EECE 5614 — Reinforcement Learning and
Decision Making Under Uncertainty

Project 1: 2-Armed Bandit Problem

Spring 2026

This report presents experimental results for the 2-armed bandit problem. Lever 1 has a Gaussian reward distribution with $\mu_1 = 8$, $\sigma^2_1 = 20$ ($Q^*(a^1) = 8$). Lever 2 has a mixture-of-Gaussians reward with equal mixing weights, $\mu_{21} = 8$, $\sigma^2_{21} = 15$ and $\mu_{22} = 14$, $\sigma^2_{22} = 10$ ($Q^*(a^2) = 11$). All experiments use 1000 steps averaged over 100 independent runs.

Part 1: ε-Greedy with Different Learning Rates

In this part, the agent uses an ε-greedy policy with initial Q-values set to zero. Four learning rate schedules are evaluated ($\alpha = 1$, $\alpha = 0.9^k$, $\alpha = 1/(1 + \ln(1+k))$, and $\alpha = 1/k$) across four exploration parameters ($\varepsilon = 0, 0.1, 0.2, 0.5$).

Key Code: ε-Greedy Agent

```
Part 1 & 2: ε-greedy Agent (action selection + Q-value update)
```

```python
class EpsilonGreedyAgent(BaseAgent):
    def __init__(self, epsilon=0.1, learning_rate_fn=None, initial_q=None):
        super().__init__()
        self.epsilon = epsilon
        self.learning_rate_fn = learning_rate_fn or (lambda k: 0.1)
        self._initial_q = initial_q if initial_q is not None else np.zeros(N_ACTIONS)
        self.q_values = self._initial_q.copy()

    def select_action(self) -> int:
        if np.random.random() < self.epsilon:
            return np.random.randint(self.n_actions)
        return self._argmax_random_tie_break(self.q_values)

    def update(self, action: int, reward: float) -> None:
        self.step_count += 1
        alpha = self.learning_rate_fn(self.step_count)
        self.q_values[action] += alpha * (reward - self.q_values[action])
```

