Calyber: Reinforcement Learning for Ride-Sharing Optimization[1]

Group: Shang_chun_shan
Group members: Malcolm Zhao, Lingxin Li, Weixiao Wang

1. Overview

In the bustling streets of Chicago, the ride-sharing platform Calyber has emerged as a pivotal solution to urban transportation challenges, offering shared ride services that enhance cost efficiency and reduce congestion and emissions. This project aims to boost Calyber's operational efficiency and profitability by applying advanced machine learning techniques, notably leveraging Dueling Deep Q-Networks (Dueling DQN) within a classical Markov Decision Process (MDP) framework, which is a typical solution for dynamic pricing problems.

Ride-sharing platforms like Calyber operate in dynamic and complex environments, requiring real-time pricing and matching decisions that balance immediate profits with long-term customer satisfaction. Traditional models require help with the fast-paced changes in urban traffic and rider behavior. However, leveraging reinforcement learning allows continuous learning and adaptation, presenting a unique solution to these challenges.

The goal of this project is twofold:

- **Optimize Pricing**: Develop an intelligent pricing model that adapts to various factors such as demand fluctuation, rider preferences, and competitive pricing strategies.
- **Efficient Rider Matching**: Enhance the matching algorithm to minimize waiting times and driving costs, thereby maximizing the utilization of vehicles and improving rider satisfaction.

Such a decision framework is a classical MDP in which both decisions and randomness will play a role in the feedback (rewards) of the environment. Given a state to make decisions, different options will yield an immediate reward and a future impact on the environment. Q-learning established on MDP has assumptions to quantify the combination of the two, namely Q-value, to imply the optimal decision in the given state. In other words, the decision with the largest Q-value is the optimal one to take in the state.

$$Q(s,a) \leftarrow_{a} r + \gamma \cdot max_{a'} Q(s',a')$$

In the equation provided by Géron (2019), each state-action pair (s, a) has reward r that the system receives immediately, plus the sum of discounted future rewards it expects to get. To estimate this sum, we take the maximum of the Q-value estimates for the next state s', which is the one for the next state if taking optimal decisions for all future states. To calculate the long-term effect, we use the discount factor γ to quantify the significance of future rewards, allowing it to integrate these future potential outcomes into the current decision-making process effectively.

To set up Q-learning based on the above settings, a simulation and training process was conducted.

2. Environment and Simulation

To achieve our project objectives in optimizing ride-sharing operations, we employ a Dueling DQN approach within a simulated environment based on six one-hour windows of rider information (11,788 riders in total) who arrived at the platform in the past to simulate the dynamics of the ride-sharing platform. This environment is specifically designed to mirror real-world ride-sharing operations, allowing for meticulous testing and refinement of our algorithms. Here's a detailed breakdown of this environment's setup and operational flow:

[1] Training and Simulation codes are released in the Calyber Github repository: shang chun shan

2.1 Preparation

First, we created a dataset of rider interactions to reflect their behavior patterns. This data includes rider pickup and dropoff locations, willingness to pay (WTP), tolerance (patience), and other relevant attributes necessary for simulation. More details will be provided later.

We also initialized some parameters:

- **Interarrival Time** λ_1: We assumed the rider coming event follows a Poisson Process, and its hyperparameter assumptions will be provided later.
- **Maximum Simulation Time**: This is the upper limit of the simulation time for each run. Given that the training data is provided on a one-hour frequency, we set the limit to 3600 seconds as a termination criterion.
- **Maximum Queue Length**: Due to the limitation given by computation abilities, we restricted the number of possible matching options to 25. It is worth noticing that we did not restrict the size of the queue. Rather, the system has restrictions that only the first 25 riders in the queue can be matched, while the remaining ones will be kept in the queue until the prior riders have left. Hopefully, the model will gradually learn to allow a limited number of riders in the queue (promising to be smaller than 25).

2.2 Dataset Split

Given the weekly sequential nature of the training data, we kept the first five weeks as training data and the last week as test data.

2.3 Simulation of Coming Rider and Assumptions

For each state, we considered an incoming rider. The incoming rider was randomly selected from training data, and the arriving time was sampled according to the assumed Poisson Process. The rider's WTP and tolerance were also simulated according to the following assumptions.

