

CSCI B659: Reinforcement Learning

Assignment 2: Everything Printout

LJ Huang

Spring 2025

Contents

1	Task 1: SARSA	2
1.1	Experimental Plots for Task 1	2
1.1.1	SARSA FrozenLake4	2
1.1.2	SARSA FrozenLake8	3
1.1.3	SARSA CartPole	4
1.2	Observations on Task 1	4
2	Task 2: Model-Based RL	5
2.1	Experimental Plots for Task 2	5
2.2	Observations on Task 2	6
3	Code Snapshot	7
3.1	SARSA Implementation	7
3.2	Model-Based RL Implementation	9
4	README File	15

1 Task 1: SARSA

In this section, we present our experimental results for Task 1, where we applied the SARSA algorithm to three distinct environments: FrozenLake4, FrozenLake8, and CartPole. Our goal is to demonstrate how SARSA performs across different reinforcement learning problems – from discrete navigation tasks in FrozenLake to the more dynamic control challenge of balancing a pole in CartPole.

1.1 Experimental Plots for Task 1

1.1.1 SARSA FrozenLake4

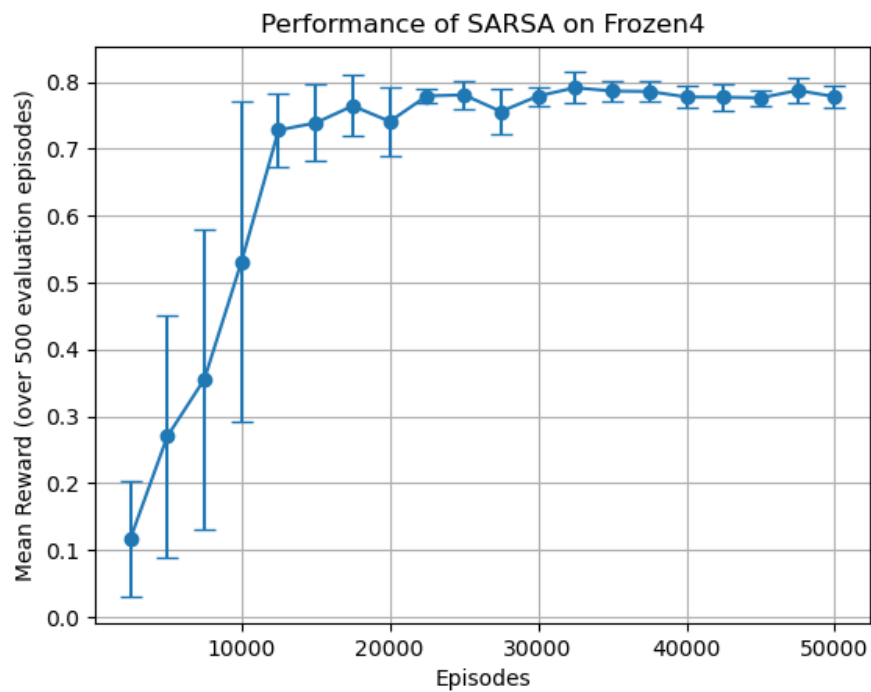


Figure 1: SARSA performance on FrozenLake4

1.1.2 SARSA FrozenLake8

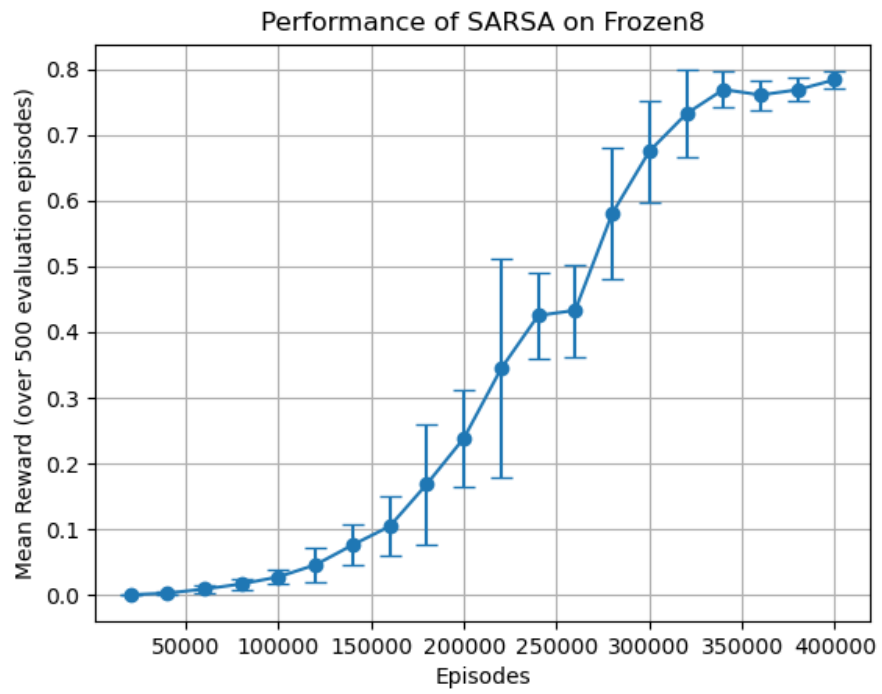


Figure 2: SARSA performance on FrozenLake8

1.1.3 SARSA CartPole

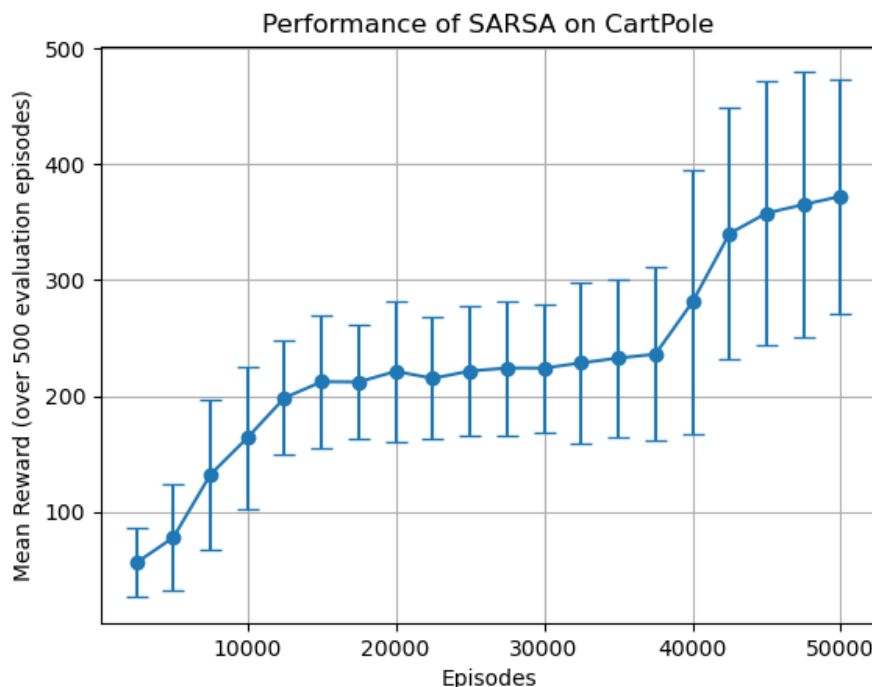


Figure 3: SARSA performance on CartPole

1.2 Observations on Task 1

FrozenLake4: SARSA converges within about 10,000 episodes to an average reward near 0.8, with error bars indicating stable performance after roughly 20,000–30,000 episodes.

FrozenLake8: Due to its larger state space, the agent requires hundreds of thousands of episodes to stabilize, eventually achieving mean rewards between 0.8 and 0.85 after extensive exploration.

CartPole: The mean reward steadily increases during the first 30,000 episodes and then accelerates, reaching values near 400–450 by 50,000 episodes, which reflects improved balance over time despite some variability.

