logs/ --port 6006

What it does:
This command launches TensorBoard, a web-based tool used to:

- Monitor training progress
- Visualize reward curves
- Track loss, episode length, and performance metrics

6. INTERPRETATION AND DISCUSSION OF DIFFERENT PLOTS

6.1. Q-LEARNING PLOTS



REWARD CURVE ANALYSIS

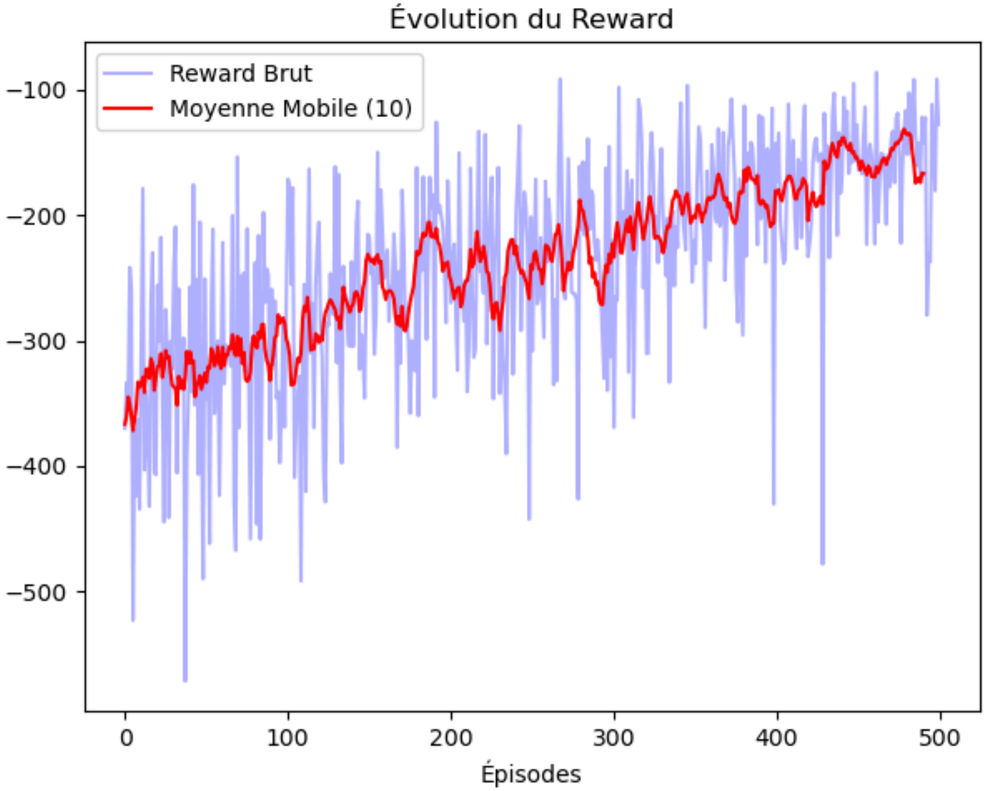
Interpretation :

The graph shows the total reward obtained by the agent at each episode.

- **Overall Trend (Red Line):** We can see a clear upward slope in the moving average. This shows that the agent is learning to optimize its behavior to maximize rewards and reduce penalties over time.

- **Reduction in failure frequency :** The downward “spikes” represent collisions or going off the road (large penalties). These spikes become less frequent and less severe after episode 300, which indicates safer driving behavior.

Convergence:
Towards the end (episodes 400–500), the curve starts to stabilize (plateau). This means the agent has found an “optimal” strategy and the learning process has converged.