2.3.1 Simulation of the interarrival time

It was assumed that the rider coming follows a Poisson Process, so we simulated the interarrival time from exponential distribution:

$$INTERARRIVAL\ TIME \sim exp(\lambda_1)$$
$$where\ \lambda_1 = avg(observed\ interarrival\ time) = 1.83\ secs$$
$$(point\ estimator\ in\ the\ training\ data)$$

2.3.2 Simulation of willingness to pay (WTP)

We assumed that the WTP follows a uniform distribution in a specific range. The two endpoints in the range are determined in two cases:

- **Converted Riders**: If a rider is converted, its WTP is at least the quoted price, and it will not exceed the observed max quoted price (0.98). So, we simulated converted customers' WTP from uniform distribution:

$$WTP \mid CONVERTED \sim Uniform\ (quoted\ price,\ 0.98)$$

- **Lost Riders**: If a rider is not converted, its WTP is below the quoted price and no less than the min quoted price observed, which is 0.35 in the training dataset. 0.35 also makes sense because the

matching allows a maximum 50% reduction in the driving cost. Therefore, any price below 0.35 will be unprofitable. So, we simulated non-converted riders' WTP from uniform distribution:

$$WTP \mid LOST \sim Uniform\ (0.35,\ quoted\ price)$$

2.3.3 Simulation of tolerance

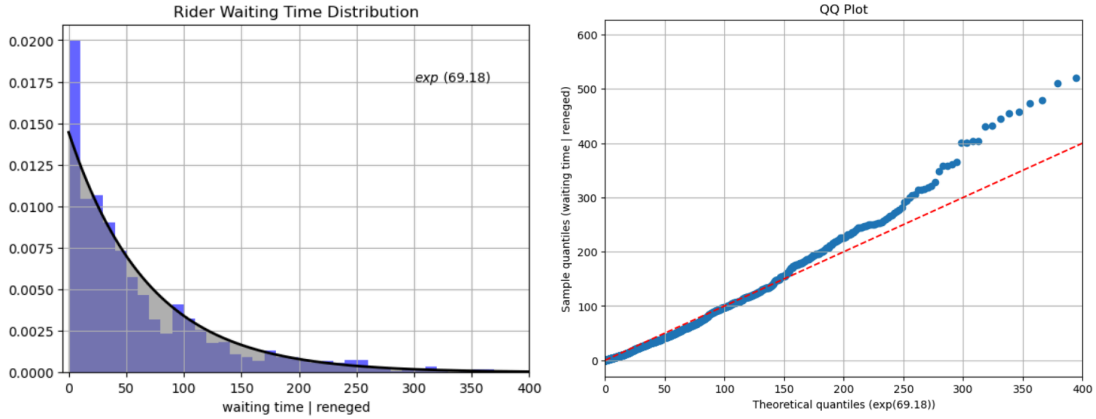
Given the histogram and QQ plot observation (Figures 1 & 2), we believe the tolerance follows an exponential distribution. The simulation for such a random variable is determined in three cases:

- **Reneged Riders**: For riders who reneged, their tolerance equals their waiting time, as they left immediately after their patience expired. So we set:

$$TOLERANCE \mid RENEGED = tolerance \mid reneged = waiting\ time$$

- **Lost Riders**: Tolerance for these riders was not given. As stated before, we believe it follows an exponential distribution. So we sampled it from the below assumption:

$$TOLERANCE \mid LOST \sim exp(\lambda_2)$$
$$where\ \lambda_2 = avg(waiting\ time|reneged) = 69.18\ secs$$
$$(point\ estimator\ among\ the\ reneged\ riders\ in\ the\ training\ data)$$



Figures 1 & 2: Reneged Rider Waiting Time Distribution and QQ plot

- **Matched Riders**: For converted and matched riders, their tolerance is at least as long as the waiting time they experienced. This leads to the **right-censoring issue** since we only observe that it exceeds the waiting time.

  To deal with this problem, we utilized exponential distribution's memoryless property. This property implies that the future probability distribution does not depend on how much time has already passed and observed.