Variant of SARSA Used: We employed an on-policy SARSA algorithm with an ϵ -greedy exploration strategy, where ϵ decays linearly from 0.5 to 0. Key hyperparameters are:

- Learning rate: $\alpha = 0.01$
- Discount factor: $\gamma = 0.999$

Sensitivity to Hyperparameters: SARSA is sensitive to these settings. A too-large α may cause unstable updates, while a too-small α can slow convergence. Similarly, if ϵ decays too quickly, the agent may exploit prematurely; if it remains high too long, convergence is delayed.

2 Task 2: Model-Based RL

2.1 Experimental Plots for Task 2

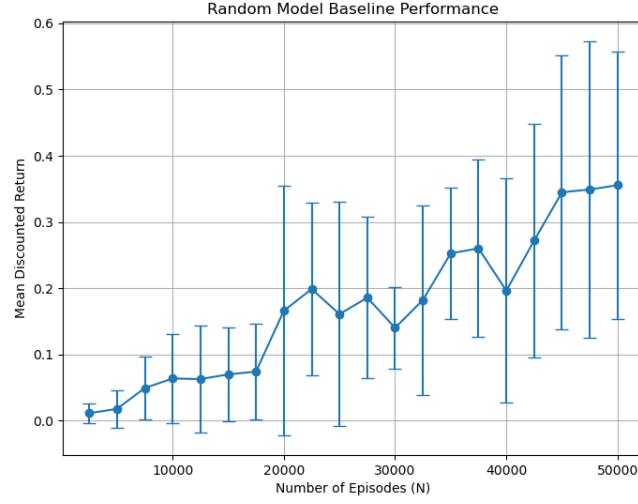
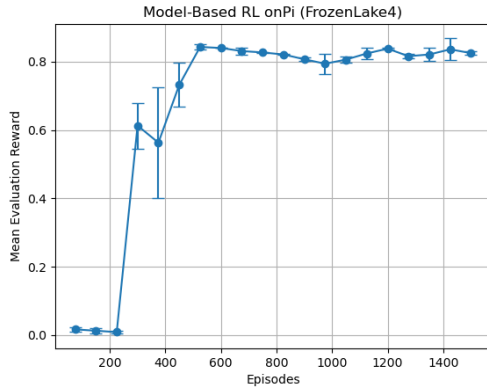
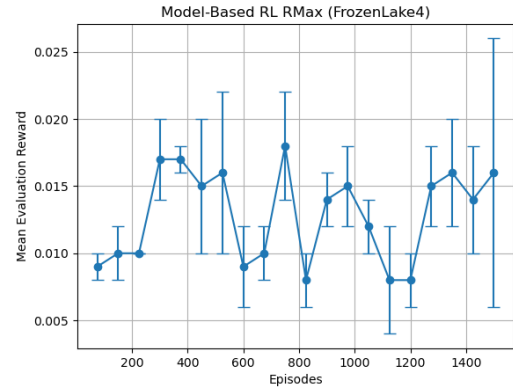


Figure 4: Random Model Baseline (FrozenLake)

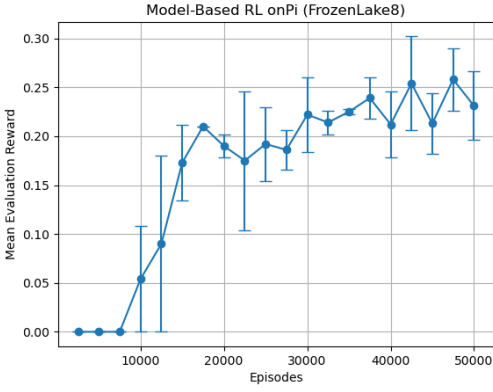


(a) onPi Variant on FrozenLake4

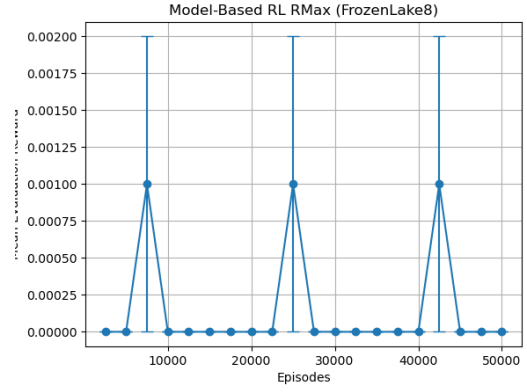


(b) RMax Variant on FrozenLake4

Figure 5: Results on FrozenLake4



(a) onPi Variant on FrozenLake8



(b) RMax Variant on FrozenLake8

Figure 6: Results on FrozenLake8

2.2 Observations on Task 2

Success of Each Algorithm:

