



Projet de Deep Learning

APPRENTISSAGE PROFOND PAR RENFORCEMENT (DRL) 1

Présenté le Lundi 23 Juin 2025

Encadreurs :

Mme DIOP & Mr SECK

DIC2 GIT 2025

MEMBRES DU GROUPE



**Cheikh Ahmadou
Bamba SYLL**



Mouhamed SARR

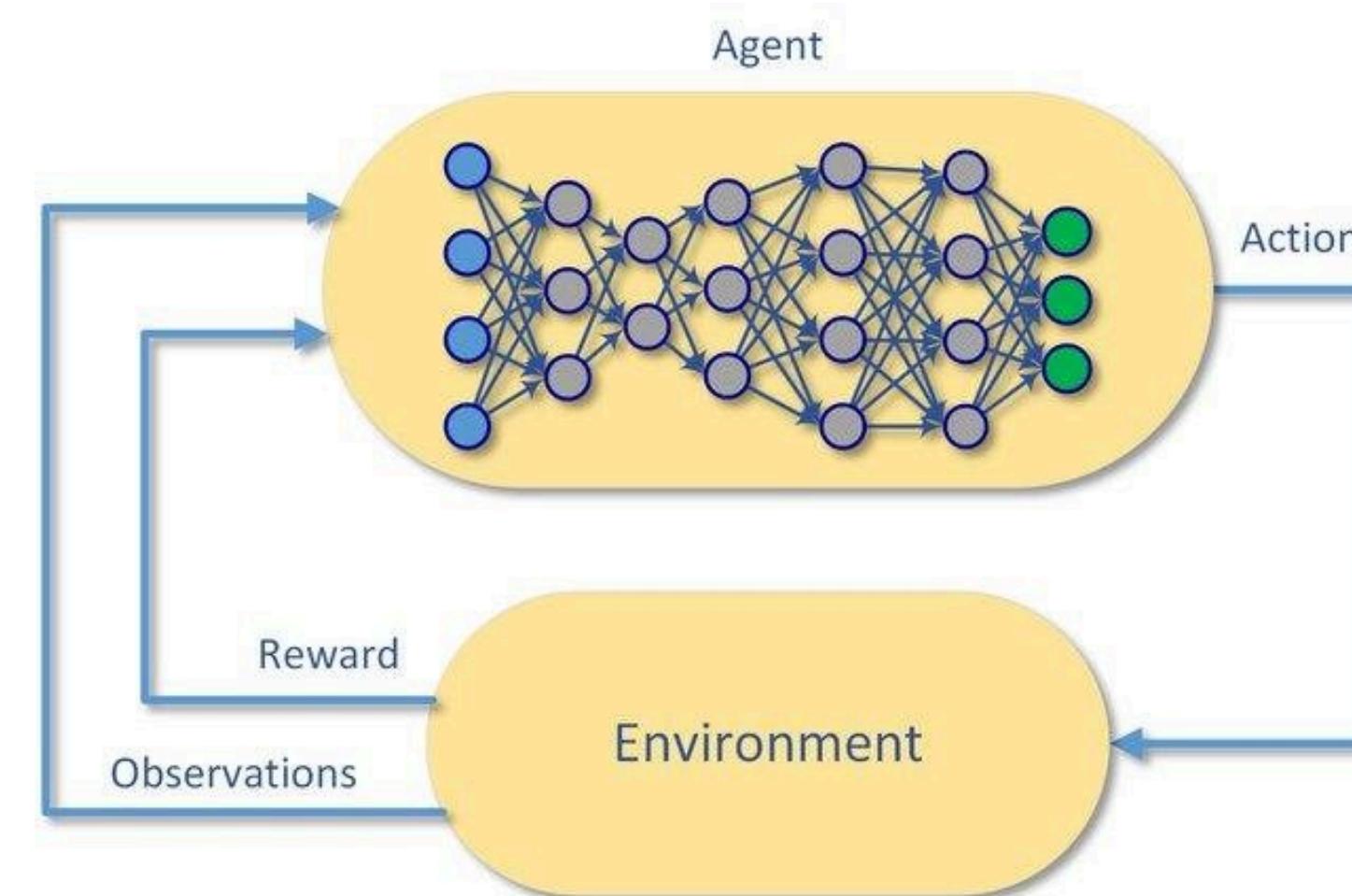
PLAN

- 1.** Introduction
- 2.** Fondamentaux du Deep Reinforcement learning
- 3.** Application pratique
- 4.** Méthodologie
- 5.** Analyse de performances
- 6.** Démonstration
- 7.** Avantages et limites
- 8.** Perspectives et conclusion

INTRODUCTION

L'apprentissage par renforcement (Reinforcement Learning) est une branche de l'IA où un agent apprend à prendre des décisions en interagissant avec un environnement. L'agent reçoit des récompenses en fonction des actions qu'il effectue, ce qui lui permet d'adapter sa stratégie (ou politique) au fil du temps.

Le Deep Reinforcement Learning combine la puissance des réseaux de neurones profonds avec le cadre du Reinforcement Learning traditionnel afin de résoudre des tâches complexes.



FONDAMENTAUX DU DRL



CADRE MDP (MARKOV DECISION PROCESS)

Un MDP est défini par le quintuplet $\langle S, A, P, R, \gamma \rangle$ où :

- S : ensemble fini ou infini d'états possibles de l'environnement
- A : ensemble fini ou infini d'actions possibles que l'agent peut choisir
- $P : S \times A \times S \rightarrow [^1]$: fonction de transition probabiliste, où
$$P(s'|s, a) = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$
représente la probabilité que l'état suivant soit s' sachant que l'agent était en état s et a pris l'action a à l'instant t
- $R : S \times A \rightarrow \mathbb{R}$: fonction de récompense, où
$$R(s, a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$$
est la récompense attendue après avoir pris l'action a en état s
- $\gamma \in [^1]$: facteur d'actualisation (discount factor) qui pondère l'importance des récompenses futures

Propriété de Markov

La dynamique du système satisfait la propriété de Markov : la probabilité de transition vers un état futur dépend uniquement de l'état courant et de l'action choisie, pas des états ou actions passés :

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1}|s_t, a_t)$$

FONDAMENTAUX DU DRL



CADRE MDP (MARKOV DECISION PROCESS)

Équations de Bellman

La valeur d'un état peut s'exprimer par :

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) V^\pi(s') \right]$$

Pour la valeur de Q, nous avons :

$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) \sum_{a'} \pi(a' \mid s') Q^\pi(s', a')$$

FONDAMENTAUX DU DRL



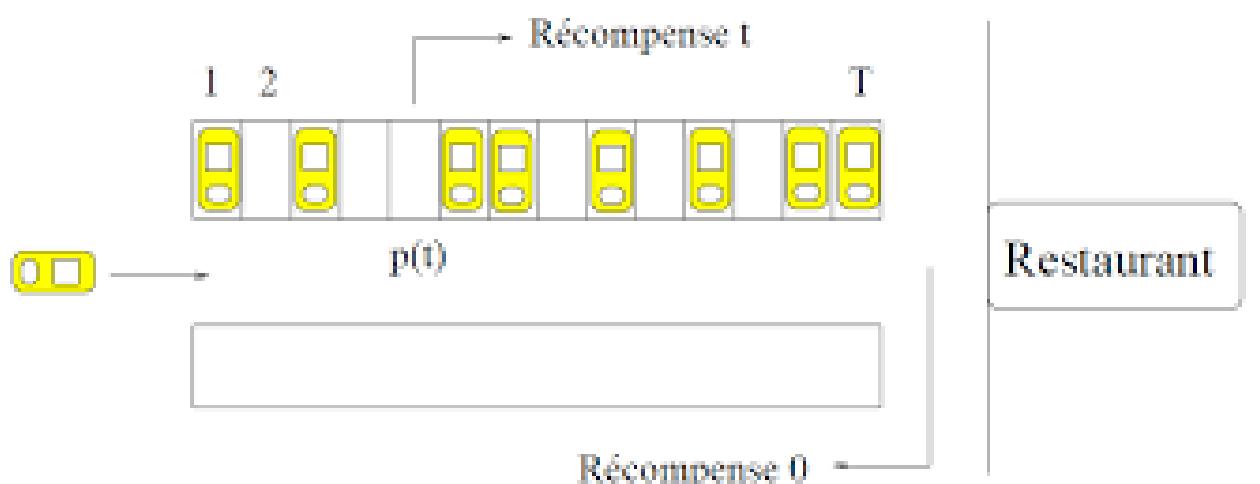
NOTION DE POLITIQUE

Politique (π) – Stratégie de décision

- La politique π définit comment l'agent choisit ses actions en fonction de l'état actuel.
 - Déterministe: une action fixe par état $\rightarrow \pi(s) = a$
 - Stochastique: distribution de probabilité $\rightarrow \pi(a|s) = P(\text{choisir } a \text{ depuis } s)$
 - Objectif: maximiser la récompense cumulative espérée

$$\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

- Notion clé d'**exploration** vs **exploitation**:
 - explorer pour apprendre,
 - exploiter pour maximiser les gains.



