Concurrent Order Dispatch for Instant Delivery with Time-Constrained Actor-Critic Reinforcement Learning

Baoshen Guo*, Shuai Wang*†, Yi Ding‡§, Guang Wang¶, Suining He‖, Desheng Zhang¶, Tian He*
*Southeast University, ‡Alibaba Group, §University of Minnesota, ¶Rutgers University, ‖University of Connecticut
{guobaoshen, shuaiwang, tianhe}@seu.edu.cn, dingx447@umn.edu,
suining.he@uconn.edu, {gw255, desheng}@cs.rutgers.edu

2021 IEEE Real-Time Systems Symposium (RTSS) | 978-1-6654-2802-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/RTSS52674.2021.00026

***Abstract*—Instant delivery has developed rapidly in recent years and significantly changed the lifestyle of people due to its timeliness and convenience. In instant delivery, the order dispatch process is *concurrent*. Couriers take new orders continuously and deliver multiple orders in a delivery trip (i.e., a batch). The delivery time of orders in a batch is often overlapped and interlinked with each other. The pickup and delivery sequence of the existing orders in a batch changes dynamically due to time constraints and real-time overdue possibility (i.e., the rate of deliveries that are not finished in promised time). Most of existing order dispatch mechanisms are designed for independent order dispatch or concurrent delivery without strict time constraints, hence are incapable of handling real-time concurrent dispatch with strict time constraints in on-demand instant delivery. To address the challenge, we propose a Time-Constrained Actor-Critic Reinforcement learning based concurrent dispatch system called TCAC-Dispatch to enhance the long-term overall revenue and reduce the overdue rate. Specifically, we design a deep matching network (DMN) with a variable action space, which integrates the state embedding (including route behaviors encoding) and actions embedding features into a long-term matching value. Then the Actor-Critic model tackles the concurrent order dispatch problem considering strict time constraints and stochastic demand-supply in instant delivery. An estimated time-based action pruning module is designed to ensure time constraints guarantee and accelerate the training as well as dispatching processes. We evaluate the TCAC-Dispatch with one-month data involved with 36.48 million orders and 42,000 couriers collected from one of the largest instant delivery companies in China, i.e., Eleme. Empirical experiments are conducted on a data-driven emulator deployed on the development environment of Eleme and results show that our method achieves 22% of the increase in total revenue and reduces the overdue rate by 21.6%.**

***Index Terms*—Instant Delivery, Concurrent Order Dispatching, Reinforcement Learning**

I. Introduction

Thanks to the rapid development of online digital platforms, instant delivery services (e.g., Instacart [1], Uber Eats [2], and Eleme [3]) have become a popular choice for people to order food, medicine, and groceries online, especially after the impact of COVID-19. In the third quarter of 2019, the number of instant delivery service customers in China reaches 470 million, and the market scale reaches 11.8 billion USD dollars [4]. In instant delivery, the delivery process needs to be finished in a short time [3] because packages to be delivered contain food or other urgent items. For example, Amazon Prime Now promises one-hour delivery for some orders [5]. Instant delivery platforms provide a big discount or even make the order free if the courier fails to deliver the order within the promised time window [3]. Given such a conflict between the massive number of orders and the limited couriers, it is essential to improve the efficiency of the dispatching system by solving *concurrent order dispatch* (i.e., instant delivery platforms assign orders to couriers) in a timely fashion.

Many studies have been conducted on real-time applications and systems such as real-time order dispatch and fleet management in ride-sharing [6]–[9] and real-time scheduling for electric vehicles [10]–[13]. For example, some studies solve the independent order dispatch problem in ride-sharing systems with reinforcement learning to enhance the profitability [14]–[17]. Compared to existing studies, however, there are two unique characteristics in the order dispatch problem for instant delivery. (1) *Overlapping Delivery Time due to Concurrent Dispatch*: Order dispatch and delivery process in instant delivery are *concurrent*, i.e., couriers carry multiple orders simultaneously in a trip, and the delivery time of these orders is overlapped especially in rush hours, while most existing studies focus on independent order dispatch; (2) *Delivery Time Constraints*: Instant delivery imposes a strict delivery deadline for each order, whereas the time constraints in traditional logistics and online retail are relatively loose. So the dispatch decision should be appropriate to avoid order overdue. Concurrent order dispatch in instant delivery should consider *time constraints* of orders and *time constraints relations* in subsequent dispatching.

Concurrent order dispatch decisions in the instant delivery are sequential and highly repetitive, thus generating massive historical dispatch and route records, which gives us a new opportunity to train reinforcement learning based methods to learn optimal dispatching decisions and improve efficiency. However, to seek an optimal concurrent order dispatch policy from historical decision data is not straightforward due to two challenges: (i) conflicts between enhancing the future overall revenue of concurrent orders and reducing the overdue rate of existing orders; (ii) instant delivery imposes a strict order *delivery deadline* and the delivery time of orders in a delivery trip is overlapped. To complete a concurrent delivery process, couriers will match orders “on the fly” many times. When we dispatch a new order to a courier, the delivery sequence and delivery time of orders that the courier already has changed dynamically, which may make existing orders overdue. In addition, future revenues and impacts in subsequent dispatching are also critical for concurrent dispatch. An urgent or long-

† Shuai Wang is the corresponding author.

distance order leaves couriers little time to take other orders.

To conquer these challenges, we model concurrent order dispatch as a sequential decision-making problem and propose a Time-Constrained Actor-Critic (**TCAC** for short) based order dispatch system. Specifically, we have the following *novel* designs in TCAC. (i) We design a Deep Matching Network (DMN) with a variable action space because when conducting the order-courier matching in instant delivery, the action space for different agents (i.e., couriers) in the different time slot is not fixed in advance, and the number of actions is based on real-time order locations, courier locations and number of couriers nearby, whereas traditional multi-agent actor-critic reinforcement learning algorithms have either a discrete action space (as output) or a continuous action space with a given range. Both of them are fixed in advance. With the variable action space, the DMN integrates time constraints of orders, couriers' real-time status, dynamic demand and supply, and other real-time contextual information into a long-term matching value. We construct the matching graph with the value and then conduct online order dispatch. (ii) Considering the overlapping delivery time of existing orders and future orders, we design a time-constrained action pruning module to prune dispatch actions that could lead to overdue and accelerate both the training and dispatching processes. In summary, the key contributions of our work are as follows.

- In particular, by modeling concurrent dispatch as a sequential decision problem with strict time constraints, we design **TCAC-Dispatch**, a Time-Constrained Actor-Critic based order Dispatch system. TCAC-Dispatch learns an efficient concurrent dispatch policy considering the spatial-temporal distribution of supplies and demands, orders' delivery deadline, couriers' uncertain route behaviors, and contextual factors in the offline learning phase. In the online dispatching phase, TCAC accounts for real-time environmental information and determines appropriate dispatch decisions between couriers and orders with strict time constraints.
- We design a time-constrained Actor-Critic based dispatch algorithm, which integrates a Deep Matching Network (DMN) and a value network (Critic). (i) The DMN has a variable action space considering the nature of concurrent dispatch. (ii) The DMN takes both state embedding features (integrates dynamic environmental factors and courier's route behaviors embedding) and vectorized action features (order features and order-courier matching features) as input and outputs the long-term matching score. (iii) A time-constrained action pruning module is designed to satisfy practical time constraints and to filter the actions to speed up training and improve the dispatching efficiency.
- To train the Deep Matching Network and evaluate TCAC-Dispatch, we design a data-driven emulator to emulate real-time concurrent order dispatch activities and deploy it on the development environment of the Eleme platform. Then we evaluate our TCAC-Dispatch with one-month data involved with 36.48 million orders and 42,000 couriers collected from Eleme. The experimental results show that TCAC-Dispatch achieves a 22% increase in total revenue and reduces the overdue rate by 21.6%. In addition, we rigorously evaluate the effectiveness of the time-constrained actor-critic module and present both the training and dispatching time cost comparison in Sec. VI-C.