- **Random:** Learns a workable policy eventually but is slow and requires large amounts of data.
- **onPi:** Achieves good performance on FrozenLake4 quickly and learns a reasonable policy on FrozenLake8, indicating faster convergence.
- **RMax:** RMax did not perform well in our experiments, possibly due to code implementation issues in `Model_Based_RL.py` or the need for additional parameter tuning.

Performance and Speed of Learning:

- **onPi** demonstrates the fastest and most stable learning, particularly in FrozenLake4.
- **Random** improves only with large data collection and is slower overall.
- **RMax** may have potential when correctly implemented and tuned but did not show a clear advantage in these experiments.

3 Code Snapshot

Snapshot of codes provided in this section used in this assignment.

3.1 SARSA Implementation

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from pp2starter import prepCartPole, prepFrozen4, prepFrozen8
4
5
6 def choose_action(Q, state, nA, epsilon):
7     if np.random.rand() < epsilon:
8         return np.random.choice(nA)
9     else:
10         best_actions = np.flatnonzero(Q[state] == np.max(Q[state]))
11         return np.random.choice(best_actions)
12
13 def sarsa(env, phi, nS, nA, episodes, max_steps, alpha, gamma, initial_epsilon
14         =0.5, initial_Q_value=0.0):
15     Q = np.full((nS, nA), initial_Q_value, dtype=np.float64)
16
17     for ep in range(episodes):
18         epsilon = initial_epsilon * (episodes - ep - 1) / (episodes - 1)
19
20         observation, info = env.reset()
21         state = phi(observation)
22         action = choose_action(Q, state, nA, epsilon)
23
24         for t in range(max_steps):
25             observation, reward, terminated, truncated, info = env.step(action)
26             next_state = phi(observation)
27             if terminated or truncated:
28                 delta = reward - Q[state, action]
29                 Q[state, action] += alpha * delta
30                 break
31             else:
32                 next_action = choose_action(Q, next_state, nA, epsilon)
33                 delta = reward + gamma * Q[next_state, next_action] - Q[state,
34                     action]
35                 Q[state, action] += alpha * delta
36                 state, action = next_state, next_action
37
38     return Q
39
40 def evaluate_policy(env, phi, Q, num_episodes=500):
41     rewards = []
42     for _ in range(num_episodes):
43         observation, info = env.reset()
44         state = phi(observation)
45         total_reward = 0
46         for _ in range(env._max_episode_steps):
47             best_actions = np.flatnonzero(Q[state] == np.max(Q[state]))
48             action = np.random.choice(best_actions)
49             observation, reward, terminated, truncated, info = env.step(action)
50             total_reward += reward
51             state = phi(observation)
52             if terminated or truncated:
53                 break
```

