Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Solution for Homework 6:

Multi-Armed Bandits

Designed By:

Mohammad Mohammadi
mohammadm97i@gmail.com

Arshia Gharooni
arshiyagharoony@gmail.com

Spring 2025

Contents

1 Task 1: Oracle Agent

1.1 Oracle Reward Calculation (Cell 13)

1.1.1 Task Overview

What to do: Compute the maximum expected reward given privileged information (true reward probabilities).

Why: The Oracle Agent knows all true probabilities and will always pull the arm with the highest reward probability:

$$\text{oracleReward} = \max_i p_i$$

This provides a performance upper bound for all other agents.

1.2 Short Questions

Q: What insight does the oracle reward give us about the best possible performance?
A: It sets an upper bound for performance. No realistic agent can exceed this bound, so it helps measure the regret of other algorithms.

Q: Why is the oracle considered "cheating" in a practical sense?
A: Because it has access to privileged information (i.e., the true probabilities), which is not available in real-world scenarios.

2 Task 2: Random Agent (RndAg)

2.1 Random Agent – `get_action` (Cell 15)

2.1.1 Task Overview

What to do: Implement action selection via:

$$\text{action} \sim \text{Uniform}(\{0, 1, \ldots, N-1\})$$

Why: This agent acts as a baseline by selecting arms uniformly at random, with no learning or memory.

2.2 Plotting Reward Curve Average Arm Reward (Cell 18)

2.2.1 Task Overview

What to do: Plot the agent's average reward over time. Add a constant horizontal line for the mean of all arms:

$$\text{AvgArmReward} = \frac{1}{N}\sum_{i=1}^{N} p_i$$

Why: Comparing the agent's performance to this baseline reveals whether it is learning anything useful.

2.3 Short Questions

Q: Why is the reward of the random agent generally lower and highly variable?
A: Because it selects arms without learning or optimizing, so its performance fluctuates with chance and it often misses the high-reward arms.

Q: How might you improve a random agent without using any learning mechanism?
A: You could incorporate a simple heuristic (e.g., use prior knowledge about arm distributions) to bias the random selection, though it still wouldn't learn from outcomes.

3 Task 3: Explore-First Agent (ExpFstAg)

3.1 Explore-First Agent – `update_Q` and `get_action` (Cell 21)

3.1.1 Task Overview

(a) `update_Q`

What to do: Use incremental update to track average reward for an arm:

$$Q(a) \leftarrow Q(a) + \frac{1}{n(a)}(r - Q(a))$$

Why: This allows us to update value estimates efficiently without storing entire reward histories.

(b) `get_action`

What to do: Choose a random arm during exploration, then pick:

$$a^* = \arg\max_a Q(a)$$

Why: The agent first samples arms to estimate their value, then commits to the one it believes is best.

3.2 Varying `max_ex` for Explore-First (Cell 21)

3.2.1 Task Overview

What to do: Run experiments with `max_explore` = {5, 10, 50, 100, 200}.

Why: The length of the exploration phase critically affects performance. This experiment visualizes the exploration-exploitation trade-off.

3.3 Short Questions

Q: Why might the early exploration phase (e.g., 5 steps) lead to high fluctuations in the reward curve?
A: With very few samples, the estimate of each arm's reward is noisy and unstable, causing high variance and fluctuating performance.

Q: What are the trade-offs of using a fixed exploration phase?
A: A fixed phase might not sample all arms adequately if the phase is too short, or it might waste time if too long, delaying exploitation of the best arm.

Q: How does increasing max_ex affect the convergence of the agent's performance?
A: Increasing max_ex gives the agent more time to learn the best arm, leading to a more accurate estimate, but can also delay exploitation and slow overall cumulative reward.

Q: In real-world scenarios, what challenges might arise in selecting the optimal exploration duration?
A: The optimal duration is problem-dependent and may vary over time; a fixed value might not adapt well to changes in the environment, suggesting the need for adaptive strategies.

4 Task 4: UCB Agent (UCB_Ag)

4.1 UCB Agent (Cell 29)

4.1.1 Task Overview

(a) `update_Q`

What to do: Same incremental update as before.

(b) `get_action`

What to do: Use the UCB formula:

$$Q(a) + \sqrt{\frac{2\log(t+1)}{n(a)+\delta}}$$

Why: UCB balances exploration and exploitation by adding a confidence bonus that shrinks over time.

4.2 UCB vs. Explore-First (max_ex=5) (Cell 33)

4.2.1 Task Overview

What to do: Run and plot both agents under same conditions.

Why: To compare adaptive exploration (UCB) versus fixed-phase exploration.

4.3 UCB vs. Explore-First (max_ex=20) (Cell 35)

4.3.1 Task Overview

What to do: Increase `max_ex` to 20 and repeat comparison.

Why: To test whether more exploration improves greedy agent's performance.

4.4 Short Questions

Q: Why does UCB learn slowly (even after 500 steps, not reaching maximum reward)?
A: UCB algorithm can learn slowly because it heavily explores uncertain actions, especially early on. It balances estimated rewards with confidence bounds, which may cause prolonged sampling of suboptimal arms in environments with many similar options or high reward variance. This slows convergence and delays optimal performance, even after many steps.

