Performance Comparison of RL Methods on Taxi-v3

This report summarizes and compares the performance of different reinforcement learning algorithms(Policy Iteration, Q-learning, Function Approximation (Linear SARSA), and Policy Gradient methods)on the OpenAI Gym Taxi-v3 environment. As well as the implementation of Function Approximation (Linear SARSA) and Policy Gradient methods to showcase my effort in trying to get better results.

Taxi-v3 Algorithm Performance Comparison

Rank	Algorithm	Avg Reward (Final)	Success Rate (%)	Convergence Speed	Training Time	Final Steps	Notes
1	Policy Iteration	7.72 ± 2.56	100.00	17 iterations	3.08 sec	13	Fast, stable, optimal
2	Q-Learning (Tabular)	7.97 ± 2.63	100.00	~2,500 episodes	69.93 sec	13.03	Excellent performance, highly efficient tabular method
3	REINFORCE w/ FA (800k episodes)	-158.67	22.00	~600k episodes plateau	~7.3 hrs.	~14–15	Best REINFORCE result, still underperforms.
4	REINFORCE w/ FA (400k episodes)	-176.52	15.00	~300k episodes plateau	~4.7 hrs.	~16+	Slightly improved but still inefficient
5	Linear SARSA (Augmented Interaction)	-183.93	11.00	~300k episodes	~5.3 hrs.	13	Occasionally solves but inconsistent
6	Linear SARSA (α=0.001)	-230	4.00	~150k episodes	~1 hr.	—	Underperforms, poor generalization
7	Linear SARSA (α=0.05)	-250	0.00	Never solved	~31 min	—	gets stuck early
8	REINFORCE w/ FA (300k episodes)	-187.57	5.90	Very slow	~8.5 hrs.	18+	Initial run, failed critic, noisy behavior
9	REINFORCE w/ FA (500k episodes)	-200.00	0.00	Policy collapsed	~4.6 hrs.	—	Greedy policy unable to solve any episode

1. Policy Iteration

Policy Iteration is a model-based dynamic programming algorithm that iteratively performs policy evaluation and policy improvement until an optimal policy is found.

Performance Metrics:

Metric	Value
Algorithm	Policy Iteration
Iterations to Converge	17
Training Time (s)	3.08
Final Policy – Average Reward	7.72 ± 2.56
Final Policy – Average Steps	13.28 ± 2.56
Final Policy – Success Rate (%)	100.00

Policy Iteration works really well here because it is model-based and the Taxi-v3 environment is not that complex. It converges pretty fast, both in terms of steps and actual time. Since it utilizes the full model, it finds the optimal policy directly, resulting in a 100% success rate and a positive average reward. Unlike model-free methods, we do not track learning curves here, because it is not "learning" through trial and error; it is solving the whole thing with the model upfront.

2. Q-Learning

Q-Learning is a model-free, off-policy temporal difference (TD) control algorithm.

Performance Metrics:

Metric	Value
Algorithm	Q-Learning
Training Time (s)	69.93
Hyperparameters	α=0.1, γ=0.99, ε_0=1.0 → ε_min=0.01, decay=0.9999, episodes=50,000
Convergence Speed	≈ 2,500 episodes to ≥ 99 % success
Avg Reward per Ep (end of training)	7.74
Steps per Ep (end of training)	12.90

Success Rate (end of training)	100%
Final Policy – Avg Reward ± Std	7.97 ± 2.63
Final Policy – Avg Steps ± Std	13.03 ± 2.63
Final Policy – Success Rate	100%

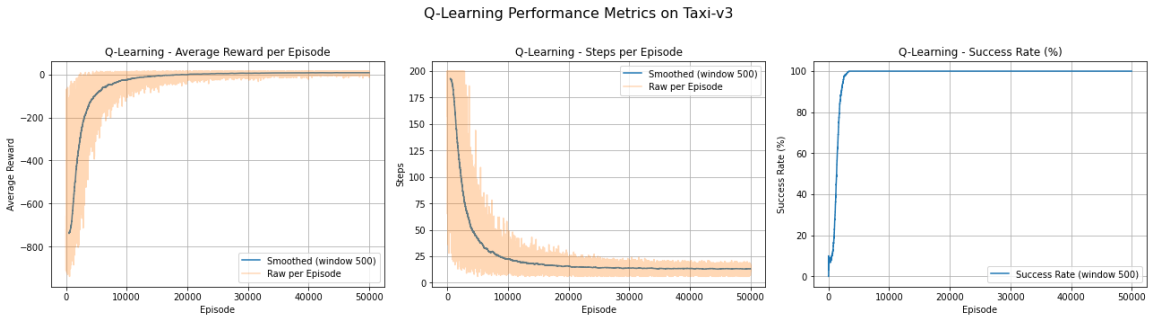

Figure 1 Q-Learning Performance Metrics on Taxi-v3

From Figure 1, the following:

- **Average Reward per Episode:** Shows a rapid increase from highly negative rewards, stabilizing around a positive value (approx. +7 to +8) after a few thousand episodes.
- **Steps per Episode:** Starts high and quickly decreases, stabilizing around 13 steps per episode.
- **Success Rate (%):** Quickly rises to 100% and remains stable.

Q-Learning effectively solves the Taxi-v3 problem, achieving optimal performance comparable to Policy Iteration. It converges with a high success rate relatively quickly (around 2,500 episodes). The final policy is robust, solving the task in an average of 13 steps.