```

51         rewards.append(total_reward)
52     return np.mean(rewards)
53
54 def run_experiments(prepare_fn, total_episodes, eval_interval, n_repeats, alpha,
55                    gamma,
56                    initial_epsilon=0.5, initial_Q_value=0.0):
57     nS, nA, env, phi, dname = prepare_fn()
58     max_steps = env._max_episode_steps
59     eval_points = list(range(eval_interval, total_episodes + 1, eval_interval))
60     all_scores = np.zeros((n_repeats, len(eval_points)))
61
62     for rep in range(n_repeats):
63         print(f"Starting repeat {rep+1}/{n_repeats}...")
64         Q = np.full((nS, nA), initial_Q_value, dtype=np.float64)
65         eval_counter = 0
66         for ep in range(total_episodes):
67             epsilon = initial_epsilon * (total_episodes - ep - 1) / (
68                 total_episodes - 1)
69             observation, info = env.reset()
70             state = phi(observation)
71             action = choose_action(Q, state, nA, epsilon)
72
73             for t in range(max_steps):
74                 observation, reward, terminated, truncated, info = env.step(action)
75
76                 next_state = phi(observation)
77                 if terminated or truncated:
78                     delta = reward - Q[state, action]
79                     Q[state, action] += alpha * delta
80                     break
81                 else:
82                     next_action = choose_action(Q, next_state, nA, epsilon)
83                     delta = reward + gamma * Q[next_state, next_action] - Q[state,
84                         action]
85                     Q[state, action] += alpha * delta
86                     state, action = next_state, next_action
87
88             if (ep + 1) % eval_interval == 0:
89                 score = evaluate_policy(env, phi, Q)
90                 all_scores[rep, eval_counter] = score
91                 eval_counter += 1
92
93     env.close()
94
95     means = np.mean(all_scores, axis=0)
96     stds = np.std(all_scores, axis=0)
97     return dname, eval_points, means, stds
98
99 def plot_results(dname, eval_points, means, stds):
100     plt.figure()
101     plt.errorbar(eval_points, means, yerr=stds, fmt='-o', capsize=5)
102     plt.title(f'Performance of SARSA on {dname}')
103     plt.xlabel('Episodes')
104     plt.ylabel('Mean Reward (over 500 evaluation episodes)')
105     plt.grid(True)
106     plt.show()
107
108 if __name__ == '__main__':

```



```

106     alpha = 0.01
107     gamma = 0.999
108     n_repeats = 10
109     initial_epsilon = 0.5
110
111     total_episodes_f4 = 50000
112     eval_interval_f4 = total_episodes_f4 // 20
113     dname, eval_points, means, stds = run_experiments(prepareFrozen4,
114                                                       total_episodes_f4, eval_interval_f4,
115                                                       n_repeats, alpha, gamma,
116                                                       initial_epsilon=
117                                                       initial_epsilon,
118                                                       initial_Q_value=0.0)
119
120     plot_results(dname, eval_points, means, stds)
121
122     total_episodes_f8 = 400000
123     eval_interval_f8 = total_episodes_f8 // 20
124     dname, eval_points, means, stds = run_experiments(prepareFrozen8,
125                                                       total_episodes_f8, eval_interval_f8,
126                                                       n_repeats, alpha, gamma,
127                                                       initial_epsilon=
128                                                       initial_epsilon,
129                                                       initial_Q_value=0.0)
130
131     plot_results(dname, eval_points, means, stds)
132
133     total_episodes_cp = 50000
134     eval_interval_cp = total_episodes_cp // 20
135     dname, eval_points, means, stds = run_experiments(prepareCartPole,
136                                                       total_episodes_cp, eval_interval_cp,
137                                                       n_repeats, alpha, gamma,
138                                                       initial_epsilon=
139                                                       initial_epsilon,
140                                                       initial_Q_value=0.0)
141
142     plot_results(dname, eval_points, means, stds)

```

Listing 1: SARSA.py

3.2 Model-Based RL Implementation

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from pp1starter import prepareFrozen
4 from VI_PI_MPI import value_iteration, policy_evaluation
5 from pp2starter import prepareFrozen4, prepareFrozen8
6
7 def collect_data(env, N):
8     counts = {}
9     transition_counts = {}
10    reward_sums = {}
11
12    for episode in range(N):
13        observation, info = env.reset()
14        terminated = False
15        truncated = False
16        while not (terminated or truncated):
17            action = env.action_space.sample()
18            next_obs, reward, terminated, truncated, info = env.step(action)

