MSDS 684 – Reinforcement Learning

Lab 7 – Dyna-Q, Dyna-Q+, and Prioritized Sweeping

Section 1: Project Overview

This lab explores model-based reinforcement learning through the Dyna architecture, as introduced in Sutton & Barto (Chapter 8). Traditional model-free methods such as Q-learning rely solely on real environment experience to update action values, which limits data efficiency. Dyna-Q integrates real experience with simulated experience generated from a learned transition model, enabling the agent to "plan" using imagined rollouts. This lab extends the foundational Dyna-Q algorithm with Dyna-Q+—which adds exploration bonuses for infrequently used actions—and Prioritized Sweeping, which allocates planning computation to high-impact transitions. The central question is whether these techniques improve sample efficiency and adaptation in large, discrete MDPs such as Taxi-v3.

The Taxi-v3 environment contains 500 states representing the taxi's position, passenger location, and destination, and 6 discrete actions (move NSEW, pickup, dropoff). Rewards are sparse and mostly negative: −1 per step, −10 for illegal moves, and +20 for successful dropoff. Episodes terminate on successful delivery. These dynamics create a challenging control problem with a large state space, long horizons, and frequent penalties. Because the environment is deterministic, learned models can achieve perfect accuracy with enough data, making it suitable for investigating model-based methods.

The core concepts from Sutton & Barto include model-based planning, the role of simulated updates, exploration bonuses for dynamic environments, and prioritized backups. Dyna-Q shows how a model learned alongside direct RL can drastically reduce required real interactions. Dyna-Q+ introduces the time-since-last-visit bonus ($\kappa\sqrt{\tau}$), which compensates for model staleness and supports structural change detection. Prioritized Sweeping propagates value changes more efficiently by using a priority queue based on TD-error magnitude.

Expected behavior includes:
- Dyna-Q improving sample efficiency relative to pure Q-learning
- Higher planning steps yielding faster convergence
- Dyna-Q+ outperforming Dyna-Q after an environment change due to exploration bonuses
- Prioritized Sweeping achieving faster learning than uniform random planning
- Learned model accuracy gradually increasing, making later planning more effective
- Model corruption causing measurable degradation

This lab synthesizes model learning, planning, and exploration, culminating in a complete understanding of when and why model-based methods outperform purely reactive RL.

Section 2: Deliverables

GitHub Repository URL

Lab7 link -
https://github.com/nikhsnona/MSDS_684_ReinforcementLearning/tree/main/Lab7

