An Online Deep Reinforcement Learning-Based Order Recommendation Framework for Rider-Centered Food Delivery System

Xing Wang, Ling Wang, *Member, IEEE*, Chenxin Dong, Hao Ren, and Ke Xing

***Abstract*— As an important part of intelligent transportation systems, On-demand Food Delivery (OFD) becomes a prevalent logistics service in modern society. With the continuously increasing scale of transactions, rider-centered assignment manner is gaining more attraction than traditional platform-centered assignment among food delivery companies. However, problems such as dynamic arrivals of orders, uncertain rider behaviors and various false-negative feedbacks inhibit the platform to make a proper decision in the interaction process with riders. To address such issues, we propose an online Deep Reinforcement Learning-based Order Recommendation (DRLOR) framework to solve the decision-making problem in the scenario of OFD. The problem is modeled as a Markov Decision Process (MDP). The DRLOR framework mainly consists of three networks, i.e., the actor-critic network that learns an optimal order ranking policy at each interaction step, the rider behavior prediction network that predicts the grabbing behavior of riders and the feedback correlation network based on attention mechanism that identifies valid feedback information from false feedbacks and learns a high-dimensional state embedding to represent the states of riders. Extensive offline and online experiments are conducted on Meituan delivery platform and the results demonstrate that the proposed DRLOR framework can significantly shorten the length of interactions between riders and the platform, leading to a better experience of both riders and customers.**

***Index Terms*— On-demand food delivery, deep reinforcement learning, order recommendation, attention mechanism, feedback information.**

I. Introduction

With the rapid development of intelligent transportation systems, online-to-offline (O2O) business has become an indispensable service in modern daily life. People can buy diverse kinds of goods online without walking out of the door. Among various types of O2O services, on-demand food delivery (OFD) plays an important role which offers wide selection of cuisine for consumers and sets them free from tedious household chores at the same time [1], [2]. In China, over 400 million people are creating orders on Meituan (a major OFD platform in China) with different kinds of food provided by more than 6 million merchants [3]. With such large potential demands, OFD service shows broad market prospects and attracts attention of many researchers and investors.

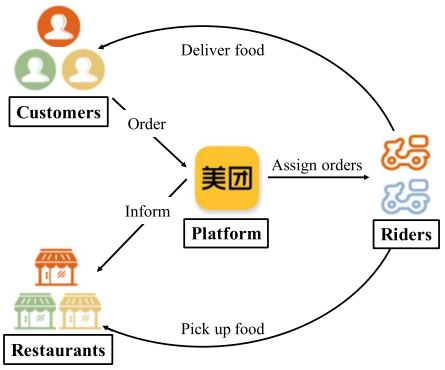


Fig. 1. On-demand food delivery system.

Fig. 1 briefly introduces the OFD system. The OFD platform collects orders from customers and assigns them to appropriate riders. After riders accept to serve the order, they will head to the corresponding restaurant to pick up the food package and then deliver it to the address of the customer. Here, the most important decision part in the whole process is how to assign customer orders to riders. Normally, there are two manners to accomplish this assignment procedure.

1) Platform-centered assignment manner. This manner assumes that riders will absolutely follow the order allocation result by the platform [4], [5], [6], [7]. Therefore, the platform will automatically calculate the fitness of each order-rider pair and arrange only one order for each rider at every decision moment through one-to-one matching algorithms.
2) Rider-centered assignment manner. In this manner, rider does not have to obey the instruction of the platform but can make their own decision of serving orders [8]. They can browse through the order list recommended by the platform and select their favorite orders. In other

Manuscript received 27 September 2022; revised 13 December 2022; accepted 13 January 2023. Date of publication 23 January 2023; date of current version 8 May 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62273193, in part by the Tsinghua University–Meituan Joint Institute for Digital Life, and in part by the Research and Development Project of China Railway Signal & Communication Corp. (CRSC) Research and Design Institute Group Company Ltd. The Associate Editor for this article was T. Alskaif. *(Corresponding author: Ling Wang.)*

Xing Wang and Ling Wang are with the Department of Automation, Tsinghua University, Beijing 100080, China (e-mail: wang-x17@mails.tsinghua.edu.cn; wangling@mail.tsinghua.edu.cn).

Chenxin Dong is with the School of Mechanical and Automotive Engineering, Qingdao Hengxing University of Science and Technology, Qingdao, Shandong 266100, China (e-mail: deyangxuanyi@hotmail.com).

Hao Ren and Ke Xing are with Meituan, Beijing 100015, China (e-mail: renhao05@meituan.com; xingke@meituan.com).

Digital Object Identifier 10.1109/TITS.2023.3237580

words, riders dominate whether or when an order is served instead of the platform.

Platform-centered assignment manner generally requires a delivery team of full-time riders that exclusively offers professional door-to-door delivery for contracted merchants [9]. However, maintaining such a delivery team is costly both for restaurants and platforms. Therefore, it is a trending fashion to recruit ordinary people as part-time riders to enrich the delivery ability of food delivery systems, which makes the rider-centered assignment manner become more popular among OFD companies.

Under such circumstance, the decision-making problem in the rider-centered food delivery system is to conjecture the interests of riders and provide order recommendation lists that riders are interested in so that the orders can be selected and served as soon as possible, which will reduce the whole delivery time and avoid customer dissatisfaction. However, to the best of our knowledge, there are very few researches considering the personal preference of riders and the rider-centered assignment manner in OFD scenario, which is probably due to the following challenges.

1) The dynamism of food delivery system. Customer orders are generated and selected by riders on the OFD platform every second. Therefore, at each decision step, newly generated orders should be included in the recommendation list while selected orders should be excluded so that one order is exactly served by only one rider. Besides, riders are also moving with respect to time. They may be in different status (parking, riding, walking, etc.) at different decision moment. The dynamism increases the difficulty of finding a satisfactory solution for the decision-making problem at each step.
2) The uncertainty of rider behaviors. Since the rider-centered assignment manner allows riders to make their own decision of serving orders, the uncertainty of rider grabbing behaviors cannot be neglected. Given an order recommendation list, a rider may choose to grab one of the orders in current list or choose to refresh the list. Accurately predicting the rider behaviors is extremely difficult since different riders have different thoughts. The lack of massive rider behavior data and effective prediction model inhibits the modeling and disposal of the rider behavior uncertainty.
3) The difficulty of identifying valid feedback information. As mentioned above, riders may give two types of feedback for a given order recommendation list. One is positive feedback which means the rider grabs an order from the list. The other is negative feedback which indicates that rider refreshes the list without grabbing any order. These feedbacks can be used to guide the OFD platform to capture the interests of riders. However, positive feedbacks are effective while negative feedbacks suffer from false-negative phenomenon. That is to say, although the riders do not pick up any order from current recommendation list, there are still orders in the list that riders might be interested in. For example, out of the possibility of order comparison, riders may choose to refresh the list to try to find a probably better order. If there is no such order, they may select the one they are interested in at the previous step. Under this situation, we may need to retain the interesting order for several refreshing steps so that the riders will not miss it. Therefore, to make sure the positive feedbacks are not submerged in the negative feedbacks and to better capture the interests of riders, effective identification methods should be designed.

To address the above challenges, we propose an online deep reinforcement learning-based order recommendation (DRLOR) framework. The bulk of the DRLOR framework is composed of three networks, i.e., the actor-critic (AC) network, the rider behavior prediction (RBP) network and the feedback correlation (FC) network. Firstly, to deal with the dynamism of the delivery system, we model the original problem as a sequential decision process and formulate it with Markov Decision Process (MDP). The AC network is then designed to learn an optimal order ranking policy at each step, which will determine the final order recommendation list for each rider. To handle the uncertainty of rider behaviors, we establish the RBP network to predict whether a rider will grab an order from current order recommendation list or not. Finally, to identify valid feedback information and capture the interests of riders, the FC network is constructed with attention mechanism to find the relationship between historical recommendation lists and orders. The output of the network is a high-dimensional embedding of current state of the rider, which can be used by subsequent networks. For the AC network, it serves as a state representation which enters the actor and critic modules to generate the action and the estimated Q-value. For the RBP network, it condenses the information from original inputs and former layers, which helps improve the accuracy of predicting the rider behavior. To validate the effectiveness of our proposed framework, we conduct extensive offline and online experiments on Meituan platform. The main contributions of this paper include the following three aspects.

1) We propose an online deep reinforcement learning-based framework DRLOR to solve the decision-making problem in the rider-centered food delivery system. Actor-critic learning scheme is adopted to generate the best policy, resulting in a best order recommendation list for each rider so that the order can be grabbed and served by riders as soon as possible. To the best of our knowledge, this is the first work that employs online deep reinforcement learning for efficient order recommendation in food delivery systems with rider-centered assignment manner.
2) In the DRLOR framework, a novel FC network is proposed to identify valid feedback information. Attention mechanism is used to extract the relationship between different feedbacks. Besides, the output of this network is shared by the subsequent modules, which can effectively enhance the performance of reinforcement learning and rider behavior prediction.
3) Extensive experiments are conducted on real-world datasets from Meituan platform. The results demonstrate

that our proposed network and framework are distinctly superior to the comparative methods.

The remainder of the paper is organized as follows. Related works are investigated and summarized in section II. The decision-making problem in rider-centered food delivery system is described and formulated with MDP in section III. Section IV reveals the details of the designed DRLOR framework. Section V presents offline and online experiment results on real-world food delivery platform and some analyses and discussions of the experiment results. Finally, we conclude the paper and offer some future work directions in section VI.