Results

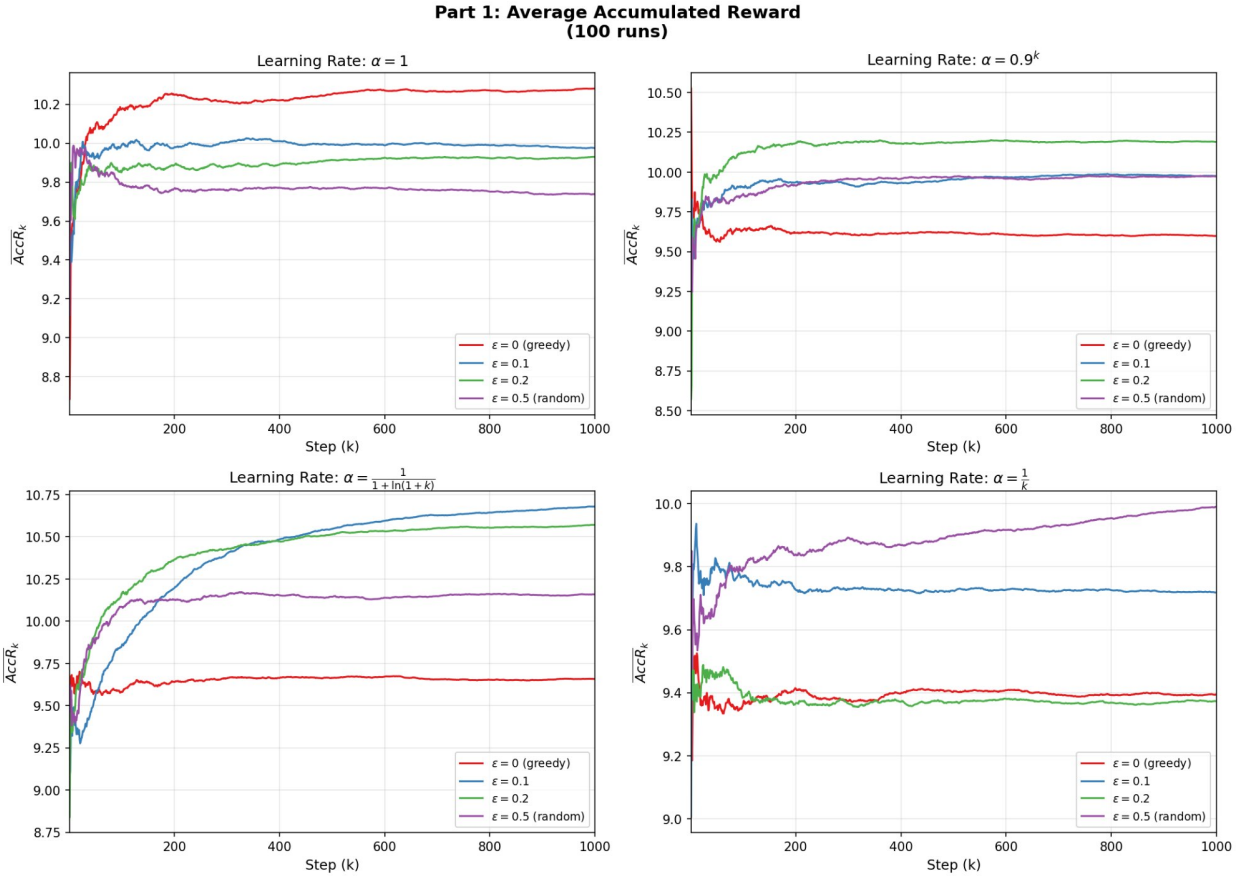


Figure 1: Average accumulated reward for each learning rate and ε value.

Average Q-Values After 1000 Steps

Table: α = 1

ε-greedy	$Q(a^1)$	$Q^*(a^1)$	$Q(a^2)$	$Q^*(a^2)$
ε = 0 (greedy)	-2.6457	8	7.5888	11
ε = 0.1	4.4512	8	7.8314	11
ε = 0.2	4.9891	8	9.0675	11
ε = 0.5 (random)	6.3266	8	9.5972	11

Table: $\alpha = 0.9^k$

ε-greedy	$Q(a^1)$	$Q^*(a^1)$	$Q(a^2)$	$Q^*(a^2)$
ε = 0 (greedy)	3.6903	8	5.8081	11
ε = 0.1	4.7889	8	8.0003	11
ε = 0.2	5.2921	8	9.5270	11
ε = 0.5 (random)	7.3708	8	10.0612	11

Table: $\alpha = 1/(1 + \ln(1+k))$

ε-greedy	$Q(a^1)$	$Q^*(a^1)$	$Q(a^2)$	$Q^*(a^2)$
$\varepsilon = 0$ (greedy)	3.5235	8	6.1840	11
$\varepsilon = 0.1$	7.5219	8	10.8431	11
$\varepsilon = 0.2$	7.6853	8	11.0832	11
$\varepsilon = 0.5$ (random)	7.7884	8	10.9710	11

Table: $\alpha = 1/k$

ε-greedy	$Q(a^1)$	$Q^*(a^1)$	$Q(a^2)$	$Q^*(a^2)$
$\varepsilon = 0$ (greedy)	4.3063	8	5.0688	11
$\varepsilon = 0.1$	4.4627	8	7.4912	11
$\varepsilon = 0.2$	6.1062	8	7.6497	11
$\varepsilon = 0.5$ (random)	7.1132	8	10.3762	11

Interpretation

α **= 1/(1 + ln(1+k))** with ε **= 0.1** achieves the highest final average accumulated reward (approximately 10.68). This learning rate decays slowly enough to continue learning throughout the 1000 steps, yet fast enough to stabilize Q-value estimates. Its final Q-values ($Q(a^1) = 7.52$, $Q(a^2) = 10.84$) are closest to the true values among all configurations with moderate exploration.

α **= 1** performs poorly because the Q-value estimate is simply replaced by the most recent reward at every step, causing high variance and preventing convergence. The greedy variant ($\varepsilon = 0$) with $\alpha = 1$ appears to perform well initially because it locks onto the action with the highest early reward, but its Q-value estimates are extremely inaccurate ($Q(a^1) = -2.65$, $Q(a^2) = 7.59$), showing that the high accumulated reward is an artifact of exploitation without learning.

α **= 0.9^k** decays exponentially fast, effectively freezing learning after a few dozen steps. This prevents Q-values from converging to the true values and limits the agent to exploiting early, inaccurate estimates.