Q: Under what conditions might an explore-first strategy outperform UCB, despite UCB's theoretical optimality?
A: In finite time horizons or when exploration duration is well-tuned, explore-first can quickly latch onto a high-reward arm, whereas UCB's cautious approach might slow down overall reward accumulation.

Q: How do the design choices of each algorithm affect their performance in short-term versus long-term scenarios?
A: Explore-first may perform better short-term if the exploration phase is adequate, while UCB is designed to be asymptotically optimal but might lag in early performance.

Q: What impact does increasing the exploration phase to 20 steps have compared to 5 steps?
A: It allows more accurate estimation of arm rewards, reducing early variability, but delays exploitation which might lower cumulative reward if exploration goes on too long.

Q: How can you determine the optimal balance between exploration and exploitation in practice?
A: By conducting experiments, using cross-validation, or employing adaptive methods that adjust the exploration rate based on real-time performance metrics.

Q: We know that UCB is optimal. Why might ExpFstAg perform better in practice?
A: UCB is provably optimal in the asymptotic sense, its conservative exploration strategy can lead to suboptimal performance in finite-sample settings. ExpFstAg, by contrast, may employ a more aggressive or adaptive exploration mechanism that quickly narrows down promising actions, thereby reducing wasted trials on clearly inferior options. This often results in lower empirical regret on practical time scales, as the algorithm efficiently balances exploration and exploitation in real-world scenarios where the theoretical worst-case guarantees of UCB are less relevant.

5 Task 5: Epsilon-Greedy Agent (EpsGdAg)

5.1 Epsilon-Greedy Agent `get_action` (Cell 41)

5.1.1 Task Overview

What to do: Sample a random action with probability ε, otherwise:

$$a^* = \arg\max_a Q(a)$$

Why: This strategy introduces simple, tunable stochastic exploration.

5.2 Epsilon-Greedy Experiments) (Cell 43)

5.2.1 Task Overview

What to do: Run agent with $\varepsilon \in \{0, 0.1, 0.2, 0.4\}$.

Why: To show how increasing exploration probability affects performance and convergence.

5.3 Short Questions

Q: Why does a high value result in lower immediate rewards?
A: Because a higher means more random (exploratory) actions, which can result in selecting suboptimal arms more frequently.

Q: What benefits might there be in decaying over time?
A: Decaying allows the agent to explore early on when uncertainty is high and gradually shift towards exploitation as the agent's estimates become more reliable.

Q: How do the reward curves for different values reflect the exploration–exploitation balance?
A: Lower values show more stable, higher rewards as the agent exploits learned knowledge, whereas higher values exhibit more variability and lower cumulative rewards due to excessive exploration.

Q: Under what circumstances might you choose a higher despite lower average reward?
A: In highly dynamic environments where constant adaptation is needed, a higher ensures continuous exploration to adjust to changes.

6 Task 6: LinUCB Agent (Contextual Bandits)

6.1 Loading Contextual Dataset (Cell 46)

6.1.1 Task Overview

What to do: Load `dataset.txt` as a NumPy array of shape $[10000, 102]$.

Why: The first two columns represent action and reward, while the rest represent context features for LinUCB.

6.2 LinUCB Agent (Cell 49)

6.2.1 Task Overview

(a) `get_ucb`

What to do: Compute predicted reward:

$$\hat{r}_a = \theta_a^\top x + \alpha\sqrt{x^\top A_a^{-1} x}$$

where $\theta_a = A_a^{-1} b_a$

Why: This incorporates both estimated reward and uncertainty.

(b) `update_params`

What to do:

$$A_a \leftarrow A_a + xx^\top, \quad b_a \leftarrow b_a + rx$$

Why: This is recursive least squares for updating arm parameters.

(c) `get_action`

What to do: Choose arm with highest UCB estimate.

Why: Follows same principle as UCB but with linear contextual modeling.

6.3 LinUCB Sweep (Cell 52)

6.3.1 Task Overview

What to do: Run with $\alpha \in \{0, 0.01, 0.1, 0.5\}$.

Why: Larger α encourages more exploration. Helps visualize sensitivity of LinUCB to exploration scale.

6.4 Compare Bandit and Contextual UCB (Cell 55)

6.4.1 Task Overview

What to do: Plot aligned CTR for bandit-UCB and LinUCB.

Why: To test whether using context actually improves decision-making. LinUCB should outperform UCB if context is informative.

6.5 Short Questions

Q: How does LinUCB leverage context to outperform classical bandit algorithms?
A: LinUCB uses the context features to estimate expected rewards with a linear model, which can provide more tailored decisions when context correlates with reward.

Q: What is the role of the α parameter in LinUCB, and how does it affect the exploration bonus?
A: The α parameter scales the confidence interval; a higher α encourages more exploration by increasing the bonus, while a lower α emphasizes exploitation based on current estimates.

Q: What does α affect in LinUCB?
A: α directly scales the uncertainty term in the upper confidence bound, which modulates the exploration-exploitation trade-off. A larger α enlarges the confidence interval, pushing the algorithm to explore actions with high uncertainty—even if their estimated rewards are lower—while a smaller α tightens the interval, leading to a focus on exploiting actions with high current estimates. This parameter is critical in

determining how aggressively the algorithm searches for potentially better actions versus relying on known performance.