II. Related Work

A. On-Demand Food Delivery Problems

Most researches have focused on the OFD problems with platform-centered assignment manner, i.e., which rider serves which orders needs to be exactly ascertained for each instance. Chen et al. [4] studied the OFD problem and abstracted the problem into a static generalized assignment problem with a rolling horizon strategy. An offline-optimization for online-operation framework and an imitation learning-enhanced iterated matching algorithm were proposed to effectively solve the problem. Wang et al. [5] decomposed the online food delivery problem and built a machine learning model to batch the orders together and then assigned orders by a modified Kuhn-Munkres algorithm. Liao et al. [6] studied a green meal delivery routing problem which integrated the meal delivery and vehicle routing problem. They combined the Nondominated Sorting Genetic Algorithm II (NSGA-II) and Adaptive Large Neighborhood Search (ALNS) to jointly optimize the solution quality. Reyes et al. [7] introduced the Meal Delivery Routing Problem and developed optimization-based algorithms to solve the courier assignment and capacity management problems encountered in meal delivery operations. Ji et al. [10] proposed a task grouping method for O2O food ordering and delivery platforms. Their method included a greedy algorithm and a replacement algorithm. Similarly, Paul et al. [11] developed an integrated optimization framework for batching and assignment of orders at a large-scale scenario. Yildiz and Savelsbergh [12] later introduced a novel formulation for a meal delivery routing problem where they assumed perfect information about order arrivals. They developed a simultaneous column and row generation method for effective solution generation. Ulmer et al. [13] considered a stochastic dynamic pickup and delivery problem where customer orders and restaurant preparation time were unknown. They presented a parametrizable cost function approximation which could postpone the assignment decisions for selected customers. Liu et al. [14] considered the uncertain travel time in on-time last mile delivery. They discussed a framework that integrates the travel time predictors with the order assignment optimization. Huang et al. [15] investigated the dynamic task scheduling problem with stochastic task arrival times and due dates in the UAV-based intelligent on-demand meal delivery system. They introduced new constraints and characteristics of UAVs in the problem model and proposed an iterated heuristic framework to periodically schedule tasks, consisting of a task collection phase and a dynamic task scheduling phase. Manchella et al. [16] considered the transportation scenario that combined passenger transportation with goods delivery. A distributed model-free deep reinforcement learning algorithm that jointly served passengers and goods workloads was developed. As for the problems with rider-centered assignment manner, Deng et al. [8] studied a version of the spatial crowdsourcing problem in which the workers autonomously selected their tasks. They tried to find a schedule for the worker that maximized the number of performed tasks. Two exact algorithms based on dynamic programming and branch-and-bound strategies were proposed. However, although workers selected tasks by themselves, the dynamism and the uncertainty of worker behaviors were not considered in this research. In our study, orders enter the food delivery system continuously and whether a rider picks up an order from the order recommendation list remains unknown until the behavior actually takes place, which is more realistic.

B. Recommendation Techniques

Since the kernel decision-making problem in the rider-centered food delivery system is to recommend an appropriate order list for rider, we investigate the related techniques for item recommendation in recommender systems. Conventional recommendation techniques include content-based filtering [17], [18], matrix factorization-based methods [19], [20], [21], logistic regression [22], factorization machines [23], [24], contextual multi-armed bandits [25], [26], [27] and some deep learning models [28], [29], [30], [31]. The common characteristic of these methods is that they regard the recommendation procedure as a static process, which assumes that the preference of users does not change. The recommender tries to learn the recommendation with the largest immediate reward without considering the long-term benefit of the system. However, the actual interaction between users and systems is a dynamic and continuous process, which means that the conventional techniques are limited in terms of the model learning performance. Therefore, modeling the recommendation problem with MDP and developing reinforcement learning (RL) techniques gradually attract people's attention. In RL-based techniques, model-free methods are more popular than model-based methods [32], [33], [34] since the latter have a higher time complexity, which is difficult to be applied to real recommendation scenario. Model-free RL techniques can be roughly divided into two categories, i.e., the value-based methods and policy-based methods. Value-based methods aim to evaluate the Q-value of the action as close as possible so as to select the best action that results in the largest benefit. Zhao et al. [35] considered the positive and negative feedbacks of user when designing the form of state in MDP and developed a novel approach to incorporate them into their proposed deep recommender system framework. Xin et al. [36] proposed a self-supervised reinforcement learning for sequential recommendation tasks, where RL output layer acted as a regularization to drive the supervised layer focusing on specific rewards. Zheng et al. [37] studied the news recommendation and considered user return pattern as a supplement to click or no click label in order to capture more user feedback information. They adopted the deep Q-network to predict the reward from different kinds

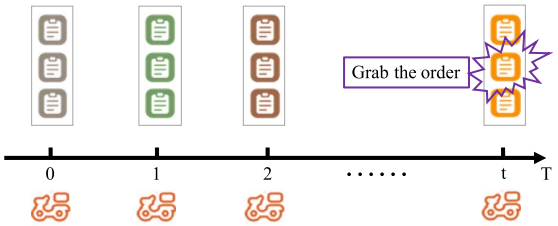


Fig. 2. Interaction process between the rider and the platform.

of features and employed a dueling bandit gradient descent algorithm to do the exploration in reinforcement learning. Hu et al. [38] proposed a novel policy gradient algorithm for learning an optimal ranking policy, which was able to deal with the problem of high reward variance and unbalanced reward distribution of a search session MDP. Zhao et al. [39] proposed a principled approach to jointly generate a set of complementary items and a novel page-wise recommendation framework based on deep reinforcement learning. Liu et al. [40] adopted actor-critic reinforcement learning scheme to model the interactions between the users and recommender systems, which considered both dynamic adaptation and long-term rewards. Chen et al. [41] proposed a tree-structured policy gradient recommendation framework, where a balanced hierarchical clustering tree was built over the items and picking an item was formulated as seeking a path from the root to a certain leaf of the tree. Deng et al. [42] introduced a recurrent deep neural network for real-time financial signal representation and trading and designed a model based on two biological-related learning concepts of deep learning and reinforcement learning. Zou et al. [43] tried to optimize long-term user engagement in the recommender system. They designed a RL-based framework which consisted of a Q-network to model complex user behaviors and a S-network to simulate the environment and assist the Q-network. Chen et al. [44] proposed knowledge-guided deep reinforcement learning to harness the advantages of both reinforcement learning and knowledge graphs for interactive recommendation.

Though many attempts have been made on policy-based reinforcement learning techniques, most of the existing works mainly focus on classic scenarios like online shopping, search session, and multi-media entertainment, where the total item sets are relatively static and the user feedback information are explicitly positive or negative. No research has considered the order recommendation problem in rider-centered food delivery system, which has a highly dynamic customer order occurrence and suffers from false-negative feedback information. Besides, novel state representation is hardly considered in most studies. Therefore, to extend the application of RL techniques to more realistic recommendation scenarios and address the above existing challenges, we propose the DRLOR framework which can effectively learn an actor that generates order recommendation lists for riders using embedded state representation and deep deterministic policy gradient algorithm.

III. Problem Description and Formulation

A. Problem Description

As introduced in previous sections, the kernel scheduling problem in the rider-centered food delivery system is to ascertain an order recommendation list for the rider at each interaction step, which can be regarded as a multi-step sequential decision-making problem. Fig. 2 shows the main process. Suppose each order is represented by m features. At each time step t, given a set of all available orders $O_t = \left\{o_1^t, o_2^t, o_3^t, \cdots\right\}, o_i^t \in \mathbb{R}^{1\times m}$, the OFD platform collects N_t orders that are appropriate for the rider from O_t and takes a ranking policy π_θ^* to generate a recommendation list $L_t = \{o_1^t, o_2^t, o_3^t, \cdots, o_{N_t}^t\}$. After the order list is displayed to the rider, the rider can browse the list and select a favorite order to serve. If no order raises the interest of the rider, the rider can launch another refresh request for a new order list. The process keeps repeating until the rider selects an order from the list to serve or the rider just stops requesting for personal reasons like resting or getting off work. Note that in our setting, the platform deals with only one rider at a time. That is to say, an action output by the actor is only suitable for generating recommendation list for the current rider. If there are multiple riders that need to be recommended order lists, each rider needs to collect his own state and use the corresponding action to rank orders and form the recommendation list.

TABLE I
Parameters for MDP Formulation

Symbol	Explanation
t	Time step
O_t	Set of available orders at time t
m	Dimension of order features
N_t	Number of orders in recommendation list at time t
L_t	Recommendation list at time t
π_θ	Policy parameterized by θ
S	State space
A	Action space
s	A certain state in state space
a	A certain action in action space
P	Transition function
p_t	Transition probability at time t
R	Reward function
γ	Discount rate

B. MDP Formulation

To formulate the multi-step sequential decision-making problem, MDP is a natural and very significant model, which is defined as (S, A, P, R, γ). S and A denote the state space and action space, respectively. P is the transition function which determines the flow direction between different states. R is the reward function which describes the feedback of the environment and guides the agent to select beneficial actions. $\gamma \in [0, 1]$ is the discount rate that balances the long-term benefit and short-term benefit. Table I explains the notations of the formulation. By treating the rider as the environment and the OFD platform as the agent, we can define the five components of MDP for our problem as follows.

1) State S: A state $s \in S$ is the original representation of the rider's current status, including four parts, i.e., carried orders, browse history, current list and rider properties. Carried orders refer to the orders that the rider already grabbed. Browse history includes all the recommendation lists generated through the interactions between the rider and the platform, describing the process of how the rider reaches the current status. Current list is the order recommendation list that the

rider is browsing at this very moment. Rider properties indicate relatively long-term characteristics of rider, such as the number of completed deliveries each day and the working hours each day.

2) Action A: An action $a \in A$ is defined as an m-dimensional vector with continuous values in the range of $[-1,1]$. When ranking the orders according to action a_t, each order receives its ranking score by calculating the inner product of a_t and the feature vector of the order. An action can be regarded as weight factors for features of each order. Therefore, the size of an action a is equal to the number of features of an order, which is set as $m = 26$ in our study. To calculate the ranking score (inner product value) of an order, for example, if order $o_1 = (feature_{1,} \cdots, feature_m)$ and the action $a = (weight_1, \cdots, weight_m)$, the ranking score of o_1 will be $score = a^{\mathrm{T}} o_1 = \sum_{i=1}^{m} weight_i \cdot feature_i$. The order recommendation list is then generated by ranking all the orders in a descending order of ranking score and sequentially selecting the first N_t orders with the highest scores.