```

```

19         key = (observation, action)
20         counts[key] = counts.get(key, 0) + 1
21         transition_key = (observation, action, next_obs)
22         transition_counts[transition_key] = transition_counts.get(
23             transition_key, 0) + 1
24         reward_sums[key] = reward_sums.get(key, 0) + reward
25         observation = next_obs
26     return counts, transition_counts, reward_sums
27
28 def estimate_model(counts, transition_counts, reward_sums, nS, nA):
29     P_est = []
30     for s in range(nS):
31         action_list = []
32         for a in range(nA):
33             action_list.append(None)
34         P_est.append(action_list)
35
36     for s in range(nS):
37         for a in range(nA):
38             key = (s, a)
39             total = counts.get(key, 0)
40             outcomes = []
41             if total > 0:
42                 s_primes = set()
43                 for s0, a0, s_ in transition_counts.keys():
44                     if s0 == s and a0 == a:
45                         s_primes.add(s_)
46                 for s_prime in s_primes:
47                     transition_key = (s, a, s_prime)
48                     count_sas = transition_counts.get(transition_key, 0)
49                     p_est = count_sas / total
50                     r_est = reward_sums.get(key, 0) / total
51                     outcomes.append((p_est, s_prime, r_est, False))
52             P_est[s][a] = outcomes
53         else:
54             P_est[s][a] = [(1.0, s, 0.0, False)]
55     return P_est
56
57 def run_random_experiment(env, nS, nA, discount, tolerance, N_values,
58     num_experiments=10):
59     results = {N: [] for N in N_values}
60     for N in N_values:
61         print(f"[Random] Data collection: N = {N}")
62         for exp in range(num_experiments):
63             counts, transition_counts, reward_sums = collect_data(env, N)
64             P_est = estimate_model(counts, transition_counts, reward_sums, nS, nA)
65             policy_est, _, _ = value_iteration(P_est, nS, nA, discount,
66                 tolerance, max_iter=500, env=None)
67             score = policy_evaluation(env, policy_est, discount, episodes=500)
68             results[N].append(score)
69             print(f"    Experiment {exp+1}: Score = {score:.4f}")
70     return results
71
72 def choose_action(Q, state, nA, epsilon):
73     if np.random.rand() < epsilon:
74         return np.random.choice(nA)
75     else:
76         best_actions = np.flatnonzero(Q[state] == np.max(Q[state]))

```

```

75         return np.random.choice(best_actions)
76
77 def build_model(data, nS, nA, variant='standard', M=10, R=1):
78     counts = np.zeros((nS, nA, nS))
79     rewards = np.zeros((nS, nA, nS))
80     sa_counts = np.zeros((nS, nA))
81     dead_end = np.zeros(nS, dtype=bool)
82
83     for (s, a, r, s_next, terminated) in data:
84         if s < nS and s_next < nS:
85             counts[s, a, s_next] += 1
86             rewards[s, a, s_next] += r
87             sa_counts[s, a] += 1
88             if terminated:
89                 dead_end[s_next] = True
90
91     if variant == 'standard' or variant == 'onPi':
92         P = np.zeros((nS, nA, nS))
93         R_model = np.zeros((nS, nA))
94         for s in range(nS):
95             for a in range(nA):
96                 if sa_counts[s, a] > 0:
97                     P[s, a, :] = counts[s, a, :] / sa_counts[s, a]
98                     R_model[s, a] = np.sum(rewards[s, a, :]) / sa_counts[s, a]
99                 else:
100                     P[s, a, s] = 1.0
101                     R_model[s, a] = 0.0
102         return P, R_model
103     elif variant == 'RMax':
104         nS_new = nS + 1
105         P = np.zeros((nS_new, nA, nS_new))
106         R_model = np.zeros((nS_new, nA))
107         for s in range(nS):
108             for a in range(nA):
109                 if sa_counts[s, a] >= M:
110                     P[s, a, :nS] = counts[s, a, :] / sa_counts[s, a]
111                     R_model[s, a] = np.sum(rewards[s, a, :]) / sa_counts[s, a]
112                 else:
113                     if not dead_end[s]:
114                         P[s, a, nS] = 1.0
115                         R_model[s, a] = R
116                     else:
117                         P[s, a, s] = 1.0
118                         R_model[s, a] = 0.0
119             for a in range(nA):
120                 P[nS, a, nS] = 1.0
121                 R_model[nS, a] = 0.0
122         return P, R_model
123
124 def value_iteration_model(P, R_model, gamma, threshold=1e-6, max_iter=10000):
125     nS, nA, _ = P.shape
126     V = np.zeros(nS)
127     for _ in range(max_iter):
128         V_prev = V.copy()
129         for s in range(nS):
130             Q_s = np.zeros(nA)
131             for a in range(nA):
132                 Q_s[a] = R_model[s, a] + gamma * np.sum(P[s, a, :] * V_prev)
133             V[s] = np.max(Q_s)