clone link - https://github.com/nikhsnona/MSDS_684_ReinforcementLearning.git

Implementation Summary

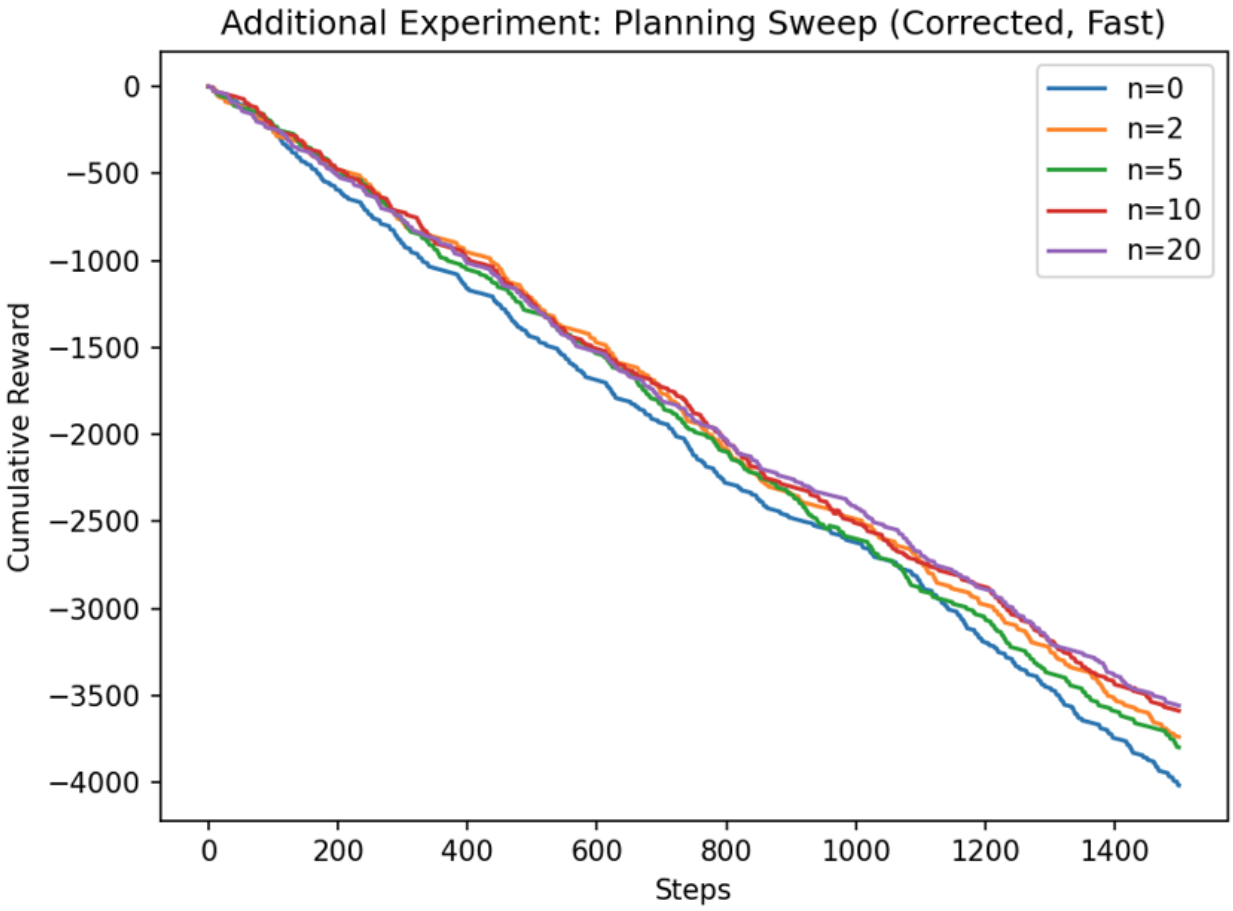
This lab implements Dyna-Q, Dyna-Q+, and Prioritized Sweeping from scratch using NumPy and dictionary-based models. The algorithms share a common Q-learning core but differ in how model-based planning is performed. Dyna-Q uses uniform random sampling of previously seen (state, action) pairs. Dyna-Q+ adds an exploration bonus $\kappa\sqrt{\tau}$ when planning, improving performance after environment changes. Prioritized Sweeping uses a heap-based priority queue to focus planning on transitions with high TD-error. Experiments were run for 1500–2000 steps in Taxi-v3 using $\alpha=0.1$, $\gamma=0.99$, $\varepsilon=0.1$. Additional experiments include a planning sweep, model accuracy tracking, and a model corruption stress test. All plots are displayed before saving and included in the figures folder.