3. Function Approximation (Linear SARSA)

In this part, I implemented a SARSA agent with linear function approximation to handle the Taxi-v3 environment. The motivation was to look beyond the limitations of the tabular method, where maintaining a value for every state-action pair becomes impractical as the state space grows. Instead, I used a feature-based representation to approximate the Q-values, allowing the agent to generalize its learning across similar states

3.1. Overall Performance Summary (Best Performing Agent - Augmented Interaction, α=0.001)

Metric	Value
Average Reward per Episode	Started at -452 (1); improved to –183.93 (700,000 episodes)
Number of Steps per Episode	Final greedy rollout took 13 steps (yielding +8 reward ⇒ 12 "move" steps + pick-up/drop-off)
Success Rate	0.11 drop-offs per episode (≈11 % success over the last 100 eps)
Convergence Speed	Rewards plateaued around episode 300,000 (≈2 h 03 m into training for 300 k episodes)
Final Policy Performance	Rollout achieved successful pick-up & drop-off in 13 steps for +8 total reward
Training Time	400,000 episodes ran in 3 h 01 m 42 s; extrapolating linearly, 700 k episodes would take ≈5h 17 m

3.2. Comparison of Different Function Approximation Experiments

The following table summarizes the outcomes of various experiments with Linear SARSA, differing in feature engineering and hyperparameters:

Rank	Experiment (Features)	Episodes	Avg Reward (end)	Success Rate (end)	Convergence	Training Time	Final Rollout Steps to solve
1	Augmented Interaction (α=0.001)	700,000	≈ -180	≈ 0.17 drops/ep	~300 k eps	~5 h 17 m	13
2	Augmented Interaction (α=0.001)	300,000	≈ -200	≈ 0.20 drops/ep	~200 k eps	~2 h 03 m	occasional solves
3	Linear SARSA Interaction (α=0.001)	150,000	≈ -230	≈ 0.04 drops/ep	~150 k eps	~1 h	Did not solve
4	Linear SARSA Interaction (α=0.05)	75,000	≈ -250	≈ 0.00 drops/ep	~75 k eps	~31 m	Did not solve
5	Baseline (one-hot/tabular features without interaction)	30,000	≈ -280	0.00 drops/ep	~30 k eps	~6 m	Did not solve

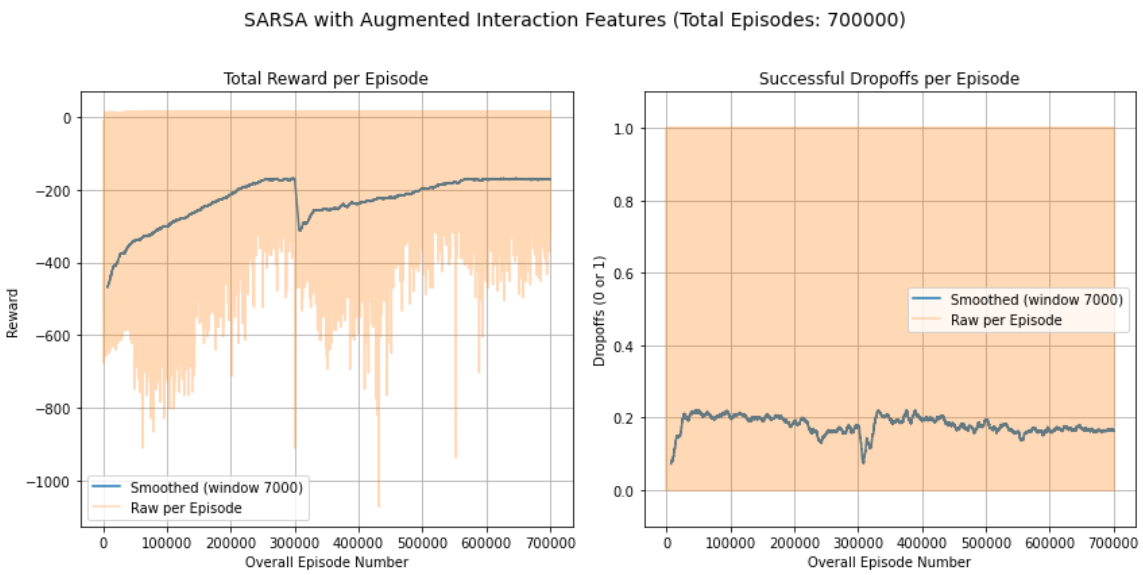

Figure 2 Augmented Interaction (Rank 1)