3) Transition Function P: At each time step t, after the agent takes the action a_t, the state s_t will change and turn to s_{t+1} with a transition probability p_t. Normally the transition function is determined by the environment, which is full of uncertainty. In our work, we train the RBP network with real-world rider behavior data from Meituan platform to approximate the real transition function.

4) Reward Function R: After the platform takes action a_t and displays the order recommendation list to the rider, the rider will respond to the action and generate different feedbacks, like selecting one order to serve from the list or not selecting any order and refreshing the list. Different feedbacks result in different reward values, which will guide the agent to effectively learn the optimal policy and avoid generating order recommendation lists that riders do not like.

5) Discount Rate γ : Usually, simply focusing on the current feedback of the environment will inevitably cause short-sighted decision. γ is the factor to balance the short-term reward and long-term reward so that the agent is not trapped in local optimum, which can help the agent obtain higher cumulative reward. When $\gamma = 0$, the agent only concentrates on the immediate feedbacks while short-term reward and long-term reward are equally taken into consideration when $\gamma = 1$.

The objective of the MDP formulation is to find a best policy π_θ^* that maximizes the expected cumulative reward from any state $s \in S$.

$$\pi_\theta^* = argmax_{\pi_\theta} \mathbb{E}_{\pi_\theta} \left\{ \sum_{i=1}^{\infty} \gamma^i r_{t+i} | s_t = s \right\}, \quad \forall s \in S, \ \forall t \geq 0 \tag{1}$$

Here, $\pi_\theta : S \times A \rightarrow [0, 1]$ defines a policy of the agent, which is represented by a parameter vector θ. $\mathbb{E}_{\pi_\theta}$ is the expectation under policy π_θ. i is the time step in the future. r_{t+i} is the immediate reward that the agent receives in time step $t + i$.

To learn the best policy π_θ^*, reinforcement learning usually tries to find the optimal state value function $V^*(s)$ or the optimal state-action value function $Q^*(s, a)$, which can be expressed as (2) and (3), respectively.

$$V^*(s) = max_{\pi_\theta} \mathbb{E}_{\pi_\theta} \left\{ \sum_{i=1}^{\infty} \gamma^i r_{t+i} | s_t = s \right\} \tag{2}$$

$$Q^*(s, a) = max_{\pi_\theta} \mathbb{E}_{\pi_\theta} \left\{ \sum_{i=1}^{\infty} \gamma^i r_{t+i} | s_t = s, a_t = a \right\} \tag{3}$$

To obtain the optimal value function in the above equations, we need to calculate the gradient of the cumulative reward to the policy π_θ, or more specifically, the parameter θ. First, we rewrite the expected cumulative reward as follows,

$$\begin{aligned} J(\pi_\theta) &= \int_S \int_S \sum_{t=1}^{\infty} \gamma^{t-1} q_0(s') P(s', \pi_\theta(s'), s) \\ &\quad \times R(s, \pi_\theta(s)) \mathrm{d}s' \mathrm{d}s \\ &= \mathbb{E}_{\pi_\theta}[R(s, \pi_\theta(s))] \end{aligned} \tag{4}$$

where $\int_S \sum_{t=1}^{\infty} \gamma^{t-1} q_0(s') P(s', \pi_\theta(s'), s) R(s, \pi_\theta(s)) \mathrm{d}s'$ is the probability of the agent visiting state s in the whole learning process. q_0 is the state distribution at the beginning time step and $P(s', \pi_\theta(s'), s)$ is the transition probability from state s' to s. Then according to the policy gradient theorem and combining (3) and (4), the gradient of the expected cumulative reward $J(\pi_\theta)$ with respect to θ can be calculated as (5).

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi_\theta}(s, a) | a = \pi_\theta(s)] \tag{5}$$

In (5), $Q^{\pi_\theta}(s, a)$ represents the cumulative reward of the state-action pair (s, a) under policy π_θ. Therefore, parameter θ can generally be updated as follows,

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi_\theta}(s, a) | a = \pi_\theta(s) \tag{6}$$

where α_θ is the learning rate. In real application, it is often difficult to accurately calculate $Q^{\pi_\theta}(s, a)$. A simple method to estimate $Q^{\pi_\theta}(s, a)$ is optimizing a function approximator Q^w with a deep neural network parameterized with w by minimizing the mean squared error $MSE(w) = \| Q^w - Q^{\pi_\theta} \|^2$.

IV. The Proposed DRLOR Framework

As mentioned earlier in section I, there are mainly three aspects of challenges in handling the problem in the rider-centered food delivery system, i.e., the dynamism of food delivery system, the uncertainty of rider behaviors and the difficulty of identifying valid feedback information. To address these difficulties, we propose the DRLOR framework, whose structure is shown in Fig. 3, mainly including three networks, i.e., the AC network, the RBP network and the FC network. The AC network is designed to learn the best order recommendation policy. The RBP network is built to accurately predict the rider behavior. The FC network is established to capture the internal relationship between historical recommendation lists and orders. In the following content, we will elaborate the three networks and then introduce the training and test methods of the proposed DRLOR framework.

A. The Actor-Critic Network

The actor network and the critic network are actually both neural networks parameterized by θ and w, respectively. The input of the actor network is the state embedding of rider. Specifically, in DRLOR framework, the state embedding of a rider is represented by a high-dimensional vector with

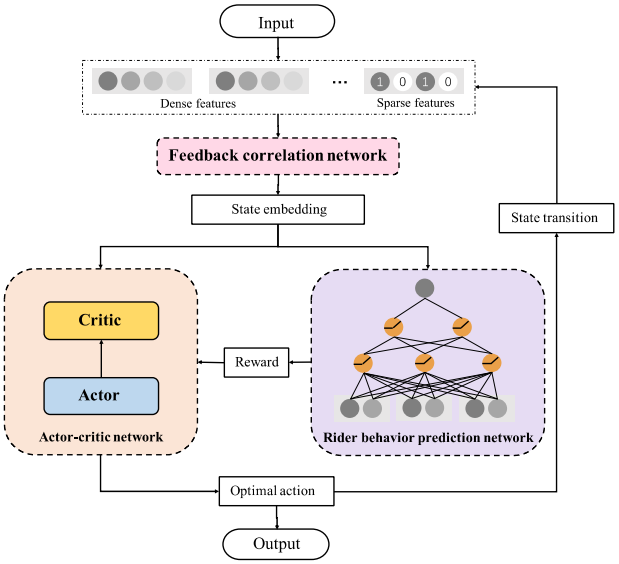


Fig. 3. The structure of the DRLOR framework.

continuous values, denoted as $FC(s_t)$, which is the output of the FC network we will introduce elaborately later in part C of Section IV. Different states of rider will lead to different state embeddings. Given the state s_t of a rider at time step t, we can calculate the final action a_t using following equation,

$$a_t = \tanh(\mathrm{ReLU}(FC(s_t))), \quad \forall s_t \in S \tag{7}$$

where ReLU denotes the activation function of multiple hidden layers of the actor network. Specifically, the leaky version of the ReLU activation is selected in our study. The activation function of the output layer of the actor network is set as tanh, which is appropriate for calculating the ranking score σ for each order as follows.

$$\sigma_i^t = o_i^t a_t^{\mathrm{T}}, \quad \forall o_i^t \in O_t \tag{8}$$

After we obtain the ranking score of each order, we can select the top N_t orders to sequentially form the recommendation list that will be exhibited to the rider.

The critic network serves as a function approximator $Q^w(s, a)$ mentioned in section III, which estimates the true state-action value function. It takes the state s and the action a as the input and outputs the Q-value of action a, i.e., the cumulative reward of state-action pair (s, a), which reflects the quality of the action generated by the actor. Similar to the actor network, we also employ a feedforward neural network to learn the best approximation of the true state-action value function. Given the state s_t and action a_t, the approximate value function $Q^w(s_t, a_t)$ can be calculated as,

$$Q^w(s_t, a_t) = \mathrm{Linear}(\mathrm{ReLU}(a_t \oplus FC(s_t))) \tag{9}$$

where $\oplus$ denotes the concatenation operation. The difference of the above equation to (7) is the activation function of the final output layer. The actor network uses a tanh activation while the critic uses a linear function, which is because the cumulative reward ranges larger than the action value.

To update the AC network, the actor needs to find the direction in which the Q-value is maximized while the critic needs to minimize the temporal difference error of the approximate value function and the true value function. According to the deterministic policy gradient theorem [45] and the temporal-difference learning approach [46], the loss functions of the actor network and critic network are calculated as (10) and (11).

$$loss^a = \frac{1}{B}\sum_t -Q^w(s_t, a_t) \tag{10}$$

$$\begin{aligned} loss^c &= MSE\left[Q^w(s,a), Q^{\pi_\theta}(s,a)\right] \\ &= \frac{1}{B}\sum_t \left(r_t + \gamma\, Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t)\right)^2 \end{aligned} \tag{11}$$

Here, B denotes the batch size. To overcome the difficulty of updating a continuously-changing network, we utilize the target network technique [47] when updating the AC network in DRLOR framework. Besides, to avoid the dramatic variation of parameters, we adopt the moving average updating method instead of directly replacing the old values with new ones, which can be expressed as (12) and (13).

$$\theta'_{t+1} = \lambda_1 \theta'_t + (1-\lambda_1)\theta_{t+1} \tag{12}$$

$$w'_{t+1} = \lambda_2 w'_t + (1-\lambda_2) w_{t+1} \tag{13}$$

θ' *and* w' represent the learnable parameters of the target networks. λ_1 and λ_2 are the factors that control the proportion of the old and new values in updating process.