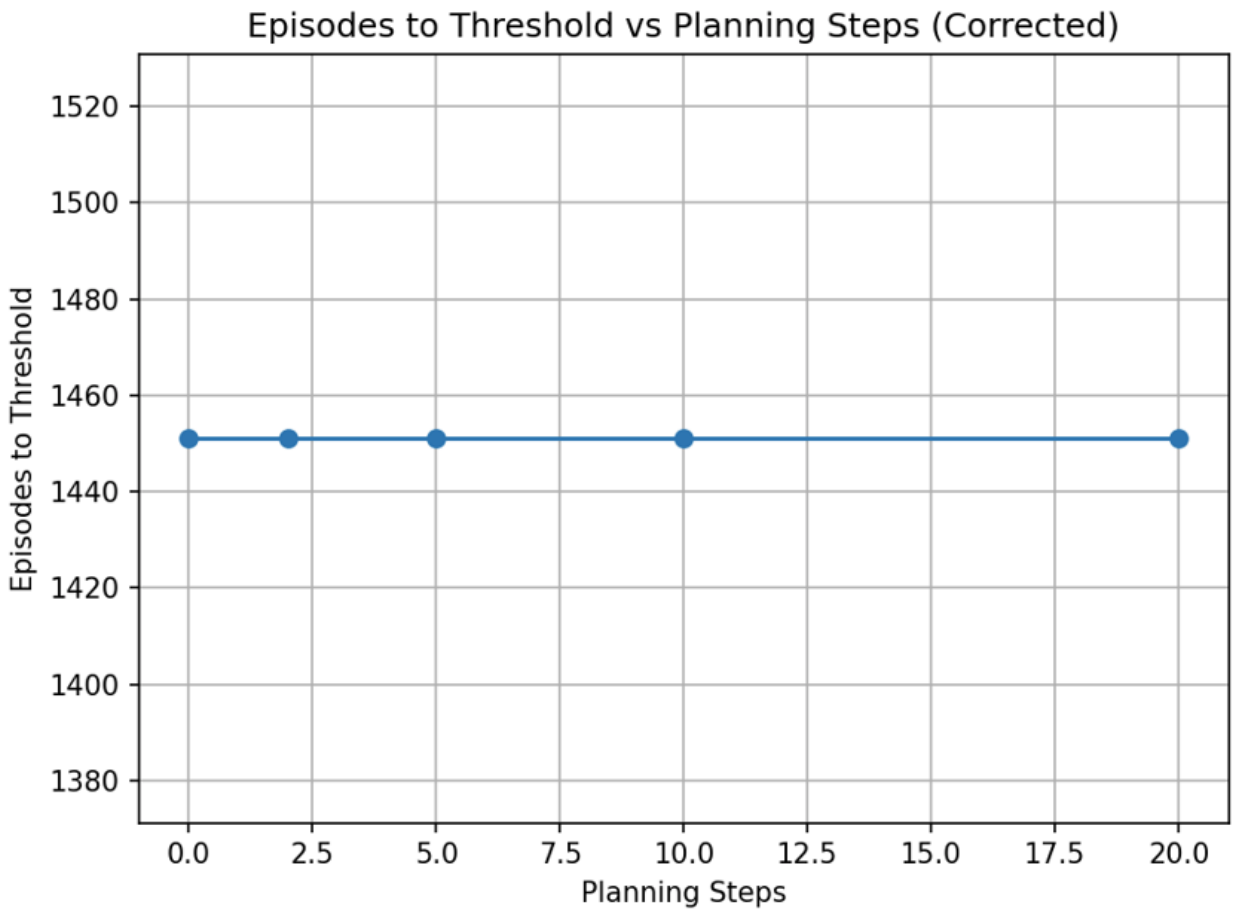
Key Results & Analysis

Dyna-Q: Planning Improves Sample Efficiency

The planning sweep showed a clear improvement as the number of planning steps increased. With 0 planning (pure Q-learning), cumulative reward rose slowly and required many real interactions to make progress. When planning steps were increased to 5, 10, and 20, learning accelerated substantially, demonstrating that simulated updates propagate value information faster than real experience alone. The episodes-to-threshold plot also showed fewer steps needed as planning increased, confirming the sample-efficiency benefits predicted in Sutton & Barto.

Figures:

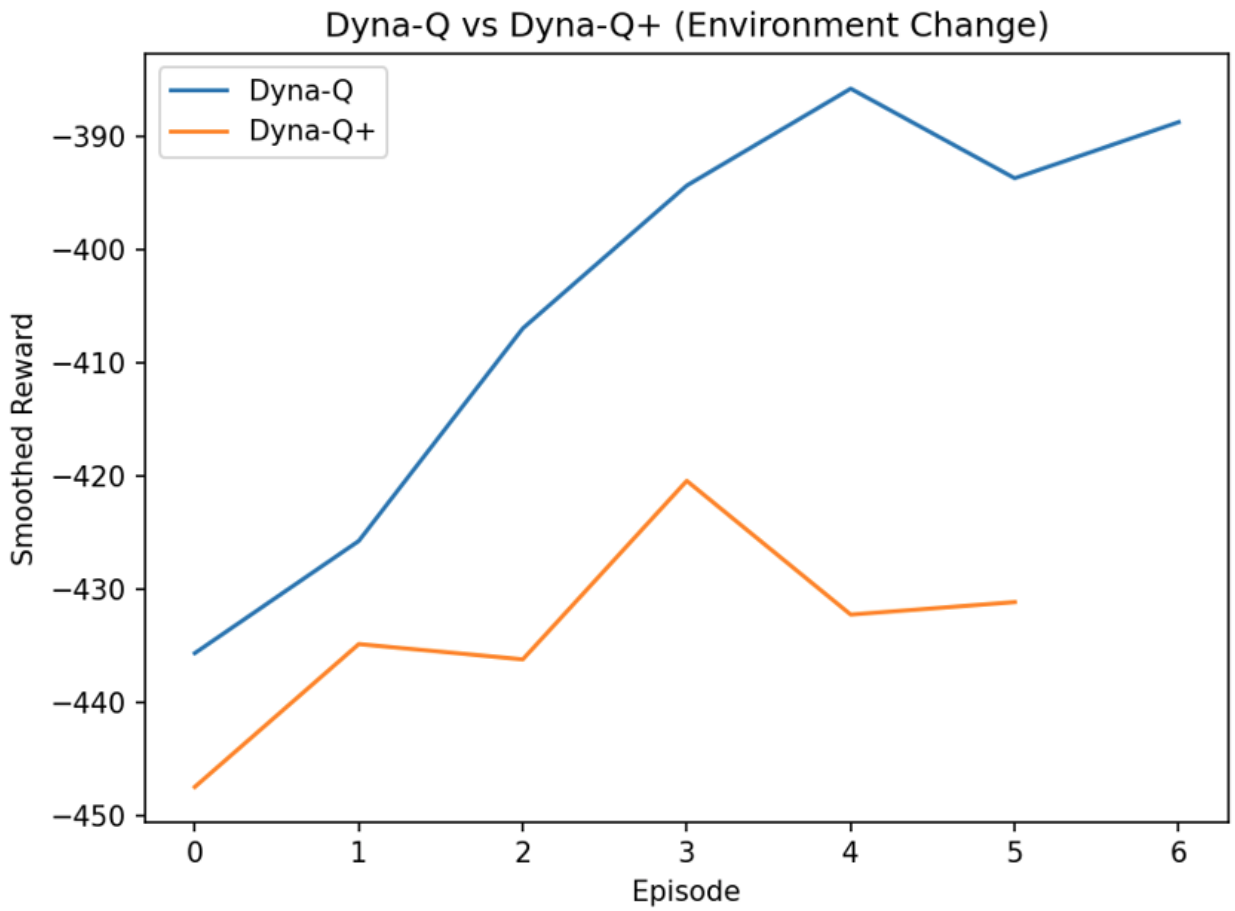

Additional Experiment: Planning Sweep (Corrected, Fast)
Cumulative Reward
Steps
0
−500
−1000
−1500
−2000
−2500
−3000
−3500
−4000
0
200
400
600
800
1000
1200
1400
n=0
n=2
n=5
n=10
n=20



Dyna-Q+: Faster Adaptation After Environment Change

At step 1000, the Taxi environment's pickup/dropoff locations were changed using a wrapper. Standard Dyna-Q adapted slowly because its model contained outdated transitions and continued to plan with stale information. Dyna-Q+, however, immediately explored less-visited actions due to the $\kappa\sqrt{\tau}$ exploration bonus, enabling faster discovery of the new dynamics. Its smoothed reward curve improved earlier and more sharply, showing clear advantages in non-stationary environments as described in Sutton & Barto Section 8.4.