  So, we first simulated tolerance from the exponential distribution similar to Lost Riders. Then, we added their waiting time to the sampled values so that the summation acts as a realization of the tolerance.

$$TOLERANCE \mid MATCHED \sim exp(\lambda_2)$$
$$tolerance \mid matched = tolerance \mid lost + existing\ waiting\ time$$

Given the above assumptions, the environment will set an incoming rider and adjust the timer to the rider's coming time.

2.4 Process action

Then, given a decision combination that includes pricing and matching, the environment will process the corresponding decisions.

- **Price decision:** The environment will check if the price is less than or equal to the simulated WTP of the incoming rider. If the rider's WTP is above the price, the rider will be converted and profit is generated based on the ride's price and trip length.
- **Match decision:** Match decision is an index referring to either a position in the queue or keeping the rider waiting. To have a matching outcome to a waiting rider, the incoming rider has to be converted, and the queue should not be empty. If so, the environment will match the incoming rider with the specified one in the queue. Then shared ride costs and profits will be calculated. Otherwise, given a converted rider, if either the matching decision is to keep the rider waiting or the queue is empty, the environment will put the rider in the queue.

Also, after this process, the environment will check if any riders in the queue have run out of patience. If so, it will clear the rider from the queue and make adjustments to the profit.

2.5 Update environment state

Finally, the system will provide an immediate reward based on the above environment adjustments (reward design in later parts) and jump to the next state by setting another incoming rider.

2.6 Environment Termination

As mentioned before, the environment keeps a timer during the simulation. Once the total simulated time has exceeded the upper limit, the system will terminate the simulation, clear all riders in the queue by treating them as reneged riders, provide rewards caused by this clearing process, and output the final profit.

3. Agent

The goal of our agent is to gather simulation information and conduct training on the model. By utilizing a Dueling DQN approach, the agent can use the model to predict and evaluate the outcomes of different actions given state information. This lets the model learn to make decisions that maximize overall profit over time.

3.1 Input Design

The model (neural network) takes a dual-channel input, which is the current state, including incoming rider and queueing information for the top 25 riders. Incoming riders and riders in the queue share the same input feature structures:

- arrival_time: simulated/actual arrival time
- pickup_cluster, dropoff_cluster: we conducted K-Means Clustering, taking the coordinates (latitude, longitude) as attributes for each area and aggregating the 70 areas into 7 clusters.

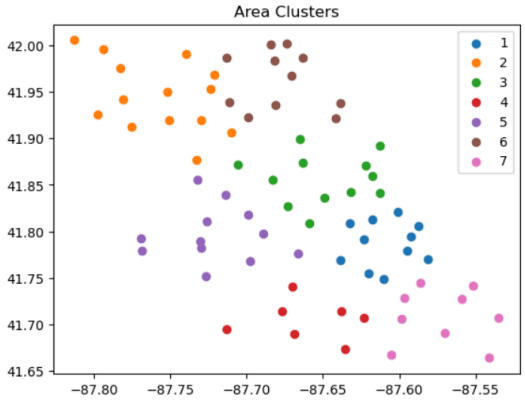


Figure 3: Clusters for Areas Given

- pickup_lat, pickup_lon, dropoff_lat, dropoff_lon, solo_length: consistent with the given data

3.2 Decision Design

We set a finite number of decision options due to the nature of Q-learning.

- **For pricing decisions**, we generated a list ranging from the lowest to the highest price, divided evenly into 11 options. The lowest and highest prices were determined through trial and testing as 0.45 and 0.9, respectively.
- **For matching decisions**, Considering that we observe at most 20 riders waiting simultaneously in the training data, we eventually set 26 options to indicate 25 top riders in the queue to be matched and 1 option for not matching.

Two networks are designed to predict the Q-value for each decision, either separately or collectively. More details will be provided later.