Q: Do the reward curves change with α? Explain why or why not.
A: The effect of α on reward curves depends on its role in the algorithm. If α directly scales the reward signal, then the curves will shift accordingly since the absolute magnitude of rewards changes. However, if α only adjusts the learning dynamics—say, by modulating the update rate without altering the inherent reward values—the shape of the reward curve remains the same, although the speed of convergence or the variance along the curve might differ. In essence, whether the reward curves change with α hinges on whether α modifies the reward structure or simply influences how quickly the algorithm learns from the same rewards.

Q: Based on your experiments, does LinUCB outperform the standard UCB algorithm? Why or why not?
A: Answers may vary, but a typical answer is that LinUCB can outperform standard UCB when the context is informative because it personalizes decisions; however, if the context is noisy, its advantage may diminish.

Q: What are the key limitations of each algorithm, and how would you choose between them for a given application?
A: UCB is simple and parameter-free but may converge slowly, while LinUCB requires context and proper tuning of α. The choice depends on the availability of side information and the time horizon of the problem.

7 Task 7: Final Deep-Dive Questions

- **Finite-Horizon Regret and Asymptotic Guarantees**

  Many algorithms (e.g., UCB) are analyzed using asymptotic (long-term) regret bounds. In a finite-horizon scenario (say, 500–1000 steps), explain intuitively why an algorithm that is asymptotically optimal may still yield poor performance. What trade-offs arise between aggressive early exploration and cautious long-term learning? Deep Dive: Discuss how the exploration bonus, tuned for asymptotic behavior, might delay exploitation in finite time, leading to high early regret despite eventual convergence.

  A: Consider an algorithm like UCB, which is proven to be asymptotically optimal. Its performance guarantee focuses on long-term behavior—as the number of steps grows, the algorithm eventually identifies the optimal arm and minimizes regret. However, in a finite-horizon scenario (e.g., 500–1000 steps), the following issues arise:

  1. **Exploration Bonus and Its Role:**
     UCB-type algorithms add an exploration bonus to the estimated reward of each arm. This bonus is typically of the form:
     $$\sqrt{\frac{\log(n)}{N(i)}},$$
     where n is the current time step and $N(i)$ is the number of times arm i has been pulled. Initially, when $N(i)$ is small, this bonus is large.

2. **Delayed Exploitation in Finite Time:**
   Intuition: Early in the process, every arm—including suboptimal ones—receives a significant exploration bonus. This encourages balanced exploration across all arms.
   Consequence: The algorithm delays committing to the best arm. Despite eventually identifying the optimal arm, the early-stage regret is high due to excessive exploration.
3. **Trade-Offs Between Exploration and Exploitation:**
   - **Aggressive Early Exploration:**
     * *Pros:* Ensures robust knowledge about all arms, avoiding premature convergence to suboptimal choices.
     * *Cons:* In finite-horizon settings, too much exploration leads to high early regret and reduces overall reward.
   - **Cautious Long-Term Learning:**
     * *Pros:* Guarantees long-term optimality by thoroughly evaluating each arm.
     * *Cons:* Inefficient in short horizons where rapid exploitation of the best arm is more valuable.
4. **Finite-Horizon Impact:**
   The exploration bonus in UCB is tuned for asymptotic optimality—it decreases slowly due to the $\log(n)$ factor. Over a short horizon:
   - *High early regret:* Suboptimal arms are explored too often.
   - *Lower cumulative reward:* The algorithm delays switching to full exploitation.

- **Hyperparameter Sensitivity and Exploration–Exploitation Balance**

  Consider the impact of hyperparameters such as ϵ in ϵ-greedy, the exploration constant in UCB, and the α parameter in LinUCB. Explain intuitively how slight mismatches in these parameters can lead to either under-exploration (missing the best arm) or over-exploration (wasting pulls on suboptimal arms). How would you design a self-adaptive mechanism to balance this trade-off in practice? Deep Dive: Provide insight into the "fragility" of these parameters in finite runs and how a meta-algorithm might monitor performance indicators (e.g., variance in rewards) to adjust its exploration dynamically.

  A: Slight mismatches in hyperparameters can have a dramatic impact on the exploration–exploitation trade-off. For example, in ε-greedy, if ε is set too low, the algorithm rarely explores and may quickly commit to an arm that appears promising early on—even if it isn't the best—resulting in under-exploration. Conversely, if ε is too high, the algorithm wastes many pulls on arms that are unlikely to be optimal, leading to over-exploration. A similar phenomenon occurs with the exploration constant in UCB: setting it too low narrows the confidence intervals too much, discouraging exploration of uncertain arms, while an overly high constant inflates the uncertainty bonus, causing excessive sampling of arms with high variance but low mean rewards. LinUCB's α parameter plays an analogous role when working with linear models; if α is mismatched, it can either overstate or understate the uncertainty in the predictions, leading again to either missing the best arm or over-investing in suboptimal ones.
  This fragility is especially pronounced in finite runs where the initial sampling decisions have outsized effects on the estimates of each arm's potential. Early misallocations can cause the algorithm to lock in on a suboptimal arm if it doesn't explore enough, or waste valuable pulls if it over-explores. To mitigate this, one can design a self-adaptive mechanism that dynamically adjusts these parameters based on observed performance indicators. For instance, a meta-algorithm might continuously monitor the variance in rewards and the convergence of estimated means. If high variance persists or