α **= 1/k** (the sample-average rule) converges theoretically but decays quickly, so by step 1000 the learning rate is very small. Combined with low exploration ($\varepsilon = 0$ or 0.1), the agent may not have sampled lever 2 enough to discover its superiority, leading to suboptimal final rewards.

Across all learning rates, ε **= 0 (greedy)** consistently underperforms because it gets locked into a suboptimal action early and never explores. Higher ε values improve Q-value accuracy (since both arms get sampled) but reduce reward due to excessive exploration. The sweet spot is $\varepsilon = 0.1$, which balances exploration and exploitation effectively.

Part 2: Optimistic Initial Values

This part examines how optimistic initialization of Q-values affects learning. Using a fixed $\alpha = 0.1$ and $\varepsilon = 0.1$, three initializations are compared: Q = [0, 0], Q = [8, 11], and Q = [20, 20]. The same ε-greedy agent from Part 1 is used.

Key Code

The same ε-greedy agent is used (see Part 1 code). The only difference is the **initial_q** parameter passed to the constructor.

Results

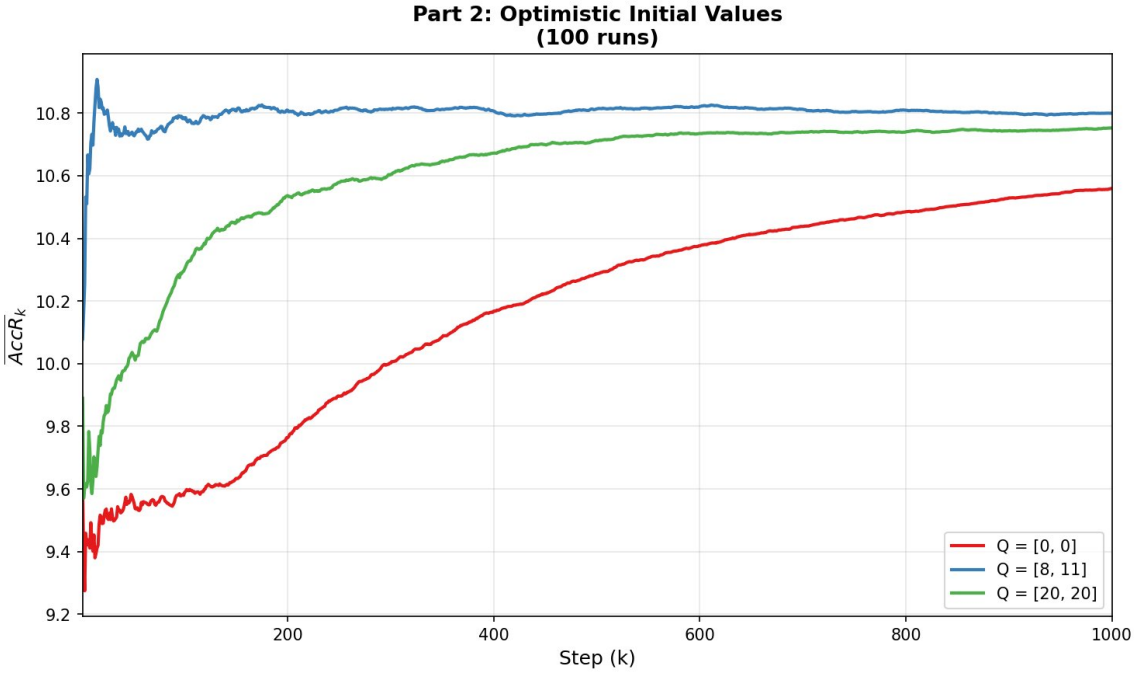


Figure 2: Average accumulated reward for different initial Q-values.

Average Q-Values After 1000 Steps

ε-greedy	$Q(a^1)$	$Q^*(a^1)$	$Q(a^2)$	$Q^*(a^2)$
Q = [0, 0]	7.7081	8	11.0824	11
Q = [8, 11]	7.6620	8	10.9850	11
Q = [20, 20]	7.5776	8	11.0648	11

Interpretation

Q = [8, 11] (the true Q-values) achieves the highest accumulated reward from the very start. Since the initial estimates are already accurate, the agent immediately begins exploiting lever 2

(the optimal action) while ε-exploration provides enough sampling to maintain accurate estimates. This results in the fastest convergence and highest cumulative performance.