3.3 Overall Neural Network Design

The neural network architecture can be elaborated in the following:

1. **LSTM Layers for Queue Input**: These are employed for the queueing input to capture temporal dynamics and dependencies in sequences of rider arrivals.
2. **Dense Layers for Rider Input**: These are employed after the incoming rider input.
3. **Fully Connected Layers**: These are employed after concatenation between Block 2 and 3.
4. **Output Layers for State Prediction**: Sequentially after Block 3, single value output is used to predict the current state's Q-value, assuming optimal decision is taken.
5. **Output Layers for Decision Prediction**: These are used to predict each decision's Q-value.
6. **Final Transformation Blocks (Dueling Mechanism):**
    - Subtracting each decision prediction by the largest value of them (which is the optimal decision's Q-value) to calculate the Q-value deduction (advantages) if the given decision is taken instead of the optimal one.
    - Add the predicted state value (Block 4) to the advantages. This is to recalculate each decision's Q-value.
    - The above transformation could be summarized in the below equation, where $Q(s, a)$ is the Q-value of state-action pair (s, a), $V(s)$ is the value of state s given that optimal decision a^* is

taken, and $A(s, a)$ is the advantage of taking action in states compared to all other actions in s, where $A(s, a^*) \geq A(s, a)$ and $A(s, a^*) = 0$:

$$Q(s, a) = V(s) + A(s, a)$$

- The dueling mechanism allows smoother convergence. Besides, separating state and decision values prediction enables it to observe delicate and subtle changes for each decision standing on the overall prediction for the given state.

3.4 Model Output Architectures: Dual v.s. Single

We explored two types of neural network architectures:

3.4.1 Dual-Channel Output Network

This network would split the output of Block 3 into two channels, one to predict pricing decisions' Q-value and one for matching. Since selection for one channel would naturally influence the outcome for another channel (e.g. an appropriate matching would possibly increase the Q-value for setting a low price), to consider this dependency issue, we added a cross-summation layer so that the intermediate output for one channel is added into the other channel.

Such designs separate the reward into pricing and matching channels with the below definition:

- **Discount factor (γ)**:
  - As when setting prices, the cost savings would happen in later stages, pricing decisions will have a long-lasting effect. So the discount factor for pricing is set to be 0.9999.
  - In contrast, most matching decisions would receive immediate feedback, and long-term benefits are mostly not obvious, so the discount factor is set at 0.965.
  - However, for the reward of not matching, the design is incapable of handling its long-term benefit, which can be regarded as a future enhancement aspect. In contrast, the cost for not matching can be quantified through discount restoration calculation, which will be discussed later.
- **Immediate Profit**: The profit generated by a converted customer is counted as the reward for pricing decisions, which is the revenue minus the single ride cost.
- **Lost Profit**: Opportunity cost (below equation) is calculated as a penalty for setting high prices so that a rider is lost. In the equation below, cost_saving_factor is the average cost-saving fraction because of matching decisions in the training data with a value of 0.86. Price divided by WTP depicts how much the pricing decision deviates from the optimal price for the rider, and 5 is the hyperparameter that controls how aggressively the agent is penalized due to losing revenue. A higher index leads to lower pricing.

$$pricing\ reward\ (lost\ profit) = -\ max\{0,\ (WTP - 0.7 \cdot cost\_saving\_factor) \cdot solo\_length \cdot (price\ /\ WTP)^5\}$$

- **Cost-Saving for Pricing Given a Shared Drive between (i and j)**: Cost saving is added to the pricing reward to compensate for the overestimate for driving cost in the Immediate Profit section.

$$pricing\ reward\ (shared\ drive) = (i\_solo\_length + j\_solo\_length - trip\_length) \cdot 0.7$$

- **Cost-Saving for Matching Given a Shared Drive**: Matching rewards are characterized by cost comparisons, such that appropriate matching leads to smaller costs, so they will always be nonpositive. To better capture its change, "earnings" is calculated to control how aggressively the agent is awarded to make matchings, which is the estimated profit for the two riders. In the below

equations, when earnings are positive, the more earnings acquired, the smaller the cost will be counted as a penalty. And similar for the opposite. The base number in the final equation (0.6) is the hyperparameter used to control aggressiveness. The larger the base is, the more frequent matching is encouraged.

$$cost = 0.7 \cdot trip\_length$$
$$earnings = price \cdot i\_solo\_length + WTP \cdot j\_solo\_length \cdot 0.9 - cost$$
$$match\ reward\ (shared\ drive) = -(cost \times 0.6^{earnings})$$