According to the policy gradient theorem, the critic will finally converge to a state where it can approximately estimate the true Q-value with the minimization of loss. The actor will finally converge to generating the policy with the highest estimated Q-value during the parameter learning in the training stage. Therefore, the AC network is capable of finding the optimal policy with the largest cumulative reward from any state.

B. The Rider Behavior Prediction Network

As mentioned in previous sections, our reinforcement learning model falls into the area of online reinforcement learning, which requires environment (or an environment simulator) to give instant feedback to the agent. The response of the riders to the recommendation list generated by the platform determines the reward, which is very important for guiding the actor to learn towards a promising direction. However, in a real interaction process between riders and platform, rider behaviors are generally full of uncertainties. We cannot acquire direct feedback information for conducting offline experiments. Therefore, to accurately predict the behaviors of riders, we build the RBP network based on feedforward neural network, which can effectively simulate the real behaviors of riders and provide an accurate reward for the AC network. As shown in Fig. 3, the structure of the RBP network contains three parts, i.e., an input layer, two hidden layers and an output layer. The input layer receives the state embedding output by FC network. Then two hidden layers will transform the data from former layer based on the activation functions. Here, we select the commonly-used ReLU and tanh functions as the activation of the hidden layers. Finally, the activation function of the output layer is Sigmoid since the prediction task of the RBP network is binary, i.e., whether a rider grabs an order from the current recommendation list or not. Given the above

model structure, the output of the RBP network $\hat{y}_t$ can be calculated as follows.

$$\hat{y}_t = \text{Sigmoid}(\tanh(\text{ReLU}(FC(s_t)))) \quad (14)$$

The value of $\hat{y}_t$ is within the range of 0 and 1, which can be transformed to a binary result by regulating a probability threshold (normally set as 0.5). Specifically, we adopt the binary cross entropy as the loss function of the RBP network, which can be expressed as the following equation.

$$loss^r = -\frac{1}{B}\sum_{i=1}^{B} y_i \log\hat{y}_i + (1-y_i)\log(1-\hat{y}_i) \quad (15)$$

Here, y_i is the true label and $\hat{y}_i$ is the predicted value of the input. Through the back propagation of gradient, the weights of the neurons in RBP network can be updated in the direction of minimizing $loss^r$.

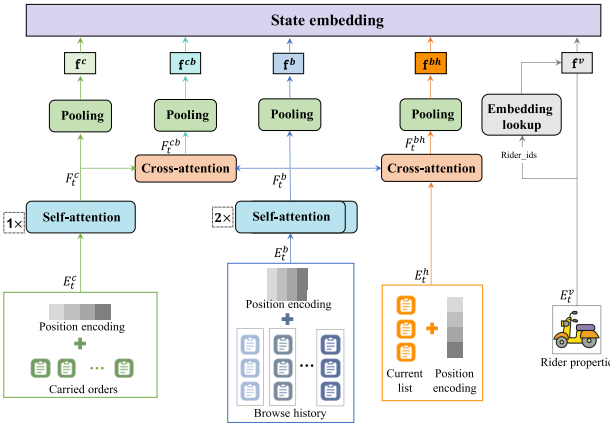


Fig. 4. The structure of the FC network.

C. The Feedback Correlation Network

Various feedback information exists in the interaction process between riders and the platform, including positive and negative feedbacks. In the scenario of OFD, positive feedbacks are easy to identify since the behavior of rider grabbing an order from the recommendation list can be explicitly observed. The grabbing behavior shows the positive interest of the rider to the grabbed order and the recommendation list. However, things are not that simple when it comes to the negative feedbacks. Literally, negative feedback refers to the scenarios when riders have no interest in the current recommendation list and do not grab any order from the list and choose to refresh. Nevertheless, not all the refreshing behavior means that riders are not interested. Sometimes, riders simply refresh the list, trying to compare the next list with the current one, aiming to find a better order if possible. We refer to this kind of negative feedback as false-negative feedback. Excavating the internal relationship between historical recommendation lists is essential for the identification of feedback types, which helps accurately predict the rider behaviors.

To tell the false-negative feedbacks from real negative feedbacks and excavate relationship between different feedbacks, we design the FC network, which serves as an essential part in the whole DRLOR framework. Inspired by the famous attention mechanism proposed by Vaswani et al. [48], we calculate the attention information based on positive and negative feedbacks to capture the latent intention of riders. The attention mechanism was firstly proposed and used in the scenario of machine learning translation and natural language processing. It calculates a specific weight for each word in a sentence such that the model can focus on the essential parts and understand the meaning of sentences when translating instead of treating each word equally. Similarly, we try to find essential orders and orders lists that affect the behavior of riders. In the proposed FC network, two types of attention are utilized, which are the self-attention and the cross-attention. The self-attention mainly focuses on the inside information in positive or negative feedbacks while the cross-attention concentrates on the interrelationship between positive feedbacks, negative feedbacks and current recommendation status. Fig. 4 elaborates the design of the FC network. It takes the following four types of features as inputs.

1) Carried orders: refer to the orders that a rider is currently carrying when interacting with the platform. These orders represent the positive interest of rider, denoted as $C_t = \{o_1^t, o_2^t, o_3^t, \ldots\}$. Besides, since the carried orders reach the rider sequentially, we also model this spatial information by adding a standard learnable one-dimensional position encoding for each set of carried orders C_t.
2) Browse history: refers to the set of all the order recommendation lists, starting from the first refreshing action after last grabbing behavior and ending at time step $t-1$. The browse history represents the negative feedbacks, in which many false-negative feedbacks also exist. The browse history of a rider at time t is defined as $H_t = \{h_0, h_1, h_2, \ldots, h_{t-1}\}$ and each history list consists of N_t orders. As for the position information, a standard learnable two-dimensional position encoding is constructed based on raw position information of the browse history, i.e., (1,1) for the first order in the first recommendation list, (1,2) for the second order in the first recommendation list, etc.
3) Current list: is the order recommendation list that the platform generates for the rider at time t, which can be expressed as $h_t = \{o_1^t, o_2^t, \ldots, o_{N_t}^t\}$. With the increase of time step, current list at time step t will become the latest history list in browse history at time step $t+1$. For position encoding, we adopt similar setting as browse history but without the position information of time step.
4) Rider properties: include 28 offline and online features of the rider such as rider identification number, the current position, the number of completed orders, the average workload of the rider and the average work duration of the rider. These features can uniquely reflect the characteristic of each rider and make the model capable of distinguishing different riders. Denote all the properties of rider as $v_t = \{v_1^t, v_2^t, v_3^t, \ldots\}$.

After entering the network, the above four types of inputs are firstly projected into a high-dimensional space with a dense layer to form corresponding initial embeddings. After adding corresponding position encodings, these initial embeddings will be transformed into complete input embeddings, which we will denote as E_t^c, E_t^b, E_t^h and E_t^v, respectively.

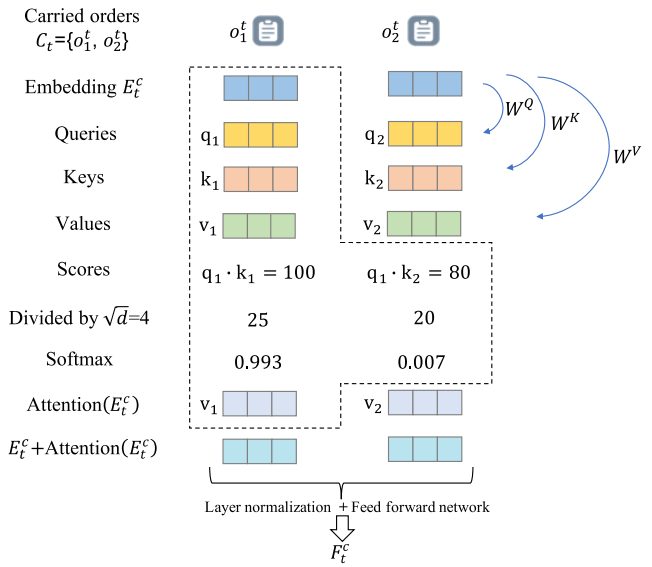


Fig. 5. A graphical example of calculating self-attention.

Then the positive feedbacks E_t^c and negative feedbacks E_t^b are subsequently sent into the self-attention module, which adopts similar structure of the encoder in transformer [48]. Taking the embedding of carried orders E_t^c as an example, the corresponding output F_t^c of the self-attention module can be sequentially calculated as,

$$\text{Attention}\left(E_t^c\right) = \text{Softmax}\left(\frac{\left(W^Q E_t^c\right)^{\mathrm{T}} W^K E_t^c}{\sqrt{d}}\right) W^V E_t^c \quad (16)$$

$$Z_t^c = \text{LN}\left(E_t^c + \text{Attention}\left(E_t^c\right)\right) \quad (17)$$

$$F_t^c = \text{LN}\left(Z_t^c + \text{FFN}\left(Z_t^c\right)\right) \quad (18)$$

where $W^{\cdot}$ represents the projection matrices for calculating the query, key and value. d is the dimension of the query, key and value. LN represents the layer normalization and FFN denotes the feed forward network. Fig. 5 gives a graphical illustration of the above process of calculating self-attention output, where the calculation of attention weights focuses on the first order o_1^t. Firstly, we calculate the query, key and value vectors with the projection matrices W^Q, W^K and W^V. Then the scaled dot-product attention is applied between query and key to calculate the score of each order. The score is then normalized by $\sqrt{d}$ (equal to 4 in the graphical example) to guarantee the stability of the gradient. The score is subsequently used for softmax operation to calculate the importance weight. Finally, by multiplying the value vector with the importance weight, we obtain the attention weight of order o_1^t. Then through layer normalization and feed forward network, the corresponding output F_t^c of the self-attention module is calculated. Similarly, the self-attention output of browse history E_t^b is denoted as F_t^b. Note that when calculating the self-attention result, multi-head technique is adopted to increase the capability of model learning useful information in embedding space. Besides, for carried orders we use only one self-attention module to find the relationship between orders while for the browse history we use two to capture information of both orders and recommendation lists.