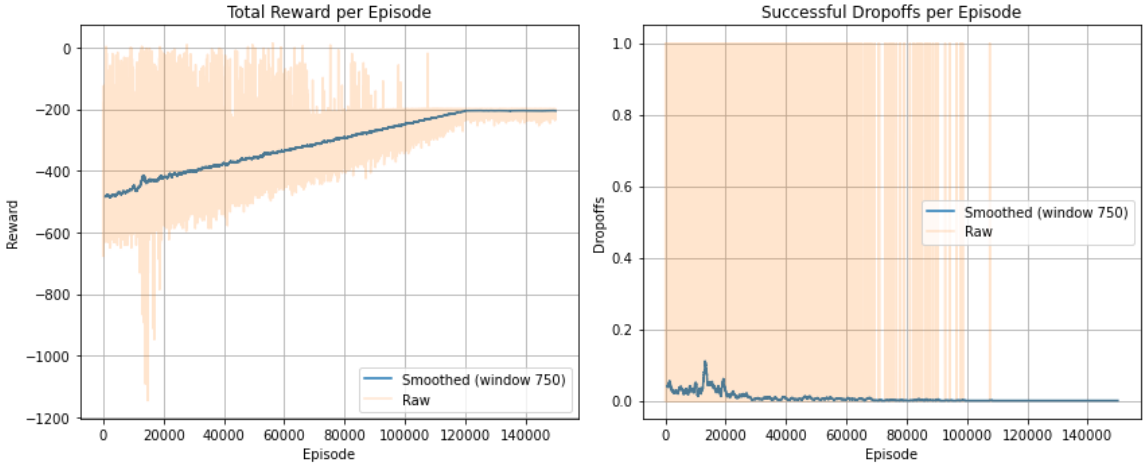

Figure 3 Augmented Interaction (Rank 2)

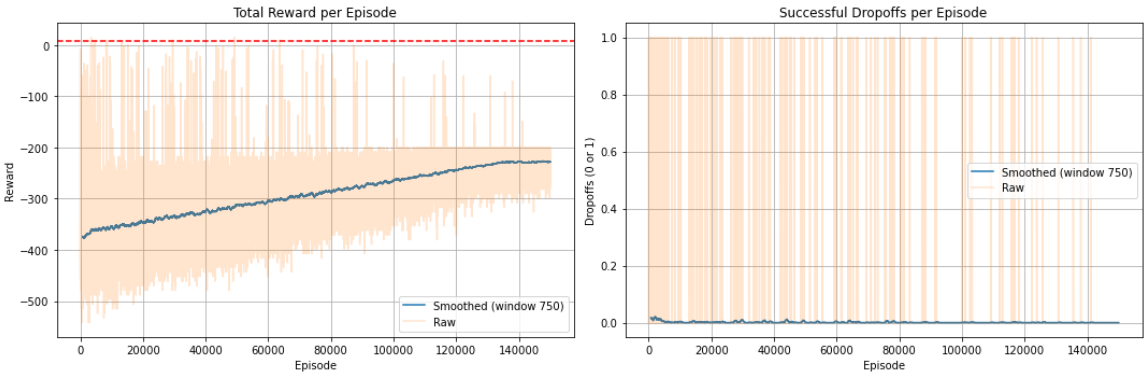

Figure 4 Linear SARSA Interaction (Rank 3)

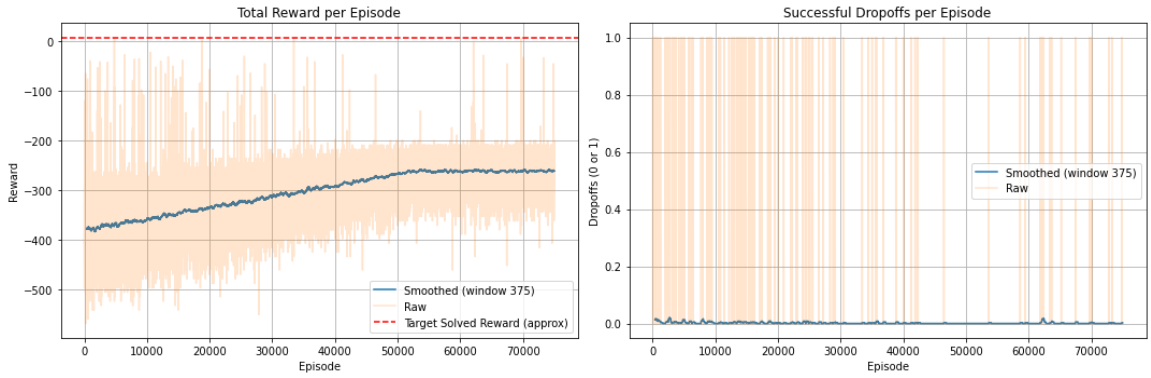

Figure 5 Linear SARSA Interaction (Rank4)

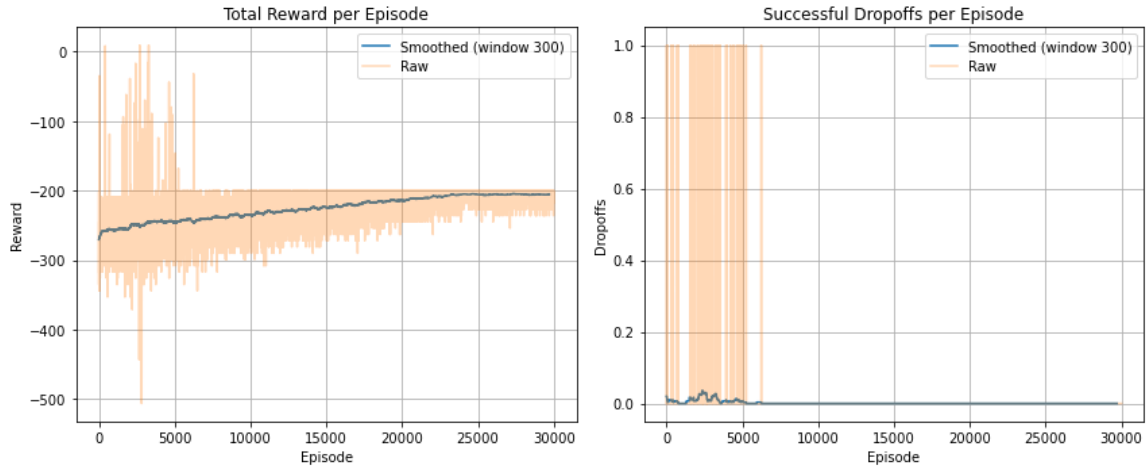

Figure 6 Baseline (Rank 5)

