# Reinforcement Learning with Dynamic Programming and Model-Free Control

Nikhil Solanki
nikhil.u.solanki@gmail.com

April 2026

## 1 Introduction

Modern day machine learning frameworks have diverse applications in various industries. Leveraging machine learning algorithms in domain-specific settings have proved to be imperative to optimizing complex systems where static rules fail. While Supervised Learning (SL) algorithms excel at classification, many real-world problems (such as robotics or autonomous driving) require a machine learning framework, or *agent*, to make a sequence of decisions. These agents must navigate a series of dynamic states to reach a long-term goal. This challenge is addressed through Reinforcement Learning (RL), providing the framework for learning through trial and error, formally modeled using Markov Decision Processes (MDPs).

This paper analyzes both model-based and model-free reinforcement learning approaches to better understand their convergence behavior across different types of MDPs. The model-based dynamic programming algorithms, included Value Iteration (VI) and Policy Iteration (PI). These methods assume full knowledge of the environment's transition dynamics and iteratively compute optimal value functions and policies. The model-based methods included Q-Learning and SARSA (State-Action-Reward-State-Action). These methods learn action-value functions directly from experience without requiring an explicit model of the environment. SARSA is an *on-policy* method that updates its values based on the agent's current policy, whereas Q-Learning is an *off-policy* method that updates towards a greedy optimal policy.

These methods were evaluated on two fundamentally different environments. The first is the Blackjack environment, a discrete and stochastic card game where the agent must make decisions under uncertainty. The second is the CartPole environment, a continuous and deterministic control problem, with the goal being to balance a pole on a moving cart. Since dynamic programming and tabular RL methods require a discrete state space, the CartPole environment was discretized.

The following hypotheses were proposed for this study:
For **Blackjack**, I expect VI and PI to converge quickly and produce nearly identical policies because the environment is already discrete and relatively compact. Between the model-free methods, I expect Q-Learning to exhibit smoother and more stable learning than SARSA. It should have a higher final return, due to the off-policy target in a stochastic setting.

For **CartPole**, I expect discretization resolution to have a larger effect on performance than small changes in discount factor ($\gamma$) alone. Coarse discretization should cause state aliasing and weaker policies, while increasing resolution (especially in pole angle and angular velocity) should improve both DP and tabular RL performance to a degree. Under fixed exploration schedules, Q-Learning should ultimately outperform SARSA in average return.

## 2 Markov Decision Processes

### 2.1 Blackjack

The Blackjack environment is a discrete, stochastic Markov Decision Process (MDP) based on the standard card game. At each step, the agent observes a state defined by 3 variables: the player's current sum, the dealer's visible card, and whether the player has a usable Ace. The action space consists of two possible actions: **hit** (draw a card) or **stick** (end the player's turn).

The reward structure is sparse. The agent receives a reward of $+1$ for winning, $-1$ for losing, and 0 for a draw. Due to the randomness of card draws, the transition dynamics are stochastic. This prompts the agent to learn the policy that balances risk and reward under uncertainty. Blackjack's small and discrete state space prompts the use of tabular RL methods, along with intuitive policy visualization via heatmaps.

### 2.2 CartPole

The CartPole environment is a continuous, deterministic MDP in which the agent must balance a pole on a moving cart by applying forces to the left or right. The state is represented by 4 variables: cart position, cart velocity, pole angle, and pole angular velocity. The action space is discrete, consisting of two actions: applying a force to move the cart either **left** or **right**.

Unlike Blackjack, CartPole does not naturally support aforementioned methods due to its continuous state space. To enable the use of VI, PI, Q-Learning, and SARSA, the state space was discretized into a finite number of bins, with higher resolution allocated to pole angle and angular velocity. These crucial parameters were prioritized for maintaining balance, while coarser representations were explored for favoring cart position and velocity. Multiple discretization configurations were evaluated, and their impact on performance is analyzed in later sections.

To construct a model suitable for dynamic programming methods, transition probabilities for the discretized state space were estimated empirically. This was done by sampling interactions with the environment, and recording transition counts between discretized states. These samples were then normalized to approximate the transition probabilities.

The reward structure for CartPole is $+1$ for every timestep the pole remains balaned, encouraging the agent to "keep it up for as long as possible." Discretization fundamentally introduces approximation error, with multiple continuous states potentially mapping to the same discrete representation. This trade-off between computational efficiency and policy accuracy is explored in later sections.

# 3 Methods

## 3.1 Value Iteration (VI)

VI is a dynamic programming algorithm that computes the optimal value function by iteratively applying the Bellman Optimality update. At each iteration, the value of each state is updated according to the maximum expected return over all possible actions, given the current estimate of the value function ($V$). This process continues until convergence, defined by the maximum change in value across all of the states falling below a predefined threshold. In this experiment, VI is applied to both Blackjack and discretized CartPole environments. For CartPole, VI operates on an empirically estimated transition model, derived from sampled interactions with the environments. The discount factor ($\gamma$) is treated as a tunable hyperparameter, and studied to understand its effect on convergence and policy quality.

## 3.2 Policy Iteration (PI)

PI is another dynamic programming algorithm that alternates between two steps: policy evaluation and policy improvement. During policy evaluation, the value function $V$ is computed for a fixed policy. During policy improvement, the policy is updated greedily with respect to the current value function $V$. This process repeats until the policy stabilizes.

Compared to VI, PI typically converges in fewer iterations since it reaches the optimal policy faster. This is because it solves the system of equations for the current policy's value exactly (or near-exactly) during the evaluation phase. PI is applied to both environments under the same transition model and hyperparameter settings as VI, promoting direct comparisons of convergence behavior and resulting policies.