After calculating the self-attention, we can calculate the cross-attention using a similar process with the only difference of calculating query, key and value vectors. In self-attention, we calculate the query, key and value based on the same input embeddings (E_t^c or E_t^b). For example, the query, key and value in Fig. 5 are all from embedding E_t^c. However, in cross-attention, we use different inputs to obtain queries, keys and values. On the one hand, for the cross-attention output between carried orders and browse history (denoted as F_t^{cb}), we select F_t^c to calculate the query and F_t^b to calculate the key and value. This will help the model find which orders the rider used to be interested in, which corresponds to the target of identifying false-negative feedbacks. On the other hand, for the cross-attention output between current list and browse history (denoted as F_t^{bh}), we use E_t^h to calculate the query and F_t^b to calculate the key and value, which can model the similarity between current list and history lists.

Through average pooling operation, the attention outputs F_t^c, F_t^{cb}, F_t^b and F_t^{bh} can be finally transformed into encodings f^c, f^{cb}, f^b and f^{bh}, respectively. Besides, for the rider properties, we use an embedding lookup layer to handle discrete identity features and use a dense layer to generate the embedding of rider properties, which is denoted as f^v. With the above encodings and embeddings, the final state embedding of rider can be calculated as the following equation.

$$FC\left(s_t\right) = \text{Dense}(\mathrm{f}^c \oplus \mathrm{f}^{cb} \oplus \mathrm{f}^b \oplus \mathrm{f}^{bh} \oplus \mathrm{f}^v) \quad (19)$$

Here, Dense is a full-connection layer and $\oplus$ still represents the concatenation operation. As mentioned earlier, the state embedding $FC\left(s_t\right)$ can be used by both the AC network and the RBP network, to help improve the performance of the whole DRLOR framework.

D. Training of DRLOR Framework

There are two training procedures that need to be conducted in the DRLOR framework, which are the training of the AC network and the joint pre-training of RBP network and the FC network. The latter pre-training process needs to be completed before training the AC network since the RBP network directly determines the reward of the MDP and the FC network determines the state embedding of the MDP.

We first conduct the joint pre-training procedure of the RBP network and the FC network, which is shown in Algorithm 1. For each data batch, the embeddings and attention results of the rider's positive and negative feedback information are sequentially calculated by the FC network. Then the output of the FC network is sent into the RBP network, which utilizes the feedforward neural network to predict the rider behavior $\hat{y}_t$. At last, the stochastic gradient descent algorithm is adopted to back propagate the gradient and update the weights of the networks.

After the joint pre-training is done, we can then train the AC network to generate best policy for recommending orders to riders. Specifically, we utilize the deep deterministic policy gradient algorithm to train the actor critic network. The reward function is set as follows.

$$r_t = \begin{cases} g_1 & \hat{y}_t \geq 0.5 \\ -g_2 & \hat{y}_t < 0.5 \end{cases} \quad (20)$$

Algorithm 1 Joint Pre-Training of the RBP and FC Networks

Input: State $s_t = [C_t, H_t, h_t, v_t]$, ground truth y_t, batch size B, number of data batches $batch\_num$
Output: Trained parameters of RBP and FC networks
Initialize the RBP and FC networks with random weights
For $num = 1$ **to** $batch\_num$ **do**
  Obtain the initial embeddings E_t^c, E_t^b, E_t^h and E_t^v of the inputs
  Calculate the final feature embeddings $\mathrm{f}^c, \mathrm{f}^{cb}, \mathrm{f}^b, \mathrm{f}^{bh}, \mathrm{f}^v$ according to (16), (17) and (18)
  Calculate the state embedding as $FC(s_t) = \mathrm{Dense}(\mathrm{f}^c \oplus \mathrm{f}^{cb} \oplus \mathrm{f}^b \oplus \mathrm{f}^{bh} \oplus \mathrm{f}^v)$
  Put the state embedding into the RBP network and obtain the binary prediction result of rider behavior as $\hat{y}_t = \mathrm{Sigmoid}(\tanh(\mathrm{ReLU}(FC(s_t))))$
  Update all the trainable parameters of RBP and FC networks by minimizing the cross entropy loss: $loss^r = -1/B \sum_{i=1}^{B} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$
End for
Return the trained parameters of RBP and FC networks

Algorithm 2 Training Algorithm of the AC Network

Input: Batch size B, critic learning rate l^r, actor learning rate l^a, discount factor γ
Output: Trained parameters of AC networks
Randomly initialize the actor and the critic network with parameters θ and w
Initialize the replay memory D
For $session = 1$ **to** $\mathcal{M}$ **do**
  Generate the initial state s_0 randomly
  For $t = 1$ **to** T **do**
    Observe state $s_t = [C_t, H_t, h_t, v_t]$
    Calculate the state embedding $FC(s_t)$ using the well-trained FC network
    Generate an action $a_t = \pi_\theta(s_t)$ and form a corresponding order recommendation list for the rider
    Calculate the reward r_t according to (14) and (20)
    Transfer state s_t to state s_{t+1} by changing C_t, H_t, h_t *and* v_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay memory D
    Sample a minibatch B from the replay memory D if there is enough replay data
    Update the critic network by minimizing the $loss^c$ in (11)
    Update the actor network by minimizing the $loss^a$ in (10)
    Update the parameters θ' and w' of targe networks by (12) and (13)
  End for
End for
Return parameters θ and w

g_1 and g_2 are positive constants. Note that when the rider does not grab orders from current recommendation list, the reward is not zero but a negative value, meaning that each interaction step has a cost, which will promote the agent to generate appropriate recommendation lists as soon as possible. The details of training the AC network are shown in Algorithm 2. Firstly, we initialize the actor and critic randomly. The actor generates an action a_t based on the current state s_t. The rider responds to the action and the state transfers from s_t to s_{t+1}. According to the rider's feedback, the reward can be ascertained by (20) and the actor can adjust the policy to maximize the reward. According to Algorithm 2, the computational complexity of the training algorithm of AC network mainly consists of two parts, i.e., the interaction length with between the agent and the environment, and the number of total interactions. Therefore, the complexity can be generally calculated as $\mathrm{O}(Num \cdot \mathcal{M} \cdot T)$, where *Num* is the number of training samples, $\mathcal{M}$ is the number of episodes for each sample and T is the maximum interaction step in the training stage of each sample.

E. Online Test of DRLOR Framework

To test the performance of the trained AC network, we still use the prediction result of RBP network to simulate the real interaction result between riders and the platform. The joint pre-training of the RBP and FC networks is conducted on real rider behavior data and the subsequent experimental results will show that RBP can accurately simulate the real online environment, which enables the online test of DRLOR framework using the well-trained model.

To test the DRLOR framework, we first randomly initialize the rider state. Then the action a_t is generated by actor based on current state s_t. Using the action a_t, the ranking score of each order can be calculated and an order recommendation list is correspondingly generated. The feedbacks of rider to the action a_t can be predicted by the RBP and FC networks, with which the final reward can be ascertained by (20). The detailed online test process of DRLOR framework is presented in Algorithm 3. Note that according to the definition of the former reward function, the upper bound of the cumulative reward is equal to g_1, which means that the rider grabs an order at the first interaction step with the platform. The lower bound of the cumulative reward is $-g_2 \times T$, where T is the maximum number of each interaction. The upper bound and the lower bound are valid for all instances. Besides, they are both achievable although it is difficult to reach the upper bound. Fig. 6 shows the cumulative reward when testing the DRLOR on a specific example whose interaction step is 11. The upper bound and lower bound are also drawn to show how far the DRLOR method is to the optimum.

V. Computational Study

A. Experimental Settings

To effectively evaluate the performance of our proposed DRLOR framework, we collect real-world data from Meituan platform to generate the datasets. To be specific, 2.93 million

Algorithm 3 Online Test of the DRLOR Framework

Input: Batch size B, critic learning rate l^r, actor learning rate l^a, discount factor γ
Output: The cumulative reward r

```
Initialize the actor network with well-trained parameters θ
For session = 1 to M do
  Randomly generate the initial state s_0 of rider
  r = 0
  For t = 1 to T do
    Observe state s_t and calculate the state embedding FC(s_t)
    Execute action a_t and expose the corresponding order recommendation list to the rider
    Observe the feedback of riders using FC and RBP network and obtain the reward r_t
    Calculate the cumulative reward r = r + r_t
    Transfer state s_t to state s_{t+1}
  End for
End for
Return r
```

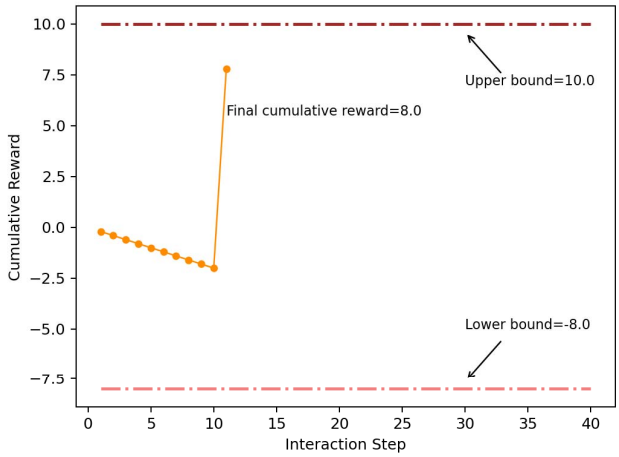


Fig. 6. The cumulative reward at test stage.

interactions between different riders and Meituan platform are collected from February 11th, 2022 to February 18th, 2022. For the training of FC and RBP networks, we use data from February 11th, 2022 to February 17th, 2022 to form the training set while the data on February 18th, 2022 serves as the test dataset. For the training of AC network, we select data on February 18th, 2022 so as to maximally simulate the real environment and utilize the first 70% data as the training set and the remaining 30% as the test dataset.