EPSILON DECAY (E-GREEDY) ANALYSIS

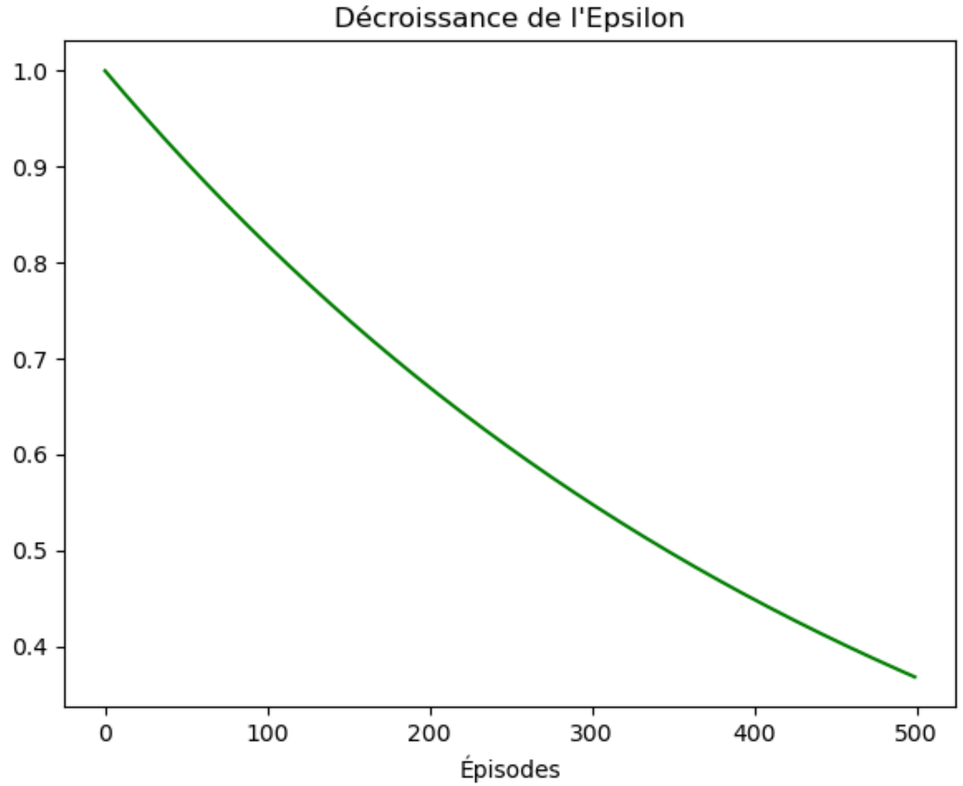
Interpretation :

The graph on the right shows how the agent moves from exploration to exploitation.

- **Exploration Phase (Episodes 0 to 150):** Epsilon is high. The agent makes random decisions to explore the environment (unnecessary lane changes, testing limits). This explains why the reward is very unstable at the beginning.

- **Exploitation Phase (Episodes 300 to 500):** Epsilon drops below 0.5. The agent starts to rely on what it has learned (the Q-table). It no longer makes “silly” mistakes and uses the maneuvers that worked before.

Final Outcome:
At episode 500, the agent follows its learned policy about 65% of the time, which explains the stable performance at the end.



PERFORMANCE EVALUATION

By analyzing both graphs, we observe that the agent's learning is successful:

- **Safety Achieved :**The agent understands that collisions are highly penalized and actively avoids them.

- **Efficiency Maintained :** Despite fixed obstacles, the agent keeps a stable speed and performs intelligent lane changes.

- **Robust Learning :** The average reward improves significantly, from -400 to -120, showing better decision-making over time.

DEEP DIVE INTO THE LEARNED POLICY (Q-TABLE)

Interpretation :

- **State-Space Mapping:** The terminal logs confirm a structured (18,3) Q-Table, covering 18 discrete highway scenarios (Lane position × Distance to obstacles).

Decision Weights

- **High-Risk States**: State 8 shows a maximum penalty of -70.29, identifyin it as a critical "near-collision" scenario where the agent has learned which actions lead to failure.

- **Action Preferences:** In State 5, the agent learned that "Lane Change Left" (-7.29) is significantly safer than "Staying" (-27.00) or "Right" (-16.60).