II. Motivation

A. Characteristics of Concurrent Dispatch

As an Online-platform-to-Offline-delivery (O2O) service, the paradigm of instant delivery includes the following steps [3]. (1) Customers place orders online on the platform with mobile phones; (2) Merchants receive orders online from the platform and start to prepare these orders; (3) The platform dispatches orders to appropriate couriers; (4) Couriers pick up and deliver the orders to customers [18]. After customers receive orders, couriers, merchants, and the platform obtain corresponding revenue.

There are two *unique characteristics* in instant delivery systems. (1) *Concurrent order dispatch and dynamic delivery sequence:* Instant delivery involves multiple destinations in a delivery batch, which is much more complicated than delivering orders one by one independently, and hence calls for an efficient framework to optimize the concurrent dispatching and sequencing problem. In instant delivery services, couriers take multiple orders in one delivery trip of which the delivery time is overlapped, and new orders may arrive during a delivery trip. Taking a new order on the way, and other real-time changes in spatial-temporal information require dynamic updates of delivery sequence "on the fly". (2) *Strict delivery time requirements:* While customers of ride-sharing systems are more tolerant about arrival time [18], customers of instant delivery expect delivery in the promised time window [19].

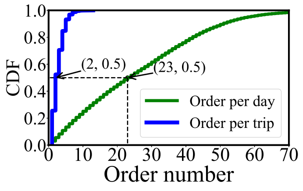

Fig. 1. Delivery number

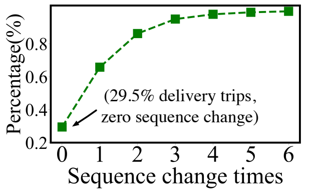

Fig. 2. Delivery sequence

Particularly, we conduct a detailed analysis of 42,000 couriers and 36.48 million order records from 2019/09/01 to 2019/09/30 in Shanghai. As shown in Fig. 1, the median of the number of orders that one courier served per day is 23, and the median of the number of orders per trip is 2. Fig. 2 illustrates the cumulative distribution function of

TABLE I
An Example of Delivery Sequence Changes

Timestamp	**Order Set**	**New Order**	**Delivery Sequence**
12:01	$\varnothing$	A	$\{A_o, A_d\}$
12:05	A	B	$\{B_o, B_d, A_d\}$
12:11	A, B	C	$\{C_o, B_d, A_d, C_d\}$
12:25	A, B, C	$\varnothing$	$\{B_d, A_d, C_d\}$

delivery sequence change times per trip. We observe that less than 30% of order delivery sequences do not change, while the delivery sequences of some couriers may change as many as six times in one trip due to new orders. Given the similar problem setting, delivery sequences of passengers in real-time ride-sharing hardly change considering passengers' willingness and potential complaints.

We further analyze the concurrency of order dispatch and changes of delivery sequence with an example. As shown in Table. I, the courier takes order A at 12:01. Then she/he picks up order A at 12:05 and takes new order B, the sequence changes to $\{B_o, B_d, A_d\}$. The courier delivers B firstly in this example. It is because that order B is an order on the way or order B has a more urgent due time. At 12:11, the courier picks up order B and takes new order C, the sequence changes to $\{C_o, B_d, A_d, C_d\}$. After that, the sequence does not change since the courier does not take new orders on this trip. While dynamic delivery sequence and strict delivery time constraints make concurrent order dispatch more challenging, they also bring the potential to dramatically improve the user experience (e.g., less delivery time) and courier experience (e.g., more profit).

B. Order Delivery Time Coupling

Upon identifying the unique challenges for order dispatch in instant delivery, we analyze and present the coupling delivery time and conflicting relationships between the impact of existing orders and future revenue. The delivery time of orders in the same delivery trip is coupling and interacts with each other due to concurrency and overlapped delivery time of orders. When dispatching a new order to a courier, the impact is as follows: (1) the impact on existing orders: the current order dispatch may increase the delivery time of existing orders and even make them overdue. In addition, existing orders influence the dispatch decision because it is a good decision to dispatch the order to the courier if there is a high degree of path coincidence between the new order and existing orders; (2) the impact on future revenue, that is, if an urgent order or a long-distance order is received, due to constraints of delivery time, there will be less time for the courier to take future orders. Therefore, the current dispatch will have an impact on possible future revenues.

III. System and Formulation

A. System Framework

Fig. 3 overviews the system framework of TCAC-Dispatch, which consists of two phases: *offline learning* and *online dispatching*. The offline learning phase has three components: the data-driven emulator for instant delivery, action features and state extraction, and TCAC model. (1) The *emulator* provides emulated environment derived from real-world order tracking data and courier trajectory data for model training. The emulator for instant delivery also accounts for time prediction and route prediction, which is detailed in Sec. V-B. (2) Given real-world order data, courier data, and other contextual factors from environments, *action features and state*

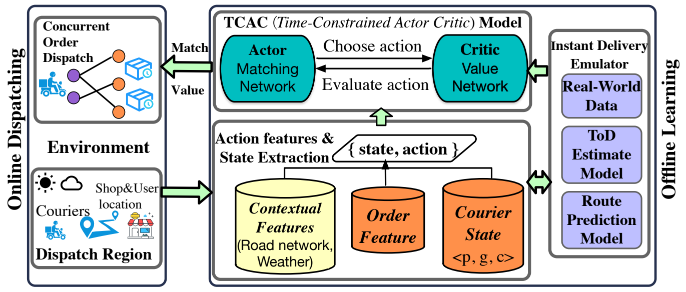

Fig. 3. System architecture of TCAC

extraction module is utilized to extract couriers' states and orders' features for ease of following model learning. (3) The *TCAC model* captures the dynamic spatial-temporal features, courier's state, and order-courier matching features at each time step to train the RL model. TCAC consists of two networks: *actor* and *critic*. We apply a deep matching network as the actor, which aims to learn a policy to choose the best action for each courier. The critic is a value network to evaluate dispatch actions, which also helps achieve a more stable and efficient model learning process.

In the online order dispatching phase, in each time step, we collect the information of new orders and couriers nearby in the same dispatch region and then calculate the long-term expected revenue value Q with the TCAC model. Then we match couriers and orders with Q value continuously. Considering the complex real-time factors and the concurrent features in instant delivery, the state of the courier updates dynamically and the order delivery sequence also changes dynamically after the courier takes a new order. After finishing order dispatch and delivery, we calculate the total revenue of platforms and the overdue rate of orders.

