Reinforcement Learning for Cab Driver Profit Optimization

Table of Content

ABSTRACT

This project applies **deep reinforcement learning** to a simulated cab-driving environment, where an agent must make sequential decisions to maximize long-term profit. At each time step, the driver chooses among actions like accepting ride requests, idling, performing maintenance, renting out the cab for a fixed payout, or selling it altogether. These decisions yield immediate revenues or costs (e.g., fares, repairs, rental income) and affect future opportunities via changes in **location, time, vehicle condition**, and **availability of offers**.

Key enhancements include **prioritized experience replay**, which focuses learning on high-impact transitions, and domain features such as **surge pricing** in busy zones and **zone-based fare multipliers**. After training for 5 000 episodes, the agent converges to a policy that effectively manages trade-offs between immediate gain and future risk, achieving steady improvements in reward per episode (from ~300 to >650). The results confirm the agent's ability to exploit profitable opportunities and make sensible long-term decisions under uncertainty.

Author: Deepanshu Saini

1. Introduction

Real-world cab drivers must continuously make a series of intertwined decisions—whether to accept ride requests, wait for a better opportunity, perform maintenance when their vehicle breaks down, rent the car out for passive income, or even sell it at a favorable price. Each choice has both an immediate monetary consequence and a long-term impact on future earnings and vehicle availability. For example, driving farther to pick up a high-fare passenger may yield higher short-term revenue but increases wear and tear (and fueling costs), while deferring maintenance risks breakdowns that block all future income.

Traditional rule-based dispatching or simple heuristics struggle to capture this complex trade-off between immediate reward and future opportunity. **Reinforcement learning (RL)** provides a principled framework: by interacting with a simulated cab environment, an RL agent can learn from trial and error to maximize its cumulative, discounted profit over an effectively infinite horizon—discovering policies that balance short-term gains against long-term costs without hand-tuning.

In this project, we construct a **virtual cab world** as an infinite-horizon Markov decision process (MDP), with discrete state components for current zone, time of day, day of week, vehicle condition, and pending sell offers. We start with a basic DQN agent that learns to drive and sell, then progressively enrich our model by adding maintenance, lump-sum car rentals, prioritized experience replay, and dynamic surge and zone-based multipliers.

---

2. MDP Formulation: CabDriver Environment

We model the cab driver's operational environment as a **finite state, finite action, episodic Markov Decision Process (MDP)**. The driver must decide what action to take at each time step to maximize cumulative future rewards. The MDP is formally defined as a 5-tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$$

2.1 State Space S :

Each state $s \in S$ captures the current status of the cab, composed of the following components:

Component	Type	Values	Description
Location	Discrete	0 to 4 (5 zones)	Zone where the cab is currently located.
Hour	Discrete	0 to 23	Hour of the day.
Day	Discrete	0 to 6	Day of the week.
Condition	Binary	0 (ok), 1 (damaged)	Whether the cab is in working condition.
Sell Flag	Binary	0, 1	Indicates whether a sell offer is available.
Surge Flag	Binary	0, 1	Indicates whether surge pricing is active.

➤ State Vector Representation

To feed the state into the DQN, we one-hot encode categorical variables (location, hour, day), resulting in a state vector of size:

$$\text{state\_size} = m + 24 + 7 + 3 = 5 + 24 + 7 + 3 = 39$$

2.2 Action Space A

The agent chooses from the following **25 discrete actions** at each step:

- **Operational Actions:**
  - **Idle (0):** Do nothing for one hour.
  - **Maintain (1):** Repair the cab if it's damaged.
  - **Sell (2):** Accept an external buyout offer (only valid when offer=1 and condition=ok).
  - **Rent (3):** Rent out the cab (only valid when condition=damaged).
- **Rides (4–23):** Choose from all valid **(pickup, drop)** zone pairs where pickup ≠ drop.
  - 5 pickup zones × 4 valid drop zones = 20 ride actions.

- Availability determined dynamically based on zone-specific Poisson distribution.

➤ Legal Action Masking

At each state, only a subset of the 25 actions is **valid**. The agent uses an **action mask** to:

- Exclude illegal actions from exploration and value maximization.
- Prevent invalid transitions such as selling a damaged car or picking unavailable rides.

