Optimizing matching radius for ride-hailing systems with dual-replay-buffer deep reinforcement learning

Gao, Jie; Cheng, Rong; Wu, Yaoxin; Zhao, Honghao; Mai, Weiming; Cats, Oded

DOI
10.1016/j.cie.2025.111296

Publication date
2025

Document Version
Final published version

Published in
Computers & Industrial Engineering

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Optimizing matching radius for ride-hailing systems with dual-replay-buffer deep reinforcement learning

Jie Gao [a], Rong Cheng [b], Yaoxin Wu [c],*, Honghao Zhao [a], Weiming Mai [a], Oded Cats [a]

[a] *Department of Transport and Planning, Delft University of Technology, Delft, The Netherlands*
[b] *College of Transportation Engineering, Dalian Maritime University, Dalian, China*
[c] *Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, Eindhoven, The Netherlands*

ARTICLE INFO

Keywords:
Ride-hailing
Matching radius
Markov decision process
Deep reinforcement learning

ABSTRACT

The matching radius, defined as the maximum pick-up distance within which waiting riders and idle drivers can be matched, is a critical variable in ride-hailing systems. Optimizing the matching radius can significantly enhance system performance, but determining its optimal value is challenging due to the dynamic nature of ride-hailing environments. The matching radius should adapt to spatial and temporal variations, as well as to real-time fluctuations in supply and demand. To address this challenge, this paper proposes a dual-reply-buffer deep reinforcement learning method for dynamic matching radius optimization. By modeling the matching radius optimization problem as a Markov decision process, the method trains a policy network to adaptively adjust the matching radius in response to changing conditions in the ride-hailing system, thereby improving efficiency and service quality. We validate our method using real-world ride-hailing data from Austin, Texas. Experimental results show that the proposed method outperforms baseline approaches, achieving higher matching rates, shorter average pick-up distances, and better driver utilization across different scenarios.

1. Introduction

Ride-hailing services such as Uber, Lyft, and DiDi have reshaped urban mobility by providing a flexible and accessible alternative to traditional transportation. As cities worldwide pursue net-zero emissions by 2050, ride-hailing has become an integral part of sustainable mobility strategies (Wu et al., 2022). These services reduce reliance on private car ownership and improve vehicle utilization, contributing to lower emissions and enhanced transportation efficiency.

Despite these advantages, efficient rider-driver matching remains challenging due to the highly dynamic and imbalanced nature of ride-hailing markets. A key factor influencing matching efficiency is the *matching radius*, which determines the maximum distance within which a driver can be assigned to a ride request (Xu et al., 2020). Most existing studies focus on refining matching algorithms or optimizing the timing of assignments, yet they often treat the matching radius as a fixed parameter rather than a decision variable (Wang & Yang, 2019). However, a static matching radius fails to account for real-time supply–demand fluctuations, which play a crucial role in service quality and operational efficiency (Chen et al., 2025; Xu et al., 2020). When the matching radius is too small, ride requests may go unfulfilled due to an insufficient number of nearby drivers. Conversely, an excessively large radius increases passenger wait times and leads to inefficient driver allocation. A more adaptive approach that dynamically adjusts the matching radius based on system conditions can improve ride-hailing efficiency by forming more effective and balanced matching pools.

To this end, we propose a deep reinforcement learning (DRL) approach for optimizing the matching radius in ride-hailing systems. Compared to the fixed radius strategy, the proposed method can promise better ride-hailing matching results by adaptively determining the matching radius according to the system dynamics. Specifically, we first formulate the matching radius optimization problem as a Markov Decision Process (MDP), in which the policy is parameterized as a neural network to learn adaptive matching radius at each decision step. The overall performance of the ride-hailing system is considered as the reward, which comprehensively concerns the matching rate, the average pick-up distance, and driver utilization rate. Besides the fully connected hidden layers, we split the input and output layers of the neural network to process local observations in different regions and calculate the respective actions (i.e., matching radii), which enhance the awareness of local dynamics, such as regional supply–demand relationships. To train the neural network, we propose a Dual-replay-buffer Deep Deterministic Policy Gradient (D-DDPG) algorithm, which

* Corresponding author.
E-mail addresses: J.Gao-1@tudelft.nl (J. Gao), r.cheng@tue.nl (R. Cheng), y.wu2@tue.nl (Y. Wu), zhh2000522@163.com (H. Zhao), w.m.mai@tudelft.nl (W. Mai), o.cats@tudelft.nl (O. Cats).

https://doi.org/10.1016/j.cie.2025.111296

Available online 21 June 2025

introduces two reply buffers to differentiate the transitions with elite actions from the standard ones, and thus facilitates the learning from the matching radii that result in superior ride-hailing performance.

This work makes three key contributions to optimizing the matching radius in ride-hailing systems:

- We formulate the dynamic matching radius decision as a Markov Decision Process (MDP) and introduce a neural network architecture that captures both local and global system dynamics, supporting more informed region-specific radius decisions.
- We develop a Dual-replay-buffer Deep Deterministic Policy Gradient (D-DDPG) algorithm with an elite-action replay buffer which enhances learning efficiency by prioritizing transitions that lead to superior system performance.
- We develop a realistic ride-hailing simulator calibrated with real-world data from Austin, Texas, and conduct extensive experiments under various supply–demand scenarios. The results demonstrate that our approach consistently outperforms fixed-radius baselines and achieves a favorable balance across key performance metrics, highlighting its robustness and adaptability in dynamic ride-hailing environments.

The remainder of this paper is organized as follows. Section 2 reviews the relevant literature and highlights the research gap. Section 3 introduces the problem formulation, followed by the proposed methodology for learning the optimal matching radius in Section 4. Section 5 presents the experimental setup and results. Finally, Section 6 concludes the paper and outlines potential directions for future research.

2. Literature review

A key operational challenge for ride-hailing systems, as with any on-demand transportation system, is the problem of matching, finding a suitable driver to serve a ride request. Significant research has focused on designing and analyzing matching strategies to improve the efficiency and performance of ride-hailing systems. According to a comprehensive review by Wang and Yang (2019), matching approaches are broadly categorized into two types: greedy matching and batched matching. Greedy matching algorithms, such as those proposed in Feng et al. (2021), Lee et al. (2004), aim to assign the nearest driver or the shortest-travel-time driver to each individual ride request. Although these methods are easy to implement and manage, they are myopic in the sense that they prioritize immediate individual rider satisfaction over efficient resource utilization across many riders, which jeopardizes rider satisfaction at a larger scale. As an alternative, batched matching strives to accommodate the needs of more riders at a time by optimizing the matching among a group of drivers and riders accumulated in a pre-determined batching window. To maximize the matching rate in a batched matching solution, Zhang et al. (2017) propose a combinatorial optimization model to solve the order dispatch (matching) problem at Didi Chuxing. With the consideration of future rider demand, (Lowalekar et al., 2018) formulate a multi-stage stochastic optimization formulation to maximize the number of matched requests in a batch. To reduce the riders' total waiting times within a batch, (Gao et al., 2020) propose a learning-based approach that integrates machine learning with a two-stage stochastic programming model to guide open driver rebalancing prior to the batching process. For the benefit of drivers, Zhan et al. (2016) propose two matching algorithms, namely optimal matching and trip integration, to minimize drivers' idling driving time and distance across all open drivers in the system. The computation results show that these algorithms could find the optimal strategy that minimizes the cost of empty trips and the number of taxis required to serve all observed trips. In addition, Gao et al. (2021) introduce a data-driven optimization framework for ride-hailing matching with the goal of maximize the number of matched riders and drivers. More recent developments expand beyond traditional ride-hailing systems. Guo et al. (2024) address the autonomous vehicle ride-hailing problem using a multi-objective optimization approach that incorporates interpretable demand predictions, while (Essus et al., 2024) extend this line of work to ambulance services, developing a dynamic relocation policy based on real-time vehicle utilization metrics to improve patient survival outcomes.

While traditional approaches rely on deterministic optimization frameworks, the dynamic and highly evolving nature of ride-hailing systems necessitates more adaptive solutions. Reinforcement Learning (RL) has emerged as a powerful tool to address these challenges, offering the flexibility to model and solve dynamic decision-making problems in real time. Readers are referred to Liu, Jia, et al. (2022), Qin et al. (2022) for comprehensive reviews of RL applications in ride-hailing systems. For instance, Jin et al. (2019) develop a hierarchical multi-agent RL framework to optimize fleet management and order dispatching across regional zones, leveraging a hierarchical structure to coordinate long-term benefits. Similarly, Liu, Wu, et al. (2022) introduce a single-agent RL model for vehicle dispatching, reallocating vacant vehicles to regions with higher demand. These approaches highlight the ability of RL to handle spatial and temporal complexities inherent in ride-hailing systems. Recently, an RL-based approach is proposed to optimize the order dispatching strategy to maximize the system's revenue, considering the heterogeneity of passenger cancellation behavior and driver work pattern (Wang et al., 2024). Other less related studies use RL for dynamic pricing, charging and relocation (Huang et al., 2022; Ke et al., 2020; Liang, 2024; Wang et al., 2020).

Despite the advances brought by RL-based approaches, most studies assume fixed batching intervals and matching radii. However, both factors are crucial in determining the size of the matching pool, which significantly impacts system efficiency and overall performance (Yang et al., 2020). In other words, matching strategies are shaped by these two spatiotemporal variables. Several existing studies have focused on optimizing matching time intervals for ride-hailing systems. For example, Qin et al. (2021) present a family of policy-gradient based RL algorithms for the delayed matching problem and propose an actor-critic method to optimize matching time intervals, balancing wait time penalties with improved matching efficiency. Ke et al. (2020) apply a multi-agent RL approach to dynamically determine the delayed entry time for each ride request, showing empirical improvements in system performance by balancing pick-up time, matching time, and successful matching rate.