B. Problem Formulation

We model the concurrent order dispatch problem in instant delivery as a sequential decision task and then tackle it with deep reinforcement learning (DRL). The first goal of concurrent order dispatch in our problem is to *maximize the total revenues of orders per day*. In addition, to promote user experience and improve the service quality, couriers should finish delivering within the ToD (i.e., the promised delivery time of one order), otherwise, the order is overdue. Hence, another goal for our system is to *reduce the overdue rate by assigning orders to couriers appropriately.*

Then we formulate the concurrent order dispatch in instant delivery as a Markov decision process. Couriers choose actions to take orders at each time step and get rewards continuously. The state of a courier will change when she/he takes an action, i.e., the route plan and the total delivery time will change correspondingly after a courier takes a new order. Formally, this problem is characterized by four major components: $\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \pi\}$. π is the policy to make action-choosing decisions and we will discuss it in the model design section. The $\{\mathcal{S}, \mathcal{A}, \mathcal{R}\}$ are listed as follows.

State $\mathcal{S}$: The state of an agent is defined as $s_t = \{p_t, g_t, c_t\}$ and the state s_t^k of the courier k at time step t is defined as

$s_t^k = \{p_t^k, g_t^k, c_t^k\}$. where p_t^k is personal state of courier k and g_t^k is the global demand and supply information that the agent observes from a dispatch region. In addition, since traffic status (e.g., congestion or not), weather conditions (e.g.,rainy or sunny), and day of the week (e.g., weekend or weekday) affect the delivery behaviors of the courier, we form them into a vector as contextual state and utilize c_t^k to represent the spatial-temporal features at time step t.

- p_t: p_t is the personal state information and is defined as $p_t = \{loc_t, t, n_c, DT, route\}$. loc_t is the real-time location of the courier. t is current time step. n_c is the current capacity of the courier. DT is the total delivery time of existing orders of this courier. $route$ is the route plan of all stops (i.e., orders' origin and destination) of existing orders. To model courier's route behavior, we utilize a GRU network [20] to encode courier's route sequences into a vector

$$\boldsymbol{route_j} = \boldsymbol{GRU}(\boldsymbol{s_j}, \boldsymbol{route_{j-1}}) \tag{1}$$

  where $\boldsymbol{route_j}$ is the embedding vector of total j stops produced by the GRU network, $\boldsymbol{s_j}$ is the vector of j-th stop, and $\boldsymbol{route_{j-1}}$ is the hidden vector of first $j-1$ stops.
- g_t: g_t is the global demand and supply distribution observed from the environment at time step t. g_t consists of V_t and D_t. Two matrices capture the distribution of couriers and order requests at time step t, which characterizes the supplies and demands at a certain spatial-temporal point.
- c_t: contextual spatial-temporal features at time step t. c_t consists of weather conditions (e.g., sunny, cloudy, fog, haze, heavy-rain, light-rain, partly-cloudy) and day of the week as well as real-time traffic status. We process these contextual spatial-temporal features through one-hot encoding and form them into a vector as the additional hints for model training.

Note that the g_t and c_t are updated at the beginning of each time slot, whereas the personal state information p_t^k is updated in real-time when we dispatch a new order to courier k.

Action $\mathcal{A}$: An action means one matching between a courier and an order. In instant delivery, the number of couriers around each order is different, so the action space is variable due to real-time order locations, courier locations, and the number of couriers nearby. To address the variable action space and concurrent dispatch, we utilize a vector to represent action features. The action features consist of two parts: (1) Orders' features including the merchant location l_s , the user location l_d , price $\boldsymbol{rev_i}$, delivery fee $\boldsymbol{fee_i}$, and remaining time (i.e., the difference between the promised delivery time and current time). (2) Order-courier matching features including distance and time cost between the merchant and the current location of the courier, the increased delivery time T_{in} of this matching. The increased time is defined as $T_{in} = DT_{i+1} - DT_i$, where DT_i is the total delivery time of the courier's existing orders based on current route plan, and DT_{i+1} is the new total delivery time after he/she receives order i and adds this order to his/her route plan, which will be detailed in Sec. IV-A.

Reward $\mathcal{R}$: Given the state s_t^k of the courier k and the action a_t^i of the order i, the immediate reward for this match is defined as r_t^i. When an available courier takes and delivers an order, the courier receives the immediate reward r_t^i.

$$\boldsymbol{r_t^i} = \gamma^{\Delta t} \times \boldsymbol{rev}_i \times \frac{N_o}{N_c}, \tag{2}$$

where $\Delta t = \frac{T_{in}}{t_{step}}$ is the time step number of the increased delivery time when a courier adds the new order to her/his route plan. $\gamma \in [0,1]$ is the discount factor. $\gamma^{\Delta t}$ accounts for the influence of increased time. Increased delivery time serves as a penalty term here. The longer the increased time, the less the corresponding reward. When a new order is added to a courier's route, if the T_{in} is high, the reward will be discounted. We estimate the delivery time of order i when we try to dispatch it to courier k and compare the delivery time with ToD of this order, the reward is set as $r_t^i = 0$ if the order i is overdue. rev_i is the price of order i. N_o is the total number of orders in the destination grid of this action (e.g., picking up an order and arriving at the grid of merchant of this order), N_c is the total capacity of all couriers in the destination grid and denoted as $N_c = \sum n_c^k$. n_c^k is current capacity of courier k. $\frac{N_o}{N_c}$ captures the capacity supply and order demand, which helps conduct courier-order matching considering future orders distribution. The higher the $\frac{N_o}{N_c}$, the greater reward that this courier obtains when he/she takes action a_t^i, so that she/he will be more likely to be assigned to areas with higher demand.

State Transition: At the beginning of the next time step, we obtain the global demand and supply information g_{t+1} . The route plan will change when the courier takes a new order and the personal state p_t will also change correspondingly. Then we integrate $\{p_{t+1}, g_{t+1}, c_{t+1}\}$ to the next state s_{t+1}^k of courier k. Note that personal state p_t changes continuously since the courier takes new orders continuously and the order delivery sequence will also change continuously due to new orders. The global demand and supply g_t changes at the beginning of each time step.

IV. Concurrent Dispatch Model Design

In this paper, we design a time-constrained actor critic model to conduct concurrent order dispatch considering (i) time constraints and matching reward; (ii) possible overdue rate and long-term revenue. In this section, we first present an increased delivery time estimate model, which utilizes XGBoost Ranking [21] to predict couriers' uncertain route and estimate the increased time cost. Then the time estimate model eliminates the infeasible dispatch action by judging whether the dispatch action is time-constraint satisfying. Then we introduce details of the time-constrained actor-critic reinforcement learning model, which consists of a Deep Matching Network (Actor) and a state-value network (Critic). Lastly, we present the online order dispatching module in Sec. IV-E.

A. Time Cost Estimated and Constrained Action Pruning

We estimate the increased delivery time cost by predicting the possible new route of the courier when we try to dispatch an order to a courier. The goal of the time estimate module is to make sure that every exploration of dispatch action guarantees

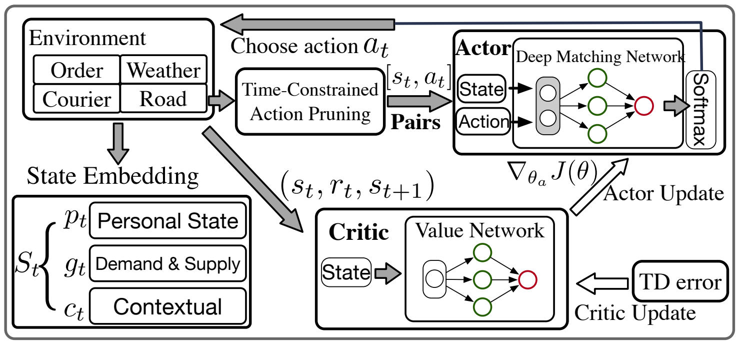

Fig. 4. Illustration of Time-Constrained Actor-Critic

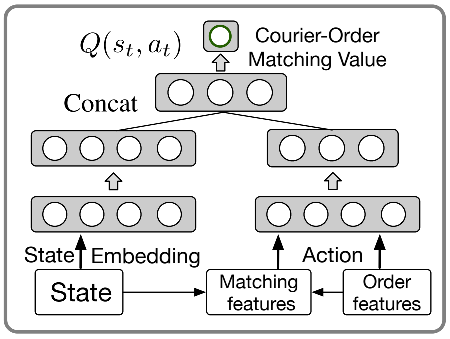

Fig. 5. Deep Matching Network

instant time constraints, which can accelerate offline learning and improve online dispatching of RL methods.