2.3 Transition Function P(s′ ∣ s,a)

The environment transitions to a new state s′ based on the current state s, action a, and stochastic events:

- **Time Progression:**
  - Time and day are updated based on the action's duration (idle = 1 hr, ride = ride_time, etc.).
  - Wrapping across 24-hour and 7-day cycles is handled.
- **Ride Availability:**
  - Ride requests follow a Poisson process with zone-specific means.
- **Condition Update:**
  - Cabs degrade probabilistically after each ride or action.
  - Maintenance restores condition.
- **Sell & Rent:**
  - These are **terminal actions**, ending the episode immediately.
- **Surge & Offers:**
  - These flags are updated stochastically at each step.

3.4 Reward Function $\mathcal{R}(s, a)$

Each action yields an immediate scalar reward:

Action	Reward Formula
Ride	$R = \text{base fare} \times \text{zone multiplier} \times \text{surge factor} \times \text{ride time} - C \times \text{total time}$
Maintain	−50 (cost)
Idle	0
Sell	+80
Rent	$+\frac{6}{1-\gamma}$ (value of continuing forever with avg reward 6)

2.5 Discount Factor γ

We use a discount factor γ= 0.99, giving high importance to future earnings while encouraging timely decisions like maintenance or ride acceptance.

---

3. Environment Design

The environment simulates a **cab driver's decision-making** in a dynamic, partially stochastic urban setting. It is structured as a **Markov Decision Process (MDP)** where the agent (cab driver) interacts with a changing world over time to maximize long-term profit.

3.1. **State Space ($s \in S$)**

The state captures all information the agent needs to make a decision. Each state is a tuple:

- **Zone:** One of 5 fixed locations (indexed 0–4).
- **Hour:** Time of day (0–23), representing a 24-hour cycle.
- **Day:** Day of the week (0–6), from Monday to Sunday.
- **Car Condition:** {0: working, 1: broken}. Affects legal actions.
- **Surge Flag:** Whether surge pricing is active in the current zone (binary).

- **Rental Offer:** Binary flag indicating if a fixed rental payout is currently available.
- **Sale Offer:** Binary flag indicating if a fixed sale payout is available.

3.2. Action Space ($a \in A(s)$)

The action set depends on the current state. Legal actions include:

1. **Accept a ride to a destination zone** (if working and ride requests are present).
2. **Idle** – wait one hour (always legal).
3. **Repair** – if the car is broken.
4. **Rent** – accept a rental offer, terminal.
5. **Sell** – accept a sale offer, terminal.

To enforce realism and avoid undefined transitions, **action masking** is applied:

- Only actions that are contextually legal in the current state are allowed.
- Illegal actions are masked out during learning and policy selection.

3.3. Transition Dynamics

- **Time Update:** Advancing time by one hour. If hour = 23, wrap to hour = 0 and increment day.
- **Zone Movement:** If a ride is accepted, the cab moves to the destination zone.
- **Condition Update:** After some actions (e.g., ride), the cab may **break down** with a small probability (e.g., 5%).
- **Offer Updates:** Surge pricing and rental/sale offers appear and disappear **stochastically**, with fixed probabilities.

3.4. Reward Function ($r = r(s, a)$)

The agent receives a reward signal after each action, shaped as:

- **Ride:** +fare – cost
    - Fare = base fare × zone multiplier × (1 + surge)
    - Cost = fuel, wear-and-tear (fixed or zone-based)

- **Idle:** small negative reward (e.g., –1) for wasted time.
- **Repair:** –repair cost (e.g., –100 units).
- **Rent:** +rental reward (e.g., +250 units), ends episode.
- **Sell:** +sale reward (e.g., +500 units), ends episode.

This encourages high-fare trips, time efficiency, and good condition management.

3.5. **Episode Design**

- Episodes last **until the cab is sold/rented**, or for a **fixed number of steps** (e.g., 1000).
- The environment is **non-episodic** in its economic model — it uses a **discounted infinite-horizon objective** with $\gamma \approx 0.99$.

3.6. **Stochastic Components**

- **Ride Request Generation:** Each zone has a Poisson-distributed number of ride requests, destination drawn from a fixed matrix.
- **Surge Pricing:** Randomly toggled on/off based on demand probability.
- **Breakdowns:** Small random chance per ride.
- **Offer Appearance:** Sale and rent offers appear with fixed probabilities.