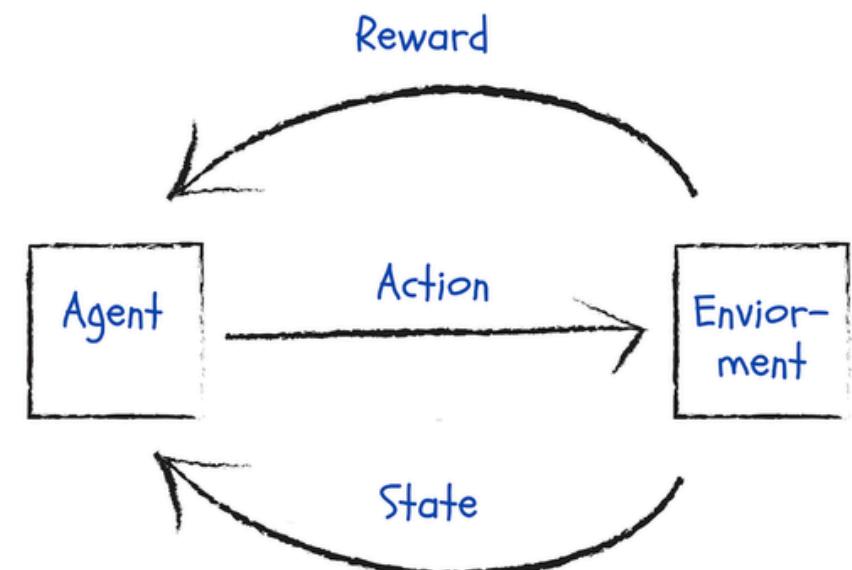
FONDAMENTAUX DU DRL



AGENT RL

Agent RL – Interaction et apprentissage

- L'agent agit dans un environnement en boucle continue :
 1. Observe l'état courant s_t
 2. Choisit une action a_t selon la politique π
 3. Exécute l'action, reçoit une récompense r_{t+1} et observe le nouvel état s_{t+1}
- Objectif : apprendre une politique π^* qui maximise la somme des récompenses futures
- L'apprentissage se base sur les retours d'expérience (états, actions, récompenses) pour améliorer la prise de décision
- Fonctionne selon un cycle d'observation, action, récompense, mise à jour



FONDAMENTAUX DU DRL



ALGORITHMES TABULAIRES MONTE CARLO (MC)



Monte Carlo (MC) – Apprentissage par épisodes complets

- Méthode basée sur l'évaluation des états ou actions à partir de la récompense cumulée observée en fin d'épisode
- Pour chaque visite d'un état s (ou d'une paire (s, a)) pendant un épisode, on calcule le **retour** :

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}$$

avec T la fin de l'épisode

- Mise à jour de la valeur estimée par une moyenne des retours observés :

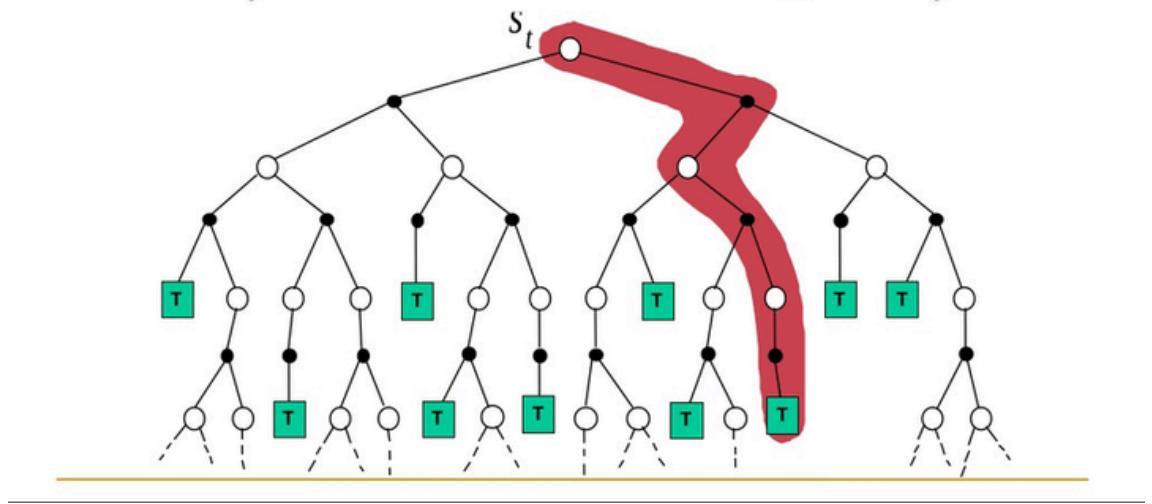
$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

- Avantages :

- Apprentissage simple, fondé sur des retours complets.
- Pas besoin de connaître la dynamique du système.

- Limites :

- Nécessite des épisodes complets et finis.
- Lent pour épisodes longs ou non-terminaux.



FONDAMENTAUX DU DRL

ALGORITHMES TABULAIRES

TD-LEARNING

TD-Learning – Apprentissage par mises à jour incrémentales

- TD-Learning met à jour la valeur d'un état immédiatement après chaque transition, sans attendre la fin de l'épisode.

- Mise à jour de la valeur $V(s_t)$:

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

où

- α est le taux d'apprentissage,
- r_{t+1} est la récompense reçue,
- γ est le facteur de discount.

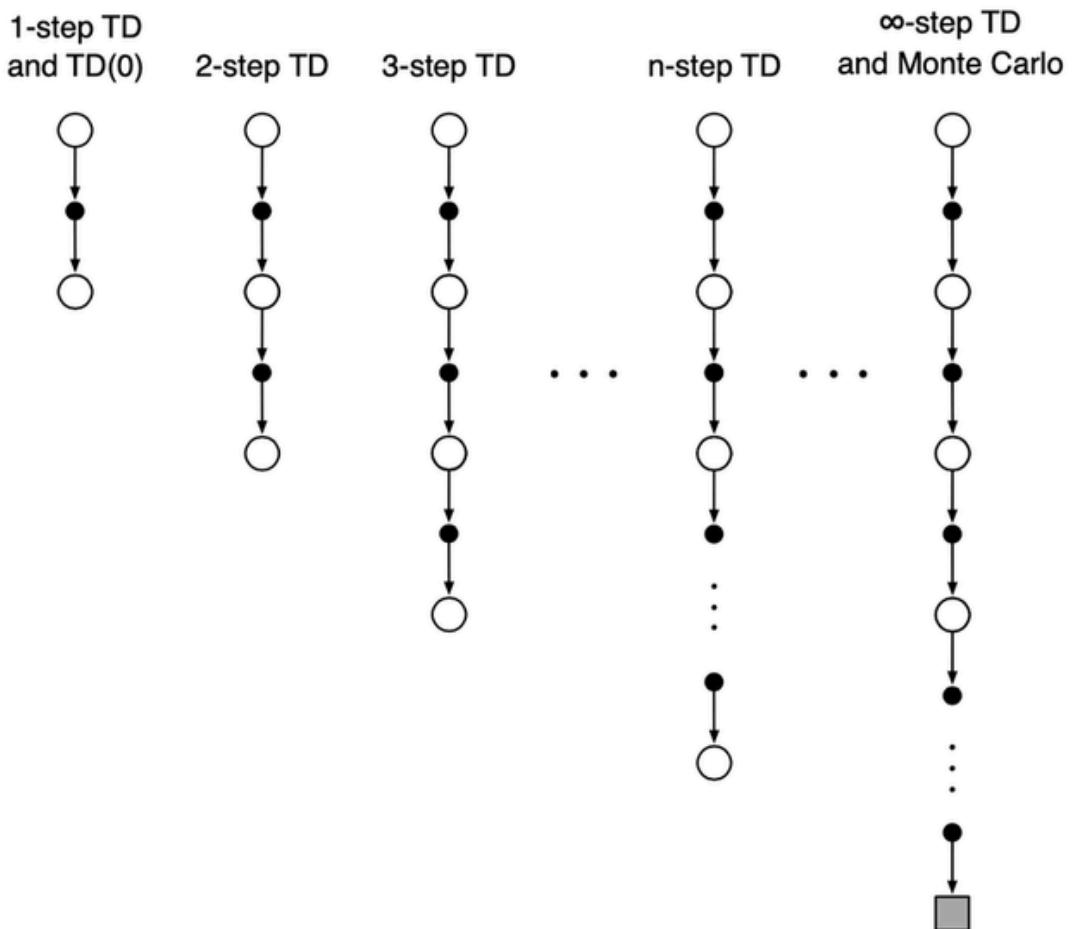
- Cette méthode combine l'idée de Monte Carlo (apprentissage par récompenses) et le bootstrapping (utilisation des estimations actuelles).