In contrast, the role of the matching radius has received comparatively less attention, despite its critical influence on the number of potential matches and the balance between ride requests and available drivers. Most studies treat the matching radius as a fixed parameter rather than a decision variable. Xu et al. (2020) investigate the impact of the matching radius on system efficiency and theoretically demonstrate that adjusting it can mitigate inefficiencies. However, their approach relies on static assumptions and does not propose a learning-based mechanism to adaptively adjust the radius in response to dynamic demand and supply conditions.

To bridge this gap, this work introduces an RL algorithm to automatically learn and adaptively adjust the matching radius based on ride-hailing system dynamics, in order to enhance resource utilization, and improve adaptability to fluctuating supply–demand scenarios.

3. Problem description

Batched matching in ride-hailing systems is an optimization problem that assigns available drivers to ride requests within discrete time intervals, referred to as batching windows. Instead of processing requests individually in real time, the system accumulates them over a fixed batching window and executes a matching algorithm at the end of the time interval. Unmatched requests from one batching window are

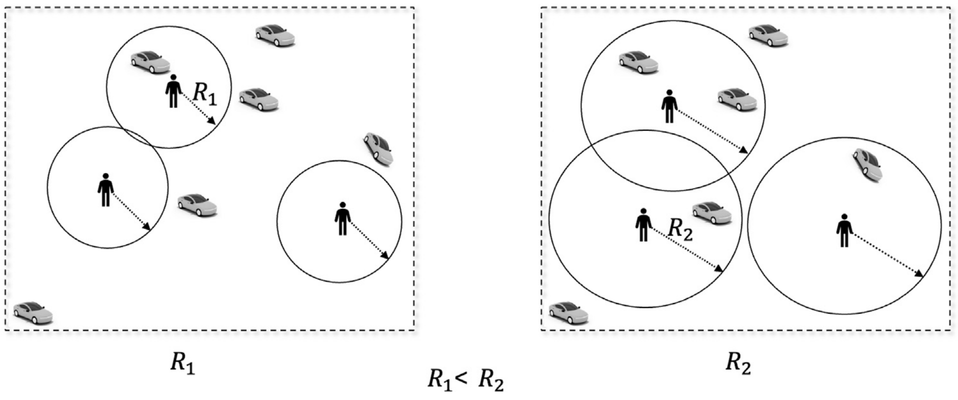


Fig. 1. An illustration of different matching radii in a region, denoted as $\mathcal{R} = R_1$ and $\mathcal{R} = R_2$, respectively, where $R_1 < R_2$. The left panel shows that with a smaller matching radius, fewer drivers receive ride requests, potentially leading to unfulfilled demand. In contrast, the right panel demonstrates that increasing the matching radius expands the search area, increasing the likelihood of successful matches.

carried over to the next, where they are reconsidered alongside newly arriving requests. The primary objective of batched matching is to optimize the performance in terms of service quality, operational efficiency, or economic benefits, while ensuring that feasibility constraints are met.

Given a fixed batching window, the effectiveness of the matching process depends not only on the availability of drivers and riders but also on spatial constraints, particularly the **matching radius**. The matching radius defines the maximum distance allowed for assigning a driver to a rider. Given a region and the corresponding matching radius $\mathcal{R}$, as illustrated in Fig. 1, a rider can be assigned to a driver if the driver is within a circular range centered at the rider, with the radius $\mathcal{R}$. Expanding the matching radius increases the likelihood of successful matches because more drivers and riders would be involved but also leads to longer pick-up distances, potentially affecting service quality and driver efficiency.

This trade-off between match probability and pick-up distance highlights the need to optimize the matching radius to balance these competing factors. The following section presents an optimization framework for dynamically learning the matching radius to improve overall system performance.

4. Methodology

In this section, we present the methodology for optimizing the matching radius in a ride-hailing system. We first formulate the problem as a Markov Decision Process (MDP) and then propose a deep reinforcement learning (DRL) approach with a dual-replay-buffer mechanism to improve learning efficiency and convergence.

4.1. MDP for matching radius optimization

The problem can be modeled as a Markov decision process (MDP) defined by a 5-tuple (S, A, Pr, R, γ), where S, A, Pr, R, γ are the set of states, the set of actions, the transition probability, the reward function and a discount factor, respectively. In the MDP, we divide the target ride-hailing area into cells (i.e., non-overlapping regions), and the decision-making is made by an agent for determining the matching radius in each cell.

4.1.1. State

The state represents the observations from the environment that can reflect the current system status to the agent (Sutton, 2018). In the context of a ride-hailing system, the state can be represented by features such as the number of ride requests and the distribution of idle vehicles across different regions. Specifically, the state comprises observations from all cells at time step t, which is denoted as $s_t \in S$. Given the set of cells in the target area $\mathcal{A}$, the observation of cell i at time step t is denoted as o_{it}. Therefore, the state can be derived as:

$$s_t = (o_{1t}, o_{2t}, \dots, o_{it}, \dots, o_{|\mathcal{A}|t}),\ \ i \in \mathcal{A}. \tag{1}$$

In fact, the MDP environment encompasses all (observable and unobservable) entities other than the agent itself, with which the agent interacts to sequentially obtain states and take actions (Sutton, 2018). In the case of the ride-hailing system, the environment reflects the dynamics of the ride-hailing market, which are implemented in the ride-hailing simulator and detailed in Section 5.2.

4.1.2. Action

Given a state s_t, the agent takes an action $a_t \in A$ that represents the matching radii for all cells. With the collection of matching radii, the matching between riders and drivers can be performed. A matching pool in a cell refers to a set of riders with available drivers within the matching radius. The matching radius within a cell determines the number of drivers in the matching pool. The larger the matching radius, the bigger the pool. The matching radius can vary across the cells.

Let a_{it} denote the matching radius in cell i at time step t. All the riders in the same cell share the same matching radius, while riders in different cells can have different matching radii. Accordingly, the action at time step t can be derived as:

$$a_t = (a_{1t}, a_{2t}, \dots, a_{it}, \dots, a_{|\mathcal{A}|t}),\ \ i \in \mathcal{A}. \tag{2}$$

4.1.3. Reward

After the agent takes an action at each time step, the environment returns a reward to the agent. The goal of MDP is to maximize the total reward the agent receives after its sequential interactions with the environment. In the MDP for matching radius optimization, we define the reward $R_t(s_t, a_t)$ as a performance score for the overall ride-hailing system. This performance score at time step t is a weighted sum of three performance indicators, i.e., matching rate r_t^{mr} with weight w_1, average pick-up distance r_t^{pd} with weight w_2, and driver utilization rate r_t^{du} with weight w_3, such that:

$$R_t(s_t, a_t) = w_1 r_t^{mr} + w_2 r_t^{pd} + w_3 r_t^{du}, \tag{3}$$

where w_1, w_2, w_3 are optimization weights for each of the reward components. In this work, we conducted a series of experiments with different weights for sensitivity analysis (in Section 5.4.5), which was used to identify the suitable weights that balance different performance indicators.

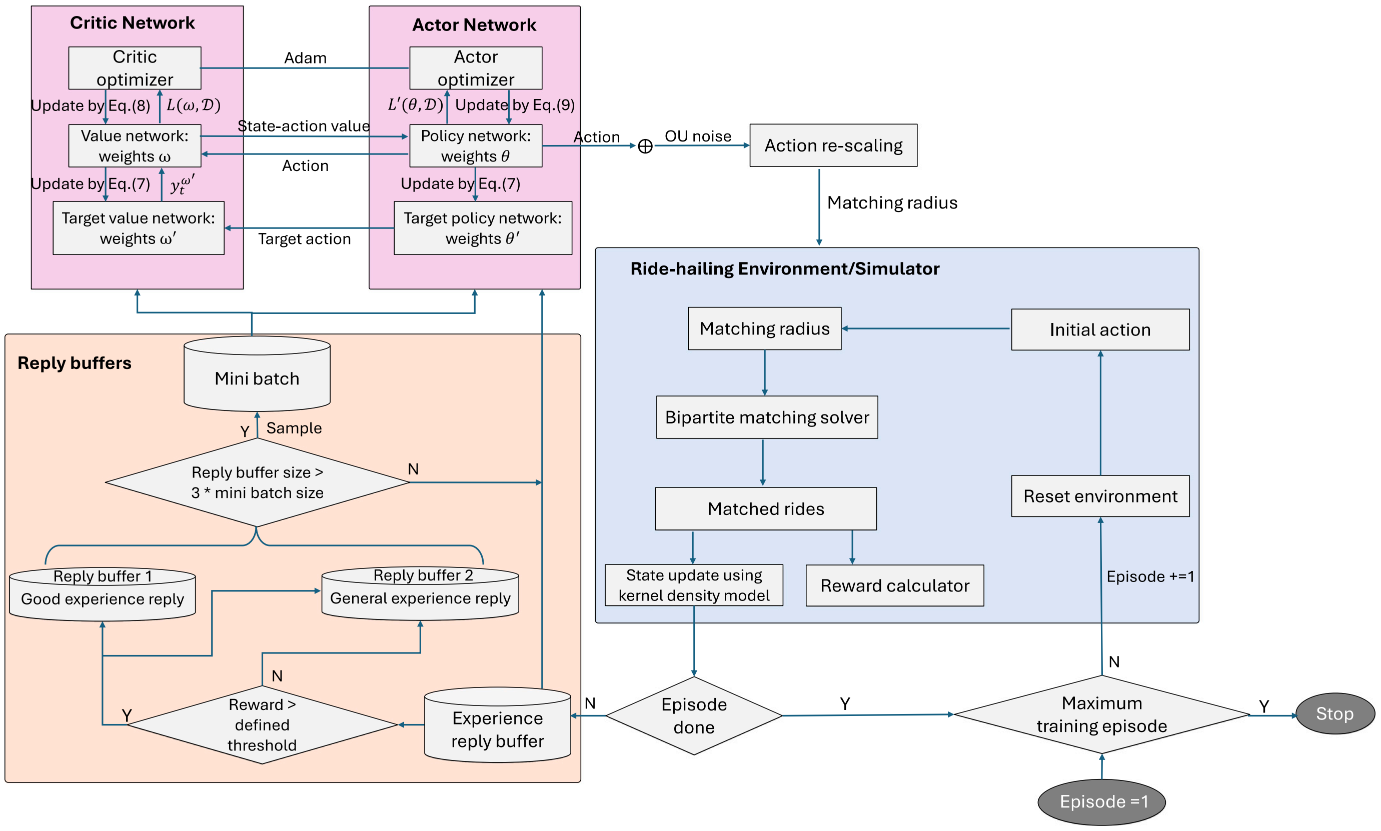


Fig. 2. The working flow of the D-DDPG algorithm in ride-hailing systems.

