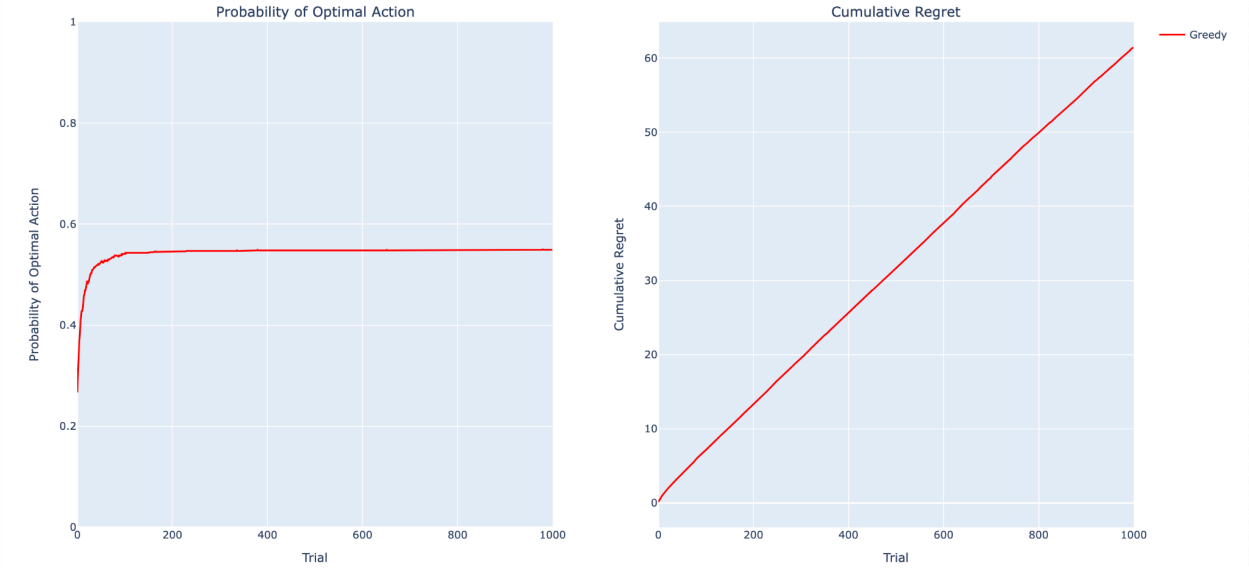
Problem 2:

Expectations:

Given the agent's behavior of randomly exploring in the initial state and then choosing the arm with higher cumulative rewards, the agent may achieve stable and consistent performance over time. However, if the initial random exploration does not identify the true optimal arm, the result in the long term is suboptimal.

Sim Results:



Interpretation:

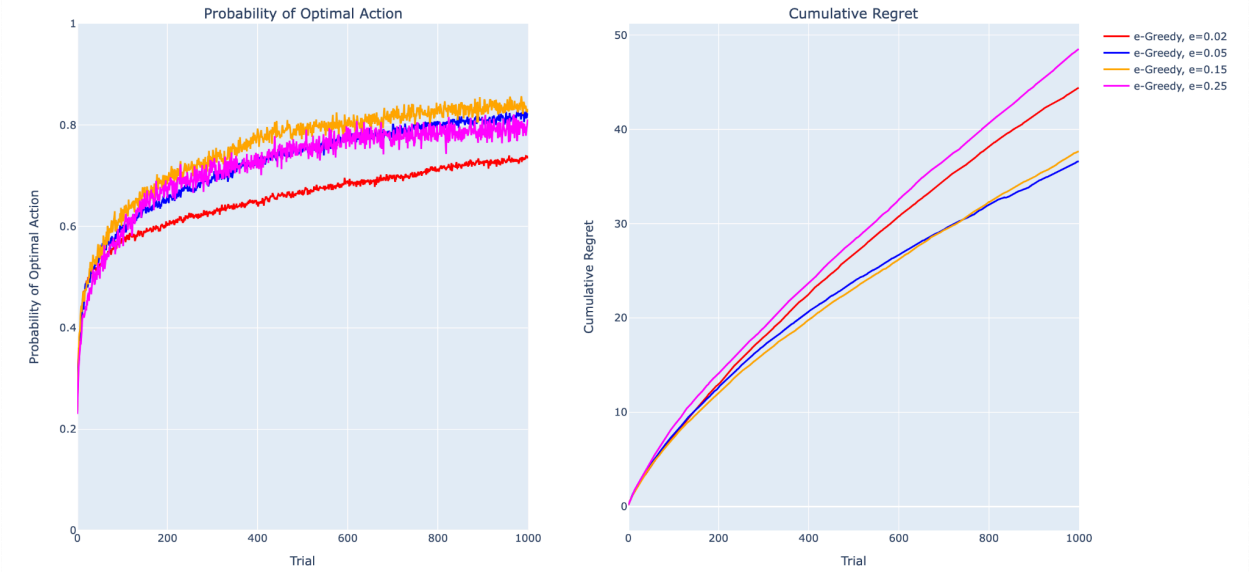
1. 0.55
2. In our simulation, the agent initially explores and exploits randomly, choosing arms without a clear preference. However, after around 100 trials, the agent identifies an arm with consistently higher cumulative rewards and, based on a greedy strategy, starts favoring this arm. As a result, the probability of choosing the correct action stabilizes quickly, indicating that the agent, following a less exploratory approach, tends to exploit the perceived optimal arm rather than actively exploring other possibilities.

Problem 3:

Expectations:

While we will close in on a range, we will never converge to true P(At=a*). For lower Ɛ values (eg. 0.02) will result in little exploration, leading to a suboptimal result. The greater the value of Ɛ, the more exploring we will do. However, the higher Ɛ values (e.g. 0.25) might explore too much, leading to less time to exploit the optimal arm within the given time.

Sim Results:



Interpretation:

1. e = 0.02 0.72
   e = 0.05 0.81
   e = 0.15 0.82
   e = 0.25 0.79
2. e = 0.05 ended up with the least cumulative regret. This is not the same as the one with the highest OPT(T=1000) because the e=0.05 is a smaller epsilon value, which means it takes less risk (as in exploring) and is more likely and is picking the action with the highest probability up to t trials more often than e=0.15. Given the agent with e=0.15 does more exploring by picking greater total random actions, it is able to get a higher probability for optimal action, while being punished in cumulative regret.

Problem 4:

Expectations:

For lower values, such as explore_until_t = 10, might explore too little and miss out on the optimal result. On the other hand, higher values, such as explore_until_t = 200, might explore too much, and result in high initial cumulative regret with hopes it tapers off significantly after much exploration.

Sim Results:

Interpretation:

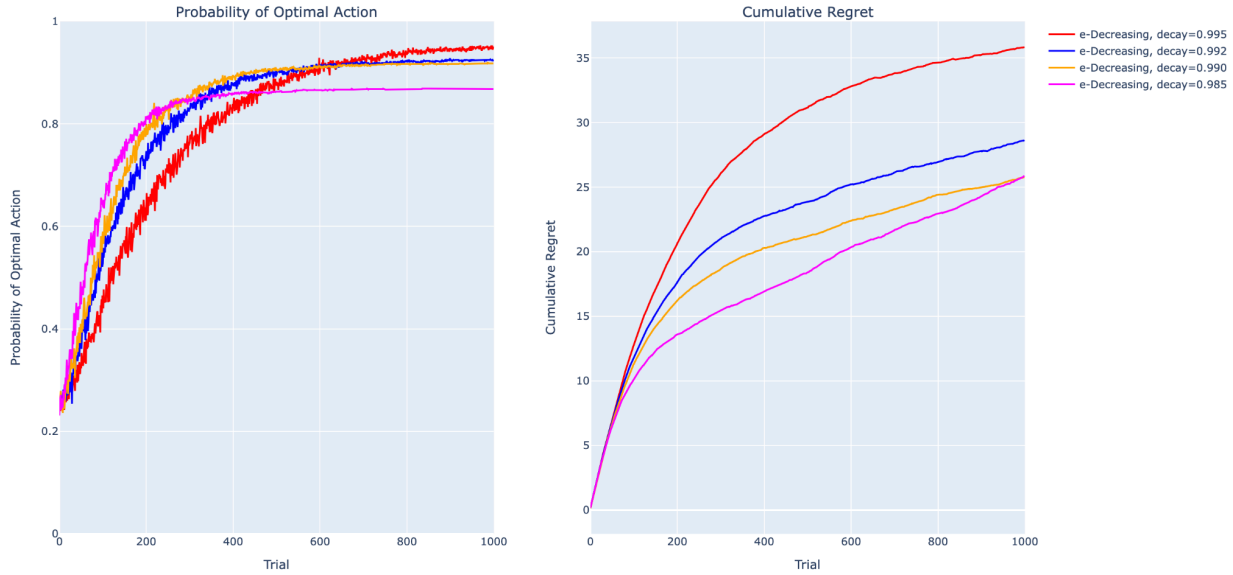
1. e = 10 0.64
   e = 50 0.83
   e = 100 0.89
   e = 200 0.94
2. e = 0.50 ended up with the least cumulative regret. This is not the same as the one with the highest OPT(T=1000) because the agent that explored 200 trials was able to find a high probability in finding the correct action, but cost them heavily in exploring in the first 200 trials. This means even though the regret had tapered off after reaching a high probability of optimal action, it had already accumulated so much regret from the initial exploration. On the other hand, the agent that explored up to 50 trials had more errors in selecting the correct action through T trials, but given it had accumulated much less initial regret whilst exploring, the cumulative regret when reaching T=1000 was much less.

Problem 5:

Expectations:

A higher decay rate, such as 0.995, might cool the exploration rate too quickly, while a lower decay rate, such as 0.985, might cool the exploration rate too slowly. In order to choose the optimal arm, agents need to gain enough information about the arms. Thus, agents with higher decay rates would expect to have higher cumulative regret scores since they explore too little. On the other hand, agents with lower decay rate would have lower cumulative scores but pay off their probability of optimal action since they explore too much.

Sim Results:



Interpretation:

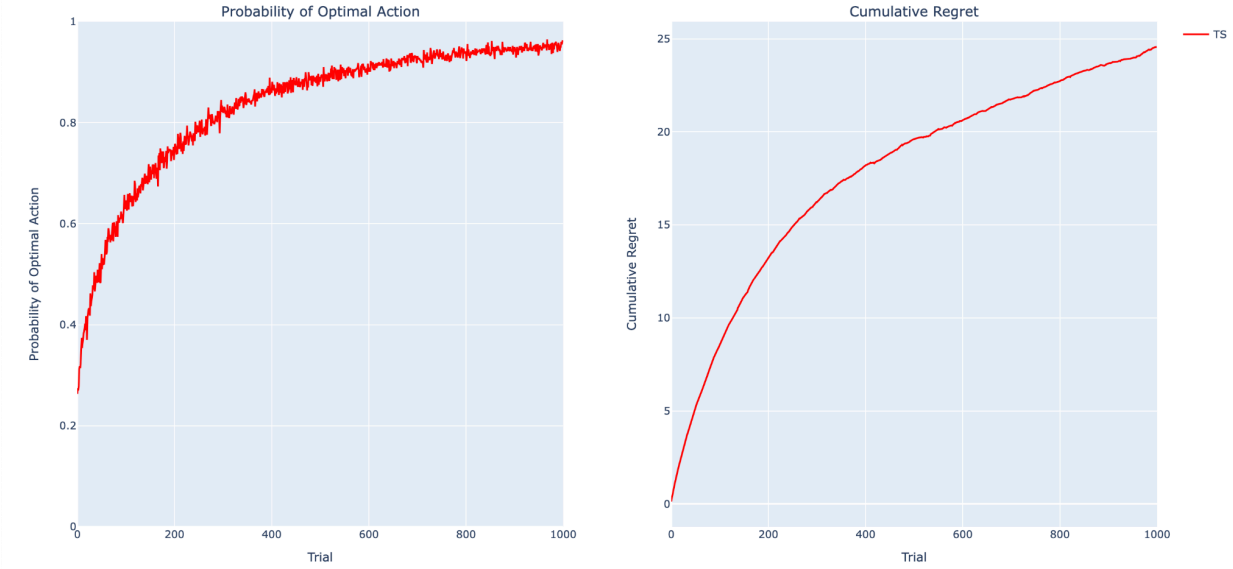
3. e = 0.995 0.947
   e = 0.992 0.924
   e = 0.990 0.917
   e = 0.985 0.868
4. e = 0.985 ended up with the least cumulative regret. This is not the same as the one with the highest OPT(T=1000). Epsilon's first algorithm tends to balance between exploration and exploitation. Agents with a decay rate of 0.985 have the lowest probability of optimal action. This is because they have a longer time of exploration, meaning choosing random solutions; thus decreasing the probability of choosing optimal action. However, since they have a longer time of exploration, they gain more complete information about the available arms; thus, they are able to find optimal solutions when they are exploiting, resulting in lower scores for cumulative regret.

Problem 6:

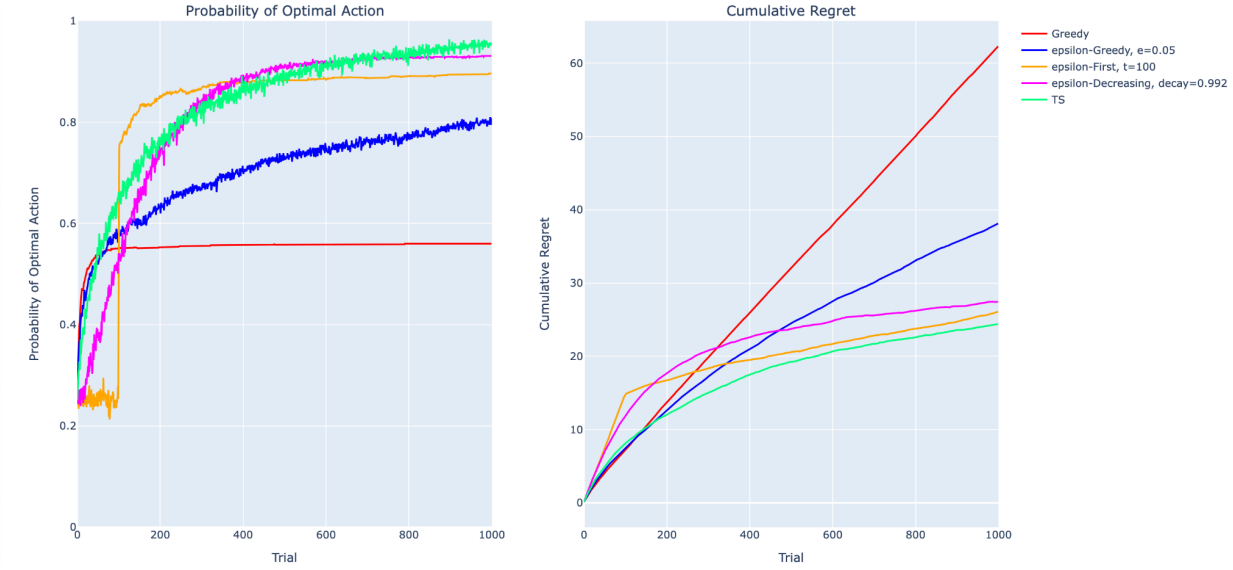
Expectations:

Thompson Sampling assumes a simple prior distribution on the reward of every arm, typically centered around 0.5. It maintains posterior distributions for each arm's true probability of success and updates these parameters based on the outcome of each time step. To update, it samples from these distributions according to the result of each time step and chooses the arm with the highest sampled value.