- Avantages :

- Apprentissage en ligne, pas besoin d'attendre la fin d'épisode.
- Efficace dans les environnements non-épisodiques ou continus.

- Limites :

- Nécessite une bonne exploration pour converger vers la valeur réelle.



FONDAMENTAUX DU DRL



ALGORITHMES TABULAIRES



SARSA

SARSA est un algorithme d'apprentissage par renforcement **on-policy** qui met à jour la fonction de valeur d'action $Q(s, a)$ en utilisant la séquence d'expériences $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ réellement suivie par la politique courante. Contrairement à Q-learning (off-policy), SARSA apprend la valeur de la politique effectivement exécutée, ce qui le rend plus prudent dans des environnements où l'exploration peut être risquée.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- State (S): The situation the agent is in at a given time.
- Action (A): The decision the agent makes based on that state.
- Reward (R): The feedback the agent gets after performing the action.
- Next State (S'): Where the agent ends up after the action.
- Next Action (A'): The next move the agent decides to make from the new state.

FONDAMENTAUX DU DRL

ALGORITHMES TABULAIRES

Q-LEARNING

Q-Learning : Apprentissage de la valeur optimale

Q-learning est un algorithme d'apprentissage par renforcement off-policy.

Il permet à un agent d'apprendre la **politique optimale** indépendamment de la stratégie utilisée pendant l'entraînement.

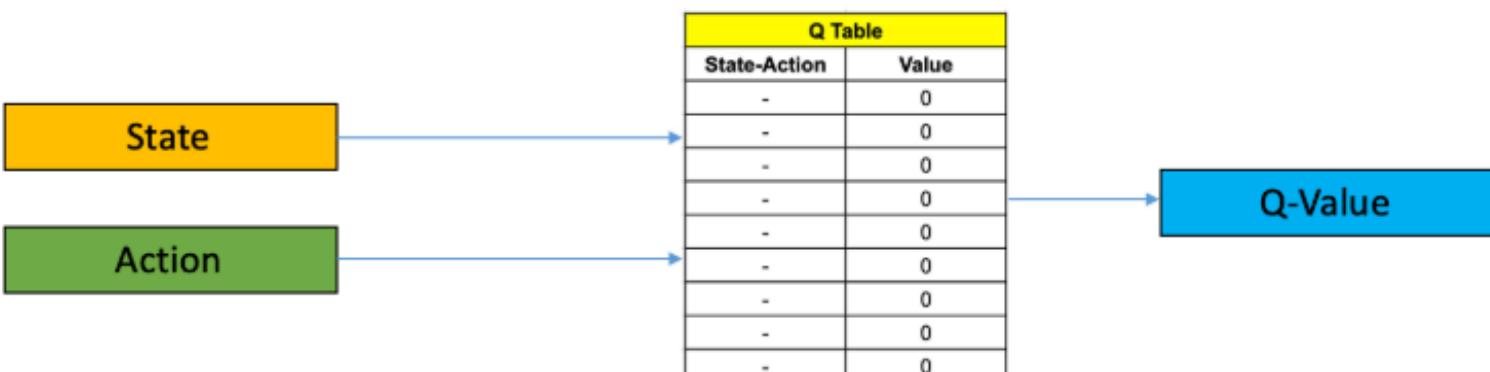
Cadre

L'agent apprend une fonction $Q(s, a)$ qui estime la **valeur maximale attendue** de la récompense à long terme après avoir effectué l'action a dans l'état s .

Mise à jour

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

- α : taux d'apprentissage
- γ : facteur d'actualisation
- r_{t+1} : récompense observée
- $\max_{a'} Q(s_{t+1}, a')$: meilleure valeur estimée pour l'état suivant



Q Learning

Objectif

Trouver la politique optimale :

$$\pi^*(s) = \arg \max_a Q(s, a)$$

FONDAMENTAUX DU DRL

Q-LEARNING PROFOND (DQN)

Q-Learning profond (DQN) – Approximation des valeurs par réseau de neurones

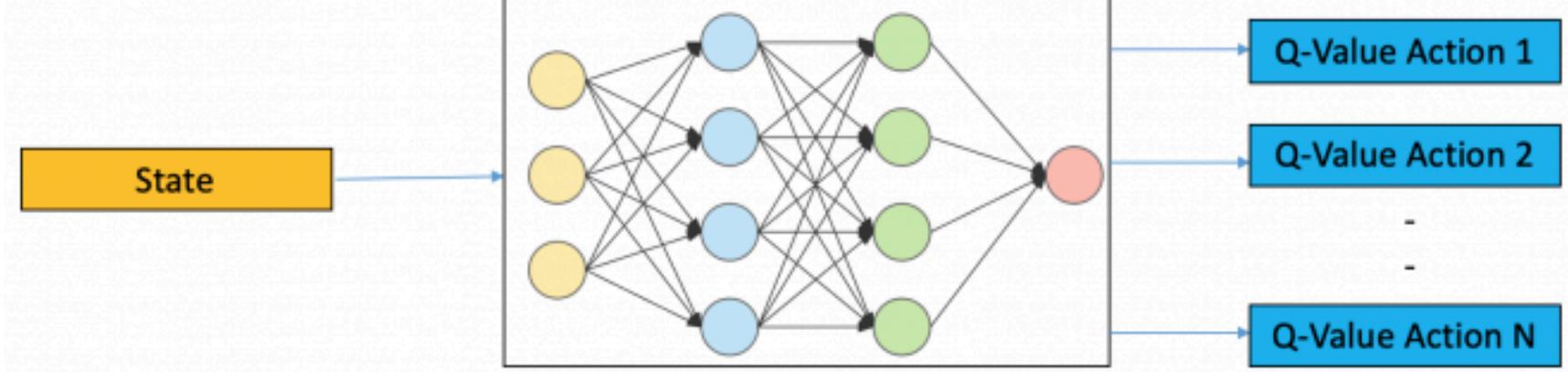
- DQN est une extension de Q-learning utilisant un réseau de neurones pour approximer la fonction $Q(s, a)$.
- Le réseau apprend à prédire :

$$Q(s, a; \theta) \approx \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

- La mise à jour s'effectue en minimisant l'erreur quadratique entre la valeur actuelle et la cible :

$$L(\theta) = \mathbb{E}_{(s, a, r, s')} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$$

- θ : paramètres du réseau actuel
- θ^- : paramètres du **réseau cible** (fixé temporairement)
- Techniques clés :
 - **Experience replay** : mémorisation et rééchantillonnage aléatoire des transitions
 - **Target network** : stabilisation de l'apprentissage via un second réseau gelé
- Avantage : capable de traiter de grands espaces d'états (images, capteurs...)