Estimated Delivery Time based on Route Prediction: In food delivery, the courier's route is the list of all stops, i.e., orders' origin (pickup stop) and destination (delivery stop). We generate couriers' routes by predicting the next stop one by one. Once the next-stop prediction is finished, we move the courier's location to the next stop and then predict the next stop. Particularly, we utilize the XGBoost ranking model [21] to predict couriers' route because (i) XGBoost can deal with massive data and be trained efficiently, which is suitable to process massive courier route data in instant delivery; (ii) XGBoost model is explicable and provides the importance of features which further help to choose features for our DRL model. The features of each stop item consist of the stop (an order origin or destination) location, the distance and time cost between the courier's current location and the candidate stop, and the remaining time to the promised time of this order. We feed them into the ranking model to obtain the possibility of each stop and choose the stop with the largest value in a greedy manner. When the courier's new route is generated, we obtain the increased delivery time T_{in} by calculating the difference of the total delivery time between the new route and the current route. Finally, we utilize the actual historical delivery route in the XGBoost ranking training phase.

Time-Constrained Dispatch Action Space Pruning: In instant delivery, algorithms of real-time order dispatch are required to be rather efficient (i.e., dispatch orders in a short time) and time-constraint satisfying (i.e., orders will not be overdue when being dispatched to couriers in the current situation). Our observation from real-world delivery data indicates that for an order, due to picking up time and waiting time constraints, only its nearby couriers can be dispatched. For each new order, we collect couriers nearby the order (i.e., usually within a few kilometers) and define C_t^i as the candidate courier-set of order i at time step t. Couriers in C_t^i are the possible ones who take order i. After estimating delivery time, we filtrate couriers who cannot guarantee the time constraints from C_t^i (the estimating delivery time is larger than the promised delivery time of the order), which safely prunes possible courier-order matching pairs and achieves the action space pruning for RL decisions. The time-constrained dispatch action space pruning module estimates the delivery time and prunes matching pairs that do not satisfy time constraints, which shrinks the action space of RL and improves the learning performance.

B. Deep Matching Network

Deep Matching Network is used to calculate the long-term revenue of the matching between couriers and orders. In reinforcement learning, we aim at maximizing the expected reward in each episode [22]. In our problem, we set one day as an episode. At each time step t , the expected discounted return is $\sum_{k=0}^{\infty} \gamma^k r_{t+k}^i$, where γ is the discount factor. For a new order i, we collect nearby couriers and add them to the candidate couriers set C_t^i of order i at time step t. Couriers in C_t^i are the possible ones who take order i. Traditional deep Q-learning network (DQN) takes the state as input and returns a vector of Q values whose dimension is equal to the dimension of fixed action space. In instant delivery, however, the number of orders around a courier is different from other moments. When we define serving an order as an action, we cannot guarantee the action space is consistent along with the whole episode [23]. Besides, in the concurrent order dispatch process, a courier takes multiple orders continuously and the delivery process of these orders is overlapped.

We design Deep Matching Network whose network structure is shown in Fig. 5. We utilize the pair $\langle state, action \rangle$ to represent the input of Deep Matching Network and the matching value as the single output. The deep model first converts the sparse input features of states into dense features, and then we feed state embedding and action vectors into feature embedding layers respectively. The embedding is to represent each category of high dimensional sparse features by a compact feature vector. Then we concatenate two features and get the matching value Q after feeding these features into the feed-forward hidden layers. Specifically, we define weight θ_a as the weight of the deep matching network, i.e., actor (policy) network. For each order, we get its candidate couriers set C_t^i and collect all $\langle state, action \rangle$ pairs. Then we run the Deep Matching Network forward to calculate the matching value $Q(s_t, a_t)$, which represents the possible long-term reward of this order dispatch. The larger the matching value, the more likely the courier will take this order.

C. Matching Policy (π)

After calculating all possible $Q(s,a)$ through Deep Matching Network, we utilize policy $\pi_{\theta_a}(s_t^k = c|a_t^i)$ to make order dispatch decisions. Given all possible $\langle state, action \rangle$ pairs, we calculate the matching value corresponding $Q(s_t^k, a_t^i)$. Then, the policy function $\pi_{\theta_a}(s_t^k = c|a_t^i)$ and is given by a softmax function [24]:

$$\pi_{\theta_a}(s_t^k = c|a_t^i) = \frac{exp(Q(s_t^k = c, a_t^i))}{\sum_{c'=c_1}^{C} exp(Q(s_t^k = c', a_t^i))}, \tag{3}$$

where s_t^k is the candidate courier for order i at time step t and θ_a as the weight of policy network(Actor). Since we aim to obtain an optimal policy to make order dispatch decisions, we train our model of the deep matching network and save the final weight θ_a, which is utilized to make decision in the order dispatch process.

D. Learning through Critic

The critic is another function approximator, which receives the state of agents as input and outputs the state value (V-value) for the given pair. As Fig. 4 shows, the critic is used to evaluate the action and give feedback to the actor. The state value function $V_{\theta_c}(s_t^k)$ is denoted as expected long-term value of courier k at time step t under state s_t^k. For each courier, at time step t, the expected long-term value in the remaining steps of an episode is defined as

$$V_{\theta_c}(s_t^k) = \mathbb{E}\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{n-t} r_{n-t+1}\big], \tag{4}$$

where $\gamma \in [0,1]$ is the discounted ratio of future rewards. θ_c collects the weights of value network (critic).

