# Let’s Take a Ride:
# Goal-Conditioned Reinforcement Learning for Taxi Navigation in NYC

Matthew Dall’Asen · Arpit Dang · Varun Satheesh

https://github.com/mdallasen/Lets-Take-A-Ride

## Introduction

The objective of this report is to develop an autonomous taxi agent capable of navigating between randomly selected start and end points within New York City. Traditional approaches have typically relied on rule-based algorithms for pathfinding, which often falter in dynamic or unpredictable urban environments. In contrast, this research aims to enhance the robustness and adaptability of autonomous navigation by leveraging Reinforcement Learning (RL).

By framing navigation as a sequential decision-making problem, we train an agent to optimize its route while balancing efficiency, safety, and adaptability. Using a simulated urban environment, the agent learns through interaction to make decisions that account for real-world complexity offering a significant advantage over static methods.

This report employs Deep Q-Networks (DQN), a model-free RL algorithm that enables scalable learning and effective approximation of optimal action-value functions. Moreover, the task is structured as goal-conditioned reinforcement learning, requiring the agent to condition its policy not only on its current state but also on a specified destination. This formulation allows for better generalization across a wide range of goals and environments, moving closer to the deployment of intelligent, self-navigating agents in real-world urban settings.

## Methodology

This study trained an agent using a progressive reinforcement learning (RL) framework, beginning with simple fixed-route navigation in static environments and gradually introducing more complex and goal variables. The architecture implements a goal-conditioned, model-free Deep Q-Network (DQN), where graph-structured environments (nodes and edges) are encoded as embedded vectors and combined into a state representation for decision-making. An epsilon-greedy policy guided exploration, while experience replay and Q-learning updates optimized the policy. This approach enabled the agent to generalize effectively across varying navigation tasks and adapt to increasingly complex urban scenarios.

### Dataset & Environment

To create a realistic yet computationally efficient urban navigation environment, node- and edge-level data were extracted from OpenStreetMap (OSM) for a focused region of New York City (centered at 40.7780, -73.9580 with a 300m radius). This localization significantly reduced graph complexity while preserving essential urban features. Using OSMnx with local caching, the area was processed into a directed graph containing only drivable roads (network_type='drive'). The resulting environment retained critical topological attributes, such as intersection coordinates, road directionality, street classifications, and traffic infrastructure, while reducing the graph to a manageable scale, ~50 nodes.

A custom Gymnasium environment was developed to frame this as a reinforcement learning task featuring: (1) A discrete action space constrained to valid neighboring nodes (up to 4 options per step); (2) State representations based on node/grid pairs; and (3) Goal-conditioned sampling from five predefined destinations. The reward function combined distance-based shaping with penalties for node revisits and inefficient paths, and episodes were capped at 50 steps to avoid infinite loops. The resulting environment maintained enough complexity to foster learning of meaningful navigation policies, while remaining tractable for training on CPU hardware.

### Reward Criteria

The agent’s reward function was carefully designed to promote efficient, lawful navigation while discouraging suboptimal behaviors. At its core, the system employs a distance-based shaping reward: the agent receives a positive reward (+1.0) for moving closer to the goal (as measured by Euclidean distance) and a penalty (-1.0) for moving farther away. This provides dense, incremental guidance toward the destination. To ensure goal-directed behavior, a large terminal reward (+1000.0) is granted upon successful arrival, ensuring that the optimal policy aligns with task completion.

To discourage inefficient exploration, the agent incurs penalties for revisiting nodes (-5.0 per recurrence), preventing cyclical behavior. Additionally, the environment enforces traffic rules inherently by restricting actions to valid neighboring nodes, eliminating the need for explicit penalties for illegal moves (e.g., wrong-way travel on one-way streets). The reward function further incorporates subtle incentives for navigating complex intersections by scaling rewards with street density (+0.1 per lane), encouraging the agent to learn adaptive routing strategies.