Q = [20, 20] (optimistic initialization) starts lower because both Q-values are overestimated. Every observed reward is below the initial estimate, causing Q-values to decrease. This naturally encourages exploration: when one action's Q-value drops, the agent tries the other. The optimistic values act as a built-in exploration bonus, helping the agent quickly discover lever 2's superiority. After approximately 400 steps, its performance converges close to Q = [8, 11].

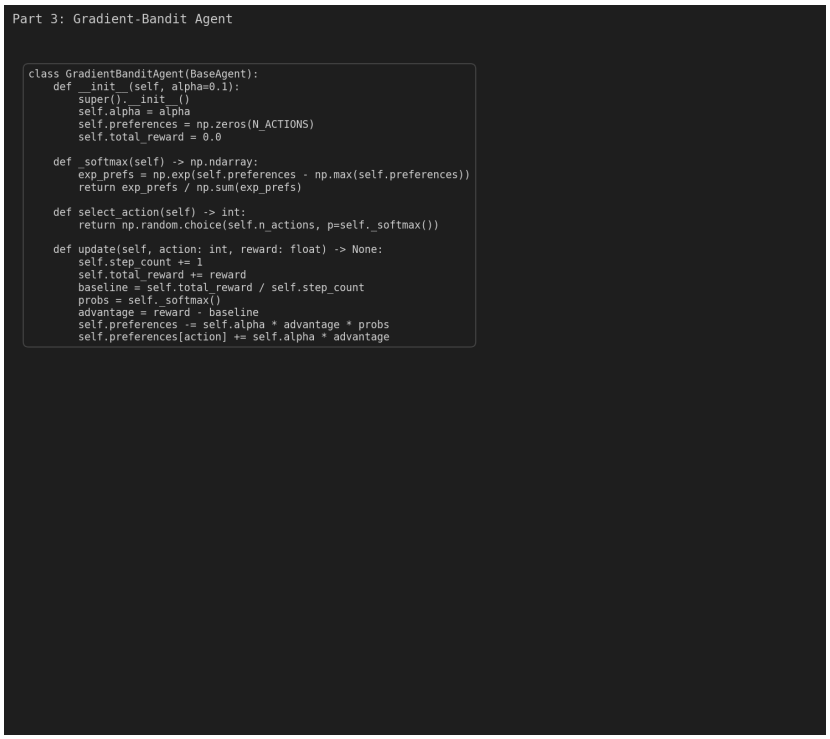
Q = [0, 0] (pessimistic/zero initialization) starts with the lowest performance. Both Q-values are well below the true values, so the agent must learn from scratch. With $\alpha = 0.1$, convergence is slow: the Q-estimates only gradually approach the true values. However, by step 1000, all three initializations converge to nearly identical final Q-values (~7.6-7.7 for a^1 and ~11.0 for a^2), demonstrating that the constant learning rate $\alpha = 0.1$ provides sufficient adaptability for all initializations to eventually find accurate estimates.

The key takeaway is that good initialization dramatically affects *transient* performance but has diminishing impact on *asymptotic* performance. Optimistic initialization is valuable because it encourages systematic exploration without requiring prior knowledge of the true Q-values.

Part 3: Gradient-Bandit Policy

The gradient-bandit method uses preference values H(a) instead of Q-value estimates. Actions are selected via a softmax distribution over preferences, and updates use the reward baseline (running average) to adjust preferences in the direction of better-than-average actions. Here $\alpha = 0.1$ with $H_1(a^1) = H_1(a^2) = 0$.