## 3.3 SARSA

SARSA ($S$ - current state, $A$ - action taken in that state, $R$ - reward received, $S'$ - next state, $A'$ - next action chosen, based on current policy) is an on-policy temporal-difference learning algorithm that updates the action-value function based on the action actually taken by the current policy (using $A'$ to update the current state-action pair $(S, A)$). The update rule incorporates the next state-action pair, making the learning process dependent on the agent's exploration strategy.

In this study, SARSA is implemented using an $\varepsilon$-greedy exploration strategy, where the agent selects a random action with probability $\varepsilon$, and follows the current policy otherwise. SARSA learns action-value estimates that reflect the agent's exploration policy. This can lead to more conservative and stable behavior in environments where exploratory actions carry risk. This makes it robust in environments like CartPole, where a risky exploratory move can lead to immediate failure, or Blackjack, where the agent's performance in the game through a hand directly dictates the learning signal. The learning rate $\alpha$, discount factor $\gamma$, and exploration schedule ($\varepsilon$-decay) were treated as tunable hyperparameters. The algorithm is evaluated based on its ability to learn stable policies and maximize cumulative reward over time.

## 3.4 Q-Learning

As opposed to SARSA's on-policy approach prioritizing stability by learning from the agent's actual trajectory, Q-Learning offers a more optimistic and greedy approach. As an off-policy temporal-difference learning algorithm, it updates the action-value function using the *maximum* estimated reward of the next state, rather than the action actually taken. This enables Q-Learning to aggressively converge toward the optimal policy even while following an exploratory behavior policy. The learning rate $\alpha$, discount factor $\gamma$, and exploration schedule ($\varepsilon$-decay) were treated as tunable hyperparameters again. Q-Learning can achieve higher performance at the cost of higher variance during training, especially in environments with stochastic transitions or coarse discretizations.

## 3.5 Experimental Setup

All algorithms were evaluated on the Blackjack and CartPole environments using consistent experimental protocols. For CartPole, the continuous state space is discretized into a finite number of bins to enable the use of tabular/dynamic programming methods. For VI and PI, we validated $\gamma$ on Blackjack and discretization resolution on CartPole, reflecting the primary sources of variation for each environment. Multiple discretization configurations were explored for CartPole, with an emphasis on higher resolution for pole angle and angular velocity. For SARSA and Q-Learning, we conducted staged hyperparameter sweeps over $\gamma, \alpha$, and $\varepsilon$-decay, using a pilot evaluation to rank candidates followed by multi-seed confirmation of the top configurations.

### 3.5.1 Transition Model Estimation (VI/PI)

For dynamic programming methods (VI/PI), an empirical transition model was constructed for both environments. This was achieved by running exploratory episodes using a uniform random policy, and recording transition counts for each observed state-action pair. Specifically, for each state $s$ and action $a$, counts of transitions to $(s', r, done)$ were accumulated and then normalized to estimate the transition probabilities. For both Blackjack and CartPole environments, 5000 sampled episodes were used to estimate transition dynamics. Coverage diagnostics were also computed to measure how much of the state-action space was explored during data collection.

### 3.5.2 Discretization

As a result of the CartPole environment having a continuous state space, discretization was used to section the space into a finite set of bins. This enables the use of tabular and dynamic programming methods. The state variables (cart position, cart velocity, pole angle, pole angular velocity) were first clamped to fixed bounds: $(-2.4, 2.4)$ for cart position, $(-3.0, 3.0)$ for cart velocity, $(-0.2, 0.2)$ for pole angle, and $(-3.5, 3.5)$ for pole angular velocity. It was then discretized into bins leveraging uniform partitioning.

Multiple discretization configurations were evaluated: $(1, 1, 6, 12), (1, 1, 12, 12), (3, 3, 6, 12), (4, 4, 8, 8), (6, 6, 6, 6)$. These configurations were chosen to explore the tradeoffs

between state space granularity and computational complexity. Configurations such as $(1, 1, 12, 12)$ emphasize angular resolution while ignoring cart position and velocity, while $(6, 6, 6, 6)$ represents a more balanced but significantly larger state space (1296 states).

#### 3.5.3 Value Iteration and Policy Iteration

For both Blackjack and CartPole, VI and PI were evaluated across discount factors: $\gamma \in \{0.95, 0.99\}$. Convergence for VI was determined using a threshold on the maximum change in the value function ($\Delta V < 10^{-6}$), while PI was terminated when the policy stabilized (no further changes between iterations). Final policies were evaluated by executing them in the environment for 500 episodes, and performance was measured using the rolling average reward over episodes. Since VI and PI are deterministic given a fixed model, variability arises only from transition estimation. Therefore, multi-seed evaluation was applied at the final stage to assess robustness, while sweep-stage comparisons used a fixed seed for consistency.

#### 3.5.4 Model-Free Methods

For model-free methods (SARSA and Q-Learning), agents interact directly with the environment over multiple episodes to learn action-value functions. An $\varepsilon$-greedy exploration strategy is used for both algorithms, with $\varepsilon$ decaying over each of the 5000 episodes (per configuration) to balance exploration and exploitation. Both SARSA and Q-Learning began their $\varepsilon$-greedy exploration with an initial $\varepsilon = 1.0, \varepsilon$-decay$\in \{0.995, 0.999\}$, and a minimum $\varepsilon = 0.05$ (Blackjack) or $\varepsilon = 0.01$ (CartPole). This ensured sufficient exploration early in training while allowing policies to stabilize over time.

