

## Part B — Long Question (8 Marks)

### Q1. Explain the interaction loop between an agent and its environment in Reinforcement Learning (RL). Describe the roles of state, action, and reward.

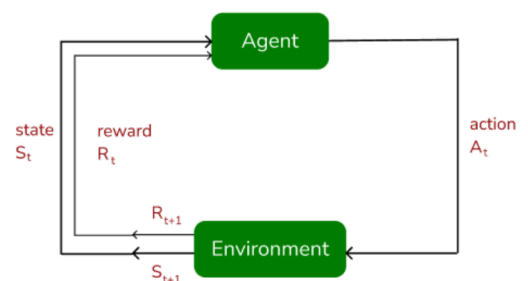
Reinforcement Learning (RL) is a type of machine learning where an **agent** learns to make decisions by interacting with an **environment**. The agent's goal is to maximize its **cumulative reward** over time through trial-and-error.

The agent–environment interaction is modeled as a **Markov Decision Process (MDP)**, consisting of **states**, **actions**, **rewards**, and **transition dynamics**.

#### Interaction Loop

At discrete time steps  $t = 0, 1, 2, \dots$ :

1. The **agent** observes the current state  $S_t$ .
2. Based on its policy  $\pi(a|s)$ , the agent chooses an **action**  $A_t$ .
3. The **environment** responds by returning:
  - A reward  $R_{t+1}$  (feedback signal).
  - A new state  $S_{t+1}$ .



This process repeats, forming a **loop**:

$$S_t \xrightarrow{A_t} \text{Environment} \xrightarrow{(R_{t+1}, S_{t+1})} \text{Agent}$$

#### Roles of State, Action, and Reward

##### (a) State ( $S_t$ )

- A state represents the **situation of the environment** at time  $t$ .
- It contains all necessary information for decision-making.
- Example:
  - In chess  $\rightarrow$  the board configuration.
  - In navigation  $\rightarrow$  agent's current grid location.

**Role:** Provides the **context** for the agent's decision.

##### (b) Action ( $A_t$ )

- An action is the **choice made by the agent** to interact with the environment.
- Actions can be:
  - **Discrete** (e.g., move left, right, up, down).
  - **Continuous** (e.g., steering angle, velocity).
- Example: In a navigation task, actions are {Up, Down, Left, Right}.

**Role:** Represents the **control** the agent has to influence the environment.

### (c) Reward ( $R_{t+1}$ )

- A reward is a **scalar feedback signal** from the environment.
- It measures the **immediate quality of an action** in a given state.
- Example:
  - +1 for reaching a goal, 0 for any other step.
  - +10 for winning a game, -1 for losing.

**Role:** Provides the **objective signal** that guides learning.

### Objective of the Agent

The agent's goal is to maximize the **expected cumulative discounted reward (return)**:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where:

- $\gamma \in [0, 1]$  is the **discount factor**.
- $\gamma$  close to 0  $\rightarrow$  agent focuses on **immediate rewards**.
- $\gamma$  close to 1  $\rightarrow$  agent values **long-term rewards**.

The **objective** is:

$$\max_{\pi} \mathbb{E}[G_t]$$

### Example: Navigation Task

- **State:** Agent's current grid position.
- **Actions:** {Up, Down, Left, Right}.
- **Rewards:** +1 for reaching the goal, 0 otherwise.

**Interaction:**

1. Agent at start  $\rightarrow$  observes  $S_0$ .
2. Chooses action "Right"  $\rightarrow$  moves to new position  $S_1$ .
3. Environment returns  $R_1=0$ .
4. Repeats until the goal is reached (reward = +1).

Over time, the agent **learns an optimal policy** to reach the goal efficiently.

**Q2. Define the multi-armed bandit (MAB) problem. Compare the MAB setting with full reinforcement learning (RL), explaining what is missing in the bandit problem.**

The **multi-armed bandit (MAB) problem** is one of the simplest yet most fundamental problems in Reinforcement Learning (RL). It captures the **exploration–exploitation trade-off** in decision-making, where an agent must repeatedly choose among a set of uncertain options and learn which ones yield the highest rewards.

The name comes from the analogy of a gambler facing multiple slot machines (bandits) in a casino, each with an unknown payout distribution.