- **Renege Cost**: The cost for a reneged rider will first be transformed so that it can be restored to the state when the agent decides not to match the rider. In the below equation, "num states" refers to the estimated number of states between the rider coming to and leaving the system, where "ave interarrival time" is the hyperparameter for the assumed Poisson Process (λ_1 = 1.83 secs). -2 is the hyperparameter used to control rider waiting time. The more negative the value is, the more waiting time the rider will suffer. Then, the "restored cost factor" is calculated because of the discount equation for Q-value (discounted future reward = $\gamma^{num\ states}$·future reward). The factor is used to enlarge the cost at the current stage so that it will be restored to the actual cost in the state when the rider comes. As the reward for matching is measured based on the opportunity cost concept, we believe that cost comparison should exclude discount calculation for fair comparison purposes.

$$num\ states = max\{-2, \lfloor \frac{current\ time - arrival\ time}{ave\ interarrival\ time} - 2 \rfloor\}$$
$$restored\ cost\ factor = \gamma^{-num\ states}$$
$$match\ reward\ (renege\ cost) = -(0.7 \cdot solo\_length) \cdot restored\ cost\ factor$$

The dual-channel output network has relatively fewer decisions to predict. However, it requires a much more complicated design for rewards and has challenges dealing with dependency issue. In the training stage, the network encountered difficulties with convergence and had considerable fluctuations in the matching reward (caused by the discount restoration calculation).

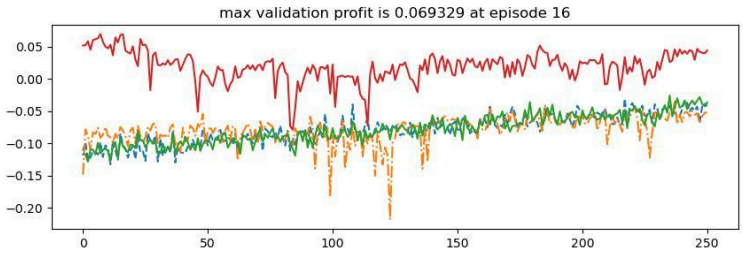


Figure 4: Dual Channel Training Log (red: test profit; yellow: matching reward)

3.4.2 Single-Channel Output Network

For this network, we created a matrix where each row represents a pricing decision and each column represents a matching decision. Then we used the neural network with a single channel output structure to predict the Q-value for each entry in the matrix. Such a design overcomes the cross-channel dependency issue and has a rather simple reward design:

- **Discount Factor (γ)**: Combining the two decisions naturally forces each decision to have a long-lasting influence. So we set $\gamma = 1 - 10^{-10}$
- **Revenue:** Revenue generated by a converted rider is counted as the reward (not the profit).
- **Matching Cost**: We still used the "earnings" and the same techniques to control how aggressive the agent should be to match a rider and select 0.48 as the final realization value.

- **Renege Cost**: As we selected a rather large discount factor, we decided not to conduct discount restoration and simply subtract the renege cost from the reward.

In the training stage, we observed a better performance with larger profit on the test set and continuously improving patterns:

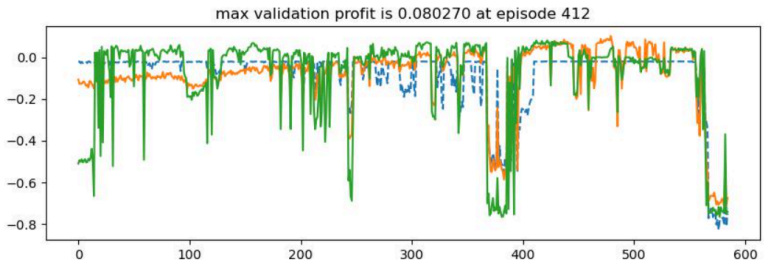


Figure 5: Single-Channel Training Log (green: test profit)

Consequently, we selected the single-channel output network as the final network design.

3.5 Training Process

- **Computation Resources**: We applied support from Berkeley Open Computing Facility and were eventually granted access to industrial computing resources, including 2 Intel Xeon Platinum 8352Y CPUs (32c/64t @ 2.4GHz), 4 NVIDIA RTX A6000 GPUs and 256GB ECC DDR4-3200 RAM. These configurations were used for model training.

- **Agent Memory**: During each training episode, the agent will simulate a series of actions in the resetted environment till termination. We used a memory queue to store the simulated information, including state information, action, next state, and reward.