Hyperparameters are selected through a staged hyperparameter search process, beginning with coarse exploration of parameter ranges followed by refinement around promising configurations. Key hyperparameters include the discount factor $\gamma$, learning rate $\alpha$, and $\varepsilon$-decay schedule.

All experiments were evaluated across five independent random seeds $\{0, 66, 420, 820, 9999\}$, and results are reported using mean performance with standard deviation to account for variability across runs.

#### 3.5.5 Hyperparameter Search Strategy

A staged hyperparameter search strategy was used for both SARSA and Q-Learning. **Stage 1: Pilot Sweep** — A coarse grid of hyperparameters were evaluated over 200 episodes to identify promising configurations: $\gamma \in \{0.95, 0.99\}, \alpha \in \{0.1, 0.5\}$, $\varepsilon$-decay$\in \{0.995, 0.999\}$. Configurations were ranked based on the average reward over the final $k$ episodes, and the top-performing configurations were selected for further evaluation.

**Stage 2: Final Evaluation** — The top configurations from the pilot sweep were then evaluated over 5000 episodes across all seeds. Performance was measured using the mean final reward (rolling average) and standard deviation across seeds.

# 4 Results & Analysis

## 4.1 Value Iteration & Policy Iteration (Blackjack)

To evaluate the performance and convergence behavior of model-based methods, VI and PI were applied to the Blackjack environment using the empirically estimated transition model described in the previous section.

#### 4.1.1 Convergence Behavior

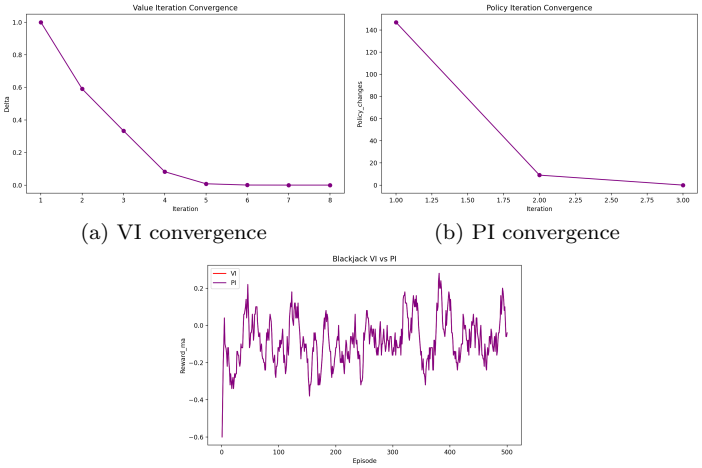
Figure 1a depicts the convergence of VI in terms of the maximum changes in the value function ($\Delta V$), requiring approximately $6 - 8$ iterations to reach the convergence threshold. This portrays the incremental nature of VI, where Bellman backups refine value estimates iteratively across the entire state space. In the context of Blackjack, this rapid convergence reflects the environment's shallow decision horizon with the agent always "thinking" one step into the future; a typical hand is resolved in only a few moves (hits; $\approx 3 - 5$ moves), enabling rewards to propagate from terminal states back to the initial deal very quickly. Since information propagates one step per iteration, the $6-8$ iterations represent the *lookahead horizon* needed for terminal rewards to reach initial states. The value function stabilizes once the mathematical expectations for all possible hand totals are accounted for and the update change falls below the threshold $\theta$.

In contrast, Figure 1b illustrates the convergence behavior of PI, measured in the number of policy changes per iteration. PI demonstrates a rapid reduction in policy changes (from $\sim 150 \rightarrow \sim 10 \rightarrow 0$), dropping from a large number of updates to 0 in about 3 iterations. This indicates PI converges faster in terms of iterations, by performing a full policy evaluation and improve at each iteration. While VI slowly refines granular values, PI identifies the optimal "hit" or "stick" boundaries much earlier. This is due to the discrete nature of the decision-making process enabling policy stabilization, even before the underlying value functions have fully converged.

These findings align with theoretical expectations, where PI typically convergences in fewer iterations than VI because it makes larger updates through policy improvement. On the contrary, VI performs gradual updates to the value function $V$. As seen in Figure 1c, the rolling average of rewards for both VI and PI over 500 evaluation episodes are identical, resulting in overlapping plots. This confirms that despite their different internal implementations (VI's granular refinement versus PI's direct policy search), both algorithms successfully converge to the mathematically optimal policy for Blackjack. The high variance and oscillations seen in the reward signal ($\{-0.4, 0.2\}$) reflect the inherent stochasticity of Blackjack, where even an optimal strategy cannot guarantee a win in every hand.

#### 4.1.2 Policy Analysis

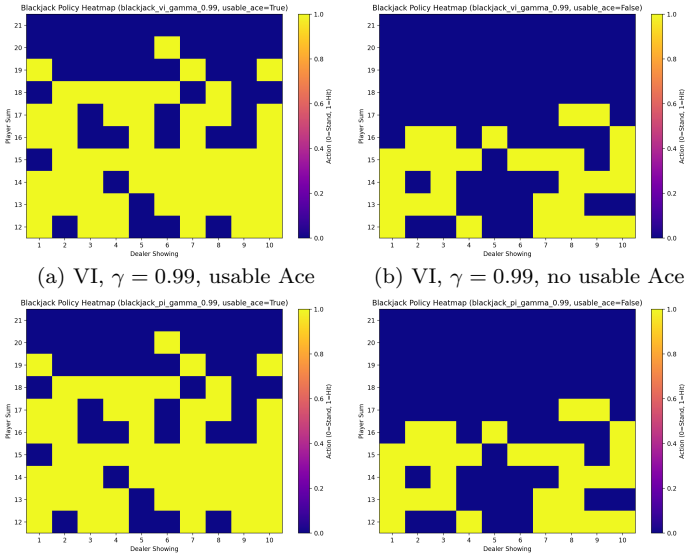
To better understand the structure of the learned policies, heatmaps were generated for both VI and PI under a discount factor of $\gamma = 0.99$ (most favorable $\gamma$ from sweep). These visualizations portray the action selected (hit or stick) for each combination of player sum and dealer showing their card, separated by whether the player holds a usable Ace.




