# Smart Taxi: Elevating Taxi Services with Reinforcement Learning

Sanjida Sharna
*Dept. of Computer Science*
*Tennessee Technological University*
Cookeville, United States
sasharna42@tntech.edu

Chern Chao Tai
*Dept. of Computer Science*
*Tennessee Technological University*
Cookeville, United States
ctai42@tntech.edu

***Abstract*—Recently, there is a significant demand for autonomous vehicles that leverage reinforcement learning algorithms to develop intelligent navigation systems that enable vehicles to make informed decisions in real-time, enhancing safety and efficiency on the roads. Inspired by this, we explore the application of Reinforcement Learning (RL) techniques to improve the navigation abilities of a taxi agent within the 'Taxi' environment provided by OpenAI Gym. The environment simulates a grid world with four designated locations (Red, Green, Yellow, and Blue). The agent's objective is to efficiently pick up passengers from assigned (state space specified) locations and deliver them to destinations(state space specified), aiming to accomplish this task in the shortest number of moves possible. Also our agent will be able to calculate fare upon each successful drop off and total income. With 500 discrete states representing various configurations of the taxi, passenger, and destination positions, we investigate how SARSA algorithm can learn optimal policies to efficiently navigate the complex environment. Through experimentation and analysis, our project aims to showcase the effectiveness of SARSA in solving real-world navigation problems and its potential for broader applications.**

***Index Terms*—taxi, reinforcement learning, SARSA, OpenAI Gym, navigation**

## I. Introduction

Reinforcement Learning (RL) provides a robust framework for training agents to make consecutive decisions in dynamic environments. In this project, we focus on applying the SARSA algorithm, a temporal-difference RL method, to address the navigation task presented by the 'Taxi' environment from OpenAI Gym [1]. This environment serves as a standardized platform for developing and evaluating RL algorithms which simulate a grid world where a taxi agent is tasked with picking up passengers and dropping them off at designated locations at their destinations within the fewest moves possible. our agent will possess the capability to calculate fare upon each successful drop-off and track total income throughout multiple episodes. This feature enables the agent to not only optimize its navigation strategy for efficiency but also to assess its financial performance and revenue generation potential in a simulated taxi service scenario.

The SARSA algorithm is particularly well-suited for this task as it learns action-value functions directly from experience by updating estimates based on observed transitions. By iteratively interacting with the environment, SARSA learns optimal policies that balance exploration and exploitation, enabling the taxi agent to navigate efficiently to its destinations while maximizing cumulative rewards. The primary objective of our project is to investigate the effectiveness of the SARSA algorithm in learning navigation policies within the 'Taxi' environment. By training the agent using SARSA and evaluating its performance, we aim to demonstrate the algorithm's ability to learn complex decision-making strategies and efficiently solve navigation tasks in discrete state spaces. Through empirical analysis and experimentation, we seek to gain insights into the strengths and limitations of SARSA in addressing real-world navigation problems, ultimately contributing to the broader understanding of RL algorithms and their practical applications.

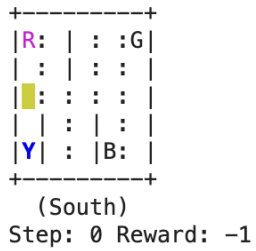


Fig. 1: Visualization of Open Ai Gym Taxi-v3 environment

## II. Related Work

Several studies have investigated the application of Reinforcement Learning (RL) techniques to train agents for tasks such as navigation and control in complex scenarios.

Such as Shou et al. [2] utilized Markov Decision Processes (MDPs) to model optimal sequential decision-making for e-hailing drivers, who match passengers to platforms similar to Transportation Network Companies (TNCs). Their scheme, validated through Monte Carlo simulations, outperformed baseline heuristics like hotspot strategies. The MDP model effectively handled supply-demand ratios, minimizing unmatched orders due to a sufficient number of drivers. Dynamic adjustments among driver orders allowed for competition calibrations, enhancing match efficiency.

Wang et al. [3] proposed a method combining MDPs and DQNs to improve dispatching for ride-sharing drivers using data from DiDi. Their approach, CFPTs, enhanced learning efficiency across varied demand-supply conditions by transferring information spatially and temporally. The study showed that policies learned with transfer learning outperformed those without.

Yang et al. [4] utilized Q-learning RL algorithm to train a taxi agent for efficient navigation in a grid-world environment. The agent underwent training for 100,000 episodes, achieving its highest reward around the 600th episode. RL outperformed non-RL methods in terms of penalties, timesteps, and rewards. However, scalability issues prompted the introduction of Deep Q-learning Network (DQN) to handle larger state spaces more effectively.