Deep Q Learning

FONDAMENTAUX DU DRL



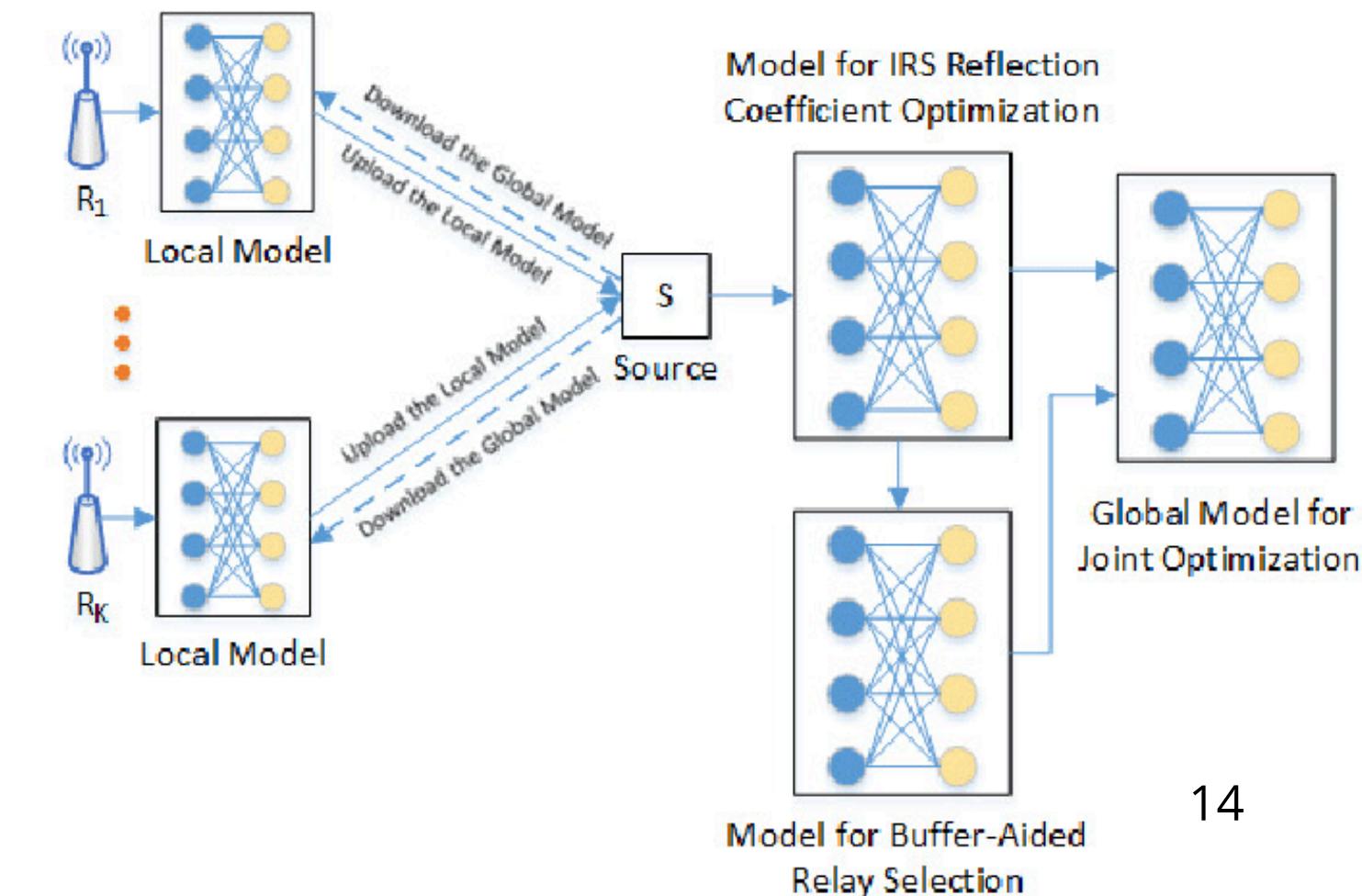
DRL MULTI-AGENTS...

DRL Multi-agents – Apprentissage dans des environnements partagés

- Plusieurs agents interagissent dans un **même environnement**, avec objectifs parfois communs (coopération) ou opposés (compétition).
- L'environnement devient **non-stationnaire** : la politique d'un agent évolue pendant l'apprentissage des autres.
- Chaque agent i apprend une politique $\pi_i(a_i|s_i)$, en maximisant sa récompense attendue :

$$J_i(\theta_i) = \mathbb{E}_{\pi_1, \dots, \pi_N} \left[\sum_{t=0}^{\infty} \gamma^t r_i^t \right]$$

- Deux grandes approches :
 - Centralisées** (ex. : MADDPG) : apprentissage avec accès aux observations et actions globales
 - Décentralisées** (ex. : IQL, QMIX) : chaque agent apprend à partir de ses propres observations
- Défis majeurs :
 - Coordination, partage d'informations
 - Stabilité de l'apprentissage
 - Scalabilité au nombre d'agents

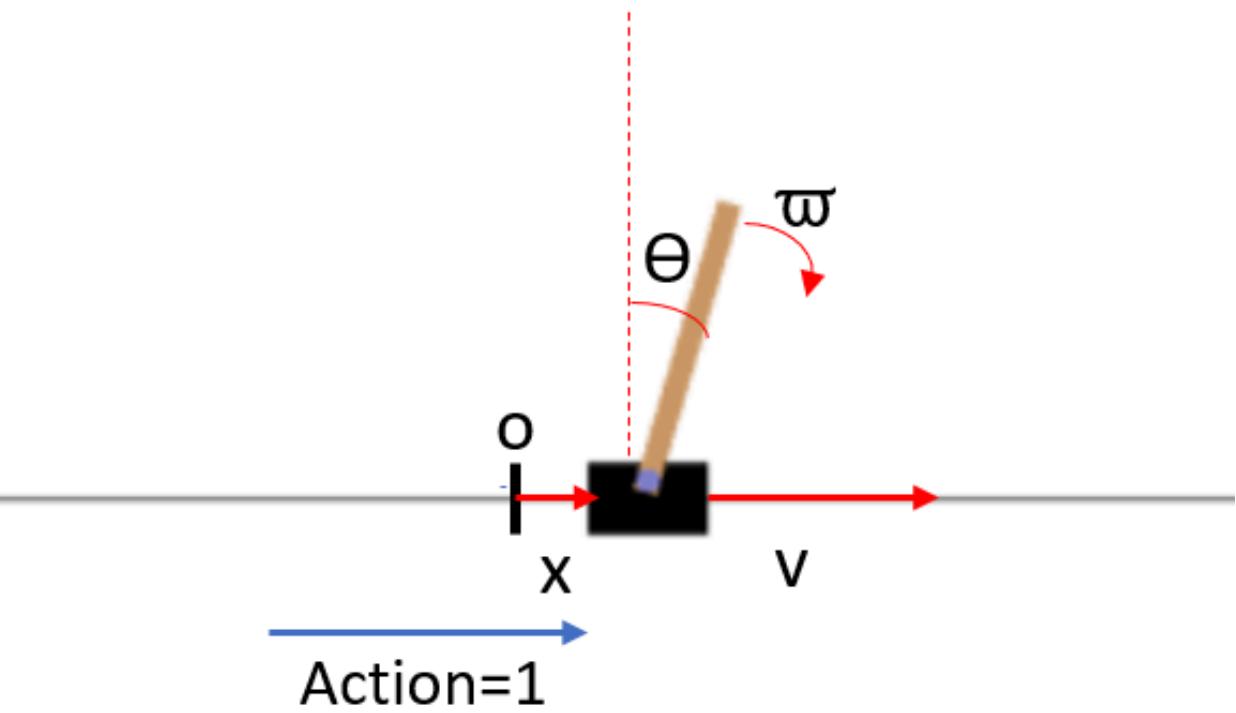


APPLICATION PRATIQUE

CARTPOLE

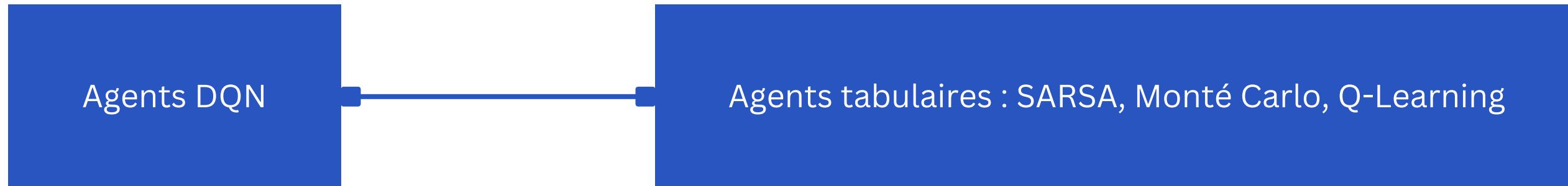
Dans CartPole, une tige (le "pôle") est fixée à un chariot (le "cart") qui peut se déplacer horizontalement sur une piste. L'objectif est de maintenir la tige en équilibre vertical aussi longtemps que possible en déplaçant le chariot vers la gauche ou la droite.