(a) VI convergence (b) PI convergence

(c) VI vs. PI convergence

Figure 1: Value Iteration & Policy Iteration on Blackjack




(a) VI, $\gamma = 0.99$, usable Ace (b) VI, $\gamma = 0.99$, no usable Ace

(c) PI, $\gamma = 0.99$, usable Ace (d) PI, $\gamma = 0.99$, no usable Ace

Figure 2: Value Iteration & Policy Iteration Heatmaps (Blackjack)

Across both usable Ace and no usable Ace scenarios, the policies learned by VI and PI are nearly identical. The structure and decision boundaries of the heatmaps do not show meaningful differences between the two methods. This further supports the findings from section 4.1.1, where both algorithms exhibited similar performance and convergence to the same optimal policy. Despite varying convergence behavior, both methods ultimately produce equivalent decision-making strategies in the Blackjack environment.

A key distinction in the learned policies is shown when comparing states with and without a usable Ace. When the player holds a usable Ace, the policy tends to favor more aggressive behavior, promoting the agent to "hit" in a wider range of states, even at higher player sums. This is likely due to the flexibility provided by a usable Ace, since an Ace can be counted as a 1 or 11 (mitigating a "bust" or loss).

On the contrary, with no usable Ace present, the learned policy becomes significantly more conservative. The agent tends to "stick" at lower player sums, due to the increased risk of busting without the safety net of an Ace. This shift in behavior highlights the agent's ability to adapt its strategy according to the underlying risk structure of its current state.

The learned policies portray a clear decision boundary based on the player's sum and dealer's visible card. In general, the agent tends to "stick" at higher player sums (around $17 - 18+$) while choosing to "hit" at lower sums. The exact boundary shifts depending on the dealer's card, reflecting the increased likelihood of dealer strength when showing higher-value cards; alluding to the notorious assumption that "the House always wins." Overall, the learned policies align well with an intuitive Blackjack strategy, where decisions are made while balancing probability of a bust against the likelihood of beating the dealer's final hand.

## 4.2 SARSA vs. Q-Learning (Blackjack)

To evaluate model-free methods, SARSA and Q-Learning were applied to the Blackjack environment using hyperparameter configurations selected through the staged search process mentioned in section 3.5.5. Performance was measured as the rolling average reward across 5000 training episodes, averaged over multiple random seeds.

### 4.2.1 Learning Behavior

Figures 3a & 3b depict the learning curves for SARSA and Q-Learning respectively. The shaded regions depict $\pm 1$ standard deviation across seeds. Both methods exhibit gradual improvement over time, with the average reward increasing from approximately $-0.6$ early on to around $-0.1$ to 0.0 in later episodes. A plateau near zero represents near-optimal performance, as the expected return in standard Blackjack without card counting is slightly negative ($\sim \{-0.05, -0.1\}$).

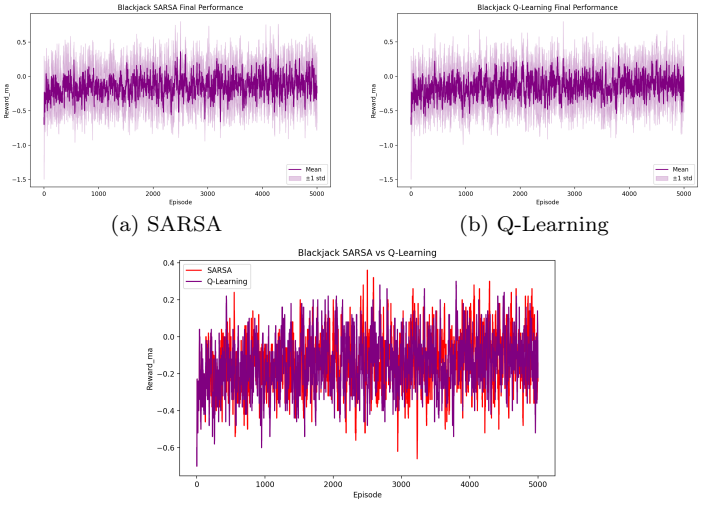
An important observation is the learning process maintains high variance throughout training. This is shown with the large and persistent standard deviation bands indicating high variability in episodic rewards, even after convergence. This behavior is expected since Blackjack will naturally have stochastic transitions and sparse reward signals, introducing inherent randomness into the learning process.

### 4.2.2 Model-Free Comparison

Figure 3 portrays a direct comparison between SARSA and Q-Learning. The two methods demonstrate relatively similar learning trajectories, with neither method having a statistically significant advantage. Both approaches converge to comparable policies, achieving similar average rewards over time.

A slight difference can be seen in the variability of the learning curves. SARSA appears to be more stable behavior, with fewer extreme fluctuations compared to that of Q-Learning. This is due to SARSA's on-policy implementation, which updates based on the agent's actual trajectory. This leads to more conservative and stable learning. In contrast, Q-Learning's off-policy updates rely on the maximum estimated value of the next state (greedy), which can introduce additional variance in training.