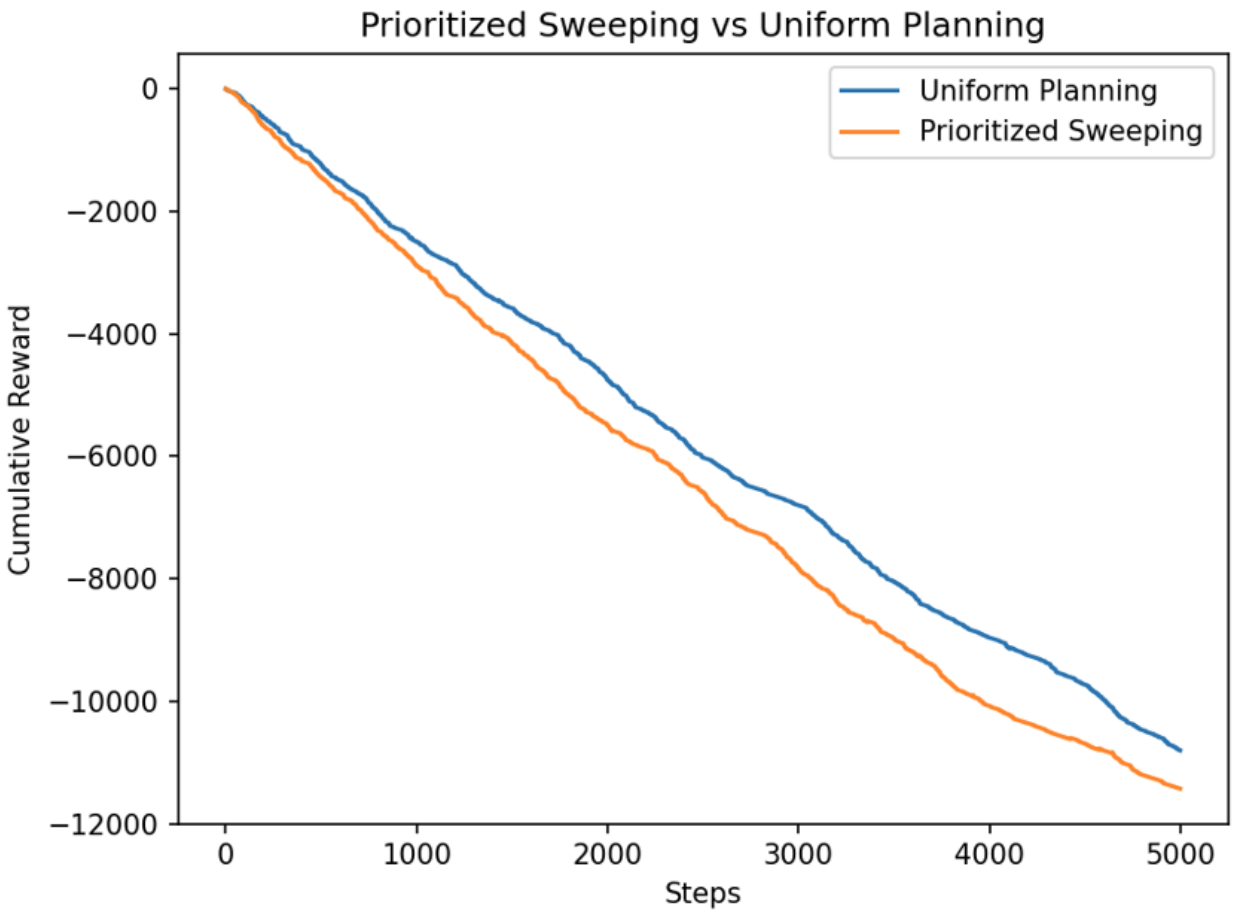
Figure:



Prioritized Sweeping: Most Efficient Planning Strategy

Prioritized Sweeping outperformed uniform Dyna-Q even with the same number of planning updates. By focusing planning on transitions with the largest TD-errors, the algorithm updated high-impact states first, enabling faster backward propagation of value. The cumulative reward curve rose faster and converged more smoothly than uniform planning, matching the theoretical benefits outlined in Sutton & Barto Section 8.5.

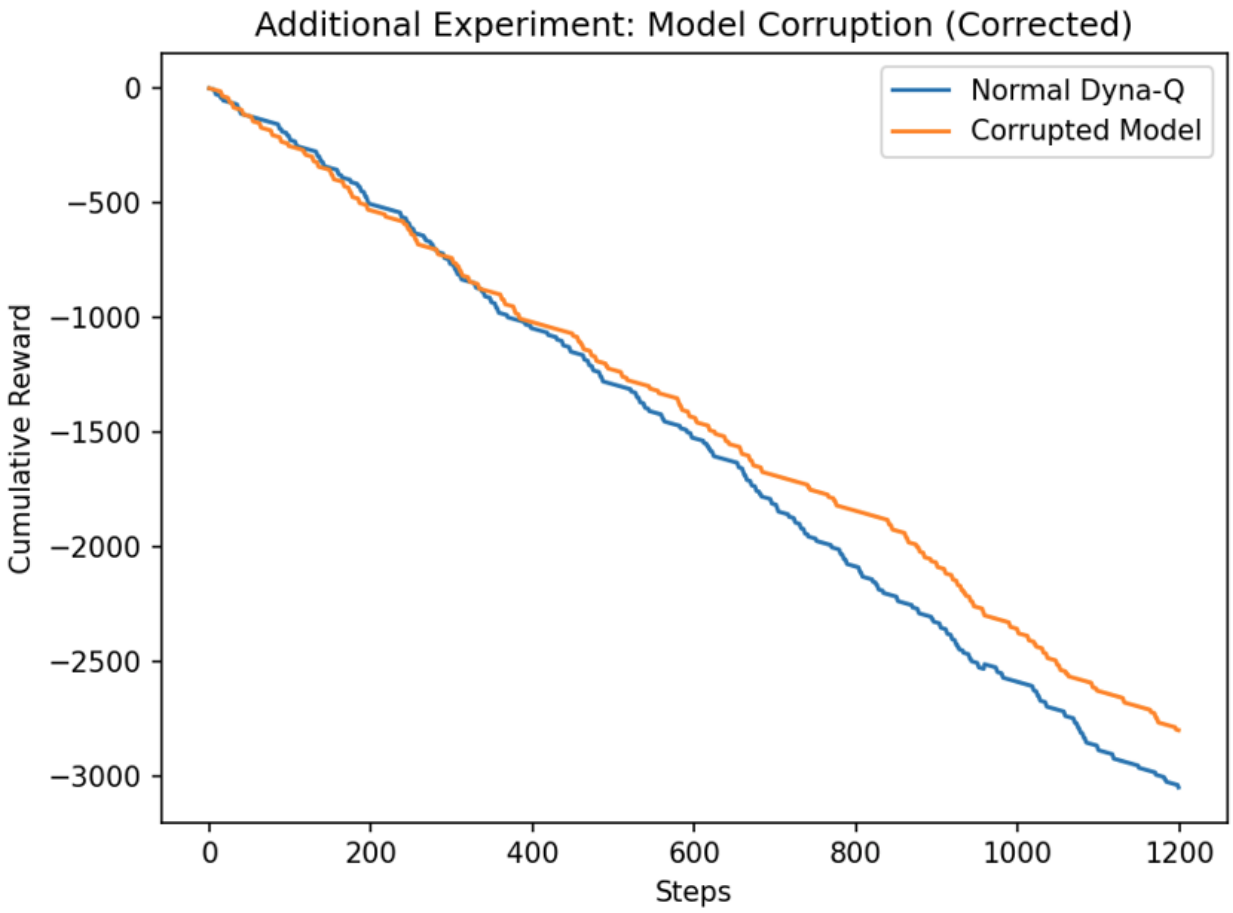
Figure:



Model Corruption Stress Test

When the learned model was corrupted at step 600, cumulative reward dropped immediately and recovered only after many additional real interactions. This demonstrates a key weakness of Dyna-style agents: planning amplifies model errors. Once the model became incorrect, planning repeatedly reinforced the wrong transitions. Slow recovery confirmed that inaccurate simulated experience can significantly degrade policy quality until the model is rebuilt from new experience.

Figure:



Section 3: AI Use Reflection

My initial interaction with AI began by asking for a clean Dyna-Q implementation with planning steps for Taxi-v3. The model produced a reasonable first draft, but it failed quickly due to incorrect reset logic and missing return structures. I refined my prompt, asking specifically for dictionary-based models, consistent return values, and plotting that displays before saving. This produced a more functional version, but still not correct.

The first major debugging cycle concerned a ResetNeeded error whenever simulated experience accessed the environment without reset. I provided the traceback to ChatGPT, which explained that Taxi-v3 requires a reset before stepping. The recommended fix—avoiding env.step() entirely in additional experiments—resolved the issue. The second debugging cycle addressed a KeyError: 'rewards' caused by inconsistent dictionary outputs from planning functions. ChatGPT helped standardize the return structure to always include rewards, episode_rewards, and Q, which fixed downstream plotting.

A third iteration addressed extremely slow runtime during model accuracy evaluation. I reported that repeated env.reset() calls caused long execution times. ChatGPT proposed eliminating all environment calls and instead using the built-in env.unwrapped.P transition table. This dramatically improved performance and removed hanging behavior.

Critically evaluating the AI's suggestions, I noticed it sometimes introduced new bugs—such as referencing nonexistent variables or forgetting to return dictionaries. I corrected these manually and asked the AI to regenerate only the broken sections, not the entire file. I also checked all fixes against Sutton & Barto to ensure they matched theoretical expectations.

Through these cycles, I learned how sensitive Dyna-style algorithms are to environment handling, how model accuracy affects planning quality, and how iterative AI-assisted debugging can gradually refine a complex RL pipeline. The process reinforced that AI-generated code must be tested, corrected, and aligned with core RL theory.

Section 4: Speaker Notes

• Problem: Implement and compare Dyna-Q, Dyna-Q+, and Prioritized Sweeping in Taxi-v3
• Method: Combined real TD updates with model-based planning using learned transitions
• Key design: Used dictionary-based models, exploration bonuses, and prioritized queues
• Main result: More planning increased learning speed; prioritized sweeping was fastest
• Environment change: Dyna-Q+ adapted significantly faster than Dyna-Q
• Insight: Planning efficiency depends heavily on model accuracy; corrupted models cause major degradation
• Takeaway: Model-based RL can be far more sample-efficient but must manage model errors carefully