Key Code: Gradient-Bandit Agent



Results

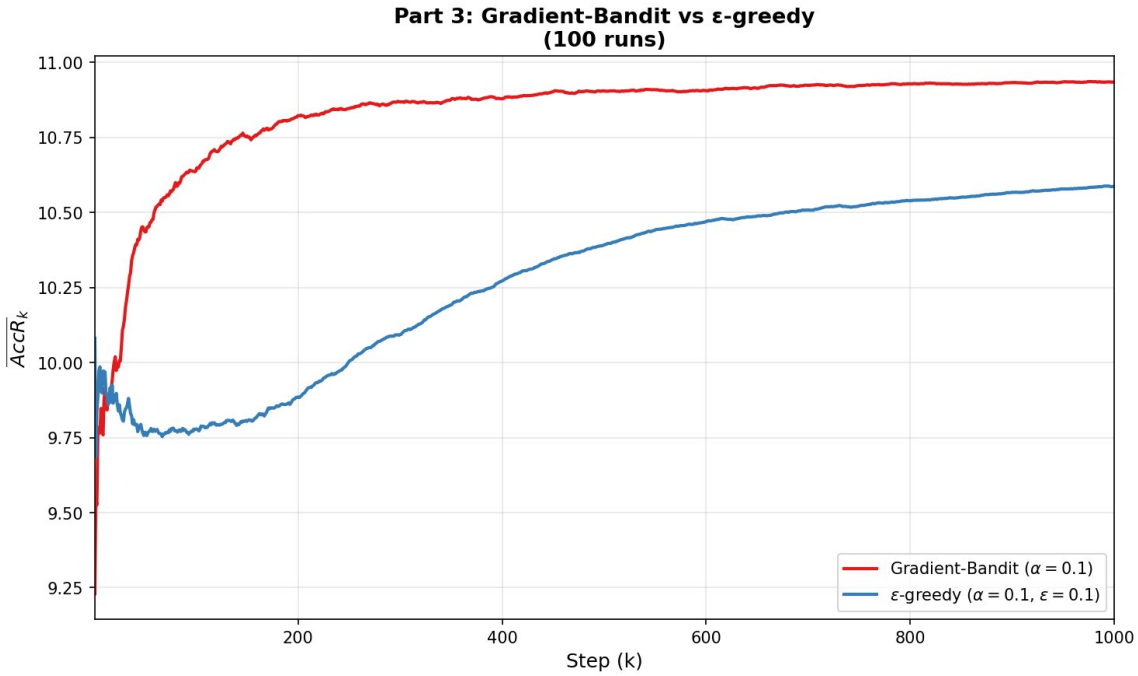


Figure 3: Gradient-Bandit (α = 0.1) vs. ε-greedy (α = 0.1, ε = 0.1).

Method	Final Avg. Accumulated Reward
Gradient-Bandit (α = 0.1)	10.9341
ε-greedy (α = 0.1, ε = 0.1)	10.5874

Interpretation

The gradient-bandit method significantly outperforms ε-greedy with matching α = 0.1. Its final average accumulated reward is 10.93 compared to 10.59 for ε-greedy, and the performance gap is most pronounced in the first 200-300 steps.

The gradient-bandit learns faster because it uses a fundamentally different mechanism: instead of maintaining Q-value estimates and making hard greedy/random decisions, it maintains a probability distribution over actions via the softmax function. This allows for *smooth* adjustment of action probabilities. When lever 2 yields above-average rewards, its preference increases and its selection probability rises continuously, whereas ε-greedy requires the Q-value of lever 2 to exceed that of lever 1 before exploitation kicks in.

Additionally, the reward baseline (running mean) in the gradient-bandit update rule ensures that the agent adapts relative to its own performance history. Even in the early steps when both actions yield similar-looking rewards, the gradient update pushes probability mass toward the action that tends to exceed the baseline, enabling faster identification of the optimal arm.

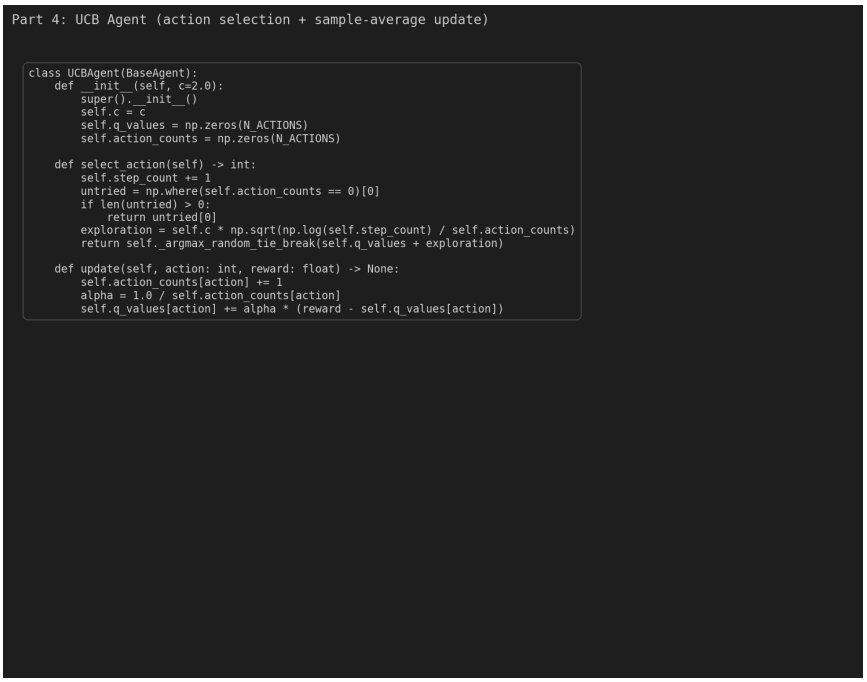
The ε-greedy method, with a constant α = 0.1, converges more slowly because 10% of its actions are random regardless of how confident the agent is, and Q-value convergence with a