The observed results suggest that in the Blackjack environment, the distinction between on-policy and off-policy learning has a limited impact on performance. Both SARSA and Q-Learning converge to similar policies, with neither method proving to be advantageous over the other. This behavior




(a) SARSA (b) Q-Learning

(c) SARSA vs. Q-Learning

Figure 3: SARSA vs. Q-Learning behavior in 5000 episodes across 5 seeds (Blackjack)

can be attributed to Blackjack's relatively small state space ($\sim 280$), as well as the stochasticity of the game itself. Since optimal decisions depend primarily on the current state rather than long-term trajectory planning, both algorithms are able to similarly learn effective strategies despite differences in their update rules. This suggests that exploration strategy and environment stochasticity dominate performance differences more than the choice of update rule in this setting.

## 4.3 Final Model Comparison (Blackjack)

To summarize performance across all approaches, Figures 6a and 6b presents the final evaluation results for Value Iteration (VI), Policy Iteration (PI). Figures 3a and 3b presents SARSA and Q-Learning on the Blackjack environment. Each method was evaluated using its selected hyperparameter configuration, with performance measured as the rolling average reward across evaluation episodes and averaged over multiple seeds.

### 4.3.1 Model-Based vs. Model-Free Methods

A clear distinction can be seen between model-based and model-free approaches. Value Iteration and Policy Iteration exhibit more stable reward trajectories with lower variance compared to SARSA and Q-Learning. This is expected since both VI and PI compute policies using an explicit transition model, allowing them to directly approximate the optimal value function ($V$) without relying on incremental learning from sampled experience.

In contrast, SARSA and Q-Learning demonstrate significantly higher variability in performance. Their large fluctuations in reward especially late in training, reflect the stochastic nature of the Blackjack environment combined with the inherent noise in temporal-difference learning. Despite this variability, both model-free methods are able to learn policies that achieve comparable average rewards over time.

Within model-based methods, VI and PI achieve nearly identical performance. They both converge to the same optimal policy despite differences in convergence speed. Similarly, SARSA and Q-Learning both exhibit comparable final performance among model-free approaches. SARSA demonstrates slightly more stable learning while Q-Learning shows higher variability due to its off-policy updates.

Across all methods, final performance differences were relatively small. All algorithms converged to similar average reward levels. This suggests that the Blackjack environment, due to its small and discrete state space, does not strongly differentiate between model-based and model-free approaches in terms of asymptotic performance.

However, important distinctions arise in terms of learning dynamics and overall stability. Model-based methods converge more reliably and exhibit lower variance. Model-free methods instead require extensive interaction with the environment and exhibit higher variability during training.

Overall, these results highlight that while all methods are capable of learning effective policies for Blackjack, model-based approaches provide more stable and efficient convergence, whereas model-free methods offer a more flexible but noisier alternative.

## 4.4 Value Iteration vs. Policy Iteration (CartPole)

To evaluate model-based methods in the CartPole environment, VI and PI were applied using discretized state representations and empirically estimated transition models. Performance was measured as the rolling average reward across evaluation episodes.

### 4.4.1 Convergence Behavior

Figure 4a and 4b illustrate the convergence behavior of VI and PI respectively. PI converges in a very small number of iterations ($\sim 3-4$), with the number of policy changes dropping to zero after only a few updates. This reflects PI's ability to directly evaluate and improve policies, allowing it to rapidly reach a stable solution.

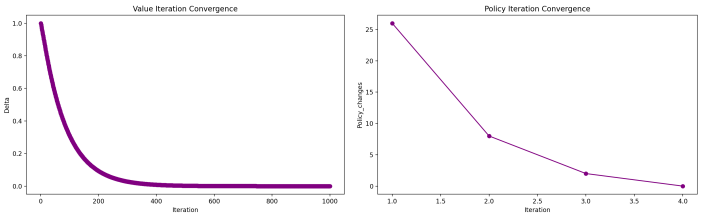
VI exhibits a more gradual convergence pattern. The maximum change in the value function $V$ decreases smoothly over several hundred iterations ($\sim 400-500$) before reaching a sufficiently small threshold. The slower convergence of VI is expected since it incrementally refines value estimates across the entire state space.

### 4.4.2 Performance Comparison

Figures 6c and 6d compares the performance of VI and PI in terms of average reward. Both methods achieve similar levels of performance, with average rewards stabilizing around $195-205$. This indicates that both algorithms are able to learn near-optimal policies for the discretized CartPole environment.

Despite the difference in convergence speed, the final policies produced by VI and PI yield comparable outcomes. Occasional perturbations are observed, including peaks of above 200, corresponding to episodes where the pole remains balanced for extended durations.

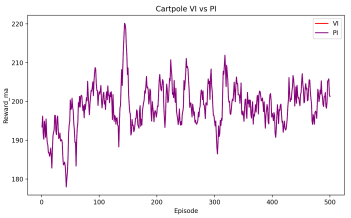
The results demonstrate that while PI converges significantly faster than VI, both methods ultimately achieve similar performance. This highlights a key distinction between the algorithms: PI offers faster convergence through explicit




(a) VI convergence




(b) PI convergence




(c) VI vs. PI convergence

Figure 4: Value Iteration & Policy Iteration on CartPole

policy updates, while VI provides a more gradual but stable refinement of value estimates.

In the context of CartPole, where the transition model is approximated through discretization and sampling, both approaches remain effective despite the underlying approximation. The comparable performance suggests that, given a sufficiently accurate transition model, either method can be used to obtain high-quality policies. The rewards differ from those of Blackjack, proving learning to be more stable and interpretable (rewards are dense and continuous).