the rate of improvement in cumulative rewards slows down, the system could increase the exploration parameter (ε, the UCB constant, or α) to gather more information about under-sampled arms. On the other hand, if the variance is low and the estimates have stabilized, the mechanism could reduce the exploration bonus to focus more on exploitation.
This adaptive tuning could be implemented via a feedback loop where the exploration parameter is updated using a rule inspired by gradient descent or control theory—adjusting proportionally to the discrepancy between the current performance and a target metric (like a desired rate of reward improvement). Alternatively, a Bayesian approach might maintain a posterior over the optimal hyperparameter value, updating it as new data arrives to select the most promising exploration parameter for the current context.

- **Context Incorporation and Overfitting in LinUCB**

  LinUCB uses context features to estimate arm rewards, assuming a linear relation. Intuitively, why might this linear assumption hurt performance when the true relationship is complex or when the context is high-dimensional and noisy? Under what conditions can adding context lead to worse performance than classical (context-free) UCB? Deep Dive: Discuss the risk of overfitting to noisy or irrelevant features, the curse of dimensionality, and possible mitigation strategies (e.g., dimensionality reduction or regularization).

  A: In LinUCB, the algorithm estimates the expected reward of each arm as a linear function of the observed context, relying on the assumption that this relationship is indeed linear. However, when the true reward function is complex or nonlinear, this assumption introduces a model mismatch. The linear approximation may fail to capture key interactions or dependencies in the data, leading to biased or systematically inaccurate reward estimates. This, in turn, results in suboptimal arm selection and a degradation in overall performance.
  Moreover, in high-dimensional context spaces, the risk of overfitting becomes prominent. If many of the context features are irrelevant or noisy, the algorithm may learn spurious correlations that do not generalize well. This is particularly harmful during early rounds of exploration when data is scarce and uncertainty estimates are less reliable. The curse of dimensionality exacerbates this problem—more features mean more parameters to estimate, increasing variance and requiring significantly more data to achieve stable estimates. As a result, the confidence bounds used by LinUCB can become misleading, either overly tight (leading to premature exploitation) or overly wide (resulting in excessive exploration).
  Under such conditions, adding context can lead to worse performance than context-free UCB, which simply estimates mean rewards for each arm without regard to context. Classical UCB avoids the pitfalls of modeling irrelevant features and performs robustly when the environment is stationary or when context does not add meaningful predictive power. Thus, unless context features are informative and the model is sufficiently regularized, LinUCB may suffer from degraded performance compared to its simpler counterpart.
  To mitigate these issues, dimensionality reduction techniques such as PCA or random projection can be employed to reduce the noise and retain the most relevant aspects of the context. Regularization methods like ridge regression (L2) or lasso (L1) can control the model complexity, helping to prevent overfitting and improve generalization. Additionally, techniques like feature selection or embedding methods can help focus the model on the most relevant dimensions, thereby enhancing the robustness and performance of LinUCB in complex or noisy environments.

- **Adaptive Strategy Selection**

Imagine designing a hybrid bandit agent that can switch between an explore-first strategy and UCB based on observed performance. What signals (e.g., variance of reward estimates, stabilization of Q-values, or sudden drops in reward) might indicate that a switch is warranted? Provide an intuitive justification for how and why such a meta-strategy might outperform either strategy alone in a finite-time setting. Deep Dive: Explain the challenges in detecting when exploration is "enough" and how early exploitation might capture transient improvements even if the long-term guarantee favors UCB.

A: A hybrid agent that dynamically switches between an explore-first phase and a UCB strategy seeks to capitalize on the complementary strengths of both approaches. In the early stages of learning, an explore-first strategy aggressively samples all available options to build a robust initial understanding of their reward distributions. This initial phase can rely on signals such as high variance in the reward estimates and rapidly changing Q-values, which indicate that the agent's knowledge is still in flux. When the observed estimates begin to stabilize—that is, when the variance decreases and successive Q-values show only marginal differences—the agent can infer that it has accumulated sufficient data about the arms. At that point, switching to a UCB strategy becomes appealing because UCB is designed to balance exploitation and exploration using confidence bounds; it can then more precisely allocate trials to the arms with the best potential while still accounting for residual uncertainty.
One intuitive signal for such a switch is the stabilization of the Q-values. Early on, the estimates might oscillate or change markedly as new samples are collected, reflecting uncertainty and insufficient data. As the estimates converge, it suggests that the exploration has been effective, and further pure exploration is unlikely to yield significantly different information. Similarly, a drop in the variance of reward estimates is a strong indicator that the underlying reward distributions have been sufficiently sampled. A sudden drop in the average reward, however, could also signal that the environment might be shifting or that a previously overestimated arm is revealing its true performance. Such drops might necessitate a temporary return to exploration to recalibrate the estimates before committing again to exploitation.
In a finite-time setting, neither a purely exploratory approach nor a strictly UCB-based method is optimal on its own. An exploration-first strategy ensures that the agent does not prematurely commit to a suboptimal arm due to early noise. On the other hand, UCB's long-term guarantees stem from its balanced approach, but in the short run, its cautious nature can delay the exploitation of arms that show promising early performance. By adopting a meta-strategy that monitors for stabilization signals—such as the convergence of Q-values and reduced variance in reward estimates—the hybrid agent can switch at a moment when it is statistically confident about its estimates. This early switch to exploitation enables the agent to secure higher rewards during the finite time horizon, effectively capturing transient improvements that a pure UCB algorithm might miss due to its inherent exploration bias.
Detecting when exploration is "enough" poses its own challenges. The inherent randomness in reward signals means that the estimates of an arm's value may fluctuate due to noise rather than genuine differences in performance. Setting thresholds for stabilization is nontrivial because the agent must distinguish between temporary fluctuations and true convergence. Moreover, a transient improvement in an arm's performance might be significant in a finite-time scenario, even if long-term guarantees favor a more cautious UCB approach. This means that the hybrid strategy must be designed to recognize and capitalize on such transient improvements by allowing an early shift to exploitation when evidence suggests that further exploration is unlikely to yield better long-term estimates. In essence, while long-term guarantees favor UCB for its systematic exploration, the hybrid strategy benefits from an early exploitation phase that locks in gains based on high-confidence estimates, thereby outperforming either strategy applied in isolation when operating