We update weights of the critic using time difference methods [25]. We dispatch orders to couriers sequentially and store the transitions $(s_t^i, r_{t+1}^i, s_{t+1}^i)$ to the experience memory $\mathcal{D}$ (i.e., store experience samples of agents) [26]. Then we sample some transitions for further training critic. The network parameters θ_c are updated by minimizing the following loss function, with respect to the transitions collected from all agents

$$\mathcal{L}_{\theta_c} = \frac{1}{2}\big[r_{t+1}^k + \gamma V_{\theta_c}(s_{t+1}^k) - V_{\theta_c}(s_t^k)\big]^2 \tag{5}$$

where θ_c includes parameters of value network. We utilize the state s_t^k and state s_{t+1}^k in next time step $t+1$ as the input of Critic Network. The output of the Critic Network is a scalar $V(s)$, which represents the long-term value when a courier is at state s_t after choosing action a. Then we evaluate all available state-order pairs. The update process of DMN is based on an advantage function

$$A(s_t^k, a_t^i) = r_{t+1}^k + \gamma V_{\theta_c}(s_{t+1}^k) - V_{\theta_c}(s_t^k). \tag{6}$$

where the advantage function is used to reduce the high variance of policy networks and stabilize the model. We utilize Deep Matching Network to try to choose a good dispatch action and the advantage function tells us how better the action is by a TD method [25]. With the advantage function, we define the gradient of actor by

$$\nabla_{\theta_a} J(\theta) = \nabla_{\theta_a} \log \pi_{\theta_a}(s_t^k, a_t^i) A(s_t^k, a_t^i), \tag{7}$$

where θ_a is the weight of actor model and $\pi_{\theta_a}(s_t^k, a_t^i)$ is the policy probability function. In the Offline Learning phase, we train the Deep Matching Network by sampling a batch historical transition $\{s_t^k, a_t^k, r_t^k, s_{t+1}^k\}$ from Replay Memory. Then, based on the gradient in Eq. (7), we update the actor's neural network parameters θ_a by the gradient descent rule as

$$\theta_a \leftarrow \theta_a + \alpha \nabla_{\theta_a} J(\theta), \tag{8}$$

where α is the learning rate. The detailed descriptions and learning process of TCAC are summarized in Algorithm 1.

ALGORITHM 1: Time-Constrained Actor-Critic for Concurrent Order Dispatch

```
Input: Historical order demand and courier supply,
       historical delivery process
1  Initialize value network(critic) $V$ with parameter $\theta_c$.
2  for $m = 0$ to maxIteration do
3      for $t = 0$ to $T$ do
4          for each order $i$ do
5              Pre-matching: Compute order similarity
                and search the candidate courier set.
6              Find optimal match (dispatch action) :
                Sample optimal state-action pair according
                to state-action matching probability
                computed by Equation 3, given all
                constraint candidate $< s_t, a_t >$ pairs.
7              Execute $a_t$: update courier's state base on
                his/her route plan.
8              Compute $V(s_t^k)$ and $A(s_t^k, a_t^k)$ and store
                the transitions $< s_t^k, a_t^k, r_t^k, s_{t+1}^k >$ to
                replay memory $\mathcal{D}$.
9      Updating parameters
10     for $j = 0$ to criticMaxUpdateStep do
11         Sample a batch of experience
            $< s_t^k, r_{t+1}^k, s_{t+1}^k >$ from $\mathcal{D}$
12         Update value network(critic) by minimizing the
            loss according to Equation 5
13     for $j = 0$ to actorMaxUpdateStep do
14         Sample a batch of experience
            $< s_t^k, a_t^k, r_t^k, s_{t+1}^k >$ from $\mathcal{D}$
15         Update policy network (actor) according to
            Equation 8
```

E. TCAC Online Dispatching

In the offline learning phase, we train the DMN through the TCAC framework. In the online dispatching phase, we make appropriate dispatch decisions based on the learned policy network to promote total revenue and reduce the overdue rate.

TCAC based Online Concurrent Order Dispatch: For each order i at a certain time step, we obtain the candidate courier set C_t^i, which consists of couriers around the order in the same region. Then, we formulate the order dispatch as a bipartite graph matching problem and utilize the *Kuhn and Munkres* (KM) Method [27] to solve it. In the online dispatching phase, the dispatch problem is represented as a directed graph. In this graph, there is a set U of order vertices u, a set C of courier vertices c, and a set E of edges $e = (u, c)$. Based on Deep Matching Network, we calculate the long-term value $Q(s, a)$ for each courier-order pair. Then we set the $Q(s, a)$ as the weights of edges. Suppose there are M orders and N couriers in a region at a certain time step, order dispatch is to find a proper match between the M orders and N couriers. For each iteration, the graph contains vertices of the first N placed orders and N couriers. Then, we aim to find the optimal match which is formulated by Eq. (9) and KM method.

$$\arg\max_{a_{ik}} = \sum_{i=0}^{N}\sum_{k=0}^{N} a_{ik} Q(s_t^k, a_t^i) \tag{9}$$

The $Q(s_t^k, a_t^i)$ is calculated through the Actor-Critic model. If courier k is in the search set C^i of order i, then we set $a_{ik} = 1$. It means that courier k is the candidate one who takes order i. Otherwise, we set $a_{ik} = 0$, then there is no edge between courier k and order i in the orders-couriers bipartite graph.

F. Training and Dispatching Computation Time Analysis

Considering massive orders of instant delivery services in rush hours and the strict delivery deadline constraint, we accelerate our TCAC through the following two aspects and leverage the time constraints as the opportunity: (i) In the offline learning phase of TCAC, we shrink the action space and constrain the solution space by time-constrained action pruning. The action pruning module ensures that every exploration of order dispatch action is efficient and time-constraint satisfying and thus reduces the training time of DRL algorithms. (ii) In the online dispatching phase, the time-constrained courier-order matching pairs pruning module can eliminate edges representing matching actions that could cause overdue orders. With the time constraints, we eliminate most of the edges in the matching graph described in Sec. IV-E to reduce the dispatching time. Sec. VI-D gives the computation time cost comparison. The results show that after the deep matching network and value network have been learned, dispatching a batch of orders to couriers at region-level takes less than 40 milliseconds, which is efficient to meet time requirements of the order dispatch in instant delivery services.

V. Implementation

We deploy our data-driven emulator in the real-world development environment of Eleme, one of the largest instant food delivery companies in China. In this section, we first introduce how we plug our emulator in the development environment and the overview of the platform. Then we detail our data-driven emulator for order dispatch.

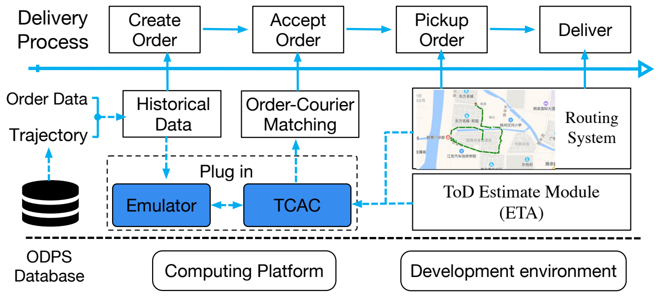

Fig. 6. Illustration of TCAC and Emulator deployment in development environment of an Instant Delivery Platform

A. Plug in System Development Environment

Before implementing TCAC in real-world production environment, we first train and test our model in development environment. The development environment has the same dataflow and interfaces with real-world production environment in an instant delivery company (e.g., Ele.me Inc.). Fig. 6 plots main components of the instant delivery system. In real-world delivery processes, customers, merchants, and couriers upload the real-time order status and trajectories to the platform and the platform stores these data in ODPS database [28]. We access these historical data in the development environment and conduct feature generation. Our emulator and TCAC-dispatch system perform as plug-in components. The data-Driven emulator emulates instant delivery scenarios with historical data and simulates the real-time situation of order dispatch. TCAC system makes dispatch decisions and feeds courier-order matching results to the emulator. The emulator and the TCAC system are realized in Python 3.7 and deployed in the computing platform.

B. Data-Driven Emulator in Development Environment

We design a systematic emulator to emulate instant delivery scenarios with real-world datasets. We apply the emulator to train our TCAC-Dispatch model and evaluate the timely dispatch performance.