4.1.4. Policy

The policy is used to determine the dynamic matching radius at each time step. Based on the defined ride-hailing MDP, the objective of DRL is to maximize the expected total reward $Q_\pi(s,a)$ at each time step. Let G_t denote the action value of action $a_t = a$ under the state $s_t = s$. Formally, the optimization objective in DRL can be expressed as:

$$\max Q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a], \tag{4}$$

where $G_t = \sum_{i=t}^{T} \gamma^{i-t} R_i$ is the accumulated reward from the time step t, and γ is the discount factor. In this work, we propose a novel DRL algorithm to optimize the policy π for maximizing the expected total reward.

4.2. Dual-replay-buffer deep deterministic policy gradient

Given the MDP formulation, we parameterize the policy using a neural network, and then propose a Dual-replay-buffer Deep Deterministic Policy Gradient (D-DDPG) algorithm to update the policy network.

4.2.1. Traditional DDPG

The DDPG algorithm is a model-free off-policy DRL algorithm for handling continuous actions (Lillicrap, 2015). It adopts two neural networks, i.e., the actor and critic network, to learn the policy and estimate the Q-function, respectively. The DDPG also adopts target critic and actor networks to stabilize training. It applies a soft update approach for the target networks, gradually adjusting their parameters towards the source networks. In general, the DDPG algorithm alternates between Q-learning and policy learning. The objective of Q-learning is to minimize the mean-squared Bellman error:

$$L(\omega,\mathcal{D}) = \mathbb{E}_{(s_t,a_t,R_t,s_{t+1})\sim\mathcal{D}}\left[\left(Q_\omega(s_t,a_t) - \left(R_t + \gamma(1-d)Q_{\omega'}(s_{t+1},\mu_{\theta'}(s_{t+1}))\right)\right)^2\right], \tag{5}$$

where the critic network Q_ω is updated using stochastic gradient descent. $\mu_{\theta'}$ and $Q_{\omega'}$ are target actor and critic networks, respectively. $\mathcal{D}$ denotes the collected transitions (s_t, a_t, R_t, s_{t+1}) in the replay buffer. d denotes if the state s_{t+1} in a transition is a terminal state or not. When $d = 1$, the state s_{t+1} is terminal, and thus the agent gets no additional reward after this state.

The objective of policy learning is to maximize the expected total reward:

$$L'(\theta,\mathcal{D}) = \mathbb{E}_{s\sim\mathcal{D}}\left[Q_\omega(s,\mu_\theta(s))\right], \tag{6}$$

where the actor network μ_θ is updated using stochastic gradient descent. After updating the actor and critic networks, the target networks are updated:

$$\omega' \leftarrow \tau\omega' + (1-\tau)\omega, \;\; \theta' \leftarrow \tau\theta' + (1-\tau)\theta, \tag{7}$$

where $\tau \in (0,1)$ is the soft update coefficient that controls the rate of change. This update helps stabilize training by slowly tracking the learned networks.

The DDPG offers several advantages, such as the high sampling efficiency and the suitability for continuous action spaces. To improve the efficiency of policy training, we extend the standard DDPG framework by incorporating a dual replay buffer mechanism, referred to as D-DDPG. In addition to the standard replay buffer, which stores all observed transitions, we introduce a second buffer that selectively stores transitions associated with high cumulative rewards. The rationale behind this design is to differentiate between standard experiences and those that reflect elite actions, which in our case are highly effective matching radius decisions. By prioritizing these high-reward transitions during training, the algorithm is encouraged to learn more effectively from successful strategies. This helps guide the policy towards high-performing behaviors, accelerates convergence, and enhances sample efficiency, particularly in environments with sparse or noisy rewards. The designed D-DDPG is detailed in the next subsection.

4.2.2. Description of the D-DDPG

Following the DDPG, we employ two sets of neural networks in the D-DDPG algorithm, i.e., the source and target actor networks μ_θ and $\mu_{\theta'}$, along with the source and target critic networks Q_ω and $Q_{\omega'}$. The actor network determines actions based on states, while the critic network assesses the actions by estimating their expected rewards. The target networks are used to stabilize the training for both actor and critic networks. In addition, we adopt two replay buffers in the D-DDPG algorithm, i.e., one for storing all transitions as in traditional DDPG, and the other for storing transitions with high rewards, which further improves learning efficiency and accelerates convergence. The workflow of D-DDPG algorithm is illustrated in Fig. 2.

Transitions in D-DDPG. The blue part in Fig. 2 includes the components of the ride-hailing environment. To start an MDP episode, the environment is reset to initialize the state, including the number of idle drivers and ride requests in each cell. In each step within an episode, the current state is fed to the actor network for obtaining an action. Afterward, the action is intervened by adding noise generated by a noise function. The intervened action is further rescaled using the min–max normalization to match the corresponding radius range, and the rescaled radius is passed to the environment for interaction. In the environment, the input matching radius for each cell is used to form a matching pool, and then the ride-hailing matching problem is solved as a bipartite matching problem with weighted edges of pick-up distance (Karp et al., 1990). The immediate reward is then calculated by assessing the ride-hailing system with the three performance indicators, according to the reward definition in Section 4.1.3. After reward calculation, the environment also updates the behavior of unmatched riders and drivers to generate new ride requests, thereby completing a state transition. Given that the new state does not meet the termination criterion, it is provided as feedback to the agent along with the reward. If the termination criterion is met, it indicates that an episode ends, where the environment is reset to start the next episode. We apply a number of episodes to collect transitions in the replay buffer, which are used for training.

Dual replay buffers. After tth step in the D-DDPG algorithm, the data in the transition, i.e., $(s_t,\ a_t,\ R_t,\ s_{t+1})$, is stored in the replay buffer, as shown in the bottom-left corner of Fig. 2. If the reward in the transition is higher than a given threshold, we also store the corresponding data in a separate replay buffer, referred to as the elite-action buffer. We continue collecting transitions until the number of transitions in the replay buffer exceeds a predefined threshold, which is set to three times the mini-batch size for training. This indicates that the replay buffer has accumulated sufficient transitions to start the training. Hence, the neural networks are updated to learn the matching radius from the stored transitions.

Neural network update. We sample a mini-batch of transitions from both replay buffers with a uniform distribution as shown in the bottom-left corner of Fig. 2. The sampled transitions are passed to the actor and critic networks for updating their weights. In the upper half of Fig. 2, the source actor network is used to output actions based on the states in transitions. Then, we apply Adam optimizer to update weights in the neural networks. The weights of critic network are updated by the gradient:

$$\nabla_\omega L(\omega, \mathcal{D}) = \nabla_\omega \frac{1}{|\mathcal{B}|} \sum_{(s_t,a_t,R_t,s_{t+1})\sim\mathcal{B}} (Q_\omega(s_t,a_t) - y_t^{\omega'})^2, \tag{8}$$

where $y_t^{\omega'}$ is an expected state–action value and defined as $y_t^{\omega'} = R_t + \gamma(1-d)Q_{\omega'}(s_{t+1}, \mu_{\theta'}(s_{t+1}))$. The weights of actor network are updated to increase the probability of selecting actions with higher values, using the gradient:

$$\nabla_\theta L'(\theta, \mathcal{D}) = \frac{1}{|\mathcal{B}|} \sum_{s_t\in\mathcal{B}} \nabla_\theta \mu_\theta(s_t) \nabla_a Q_\omega(s_t, a)|_{a=\mu_\theta(s_t)}. \tag{9}$$

After updating source neural networks, the parameters of the target neural networks are updated according to Eq. (7). We continue the aforementioned transition storage and network update until the maximum number of episodes is reached. The finally trained source actor network represents the optimized policy for determining the matching radius. The pseudocode of the D-DDPG algorithm is shown in Algorithm 1.