Before starting the training process, the hyper-parameters of the models need to be determined. Firstly, we set $N_t \leq 40$ for each recommendation list empirically based on the actual circumstance of delivery platform. The maximum interaction step is determined as $T = 40$ according to the statistics of the datasets. Besides, the maximum number of orders carried by riders is 15 and the maximum number of time steps in browse history data is set as 40. For the parameters of the FC network, the dimension of the state embedding output by FC network is 128. The number of heads in the attention procedure is 8. Dropout rate is 10^{-1} and the learning rate is set as 10^{-4}. The RBP network consists of a 3-layer feedforward neural network with dimension of $128 \times 128 \times 1$. For the AC network, the discount factor γ is set as 0.95. The learning rate of the actor and critic is set as 10^{-4} and 10^{-3}, respectively. The factor of updating parameters $\lambda_1 = \lambda_2 = 0.99$. Batch size B is set as 64. The positive constants in reward function g_1 and g_2 is empirically set as 10 and 0.2, respectively. Finally, the total number of episodes for training each session is set as 500.

The proposed methods are all implemented with Python 3.6 and Tensorflow 2.0. The training and test procedures of the models are conducted on the servers of Meituan.

B. Joint Pre-Training Performance of RBP and FC Networks

To evaluate the performance of RBP and FC networks, we utilize two commonly-used metrics in binary classification tasks, which are the Area Under receiver operating characteristic Curve (AUC) and the Area Under Precision-Recall curve (AUPR). Both these two metrics comprehensively measure the performance of a machine learning model with a regular value in [0,1], where the larger the metric value, the better the performance of the model.

To thoroughly demonstrate the effectiveness of the proposed networks in capturing the information of rider feedbacks and excavating the relationship between these feedbacks, we conduct two types of experiments from different perspectives. Denote the initial RBP and FC networks as RBFC. Firstly, to show the superiority of the proposed methods to others, we compare the proposed methods with some representative baseline methods. Several classic classification methods in the field of supervised learning are involved, including the Naïve bayes, the Decision tree, the Random forest, the eXtreme Gradient Boosting (XGBoost) [49], and the factorization-machine based neural network (DeepFM) [50]. Secondly, to validate the effectiveness of the network structure, we conduct the ablation experiment, which includes multiple variants of the proposed RBP and FC networks. The variants are named and explained as follows.

1) RBFC_noid: refers to the variant of RBFC which eliminates the identity-related features of riders. Correspondingly, the embedding lookup module in the FC network is also eliminated. This variant is designed to show the influence of the identity-related features.
2) RBFC_noatn: refers to the variant that has no attention-related modules in the network. That is to say, the self-attention modules for positive and negative feedbacks and the attention modules between positive and negative feedbacks are removed from FC network, which can demonstrate the importance of excavating relationships inside and between the inputs.
3) RBFC_nofb: denotes the variant that does not have any feedback information to assist the prediction process of the model. The features of carried orders (positive feedbacks) and the browse history (negative or false-negative feedbacks) are eliminated from the inputs. This variant aims to demonstrate whether the feedback information can improve the performance of the predictive model,

TABLE II
COMPARISONS WITH DIFFERENT METHODS

Methods	AUC	AUPR
Naïve bayes	0.5750	0.3261
Decision tree	0.7367	0.4877
Random forest	0.7414	0.5140
XGBoost	0.8050	0.5715
DeepFM	0.8357	0.6749
RBFC (our model)	**0.8864**	**0.7514**

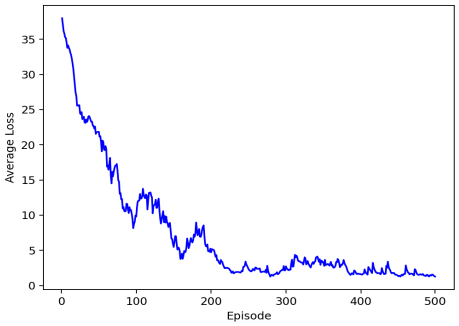
TABLE III
RESULTS OF ABLATION EXPERIMENT

Variants	AUC	AUPR
RBFC	**0.8864**	**0.7514**
RBFC_noid	0.8625	0.6987
RBFC_noatn	0.7217	0.3902
RBFC_nofb	0.5295	0.2919

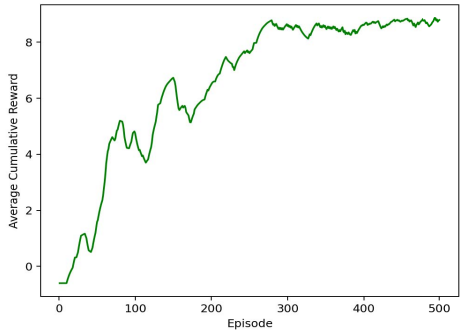
which helps make a good decision in real food delivery process.

We firstly exhibit the comparative results between different machine learning classification methods. Table II shows the AUC and AUPR value of each comparative method. From the results we can observe that the proposed RBP and FC network significantly outperforms other classification methods on the task of predicting the grabbing behavior of riders, showing an AUC value of 0.8864. The performance of DeepFM method follows the RBFC due to the low-order and high-order feature interactions learned by its factorization machine and deep feedforward components. XGBoost beats the other three classic methods due to the advantage of ensemble learning fashion, which can increase the accuracy of evaluation and avoid the occurrence of overfitting. However, since these two methods simply use different feedback information instead of fully excavating the relationship between them, their performance is not superior to RBFC. Naïve bayes, Decision tree and Random forest obtain worse performance than DeepFM, XGBoost and RBFC, which is because they treat all the features in the input data equally and ignore the temporal information of the browse history and the rank information inside every single recommendation list. Therefore, given the performance in Table II, we can draw the conclusion that the proposed RBP and FC networks are superior to the comparative models and can accurately simulate the real decision result of rider during the interaction process with the platform.

Secondly, we present the results of the ablation experiment in Table III. It can be seen that the performance of the first variant RBFC_noid slightly declines due to the lack of identity-related features in the inputs. Without identity-related features, it will be more difficult for the model to learn the preference of different riders. Under such circumstance, identifying different riders can only be completed by other continuous features, which is neither accurate nor efficient for conjecturing riders' preferences. RBFC_noatn shows a significantly worse performance than RBFC and RBFC_noid since no attention module exists in this variant, which makes the model lose the ability of generating embeddings of relationship between different feedbacks, proving the importance and the effectiveness of the attention mechanism in FC network. Finally, RBFC_nofb gives the worst performance of the comparatives, which means that the decision-making process of rider is highly dynamic and sequential. Rider behavior largely relies on the whole interaction process with the platform rather than only on current recommendation list, i.e., the interaction history contributes a lot to the occurrence of rider's grabbing behavior.



(a) Loss of AC network (critic)



(b) Reward of AC network

Fig. 7. Loss and reward of AC network in the training stage.

C. Offline Comparative Performance of DRLOR Framework

Since the test of DRLOR framework is conducted in an online fashion as mentioned earlier, we leverage the number of time step and the accumulated rewards in the interaction session as the evaluation metrics. Fig. 7 show the curve of the training loss and reward of the AC network. From Fig. 7 we can see that with respect to the training episodes, the loss and reward gradually converge which means that the agent has already been well trained on the training data. The fluctuation of the curves is actually the agent exploring a new state that has never been met and trying to adjust the approximate Q-value closer to the real Q-value.

In the testing stage, in order to show the superiority of the DRLOR framework, we adopt several typical ranking methods in OFD as the comparatives.

1) Rem_time: refers to the heuristic that generates recommendation lists for riders by ranking orders in the descending order of the remaining delivery time, which is the difference of the estimated delivery time promised for the customers and current time.
2) Delivery_dis: refers to the heuristic that ranks orders in an ascending order of the delivery distance of each order, which is the distance between the restaurant and the customer address.

TABLE IV
COMPARATIVE RESULTS WITH DIFFERENT NUMBER OF SURROUNDING ORDERS

Surrounding Orders	Number of Time Steps											
	DRLOR		Rem_time		Delivery_dis		Fetch_dis		Rider_pref		Rand	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
[0,40)	**3.198**	**1.803**	10.307	4.973	9.461	4.987	5.818	3.742	3.385	3.666	15.440	4.078
[40,80)	**2.973**	2.810	11.378	4.728	10.476	4.866	5.823	3.693	3.149	**2.789**	17.388	4.774
[80,120)	**2.642**	**2.948**	11.438	5.819	9.570	5.689	5.769	4.107	2.820	3.302	17.507	4.975
[120,160)	**2.513**	**2.163**	12.546	6.044	10.786	5.237	5.640	3.706	2.782	3.421	18.604	5.163
[160,200)	**2.527**	**2.126**	11.201	5.729	10.818	5.061	5.632	3.654	2.752	2.578	18.145	5.686

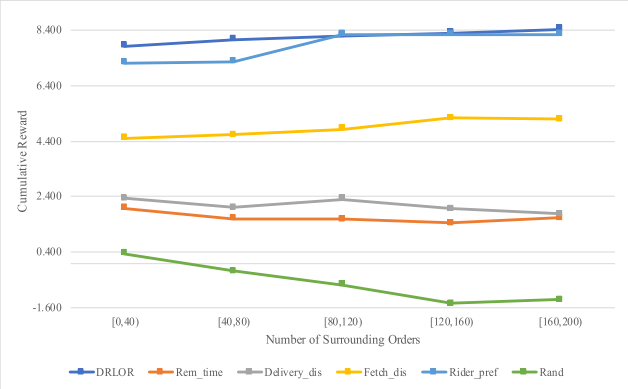


Fig. 8. Cumulative reward under different numbers of surrounding orders.