Chen et al. [5] devised a system utilizing Deep Reinforcement Learning (DRL) to tackle demanding urban autonomous driving situations. Their approach automatically improved performance by capturing low-dimensional latent states and employing specialized input representations and visual encoding. The framework integrated multiple model-free deep DRL algorithms and boosting techniques to enhance performance. Evaluation in high-definition driving simulations demonstrated superior task completion compared to predefined baselines.
In summary, previous research has underscored the efficacy of reinforcement learning methods in enhancing navigation tasks, emphasizing the need for continued exploration of novel algorithms to further optimize autonomous systems' performance in dynamic environments.

## III. Problem Statement

In this project, we aim to tackle the challenge of efficient navigation within a grid-world environment using Reinforcement Learning (RL) techniques. Our objective is to train a taxi agent to autonomously navigate the environment, pick up passengers from pickup locations (state specified), and drop them off at specified destinations(state specified) while minimizing the number of moves and maximizing cumulative rewards.

The environment is represented as a grid world with four designated locations: Red, Green, Yellow, and Blue. In simulation the Blue colored location denotes pick-up location and purple colored location denotes drop-off destination.The taxi agent operates within this environment and can perform actions such as moving in four directions (north, south, east, west), picking up passengers, and dropping off passengers. The state space comprises 500 discrete states representing various configurations of the taxi, passenger, and destination positions. Each state is represented as a tuple of (taxi row, taxi column, passenger location, destination). Actions available to the agent include moving south, moving north, moving east, moving west, picking up a passenger, and dropping off a passenger. Rewards are assigned based on the agent's actions. Successful passenger drop-offs yield a positive reward (+20), illegal moves incur a negative reward (-1), and a small negative reward is given for each timestep to encourage efficiency.

Our goal is to optimize the agent's policy to maximize cumulative rewards over time by learning an optimal action-value function (Q-function) through RL algorithm specifically SARSA and Dyna-Q algorithms. Also we are integrating the fare calculation and income tracking that enriches the agent's capabilities and contributes to a more holistic evaluation of its performance in the context of autonomous navigation and taxi service operations.

Our key challenges include efficiently exploring the state space, balancing exploration and exploitation, and addressing scalability issues with large state spaces.

**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Loop for each step of episode:
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

Fig. 2: Pseudocode of the SARSA Algorithm

## IV. Data Collection

Our primary source of data is the Taxi-v3 environment which is a simulated grid world by OpenAI Gym. The data of our model consists of the observation space, action space, number of episodes, rewards per episode, number of steps taken by the agent per episode, and the drop-off fare. The action space consists of 6 actions: 1) moving south, 2) moving north, 3) moving east, 4) moving west, 5) pick up passenger, and 6) drop off passenger. The observation space includes 500 discrete states. They are made up of 25 taxi positions, 5 possible locations of the passenger which includes the scenario when the passenger is picked up, and 4 destination locations [8].

The passenger's initial locations could be one of the following: 1) red (R), 2) green (G), 3) blue (B), 4) yellow (Y), and 5) in taxi. The destinations will be indicated by one of the four colors, RGBY. However, the passenger's initial location and the destination of a particular episode will never overlap which means that the initial location of the passenger cannot simultaneously be the destination. Each step taken by the agent will return a reward of -1, illegal pick ups and drop-offs will return -10 reward, and the agent will receive reward of +20 for dropping off the passenger at the destination.

## V. Model Development

### A. Choosing a Reinforcement Learning Algorithm

The reinforcement learning algorithm that we have selected for our project is the State-Action-Reward-State-Action (SARSA), an on-policy learning algorithm, and Dyna-Q, an algorithm that combines Q-learning and Q-planning. On-policy means that the algorithm learns the value of state-action pairs under the current policy that is being followed. SARSA uses

Temporal-Difference (TD) learning to enable the value estimates to be updated based on the current state and the next state-action pair. It is an experience-based algorithm that allows the agent to learn from individual experiences and adapt at every time step and episode.

Dyna-Q is a model-based reinforcement learning algorithm that combines direct RL like Q-learning and planning. As such, Dyna-Q uses the simulated experience from past interactions with the environment. It does this by integrating planning to update the value function based on these simulated experiences and real experiences with the environment. Theoretically, the planning part of Dyna-Q should lead to more efficient explorations which is a highly desired characteristic because it improves decision-making and learning of the model.