#### 4.4.3 Discretization Study

Since CartPole operates in a continuous state space, discretization is required to apply model-based methods (VI & PI). To evaluate the impact of discretization, multiple bin configurations were tested while keeping the discount factor fixed at $\gamma = 0.99$.

A grid search was performed over several discretization schemes, varying the number of bins across the four state dimensions (cart position, cart velocity, pole angle, and pole angular velocity). The configurations were: $(1, 1, 6, 12), (1, 1, 12, 12), (3, 3, 6, 12), (4, 4, 8, 8), (6, 6, 6, 6)$.

For clarity and to avoid redundancy, results are presented using VI as a representative model-based method and include a subset of three discretization schemes, since both PI and additional configurations exhibited similar trends without providing additional insight. Performance was evaluated with average reward measured over evaluation episodes.

- $(1, 1, 6, 12)$: coarse discretization for position/velocity & finer resolution for angle
- $(4, 4, 8, 8)$: moderate discretization across all dimensions
- $(6, 6, 6, 6)$: uniformly fine discretization

The **results** show that discretization has a significant impact on both learning stability and final performance.

The $(1, 1, 6, 12)$ configuration achieves the most stable and consistently high performance. By allocating more resolution to the most critical features to balance (pole angle and angular velocity), the model is able to capture the essential dynamics of the system while avoiding unnecessary complexity in less relevant features.

The $(4, 4, 8, 8)$ configuration provides a more uniform representation of the state space. It achieves comparable performance, with slightly increased variability. This suggests that while additional resolution can improve representational capacity, it also introduces more noise due to increased state sparsity.

Lastly, the $(6, 6, 6, 6)$ configuration performs worse despite its higher resolution. The larger state space leads to sparse transition estimates, making it difficult for VI to accurately propagate value information. This leads to less stable learning and reduced overall performance.

| Configuration | Visited States | Total States | Coverage (%) | Reward |
|---|---|---|---|---|
| (1,1,6,12) | 42 | 72 | 58.3 | 214.5 |
| (1,1,12,12) | 82 | 144 | 56.9 | 478.6 |
| (3,3,6,12) | 46 | 648 | 7.1 | 214.5 |
| (4,4,8,8) | 109 | 1024 | 10.6 | 225.0 |
| (6,6,6,6) | 106 | 1296 | 8.2 | 171.8 |

Table 1: Effect of state space discretization on coverage and policy performance in CartPole.

Although $(1, 1, 12, 12)$ achieves the highest final reward, it requires a state space that is twice as large as $(1, 1, 6, 12)$ while providing nearly identical coverage (56.9% vs. 58.3%). As a result, $(1, 1, 6, 12)$ was selected as the final configuration due to its more efficient representation, offering a better trade-off between performance and computational complexity.

### 4.5 SARSA vs. Q-Learning (CartPole)

To evaluate model-free methods on the CartPole environment, SARSA and Q-Learning were trained using discretized state representations. Both algorithms were evaluated across multiple random seeds, and results are reported using mean reward with variability bands ($\pm 1$ standard deviation).

#### 4.5.1 Learning Behavior

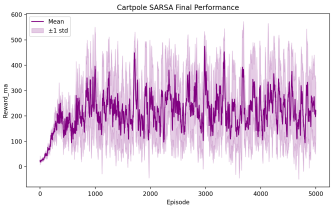
Both SARSA and Q-Learning demonstrated rapid improvements during the early stages of training. Performance increases significantly within the first few hundred episodes. This indicates that both algorithms are able to quickly learn a reasonable balancing strategy in the discretized CartPole environment.

After this initial improvement, performance increases in variance for both methods. The reward curves exhibit frequent perturbations, reflecting the instability in the learned policies. This variability is further confirmed by the wide standard deviation bands depicted in both methods.

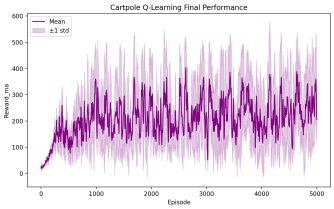
#### 4.5.2 Model-Free Comparison

Overall, SARSA and Q-Learning achieve comparable final performance levels, with both methods stabilizing around similar average rewards. Neither algorithm consistently dominates across the full training horizon. This suggests that in this discretized setting, the choice between on-policy and off-policy learning has a limited impact on final reward.

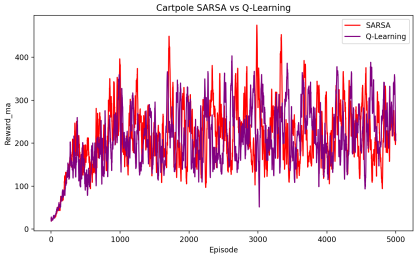
SARSA tends to produce slightly smoother and more stable learning curves. This is likely due to SARSA updating its value estimates using actions actually taken under the current policy. The agent implicitly accounts for exploration, leading




(a) SARSA




(b) Q-Learning




(c) SARSA vs. Q-Learning

Figure 5: SARSA vs. Q-Learning behavior in 5000 episodes across 5 seeds (CartPole)

to more conservative updates. Q-Learning exhibits more aggressive and higher-variance behavior. This is expected behavior of Q-Learning's greedy update approach of tending toward the maximum possible future value. Q-Learning can overestimate action values, which leads to larger fluctuations in performance. This can be seen in Figure 5b, with sharper spikes and dips in the reward trajectory.