### Definition of the Multi-Armed Bandit Problem

- A **K-armed bandit** provides **K independent actions (arms)**.
- At each time step  $t$ :
  1. The agent selects an **arm**  $A_t \in \{1, 2, \dots, K\}$ .
  2. The chosen arm produces a **reward**  $R_t$  drawn from a fixed but unknown probability distribution with mean  $\mu_a$ .

Mathematically:

$$R_t \sim P(\cdot | A_t = a), \quad \text{with mean reward } \mu_a = \mathbb{E}[R_t | A_t = a]$$

- The rewards are **i.i.d.** (independent and identically distributed) over time for each arm, in the **stationary case**.

**Objective:** Maximize the **cumulative reward** (or equivalently minimize **regret**) over time by finding the best arm(s).

### Exploration vs. Exploitation

The key challenge in MAB is balancing two behaviors:

- **Exploration:** Trying out different arms to estimate their rewards.
- **Exploitation:** Using current knowledge to select the arm with the highest estimated reward.

An agent must carefully manage this trade-off, since too much exploration wastes opportunities, while too much exploitation risks getting stuck with a suboptimal arm.

### Intuition: How RL Generalizes Bandits

- The MAB problem can be seen as a **special case of RL**:
  1. It is equivalent to an MDP with **a single state** and **only immediate rewards**.
- In contrast, RL extends this setting by:
  1. **Introducing states** → the agent must consider where it is.
  2. **Allowing state transitions** → actions influence future possibilities.
  3. **Handling delayed consequences** → the agent must plan sequences of actions, not just single-step choices.
  4. **Learning value functions** → assigning value to states and actions to optimize long-term returns.

### Example

#### (a) MAB Example

- Suppose a gambler has 3 slot machines:

- Arm 1: Pays with probability 0.3.
  - Arm 2: Pays with probability 0.5.
  - Arm 3: Pays with probability 0.7.
- The gambler does not know these probabilities and must learn by trial and error which machine is best.

### (b) RL Example

- A robot in a grid-world:
  - **State:** Current position on the grid.
  - **Actions:** Move up, down, left, right.
  - **Reward:** +1 for reaching the goal, 0 otherwise.
- Unlike MAB, the robot's actions influence **future states** (its position), and rewards may be **delayed** (goal reached after several steps).

### Comparison of MAB vs. Full RL

Aspect	Multi-Armed Bandit (MAB)	Full Reinforcement Learning (RL)
States	No states (single-state MDP). The problem is <b>stateless</b> .	Multiple states. Agent observes different states over time.
Transitions	No state transitions. Each action only gives an immediate reward.	Actions influence future states (environment dynamics).
Reward Consequences	Rewards are <b>immediate</b> only. No delayed effects.	Rewards can be <b>delayed</b> , requiring long-term credit assignment.
Planning	No planning required (myopic decision-making).	Planning is needed to optimize long-term returns.
Value Functions	Only the <b>value of each action</b> (expected reward of each arm).	Value functions are defined over <b>states</b> and <b>state-action pairs</b> .
Objective	Find the best arm (maximize immediate reward).	Learn a policy that maximizes long-term discounted return.

### Q3. Describe the Upper Confidence Bound (UCB) method in the context of K-armed bandits. Compare it with $\epsilon$ -greedy in terms of exploration efficiency.

The **multi-armed bandit (MAB)** problem requires an agent to repeatedly select among  $K$  arms, each providing stochastic rewards with unknown means. The key challenge is to balance:

- **Exploration** → trying less-sampled arms to gain more information.
- **Exploitation** → pulling the arm believed to be the best so far.

The **Upper Confidence Bound (UCB)** algorithm provides a **principled approach** to this trade-off by adding an **optimism bonus** to each arm's estimated value.

**The UCB Idea** :The intuition behind UCB is:

- Instead of only considering the **empirical mean reward** of each arm, also account for the **uncertainty** in the estimate.
- Choose the arm with the **highest upper confidence bound**, meaning the arm that could plausibly be the best given current data.

This avoids random exploration and ensures each arm is explored until there is enough evidence it is suboptimal.

