NYU

# Resources Allocation:

# Optimizing Bike Rebalancing in NYC's Citi Bike System with Reinforcement Learning

Lyric Li, Hongxin Song, Lia Wang, Maggie Xu

25.04.29

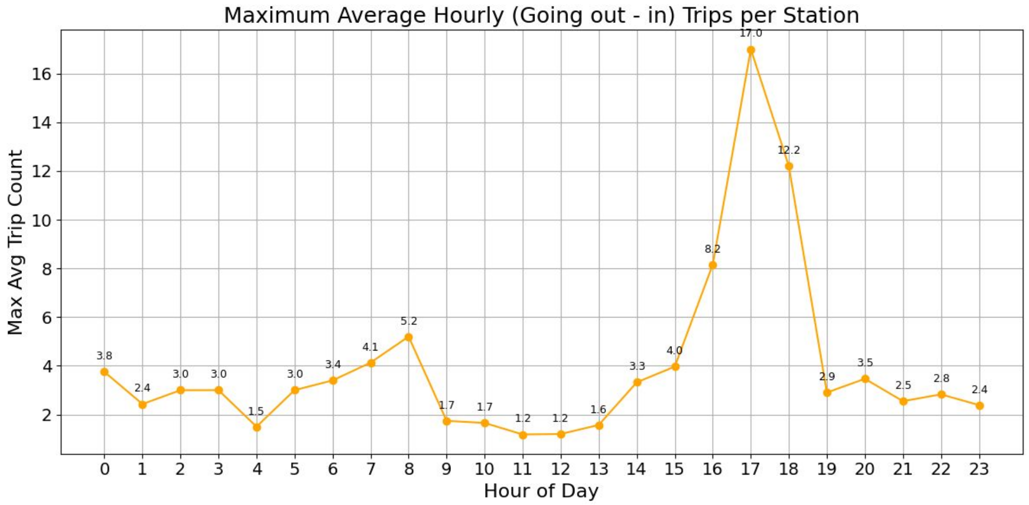
# Motivation

- Urban bike-sharing systems (e.g., CitiBike) offer convenient transportation but often face station imbalances (too full or too empty).
- **Urban Bike-Sharing Challenges**
    - Stations often become too full or too empty.
    - Leads to user frustration and operational inefficiencies.
    - Traditional methods are static and slow to adapt to real-time changes.
- **Our Goal**
    - Model CitiBike station dynamics using real-world trip data.
    - Train a reinforcement learning agent to learn adaptive rebalancing.
    - Optimize bike availability and improve commuter experience.

# Data

## Citi Bike Dataset

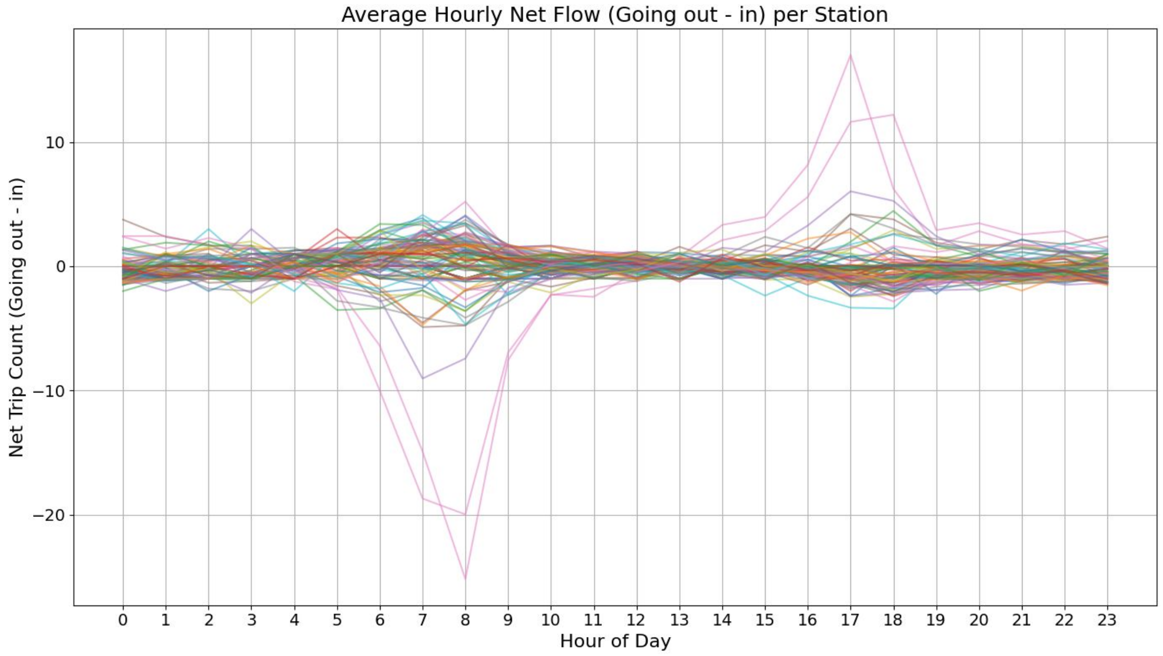
- Trip History Data (March 2025):
- Contains **detailed records of ~73,000 rides**, including ride ID, rideable type, start/end times and locations, station names and IDs, geolocation (latitude/longitude), and user type (member or casual).
- Curated to exclude staff, test trips, and rides shorter than 60 seconds.