Policy Convergence

Non-zero values across most states demonstrate that the agent has successfully filled its "memory" with navigable strategies for the highway.

Etat	Rester	Gauche	Droite
0	0.00	0.00	0.00
1	0.00	0.00	0.00
2	-29.69	-13.20	-29.59
3	0.00	0.00	0.00
4	0.00	0.00	0.00
5	-27.00	-7.29	-16.60
6	-50.12	-9.71	-55.24
7	-10.85	-49.58	-21.43
8	-70.28	-70.30	-70.24
9	0.00	0.00	0.00
10	0.00	0.00	0.00
11	-36.71	-27.62	-17.84
12	0.00	0.00	0.00
13	0.00	0.00	0.00
14	-29.46	-21.93	-13.13
15	-9.11	-13.57	-54.90
16	-8.02	-12.73	-54.37
17	-10.58	-15.61	-46.25

SUMMARY OF AGENT SUCCESS (Q-LEARNING)

- **Goal Achievement :**The agent successfully learned to navigate a two-lane highway with static obstacles while minimizing travel time and avoiding collisions.

- **Efficiency :** Despite starting from total ignorance (randomness), the agent now proactively changes lanes to avoid blockage by the red obstacle vehicles.

- **Future Work :** The results validate the Q-Learning approach, providing a foundation for more complex scenarios involving moving traffic or Deep Q-Networks (DQN).

6.2. PPO PLOTS

Total Training Loss (train/loss)

total loss is a combination of:

- Policy loss (how good actions are)
- Value loss (how accurate the value function is)
- Entropy bonus (exploration)

Very high loss at the beginning (~80–90)
→ random policy, poor value estimates

Sharp drop in first ~20k steps
→ Agent quickly learns basic structure of the environment

Stabilization around ~5–12
→ Indicating convergence

Small oscillations later
→ Normal PPO behavior due to policy updates and exploration

Policy Gradient Loss (train/policy_gradient_loss)

Measures how strongly the policy is being updated

- **Early training:** large negative values (~−0.02 to −0.025)

→ Big policy corrections (from random to structured behavior)

- **Mid training:** oscillations

→ PPO explores but is constrained by clipping

- **Late training:** values approach zero

→ Policy updates become smaller , meaning The policy is no longer changing aggressively
whci Indicates policy convergence

Value Loss (train/value_loss)

Measures how well the value network predicts expected return,

- **Very high initial value loss** (>200)

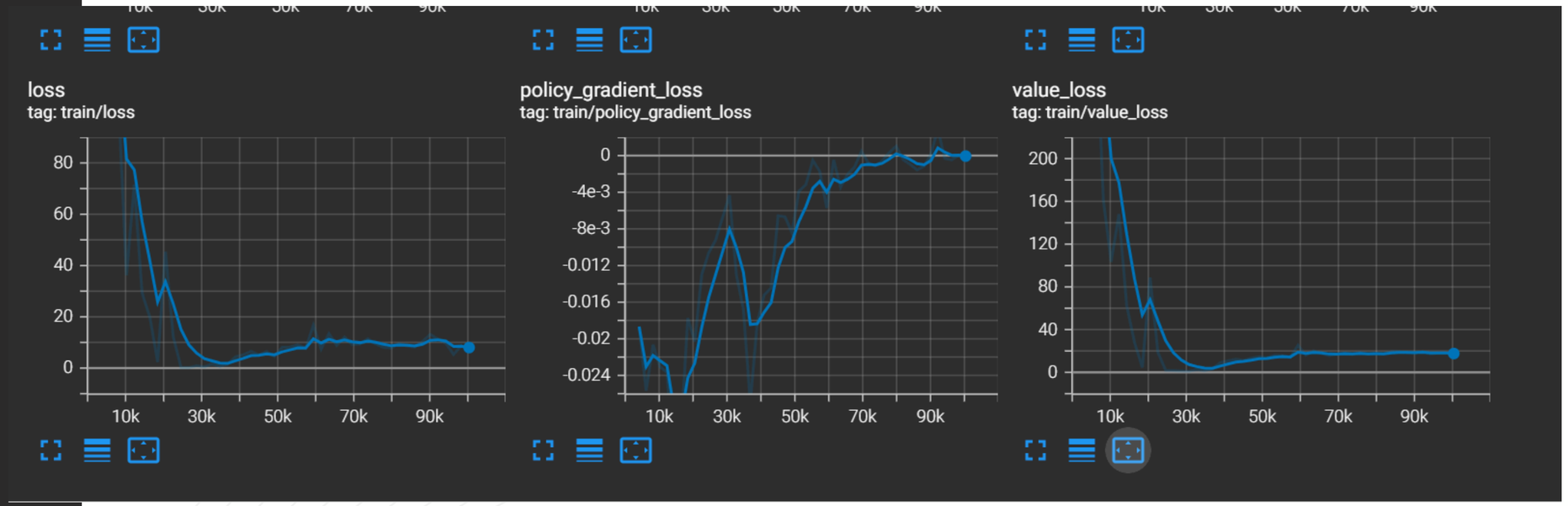
→ Value network has no idea what returns look like