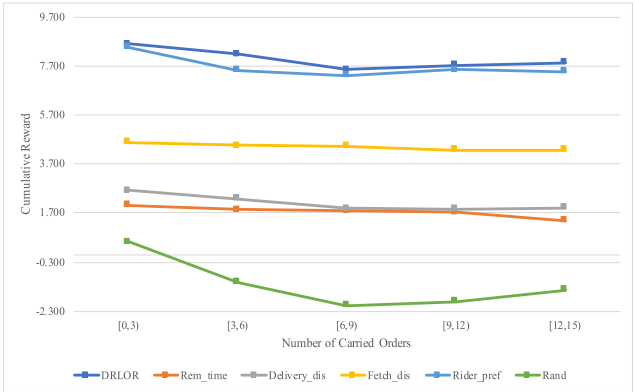


Fig. 9. Cumulative reward under different numbers of carried orders.

3) Fetch_dis: ranks orders in an ascending order of the fetching distance of each order. The fetching distance is the distance between the current location of the rider and the location of the restaurant.
4) Rider_pref: is an XGBoost-based method to rank the orders for riders. The XGBoost model assesses each rider-order pair and output a probability value to measure how much the rider likes the order. The recommendation list is then generated by ranking orders in a descending order of the probability value.
5) Rand: refers to randomly generating a recommendation list at each time step.

To comprehensively evaluate the proposed framework and explore the tendency with respect to different factors, we categorize the results according to different numbers of surrounding orders and different numbers of carried orders since these two factors are frequently discussed in real food delivery scenario. Table IV and Fig. 8 show the number of time steps and the reward of each comparative according to different levels of the number of orders, respectively. From Table IV it can be observed that DRLOR outperforms the other five comparative methods in terms of the average number of time step under all groups of test datasets, which demonstrates the strong learning ability of the actor in AC network and representative capability of the state embedding generated by FC network. Rider_pref performs the second best since it adopts multiple features of riders and orders to assess the fitness of rider-order pair. However, it does not include any dynamic feedback information from riders, which makes it less competitive than DRLOR. Note that Fetch_dis performs better than Rem_time and Delivery_dis, from which we can draw a conclusion that riders are more sensitive on the fetching distance than the delivery distance and the remaining time of orders. Small fetching distance of order will promote riders to grab the order probably due to the reason that riders need to head to the restaurant first. Grabbing the orders that are close to the current location is intuitive. Besides, Fig. 8 shows the trend of different methods with the increase of the number of orders. It can be seen that the cumulative rewards of DRLOR, Rider_pref and Fetch_dis increases while the other three methods decrease. This is because when the number of orders becomes larger, there will be more potentially appropriate orders for the rider. DRLOR, Rider_pref and Fetch_dis can pick these orders with potential out of the order pool and exhibit them to the rider while Rem_time, Delivery_dis and Rand cannot or even suffer from the increase of the number of orders. Besides, When the number of surrounding orders is very large, the difference between DRLOR and Rider_pref will diminish. This is because when the number of surrounding orders increases, there is more chance that Rider_pref can find orders that riders like and obtain similar performance as DRLOR. But when there is not enough orders, things will be difficult for Rider_pref. Note that for every method, the standard deviation value is sometimes large, this phenomenon mainly results from the stochastic nature of rider's grabbing behaviors in real delivery scenarios. For some samples in the dataset, riders may at a status where they do not have the feeling for serving any order, which will result in large interaction steps in our experiments.

Table V and Fig. 9 present the statistical and graphic results under different numbers of carried orders. From the results we can see that DRLOR still consumes the smallest number of

TABLE V
COMPARATIVE RESULTS WITH DIFFERENT NUMBER OF CARRIED ORDERS

Carried orders	Number of Time Steps											
	DRLOR		Rem_time		Delivery_dis		Fetch_dis		Rider_pref		Rand	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
[0,3)	**2.350**	**1.406**	9.967	4.564	9.082	4.613	5.252	3.573	2.631	3.062	14.509	4.631
[3,6)	**2.710**	**2.553**	10.938	5.052	9.947	4.923	5.807	3.843	3.413	3.625	17.127	4.933
[6,9)	**3.338**	**2.413**	11.596	5.669	10.770	5.367	5.739	4.221	3.632	2.961	19.528	5.098
[9,12)	**3.275**	2.820	11.482	5.865	10.923	5.263	5.937	3.537	3.477	**2.513**	19.490	4.730
[12,15)	**3.144**	**2.945**	12.910	5.960	10.410	5.877	5.941	3.922	3.589	3.345	18.763	5.628

TABLE VI
RESULTS OF ONLINE A/B TEST IN MEITUAN DELIVERY PLATFORM

Metric	Blank dates			Experiment dates			
	Control area	Experiment area	Δ_{blank}	Control area	Experiment area	Δ_{exp}	Δ_{true}
AGD	163.49	165.87	2.38	263.43	237.73	-25.71	**-28.09**
$ARTG$	11.48	11.56	0.08	13.14	11.86	-1.27	**-1.35**
GR_5	77.64%	77.18%	-0.46%	65.06%	67.45%	2.39%	**2.85%**
GR_2	59.47%	58.80%	-0.67%	44.95%	47.53%	2.58%	**3.24%**

time steps and obtain the largest average cumulative reward among all comparative methods. Fig. 9 suggests that with the increase of the number of carried orders, the cumulative rewards of all comparatives decrease, which indicates that when riders are carrying a heavy load, they will focus less on browsing and grabbing new orders, leading to fewer order grabbing actions. This also explains why the superiority of DRLOR to Rider_pref diminishes when the number of carried orders increases. Besides, the standard deviation in Table V becomes larger with the increase of the number of carried orders, suggesting that the rider behavior gains more uncertainty when a large number of orders are carried by riders.

D. Online A/B Test of DRLOR Framework

To test the proposed method in real delivery scenarios, we deploy our method on Meituan platform and conduct online A/B test experiment. Different from the offline experiment, we can directly observe the rider's behavior and hence can evaluate the proposed method with more realistic metrics that are relevant to the experience of riders and customers. Correspondingly, we define the following online observation metrics to evaluate the performance of different methods.

1) Average grab duration (unit: minute):

$$AGD = \frac{1}{|O|}\sum\nolimits_{o_i \in O}(GT_i - ET_i) \tag{21}$$

2) Average refresh times between grabs:

$$ARTG = \frac{1}{|V|}\sum\nolimits_{v_i \in V}\frac{RT_i}{O_i} \tag{22}$$

3) 5-minute grab rate:

$$GR_5 = \frac{|O_{5^-}|}{|O|} \times 100\% \tag{23}$$

4) 2-minute grab rate:

$$GR_2 = \frac{|O_{2^-}|}{|O|} \times 100\% \tag{24}$$

Here, O is the set of all accomplished orders. GT_i is the time when order o_i is grabbed by a rider. ET_i is the time when order o_i is firstly exposed in a recommendation list. V is the set of riders that serve at least one order within the experiment time. RT_i refers to the total refresh times of rider v_i. O_i is the number of all accomplished orders by rider v_i. O_{5^-} is the set of orders that are grabbed within 5 minutes since they are generated by customers. O_{2^-} is the set of orders that are grabbed within 2 minutes since they are generated by customers. Generally, the smaller the AGD and ARTG, the better the experience of riders. The larger the GR_5 and GR_2, the better the experience of customers.

Online A/B test requires comparison between two different methods (which is the reason it is called A/B test). Based on the performance on offline experiment results, we select the second best method Rider_pref to be the comparative method. Then the online A/B test is setting as follows.

Firstly, we randomly select a city to conduct the A/B test. To fairly compare the two methods and eliminate the disturbance of irrelevant factors, we need to find two areas in each city that are almost identical in terms of the predefined online observation metrics. One is for the Rider_pref method (called control area) and another is for the DRLOR method (called experiment area). In order to find these two areas, we collect the observation metrics of areas in the city for a number of days (denoted as blank dates) and select the closest two areas. After the blank dates is over, we enter the experiment dates and separately execute the Rider_pref method in the control area and DRLOR method in the experiment area to generate order recommendation lists for the riders. In our settings, the

blank dates are from November 3rd, 2022 to November 22nd, 2022. The experiment dates are from November 23rd, 2022 to November 29th, 2022.

To mitigate the inevitable error between two areas, we calculate the difference of metrics between control area and experiment area during blank dates as Δ_{blank}. Similarly, we calculate the difference between control area and experiment area during experiment dates as Δ_{exp}. Then the true difference between two comparative methods could be calculated as $\Delta_{true} = \Delta_{exp} - \Delta_{blank}$. Table VI exhibits the results of online A/B test. From the result it can be seen that compared with the Rider_pref method, the proposed DRLOR framework can effectively reduce the grab duration and refresh times of riders and improve the 5-minute and 2-minute grab rate. Therefore, the DRLOR framework is effective in improving the experience of riders and customers, which is of significant application value in real food delivery platforms.

VI. Conclusion and Future Work

In this paper, we present the decision-making problem in the rider-centered food delivery system, which is formulated with MDP. We propose an order recommendation framework DRLOR based on online deep reinforcement learning to handle the difficulties of dynamism, uncertain rider behaviors and excavating valid feedback information. The proposed DRLOR has following advantages compared with other recommendation models. Firstly, the AC network allows the actor to continuously update the policy according to the instant feedback of riders in the interaction process, which helps the platform shorten the length of interaction session and provide a faster service for customers. Secondly, the FC network uses attention mechanism to identify false-negative feedbacks and capture the internal relationship between different feedback information, which helps construct effective state embeddings to describe different states of rider. The proposed framework is trained and tested on the real-world datasets and deployed on Meituan platform. The experimental results show that the interaction process between riders and platform can be significantly accelerated by DRLOR.