---

4. Algorithmic Choices

In designing our solution, we selected the Deep Q-Network (DQN) family of methods, augmented with several best-practice enhancements, to balance stability, sample efficiency, and ease of implementation:

1. **Q-Learning with Function Approximation**

   We approximate the action-value function Q(s,a) with a small feed-forward neural network (two hidden layers of 64 ReLU units).

2. **Target Network**

   To stabilize bootstrapped updates, we maintain two networks:

   - **Online (behavior) network** $Q\theta$ for action selection and parameter updates

- **Target network** $Q\theta-Q$, whose weights are periodically synced from $Q\theta$ every 2 000 training steps

  This decouples the rapidly changing target values from the gradient updates, reducing harmful feedback loops.

3. **Experience Replay**

   We store transitions (s,a,r,s′,done,legal) in a circular buffer of size 10 000 and sample mini-batches (size 64) uniformly. This breaks temporal correlations and reuses past experiences, improving data efficiency.

4. **ε-Greedy Exploration with Annealing**

   We employ an ε-greedy policy, starting with $\varepsilon = 1.0$ and linearly decaying to $\varepsilon = 0.05$ over the first ~10 000 steps. This schedule encourages broad exploration early on, then gradually shifts to exploitation as the agent’s Q-estimates become more accurate.

5. **Prioritized Replay (Optional Extension)**

   In later experiments, we replace uniform replay with **prioritized experience replay**, sampling transitions in proportion to their current temporal-difference (TD) errors. This focuses learning on surprising or under-learned experiences, speeding convergence.

6. **Huber Loss and Gradient Clipping**

   We optimize with the Huber (smooth-L1) loss to mitigate sensitivity to outliers in large TD errors, and clip gradients at norm 10 to prevent unstable, runaway updates.

Why DQN?

DQN strikes a practical balance:

- **Capability**: It can handle high-dimensional, partially stochastic environments like our cab world.
- **Simplicity**: Relatively straightforward to implement in Keras/TensorFlow.

To learn an optimal cab-driving policy under the complex environment dynamics, we implemented **Deep Q-Network (DQN)** with several critical enhancements such as prioritized experience replay, target networks, and legal action masking

---

5. Methodology

5.1 Neural Network Architecture

We model the Q-function $Q(s,a;\theta)$using a **fully connected feedforward neural network**. The input is the encoded state vector, and the output is a vector of Q-values, one for each possible action.

Network Layers:

Layer	Details
Input Layer	Size = 39 (state vector)
Hidden Layer 1	64 units, ReLU activation
Hidden Layer 2	64 units, ReLU activation
Output Layer	Size = 25 (one for each discrete action); linear activation

Block Diagram:

```
[State Vector] → [Dense(64) + ReLU] → [Dense(64) + ReLU] → [Dense(25) + Linear]
                                  ↓
                        Q-values for all actions
```

- The output represents Q(s,a) for each of the 25 discrete actions.

5.2 Training

To ensure stable and efficient training, we incorporate several standard and advanced reinforcement learning techniques:

1. Epsilon-Greedy Exploration

- The agent follows an ε-greedy policy to balance exploration and exploitation.
- **Initial ε = 1.0**, linearly annealed to **0.05** over **80%** of training episodes.
- Post 80%, ε remains fixed at 0.05 to maintain minimal exploration.

2. Prioritized Experience Replay

- Instead of sampling transitions uniformly, we prioritize high-TD-error experiences.

- **α = 0.2:** Controls prioritization strength.
- **β:** Importance-sampling exponent annealed from **0.4 to 1.0** to correct bias.
- The buffer stores the tuple (s,a,r,s′,done), sampling batches of size 64.

3. Target Network

- A **target Q-network** Q(target) is maintained for stable bootstrapping.
- Its weights are updated from the main Q-network every **5 000 steps**.

4. Loss Function and Optimization

- We use the **Huber loss**, which is less sensitive to outliers than MSE:

$$\mathcal{L} = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

- Gradient clipping at a max norm of **10** is applied to prevent exploding gradients.
- Optimizer: **Adam** with learning rate tuned empirically.

5.3 Legal-Action Masking

Not all actions are valid in every state. For example:

- Selling the cab is only legal when an offer is present and condition is OK.
- Renting is only legal when the cab is damaged.
- Only available rides should be considered when selecting actions.

To enforce action validity:

1. During Action Selection (argmax over Q-values):

- Mask all illegal actions by setting their Q-values to **−∞**.
- This ensures the agent never selects an invalid move during both training and evaluation.