3.3. Iterative Development and Debugging Process

The development of a working SARSA agent with linear function approximation involved an iterative debugging process:

3.3.1. Initial Problem & Symptoms (The "Stuck" Agent - First Iteration)

- **Problem:** The initial agent, after some training, consistently got stuck, repeatedly choosing the same action (e.g., "South") even if it led to hitting a wall or not progressing. Q-values were observed to be very large negative numbers and changed minimally.

3.3.2. First Attempt at Fixing: Revising Feature Engineering (The "Better Feature Extractor")

- **Initial Feature Extractor (better_feature_extractor):**
    - features[0-24]: One-hot encoding of taxi's 5x5 grid position.
    - features[25]: Passenger in taxi (binary).

    - features[26]: "Misaligned" (passenger location != destination index).
    - features[27]: Taxi on edge row (binary).
    - features[28]: Normalized action (action / 5).
- **Identified Issues:**
    - **Normalized Action (action / 5):** This turned out to be a major flaw. By normalizing the action as a continuous value, I accidentally introduced an artificial ordering between actions that just does not exist in discrete environments like Taxi-v3. For example, a negative weight would push the agent toward action 0 (South), which explains why it kept getting stuck going South. In this kind of setup, actions should be one-hot encoded or handled using separate weights per action, anything else risks misleading the model.
    - **"Misaligned" Feature:** This one stayed almost constant throughout most episodes, which means it did not help the model distinguish between different states.
    - **Lack of Explicit Passenger/Destination Location Encoding:** One of the bigger gaps was not explicitly encoding the passenger's and destination's locations when the passenger was not in the taxi. These details are crucial in Taxi-v3, and without them, the linear model had no direct way to factor in where it needs to go or whom to pick up. This made it harder for the agent to learn context-aware policy.
- **Fix Attempt 1 (feature_extractor_comprehensive):**
    - Introduced separate one-hot encodings for: Taxi position (25 features), Passenger's current location (R,G,Y,B, InTaxi - 5 features), Destination location (R,G,Y,B - 4 features), Chosen action (6 features). Total features: 40.
    - This approach assumed an additive contribution: $Q(s,a) \approx w\_s\_taxi * \varphi\_s\_taxi(s) + ... + w\_a * \varphi\_a(a)$.
- **Outcome of Fix Attempt 1:** Still poor performance. The average rewards remained very low (e.g., -250s). The agent still got stuck.

3.3.3. Second Major Attempt: Introducing State-Action Interaction Features

- **Problem :** The linear model needed to learn different "state-value functions" for each action. The effect of state features differs based on the action.
- **Fix Attempt 2 (feature_extractor_interaction):**

    - **Logic:** "State-only" features replicated for each possible action; only the block corresponding to the chosen action is active.
        - phi_state(s): One-hot taxi pos (25) + one-hot pass_loc (5) + one-hot dest_idx (4) = 34 features.
        - phi(s,a): Vector of size 34 * 6_actions = 204 features. If action = k, phi(s,a) would have phi_state(s) in the k-th block of 34 features, zeros elsewhere.
    - **Effect:** Allowed weight vector w (size 204) to have dedicated segments for each action, effectively learning Q(s,a) ≈ w_a^T * phi_state(s).
- **Hyperparameter Tuning:**
    - **Learning Rate (alpha):** Initially 0.01. Reduced to 0.001 for stability with the larger, sparser feature vector, which was crucial. (e.g., at 75k episodes, rewards around -260 with alpha=0.01).
    - **Epsilon:** Adjusted for sufficient initial exploration (epsilon_initial=0.5), low late-stage noise (epsilon_min=0.01), decay over substantial training (epsilon_decay_fraction=0.8).
    - **Number of Episodes:** Increased significantly (150k, then 300k).
- **Outcome of Fix Attempt 2 (After 300k-350k episodes):**
    - **Partial Success:** Average rewards trended upwards towards -200.
    - **New Problem:** Agent learned to navigate to the destination but **without picking up the passenger first.** Got stuck at the destination, unable to drop-off.
    - **Reason:** phi_state(s) still did not easily distinguish conditions for pickup/dropoff (e.g., "taxi at passenger AND passenger not in taxi"). Model learned general tendencies like "going to destination Y is good" without conditioning on "having the passenger."

3.3.4. Third Major Attempt: Augmented Interaction Features with Explicit Task-Phase Indicators

- **Problem:** Agent needed explicit signals for sub-goals: being at the passenger location for pickup, and at the destination with a passenger for dropoff.
- **Fix Attempt 3 (feature_extractor_interaction_augmented):**
    - **Logic:** Added three new binary features to phi_state(s) before creating interaction terms:

1. **passenger_in_taxi:** Is passenger currently in taxi? (1.0 if pass_loc_idx == 4, else 0.0)
2. **at_passenger_pickup_loc_feature:** Is taxi at passenger's current location (if passenger not in taxi)? (1.0 if true, else 0.0)
3. **at_destination_with_passenger_feature:** Is taxi at destination (if passenger is in taxi)? (1.0 if true, else 0.0)

- **phi_state_augmented(s)** now has 34 (base) + 3 (task-phase) = 37 features.
- **Full interaction vector phi(s,a)** became size 37 * 6_actions = 222.