Expanding upon similarities, both methods portray significant variability even after convergence. This may be a result of exploration ($\varepsilon$-greedy) continuing to introduce stochasticity throughout training. The environment dynamics may also be sensitive to small deviations in policy. This behavior can promote further investigation into model-free methods and their performance in deterministic environments.

These results highlight that although model-free methods are capable of learning effective policies in the CartPole environment, they are less stable than model-based approaches (VI & PI). The combination of both discretization and stochastic updates introduces variance that persists across multiple seeds.

Overall, SARSA offers slightly more stable learning due to its on-policy nature, while Q-Learning provides more aggressive updates that can lead to higher variance. However, both methods converge to similar levels of performance, indicating that either approach is viable for this problem setting.

### 4.6 Final Model Comparison (CartPole)

To compare all algorithms on the CartPole environment, I evaluate the final performance of Value Iteration (VI), Policy Iteration (PI), SARSA, and Q-Learning using their respective mean reward trajectories.

#### 4.6.1 Model-Based vs. Model-Free Methods

Another clear distinction can be seen between model-based and model-free approaches. VI and PI produce nearly identical performance curves and exhibit more stable reward trajectories with lower variance compared to SARSA and Q-Learning (approximately around $200 - 210$). This again is expected behavior since both VI and PI compute policies using an explicit transition model, allowing them to directly approximate the optimal value function ($V$) without relying on incremental learning from sampled experience.

Both SARSA and Q-Learning portray rapid initial learning followed by persistent high-variance perturbations. Rewards oscillate beween roughly $100 - 400+$, indicating sensitivity to stochastic exploration. Q-Learning reaches higher peaks by targeting the maximum possible future value, while SARSA appears to be more conservative by penalizing its current policy for its actions taken during exploration.

Overall, while all four methods achieve similar average reward levels near the environment's maximum capacity ($\sim$ 200–250), their stability differs significantly. VI and PI exhibit near-zero variance and smooth convergence, since they leverage a complete transition model to solve the environment. In contrast, SARSA and Q-Learning exhibit high variance and frequent reward fluctuations. This instability in model-free methods is a direct result of the online exploration-exploitation trade-off; even a nearly optimal agent occasionally takes sub-optimal exploratory actions that, in a sensitive control task like CartPole, lead to immediate episode termination and high variance in the learning curve.

## 5 Blackjack vs. CartPole: Cross-Environment Comparison

| Method | Environment | $\gamma$ | $\alpha$ | $\epsilon$-decay | num_bins |
|---|---|---|---|---|---|
| Q-Learning | Blackjack | 0.99 | 0.5 | 0.995 | – |
| Q-Learning | CartPole | 0.99 | 0.1 | 0.995 | (1,1,6,12) |
| SARSA | Blackjack | 0.95 | 0.5 | 0.995 | – |
| SARSA | CartPole | 0.99 | 0.1 | 0.995 | (1,1,6,12) |
| VI | Blackjack | 0.99 | – | – | – |
| VI | CartPole | 0.99 | – | – | (1,1,6,12) |
| PI | Blackjack | 0.99 | – | – | – |
| PI | CartPole | 0.99 | – | – | (1,1,6,12) |

Table 2: Final selected hyperparameters for all methods and environments after staged hyperparameter sweeps

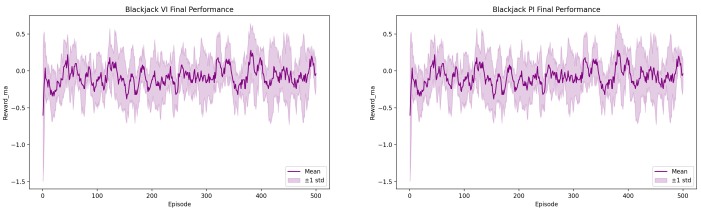
The cross-environment comparison reveals a key difference in how the algorithms behave: in CartPole, all methods achieve similar reward levels and primarily differ in stability, whereas in Blackjack, clear performance gaps emerge between model-based and model-free approaches.

Table 3 summarizes the mean final reward and variability for all methods in the Blackjack environment. VI and PI achieve the strongest final performance, both converging to a mean final reward moving average of $-0.04$. SARSA and Q-Learning conversely performed worse at $-0.24$ and $-0.22$ respectively. This suggests that in the discrete and highly structure Blackjack environment, model-based methods are better able to exploit the underlying dynamics and recover stronger policies.

| Algorithm | Hyperparameters | Mean Reward | Std Dev |
|---|---|---|---|
| Value Iteration | $\gamma = 0.99$ | -0.04 | 0.162 |
| Policy Iteration | $\gamma = 0.99$ | -0.04 | 0.162 |
| SARSA | $\gamma = 0.95,\ \alpha = 0.5,\ \epsilon = 0.995$ | -0.24 | 0.388 |
| Q-Learning | $\gamma = 0.99,\ \alpha = 0.5,\ \epsilon = 0.995$ | -0.22 | 0.264 |

Table 3: Final model performance in Blackjack (mean $\pm$ std over 5 seeds)

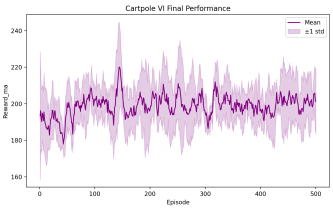
Table 4 summarizes the mean final reward and variability