constant learning rate is gradual.

Part 4: Upper Confidence Bound (UCB)

The UCB method selects actions by adding an exploration bonus c·sqrt(ln(k) / N(a)) to the Q-value estimate. This bonus decreases as an action is sampled more and increases logarithmically with the total number of steps, providing principled exploration. Q-values are updated using the sample-average rule.

Key Code: UCB Agent



UCB Performance (c = 2, 5, 100)

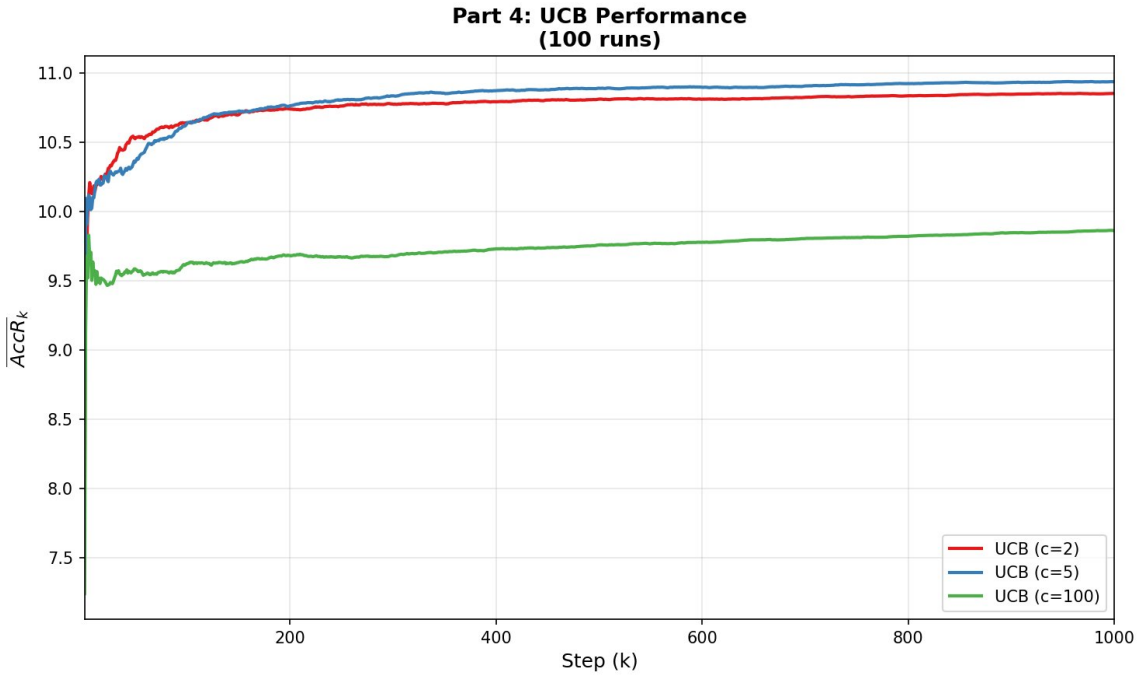


Figure 4: UCB performance for different exploration parameters c.

UCB Config	Final Avg. Accumulated Reward
UCB (c = 2)	10.8517
UCB (c = 5)	10.9381
UCB (c = 100)	9.8619

UCB with **c = 5** achieves the best performance (10.94), slightly outperforming c = 2 (10.85). The exploration bonus with c = 5 is large enough to ensure both arms are sampled sufficiently early on, allowing the agent to quickly identify lever 2 as optimal. UCB with **c = 100** dramatically underperforms (9.86) because the exploration bonus dominates the Q-value estimate for most of the 1000 steps, effectively turning the agent into a near-random policy. This demonstrates that excessive exploration is costly: the agent spends too much time pulling the suboptimal lever 1 and never fully exploits its knowledge of lever 2's superiority.