Algorithm 1 D-DDPG

```
Require: Maximum episodes $E$, Maximum time steps $T$, Replay buffer $R$, Elite replay buffer $R'$
 1: Initialize actor network $\mu_\theta$, critic network $Q_\omega$, and target networks $\mu_{\theta'}$, $Q_{\omega'}$
 2: Initialize replay buffers $R$ and $R'$
 3: Initialize ride-hailing environment (simulator)
 4: for episode = 1 to $E$ do
 5:     Reset environment and initialize state $s_0$
 6:     Reset noise process $\mathcal{N}$ and set noise factor $p = \frac{\text{episode}}{E}$
 7:     for $t = 0$ to $T-1$ do
 8:         Select action: $a_t = p \cdot \mu_\theta(s_t) + (1-p) \cdot \mathcal{N}$
 9:         Rescale action $a_t$ to $a_t'$ and apply it in the environment
10:         Execute ride-hailing matching based on $a_t'$ and update the environment
11:         Observe reward $R_t$ and transition to next state $s_{t+1}$
12:         Store transition $(s_t, a_t, R_t, s_{t+1})$ in replay buffer $R$
13:         if $R_t >$ elite-action threshold then
14:             Store transition $(s_t, a_t, R_t, s_{t+1})$ in elite replay buffer $R'$
15:         end if
16:         if buffer size > warm-up threshold then
17:             Sample a mini-batch from both replay buffers $R$ and $R'$
18:             Update critic network using Eq. (8)
19:             Update actor network using Eq. (9)
20:             Update target networks using Eq. (7)
21:         end if
22:         if termination condition met then
23:             break
24:         end if
25:     end for
26: end for
```

4.2.3. Policy network structure

Fig. 3 illustrates the workflow of the policy (actor) network used in our D-DDPG algorithm, which is designed to optimize the cell-level matching radius for ride-hailing systems based on the observed supply–demand state.

To effectively address the structure of the ride-hailing matching problem, we design a neural network architecture that follows a split–merge–split structure. Taking the actor network as an example, we split the input layer and the first hidden layer into multiple parts of neurons, with each part processing the observation from a single cell. Specifically, each cell i receives its local features, including the number of available drivers $N_{i,t}^d$ and riders $N_{i,t}^r$, as defined in Eq. (10) and Eq. (11). These are processed through cell-specific embedding modules to extract local latent features. The cell-wise embeddings are then aggregated and passed through a shared fully connected hidden layer. This aggregation allows the network to model mutual relationships across all cells, enabling coordinated system-level decision-making. The resulting comprehensive embedding is then distributed again to the split final hidden layers and output layers, each generating the matching radius $r(i,t)$ for the corresponding cell. The input and hidden layers use the Exponential Linear Unit (ELU) activation function to improve training stability and model non-linear spatial dependencies. The output layer adopts the Tanh activation function to normalize the output actions within the range $[-1, 1]$, which are then scaled to the allowable radius range.

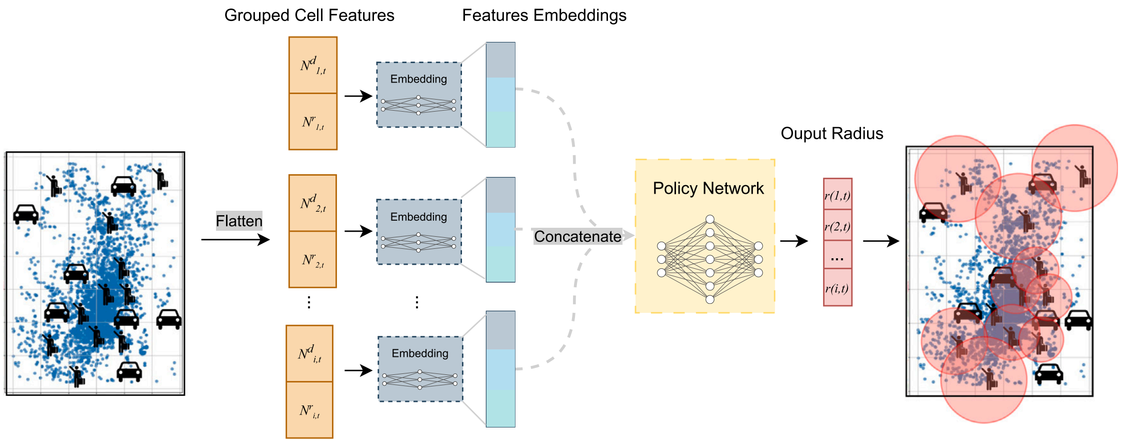


Fig. 3. The workflow of policy network. $N^d_{i,t}$ represents the number of available drivers at time t; $N^r_{i,t}$ represents the number of riders at time t; $r_{(i,t)}$ represents a matching radius in cell i at time t.

Although the figure simplifies the architecture to a single shared hidden layer for clarity, the implemented model uses four fully connected hidden layers with 128 neurons each. This hidden dimension was selected after comparing multiple configurations (64, 128, 256), with 128 providing the best performance across evaluation metrics. The same network structure and hyperparameters are used consistently across all experimental scenarios.

From the perspective of ride-hailing matching, the split parts of neurons in the neural network can effectively capture the unique supply–demand relationship and characteristics of each cell. The fully connected hidden layer can capture relationships between cells by integrating their individual embeddings. In this way, the proposed neural network can not only process the local observation, but also obtain the interdependence between cells, thereby enhancing the performance of the ride-hailing matching.

4.2.4. *Ornstein–Uhlenbeck noise*

In D-DDPG algorithm, we add random noise to the output of the actor network. This noise is used to facilitate the exploration of the action space for better policy learning. Common stochastic processes for generating random noise include Gaussian noise and Ornstein–Uhlenbeck (OU) process. Compared to Gaussian noise, the OU process describes a mean-reverting stochastic process, which helps the exploration of a wider range of actions while still maintaining stability.

The OU process is defined by $dX_t = \theta(\mu - X_t)dt + \sigma dW_t$, where X_t is the value of the process at time t, and θ is the rate of mean reversion, indicating how quickly the process returns to the mean value. μ is the long-term mean, around which the process fluctuates. σdW_t is the random part, where σ is the diffusion coefficient representing the scale of the randomness. dW_t denotes the increment of a standard Brownian motion (Wiener process). According to the definition, the OU random process is a time-correlated random process, which means the random state has correlation with the previous state. Since the action in D-DDPG is the matching radius at each time step, which has continuous effects on the temporal elements, such as supply–demand relationship across time steps, applying OU noise can reflect the temporal effect and thus benefit the policy learning.

5. Experiments

In this section, we provide experimental results on a real-world dataset. We first describe the implementation details, the dataset and the simulator, and then analyze and evaluate the proposed method from different perspectives.

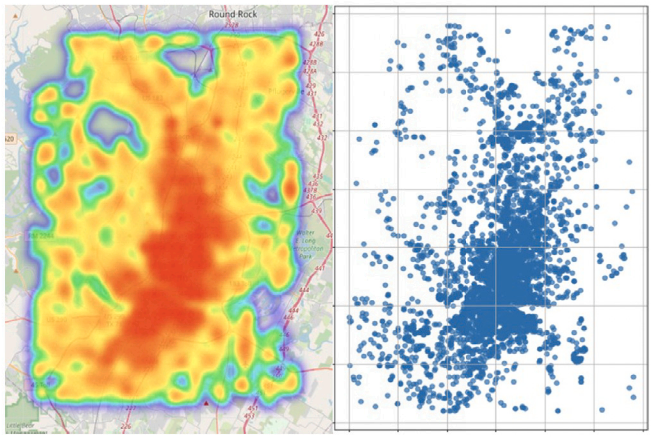


Fig. 4. Spatial distribution of ride requests in Austin.

5.1. *Dataset*

The dataset is developed based on the RideAustin dataset,[1] a publicly available real-world ride-hailing dataset. It contains detailed trip-level information, including the locations of idle drivers, the origins and destinations of travel requests, start and end times of trips, service vehicle types, and so on. Additionally, we integrate geographic data from OpenStreetMap (OSM)[2] to represent Austin's road network, enabling the creation of a realistic simulation environment that accurately depicts the city's layout. The left side of Fig. 4 presents a heat map of real-world demand locations, derived from the RideAustin dataset. The right side of Fig. 4 shows 10,000 demand points, which are randomly sampled from the kernel density estimation. The heat map effectively highlights zones of high and low demand, visually representing areas where ride requests are most and least frequent. The randomly sampled demand points closely replicate the spatial distribution of the real-world demand locations, validating the accuracy of the fitted distribution. In our experiments, we partition the area in Austin into 3 × 3 cells, and employ D-DDPG to learn the policy for determining matching radii in each cell.

[1] https://www.kaggle.com/datasets/mexwell/rideaustin-data
[2] https://www.openstreetmap.org/#map=7/52.154/5.295

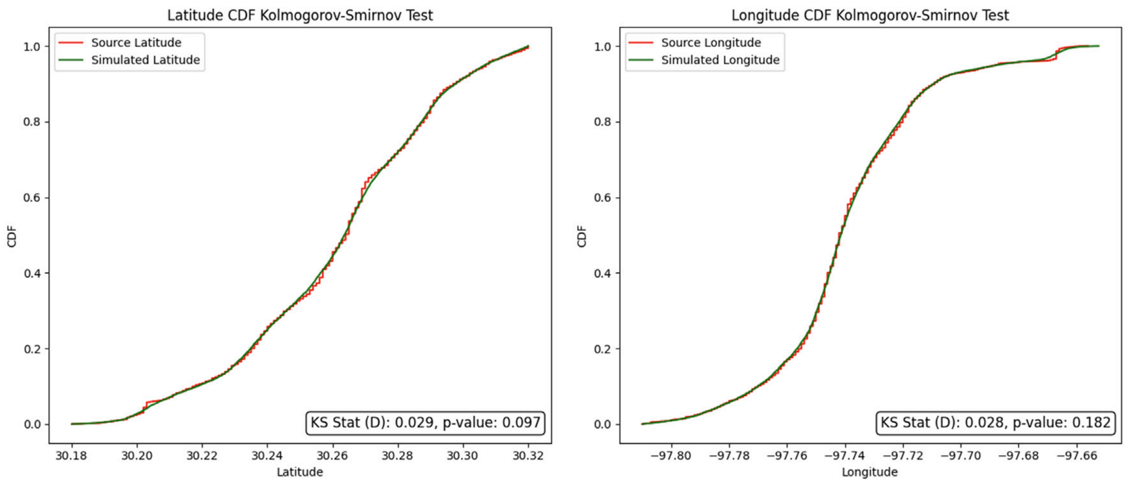


Fig. 5. KS test for rider locations generated by the simulator.