under time constraints.

- **Non-Stationarity and Forgetting Mechanisms**

  In non-stationary environments where reward probabilities drift or change abruptly, standard bandit algorithms struggle because they assume stationarity. Intuitively, explain how and why a "forgetting" or discounting mechanism might improve performance. What challenges arise in choosing the right decay rate, and how might it interact with the exploration bonus? Deep Dive: Describe the delicate balance between retaining useful historical information and quickly adapting to new trends, and the potential for "chasing noise" if the decay is too aggressive.

  A: In non-stationary environments, the core difficulty lies in the fact that past observations may no longer accurately reflect the current state of the world. Traditional bandit algorithms, like the UCB family, assume that reward probabilities remain constant over time. This assumption leads them to aggregate data uniformly from all past observations. When the underlying probabilities drift or change suddenly, this aggregation can mislead the algorithm because old data, which is no longer relevant, can dominate the statistical estimates. A forgetting or discounting mechanism mitigates this issue by progressively reducing the weight of older observations, thereby allowing more recent data—which is likely to be more indicative of the current conditions—to play a larger role in the estimation process.

  Mathematically, this is often achieved by applying an exponential decay to past rewards and counts. Consider a discount factor, γ (with $0 < \gamma < 1$); when an observation is made, its influence is recorded as γ^k at a later time step, where k represents the number of time steps since the observation was made. This exponential decay creates an effective sample size that is much smaller than the total number of observations, emphasizing recent events while diminishing the influence of older ones. In effect, the algorithm behaves as if it is looking at a sliding window of recent data, thus improving its responsiveness to changes in the environment.

  However, choosing the right decay rate is a delicate balancing act. If the decay rate is too aggressive—meaning γ is set too low—the algorithm quickly forgets useful historical information. This high sensitivity to recent data can lead to overreaction: the algorithm may interpret random fluctuations or noise as meaningful shifts in the reward structure, leading to erratic decision-making and a tendency to "chase noise." Such over-adaptation can undermine the algorithm's stability, resulting in excessive exploration of arms that appear promising only due to short-term variability.

  Conversely, if the decay is too conservative (i.e., γ is close to 1), the algorithm retains a significant amount of historical data, which might hinder its ability to adapt promptly to new trends. In this case, the estimates for the arms will continue to be influenced heavily by outdated information, causing the algorithm to lag in recognizing and reacting to genuine changes in the reward distribution. This trade-off highlights the inherent challenge: the algorithm must balance between sufficient inertia to dampen the impact of random noise and sufficient agility to quickly adapt when real changes occur.

  The interaction with the exploration bonus further complicates this balance. In many bandit algorithms, the exploration bonus is a function of the uncertainty in the estimates, often inversely related to the number of observations. With discounting in place, the effective count of observations decreases because older data is de-emphasized. This reduction inflates the estimated uncertainty, which in turn can lead to a larger exploration bonus. While a larger bonus encourages the exploration of less-sampled arms, it might inadvertently prompt the algorithm to over-explore arms that appear uncertain due to aggressive forgetting rather than genuine potential for high rewards. Thus, an improperly tuned decay rate can exacerbate the exploration–exploitation trade-off, leading to suboptimal performance.

- **Exploration Bonus Calibration in UCB**

  The UCB algorithm adds a bonus term that decreases with the number of times an arm is pulled. Intuitively, why might a "conservative" (i.e., high) bonus slow down learning—even if it guarantees asymptotic optimality? Under what circumstances might a less conservative bonus be beneficial, and what risks does it carry? Deep Dive: Analyze how a high bonus may force the algorithm to continue sampling even when an arm's estimated reward is clearly suboptimal, thereby delaying convergence. Conversely, discuss the risk of prematurely discarding an arm if the bonus is too low.