- État : 4 valeurs (position & vitesse du chariot, angle & vitesse du pôle)
- Actions : Gauche ou droite (discrètes).
- Récompense : +1 à chaque étape si la tige reste debout.
- Fin : Si la tige tombe ou si le chariot sort des limites.



METHODOLOGIE

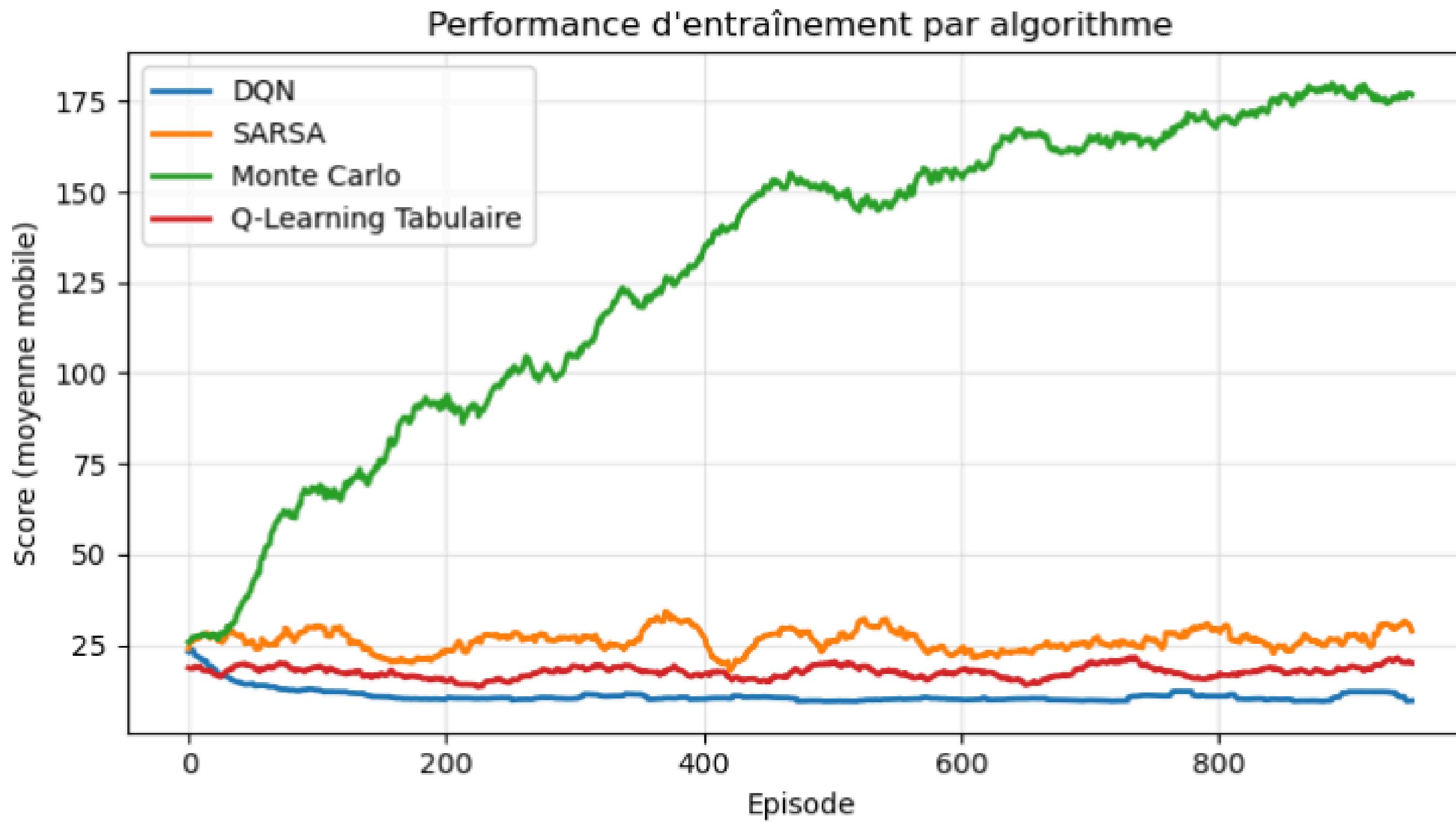
Définition



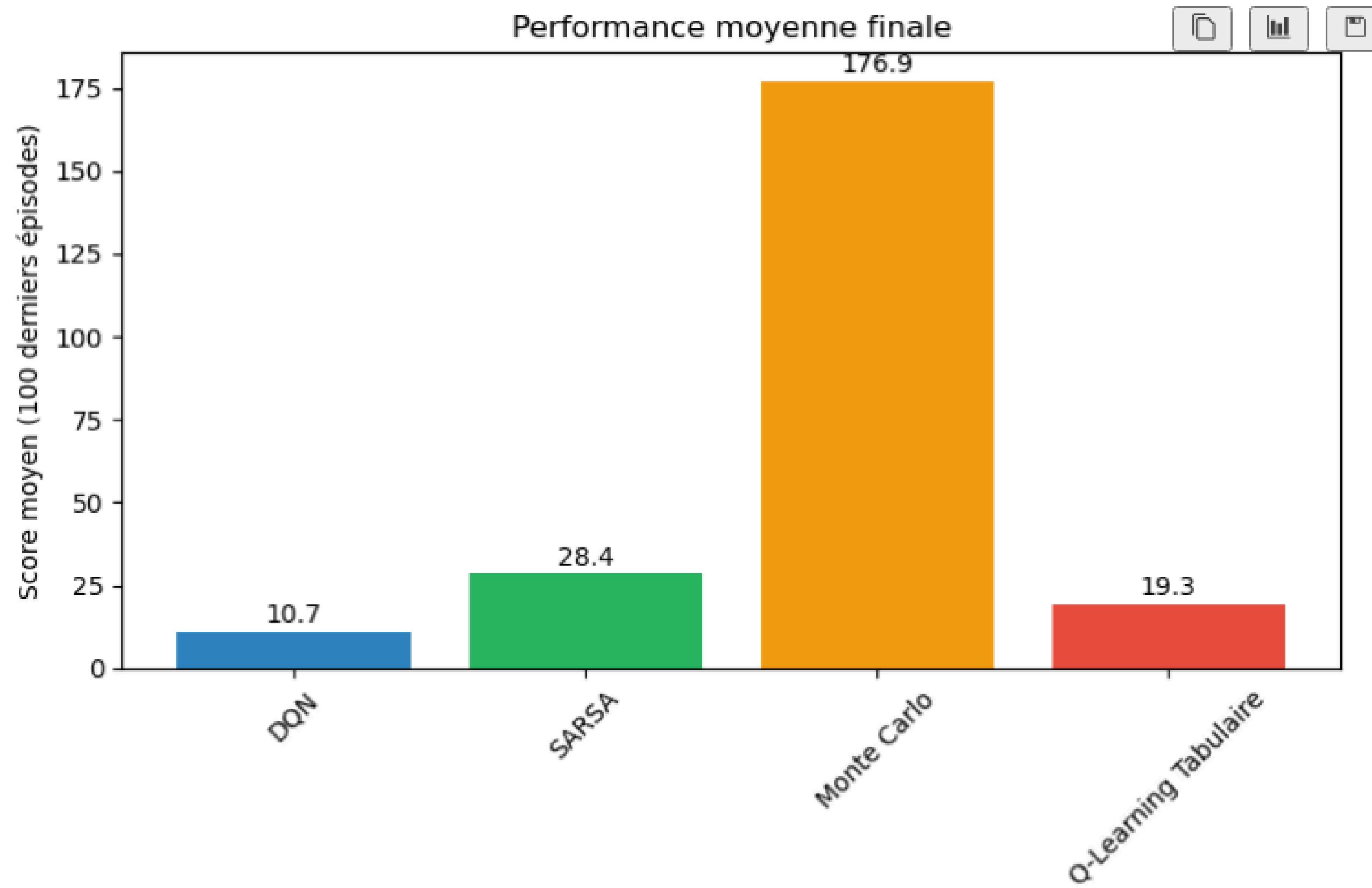
Entrainement



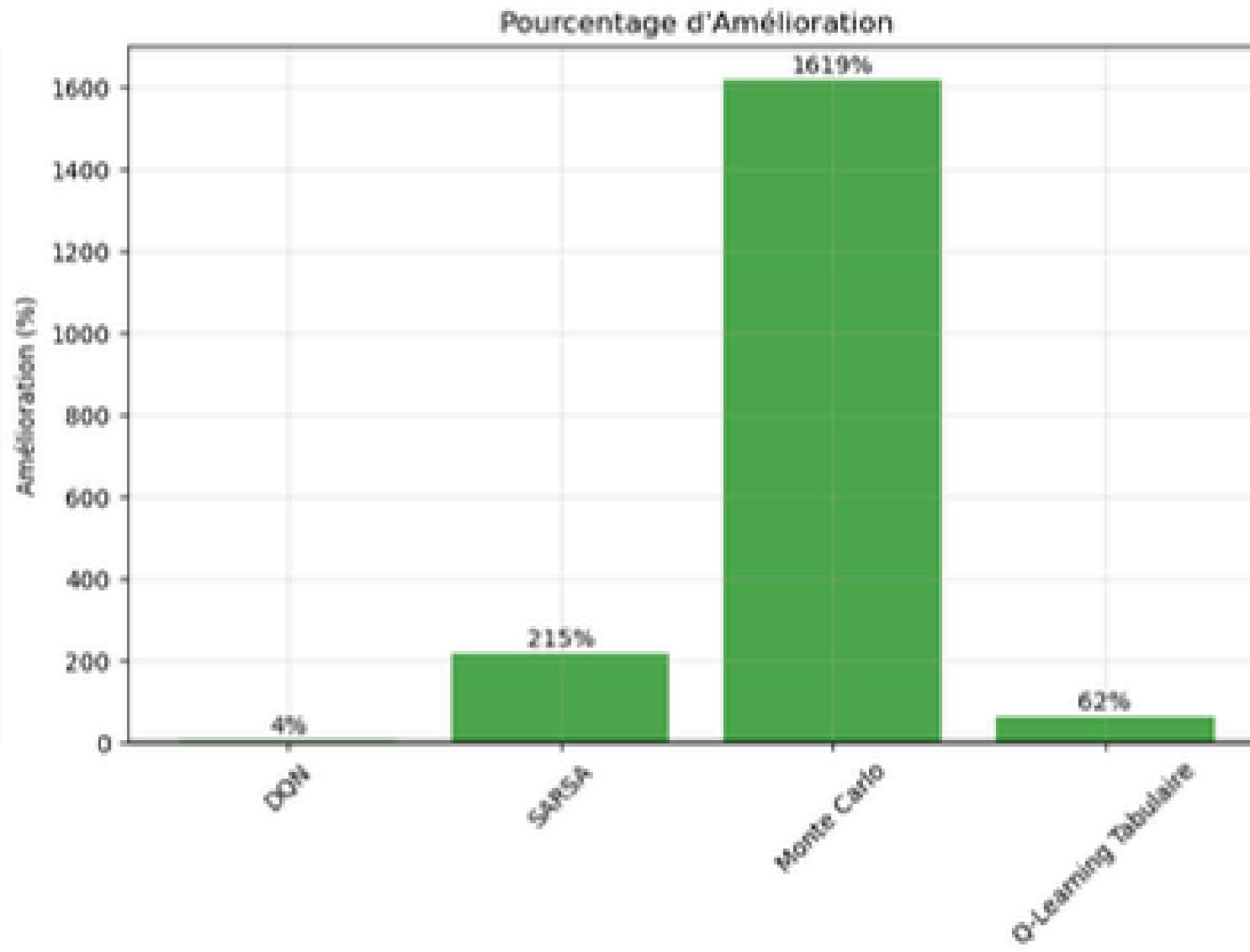
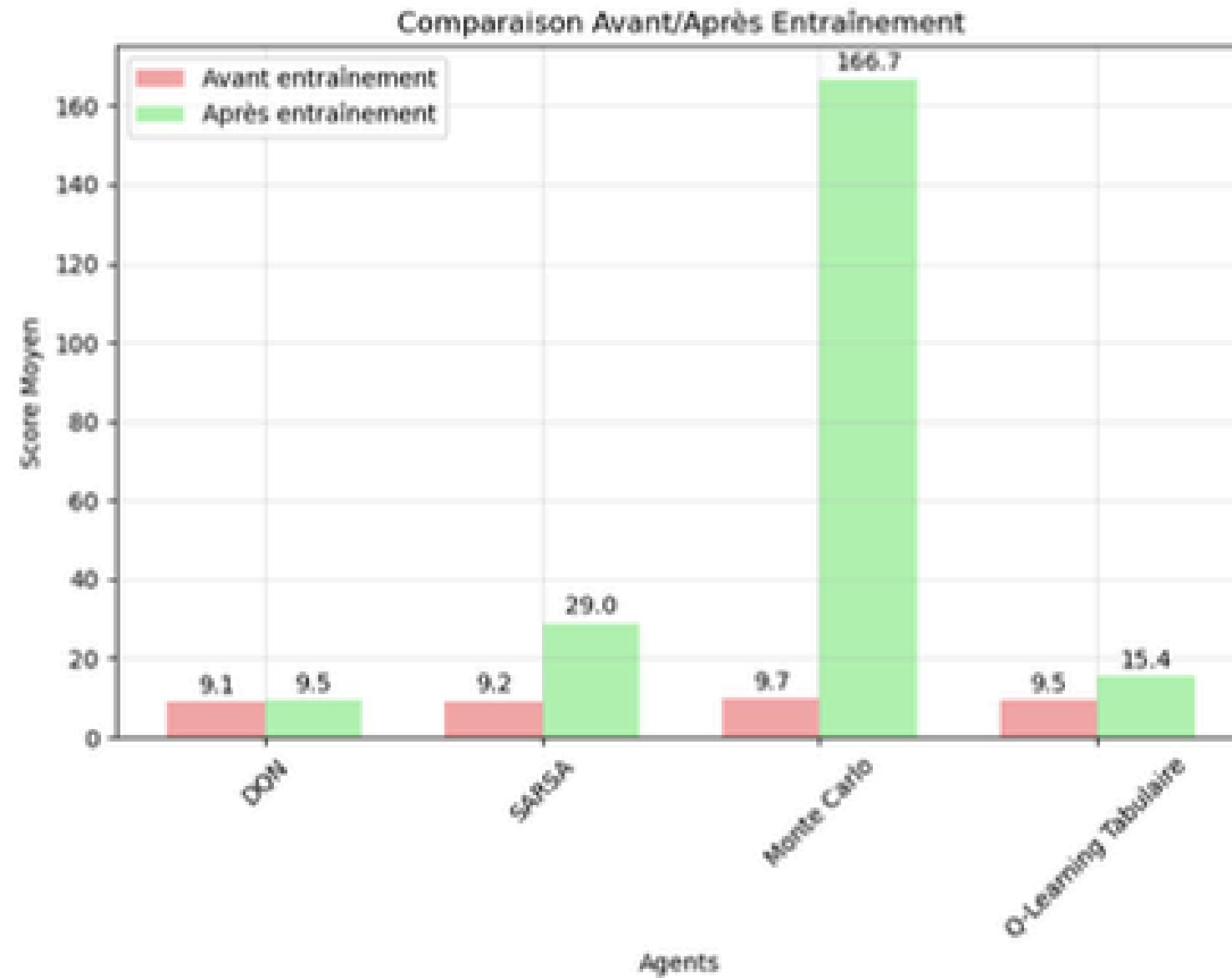
ETUDE DE PERFORMANCES