Clear peaks at 17:00 - 18:00, up to 26 trips in an hour at some stations. Consistent daily outflow of 3 - 5 per hour from midday onward. Environment setted based on that.

NYU

# Data



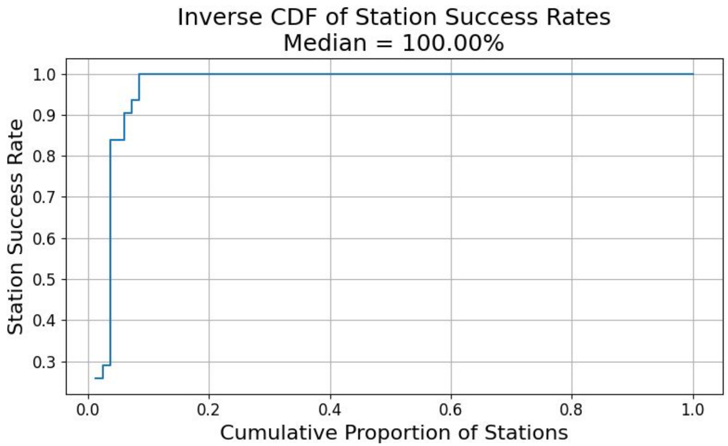

Due to the clear variance between stations, some drain bikes consistently, others accumulate, we segmented stations based on the demand level.

# Environment

- Time & Steps:
  - Discrete 2-hour decision intervals
  - 12 steps per 24-hour episode
- Initial Inventory:
  - [40, 30, 20] bikes at t=0
- Station Capacity:
  - 60 bikes for high-demand stations
  - 50 bikes for medium-demand stations
  - 40 bikes for low-demand stations
- Natural Refill:
  - Starting from 6:00 AM, +10 bikes are injected every 3 hours to simulate casual returns.
- Rebalancing (Action Space):
  - Every 2 hours
  - Actions: [–5, –3, 0, +3, +5]
  - (Negative = remove bikes, Positive = add bikes)
- State Representation:
  - [bike_count, hour_bucket (0–11), demand_level (one-hot)]
  - Demand levels: High / Medium / Low
- Demand-Level Target Ranges:
  - High: [20, 50] bikes
  - Medium: [10, 40] bikes
  - Low: [5, 30] bikes
- Reward:
  - +10 if stock within target range
  - -10 if outside range
  - -20 if empty/full
  - –0.5 per bike moved

# PPO Agent

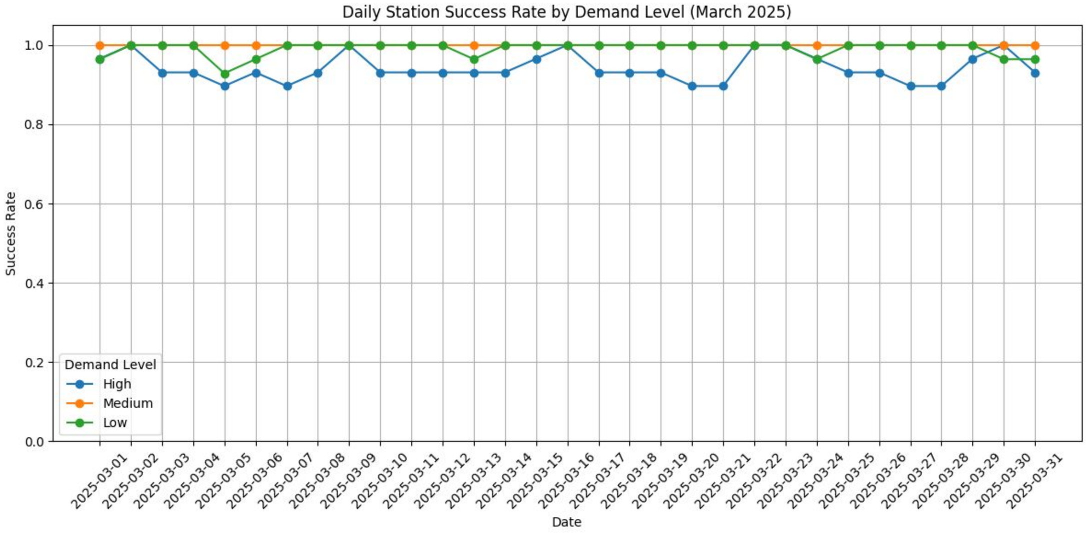
- **Algorithm**: Proximal Policy Optimization (PPO) via Stable-Baselines3
- **Policy**: MlpPolicy (fully-connected actor-critic network)
- **Actor**: produces a probability distribution over the 5 discrete rebalance actions
- **Critic**: predicts the state-value to guide policy updates
- **Advantage Estimation**: uses Generalized Advantage Estimation for low-variance learning
- **Objective**: clipped surrogate loss to constrain policy updates within a trust region
- **Training**: 240,000 timesteps for a single policy that generalizes across all stations




**Success Day:** For a given station, if at least 75% of that day's timestamps fall within the target range, the day is counted as a success.