### *B. Agent Implementation*

The Taxi-v3 environment can be seen as a finite Markov Decision Process (MDP) with discrete state and action spaces. In an MDP framework, the agent interacts with the environment, transitioning between states and taking actions that will maximize the cumulative rewards. Each coordinate in the grid cell constitutes a particular state of the environment. The state can be represented by the tuple {taxi_position, location of passenger, destination}. We took advantage of the nature of our environment to implement our agent. For our agent to select an action, we defined the epsilon-greedy function that uses the exploration-exploitation strategy for the environment. It is implemented with the aim to balance between trying new actions (exploration) and utilizing learned knowledge (exploitation) to guide our agent towards an optimal policy.

We initialized epsilon with a probability of 0.01 for exploration. During exploration, the function selects a random action from the allowed options based on the passenger's location. If the passenger is at the destination, only the "drop-off" action is allowed. If the passenger is at the pick-up location (and not in the taxi), only the "pick-up" action is allowed. Otherwise, all actions for navigating (up, down, left, right) are allowed. Conversely, during exploitation, which occurs with a probability of 1-epsilon, the function selects the action with the highest Q-value in the Q-table for the current state. This approach ensures that our agent explores new actions while also exploiting its learned knowledge without taking any illegal actions(drop off and pick up at any undesignated place) to navigate the environment effectively.

Subsequently, within each step of an episode, we perform the SARSA update to update the Q-values based on the observed state-action-state-reward-action sequence. This update rule is expressed mathematically as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

This is a key component for our agent to learn the optimal policy by iterative update on the action values.

### *C. Training the Agent*

After implementing our agent, we can then train it using the SARSA algorithm. We first initialize the Q-table which is a data structure to store action values for each state-action pair. The number of episodes is set to a constant. The agent will interact with the environment by iterating through the episodes along with a maximum number of steps of 30 per episode. By selecting the appropriate action based on the epsilon-greedy policy mentioned above, the agent observes the next state and reward and will update its Q-values accordingly. We added the decay rate on epsilon over the episodes to gradually shift the agent's behavior from exploration to exploitation as it adapts itself to the environment.

## VI. Model Tuning and Evaluation

Our model comprises of adjustable parameters such as learning rate (alpha), discount factor (gamma), exploration rate (epsilon), and decay rate for exploration. These parameters typically fall in the range of 0 to 1. Alpha determines how much weight new information carries compared to past knowledge. The smaller the value of alpha, the slower the model learns. Following standard practices, we initially set alpha to 0.1 and gradually increase it to 0.5. Gamma dictates the importance of future rewards. It ensures that future rewards are appropriately considered and prevents the total return from becoming infinite. Since our agent is far-sighted, we initialize the gamma variable to 0.95.

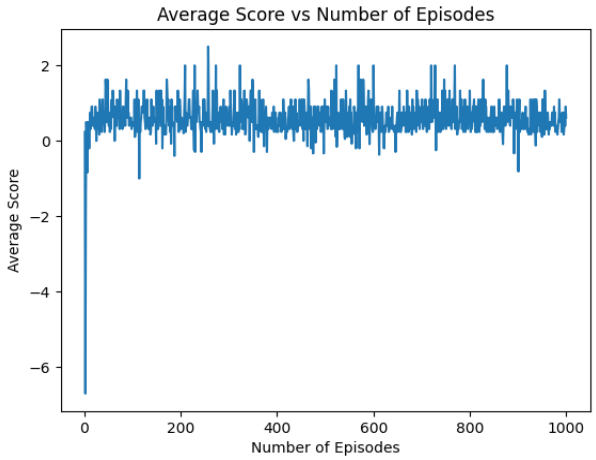


Fig. 3: Average Reward Achieved by Agent Over Training Episodes

Epsilon controls the balance between exploration and exploitation. Higher epsilon values encourage more exploration, while lower values prioritize exploitation. To encourage early exploration during our training process, we started with 0.1 for epsilon. Lastly, the decay rate controls how quickly the exploration rate diminishes over time. A value closer to 1 indicates slower decay, while a value closer to 0 indicates faster decay. We set the value for decay as 0.99.

Upon finishing 1000 episodes, the observed mean reward per episode is approximately 0.60. After tuning the parameters, it was expected that the performance of the

model would improve significantly. However, we observed that the performance of the model did not exhibit significant improvement compared to the initial parameter values.

In addition to our SARSA model, we encountered challenges with the Dyna-Q approach as the agent consistently failed to pick up and drop-off the passenger. While Dyna-Q offer advantages theoretically, its effectiveness in environments such as the taxi problem may be limited. Hence, these outcomes suggest the need for further investigation into the factors that are contributing to these issues.