Extracting patterns and distributions from data. Particularly, we first extract patterns and distributions from the trace data and order record data and then apply them to the emulation framework. The processes of extracting patterns are as follows. (i) *Couriers' online/offline distribution and initial location distribution* are obtained from order tracking records and trajectories using a maximum likelihood estimation. Orders are generated from historical order records in the emulator. (ii) *Couriers' speed* is the average speed extracting from trajectories in the same road and time slot, which is also affected by contexts such as weather conditions and congestion. In instant delivery services, the main transport mode of couriers is the electric bike. So the delivery speed is affected by spatial-temporal factors and contexts and is less affected by the mode of transportation. (iii) *The number of couriers* in our emulator is time-variant according to online/offline distribution extracting from real-world data.

Contextual Information. Contexts such as weather conditions, traffic status, and day of the week are considered in two aspects: (i) Contexts are part of the agent's real-time state and input of our model (details in Sec. III-B); (ii) Contexts are implicitly considered in the time estimate module and routing module because weather conditions, traffic status, and day of the week can affect the speed of couriers.

Dispatch Region. As a local delivery service, order dispatch in instant delivery is conducted by regions. A dispatch region is a small part of a city, which is about a few square kilometers (e.g., 5 $km\times$ 5 km) because the delivery scope of instant delivery ranges from 3 to 5 km. A courier is usually responsible for one region so that she/he is familiar with the region and road networks. In real-world instant delivery systems, each city is discretized into several regions and each region is discretized into $L_{lon} \times L_{lat}$ rectangular grids. Each grid is about 100 $m\times$ 100 m and the shape of each grid is altered to reflect the existence of buildings, rivers, roads, etc. We count the real-time distribution of couriers and orders at grid-level to capture the fine-grained demand and supply distribution and the order dispatch is conducted at region-level.

Emulator timelines. As is shown at the top of Fig. 6, at each time step, the timeline of a new order is as follows: (1) Add new orders to the emulator. (2) Extract order features and get couriers' state including their route plan. (3) Conduct the order dispatch algorithm and return courier-order matching pairs. (4) Execute matching results to the emulator and update the state and route plan of each courier.

Revenue definition. The total revenue of the platform is the sum of all orders' revenue. The revenue for an order is defined as its price minus three items: (1) the payment to the merchant for production (e.g., food is produced by merchants instead of the platform, similar to UberEat), (2) the couriers' compensations for delivery (e.g., human resource), (3) the discount caused by potential delivery overdue. Because (1) and (2) are usually fixed after an order is placed by a customer, our dispatch algorithm is to minimize overdue time to maximize the total revenue of the platform.

VI. System Evaluation

A. Dataset Description

We evaluate our design based on the real-world dataset collected from Eleme and build a data-driven emulator. The details and implementation of the Emulator are introduced in Sec. V-B. (i) **Order tracking records:** The order tracking data-set includes 36.48 million orders involved with 42,000 couriers in Shanghai. The period of this dataset is from 2019.09.01 to 2019.09.30. Due to the agreement with Eleme company, we are able to show the evaluation results in a sub-region of Shanghai (i.e., $6km \times 6km$) including 110853 orders and 1054 couriers. The sub-region is further discretized into 60×60 grids and the grid size is the same as that of the real-world platform The primary fields and detailed information of the tracking order dataset are shown in Table.II. The order information consists of order id, price, delivery fee, key temporal information, and key spatial information. (ii) **Road network:** We obtain road networks from OpenStreetMap [29] in some regions of Shanghai including road information with road types and road lengths, unidirectional or not, and detailed location node information with latitude and longitude.

TABLE II
An example of Order Tracking dataset.

Spatial	Merchant Lat.	Merchant Long.	User Lat.	User Long.
	31.6043	114.4268	31.6243	114.4356
Temporal	Create Order	Accept Order	Pick up	Delivered
	2019/9/1 11:23	2019/9/1 11:25	2019/9/1 11:35	2019/9/1 12:14

B. Experimental Setup

1) Evaluation Configuration: We implement the proposed model and other baselines with TensorFlow 1.14, Python 3.6 environment and train these with 16GB memory and Tesla V100-SXM2 GPU. The default order dispatch mechanism is batch dispatch where orders are packed into batches and dispatched to couriers. The one-month data are divided into four weeks. We use the first three weeks' data to train the deep network and the last week's data as evaluation.

2) Parameters Setting: The hyper-parameters are set as follows. The hidden layers of both the DMN and the value network are three-layer networks (256, 128, 64). Each network layer of two networks has a ReLU activation function. The batch size of two networks is 256 and the size of experience replay memory is $1e + 5$. The optimizer is Adam Optimizer with a learning rate of 0.001. The size of experience replay memory is set to be 10^5 and the batch size is set to be 256. We use AdamOptimizer as the optimizer with a learning rate of 0.001. The discount factor $\gamma = 0.9$. We set 5 min as a time slot, so that one day is divided into 288 time slots.

3) Baselines:

- **SD2** is the Shortest Distance based Dispatch (SD2) method [30]. For each new order, it will be dispatched to the nearest courier without considering time requirements and other factors. By default, the dispatch mechanism of SD2 is sequential dispatch.
- **OSquare:** OSquare is a novel dispatch method under sequential dispatch mechanism in instant delivery [18]. For each new order, OSquare searches for couriers around this order and estimates these couriers' overdue rate and the increased journey distance with the route prediction. The order will be dispatched to the courier with the least overdue rate and the least increased journey distance.
- **Double DQN with action search (DDQN-as):** DDQN-as [23] is a Double-DQN with spatio-temporal action search based dispatch method from a single driver perspective in ride-sharing. This method utilizes a deep Q network to obtain the Q value of all feasible action sets (i.e., all possible order-driver matching pairs or particular trips to a driver). The reward function only considers the total fee of each trip.

TABLE III
Performance Comparison in Total Revenue and Overdue Rate

Method	Per day		Revenue per hour in midday		Revenue per hour in evening	
	Total Revenue	Overdue Rate	11:00-12:00	12:00-13:00	17:00-18:00	18:00-19:00
SD2 [30]	100.0% $\pm$ 0.6%	3.94% $\pm$ 0.09%	100.0%	100.0%	100.0%	100.0%
OSquare [18]	106.5% $\pm$ 0.8%	1.85% $\pm$ 0.12%	113.7%	104.1%	104.5%	105.7%
DDQN-as [23]	112.9% $\pm$ 0.8%	1.64%$\pm$ 0.09%	112.3%	106.5%	107.8%	108.8%
MPBM [31]	113.1% $\pm$ 1.1%	1.47% $\pm$ 0.16%	115.9%	114.8%	108.8%	108.2%
TCAC-Dispatch	**122.9% $\pm$ 0.9%**	**1.45% $\pm$ 0.18%**	**132.6%**	**115.3%**	**111.4%**	**122.1%**

- **Multi-Iteration Packing Based Matching (MPBM):** MPBM [31] is a constrained optimization-based order dispatch method considering order price. The order dispatch problem is represented as a directed graph. There is a set U of order vertices u, a set C of courier vertices c, and a set E of edges $e = (u, c)$. The weight of e is profit gain considering the price and the increased time calculated from route prediction. MPBM runs the KM method for several iterations to complete the order dispatch process.

4) Metric: The evaluation metrics are as follows.