5.2. *Ride-hailing simulator*

We develop a simulator for modeling the realistic ride-hailing market and providing an interactive environment for the DRL agent to generate data for its learning. The simulator is built upon real-world operational characteristics and follows several assumptions to better simulate the real ride-hailing market, as follows: (1) The ride-hailing market is divided into equally sized cells (i.e., regions), each with fixed geographic boundaries, and the matching can be made across cell boundaries; (2) All riders within the same cell share the same matching radius, while the matching radius may vary between different cells; (3) Riders do not cancel orders after being successfully matched, and drivers cannot refuse orders after being assigned to riders.

To ensure that the simulator realistically represents the real-world ride-hailing market, we calibrate both spatial and temporal demand distributions using the RideAustin dataset. The spatial distributions of rider and driver locations are modeled using non-parametric Kernel Density Estimation (KDE), allowing for flexible fitting of multimodal and heterogeneous patterns. To validate the realism of the simulated data, we perform the Kolmogorov–Smirnov (KS) test to statistically compare the distributions of simulated and real-world data.

The null hypothesis in the KS test is that there is no significant difference between the distributions of the simulator-generated rider or driver locations and the actual RideAustin data. If the p-value exceeds 0.05, the null hypothesis cannot be rejected, indicating that the simulator output is consistent with the real-world distribution. As shown in Fig. 5, the KS test for the latitude and longitude of rider locations yields statistics of 0.029 (latitude) and 0.028 (longitude), with corresponding p-values of 0.097 and 0.182, respectively. For driver locations, the test results (shown in Fig. 6) exhibit a similar pattern, with p-values exceeding the 0.05 threshold. These outcomes confirm that the spatial distributions generated by the simulator do not significantly differ from the empirical RideAustin data, thus validating the simulator's spatial fidelity.

Regarding the temporal distribution of travel demand, we also apply the KS test to compare the hourly demand distributions generated by the simulator with those observed in the RideAustin data. As presented in Fig. 7, the test shows no significant difference, further validating the simulator's effectiveness in capturing realistic demand dynamics over time.

Together, these validation steps confirm that the simulator provides a statistically sound approximation of the ride-hailing environment in Austin, Texas. The locations of ride requests and drivers are generated based on the fitted KDE models, ensuring realistic supply and demand inputs. Each cell covers a square area of 5000 m by 5000 m. In the D-DDPG algorithm, each episode simulates one hour of operations with a fixed step size of 15 s, resulting in a total of 240 steps per episode.

Table 1
Parameters in three supply–demand scenarios in the simulator.

Parameters	Balanced	High demand	High supply
Initial Driver Number	100	50	150
Initial Rider Number	100	100	100
Match Time Window (sec)	15	15	15

After the matching radius derived from the actor network, each cell serves as a matching pool, assigning riders to drivers within the region using the matching radius. The assignment in the matching pool is addressed as a bipartite matching problem (Karp et al., 1990). When the matching process is done, successfully matched drivers and demands are removed from the map. Unmatched demands will be checked if they exceed the threshold Δt, which represents the tolerance time defined as the maximum number of matching time steps. If they do, they will also be removed but marked as unmatched. Otherwise, they remain in the matching system and will be involved in the matching process at the next time step $t+1$. Unmatched idle drivers have two possible behaviors of either staying in the same cell and randomly moving within a distance of 300 meters (i.e., idling status), or traveling to the nearest adjacent cell only if they are already idle for more than 5 min (i.e., relocating status). New riders and idle drivers entering the matching process at time step $t+1$ are determined by the rates $\lambda_r(i,t)$ and $\lambda_d(i,t)$, respectively. Accordingly, the number of drivers and riders in a cell evolves as below:

$$N^d_{i,t+1} \leftarrow N^d_{i,t} - n^{d,out}_{i,t} + n^{d,in}_{i,t} + \lambda_d(i,t)\Delta t, \tag{10}$$

$$N^r_{i,t+1} \leftarrow N^r_{i,t} - n^{r,out}_{i,t} + \lambda_r(i,t)\Delta t, \tag{11}$$

where $N^d_{i,t}$ and $N^r_{i,t}$ represent the number of drivers and riders at time step t in cell i; $n^{d,out}_{i,t}$ and $n^{r,out}_{i,t}$ represent the number of drivers and riders who have left cell i; $n^{d,in}_{i,t}$ denote the number of drivers arriving in cell i.

The simulation begins with an initial state, which is characterized by idle drivers and ride requests distributed across cells. We consider three supply–demand scenarios in the simulator, i.e., balanced supply–demand scenario, high-demand scenario, and high-supply scenario. These scenarios reflect different supply–demand relationships in practice, which are depicted in Table 1.

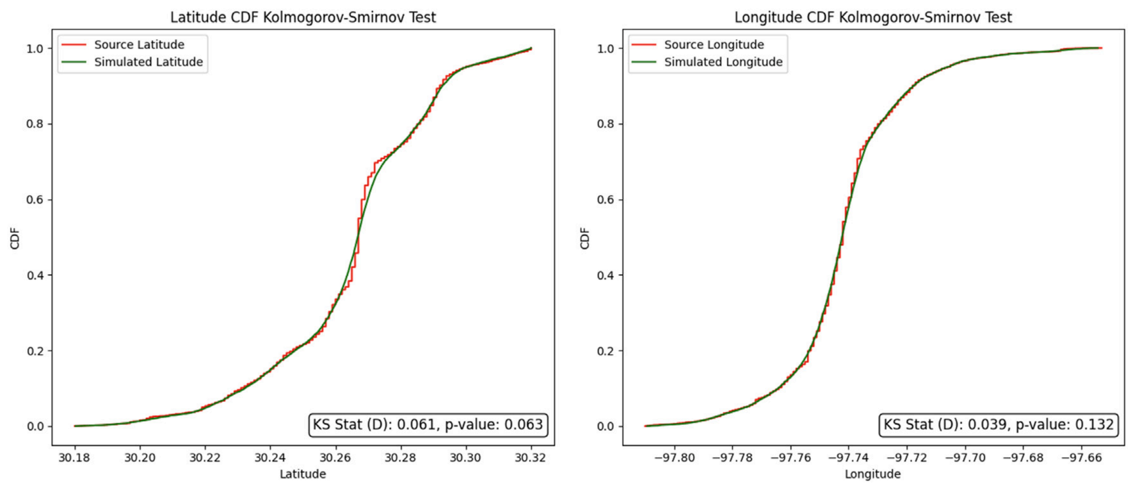


Fig. 6. KS test for rider locations generated by the simulator.

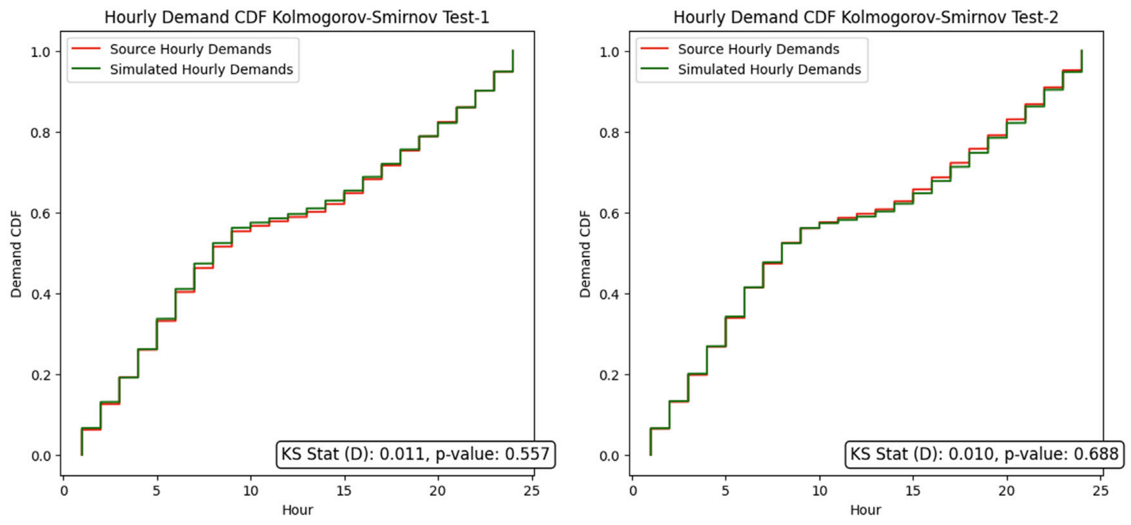


Fig. 7. KS test for rider locations generated by the simulator.

5.3. Training setup

We tune hyperparameters based on training loss and convergence. Regarding the optimization weights in the reward function, we conducted a sensitivity analysis (detailed in Section 5.4.5) with different weights to find the balance between each component. The weights $w_1 = 0.4$, $w_2 = 0.4$, $w_3 = 0.2$ with the best overall performance are used in the training process. The policy is trained in the balanced scenario (as shown in Table 1), where the supply number is approximately equal to the demand number. We train the policy on an AMD Ryzen 9 5900HX CPU (3.3 GHz) with 32 GB RAM and an NVIDIA GeForce RTX 3080 GPU with 16 GB of memory. The average computation time per episode is 30.23 s. The total training time is 6 h and 43 min.