ETUDE DE PERFORMANCES



COMPAISON



COMPAISON

Résultats détaillés

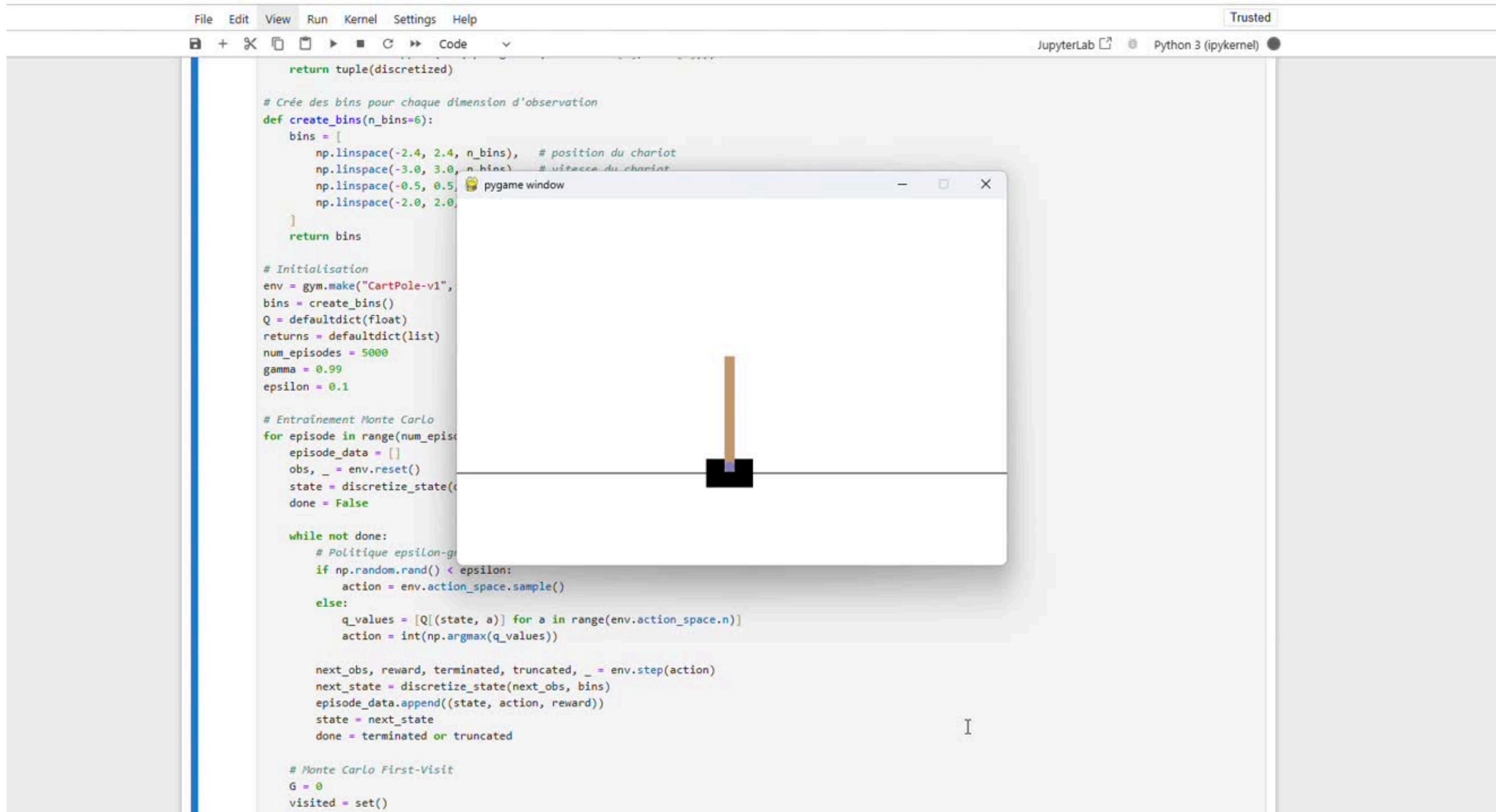
Méthode	Score moyen final	Écart-type	Score maximum	Taux de succès (≥ 200)
DQN	10.74	5.78	44	0.0%
SARSA	28.42	25.16	171	0.0%
Monte Carlo	176.87	27.84	244	17.0%

Classement final

Rang	Méthode	Score moyen final
1	Monte Carlo	176.87
2	SARSA	28.42
3	Q-Learning Tabulaire	19.30
4	DQN	10.74

DEMONSTRATION

AVANT



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for training a CartPole environment. The code uses Monte Carlo methods with an epsilon-greedy policy. A small window in the background displays the CartPole simulation, where a pole is balanced on a moving cart.

```
File Edit View Run Kernel Settings Help Trusted
JupyterLab Python 3 (ipykernel)
return tuple(discretized)

# Crée des bins pour chaque dimension d'observation
def create_bins(n_bins=6):
    bins = [
        np.linspace(-2.4, 2.4, n_bins), # position du chariot
        np.linspace(-3.0, 3.0, n_bins), # vitesse du chariot
        np.linspace(-0.5, 0.5, n_bins), # pygame window
        np.linspace(-2.0, 2.0, n_bins)
    ]
    return bins

# Initialisation
env = gym.make("CartPole-v1",
bins = create_bins()
Q = defaultdict(float)
returns = defaultdict(list)
num_episodes = 5000
gamma = 0.99
epsilon = 0.1

# Entraînement Monte-Carlo
for episode in range(num_episodes):
    episode_data = []
    obs, _ = env.reset()
    state = discretize_state(obs)
    done = False

    while not done:
        # Politique epsilon-greedy
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
            q_values = [Q[(state, a)] for a in range(env.action_space.n)]
            action = int(np.argmax(q_values))

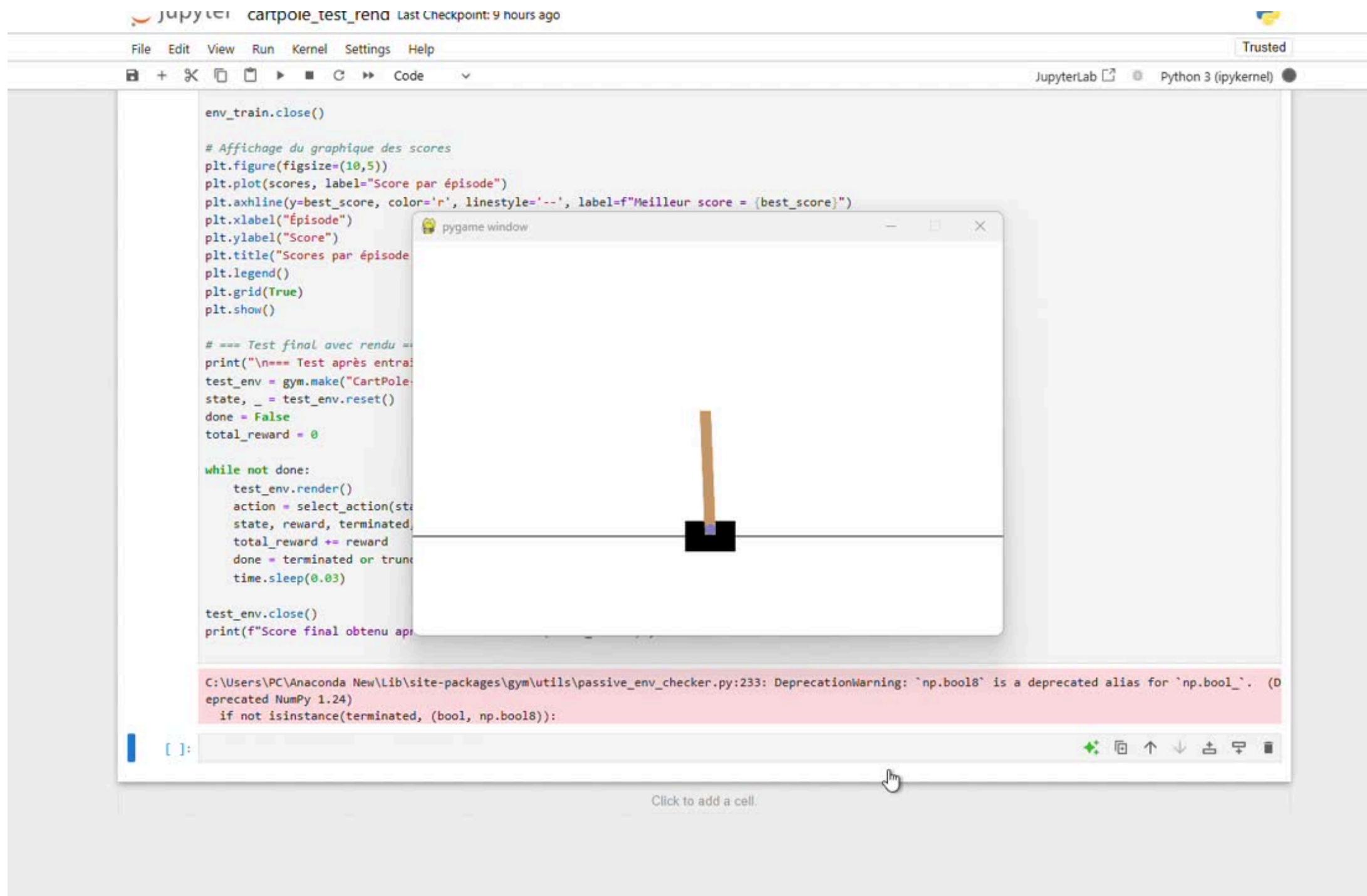
        next_obs, reward, terminated, _ = env.step(action)
        next_state = discretize_state(next_obs, bins)
        episode_data.append((state, action, reward))
        state = next_state
        done = terminated or truncated

    # Monte Carlo First-Visit
    G = 0
    visited = set()

    for t in reversed(range(len(episode_data))):
        G = G * gamma + episode_data[t][2]
        if (episode_data[t][0], episode_data[t][1]) not in visited:
            Q[(episode_data[t][0], episode_data[t][1])] = G
            visited.add((episode_data[t][0], episode_data[t][1]))
```