- **Total revenue of platform:** Total revenue of the platform is the sum of orders' revenue minus compensations for overdue orders. We also calculate the percentage increase of total revenue of each algorithm comparing with SD2.
- **Overdue rate:** For N orders, overdue rate = $\frac{|\{O_i|e_i>t_i, i=1,\ldots,N\}|}{N}$ is defined as the number of overdue orders divided by the total number of orders, where O_i is the order i, t_i is the ToD of O_i and e_i is the actual delivery time of O_i.

C. Experimental Results

Main Performance. We first present the comparison in terms of the total revenue and overdue rate during a day. Table. III shows that our model is better than baseline models in both the total revenue and overdue rate. Our TCAC model achieves 22.9% of the increase in total revenue compared with shortest-distance based Order dispatch (SD2). Besides, our model reduces the overdue rate to 1.45%, which is lower than that of other methods. Then, we study the model performance in rush hours during one day. Table. III also shows performance comparison in four rush hours one day. The performance of TCAC is better than baselines in all these hours. From 11:00 to noon, TCAC achieves 32.6% of the increase in total revenue compared with SD2. It demonstrates that TCAC is suitable for high-concurrency order dispatch scenarios. Table. III shows that TCAC-Dispatch model greatly increases the revenue of instant delivery platforms and reduces the overdue rate given massive number of orders and limited couriers.

Impact of Time of Day on Total Revenue Performance. Fig. 7 illustrates the performance in maximizing total revenue in midday and evening peak. As shown in Fig. 7, the performance of TCAC-Dispatch and other baselines dispatch methods is about the same at nine o'clock because the total number of orders is limited and the couriers' capacity is

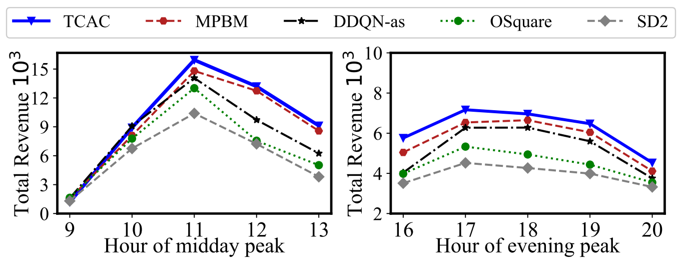

Fig. 7. Performance comparison in midday and evening peaks

enough in the systems. Then, as the number of orders increases in the following hours, the TCAC-Dispatch model always has better performance in revenue hourly. The curves in Fig. 7 illustrate that our model has the largest increase in overall revenue at noon compared with other models. The TCAC-Dispatch model has a better performance in peak hours because our model assigns abundant orders to limited couriers appropriately, which proves the effectiveness of the proposed TCAC model in maximizing the total revenue of platforms.

Ablation Study of Actor-Critic Component under Two Dispatch Mechanisms. We evaluate the effectiveness of the actor-critic module by comparing it with "TCAC-", OSquare, and SD2. "TCAC-" is the TCAC without the actor-critic module, which only includes a deep matching network and utilizes epsilon-greedy to choose actions. In real-world systems, to satisfy different scenarios and objectives, there are two dispatch mechanisms, i.e., batch order dispatch and sequential order dispatch. In batch dispatch, orders are packed into batches and dispatched to couriers. In the sequential order dispatch, we dispatch orders one by one sequentially. We evaluate the effectiveness of time-constrained actor-critic design by comparing with the aforementioned three baselines under two different dispatch mechanisms respectively. In each dispatch mechanism, TCAC leverages the actor to choose dispatch actions and uses critic to evaluate actions. OSquare and SD2 are sequential order dispatch algorithms mentioned in section VI-B3. "OSquare-b" and "SD2-b" are the versions of "OSquare" and "SD2" under the batch dispatch mechanism. Fig. 8 shows that TCAC has a better performance in reducing the overdue time of overdue orders in the batch dispatch mechanism. From Fig. 9, the performance is better in enhancing the total revenue when we choose TCAC as the order dispatch scheduler. TCAC increases 14.89 % of total revenue than "SD2-b" under the batch order dispatch mechanism.

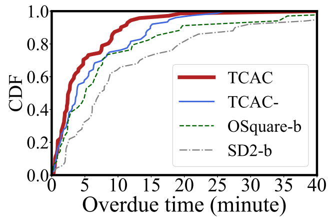

Fig. 8. Batch dispatch

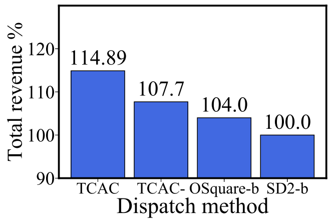

Fig. 9. Batch dispatch

From Fig. 10, we find that TCAC also has a better performance in enhancing total revenue and reducing the overdue time of overdue orders under both the sequential dispatch mechanisms. Fig. 11 shows that the total revenue of TCAC increases high to 22.9 % than SD2. The CDF of overdue time and the total revenue in both batch order dispatch and sequential order dispatch prove the effectiveness of the actor-critic module. In addition, combined with the main performance, we find that TCAC not only reduces the overdue rate of all orders but also reduces the overdue time of these overdue orders.

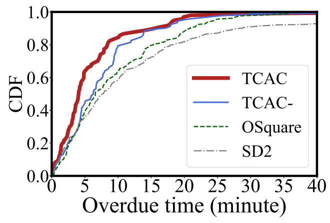

Fig. 10. Seq. dispatch

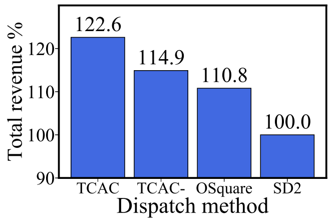

Fig. 11. Seq. dispatch

D. Convergence of TCAC and Computation Time Comparison

Convergence of TCAC: Fig. 12 demonstrates that the training of the value network (Critic) converges to a near-optimal solution after 3,000 steps of training.

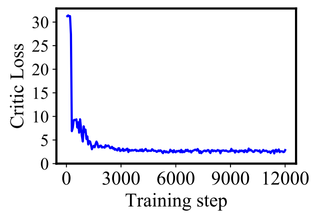

Fig. 12. Loss of critic module

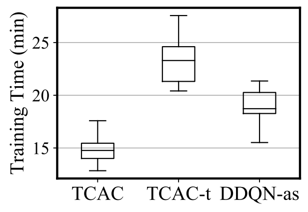

Fig. 13. Training time cost

Training Time Cost Comparison: To make the comparison of computation cost, we implement and train the proposed model, DDQN-as, and "TCAC-t" (i.e., "TCAC-t" is the TCAC without the time-constrained action pruning module) with 16GB memory and Tesla V100-SXM2 GPU. As is shown in Fig. 13, in the offline learning phase, we use average 15.3 minutes to train TCAC in our experiments, which is less than other deep reinforcement learning based baselines, i.e., "TCAC-t" (23.9 minutes) and DDQN-as (18.2 minutes).

Dispatching Time Cost Comparison: Fig. 14 shows that 64% of the accept duration of orders (i.e., the time duration between one order is created and is assigned) in real-world systems is less than 5 minutes. Fig. 15 shows that TCAC makes dispatch decisions for a batch of orders in each time slot within 40 milliseconds in the online dispatching phase, which is less than DDQN-as and MPBM and is faster enough for the real-time order dispatching requirements compared with the accepted duration in practical environments. Also, we find that "TCAC-t" has a longer dispatching time because of the lack of time-constrained dispatch action pruning module. Note that two heuristic baselines (SD2 and OSquare) are faster but their performances are far worse than the proposed method.