5.4. Experimental results

5.4.1. Comparison in the balanced scenario

Following most previous research on ride-hailing, we apply a fixed matching radius as a baseline to evaluate the performance of the learned policy for determining the matching radius (Chen et al., 2023). We design four baselines to showcase the advantage of the trained policy, i.e., FR 500 m, FR 1000 m, FR 1500 m and FR 2000 m,

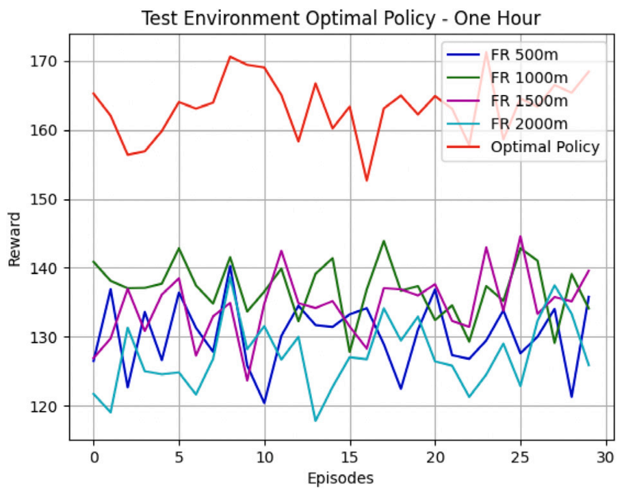


Fig. 8. Comparison with the baselines in the balanced scenario.

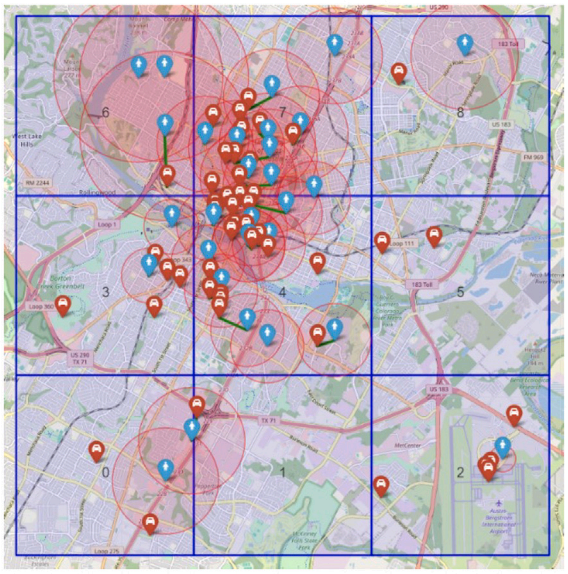


Fig. 9. A matching result with the learned matching radii.

which represent the fixed matching radii of 500, 1000, 1500 and 2000 m, respectively. We evaluate the learned policy and the baseline approaches in three different scenarios (as shown in Table 1), each of which involves 30 random episodes. The performance score, i.e., the reward, is recorded for each method.

The balanced scenario reflects a ride-hailing operational condition where the demand (i.e., the number of ride requests) are generally equal to the supply (i.e., the number of drivers), with small fluctuations in the supply–demand relationship. As illustrated in Fig. 8, the learned policy effectively optimizes the matching radius, which attains the highest performance score compared to the baselines. This indicates that the proposed D-DDPG can learn an effective policy for determining favorable matching radii in different regions, leading to improved ride-hailing performance in Austin under the balanced scenario.

Given the learned matching radii in cells, Fig. 9 demonstrates a matching result between riders and drivers, which shows riders and drivers are reasonably matched with a large matching rate and shorter pick-up distance. It indicates that the policy learned by D-DDPG offers good matching radii, which are adaptive to varying supply–demand relationships in different cells.

5.4.2. *Comparison in the unbalanced scenarios*

We evaluate the learned policy under extreme operational conditions to demonstrate its robustness and adaptability. Specifically, in a high-demand scenario, ride-hailing demand far exceeds available drivers, whereas in a high-supply scenario, drivers outnumber ride requests significantly.

Comparison in the high-demand scenario. The left subfigure in Fig. 10 illustrates the results in the high-demand scenario. As shown, the learned policy consistently outperforms all baselines and demonstrates good stability. Compared to fixed radii, which exhibit significant fluctuations in performance, the learned policy is able to dynamically adjust matching radii and maintain a more stable overall performance. The results indicate that the learned policy handles supply–demand dynamics more effectively, providing a more reliable ride-hailing service.

Comparison in the high-supply scenario. The results in the high-supply scenario are plotted in the right subfigure of Fig. 10. As shown, the learned policy significantly outperforms the baselines by successfully balancing the matching rate, pick-up distance and driver utilization rate. It indicates that the trained policy is able to manage the excess supply of drivers by producing a set of reasonable matching radii. This further verifies the adaptability and effectiveness of D-DDPG in delivering a favorable policy under the high-supply scenario.

In summary, through the comprehensive test under three scenarios, we found that the dynamic matching radius policy trained by the proposed D-DDPG algorithm outperforms the baselines in all scenarios. This trained policy demonstrates advantageous adaptability by dynamically adjusting the matching radius in response to real-time supply–demand variations.

Notably, the relative performance of fixed-radius strategies varies significantly across scenarios. In the high-demand setting, larger radii (e.g., FR-1500 m and FR-2000 m) tend to perform better by increasing the likelihood of successful matches. However, in high-supply conditions, where drivers are abundant, smaller radii (e.g., FR-1000 m) are more efficient due to reduced pickup distances and better driver utilization. In the balanced scenario, intermediate radii such as FR-1000 m or FR-1500 m provide reasonable performance, but still fail to match the consistency and overall reward achieved by the learned policy. The variation reveals a core limitation of fixed matching strategies: no single matching radius is optimal across all scenarios. In contrast, the learned policy continuously adapts the radius in each cell based on local observations, achieving higher performance across all scenarios.

Table 2
Comparison in terms of different metrics.

Policy	R_m	D_p (R_P)	R_{ult}	R
FR-500	0.21	**458.72 (0.85)**	**0.67**	0.558
FR-1000	0.61	896.67 (0.70)	0.52	0.601
FR-1500	0.75	1326.44 (0.56)	0.45	0.614
FR-2000	**0.79**	1678.38 (0.44)	0.27	0.546
Dynamic radius	0.71	993.34 (0.69)	0.63	**0.670**

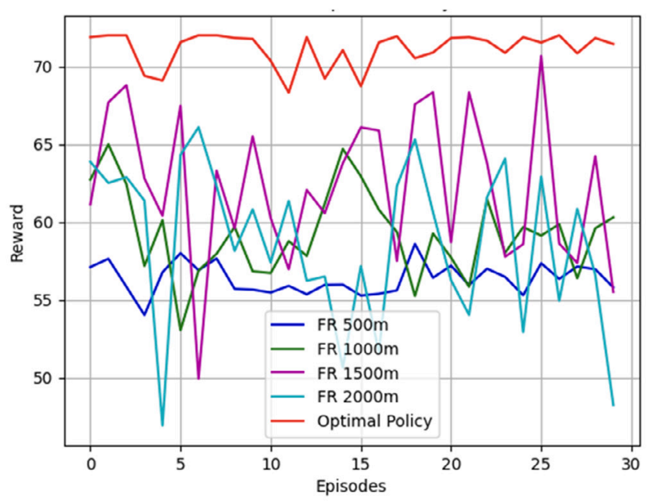
5.4.3. *Comparison in terms of different metrics*

Table 2 shows an average performance comparison for matching rate R_m, average pick-up distance D_P and driver utilization rate R_{ult}, respectively. For each performance metric, we average the results over 40 random episodes in the balanced scenario. The results show that the trained policy can achieve a good balance between multiple performance metrics and achieve the highest overall performance score. While the trained policy does not perform best for all performance metrics, it found a good balance among the three performance metrics, thereby favorably optimizing the ride-hailing system from different perspectives.

The reasons why the trained policy cannot perform best for each performance metric can be explained as follows. Expanding the matching radius to improve the matching rate may lead to a longer pick-up distance and a decrease in driver utilization rate. Conversely, reducing the matching radius to shorten the pick-up distance may reduce the matching rate. The trained policy can strike the best balance between the conflicting optimization objectives, rather than focusing on optimizing just one of them. On the other hand, maximizing driver utilization rate may give priority to the nearest ride requests to reduce the number of unmatched drivers within the matching radius and improve driver utilization rate. This will make the system more similar to a nearest-matching system rather than a batched-matching system, resulting in a significant increase in the average pick-up distance. Therefore, the results in Table 2 demonstrate that the learned policy can maintain a good balance between different performance metrics.

5.4.4. *Long-term performance evaluation*

To evaluate the long-term performance of the learned policy, we compare it with baselines over a 24-hour period in Austin, based on the real supply–demand patterns retrieved from real operating data. Fig. 11 demonstrates that the learned policy outperforms the baselines, consistently achieving the best performance. This indicates that the trained



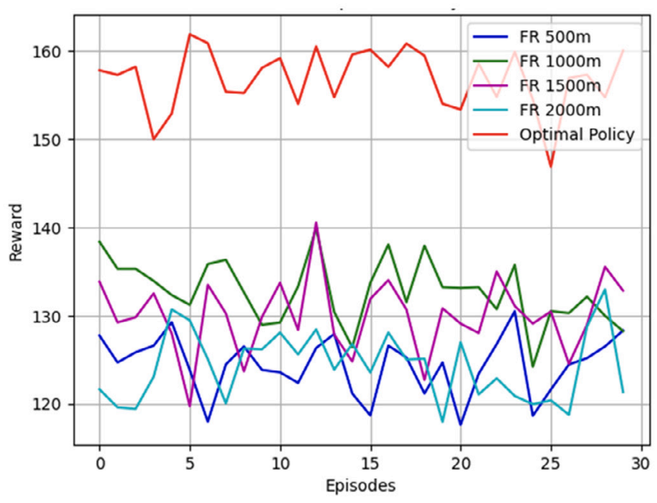


Fig. 10. Comparison with the baselines in the unbalanced scenarios. Left: The high-demand scenario; Right: The high-supply scenario.