### The UCB1 Algorithm

For each arm  $a$ :

- Maintain:
  - $\hat{Q}_t(a)$ : empirical mean reward estimate at time  $t$ .
  - $N_t(a)$ : number of times arm  $a$  has been selected so far.

At time step  $t \geq K$ :

$$a_t \in \arg \max_a \left[ \hat{Q}_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

where:

- The first term  $\hat{Q}_t(a)$  = exploitation (current estimate).
- The second term  $c \sqrt{\frac{\ln t}{N_t(a)}}$  = exploration bonus.
  - Large when  $N_t(a)$  is small (high uncertainty).
  - Shrinks as  $N_t(a)$  increases (more confidence).
- $c > 0$  is a constant controlling exploration (commonly  $c = \sqrt{2}$ ).

### Intuition Behind UCB

- Each arm is assigned an **upper bound estimate** of its reward.
- The agent always selects the arm with the maximum of these upper bounds.
- This ensures:
  - Promising arms (high empirical mean) are chosen often.
  - Unexplored arms (high uncertainty) are also tried enough times.
- Over time, exploration naturally decreases because uncertainty reduces.

## Comparison: UCB vs. $\epsilon$ -Greedy

Aspect	$\epsilon$ -Greedy	UCB
Exploration strategy	With probability $\epsilon$ , choose a random arm.	Selects arms with high estimated value <b>or</b> high uncertainty (optimism bonus).
Targeting	Exploration is <b>uniform random</b> , even on arms known to be poor.	Exploration is <b>directed</b> towards uncertain arms.
Efficiency	Can waste trials on clearly suboptimal arms.	More sample-efficient, avoids unnecessary pulls.
Adaptivity	Exploration rate fixed ( $\epsilon$ ) or decayed manually.	Exploration naturally reduces as confidence improves.
Performance	Works well but may converge slowly, especially with many arms.	Typically achieves stronger performance, especially when reward gaps are moderate to large.

### Example

- Suppose there are 3 arms:
  - Arm A: Estimated mean = 0.7, sampled 100 times.
  - Arm B: Estimated mean = 0.6, sampled 5 times.
  - Arm C: Estimated mean = 0.4, sampled 5 times.
- **$\epsilon$ -Greedy:** With probability  $1-\epsilon$ , chooses Arm A (best estimate). With probability  $\epsilon$ , randomly explores A, B, or C (wasting some pulls on C).
- **UCB:**
  - Arm A's uncertainty is low (many samples), so its bonus is small.
  - Arm B's uncertainty is high, so its upper bound might exceed Arm A's.
  - Algorithm chooses Arm B to gather more information.
  - Arm C, despite uncertainty, is less likely to be chosen if already much worse.

### Q4. Explain the key elements of reinforcement learning with suitable examples. Discuss its limitations in real-world applications.

Reinforcement Learning (RL) is a computational approach to decision-making where an **agent** interacts with an **environment** to maximize long-term cumulative rewards.

The framework is usually described by the **Markov Decision Process (MDP)** and is built upon a few key elements that define the learning problem.

### Key Elements of Reinforcement Learning

#### (i) Policy ( $\pi$ )

- A **policy** is the agent's strategy for selecting actions.
- It defines a mapping from **states to actions** (deterministic or stochastic).

- Formally:

$$\pi(a|s) = P(A_t = a \mid S_t = s)$$

- Example:
  - In robot navigation, a policy might say “If at the goal, stop; otherwise move closer.”
  - In recommendation systems, a policy maps user features to which content to show.

**Role:** Directs agent's behavior.

## (ii) Reward Signal

- A **reward** is a scalar feedback from the environment that indicates the immediate value of an action.
- It defines the **objective of the agent**.
- Example:
  - In robotics: +1 for reaching the goal, 0 otherwise.
  - In recommendation systems: +1 if a user clicks on a suggested video.

**Role:** Provides the measure of success for the agent.

## (iii) Value Functions

- While rewards evaluate **immediate actions**, **value functions** estimate the **expected long-term return**.
- Two types:
  - **State-value function**  $v^\pi(s)$ : Expected return from state  $s$  following policy  $\pi$ .
  - **Action-value function**  $q^\pi(s, a)$ : Expected return from state  $s$ , taking action  $a$ , then following policy  $\pi$ .
- Example:
  - In chess, the value of a state may estimate the probability of winning from that board position.