*Table 1. Reward Function Breakdown for Agent Navigation*

| | **Move closer to goal** | **Move farther from goal** | **Reach goal node** | **Revisit previous node** | **Navitage multi lane intersection** | **No valid neighbours** |
|---|---|---|---|---|---|---|
| **Rewards** | +1 | -1 | +1000 | -5 | +0.1 per lane | -10 |

## Model

A Deep Q-Network (DQN) was used as the learning model for this task. DQN was chosen for its efficiency in handling discrete action spaces and its relatively low computational requirements. Given the complexity of the environment and the need for the agent to plan over long, randomly generated paths, a simpler RL model like DQN was more appropriate than more complex alternatives, which may have introduced unnecessary overhead at this stage of development.

A Deep Q-Network (DQN) with a five-layer fully connected architecture was used, with layer dimensions of 256, 256, 128, 64, and 4. ReLU activations were applied to hidden layers, with no activation on the output to predict Q-values directly. To support dynamic goals, the destination information was embedded into the input state, allowing the agent to condition its actions on both its current position and the goal. This goal-conditioning approach enabled the agent to generalize across different navigation tasks without needing separate models for each start-end pair.

## Training

The training procedure employs a goal-conditioned Deep Q-Network (DQN) framework with a dual-network architecture and experience replay to enable efficient and stable learning. Both the online and target Q-networks are initialized identically, with the target network's weights periodically synchronized to stabilize value estimation over time. Each episode begins by randomly sampling valid origin-destination pairs from a predefined set of random goal candidates, ensuring exposure to diverse trajectory patterns while rejecting invalid or redundant configurations via the environment's reset routine. The agent's state is represented as a concatenation of one-hot encoded vectors for the current and goal nodes, forming a compact yet expressive input that supports generalization across varying tasks. An ε-greedy policy (ε = 0.1) balances exploration and exploitation, selecting either random actions or those with the highest Q-value predictions from the online network. Transitions (state, action, reward, next_state, done) are stored in a fixed-size replay buffer (capacity = 500), from which mini-batches (size = 10) are sampled for training. For each batch, the target Q-values are computed using the frozen target network, and the online network is updated via mean squared error loss and backpropagation, with multiple gradient steps per environment interaction. This approach leverages randomized goal sampling, structured state representations, and decoupled value updates to improve policy generalization and mitigate instability due to correlated experiences achieving robust performance on previously unseen navigation tasks.

## Metrics

*Table 2. Evaluation Metrics Used to Assess Navigation Performance*

| Metrics | Description |
|---|---|
| Success Rate | Proportion of episodes in which the agent successfully navigates from the start to the goal node. |
| Average Reward | Mean total reward accumulated per episode, calculated using the designed reward structure. |
| Average Distance Travelled | Mean physical distance (in meters) the agent covered from the origin to the destination. |
| Average Steps | Average number of node-to-node transitions taken per episode. |
| Detour Ratio | Ratio of the actual path length to the straight-line (Euclidean) distance between start and goal. |
| Average Step Efficiency | Average distance covered per step, reflecting the efficiency of each movement. |
| % of the Steps that Moves Closer to Goal | Percentage of steps during which the agent reduced its distance to the goal node. |

These evaluation metrics collectively assess both the effectiveness and efficiency of the agent's navigation strategy. While the success rate indicates the agent's ability to complete tasks, metrics like average reward, steps, and distance traveled reveal how optimal and lawful its behavior is within the environment. The detour ratio and step efficiency offer insights into path optimality, helping to identify unnecessarily long or circuitous routes. Finally, the percentage of steps that move closer to the goal acts as a fine-grained measure of directional accuracy and policy sharpness. Together, these metrics form a robust framework to evaluate real-world readiness in autonomous navigation tasks.

## Results

The model was trained over 200 episodes, during which a graph was generated to track the reward earned in each episode. While the reward values fluctuated throughout the training process, this variability reflects the agent's learning curve and the combination of starting and ending nodes. In particular, the agent initially explored inefficient paths, such as attempting to travel to the same nodes previously visited or moving away from the goal, but gradually improved its strategy through trial and error.