**Success Rate:** The number of success days for a station / the total number of days (31).

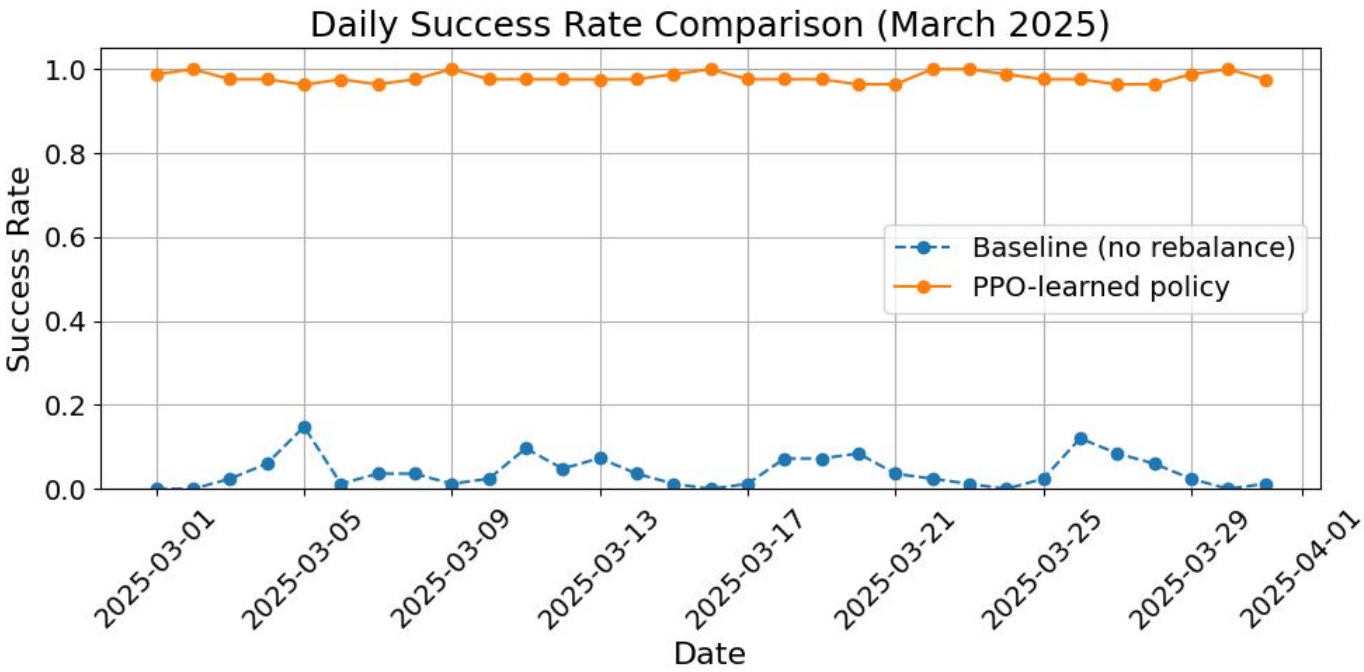
# PPO Agent




**Success Day:** For a given station, if at least 75% of that day's timestamps fall within the target range, the day is counted as a success.

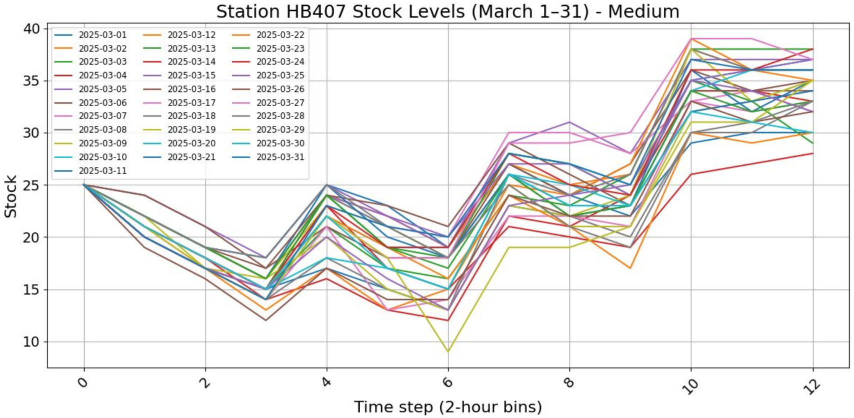
**Success Rate:** The number of stations with a success day / the total number of stations (82).