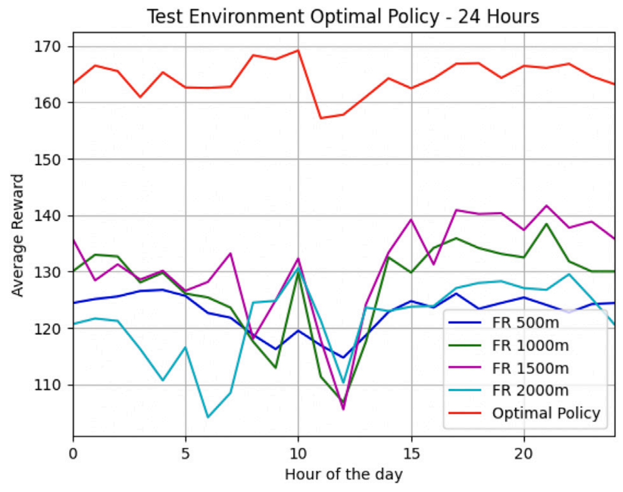


Fig. 11. Comparison with the baselines in terms of long-term performance.

policy successfully finds a balance between different performance metrics. This balance is achieved by adjusting the matching radius based on the real-time supply–demand relationship of the ride-hailing market. The performance of the baselines is prone to fluctuations over time, suggesting that the performance of the fixed matching radius is highly unstable when confronted with different levels of supply–demand imbalances in the ride-hailing system. In contrast, the trained policy demonstrates better stability, which can effectively adjust the matching radius based on the supply–demand relationships.

5.4.5. Sensitivity analysis

To evaluate how different optimization objectives influence the learned policy, we conduct a series of experiments in which the D-DDPG model is independently trained under each of the 13 weight configurations listed in Table 3. Each configuration reflects a distinct prioritization of the three reward components: matching rate (w_1), average pick-up distance (w_2), and driver utilization rate (w_3), with $w_1 + w_2 + w_3 = 1$. This setup effectively simulates a multi-objective optimization process, enabling us to analyze the resulting trade-offs and policy behavior under varying operational goals. The 13 weight combinations are categorized into multi-metric and single-metric classes, reflecting either balanced or goal-specific optimization preferences.

Specifically, the weight combinations in the multi-metric weight class are designed to consider more than one performance metric. For example, the weight combination C_1^{bal} involves the matching rate, the average pick-up distance, and the driver utilization rate. We applied this combination to train and test the policy in all the other experiments. Compared to C_1^{bal}, the weight combination C_2^{bal} focuses more on optimizing the matching rate, and C_4^{bal} pays more attention to optimizing the average pick-up distance. The weight combinations C_8^{bal} to C_{10}^{bal} evaluate the policies trained with two balanced weights. In the single-metric class, each combination fully focuses on one specific performance metric in the ride-hailing system. For example, C_1^{sin} focuses only on the matching rate. Using the single-metric class, the applicability and effectiveness of the trained policy under biased optimization goals are evaluated.

Result analysis. The results of sensitivity analysis are shown in Fig. 12, which presents statistics of the matching radii in the central region of Austin. Specifically, we illustrate the average value range and the 90% confidence interval for the matching radii obtained from policies trained with different weight combinations. All results were obtained in the balanced scenario, with each weight combination tested over 40 episodes. We observe that the matching radius with multi-metric weights fluctuates within the range of 600–1400 m. For C_2^{bal}, there is an increase in the matching radius compared to that of C_1^{bal}. This observation is reasonable, as a larger matching radius increases the matching pool, thereby raising the probability of ride requests being matched with available drivers. In contrast, the weight combination C_4^{bal} has a larger bias towards the average pick-up distance, thus leading to a significant decrease in the matching radius. Regarding C_6^{bal}, the fluctuation becomes higher. By comparing it with C_3^{sin}, which only optimizes the driver utilization rate, we observe that optimizing the driver utilization rate as the single metric is difficult. The results of C_3^{bal}, C_5^{bal} and C_5^{bal} show that slight changes in weights cannot cause large fluctuations in the matching radius of the learned policy. Taking C_3^{bal} as an example, when it is compared with C_2^{bal}, the bias towards optimizing the average pick-up distance is slightly reduced, the average matching radius is slightly increased, and the confidence interval does not change significantly. In summary, the results indicate that the policy learned by our D-DDPG algorithm has good robustness and applicability for different weights across the performance metrics.

Average performance. We further evaluate the average performance of each weight combination. We train policies under each weight combination, and validate them over 40 episodes. We gather results in Table 4, where the last column is the average weighted performance score (i.e., the overall reward R). The results demonstrate the effectiveness of the proposed D-DDPG algorithm. As shown, the learned

Table 3
Weight combinations in sensitivity analysis.

Weights	C_1^{bal}	C_2^{bal}	C_3^{bal}	C_4^{bal}	C_5^{bal}	C_6^{bal}	C_7^{bal}	C_8^{bal}	C_9^{bal}	C_{10}^{bal}	C_1^{sin}	C_2^{sin}	C_3^{sin}
w_1	0.4	0.5	0.5	0.3	0.2	0.2	0.3	0.5	0.5	0.0	1.0	0.0	0.0
w_2	0.4	0.3	0.2	0.5	0.5	0.3	0.2	0.5	0.0	0.5	0.0	1.0	0.0
w_3	0.2	0.2	0.3	0.2	0.3	0.5	0.5	0.0	0.5	0.5	0.0	0.0	1.0

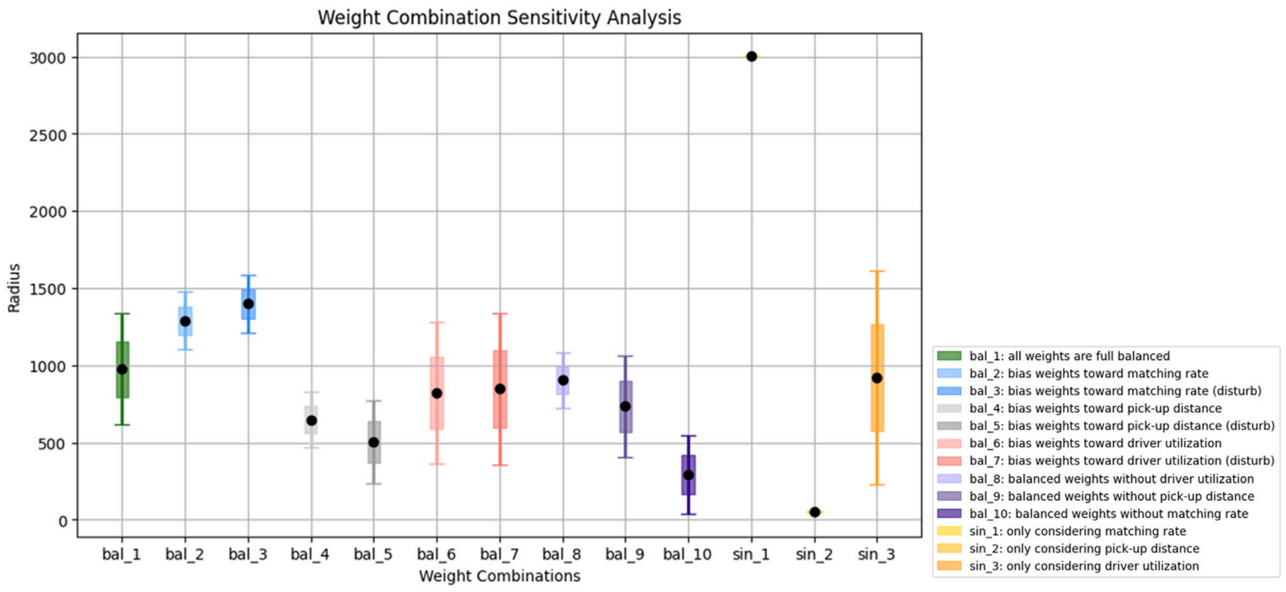


Fig. 12. Average and 90% trust interval of obtained radius.

policy can consistently reach a high reward value, indicating that the D-DDPG algorithm can be well generalized to a variety of weight combinations. Among the combinations, the weight combination (0.4, 0.4, 0.2) concurrently optimizes three performance metrics and achieves the highest overall score. Hence, we applied it to train and test the policy in all the other experiments. We note that although some weight combinations get higher R values, they cannot optimize all metrics, leading to severely biased optimization.

The sensitivity analysis confirms that the policy can adapt to different optimization objectives that may exist in the ride-hailing market in Austin, verifying its applicability and effectiveness in a real-world operating environment.

6. Conclusion and future work

This paper provides a DRL solution to optimize the ride-hailing matching radius. We propose the MDP to formulate the matching radius optimization problem. On top of it, we develop a D-DDPG algorithm to learn the policy for determining the matching radius. We design different experimental scenarios of the ride-hailing market in Austin, and analyze the effectiveness and versatility of the proposed method under various supply–demand relationships. Extensive results show that our method effectively balances different performance indicators in the ride-hailing system. From an application perspective, our approach offers ride-hailing platforms a scalable and adaptive tool for real-time operational decision-making, especially in dynamically changing environments. By integrating the learned policy, platforms can move away from static, rule-based radius settings and instead tailor the matching process to localized demand-supply patterns. This leads to reduced waiting times for passengers, higher driver utilization, and more efficient resource allocation.