Sim Results:



Sim Comparison:



Reflect:

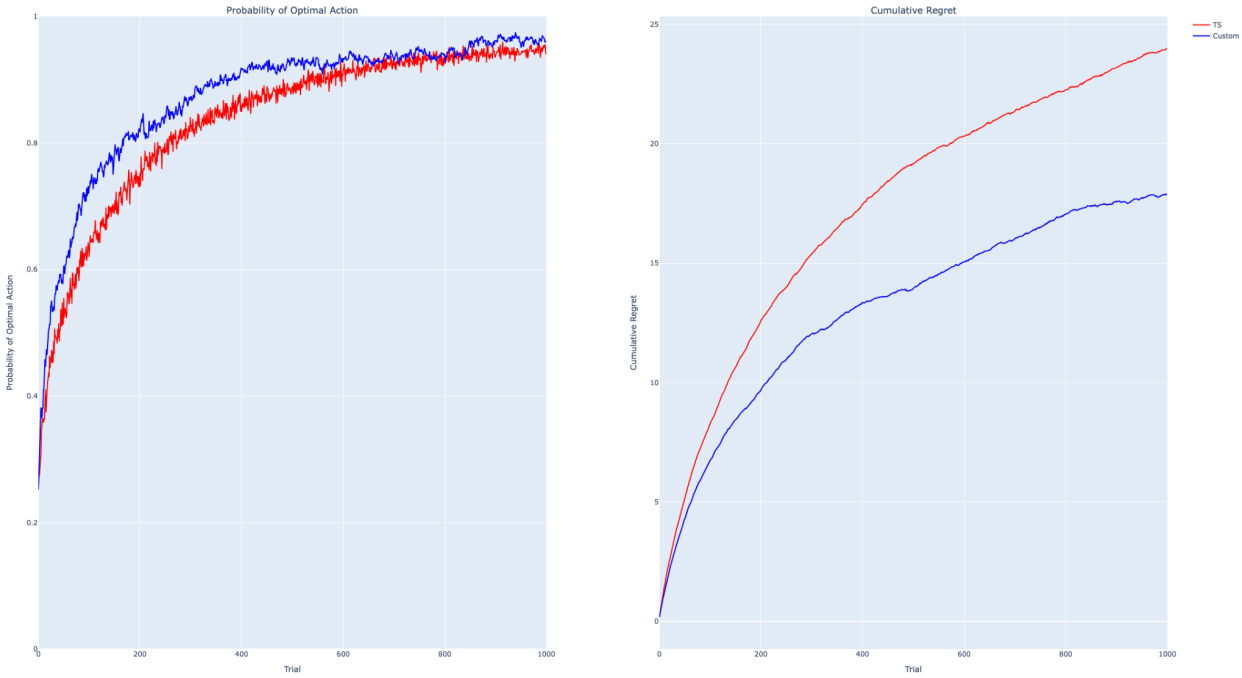
The Thompson Sampler outperforms other agents, showing the highest probability of optimal action and the lowest cumulative regret. Greedy agents exhibit the lowest optimal probability and highest cumulative regret due to minimal exploration and fixed choices based on estimated values. Epsilon Greedy, while exploring initially, lacks adaptability post-exploration. With its cyclic approach of exploration and exploitation, Epsilon Decreasing adapts better but follows a predefined schedule, limiting flexibility. In contrast, the Thompson Sampler dynamically adjusts exploration, effectively balancing exploration and exploitation by sampling from posterior distributions.