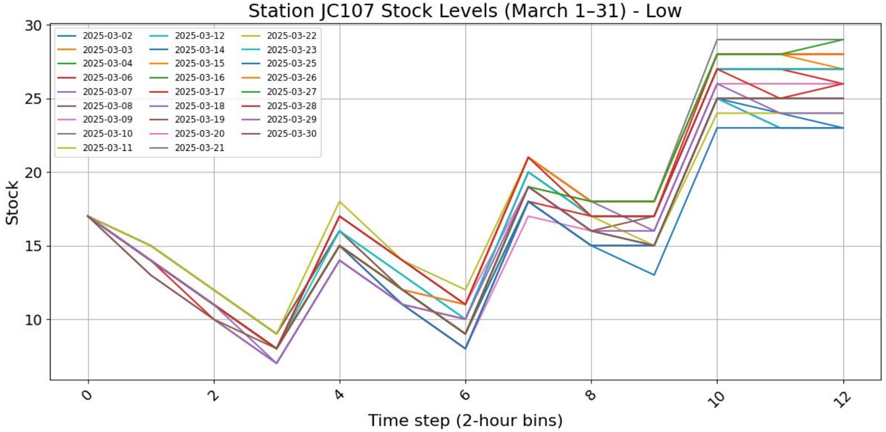
# PPO Agent

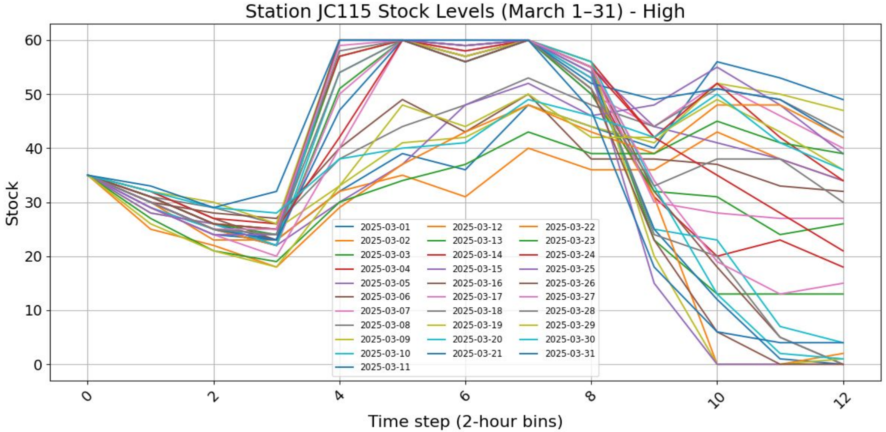


Daily Success Rate Comparison (March 2025)
Success Rate
1.0
0.8
0.6
0.4
0.2
0.0
Baseline (no rebalance)
PPO-learned policy
2025-03-01
2025-03-05
2025-03-09
2025-03-13
2025-03-17
2025-03-21
2025-03-25
2025-03-29
2025-04-01
Date

# PPO Agent




Target Stocks:

- Low: (5, 30)
- Medium: (10, 40)
- High: (20, 50)

NYU

# Q-Learning

### Configuration

- **Q-table:** defaultdict mapping each discretized state to action→value dict
- **Hyperparameters:**
  - Learning rate $\alpha = 0.1$
  - Discount factor $\gamma = 0.95$
  - Episodes = 20 000
- **Exploration:** ε-greedy, with ε starting at 1.0, decaying by ×0.9995 per episode down to 0.05

### Training Loop

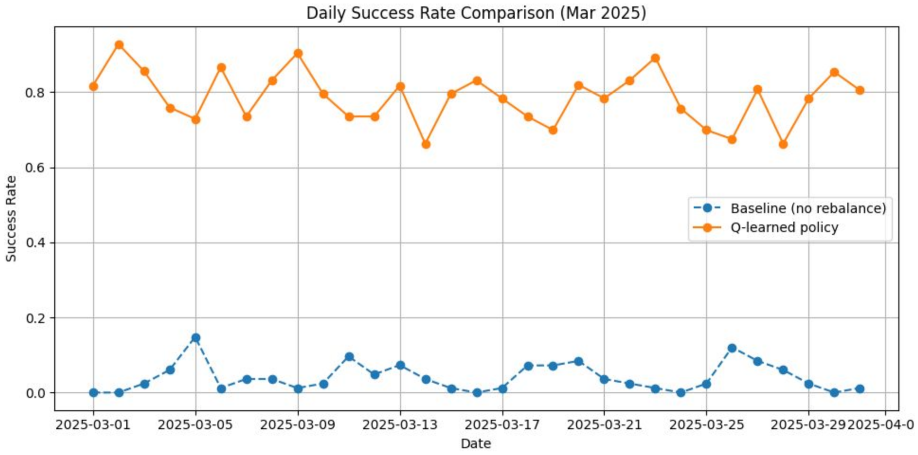
- Reset env → observe initial state s
- **While not done**:
  - Pick action a via ε-greedy
  - Step, receive reward r and next state s'
  - Update $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)]$
  - $s \leftarrow s'$
- **Decay** ε after each episode