- **Rapid drop** in first ~20k steps

→ Learns approximate reward structure

- **Stable region** around ~10–20

→ Normal for stochastic environments like traffic



EPISODE LENGTH – ROLLOUT/EP_LEN_MEAN

Interpretation :

he average number of steps per episode during training.

- **Early training:**

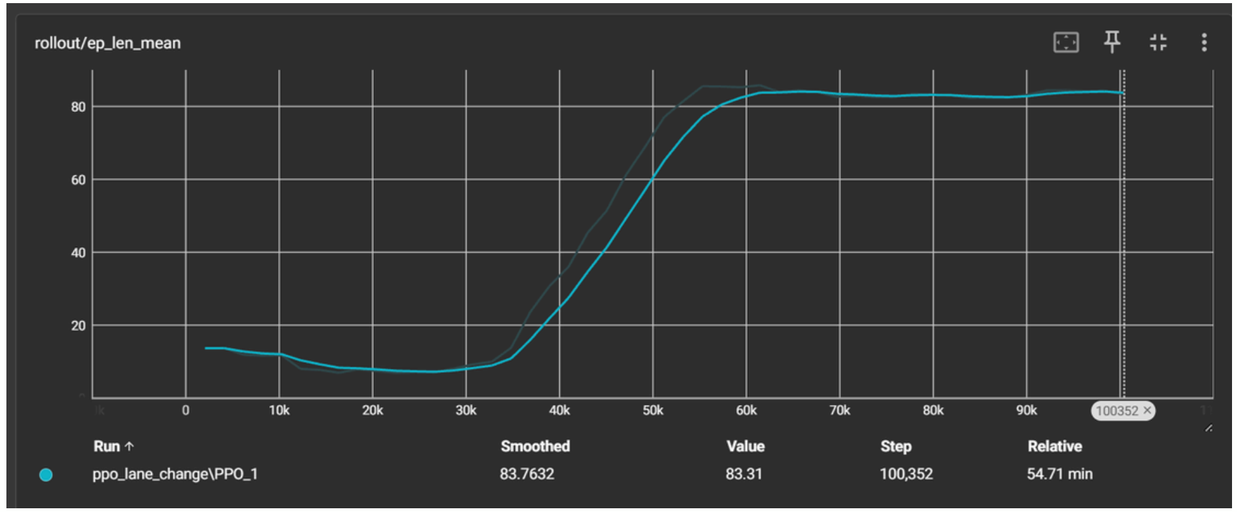
Episode length ≈ 10–15 steps
Agent frequently terminates early (unsafe or inefficient behavior)