- **Hyperparameters:**
  - **Alpha:** Maintained at a low 0.001.
  - **Epsilon schedule:** Remained similar.
  - **Number of Episodes:** Continued, reaching 700,000 total episodes.
- **Outcome of Fix Attempt 3 (After 300k initial + 400k additional = 700k total episodes):**
  - **SUCCESS**
    - Average rewards consistently improved, settling in the -160 to -190 range.
    - "Drops (last 100 ep)" metric became consistently non-zero (around 10-25% success rate in later stages).
    - Rollout demonstrations showed correct navigation, pickup, and dropoff, achieving positive total rewards.
    - Occasional "stuck" behavior in rollouts can occur due to linear FA limitations and imperfect Q-value convergence, but training data clearly showed learning.

3.4. Logic Behind the Working Code (SARSA with Augmented Interaction Features - feature_extractor_interaction_augmented)

1. **State Representation:**
   - State s (integer 0-499) is decoded into (taxi_row, taxi_col, pass_loc_idx, dest_idx).
   - Converted into a phi_state_augmented(s) vector (37 features):

        - One-hot encoding of taxi's (row, col) position (25 features).
        - One-hot encoding of passenger's location (R,G,Y,B, or InTaxi - 5 features).
        - One-hot encoding of destination location (R,G,Y,B - 4 features).
        - Binary feature: passenger_in_taxi (1 feature).
        - Binary feature: at_passenger_pickup_loc_feature (1 feature).
        - Binary feature: at_destination_with_passenger_feature (1 feature).
    - This phi_state_augmented(s) is used to create the final state-action feature vector phi(s,a) (222 features). For a chosen action, phi_state_augmented(s) populates one block of 37 features within phi(s,a), with other blocks zeroed. This creates interaction terms, allowing different weights for state features depending on the action.
2. **Linear Action-Value Function (Q-function):**
    - Q(s,a; w) = w^T * phi(s,a), where w is the weight vector of size 222.
3. **SARSA Algorithm (linear_sarsa_augmented):**
    - **On-Policy Learning:** Learns Q-values based on actions taken by its current (epsilon-greedy) policy.
    - **Initialization:** Weights w initialized to zeros.
    - **Episode Loop:**
        - Reset environment, get initial state s.
        - Choose action a from s using epsilon-greedy policy based on current Q(s,a; w).
        - **Inner Step Loop:**
            - Take action a, observe reward r and next state s'.
            - Choose next action a' from s' using epsilon-greedy policy based on Q(s',a'; w).
            - **Update Weights w:**
                - **Calculate features:** phi_current = phi(s,a) and phi_next = phi(s',a').

- Calculate Q-values: q_current = w^T * phi_current and q_next_val = w^T * phi_next.
- Calculate TD target: target = r + gamma * q_next_val (if s' not terminal), or target = r (if s' terminal).
- Calculate TD error (delta): delta = target - q_current.
- Update weights: w = w + alpha * delta * phi_current.

➢ Set s = s' and a = a'.

- **Epsilon Decay:** Epsilon starts high (e.g., 0.5) for exploration, gradually declines to a small minimum (e.g., 0.01) for exploitation over a large fraction of episodes.
- **Hyperparameters:** alpha=0.001 (small learning rate for stability), gamma=0.99 (discount factor), large number of episodes (e.g., 700,000+) are crucial for convergence with these features.

4. **Policy Derivation (derive_policy_augmented):**
   - After training, a greedy policy is derived by choosing action = argmax_a Q(s,a; w) for each state s. This policy is used for evaluation.

Linear function approximation with SARSA proved capable of learning a successful policy for Taxi-v3, but required careful feature engineering, particularly the inclusion of state-action interaction features and explicit task-phase indicators. The learning process was significantly longer (700,000 episodes) compared to tabular methods. While the final average reward was negative (-180), indicating suboptimal paths on average during training, the greedy rollout policy could solve the task optimally (13 steps, +8 reward). The success rate during training (around 11-20% drop-offs per episode) shows that the agent learned to complete the task, although not as consistently as tabular methods within the same number of episodes. This highlights the challenges of feature representation and hyperparameter tuning in function approximation.

5. Policy Gradient Methods

5.1 Overall Performance Summary

This part explains the REINFORCE implementation using a learned baseline, optimized for the Taxi-v3 environment. The goal was to directly learn a parameterized policy $\pi(a|s; \theta)$ and use a linear baseline $V(s; w)$ to reduce variance in policy gradient updates. Both components were designed using feature-based linear function approximation.

5.1.1 Feature Design and Representation

The agent operates on a structured state space (500 discrete states × 6 actions), so designing the right feature representation was critical.

- **State-Action Features φ(s,a):** A 222-dimensional vector built using one-hot encodings for:
    - Taxi position (25),
    - Passenger location (5),
    - Destination (4),
    - Plus 3 binary flags:
        1. Is passenger already in the taxi?
        2. Is the taxi at the passenger's location (if not yet picked up)?
        3. Is the taxi at the destination with the passenger onboard?