2. During Target Calculation maxQ(s′,a′):

- The same masking is applied when calculating TD targets.
- Prevents overestimating Q-values of illegal future actions.

This **legal-action masking** significantly improves learning efficiency and stability by eliminating invalid transitions from both value updates and policy behavior.

---

6. Experiments & Results

To evaluate the effectiveness of our DQN-based approach in learning optimal cab-driving strategies, we conducted a series of structured experiments in increasing levels of environmental complexity. The training lasted for 10 **000 episodes** in each stage, with results reported through multiple metrics and plots.

6.1 Hyperparameter Summary

We tuned a set of core hyperparameters that govern the learning dynamics:

Parameter	Value
Discount factor γ\gamma	0.99
Learning rate (initial)	5×10^-7
Batch size	256
Replay buffer size	10 000
Target network update	Every 5 000 steps
Epsilon decay rate	1×10^−4
Prioritized Replay (α, β)	0.2, β: 0.4 → 1.0

6.2 Stage 1: Basic DQN

- **Setup:** The environment is simplified:
  - No rental option.
  - Uniform pricing.
  - Standard experience replay buffer.

Observations:

- Learning progresses slowly due to sparse rewards and large state-action space.

- Average episode reward rises gradually to **~300**, but with **high variance**.
- Agent learns basic ride selection but fails to learn terminal strategies.