Nevertheless, there are still some limitations of the proposed framework, together with some future research directions. First, we can further reduce the gap between the simulated environment and the real interaction environment by trying more complicated model structures and realistic features. Second, more rider feedback patterns need to be investigated and implemented. For example, adding a dislike button when exhibiting each order can provide explicitly negative feedbacks for the platform to filter the orders that riders do not like [51]. Finally, adding auxiliary function to shape the reward is worth studying since a good reward function can accelerate the convergence of reinforcement learning methods.

References

[1] Y. Li, F. Chu, C. Feng, C. Chu, and M. Zhou, "Integrated production inventory routing planning for intelligent food logistics systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 3, pp. 867–878, Jun. 2018.

[2] A. Seghezzi, M. Winkenbach, and R. Mangiaracina, "On-demand food delivery: A systematic literature review," *Int. J. Logistics Manage.*, vol. 32, no. 4, pp. 1334–1355, Oct. 2021.

[3] Meituan. *Homepage of Meituan Delivery*. Accessed: Jun. 8, 2022. [Online]. Available: https://peisong.meituan.com/about

[4] J. Chen et al., "An imitation learning-enhanced iterated matching algorithm for on-demand food delivery," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 18603–18619, Oct. 2022.

[5] X. Wang, L. Wang, S. Wang, Y. Yu, J. Chen, and J. Zheng, "Solving online food delivery problem via an effective hybrid algorithm with intelligent batching strategy," in *Proc. Int. Conf. Intell. Comput.*, 2021, pp. 340–354.

[6] W. Liao, L. Zhang, and Z. Wei, "Multi-objective green meal delivery routing problem based on a two-stage solution strategy," *J. Cleaner Prod.*, vol. 258, Jun. 2020, Art. no. 120627.

[7] D. Reyes, A. Erera, M. Savelsbergh, S. Sahasrabudhe, and R. O'Neil. (Mar. 2018). *The Meal Delivery Routing Problem*. [Online]. Available: https://optimization-online.org/?p=15144

[8] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *Proc. 21st ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, Nov. 2013, pp. 324–333.

[9] A. Alnaggar, F. Gzara, and J. H. Bookbinder, "Crowdsourced delivery: A review of platforms and academic literature," *Omega*, vol. 98, Jan. 2021, Art. no. 102139.

[10] S. Ji, Y. Zheng, Z. Wang, and T. Li, "Alleviating users' pain of waiting: Effective task grouping for online-to-offline food delivery services," in *Proc. World Wide Web Conf.*, May 2019, pp. 773–783.

[11] S. Paul, S. Rathee, J. Matthew, and K. M. Adusumilli, "An optimization framework for on-demand meal delivery system," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manage. (IEEM)*, Dec. 2020, pp. 822–826.

[12] B. Yildiz and M. Savelsbergh, "Provably high-quality solutions for the meal delivery routing problem," *Transp. Sci.*, vol. 53, no. 5, pp. 1372–1388, Sep. 2019.

[13] M. W. Ulmer, B. W. Thomas, A. M. Campbell, and N. Woyak, "The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times," *Transp. Sci.*, vol. 55, no. 1, pp. 75–100, Jan. 2021.

[14] S. Liu, L. He, and Z.-J. Max Shen, "On-time last-mile delivery: Order assignment with travel-time predictors," *Manage. Sci.*, vol. 67, no. 7, pp. 4095–4119, Jul. 2021.

[15] H. Huang, C. Hu, J. Zhu, M. Wu, and R. Malekian, "Stochastic task scheduling in UAV-based intelligent on-demand meal delivery system," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 13040–13054, Aug. 2021.

[16] K. Manchella, A. K. Umrawal, and V. Aggarwal, "Flexpool: A distributed model-free deep reinforcement learning algorithm for joint passengers and goods transportation," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2035–2047, Apr. 2021.

[17] R. J. Mooney and L. Roy, "Content-based book recommending using learning for text categorization," in *Proc. 5th ACM Conf. Digit. Libraries*, Jun. 2000, pp. 195–204.

[18] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, Jan./Feb. 2003.

[19] M. Deshpande and G. Karypis, "Item-based top-N recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, 2004.

[20] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Comput.*, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[21] H.-J. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 3203–3209.

[22] H. McMahan et al., "Ad click prediction: A view from the trenches," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 1222–1230.

[23] S. Rendle, "Factorization machines," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2010, pp. 995–1000.

[24] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin, "Field-aware factorization machines for CTR prediction," in *Proc. 10th ACM Conf. Recommender Syst.*, Sep. 2016, pp. 43–50.

[25] O. Chapelle and L. Li, "An empirical evaluation of Thompson sampling," in *Proc. NIPS*, 2011, pp. 2249–2257.

[26] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. 19th Int. Conf. World wide web*, Apr. 2010, pp. 661–670.

[27] C. Zeng, Q. Wang, S. Mokhtari, and T. Li, "Online context-aware recommendation with time varying multi-armed bandit," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 2025–2034.
[28] W. Zhang, T. Du, and J. Wang, "Deep learning over multi-field categorical data—A case study on user response prediction," in *Proc. Eur. Conf. Inform. Retr. (ECIR)*, 2016, pp. 45–57.
[29] Y. Qu et al., "Product-based neural networks for user response prediction," in *Proc. Int. Conf. Data Min. (ICDM)*, 2016, pp. 1149–1154.
[30] H. Cheng et al., "Wide & deep learning for recommender systems," in *Proc. 1st Workshop Deep Learn. Recommender Syst.*, 2016, pp. 7–10.
[31] Z. Huang, X. Xu, H. Zhu, and M. Zhou, "An efficient group recommendation model with multiattention-based neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4461–4474, Nov. 2020.
[32] G. Shani, D. Heckerman, and R. I. Brafman, "An MDP-based recommender system," *J. Mach. Learn. Res.*, vol. 6, pp. 1265–1295, Sep. 2005.
[33] N. Taghipour and A. Kardan, "A hybrid web recommender system based on Q-learning," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2008, pp. 1164–1168.
[34] X. Bai, J. Guan, and H. Wang, "A model-based reinforcement learning with adversarial training for online recommendation," in *Proc. NIPS*, 2019, pp. 10735–10746.
[35] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin, "Recommendations with negative feedback via pairwise deep reinforcement learning," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 1040–1048.
[36] X. Xin, A. Karatzoglou, I. Arapakis, and J. M. Jose, "Self-supervised reinforcement learning for recommender systems," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jul. 2020, pp. 931–940.
[37] G. Zheng et al., "DRN: A deep reinforcement learning framework for news recommendation," in *Proc. World Wide Web Conf. (WWW)*, 2018, pp. 167–176.
[38] Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement learning to rank in E-commerce search engine: Formalization, analysis, and application," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 368–377.
[39] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, and J. Tang, "Deep reinforcement learning for page-wise recommendations," in *Proc. 12th ACM Conf. Recommender Syst.*, Sep. 2018, pp. 95–103.
[40] F. Liu et al., "Deep reinforcement learning based recommendation with explicit user-item interactions modeling," 2018, *arXiv: 1810.12027*.
[41] H. Chen et al., "Large-scale interactive recommendation with tree-structured policy gradient," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 3312–3320.
[42] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2016.
[43] L. Zou, L. Xia, Z. Ding, J. Song, W. Liu, and D. Yin, "Reinforcement learning to optimize long-term user engagement in recommender systems," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2810–2818.
[44] X. Chen, C. Huang, L. Yao, X. Wang, W. Liu, and W. Zhang, "Knowledge-guided deep reinforcement learning for interactive recommendation," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8.
[45] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2014, pp. 387–395.
[46] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[47] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[48] A. Vaswani et al., "Attention is all you need," in *Proc. NIPS*, 2017, pp. 1–11.
[49] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.
[50] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "DeepFM: A factorization-machine based neural network for CTR prediction," 2017, *arXiv:1703.04247*.
[51] R. Xie, C. Ling, Y. Wang, R. Wang, F. Xia, and L. Lin, "Deep feedback network for recommendation," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 2519–2525.

Xing Wang received the B.Sc. degree from Northwestern Polytechnical University, Xi'an, China, in 2017. He is currently pursuing the Ph.D. degree with Tsinghua University, Beijing, China.

His current research interests include intelligent optimization on complex scheduling problems.

Ling Wang (Member, IEEE) received the B.Sc. degree in automation and the Ph.D. degree in control theory and control engineering from Tsinghua University, Beijing, China, in 1995 and 1999, respectively.

Since 1999, he has been with the Department of Automation, Tsinghua University, where he became a Full Professor in 2008. He has authored five academic books and more than 300 refereed papers. His current research interests include computational intelligence-based optimization and scheduling. He was a recipient of the National Natural Science Fund for Distinguished Young Scholars of China, the National Natural Science Award (Second Place) in 2014, the Science and Technology Award of Beijing City in 2008, and the Natural Science Award (First Place in 2003 and Second Place in 2007) nominated by the Ministry of Education of China. He is the Editor-in-Chief of the *International Journal of Automation and Control* and an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and *Swarm and Evolutionary Computation*.

Chenxin Dong received the bachelor's degree in vehicle engineering from China Agricultural University in 2010 and the M.E.M. degree from the China University of Petroleum in 2020. In 2017, he was at Kettering University for one year. Currently, he is a Lecturer with the Qingdao Hengxing University of Science and Technology, China. His current research interests include optimization and automotive engineering.

Hao Ren received the B.Sc. and M.S. degrees in industrial engineering from Tianjin University, Tianjin, China, in 2016 and 2019, respectively.

He is an Algorithm Engineer with Meituan. His current research interests include application of machine learning in online food delivery.

Ke Xing received the B.Sc. degree in computer science from Harbin Engineering University, Harbin, China.

He is an Algorithm Engineer with Meituan. His current research interests include the application of machine learning, spatial data mining, and ubiquitous computing.