Comparison of Best Methods

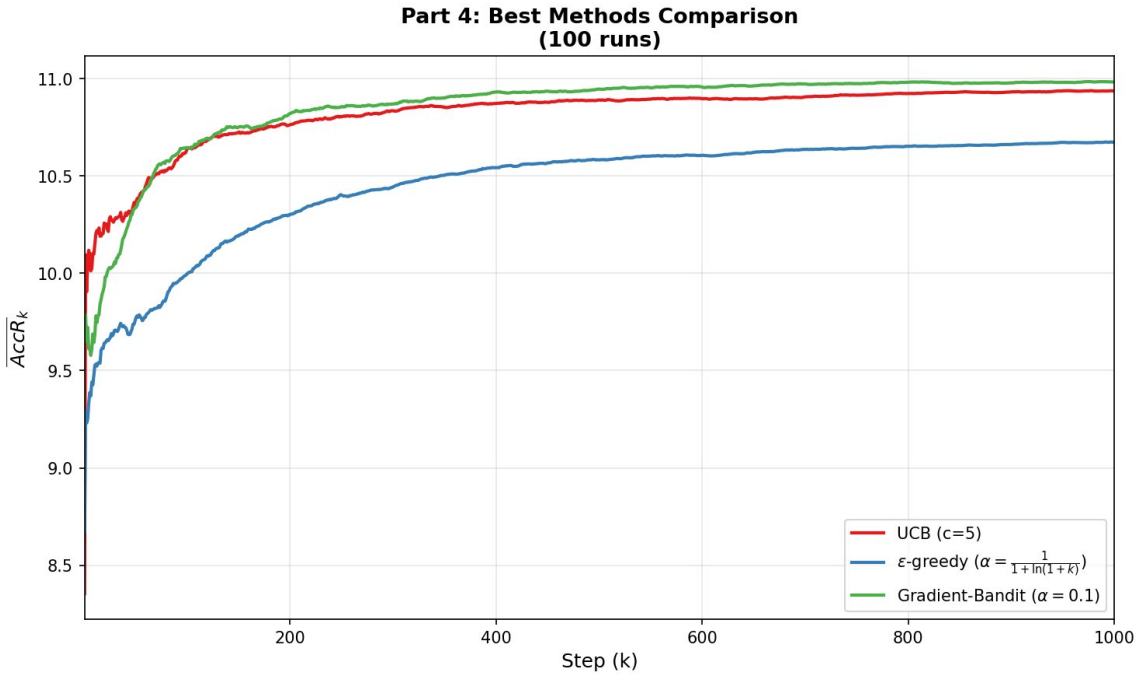


Figure 5: Best UCB (c = 5) vs. best ε-greedy (α = 1/(1+ln(1+k)), ε = 0.1) vs. Gradient-Bandit (α = 0.1).

Method	Final Avg. Accumulated Reward
Gradient-Bandit (α = 0.1)	10.9824
UCB (c = 5)	10.9381
Best ε-greedy (α = 1/(1+ln(1+k)), ε = 0.1)	10.6734

Interpretation

The **Gradient-Bandit** achieves the highest final accumulated reward (10.98), followed closely by **UCB (c = 5)** at 10.94, while the best **ε-greedy** configuration trails at 10.67. Looking at the convergence curves, UCB and Gradient-Bandit converge similarly fast in the early steps (both reach ~10.7 by step 200), while ε-greedy lags significantly during this period.

The Gradient-Bandit's advantage stems from its softmax-based action selection, which smoothly allocates more probability to the better arm as evidence accumulates. UCB achieves comparable asymptotic performance through its principled exploration bonus, which naturally decays as actions are sampled, transitioning from exploration to exploitation automatically. The ε-greedy method is inherently limited because its 10% random exploration never decreases, meaning ~5% of actions are wasted on the suboptimal lever even late in the experiment.

In summary, for this 2-armed bandit problem, both Gradient-Bandit and UCB provide effective strategies that outperform ε-greedy. The Gradient-Bandit offers the best overall performance, while UCB provides a principled exploration-exploitation tradeoff with interpretable hyperparameters. The ε-greedy approach, while simple, requires careful tuning of both ε and the learning rate schedule and still underperforms the more sophisticated methods.