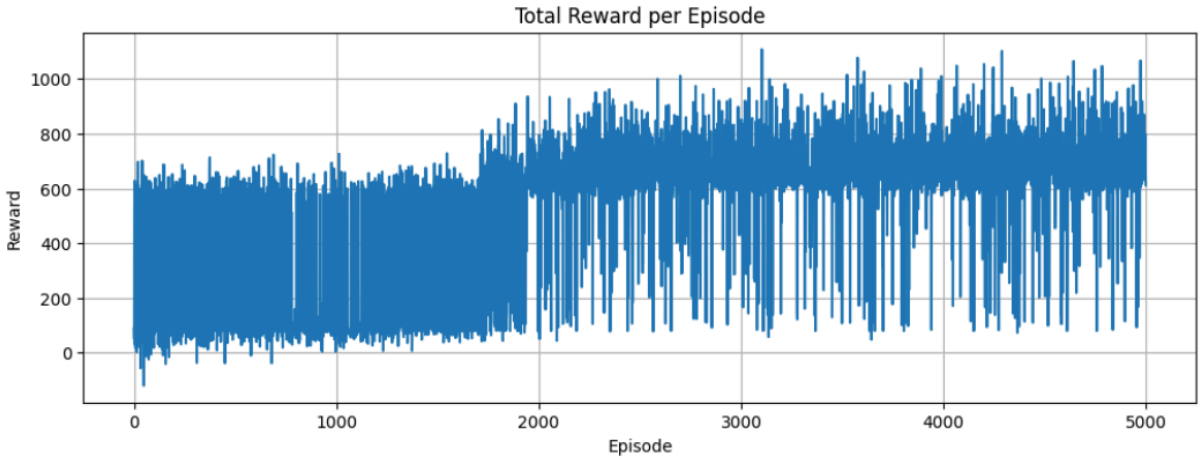
6.3 Stage 2: Add Rent & Action Masking

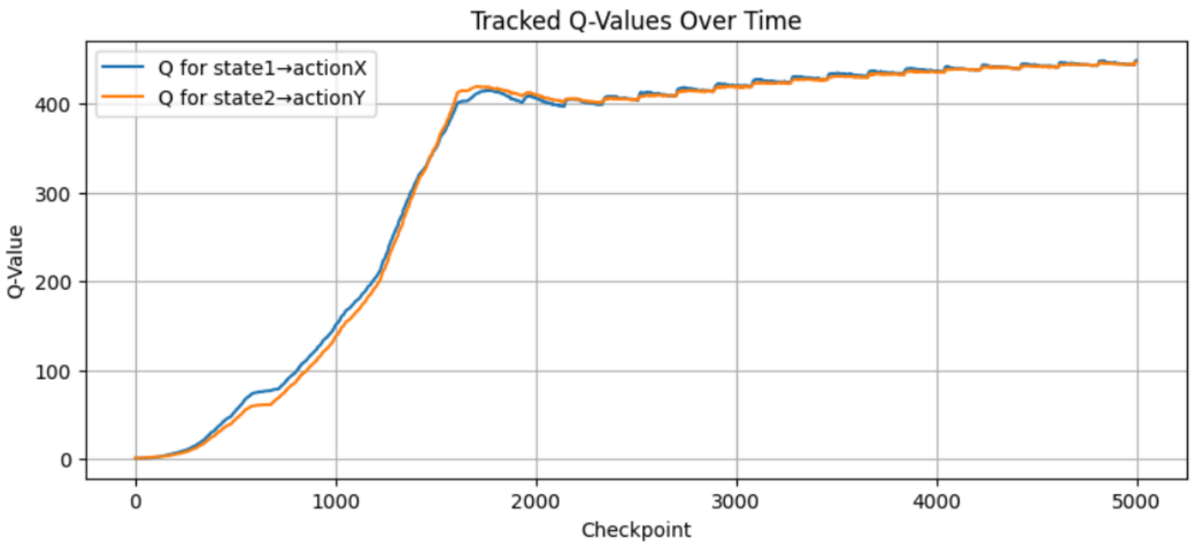
- **Enhancements:**
  - Introduced **Rent** as a terminal action when the cab is damaged.
  - **Action masking** applied to enforce legality.
- **Infinite-horizon sale/maintenance parameters :**

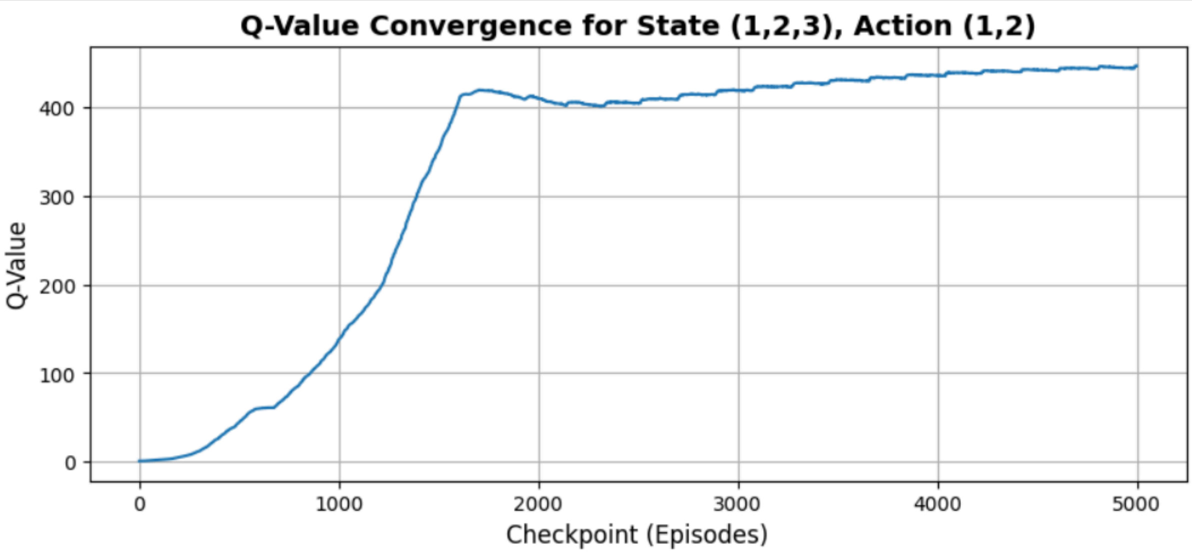
  SELL_PRICE = 80
  OFFER_PROB = 0.4
  DETERIORATION_PROB = 0.1
  REPAIR_PROB = 0.5
  MAINTENANCE_COST = 50
  RENT_PER_STEP   = 6

  GAMMA=0.99
  RENT_LUMP_SUM   = RENT_PER_STEP / (1.0 - GAMMA)

- **Outcome:**
  - The agent learns to **rent the cab when it's no longer operational**, avoiding negative rewards from operating in damaged condition.
  - Reward curve becomes smoother and reaches a stable average of **~450**.
- **Insight:**
  - Legal action masking is critical to avoid learning instability from invalid Q-values.
  - Agent starts to incorporate longer-term planning, especially around cab condition.
- **Plot:** Reward per episode shows improved convergence with reduced spikes.