  A: The UCB algorithm estimates each arm's value as the sum of its empirical mean reward and an exploration bonus that shrinks as the arm is pulled more frequently. The bonus term is designed to capture the uncertainty about the true mean, ensuring that every arm is sampled enough to provide a reliable estimate. A "conservative" bonus means that the bonus term is relatively high, so even if an arm has shown poor performance, its upper confidence bound remains elevated for a longer period. This encourages the algorithm to continue sampling the arm, delaying its exclusion from consideration. The downside of this conservatism is that it forces the algorithm to invest additional pulls in arms that are already clearly suboptimal based on accumulated data, thereby slowing down the learning process. The algorithm accumulates extra regret by exploring arms unnecessarily even when there is strong evidence suggesting they are inferior.
  In contrast, a less conservative bonus—one that decays faster—reduces the amount of exploration allotted to each arm. In environments where the reward distributions are not excessively noisy or where the differences between arms are pronounced, a lower bonus can enable the algorithm to more quickly identify and commit to the optimal arm. However, this approach carries a significant risk: if the early observations from an arm are misleading due to high variance or statistical fluctuations, the reduced bonus might cause the algorithm to underestimate its potential. In such cases, the arm could be prematurely discarded, leading to a failure to explore sufficiently and possibly converging to a suboptimal arm.
  Thus, the choice of bonus reflects a trade-off. A high bonus guarantees that exploration continues until the uncertainty is adequately resolved, ensuring asymptotic optimality, but it may prolong the learning phase by over-exploring suboptimal options. Conversely, a lower bonus can accelerate convergence in well-behaved environments but at the risk of not exploring enough, which might cause the algorithm to miss out on better-performing arms due to early underestimation.

- **Exploration Phase Duration in Explore-First Strategies**

  In the Explore-First agent (ExpFstAg), how does the choice of a fixed exploration period (e.g., 5 vs. 20 steps) affect the regret and performance variability? Provide a scenario in which a short exploration phase might yield unexpectedly high regret, and another scenario where a longer phase might delay exploitation unnecessarily. Deep Dive: Discuss how the "optimal" exploration duration can depend heavily on the underlying reward distribution's variance and the gap between the best and other arms, and why a one-size-fits-all approach may not work in practice.

  A: The Explore-First agent (ExpFstAg) divides its operation into two distinct phases: an initial exploration phase, where each arm is played a predetermined number of times, and a subsequent exploitation phase, where the arm with the highest empirical mean is selected for all remaining plays. The fixed length of the exploration period is crucial because it directly influences both the accuracy

of the arm quality estimates and the cumulative regret incurred by the agent.
When the exploration period is short—for instance, 5 steps per arm—the agent gathers only a limited number of samples from each arm. With such a small sample size, the empirical estimates of the mean rewards are highly susceptible to random fluctuations, especially if the rewards exhibit high variance. In this situation, it is quite possible that, due to chance, a suboptimal arm may appear to have a higher estimated mean than the optimal arm. Once the exploration phase is over, the agent commits to the arm that appears best based on these unreliable estimates. As a result, if the chosen arm is in fact suboptimal, the regret accrued during the lengthy exploitation phase can be unexpectedly high. Moreover, the performance becomes highly variable because different runs of the algorithm might yield very different estimates based solely on the randomness inherent in the small number of observations.
Conversely, if the exploration period is extended to 20 steps, the agent collects more data from each arm. This larger sample size tends to average out the randomness in the rewards, leading to more accurate estimates of the underlying means and, therefore, a higher probability of correctly identifying the optimal arm. However, this improved estimation accuracy comes at a cost. During the extended exploration phase, the agent is not exploiting the best arm even if it could have been identified sooner. In scenarios where the gap between the optimal arm and the others is large and the reward distributions exhibit low variance, even a short exploration phase would have sufficed to reliably detect the best arm. In such cases, prolonging the exploration phase unnecessarily delays the commencement of the exploitation phase, causing the agent to incur additional regret simply because it spent extra time sampling suboptimal arms.
To illustrate, consider a scenario with two arms where both arms have high-variance reward distributions and the difference in their expected rewards is marginal. If the agent only explores for 5 steps per arm, the statistical noise might cause the suboptimal arm's estimate to be spuriously high, leading the agent to select it for the rest of the plays. This misallocation of plays results in a significant cumulative regret over time, as the benefits of the better arm are never reaped. On the other hand, imagine a scenario where the optimal arm has a substantially higher mean reward and very low variance compared to the other arms. Here, even a very brief exploration phase would likely reveal the superiority of the optimal arm. In this setting, extending the exploration phase to 20 steps delays the switch to exploitation, causing unnecessary regret because the agent continues to play arms that are clearly inferior when it could have been maximizing rewards almost immediately.
The key insight is that the "optimal" exploration duration is highly dependent on the statistical properties of the reward distributions. When rewards have high variance or when the difference (or gap) between the best arm and the others is small, more samples are necessary to confidently distinguish among the arms. In contrast, when the rewards are less noisy or the gap is wide, fewer samples are required to accurately identify the optimal arm. Therefore, a fixed, one-size-fits-all approach to setting the exploration period may not perform well across different scenarios. In practice, adaptive strategies that dynamically balance exploration and exploitation—by, for example, increasing exploration when uncertainty is high and decreasing it when the best arm becomes apparent—tend to offer superior performance.
Thus, the trade-off is clear: a shorter exploration phase risks high regret due to misidentification of the optimal arm and increased variability across runs, while a longer exploration phase can unnecessarily delay the exploitation of a clearly superior option, thereby incurring avoidable regret. This delicate balance is at the heart of multi-armed bandit problems, highlighting why understanding the underlying reward variance and the reward gap is essential for designing effective exploration strategies.