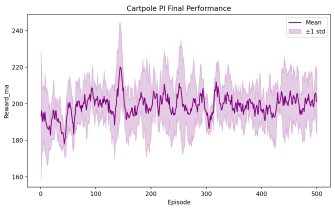

(a) VI on Blackjack




(b) PI on Blackjack




(c) VI on CartPole




(d) PI on CartPole

Figure 6: Average rolling mean performance for VI and PI across Blackjack and CartPole

for all methods in the CartPole environment. All four methods achieved broadly similar rewards levels: the mean final reward moving averages ranged between approximately 201 and 221. The largest differences appeared in variability rather than average performance. SARSA achieved the highest mean reward (221.2), but also maintained the largest standard deviation. VI and PI remained substantially more stable. This indicates that in the discretized CartPole environment, multiple methods can reach strong policies. But, the more reliable and consistent model-based methods prove to have the upper hand, whereas the model-free methods trade stability for potentially higher peak performance.

| Algorithm | Hyperparameters | Mean Reward | Std Dev |
|---|---|---|---|
| Value Iteration | $\gamma = 0.99$, bins=$(1, 1, 6, 12)$ | 201.26 | 15.94 |
| Policy Iteration | $\gamma = 0.99$, bins=$(1, 1, 6, 12)$ | 201.26 | 15.94 |
| SARSA | $\gamma = 0.99,\ \alpha = 0.1,\ \epsilon = 0.995$, bins=$(1, 1, 6, 12)$ | 221.2 | 101.58 |
| Q-Learning | $\gamma = 0.99,\ \alpha = 0.1,\ \epsilon = 0.995$, bins=$(1, 1, 6, 12)$ | 207.44 | 41.29 |

Table 4: Final model performance in CartPole (mean ± std over 5 seeds)

I further compared computational efficiency using wall-clock time across environments. For model-free methods, SARSA and Q-Learning exhibited relatively low runtime on Blackjack (approximately 0.04 minutes), but required significantly longer execution on CartPole (approximately 2.85–3.26 minutes) due to the need for thousands of interaction episodes in a higher-dimensional state space.

For model-based methods, the initial VI/PI sweep on CartPole was the most computationally expensive stage (approximately 12.79 minutes), as it required constructing and solving the transition model over a discretized state space. However, once the model was established, final evaluation was efficient (approximately 0.68 minutes). In contrast, Blackjack VI/PI remained consistently fast ($\leq 0.05$ minutes) due to the small and discrete state space.

These results highlight a key trade-off: model-based methods incur higher upfront computational cost when building the environment model, particularly in larger state spaces. However, they enable fast and stable policy evaluation thereafter. Model-free methods avoid model construction but require substantially more interaction time, leading to higher overall runtime in complex environments.

# 6 Conclusion

This study demonstrated the trade-offs between planning and learning algorithms across two distinct decision-making environments. Model-based methods such as Value Iteration and Policy Iteration provide a stable and high-quality baseline by leveraging an explicit model of the environment. Model-free methods like SARSA and Q-Learning demonstrate the ability to learn effective policies directly from interaction with the environment. Across environments and seeds, a key distinction emerged: in Blackjack, model-based methods achieved stronger final policy quality. In CartPole, all methods reached similar reward levels, with differences attributed to stability and variance. The Blackjack results partially supported the hypotheses: VI and PI converged quickly and produced nearly identical policies as expected, but Q-Learning did not consistently outperform SARSA in final performance. The CartPole results partially supported the hypotheses: discretization was critical to achieving strong performance, but Q-Learning did not consistently outperform SARSA in average return, with differences primarily observed in variance rather than mean performance. Overall, these results suggest that model-based planning is preferable when accurate environment models are available, while model-free approaches remain essential for adaptability in unknown or complex settings.

# 7 References


Blackjack: MDP: Sutton & Barto (2018), Reinforcement Learning: An Introduction (2nd ed.), Example 5.3 "Blackjack," online book. Gym Environment: Blackjack-v1.

CartPole: MDP: Barto, Sutton & Anderson (1983), "Neuronlike adaptive elements that can solve difficult learning control problems," IEEE Transactions on Systems, Man, and Cybernetics 13(5):834–846, doi:10.1109/TSMC.1983.6313077. Gym Environment: CartPole-v1.

GeeksforGeeks. "Value Iteration vs. Policy Iteration." GeeksforGeeks, 9 Feb. 2024, www.geeksforgeeks.org/machine-learning/what-is-the-difference-between-value-iteration-and-policy-iteration/.

"Policy Iteration — Introduction to Reinforcement Learning." Gibberblot.github.io, gibberblot.github.io/rl-notes/single-agent/policy-iteration.html.

Arslán. "What Is the Difference between Value Iteration and Policy Iteration?" Stack Overflow, 22 May 2016, stackoverflow.com/questions/37370015/what-is-the-difference-between-value-iteration-and-policy-iteration.

"4.3 Value Iteration." Introduction to Artificial Intelligence, 2025, inst.eecs.berkeley.edu/ cs188/textbook/mdp/value-iteration.html.

GeeksforGeeks. "QLearning in Reinforcement Learning." GeeksforGeeks, 6 Feb. 2019, www.geeksforgeeks.org/machine-learning/q-learning-in-python/.

"What Is the Q Function and What Is the v Function in Reinforcement Learning?" Data Science Stack Exchange, datascience.stackexchange.com/questions/9832/what-is-the-q-function-and-what-is-the-v-function-in-reinforcement-learning.

Wikipedia Contributors. "State–Action–Reward–State–Action." Wikipedia, Wikimedia Foundation, 5 Dec. 2021, en.wikipedia.org/wiki/State-action-reward-state-action

GeeksforGeeks. "SARSA (StateActionRewardStateAction) in Reinforcement Learning." GeeksforGeeks, 14 June 2019, www.geeksforgeeks.org/machine-learning/sarsa-reinforcement-learning/.