The policy network is designed to output a dynamic matching radius for each cell at each decision step, enabling adaptation to localized supply–demand conditions. While the dimensionality of the action space increases with the number of cells, the neural network's capacity for matrix operations allows it to handle high-dimensional outputs theoretically. The primary computational bottleneck currently lies within the simulation framework, specifically the Hungarian Matching Algorithm, which is used to match drivers and riders after the radii are determined. This process can become time-consuming with a large number of agents. Therefore, while our approach is applicable to more granular local areas, the computational cost increases due to the current simulation limitations. Future work will focus on optimizing the matching process within the simulator and exploring parallelization techniques to enhance scalability. Additionally, future research will focus on refining the spatial division of the ride-hailing service area to enable a more fine-grained and adaptive partitioning based on real-world conditions. Incorporating practical considerations, such as driver preferences for accepting ride requests and route choices, into the MDP formulation will also improve the realism and robustness of the proposed approach. In addition, another promising direction for future

Table 4
Sensitivity analysis of performance metrics.

Set	R_m	w_1	D_p (R_P)	w_2	R_{ult}	w_3	R
C_1^{bal}	0.71	0.4	993.34 (0.69)	0.4	0.63	0.2	0.670
C_2^{bal}	0.73	0.5	1131.27 (0.62)	0.3	0.50	0.2	0.651
C_3^{bal}	0.75	0.5	1236.41 (0.59)	0.2	0.48	0.3	0.637
C_4^{bal}	0.42	0.3	638.25 (0.79)	0.5	0.65	0.2	0.651
C_5^{bal}	0.36	0.2	616.32 (0.78)	0.5	0.66	0.3	0.660
C_6^{bal}	0.57	0.2	898.96 (0.70)	0.3	0.68	0.5	0.664
C_7^{bal}	0.59	0.3	911.58 (0.69)	0.2	0.67	0.5	0.649
C_8^{bal}	0.55	0.5	901.19 (0.70)	0.5	0.59	0.0	0.626
C_9^{bal}	0.49	0.5	892.93 (0.70)	0.0	0.65	0.5	0.577
C_{10}^{bal}	0.18	0.0	404.33 (0.86)	0.5	0.79	0.5	0.825
C_1^{sin}	0.80	1.0	1945.31 (0.35)	0.0	0.81	0.0	0.830
C_2^{sin}	0.03	0.0	47.12 (0.98)	1.0	0.87	0.0	0.980
C_3^{sin}	0.73	0.0	1175.90 (0.55)	0.0	0.67	1.0	0.665

research is to explore Multi-Agent Reinforcement Learning (MARL) approaches for matching radius optimization. While our current single-agent framework is effective and avoids communication overhead, MARL offers potential advantages in scalability by decomposing the decision space across agents. However, applying MARL in real-world ride-hailing systems introduces challenges such as coordination complexity and communication latency. Future work will investigate how to mitigate these issues, potentially through decentralized learning schemes or communication-efficient agent coordination.

CRediT authorship contribution statement

Jie Gao: Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization. **Rong Cheng:** Writing – review & editing, Validation, Conceptualization. **Yaoxin Wu:** Writing – review & editing, Writing – original draft, Validation, Conceptualization. **Honghao Zhao:** Writing – original draft, Methodology, Software, Formal analysis, Validation, Investigation. **Weiming Mai:** Writing – review & editing, Methodology, Conceptualization, Supervision. **Oded Cats:** Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Chen, X., Bai, S., Wei, Y., & Jiang, H. (2023). How income satisfaction impacts driver engagement dynamics in ride-hailing services. *Transportation Research Part C: Emerging Technologies*, *157*, Article 104418.

Chen, T., Shen, Z., Feng, S., Yang, L., & Ke, J. (2025). Dynamic matching radius decision model for on-demand ride services: A deep multi-task learning approach. *Transportation Research Part E: Logistics and Transportation Review*, *193*, Article 103822.

Essus, Y., De La Fuente, R., & Venkitasubramanian, A. (2024). Real-time optimization for relocation and dispatching of emergency medical services with balanced workload and outsourced ride-hailing services. *Computers & Industrial Engineering*, *187*, Article 109823.

Feng, G., Kong, G., & Wang, Z. (2021). We are on the way: Analysis of on-demand ride-hailing systems. *Manufacturing & Service Operations Management*, *23*(5), 1237–1256.

Gao, J., Li, X., Wang, C., & Huang, X. (2020). Learning-based open driver guidance and rebalancing for reducing riders' wait time in ride-hailing platforms. In *Proceedings of the 6th IEEE international smart cities conference* (pp. 1–7).

Gao, J., Li, X., Wang, C., & Huang, X. (2021). BM-DDPG: An integrated dispatching framework for ride-hailing systems. *IEEE Transactions on Intelligent Transportation Systems*, *23*(8), 11666–11676.

Guo, Y., Li, W., Xiao, L., Choudhary, A., & Allaoui, H. (2024). Enhancing efficiency and interpretability: A multi-objective dispatching strategy for autonomous service vehicles in ride-hailing. *Computers & Industrial Engineering*, *194*, Article 110385.

Huang, J., Huang, L., Liu, M., Li, H., Tan, Q., Ma, X., Cui, J., & Huang, D. S. (2022). Deep reinforcement learning-based trajectory pricing on ride-hailing platforms. *ACM Transactions on Intelligent Systems and Technology*, *13*(3), 1–19.

Jin, J., Zhou, M., Zhang, W., Li, M., Guo, Z., Qin, Z., Jiao, Y., Tang, X., Wang, C., Wang, J., et al. (2019). Coride: joint order dispatching and fleet management for multi-scale ride-hailing platforms. In *Proceedings of the 28th ACM international conference on information and knowledge management* (pp. 1983–1992).

Karp, R. M., Vazirani, U. V., & Vazirani, V. V. (1990). An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd annual ACM symposium on theory of computing* (pp. 352–358).

Ke, J., Xiao, F., Yang, H., & Ye, J. (2020). Learning to delay in ride-sourcing systems: A multi-agent deep reinforcement learning framework. *IEEE Transactions on Knowledge and Data Engineering*, *34*(5), 2280–2292.

Lee, D. H., Wang, H., Cheu, R. L., & Teo, S. H. (2004). Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record*, *1882*(1), 193–200.

Liang, Y. (2024). Fairness-aware dynamic ride-hailing matching based on reinforcement learning. *Electronics*, *13*(4), 775.

Lillicrap, T. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Liu, Y., Jia, R., Ye, J., & Qu, X. (2022). How machine learning informs ride-hailing services: A survey. *Communications in Transportation Research*, *2*, Article 100075.

Liu, Y., Wu, F., Lyu, C., Li, S., Ye, J., & Qu, X. (2022). Deep dispatching: A deep reinforcement learning approach for vehicle dispatching on online ride-hailing platform. *Transportation Research Part E: Logistics and Transportation Review*, *161*, Article 102694.

Lowalekar, M., Varakantham, P., & Jaillet, P. (2018). Online spatio-temporal matching in stochastic and dynamic domains. *Artificial Intelligence*, *261*, 71–112.

Qin, G., Luo, Q., Yin, Y., Sun, J., & Ye, J. (2021). Optimizing matching time intervals for ride-hailing services using reinforcement learning. *Transportation Research Part C: Emerging Technologies*, *129*, Article 103239.

Qin, Z. T., Zhu, H., & Ye, J. (2022). Reinforcement learning for ridesharing: An extended survey. *Transportation Research Part C: Emerging Technologies*, *144*, Article 103852.

Sutton, R. S. (2018). Reinforcement learning: An introduction. *A Bradford Book*.

Wang, E., Ding, R., Yang, Z., Jin, H., Miao, C., Su, L., Zhang, F., Qiao, C., & Wang, X. (2020). Joint charging and relocation recommendation for e-taxi drivers via multi-agent mean field hierarchical reinforcement learning. *IEEE Transactions on Mobile Computing*, *21*(4), 1274–1290.

Wang, Y., Sun, H., Lv, Y., Chang, X., & Wu, J. (2024). Reinforcement learning-based order-dispatching optimization in the ride-sourcing service. *Computers & Industrial Engineering*, *192*, Article 110221.

Wang, H., & Yang, H. (2019). Ridesourcing systems: A framework and review. *Transportation Research Part B: Methodological*, *129*, 122–155.

Wu, T., Wang, S., Wang, L., & Tang, X. (2022). Contribution of China's online car-hailing services to its 2050 carbon target: Energy consumption assessment based on the GCAM-SE model. *Energy Policy*, *160*, Article 112714.

Xu, Z., Yin, Y., & Ye, J. (2020). On the supply curve of ride-hailing systems. *Transportation Research Part B: Methodological*, *132*, 29–43.

Yang, H., Qin, X., Ke, J., & Ye, J. (2020). Optimizing matching time interval and matching radius in on-demand ride-sourcing markets. *Transportation Research Part B: Methodological*, *131*, 84–105.

Zhan, X., Qian, X., & Ukkusuri, S. V. (2016). A graph-based approach to measuring the efficiency of an urban taxi service system. *IEEE Transactions on Intelligent Transportation Systems*, *17*(9), 2479–2489.

Zhang, L., Hu, T., Min, Y., Wu, G., Zhang, J., Feng, P., Gong, P., & Ye, J. (2017). A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 2151–2159).