**Role:** Helps the agent evaluate actions beyond immediate rewards.

## (iv) Model of the Environment (Optional)

- A **model** predicts:
  - The next state  $S_{t+1}$  given the current state and action.
  - The reward  $R_{t+1}$ .
- If available, a model enables **planning** by simulating future trajectories.
- Example:
  - A physics-based simulator for a robot arm.
  - A user-behavior prediction model in recommender systems.

**Role:** Allows planning ahead, not just trial-and-error learning.

## Examples of RL Applications

### (a) Robot Control

- **States:** Joint angles, velocities, and sensor inputs.
- **Actions:** Torques applied to motors.
- **Rewards:** Progress toward a task (e.g., walking or grasping an object).

### (b) Recommendation Systems

- **States:** User context (age, history, preferences).
- **Actions:** Which content, ad, or video to show.
- **Rewards:** Clicks, views, engagement time.

These examples highlight how RL elements fit into real-world tasks.

## Limitations of RL in Real-World Applications

1. **Sample Inefficiency**
  - RL often requires millions of interactions to learn effectively.
  - Example: Training an RL robot in the real world is impractical due to time and cost.
2. **Safety and Exploration Risk**
  - Exploration may involve risky actions.
  - Example: A self-driving car cannot safely “try” random maneuvers.
3. **Non-Stationarity and Partial Observability**
  - Real environments may change over time.
  - The agent may not observe the full state (hidden variables).
4. **Reward Misspecification**
  - Designing a correct reward function is hard.
  - Agents may exploit loopholes (“specification gaming”).
  - Example: A cleaning robot might hide dirt under the carpet if the reward is “floor looks clean.”
5. **High Computational Cost**
  - Training deep RL models requires massive computational resources.
  - Example: AlphaGo training used thousands of TPUs for weeks.

**Q5. Discuss the significance of exploration strategies in reinforcement learning (RL). Compare random exploration with guided exploration methods, giving examples of when each is useful.**

In Reinforcement Learning (RL), an agent must balance:

- **Exploitation** → choosing the best-known action to maximize reward now.
- **Exploration** → trying uncertain actions to gather information that may lead to better long-term rewards.

**Exploration strategies** determine *how effectively and efficiently* an agent discovers high-reward behavior. The choice of strategy directly affects learning speed, sample efficiency, and overall performance.

### Significance of Exploration

- Without exploration, the agent may **get stuck** exploiting a suboptimal policy.
- Too much random exploration can waste resources, while too little exploration may miss the optimal solution.
- Efficient exploration is especially critical in:
  - **Sparse-reward environments** (e.g., navigation tasks where rewards are rare).
  - **Expensive real-world domains** (e.g., robotics, healthcare, online systems).

Thus, exploration strategy determines **how quickly and safely an agent learns**.

### Random Exploration

- **Definition:** The agent occasionally chooses random actions regardless of their estimated value.
- Common method:  **$\epsilon$ -greedy**, where:
  - With probability  $1-\epsilon$ , choose the best-known action.
  - With probability  $\epsilon$ , choose a random action.
- **Advantages:**
  - Simple and robust.
  - Works well when the action set is small.
  - Suitable when interactions with the environment are cheap.
- **Limitations:**
  - Exploration is uniform and undirected.
  - Can waste trials on clearly suboptimal actions.

### Example:

- In simple games or simulations (e.g., a slot machine with 3 arms),  $\epsilon$ -greedy can be effective since trying random arms is inexpensive.

### Guided Exploration

- **Definition:** Uses structured information, uncertainty estimates, or intrinsic signals to guide exploration toward actions or states that are more promising.
- **Methods:**
  - **Upper Confidence Bound (UCB):** Selects actions with either high estimated value or high uncertainty.
  - **Thompson Sampling:** Samples actions according to the probability they are optimal (posterior sampling).
  - **Optimism in the Face of Uncertainty:** Agents act as if uncertain actions may be highly rewarding until proven otherwise.