## VII. Discussion

Our project demonstrates the effectiveness of reinforcement learning (RL) techniques in training a taxi agent to navigate a grid-world environment autonomously. Initially, the agent struggled to reach its destination efficiently, requiring a significant number of episodes for learning. However, as training progressed, we observed a noticeable improvement in the agent's performance, with reduced time taken to complete tasks and increased rewards obtained. One notable challenge [6] [7] encountered during the training process was observed in the second episode, where the agent consistently encountered the same actions and rewards. This phenomenon may be attributed to a built-in problem within the taxi environment itself, as similar issues have been documented in online discussions and forums. Despite this initial obstacle, our study highlights the adaptability of RL algorithms in overcoming such challenges through iterative learning and exploration of the state space.

## VIII. Future Work

Our project and background study suggests several promising avenues for future research. Firstly, there is a need for further investigation into the underlying factors contributing to the initial status observed during the early stages of training. Understanding the intricate dynamics of the taxi environment and identifying potential modifications or enhancements could lead to notable improvements in performance and efficiency.

As a future direction, we propose extending the traditional Taxi environment by introducing weather dynamics as an additional factor influencing the agent's decision-making process. By incorporating weather conditions such as sunny, rainy, and snowy, the agent's navigation strategy will need to adapt to varying road conditions and passenger demand patterns. This extension can be implemented by augmenting the state space to include weather information and modifying the reward structure to incentivize the agent's successful navigation through adverse weather conditions. For instance, the agent could receive additional rewards for successfully completing trips during inclement weather, reflecting the increased difficulty and risk associated with such journeys. To implement this extension, we can wrap the existing state environment to include weather states and integrate weather dynamics into the decision-making process of the agent. By introducing this additional complexity, we aim to explore how RL algorithms can effectively adapt to dynamic environmental factors and enhance the agent's overall performance and robustness in real-world scenarios.

Moreover, the exploration of advanced RL techniques, such as other reinforcement learning algorithms (MDP, TD etc) and deep reinforcement learning, holds promise in accelerating the learning process and achieving optimal performance more rapidly. Additionally, expanding the scope of the study to encompass more intricate environments or real-world scenarios would provide valuable insights into the scalability and applicability of RL-based navigation systems.

In a nutshell, our project serves as a springboard for ongoing exploration and refinement of RL algorithms in autonomous navigation tasks. The overarching objective remains the development of robust and efficient systems capable of navigating dynamic real world environments with precision and adaptability.

## IX. Conclusion

In conclusion, our project demonstrates both effective and ineffective RL techniques in training a taxi agent for autonomous navigation within a grid-world environment. Despite encountering challenges, such as initial stagnation in training episodes, the agent exhibited notable improvement over time (1000 episodes), showcasing its adaptability and learning capabilities. Looking ahead, further investigation into the root causes of early training stagnation is warranted, along with potential modifications or enhancements to address these challenges. as a future work extending the project to encompass more complex environments or real-world scenarios, such as incorporating weather dynamics as an additional influencing factor, would hold promise for enhancing the scalability and applicability of RL-based navigation systems. Overall, our project serves as a foundational step towards advancing RL algorithms in the realm of autonomous navigation, with the ultimate aim of developing robust and efficient systems capable of navigating dynamic environments with precision and adaptability.

## References


[1] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[2] Shou, Zhenyu, et al. "Optimal passenger-seeking policies on E-hailing platforms using Markov decision process and imitation learning." Transportation Research Part C: Emerging Technologies 111 (2020): 91-113.

[3] Wang, Zhaodong, et al. "Deep reinforcement learning with knowledge transfer for online rides order dispatching." 2018 IEEE International Conference on Data Mining (ICDM). IEEE, 2018.

[4] Yang, Hanzhi. "An independent study of reinforcement learning and autonomous driving." arXiv preprint arXiv:2110.07729 (2021).

[5] Chen, Jianyu, Bodi Yuan, and Masayoshi Tomizuka. "Model-free deep reinforcement learning for urban autonomous driving." 2019 IEEE intelligent transportation systems conference (ITSC). IEEE, 2019.

[6] https://ai.stackexchange.com/questions/17528/why-is-this-deep-q-agent-constantly-learning-just-one-action

[7] https://www.reddit.com/r/reinforcementlearning/comments/ujd1na/qlearning on openai gym never gets reward/

[8] https://www.gymlibrary.dev/environments/toy-text/taxi/