### Policy Evaluation

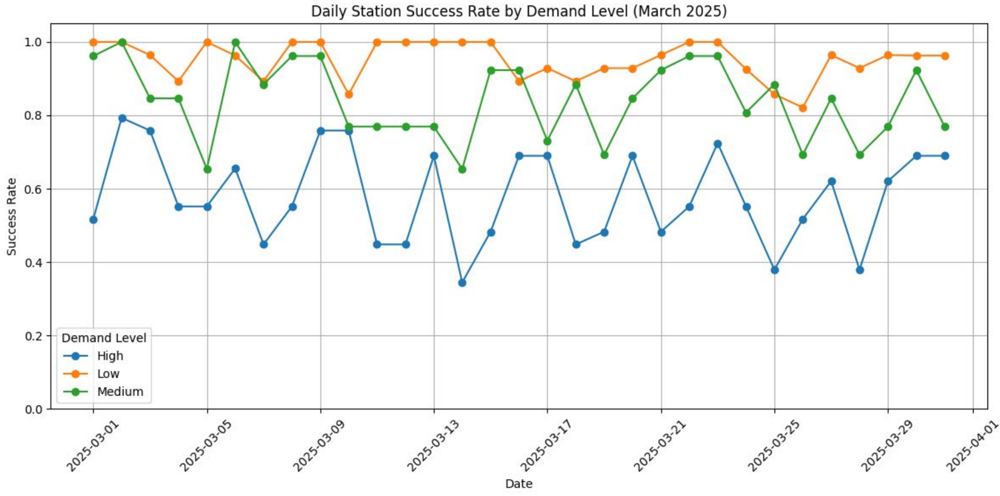
- **For every (station, date) pair:**
  - Instantiate fresh single-station JCEnvironment
  - Run greedy policy (always pick $\arg\max_a Q(s,a)$) to collect total reward
- Report average episode return

NYU

# Q-Learning

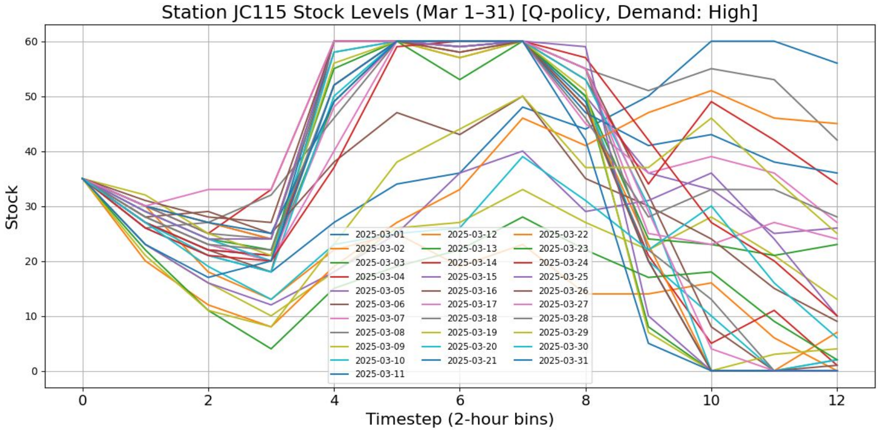


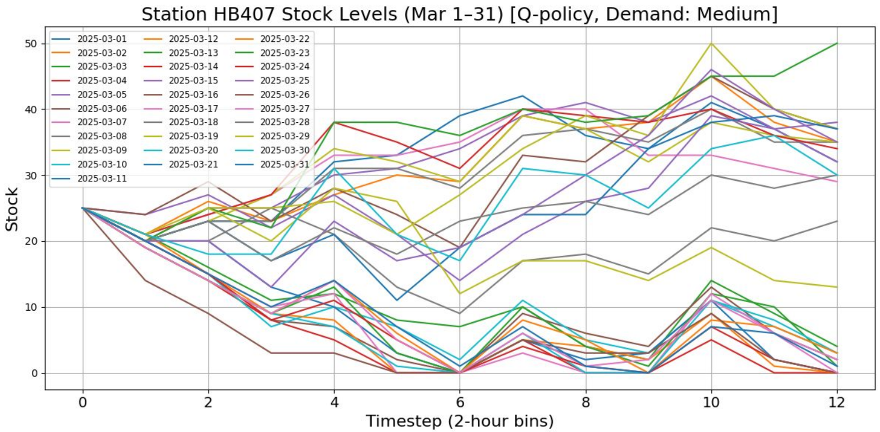
Daily Success Rate Comparison (Mar 2025)
Baseline (no rebalance)
Q-learned policy
Success Rate
Date
0.0
0.2
0.4
0.6
0.8
2025-03-01
2025-03-05
2025-03-09
2025-03-13
2025-03-17
2025-03-21
2025-03-25
2025-03-29