```

```

134         if np.max(np.abs(V - V_prev)) < threshold:
135             break
136     Q = np.zeros((nS, nA))
137     for s in range(nS):
138         for a in range(nA):
139             Q[s, a] = R_model[s, a] + gamma * np.sum(P[s, a, :] * V)
140     return V, Q
141
142 def evaluate_policy_model(env, phi, Q, num_episodes=500):
143     rewards_eval = []
144     for _ in range(num_episodes):
145         observation, info = env.reset()
146         state = phi(observation)
147         total_reward = 0
148         done = False
149         while not done:
150             best_actions = np.flatnonzero(Q[state] == np.max(Q[state]))
151             action = np.random.choice(best_actions)
152             observation, reward, terminated, truncated, info = env.step(action)
153             total_reward += reward
154             state = phi(observation)
155             done = terminated or truncated
156         rewards_eval.append(total_reward)
157     return np.mean(rewards_eval)
158
159 def model_based_rl(prepare_fn, total_episodes, eval_interval, n_repeats,
160                   gamma=0.999, initial_epsilon=0.5, variant='onPi', M=10, R=1):
161     nS, nA, env, phi, dtype = prepare_fn()
162     quality_all = np.zeros((n_repeats, total_episodes // eval_interval))
163
164     for rep in range(n_repeats):
165         print(f"[{variant}] Starting repeat {rep+1}/{n_repeats}...")
166         data = []
167         Q = np.zeros((nS, nA))
168         eval_counter = 0
169         for ep in range(total_episodes):
170             if (ep + 1) % eval_interval == 0:
171                 P_eval, R_eval = build_model(data, nS, nA, variant='standard')
172                 _, Q_eval = value_iteration_model(P_eval, R_eval, gamma)
173                 quality = evaluate_policy_model(env, phi, Q_eval)
174                 quality_all[rep, eval_counter] = quality
175                 eval_counter += 1
176                 print(f" Episode {ep+1}/{total_episodes}: Eval quality = {quality
177                       :.2f}")
178                 observation, info = env.reset()
179                 state = phi(observation)
180                 done = False
181                 while not done:
182                     if variant == 'onPi':
183                         action = choose_action(Q, state, nA, epsilon=initial_epsilon)
184                     elif variant == 'RMax':
185                         action = np.argmax(Q[state])
186                     else:
187                         raise ValueError("Unknown variant")
188                     observation, reward, terminated, truncated, info = env.step(action)
189                     next_state = phi(observation)
190                     data.append((state, action, reward, next_state, terminated))
191                     state = next_state