Thompson Sampler's strengths are its adaptability and ability to continuously refine estimates over time, balancing exploration and exploitation. Thus, if time goes on forever, Thompson Sampler's agent is able to adapt to uncertainty and provide the most optional actions.

Problem 7:

Expectations: The custom agent is implemented using the Upper-Confidence-Bound algorithm - balancing exploration and exploitation. Based on the equation, we select the action with the highest action value plus the upper confidence bound exploration term, considering what we know and are uncertain about. c is the hyperparameter term that controls the level of exploration. We decided to have c ~ 0.5 to have a balance between exploration and exploitation. After testing different values, c = 0.4 results in outperforming Thompson Sampler.

Sim Results:



Problem 8:

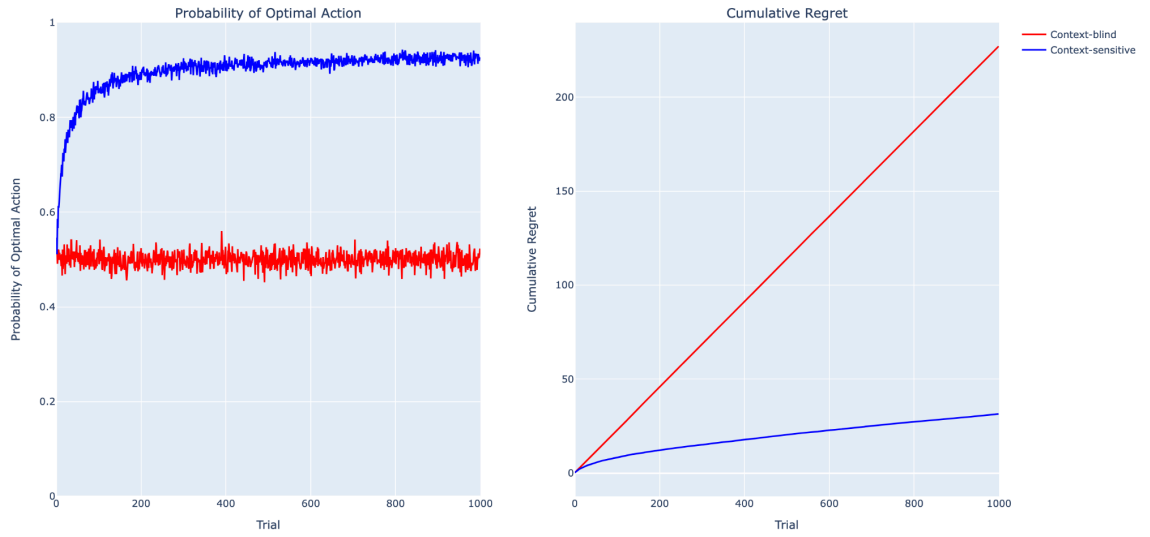
Expectations:

The agent maintains a history(dictionary) with the action and context combined in a tuple as a key and a probability (total reward/times chosen) as the value. In each cycle,