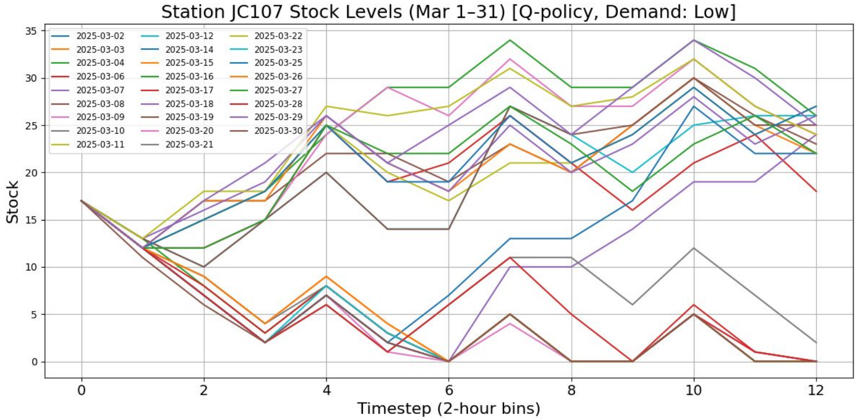



Daily Station Success Rate by Demand Level (March 2025)
Demand Level
High
Low
Medium
Success Rate
Date
0.0
0.2
0.4
0.6
0.8
1.0
2025-03-01
2025-03-05
2025-03-09
2025-03-13
2025-03-17
2025-03-21
2025-03-25
2025-03-29
2025-04-01

# Q-Learning




Target Stocks:

- Low: (5, 30)
- Medium: (10, 40)
- High: (20, 50)

NYU

# Deep Q-Learning with Online Updating

**Method Overview:**

- Training the DQN on fresh new data day-by-day during March 2025, station by station, while simultaneously updating the network weights
  - "Online": keep updating after each batch of experiences
- Interact with environment (action → step → observe → reward) in a loop until the day ends.

**Action Selection:**

- Using an epsilon-greedy policy; Epsilon starts at 0.2 and decays very slightly to encourage gradual exploration → exploitation shift

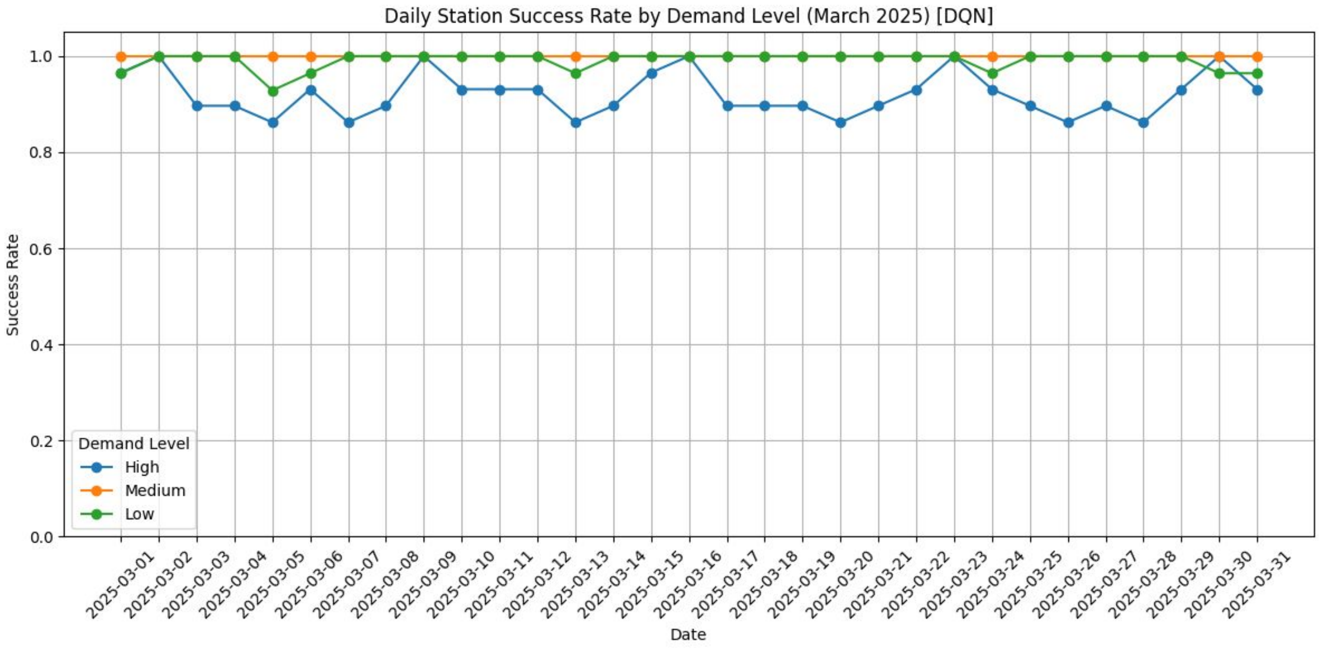
**Experience Collection:**

- Every (state, action, reward, next_state, done) tuple is <u>immediately</u> stored into a replay buffer.