These 37 features were "blocked" by action, resulting in φ(s,a) being a sparse vector where only one 37-dimensional block is active per action, allowing the linear model to assign distinct weights per action.

- **State Features φ_state(s):** A reduced 37-dimensional vector capturing only the state context, shared across all actions and used by the critic to estimate V(s).

5.1.2 Policy Architecture (Actor)

The agent's policy $\pi(a|s)$ was modeled as a softmax over action preferences:

- $h(s,a) = \theta^T \varphi(s,a)$,
- $\pi(a|s) = \text{softmax}(h(s,a))$.

The actor's parameters θ were updated via the REINFORCE rule, using:

- $\nabla\theta \log \pi(a|s) = \varphi(s,a) - E[\varphi(s,\cdot)]$,
- With $A_t = G_t - V(s_t)$ as the scaling factor.

5.1.3 Value Function Architecture (Critic)

The critic learned to predict V(s) using the 37-dimensional φ_state(s). Its parameters w were updated by:

- $w \leftarrow w + \alpha\_critic \times A\_t \times \varphi\_state(s)$,
- Essentially minimizing squared error between V(s) and the observed returns G_t.

5.1.4 Training Logic and Loop

Each episode was simulated using the current stochastic policy, collecting tuples of (s, a, r, φ_state(s), $\nabla \log \pi(a|s)$). Once the episode terminated:

- Returns G_t were computed backward.
- Advantages A_t were calculated.
- Optionally, A_t values were normalized for stability.
- The actor and critic were updated step-by-step using the previously stored gradients and features.
- Gradient clipping was used when necessary to prevent weight explosion.

5.1.5Implementation Details

- **Exploration:** Softmax policy naturally explores, but performance heavily depended on tuning the learning rate.
- **Gradient Stability:** Advantage normalization (zero mean, unit variance) was crucial, especially during early training.
- **Weights & Logs:** Actor and critic weights were stored to allow resuming long training sessions. Logs tracked episode reward, success rate, average steps, and squared advantage.

Summary (REINFORCE)

Training Size	Final Reward	Final Success (%)	Notes
300k	-187.57	5.90%	Weak, very slow learning
500k	-200.00	0.00%	Catastrophic collapse
700k	-176.52	15.00%	Mild recovery, unstable
800k	-158.67	22.00%	Best so far, still underperforming

5.3 Limitations and Observations

Even at 800k episodes, the final **greedy** policy derived from θ still failed in many rollouts. Common failure modes:

- Repeated invalid actions (e.g., hitting walls).
- Navigating to the passenger or destination but forgetting to "Pickup" or "Dropoff".
- Being stuck in a suboptimal loop due to weak preferences learned for edge cases.