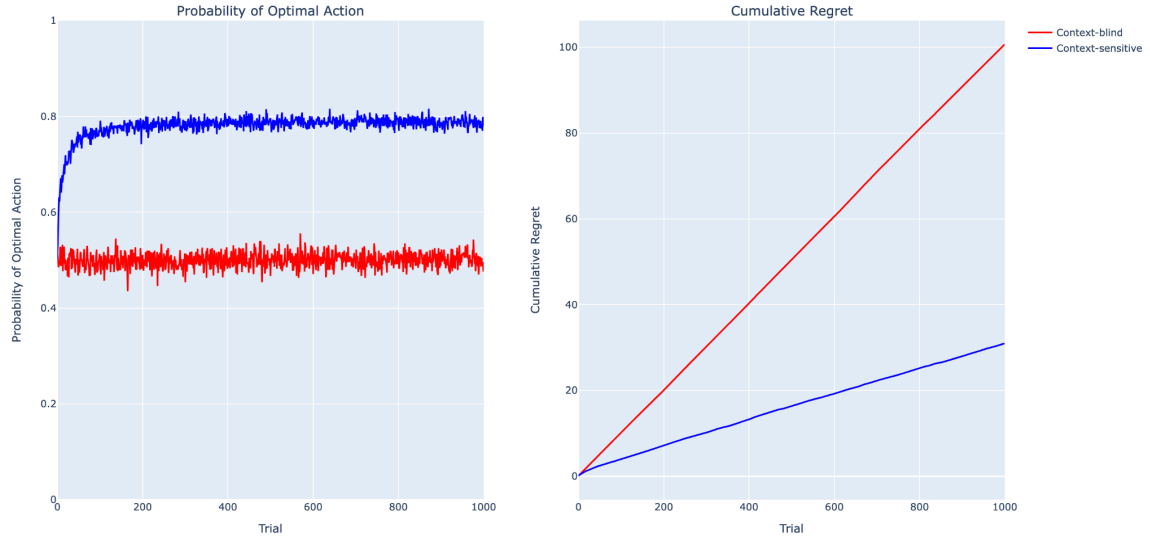
it iterates through all actions combined with the current context to see if a record exists in our history. If it does exist and has a q_at largest of all others, it will proceed with that action. If it does exist and has a q_at stored smaller than unexplored actions, it will attempt another action. It will perform better or worse compared to agents with no context, given how relevant the context given is. The context will be factored into our policy as a context is stored alongside an action and a probability associated with that. Each action in our history will be specifically tailored to the context so the agent is able to act in a very specific way.

Sim Results:

- Advance



- Basic



Interpretation:
In the basic contextual setting with a single context variable, the agent's decision-making is limited, relying on a narrow perspective of the environment. It generates an action choice based on a singular context variable and receives feedback in the form of a reward. In contrast, the advanced setting with four context variables allows the agent to consider a broader range of features, resulting in a more comprehensive understanding of the decision-making environment. This richer contextual information enables the agent to make more informed and adaptive choices, potentially leading to better performance. The number of context variables matters as it influences the depth of the agent's understanding and its ability to capture the complexity of the underlying system, although the effectiveness also depends on the quality and relevance of the provided context variables.

Problem 9:
MAB Problem Variant: Contextual Bandits

Summary: The contextual bandit problem is an extension of the traditional multi-armed bandit (MAB) problem, where the agent receives additional context or information before making each decision. In the traditional MAB problem, the agent chooses from a set of arms at each time step without any context. In the contextual bandit, the agent is provided with a set of contextual variables, and the goal is to learn a policy that maps the context to the best arm to pull.

Key Differences: In a contextual bandit problem, the agent is given a set of context variables (such as user demographics, environmental conditions, etc.) at each time step, along with the opportunity to pull one of the available arms. Given the current context, the agent must learn to exploit the arms that are most likely to yield high rewards. This adds a level of complexity compared to traditional MAB, where the arms' performance is independent of any context.

Strategy for Contextual Bandits: One common strategy for contextual bandits is to model the expected reward of each arm as a function of the context variables. This involves learning a mapping from the context to the expected reward for each arm. Techniques like linear regression, decision trees, or more complex methods like neural networks can be used for this purpose. Once the model is trained, the agent can use it to make decisions by selecting the arm with the highest predicted reward given the current context.

Application Example in Computing:

Scenario: Personalized Content Recommendation System

In the field of recommender systems for online platforms, a contextual bandit approach can be applied. Consider a scenario where a streaming service wants to recommend content (movies, shows, music) to its users based on their preferences, watching history, and current context.

Variant's Application: Traditional MAB would involve randomly recommending content to users, but in a contextual bandit setting, the system can take into account additional information such as the user's genre preferences, viewing time, or device type.