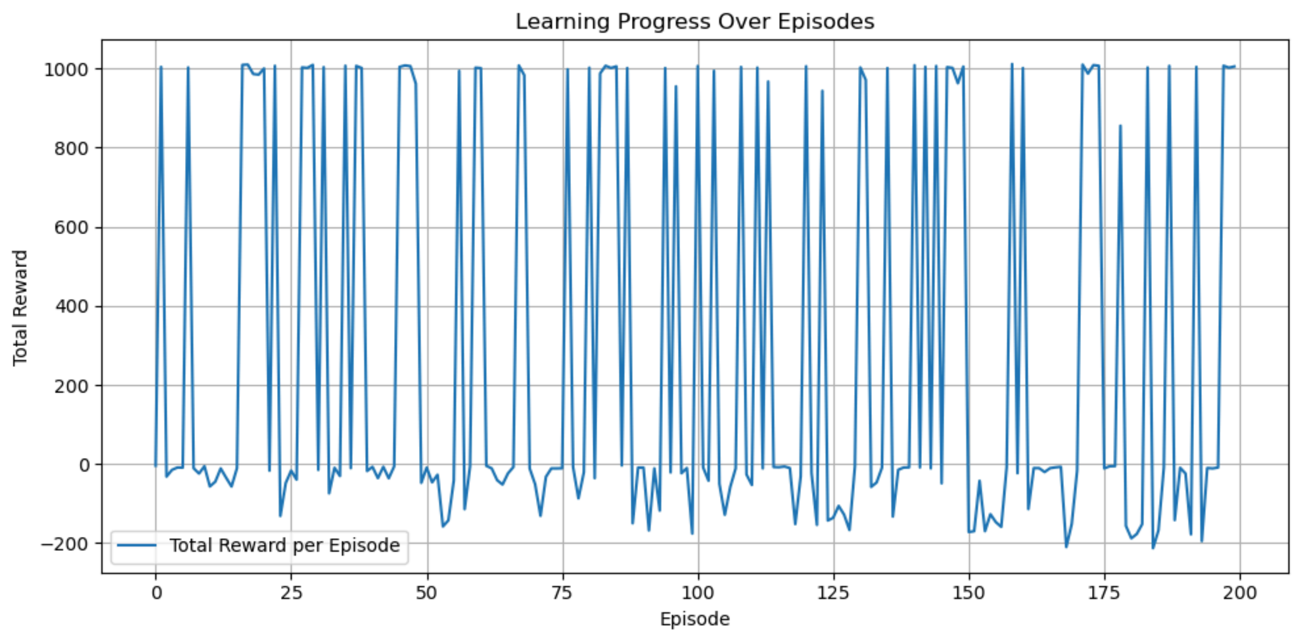

*Figure 1. Training Performance: Total Reward per Episode Over Time*

Following training, the model was evaluated on 20 test episodes, each involving randomly selected start and goal nodes. The results below summarize its performance across these trials, based on the evaluation metrics outlined in the methodology section.

*Table 3. Model Performance Summary on Test Episodes*

| **Metric** | **Value** |
|---|---|
| Success Rate | **80 %** |
| Average Reward | 46.0 |
| Average Distance Travelled | 0.0138 |
| Average Steps | 16.3 |
| Detour Ratio | 6.15 |
| Average Step Efficiency | 0.0010 |
| % of the Steps that Moves Closer to Goal | 67.8 % |

The model successfully generated a visualization of one of the completed episodes, depicting the vehicle's path from the starting node to the destination node. Below is an example of one such successful visualization produced by the model:

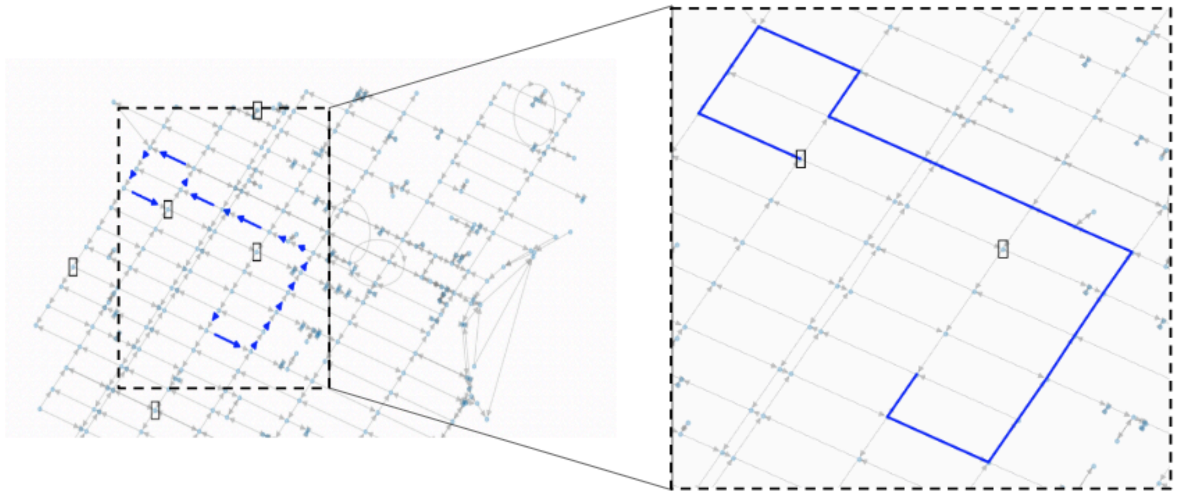

*Figure 2. Successful Navigation Path Visualized on NYC Street Graph*

## Challenges

Based on the results, we observed that the detour ratio was higher than desired. To address this, we iteratively refined the reward structure, aiming to maximize the success rate while also improving secondary metrics such as efficiency and path optimality. Designing an effective reward function proved to be one of the most challenging aspects, requiring a careful balance between incentivizing goal-directed behavior and penalizing inefficient actions.

Another major challenge involved optimizing the environment size to match the computational resources available. While the initial goal was to train the model on the entire city of New York, the complexity and scale made this impractical. As a result, we constrained the environment to a smaller, representative neighborhood in Manhattan. Despite this reduction, the current model architecture is scalable and can be extended to larger urban areas with appropriate computational support.

This project stretch goal was to integrate real-time traffic data through live traffic APIs, allowing the model to account for dynamic conditions such as congestion, accidents, and timing delays. Incorporating these real-world factors would have made the model significantly more applicable to today's urban navigation challenges. However, additional development time was required to implement this feature, and it remains a key objective for future work.

Although the maps included traffic infrastructure, we were not yet able to incorporate detailed traffic laws and regulations. This includes intersection timing based on traffic signals, mandatory stops at stop signs, and adjustments for high pedestrian areas such as Times Square in Manhattan. Enhancing the model with these real-world constraints is a priority in our ongoing development roadmap.

## Reflection

This project not only met its foundational goal of navigating fixed, static routes but also achieved the target objective of managing random start-to-goal navigation under static traffic rules. The stretch objective, integrating real-time traffic data through APIs, remains part of our future development roadmap. The model consistently behaved as expected within the discrete routing framework, effectively following defined parameters and producing reliable results across varied scenarios.

Initially, this study began with tabular Q-learning but quickly pivoted to Deep Q-Networks (DQNs) due to the high dimensionality and complexity of the dataset. In hindsight, an early shift toward a graph-attention actor-critic framework could have provided additional benefits. Such an architecture is well-suited to environments with variable action spaces and would have helped mitigate value overestimation, a common challenge in value-based methods.

Given more time, several key enhancements would be prioritized: (1) integrating live traffic speed APIs to enable dynamic edge weighting, (2) testing transferability of the model to other cities like Chicago or Boston, and (3) implementing hierarchical reinforcement learning for long-distance route planning.

The biggest conclusions from this project include; goal-conditioning significantly improves generalization without requiring separate models for each task; reward shaping is a nuanced but powerful tool for accelerating learning; and with careful design, even relatively simple DQN architectures can deliver strong performance. However, long-term robustness and scalability will likely depend on transitioning to more advanced value- or policy-based reinforcement learning frameworks.