- **Bayesian vs. Frequentist Approaches in MAB** (4 points)

Compare the intuition behind Bayesian approaches (such as Thompson Sampling) to frequentist methods (like UCB) in handling uncertainty. Under what conditions might the Bayesian approach yield superior practical performance, and how do the underlying assumptions about prior knowledge influence the exploration–exploitation balance? Deep Dive: Explore the benefits of incorporating prior beliefs and the risk of bias if the prior is mis-specified, as well as how Bayesian updating naturally adjusts the exploration bonus as more data is collected.

A: The core difference between Bayesian methods like Thompson Sampling and frequentist approaches such as UCB lies in how each framework conceptualizes and manages uncertainty. In the Bayesian perspective, the uncertainty about the reward distributions is encapsulated in a probability distribution over the unknown parameters. Before any data is observed, the algorithm starts with a prior belief about these parameters. As new rewards are collected, Bayes' theorem is used to update this belief, resulting in a posterior distribution that reflects both the prior information and the empirical evidence. Thompson Sampling leverages this by randomly sampling a parameter set from the current posterior for each arm and then choosing the arm that appears best under that sample. This sampling mechanism inherently balances exploration and exploitation: when an arm has been sampled infrequently, its posterior is typically more spread out, increasing the chance that a sample might be unusually high and lead to exploration. As more data accumulates, the posterior concentrates around the true parameter value, naturally reducing the need for exploration because the uncertainty diminishes.
In contrast, the frequentist method such as the Upper Confidence Bound (UCB) algorithm does not incorporate prior beliefs. Instead, UCB computes an empirical estimate of the mean reward for each arm along with a confidence interval that reflects the uncertainty due to the limited number of observations. The algorithm then selects the arm with the highest upper bound on this interval, an approach often described as "optimism in the face of uncertainty." The exploration bonus in UCB is mathematically determined, typically being inversely proportional to the number of times an arm has been played. Initially, when little data is available, the bonus is large, promoting exploration. As more data is gathered and the statistical estimates become more reliable, the confidence intervals shrink, reducing the exploration bonus and shifting the algorithm's focus toward exploitation.
The practical performance of Bayesian approaches can surpass that of frequentist methods under certain conditions. When reliable prior information is available, incorporating this knowledge can significantly accelerate learning by guiding the initial exploration towards more promising areas of the parameter space. In such cases, the Bayesian method's ability to integrate prior beliefs results in more informed decision-making, which is particularly advantageous in environments where data is sparse or costly to obtain. Moreover, Bayesian updating naturally adjusts the exploration bonus as more data is collected. The variability of the posterior distribution serves as an implicit measure of uncertainty, so as the number of observations increases, the algorithm automatically reduces its exploratory behavior without the need for an externally imposed exploration term.
However, this strength can also be a weakness if the prior is mis-specified. A poorly chosen prior can introduce bias, leading the algorithm to overestimate or underestimate the quality of certain actions. If the initial belief is too confident in the wrong region of the parameter space, the algorithm might unduly favor suboptimal arms, delaying convergence to the optimal solution. Although a sufficient amount of data can eventually overwhelm a mis-specified prior, the transient performance might suffer, especially in early stages or in environments where fast adaptation is critical.
Overall, Bayesian methods and frequentist approaches offer fundamentally different ways of handling uncertainty. Bayesian methods offer a flexible and principled way to incorporate domain knowledge and automatically adjust the exploration–exploitation balance as data accrues, which can lead to superior performance when the prior is well-informed. Frequentist methods, by relying on confidence

bounds derived solely from observed data, provide a more straightforward mechanism that avoids the risk of prior mis-specification but may require careful tuning of the exploration bonus. The choice between these methods often depends on the availability and reliability of prior information and the specific requirements of the application at hand.

- **Impact of Skewed Reward Distributions** (3.75 points)

  In environments where one arm is significantly better (skewed probabilities), explain intuitively why agents like UCB or ExpFstAg might still struggle to consistently identify and exploit that arm. What role does variance play in these algorithms, and how might the skew exacerbate errors in reward estimation? Deep Dive: Discuss how the variability of rare but high rewards can mislead the agent's estimates and cause prolonged exploration of suboptimal arms.