```

```

191         done = terminated or truncated
192     env.close()
193     return dname, quality_all
194
195 def plot_results(eval_points, quality_all, title):
196     mean_quality = np.mean(quality_all, axis=0)
197     std_quality = np.std(quality_all, axis=0)
198     plt.figure()
199     plt.errorbar(eval_points, mean_quality, yerr=std_quality, fmt='-o', capsize=5)
200     plt.title(title)
201     plt.xlabel('Episodes')
202     plt.ylabel('Mean Evaluation Reward')
203     plt.grid(True)
204     plt.show()
205
206
207 if __name__ == '__main__':
208
209     env_random, P_random, nS, nA, dname_random = prepFrozen()
210     tolerance = 0.001
211     discount = 0.999
212     N_values = range(2500, 50001, 2500)
213     random_results = run_random_experiment(env_random, nS, nA, discount, tolerance
214         , N_values, num_experiments=10)
215
216     N_means = []
217     N_stds = []
218     for N in N_values:
219         scores = np.array(random_results[N])
220         N_means.append(np.mean(scores))
221         N_stds.append(np.std(scores))
222     plt.figure(figsize=(8, 6))
223     plt.errorbar(list(N_values), N_means, yerr=N_stds, fmt='o-', capsize=5)
224     plt.xlabel('Number of Episodes (N)')
225     plt.ylabel('Mean Discounted Return')
226     plt.title('Random Model Baseline Performance')
227     plt.grid(True)
228     plt.show()
229
230     total_episodes_f4 = 1500
231     eval_interval_f4 = total_episodes_f4 // 20
232     dname, quality_onPi = model_based_rl(prepareFrozen4, total_episodes_f4,
233         eval_interval_f4,
234         n_repeats=2, gamma=discount,
235         initial_epsilon=0.5, variant='onPi')
236     eval_points_f4 = list(range(eval_interval_f4, total_episodes_f4+1,
237         eval_interval_f4))
238     plot_results(eval_points_f4, quality_onPi, "Model-Based RL onPi (FrozenLake4)"
239         )
240
241     dname, quality_RMax = model_based_rl(prepareFrozen4, total_episodes_f4,
242         eval_interval_f4,
243         n_repeats=2, gamma=discount,
244         initial_epsilon=0.5, variant='RMax', M
245         =10, R=1)
246     plot_results(eval_points_f4, quality_RMax, "Model-Based RL RMax (FrozenLake4)"
247         )

```

```

243 total_episodes_f8 = 50000
244 eval_interval_f8 = total_episodes_f8 // 20
245 dname, quality_onPi_f8 = model_based_rl(prepareFrozen8, total_episodes_f8,
246     eval_interval_f8,
247         n_repeats=2, gamma=discount,
248         initial_epsilon=0.5, variant='onPi')
249 eval_points_f8 = list(range(eval_interval_f8, total_episodes_f8+1,
250     eval_interval_f8))
251 plot_results(eval_points_f8, quality_onPi_f8, "Model-Based RL onPi (
252     FrozenLake8)")
253
254 dname, quality_RMax_f8 = model_based_rl(prepareFrozen8, total_episodes_f8,
255     eval_interval_f8,
256         n_repeats=2, gamma=discount,
257         initial_epsilon=0.5, variant='RMax', M
258         =10, R=1)
259 plot_results(eval_points_f8, quality_RMax_f8, "Model-Based RL RMax (
260     FrozenLake8)")

```

Listing 2: Model_Based_RL.py

4 README File

```
1 # CSCI B659: Reinforcement Learning - Assignment 2
2 **Author:** LJ Huang
3 **Semester:** Spring 2025
4
5 ## Overview
6 This repository contains the code and report for Assignment 2. The assignment
   covers two main tasks:
7 1. **Task 1:** SARSA implementation in FrozenLake4, FrozenLake8, and CartPole.
8 2. **Task 2:** Model-based RL in FrozenLake4 and FrozenLake8 environments.
9
10 ## Directory Structure
11 - **VI_PI_MPI.py**
12   Contains the implementation for Task 1.
13 - **Model_Based_RL.py**
14   Contains the implementation for Task 2.
15 - **pp2starter.py**
16   The provided startup file for setting up the FrozenLake environment (including
   rewards, number of states, and actions).
17 - **hw2_report.pdf**
18   The report containing code printouts, experimental results, plots, and
   discussion of the findings.
19 - **README.md**
20   This file.
21
22 ## Dependencies
23 - Python 3.x
24 - Gymnasium (Install with 'pip install gymnasium')
25 - NumPy (Install with 'pip install numpy')
26 - Matplotlib (Install with 'pip install matplotlib')
27
28 ## Installation and Running
29 1. **Clone or Download the Repository:**
30   Clone the repository or download the zip file and extract its contents.
31
32 2. **Run the Code:**
33   \begin{verbatim}
34   python SARSA.py           # Task 1
35   python Model_Based_RL.py  # Task 2
36   \end{verbatim}
```

Listing 3: README.file