- **Epsilon Greedy Search Policy**: During simulation, the agent will select actions either randomly or following the prediction of the model controlled by a probability hyperparameter, epsilon. In the early episodes, the agent will have a higher probability of conducting randomization to let the model learn evenly across all decision pools. In the later episode, the agent will be more likely to select decisions following the model prediction. However, probability (epsilon) will always be larger than 1% so that the agent always has a chance to cover the entire decision pool.

- **Experience Replay**: When the memory has stored enough observations, the agent will start to train the model. In each episode, 200 batches will be sampled, with gradually decreasing batch sizes (from 256 to 32). During backpropagation, the summation of immediate reward and discounted predicted Q-value for the next state will be treated as observations, and the prediction in the current stage will be used as predicted values. The agent will gradually adapt to predict benefits from a short-term view to the long-term.

- **Prioritized Experience Replay**: When selecting batches, the agent follows the technique of prioritizing experience replay. All observations will initially have the same probability of being selected into training batches. However, as the agent gradually touches each observation, it will record the deviation from the predicted values to the actual ones. The higher the deviation is recorded, the higher the probability this observation will be reselected in the future.

4. Results

The evaluation of our model on the final test set and detailed outcomes based on the critical performance metrics defined for this project are as follows:

Model	Profit ($/min)	cost_efficiency	match_rate	Revenue ($/min)	Cost ($/min)
Dueling DQN	1.9639	0.0257	0.9568	14.3920	12.4282
Benchmark	3.3612	0.1572	0.5762	22.3343	18.9731

cont.	Throughput (#/min)	conversion_rate	Ave_payment ($/mile)	Ave_quoted_price ($/mile)	Ave_wait_time (sec)
	4.8767	0.1718	0.7996	0.8057	5.7560
	9.2411	0.3255	0.6860	0.7137	56.3866

Table 1: Performance Metrics of Test Data

The analysis indicates that while our model excels in efficiency, particularly in payment and waiting time, it falls short in revenue generation and has problematic conversion rates compared to the benchmark.

Then, we looked at the detailed matching and pricing decision of our modified model re-simulated in the validation data released on April 12th.

Match Decision	Match	0	1	empty queue	25 (not match)
	Number	1151	262	3719	6

Table 2: Match Decision of Test Data

Price Decision									
	Price	0.8094	0.8542	0.7646	0.7198	0.6302	0.6750	0.5854	0.4958
	Number	4556	264	183	106	23	4	1	1

Table 3: Price Decision of Test Data

The pricing decision model predominantly selected higher price points, especially focusing around $0.8094, which was chosen 4556 times. Given that our model has a nice performance in terms of average payment, the pricing strategy successfully targets a price point that is beneficial in terms of profitability and rider acceptance.

In contrast, our model's high matching rates result from an overly aggressive matching strategy. It can be observed that the model has strong inertia to match the incoming rider with the existing riders in the queue (only 6 riders are actively kept in the queue). This aggressive approach implies the system pairs riders without sufficiently considering the quality of these matches.

5. Future Enhancements Aspects

To further refine and enhance the performance of our ride-sharing model, we will focus on several strategic areas.

The design of the reward system within our ride-sharing model plays a crucial role in driving the system's behavior, particularly in how it manages the queue. The current reward design may lack the ability to adequately highlight the benefits of placing riders into the queue, potentially leading to suboptimal matching

decisions and inefficiencies. In the future, adding a more delicate design for rewards and penalties related to this aspect is considered an improvement aspect.

In ride-sharing systems, clustering geographic locations into zones can boost efficiency. However, our model's limitation to 7 clusters may not fully capture urban diversity. Especially in the current context, we do not have adequate information to characterize the pickup and dropoff information. Simply treating latitude and longitude as input and conducting a simple cluster are not wise choices for feature engineering. In contrast to our methodology, the benchmark tried over 30 clusters, which inspired us to reflect on our design. In the future, it is recommended to expand the number of geographic clusters and try more cluster approaches to better capture the specific travel patterns.

References

Géron, A. (2019). Hands-on machine learning with scikit-learn, Keras, and TensorFlow: concepts. *Aurélien Géron-Google Kitaplar, yy https://books.google.com.tr/books*