- **Between 35k and 55k steps:**

Sharp increase in episode length

- **After ~60k steps:**

Episode length stabilizes around 83–85 steps



conclusion :

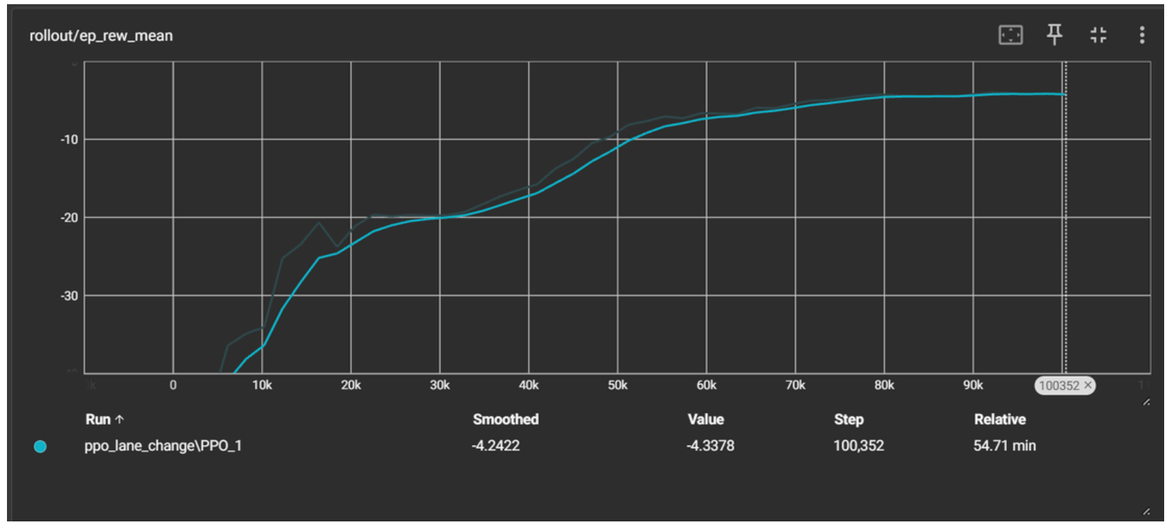
Your environment has a maximum episode length of ~82–85 steps, meaning:

- The agent now survives almost the entire episode
- Fewer crashes
- Safer and more consistent driving behavior

and Policy is stable after ~60k timesteps.

EPISODE REWARD – ROLLOUT/EP_REW_MEAN

Interpretation :



The mean total reward per episode (not discounted), which is exactly what the project requires.

- **Initial reward ≈ −35**

Frequent penalties
Unsafe lane changes
Low efficiency

- **Between 10k and 50k steps:**

Rapid improvement
Reward increases from −35 → −10

- **After ~60k steps:**

Reward stabilizes around −4.3
The reward remains negative but that's not a bad sign because Your reward function includes:

- Penalties for lane changes
- Safety penalties for close distances
- Conservative shaping

Conclusion :
Clear upward trend which means successful policy learning.Reward convergence indicates PPO stability.Final reward is consistent with a safe but conservative driving policy

TRAINING SPEED – TIME/FPS

Interpretation

This plot represents the training speed in frames per second (FPS), i.e., how many environment steps PPO processes per second.

- At the start of training, FPS is around 20–21.
- FPS drops to about 13 around 30k steps.
- After ~35k steps, FPS steadily increases, reaching ~29 FPS by the end of training.