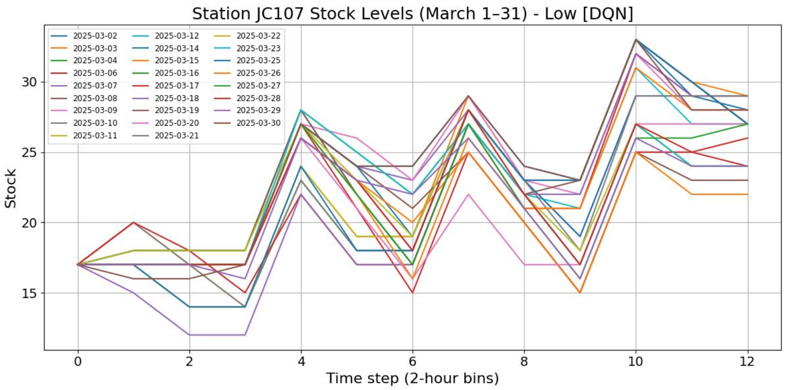
**Training & Updating:**

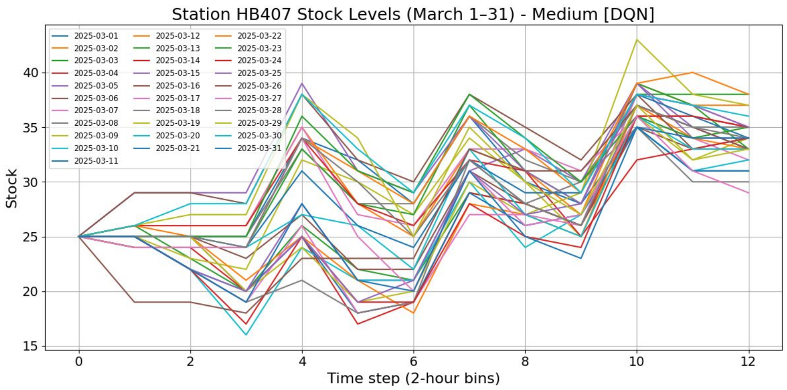
- The policy network is updated using sampled experiences with slight epsilon decay after each action, update the policy network weights using <u>Adam optimizer</u>
- Every 50 steps, the target network is synced with the policy network to stabilize training.