- **Count-based or Intrinsic Motivation (stateful RL):** Encourages the agent to visit novel or unexplored states.
- **Advantages:**
  - More efficient than random exploration.
  - Focuses exploration on arms or states with high information gain.
  - Reduces wasted trials.
- **Limitations:**
  - Requires additional computation (e.g., uncertainty estimates).
  - May be harder to implement in very large or complex environments.

### Example:

- **Online A/B testing:** Thompson sampling or UCB ensures traffic is adaptively allocated to promising website versions.
- **Sparse-reward navigation:** Count-based bonuses or curiosity-driven exploration help agents reach new states more reliably than random wandering.

### Comparison: Random vs. Guided Exploration

Aspect	Random Exploration ( $\epsilon$ -greedy)	Guided Exploration (UCB, Thompson, Intrinsic)
Mechanism	Chooses random actions uniformly.	Directs exploration using uncertainty or novelty.
Efficiency	May waste trials on poor actions.	More sample-efficient, avoids unnecessary pulls.
Complexity	Simple, easy to implement.	Requires modeling uncertainty or novelty.
Best suited for	Small action spaces, cheap interactions, unreliable uncertainty estimates.	Expensive environments, large state spaces, sparse rewards.

### Part A — Short Questions (2 marks each)

#### Q1. Mention two unique characteristics of reinforcement learning.

1. **Learning by trial and error** → The agent learns from interaction with the environment, often with delayed or sparse rewards.
2. **Sequential decision-making** → The agent's actions influence future states, making the data distribution non-i.i.d.

*(Other hallmarks include the exploration–exploitation trade-off and bootstrapping.)*

## Q2. Define reinforcement learning and explain how it differs from supervised learning.

- **Reinforcement Learning (RL):** Studies how an agent should act in an environment to maximize expected cumulative reward.
- **Difference from Supervised Learning:**
  - RL does not have labeled input–output pairs.
  - The agent receives only scalar reward signals after acting.
  - RL involves **temporal credit assignment** (deciding which past actions led to future rewards).
  - Data distribution is policy-dependent and non-stationary.

## Q3. In the context of the multi-armed bandit problem, explain what an “optimistic initial value” means.

- Initial estimates  $Q_1(a)$  for action values are set **higher than any expected true reward**.
- This artificially optimistic prior encourages the agent to **try each action at least once**, since unexplored arms remain attractive until sampled.
- Helps prevent early commitment to a suboptimal arm.

## Q4. What is the exploration vs. exploitation dilemma?

- At each step, the agent must choose between:
  1. **Exploitation** → Selecting the current best-known action for immediate reward.
  2. **Exploration** → Trying uncertain actions to improve knowledge and possibly gain higher long-term return.

This trade-off is central to reinforcement learning.

## Q5. What are the key elements of reinforcement learning?

1. **Agent** – Learner/decision-maker.
2. **Environment** – External system with which the agent interacts.
3. **Policy ( $\pi$ )** – Strategy mapping states to actions.
4. **Reward signal (R)** – Feedback defining the goal.
5. **Value functions (v, q)** – Expected return estimates for states or state–action pairs.
6. **(Optional) Model** – Predicts environment dynamics (transition and rewards).

Objective: maximize expected return

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad \gamma \in [0, 1).$$

## Q6. Why are bandit problems important to study despite being simpler than full RL?

- They isolate the **core challenge of exploration** without state dynamics.
- Provide **theoretical guarantees** via regret analysis.
- Serve as **building blocks** for large-scale RL, such as in action selection, online advertising, and A/B testing.

**Q7. What does it mean for an agent to follow a greedy policy?**

- A **greedy policy** always selects the action with the **highest current estimated value**:

$$a_t \in \arg \max_a \hat{Q}_t(a).$$

- It does not explore unless there is tie-breaking or additional exploration mechanisms.

**Q8. How does UCB (Upper Confidence Bound) encourage exploration compared to  $\epsilon$ -greedy?**

- **UCB**: Selects

$$a_t \in \arg \max_a \hat{Q}_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}},$$

where the second term is an **uncertainty bonus** that shrinks with more trials. This directs exploration to uncertain but potentially rewarding arms.

- **$\epsilon$ -greedy**: Explores by choosing a random action with probability  $\epsilon$ , regardless of uncertainty—hence less efficient.