DEMONSTRATION

APRES ENTRAINEMENT SUR 300 EPOQUES AVEC DQN



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for training and testing a DQN agent on the CartPole environment. A separate window titled "pygame window" displays the CartPole simulation, where the pole is balanced upright on the cart. The notebook also shows a warning message about a deprecated NumPy alias.

```
env_train.close()

# Affichage du graphique des scores
plt.figure(figsize=(10,5))
plt.plot(scores, label="Score par épisode")
plt.axhline(y=best_score, color='r', linestyle='--', label=f"Meilleur score = {best_score}")
plt.xlabel("Épisode")
plt.ylabel("Score")
plt.title("Scores par épisode")
plt.legend()
plt.grid(True)
plt.show()

# === Test final avec rendu ===
print("\n==== Test après entraînement ===")
test_env = gym.make("CartPole-v1")
state, _ = test_env.reset()
done = False
total_reward = 0

while not done:
    test_env.render()
    action = select_action(state, reward, terminated, truncated)
    state, reward, terminated, truncated, _ = test_env.step(action)
    total_reward += reward
    done = terminated or truncated
    time.sleep(0.03)

test_env.close()
print(f"Score final obtenu après {episodes} épisodes : {total_reward}")
```

C:\Users\PC\Anaconda New\Lib\site-packages\gym\utils\passive_env_checker.py:233: DeprecationWarning: `np.bool8` is a deprecated alias for `np.bool_`. (DeprecationWarning)

```
[ ]:
```

Click to add a cell.

AVANTAGES ET LIMITES



AVANTAGES

- **Apprentissage sans supervision explicite** : l'agent apprend par interaction.
- **Capacité à traiter des états complexes** : images, texte, capteurs... grâce aux réseaux profonds.
- **Décision séquentielle optimisée** : prend en compte l'effet à long terme des actions.
- **Flexibilité** : applicable à des contextes variés (robotique, jeux, finance, IoT, etc.).
- **Adaptabilité** : l'agent peut s'adapter à un environnement changeant.

AVANTAGES ET LIMITES

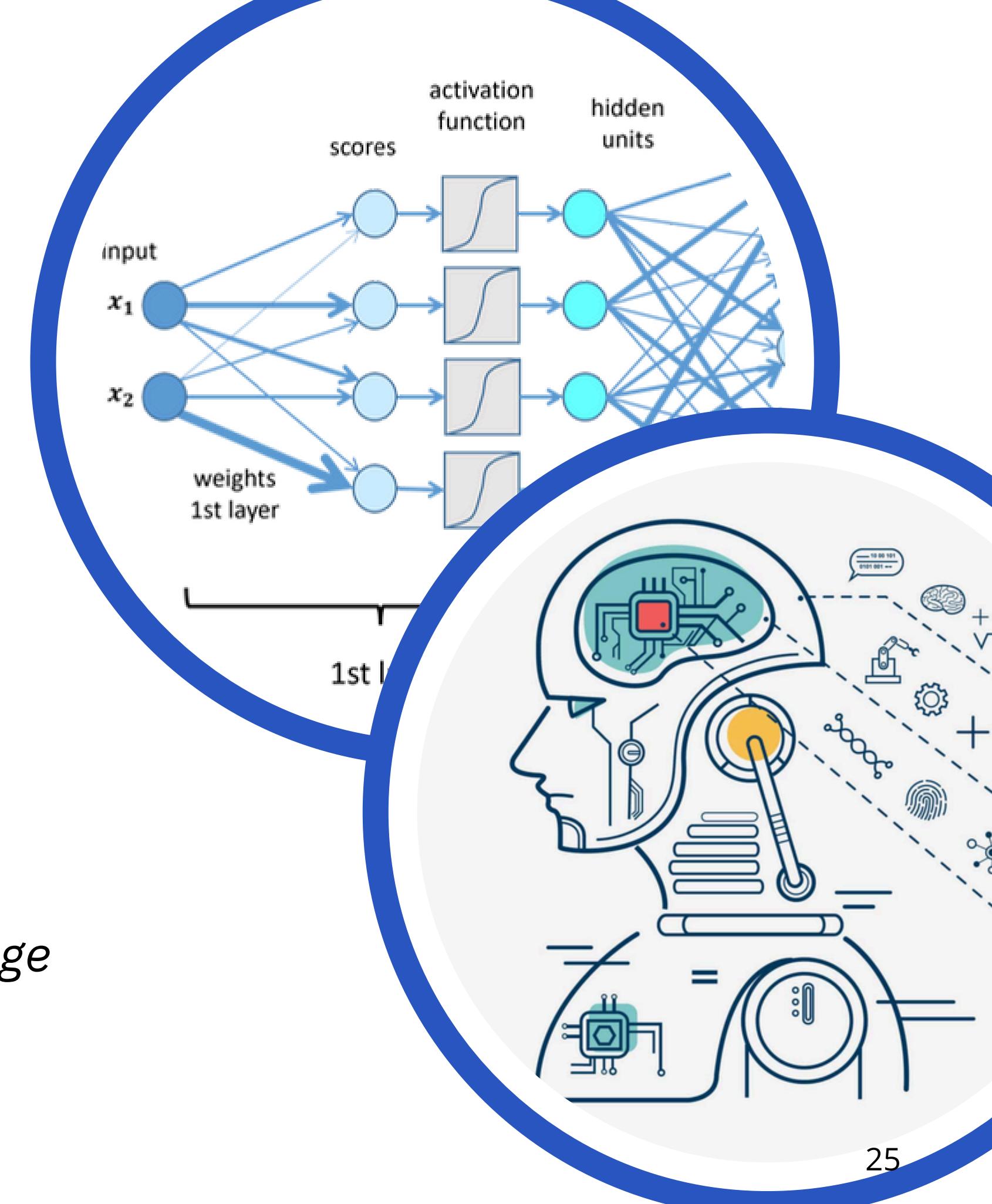


LIMITES

- **Coût computationnel élevé** : entraînement long, besoin de beaucoup de données et de ressources.
- **Instabilité de l'apprentissage** : sensible aux hyperparamètres, à la récompense et au réseau.
- **Exploration inefficace** : difficile dans des environnements vastes ou avec peu de signaux de récompense.
- **Difficulté de généralisation** : les agents peuvent surapprendre des stratégies non transférables.
- **Complexité des environnements multi-agents** : non-stationnarité, coordination difficile.

PERPECTIVES ET CONCLUSION

- Tester des variantes avancées du DQN : Double DQN et Dueling DQN.
- Intégrer le DRL avec l'apprentissage non supervisé et l'apprentissage par transfert
- Explorer des approches acteur-critique ou basées sur les politiques (PPO, A3C).
- Tester des techniques de méta-apprentissage (Meta-Learning)



**MERCIE DE
VOTRE
ATTENTION**

SOURCES

<https://www.v7labs.com/blog/deep-reinforcement-learning-guide>

<https://medium.com/@heyamit10/sarsa-algorithms-explained-b94078fd658b>

Reinforcement Learning, an introduction 2nd edition, by Richard S. Sutton and Andrew G. Barto

<https://www.davidsilver.uk/teaching/>

[https://www.ibm.com/think/topics/reinforcement-learning#:~:text=Reinforcement%20learning%20\(RL\)%20is%20a,instruction%20by%20a%20human%20user.](https://www.ibm.com/think/topics/reinforcement-learning#:~:text=Reinforcement%20learning%20(RL)%20is%20a,instruction%20by%20a%20human%20user.)