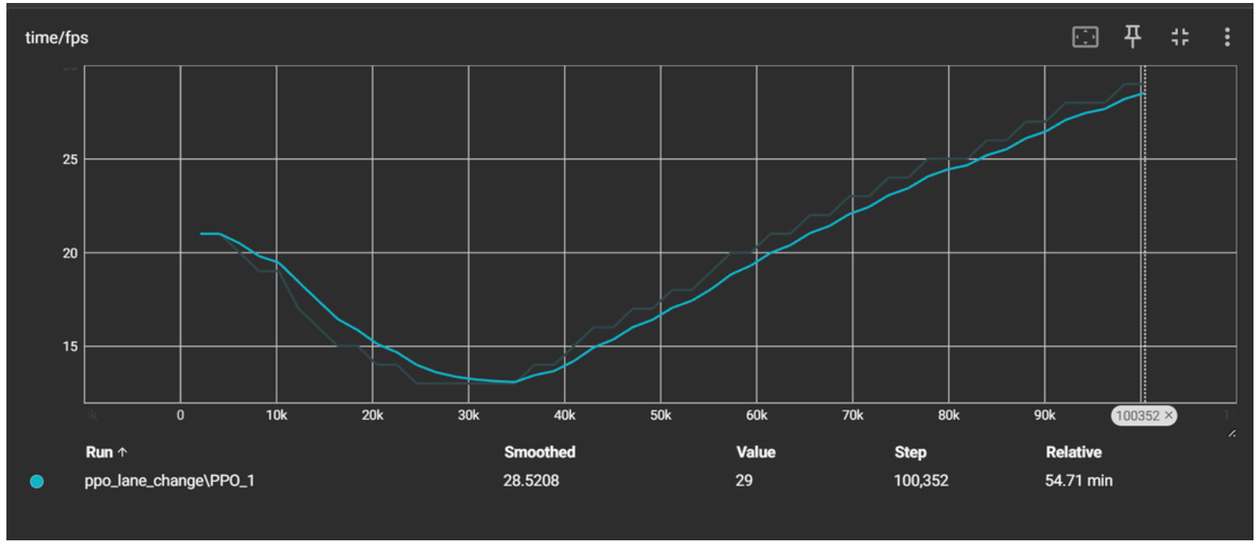
Conclusion:

- **Early training involves:**

More frequent resets
Short episodes
Less stable rollouts

- **As the policy improves:**

Episodes become longer and more consistent
SUMO interactions become smoother
CPU overhead is amortized over longer rollouts



PERFORMANCE EVALUATION

```
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 39 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 1ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 40 | Reward: -3.06 | Steps: 82
Step #83.00 (1ms ~= 1000.00*RT, ~20000.00UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 41 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 42 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 43 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 1ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 44 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 45 | Reward: -3.06 | Steps: 82
Step #83.00 (1ms ~= 1000.00*RT, ~20000.00UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 46 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 47 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 1ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 48 | Reward: -3.06 | Steps: 82
Step #83.00 (0ms ?*RT. ?UPS, TraCI: 0ms, vehicles TOT 21 ACT 20 BUF 0)
Episode 49 | Reward: -3.06 | Steps: 82
Evaluation saved to results/evaluation_results.csv
```

Evaluation module

The trained Proximal Policy Optimization (PPO) agent was evaluated using a deterministic evaluation script that loads the final trained policy and executes it over multiple independent episodes. During evaluation, the agent interacts with the environment without exploration noise, and its performance is assessed using two primary metrics: total reward per episode and episode length (number of steps). These metrics are recorded and saved in CSV format to enable post-training analysis and comparison.

Interpretation

The consistency observed in episode length and reward is a direct consequence of policy convergence. During training, the PPO agent learns a stable policy that maximizes expected reward under the given environment dynamics. Once convergence is achieved, the policy produces nearly identical action sequences when exposed to the same or similar states.

In the evaluation phase, the PPO model is executed in deterministic mode, meaning the action with the highest probability is always selected. As a result, the agent follows the same trajectory through the environment in each episode, leading to identical episode termination points and cumulative rewards.

Additionally, the environment configuration remains fixed across episodes:

- The same SUMO scenario file is used
- Initial vehicle positions and traffic conditions are not randomized
- No stochastic perturbations are applied to vehicle behavior


THANK YOU