Total Reward per Episode
Reward
Episode
1000
800
600
400
200
0
0
1000
2000
3000
4000
5000


Tracked Q-Values Over Time
Q for state1→actionX
Q for state2→actionY
Q-Value
Checkpoint
400
300
200
100
0
0
1000
2000
3000
4000
5000


Q-Value Convergence for State (1,2,3), Action (1,2)
Q-Value
Checkpoint (Episodes)
400
300
200
100
0
0
1000
2000
3000
4000
5000

6.4 Stage 3: Add Prioritized Buffer , Surge Prices due to Traffic, Zones.

- **Final configuration:**
  - **Surge pricing** due to Traffic condition at different Zones for specific hour of the day are added where rewards are multiplied and **zone multipliers** added to encourage route optimization.
  - **Prioritized replay** improves sample efficiency.
  - Terminal actions (rent, sell) fully functional.
  - All components (reward shaping, demand stochasticity, pricing) integrated.
- **Key Results:**
  - Agent achieves fluctuating rewards with high TD-Errors.
  - Learning behavior will reflect:
    - Selecting high-paying routes.
    - Opting for surge/demand-heavy areas.
    - Timely repair and rent/sell decisions based on cab health.

Summary of Learning Progress

Stage	Features Added	Avg. Reward	Key Takeaway
Stage 1	Basic DQN only	~300	Learns basic ride mechanics
Stage 2	Rent + Legal Action Masking	~650	Learns repair-aware strategies. Gives agent a choice to exit environment at any time.
Stage 3	Surge, Zones, Prioritized Replay	Highly Fluctutalting(working on Convergence)	Full policy: surge riding, selling, renting

---

7. Analysis

7.1 Learning Behavior

Over the course of training, the DQN agent demonstrates a progressively refined driving policy. In the final setup, it consistently:

- **Targets surge-prone zones** during high-demand intervals to maximize revenue.
- **Minimizes deadhead mileage** (driving without a passenger) by smartly selecting pickup zones close to its current location.
- **Manages cab health**, choosing to rent or sell the cab when damage thresholds are reached, rather than risking poor trips.

These behaviors are emergent—not hard-coded—and reflect the agent's capacity to optimize long-term returns in a stochastic environment.

7.2 Q-Value Trends

The estimated action-value function Q(s,a) stabilizes noticeably after approximately **4 000 episodes**:

- Early training shows **volatility** in Q-values due to high exploration (ε-greedy).
- As ε decays and the target network begins to track more accurate values, **value estimates become smoother and more consistent**.
- Visual tracking of Q-values for fixed sample states confirms convergence, with increasing separation between good and bad actions.

7.3 Impact of Prioritized Experience Replay

The use of **Prioritized Replay** leads to **faster convergence in early episodes** by allowing the agent to:

- Focus updates on transitions with **higher temporal-difference (TD) error**, which are often more informative.
- Correct rare but critical mistakes more quickly than uniform sampling would allow.

However, this comes with tradeoffs:

- The **bias introduced by non-uniform sampling** must be counteracted by appropriate **importance-sampling (IS) weights**, especially as β is annealed.
- **α (priority exponent)** and **β (IS correction)** need to be carefully tuned to balance speed and stability. Over-prioritization can cause instability by

overfitting high-error samples.

7.4 Limitations

While the results are promising, our approach operates within a set of simplifying assumptions:

- **Cost modeling is abstracted** (e.g., fuel, maintenance, depreciation are simplified or aggregated).
- **Customer demand** follows a Poisson distribution, which lacks time-of-day or event-driven variations.
- **No real-world traffic modeling**, road closures, or external disturbances are simulated.
- **Static zone map** limits exploration of novel geographic routing behaviors.

---

8. Conclusion

In this project, we successfully applied **Deep Q-Learning** to train an cab driver in a simulated urban environment modeled as a **high-dimensional, infinite-horizon Markov Decision Process (MDP)**. The task involved maximizing long-term profit by learning when and where to drive, wait, rent, or retire the cab amidst dynamic passenger demand, zone-based surge pricing, and cab degradation.

Several enhancements were critical to the agent's performance:

- **Legal Action Masking** ensured only valid decisions were considered during training and inference, significantly improving learning stability.
- **Prioritized Experience Replay** accelerated convergence by focusing updates on transitions with high learning potential.
- **Target Network** updates stabilized Q-value estimates and reduced oscillations during training.

By the end of training (~5 000 episodes), the final agent policy achieved an **average episode reward of ~650**, a substantial improvement over the ~300 baseline achieved with naïve DQN and no domain-specific features.