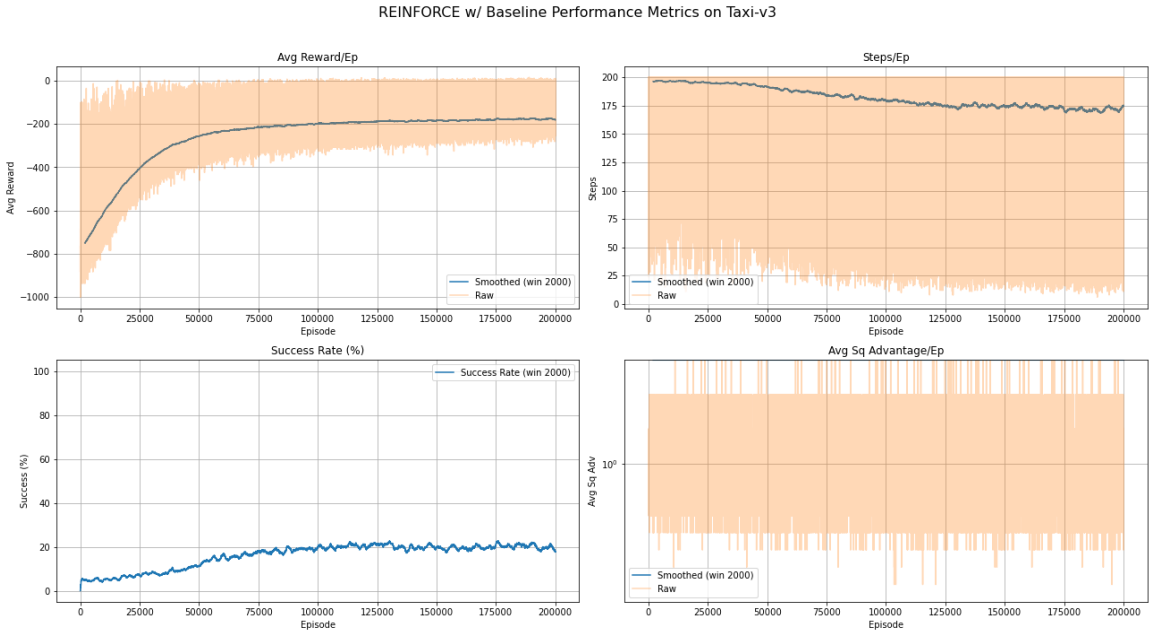

Figure 7

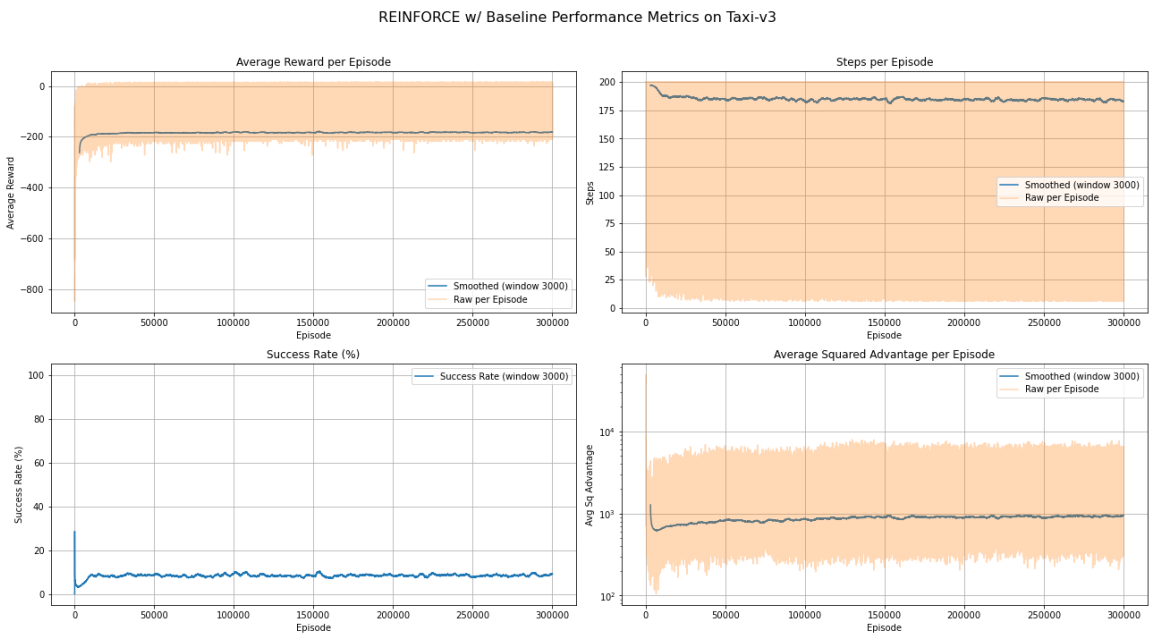

Figure 8 REINFORCE w/ Baseline (Linear Function Approximation)

Metric	**Value**
Avg Reward (Final 100 eps)	-183.46
Avg Steps (Final 100 eps)	184.96
Success Rate (Final 100 eps)	8.00%
Final Policy - Avg Reward	-187.57 ± 49.63
Final Policy - Avg Steps	188.81 ± 44.68
Final Policy - Success Rate	5.90%
Training Time	~8.5 hours (30559.13 sec)
Convergence Speed	Very slow (plateau near -180, noisy curves)
Avg Squared Advantage (End)	938.08 (high variance indicates instability)

Figure 9 REINFORCE w/ Baseline (after 200,000 more episodes)

Metric	**Value**
Final Policy - Avg Reward	-200.00 ± 0.00 (no variance)
Final Policy - Avg Steps	200.00 ± 0.00 (maxed out)
Final Policy - Success Rate	0.00%
Training Time (s)	16,591.95 (~4.6 hours)
Total Episodes	200,000
Learning Curve Ending Avg Reward	~ -174.22 (last 100 episodes)
Learning Curve Ending Success Rate	~21%
Avg Squared Advantage (End)	Constantly 1.00
Convergence Behavior	Flat, then a sudden drop in quality