  A: In a multi-armed bandit setting, algorithms like UCB (Upper Confidence Bound) and ExpFstAg aim to balance exploration and exploitation by maintaining estimates of each arm's reward and augmenting these estimates with a measure of uncertainty. The key challenge in environments with skewed probabilities is that the true "better" arm might have a reward distribution with high variance or heavy tails, while some suboptimal arms might occasionally yield exceptionally high rewards. This disparity in reward variability can mislead these algorithms in several ways.
  At the core of these algorithms is the estimation of the expected reward for each arm, usually computed as the sample mean of observed rewards. In a low-variance setting, the sample mean converges rapidly to the true expected value as more samples are collected. However, in a skewed environment, even if one arm has a higher overall expected reward, its observations may be mostly moderate with a few extremely high values that occur rarely. This leads to two problematic scenarios:
  First, consider a suboptimal arm with lower expected reward but with a chance of producing a very high reward. When such a rare event occurs, it can inflate the empirical mean of that arm. Since UCB-type algorithms add an exploration bonus based on the uncertainty (which, in turn, is related to the variance and number of pulls), this inflated mean, combined with a possibly still-large confidence bound, might temporarily convince the algorithm that the suboptimal arm is promising. The agent then allocates additional pulls to that arm in order to reduce its uncertainty. This prolonged exploration delays the process of identifying the truly optimal arm.
  Second, the optimal arm, despite having a higher overall expected reward, may deliver a series of moderate outcomes. In the short term, these outcomes might not seem as impressive compared to the occasional outlier from a suboptimal arm. Since the confidence intervals in UCB depend on both the number of observations and the variance of the rewards, a high variance forces the interval to remain wide for longer periods. This extended uncertainty implies that the algorithm requires many more samples to confidently distinguish the optimal arm's mean from those of the other arms.
  The role of variance here is twofold. High variance in the reward distributions results in greater uncertainty in the empirical estimates, which is reflected in wider confidence bounds. This means that even if an arm is truly better, its estimated performance may fluctuate significantly from one batch of observations to the next. Additionally, for algorithms that weight rewards exponentially (as in ExpFstAg), the presence of an occasional extreme reward can disproportionately affect the arm's estimated quality. The exponential weighting mechanism amplifies differences; hence, a single high reward from a suboptimal arm might cause the algorithm to overestimate its long-term potential.
  Overall, the skew exacerbates errors in reward estimation because the inherent variability makes it challenging for these algorithms to distinguish between arms based solely on empirical averages and confidence intervals. The variability due to rare but high rewards can mislead the agent by temporarily boosting the perceived performance of an arm that is not consistently optimal. This

misrepresentation leads to continued exploration of the suboptimal arm, delaying the convergence to a strategy that would otherwise consistently exploit the best arm. In summary, the high variance increases the probability of significant estimation errors, and when these errors occur in skewed environments, they can dominate the decision-making process, forcing the agent to spend more time resolving uncertainty rather than reliably exploiting the optimal arm.

- **Designing for High-Dimensional, Sparse Contexts** (5 points)

  In contextual bandits where the context is high-dimensional but only a few features are informative, what are the intuitive challenges that arise in using a linear model like LinUCB? How might techniques such as feature selection, regularization, or non-linear function approximation help, and what are the trade-offs involved? Deep Dive: Provide insights into the risks of overfitting versus underfitting, the increased variance in estimates from high-dimensional spaces, and the potential computational costs versus performance gains when moving from a simple linear model to a more complex one.

  A: When using a linear model like LinUCB in a high-dimensional contextual bandit setting, one must confront several intertwined challenges. Fundamentally, when only a few features are genuinely informative but the context is high-dimensional, the model faces the curse of dimensionality. In such cases, the estimation problem becomes statistically inefficient because the model is forced to learn a large number of parameters from limited data. This situation results in high variance in the parameter estimates, which in turn amplifies the risk of overfitting. Overfitting here means that while the model might closely capture the noise present in the training data (or historical rewards), it will likely generalize poorly to new contexts, leading to suboptimal decisions during the exploration–exploitation trade-off that is central to bandit problems.
  To mitigate these issues, techniques like feature selection become crucial. By identifying and retaining only those features that are truly informative, one can dramatically reduce the dimensionality of the problem. This not only reduces the variance of the estimates but also makes the model more robust in its predictions. However, an overly aggressive feature selection strategy might inadvertently discard features that, although weak on their own, could be important when combined with others. This risk of underfitting occurs when the model becomes too simplistic to capture the underlying reward structure.
  Regularization techniques, such as L1 (Lasso) or L2 (Ridge) regularization, offer another avenue for improvement. By adding a penalty for large coefficient values, regularization effectively shrinks the influence of noisy, non-informative features. This can help strike a balance between bias and variance: reducing variance by dampening the impact of less important features while still allowing the model to learn from the informative ones. Yet, if the regularization is too strong, it may oversimplify the model, thereby introducing bias and leading again to underfitting.
  Moving beyond linear models, non-linear function approximators like kernel methods or neural networks can capture more complex relationships within the data that linear models might miss. These methods are more flexible and can model interactions among features, potentially leading to better performance when the reward structure is inherently non-linear. However, this added flexibility comes at a cost. Non-linear models generally demand significantly more computational resources, both in terms of time and memory, and they often require a larger amount of data to achieve stable, reliable estimates. Moreover, their increased complexity can make them more susceptible to overfitting, particularly if the model is not carefully regularized or if there is insufficient data to support the additional degrees of freedom.
  Thus, the core trade-offs involve balancing computational efficiency and statistical performance. A

simple linear model like LinUCB is computationally attractive and easier to interpret, but it may suffer from high variance and overfitting in high-dimensional spaces where the true signal is sparse. On the other hand, while more complex models or non-linear approximators can potentially capture richer patterns in the data, they incur higher computational costs and require careful tuning to avoid overfitting. The decision between these approaches hinges on factors such as the size and quality of the data available, the computational budget, and the acceptable risk level of making suboptimal decisions due to model inaccuracies.