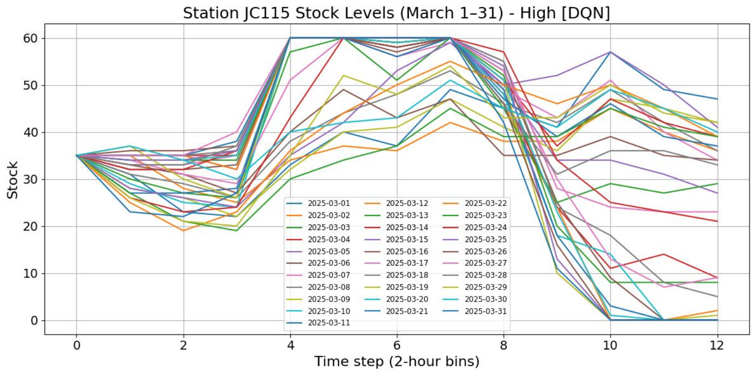
# Deep Q-Learning with Online Updating



# Deep Q-Learning with Online Updating



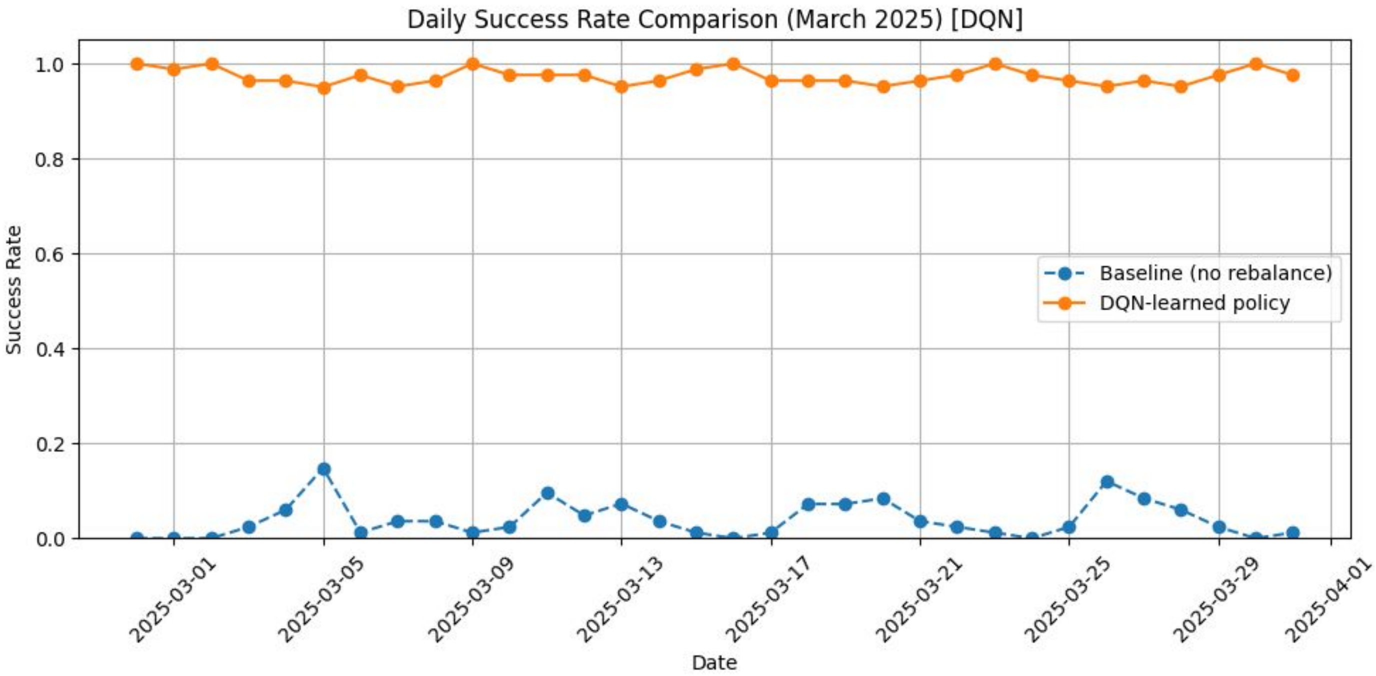




Target Stocks:

- Low: (5, 30)
- Medium: (10, 40)
- High: (20, 50)

NYU

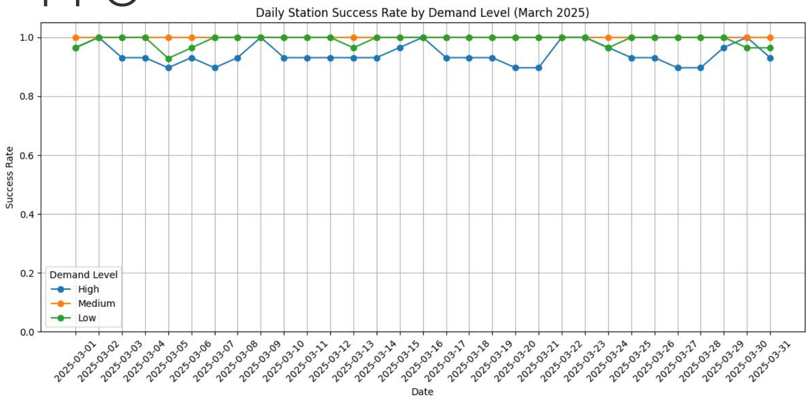
# Deep Q-Learning with Online Updating



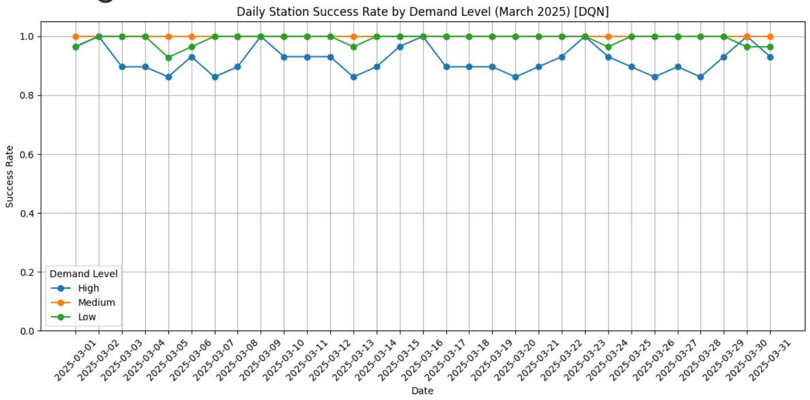
# Conclusion

- Overall, the performances of both PPO and DQN show promising result. PPO achieved the highest station success rate versus Q-Learning and DQN.
- PPO dominates in overall stability without requiring online updates or retraining.
- DQN provides adaptability for dynamic environments but shows more variability (especially for high demand)

PPO




DQN




NYU

# What Can Be Improved

- **Station-specific Modeling**
  - Current approach assigns simple “High/Medium/Low” demand categories.
  - Future: model each station’s realistic individual behavior dynamically based on actual flow patterns.
- **Natural Demand Variability**
  - Assumes average hourly net flows.
  - Future: introduce stochastic demand (simulate daily randomness and peak events like weekends, weather).
- **Policy Generalization**
  - Agent is trained and evaluated mostly on March 2025 data.
  - Future: test generalization to other months, unseen stations, and different seasonal dynamics.