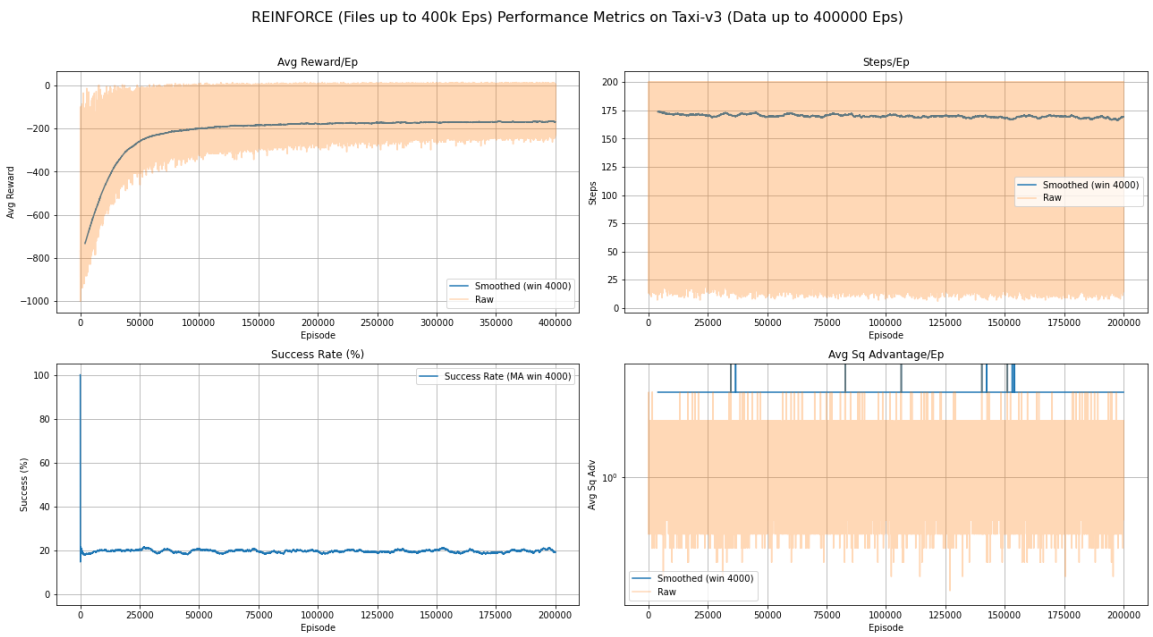

Figure 10 REINFORCE w/ Baseline (Linear FA) – After 400k Total Episodes

Metric	**Value**
Final Avg Reward (last 100 eps)	-176.52
Final Success Rate (last 100 eps)	15.00%
Training Time (This Run)	~4.7 hrs (16,969.49 sec)
Avg Squared Advantage	Constant 1.00
Policy Quality (subjective)	Slightly more stable but not optimal
Reward Trend	Improved but then fluctuates again
Success Rate Trend	Rarely crosses 30%, mostly between 15–23%

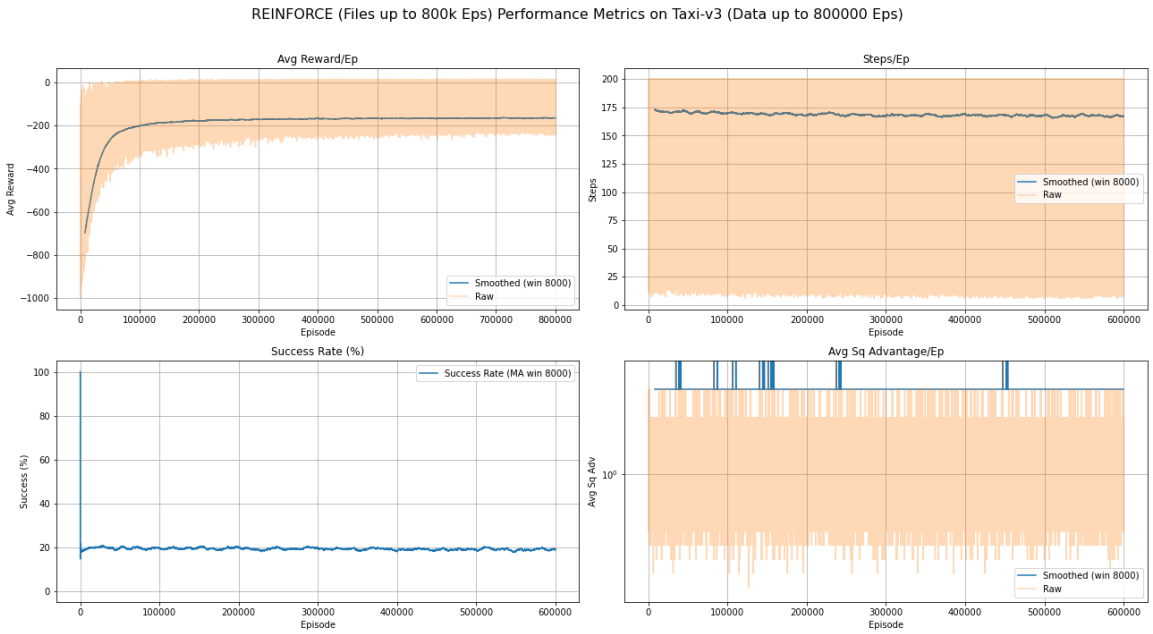

Figure 11 REINFORCE w/ Baseline (Linear FA) – After 800k Total Episodes

Metric	**Value**
Final Avg Reward (last 100 eps)	-158.67
Final Success Rate (last 100 eps)	22.00%
Training Time (This Run)	~7.3 hrs (26,306 sec)
Total Episodes Trained	800,000
Avg Squared Advantage	Constantly 1.00
Stability	Slightly improved but still fluctuating

Overall Comparison and Conclusion

- **Policy Iteration** and **Tabular Q-Learning** both found optimal policies for Taxi-v3, achieving 100% success rates and positive average rewards efficiently. Policy Iteration, being model-based, was very fast. Q-Learning converged within 50,000 episodes.
- **Linear Function Approximation (SARSA)** demonstrated the feasibility of solving Taxi-v3 without a full tabular representation. However, it required extensive feature engineering, much more training episodes (up to 700,000), and careful hyperparameter tuning. While a successful greedy policy was learned, the average performance during training was lower, and the success rate per episode was modest compared to tabular methods. This illustrates the increased complexity and challenges associated with function approximation.
- **REINFORCE with Baseline (Linear Policy Gradient)** added another layer of challenge. It required even more training (800,000+ episodes) and sophisticated techniques, like advantage normalization, careful reward tracking, and gradient clipping, to remain stable. While the actor-critic setup did show improvements over time, the final greedy policy remained weak. Agents often failed to execute critical actions like "Pickup" at the right state, despite being able to reach the goal locations. The high variance in returns and the linear model's limited capacity were significant bottlenecks. Still, the method demonstrated the potential of policy-gradient-based learning under constrained representations.

.