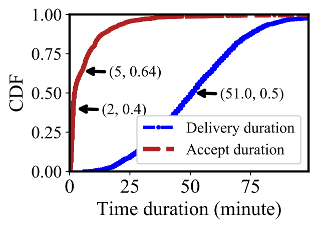

Fig. 14. Time duration in real-world

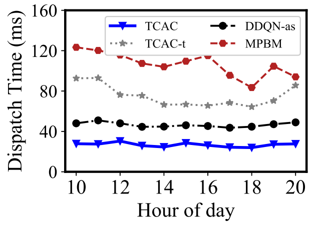

Fig. 15. Dispatching time cost

VII. Discussion

A. Insights and Lessons Learned

Concurrent order dispatch should consider the time-constrained dynamic route of couriers. Based on our data-driven analysis and evaluation results, we learned the following lessons. (i) Courier's route changes in real-time due to accepting new orders. An inappropriate dispatch decision will increase the detour time and delivery time of existing orders (even make them overdue). (ii) The route changes due to matching new orders also have an impact on future revenues. An appropriate order dispatch helps the courier match more similar orders "on the fly" to improve the efficiency of the courier and reduce average the delivery time of orders.

B. Practical Impacts

Although TCAC is not fully adopted in the dispatching process yet, we have achieved the following practical impacts: (i) It is a tentative step towards deep-learning-based dispatch from traditional operations-research-based (OR) dispatch, showing the potential obstacles; (ii) Compared to the traditional open-loop optimization-based solutions, potential time increase due to dispatching (which is strongly related to overdue rate) is used as the real-time feedback in a closed-loop fashion.

C. Implication to Real-time Cyber-Physical Systems

We study the time-constrained concurrent order dispatch for on-demand delivery to reduce the overdue rate and improve revenues. In other real-time cyber-physical systems, we could also find similar needs to dispatch and schedule appropriately to satisfy strict time constraints and improve efficiency and our TCAC has the potential to be generalized to these scenarios with minor changes. For example, drivers or couriers in ride-sharing and logistics systems need to pick up customers or packages in limited time windows. ambulances need to pick up and transport patients to hospitals as soon as possible.

D. Limitations

In designing, implementing, and deploying a deep-learning-based dispatch system, we met the following obstacles: (i) *Compatibility.* Given the compatibility requirements on a commercial platform, it is difficult to completely replace the existing system (e.g., OR-based dispatch) with a brand new system (DL-based), because massive work is needed to remold the lower-layer input modules (e.g., data collection) and the upper-layer application modules (e.g., software development kit (SDK) on courier APPs). (ii) *Across-City Generalizability*. In this paper, we study the instant delivery data from Shanghai, which may provide some obstacles when deploying our model to other cities. However, we believe this obstacle is manageable and our model has the potential to be generalized to other cities. This is because we did not use any city-specific model design and data collected in Shanghai are representative compared to other tier-1 cities in China.

E. Ethics, Consent, and Privacy

we have taken three steps for privacy protection: (i) *Anonymization:* All data analyzed is anonymized by service providers. All identifiable IDs, such as courier IDs, order tracking IDs, merchant IDs are replaced by serial identifiers; (ii) *Minimal Exposure:* We process data that are useful for our order dispatch project, including timestamps, locations, and GPS trajectory records. Then we drop other information for the minimal exposure; (iii) *Aggregation:* Our model analyzes the aggregated results and does not focus on an individual courier or a specific user. Hence, the learned model is less likely to reveal sensitive information about specific individuals.

F. Data Release for Reproducibility

To make our work reproducible and beneficial for the RTSS community, we share one-week data used in this paper for future research[1].

VIII. Related Work

Extensive studies have been conducted to optimize the real-time order dispatch process in transportation and O2O scenarios. We have witnessed unprecedented advances in applying reinforcement learning algorithms to complicated order dispatch in real-world systems [25], [26], [32]. Among existing order dispatch research, most of them aim to optimize independent order dispatch, while some others cooperatively optimize concurrent order dispatch. Based on this two dimensions, we divide order dispatch research into four different categories, which is shown in Table IV.

TABLE IV
CATEGORIES OF RELATED WORKS ON ORDER DISPATCH

Order Dispatch	Independent	Concurrent
Non-Learning based	[15]–[17], [33]	[18], [34]–[37]
Learning based	[7]–[9], [14]	TCAC(Our work)

[1]https://tianchi.aliyun.com/dataset/dataDetail?dataId=103969

A. Non-Learning based Methods

Independent Dispatch: Tong et al. [15] develop a prediction-based online dispatching algorithm to reduce the time complexity of task assignment. Some optimization-based schemes have been proposed for urban taxi dispatching with passenger-driver matching stability [33]. To reduce idle driving, Xie et al. [16] present a dispatching technique for vehicles based on a demand and supply model.

Concurrent Order Dispatch: There are some existing works for concurrent order dispatch. Xu and Chen et al. [38] formulated the dispatching problem as a non-linear multiple choice knapsack problem with KL-divergence as objective function, which is able to quickly dispatch agents to given spatial distributions. Steever et al. [34] provide a mixed-integer linear programming formulation about food delivery and develop an auction-based heuristic dispatch method. Yildiz et al. [35] model food delivery problem as a dynamic vehicle routing and pay attention to routing problems of couriers. But most of them are based on small-scale data and work on a simplified problem setting. Zhang et al. [18] study the route prediction in instant delivery and conduct order dispatch based on route prediction. A task grouping method [36] is proposed for O2O take-out food ordering and delivery services. They group delivery tasks into groups with high share-ability to improve food delivery efficiency.

B. Learning based Methods

Independent Dispatch: Reinforcement learning has been used in ride-sharing to efficiently assign orders to drivers one by one independently. Most of previous works [7]–[9] study fleet management and model the order dispatch in ride-sharing as a decision-making task considering both immediate and future rewards. Besides, Li et al. [39] apply reinforcement learning in express delivery to guide couriers to deliver and serve express orders. Oda et al. [40] propose a convolutional neural networks based system to relocate connected taxicabs. Jin et al. [14] propose a hierarchical multi-agent reinforcement learning solution to combine order dispatching and fleet management for ride-sharing. The delivery process in these studies is independent. These systems dispatch a new order to a driver when she/he finishes the previous order.

Concurrent order Dispatch: *Our TCAC is the first one in this category*. Different from existing works, this paper proposes a practical reinforcement learning based order dispatch framework for instant delivery. We conduct extensive data analysis and evaluation with a large-scale dataset. To our best knowledge, this is the first work that applies deep reinforcement learning to real-time concurrent order dispatch in instant delivery scenarios.

IX. Conclusion

In this work, we propose an actor-critic based order dispatch method for instant delivery, which aims at reducing the delivery overdue rate and maximizing the total revenue.

We first conduct an empirical data analysis based on real-world order tracking data and couriers' data, and then identify patterns and potential problems of current dispatch systems. Taking into account the characteristics of the real-time dispatch process, we propose the TCAC-Dispatch to promote the total revenue and reduce the overdue rate. We evaluate our method based on a real-world dataset from an instant delivery company called Eleme. The experimental results show that our method achieves a 22% increase in total revenue and reduces the overdue rate by 21.6% at the same time.

X. Acknowledgments

This work was supported in part by National Natural Science Foundation of China under Grant No. 61902066, Natural Science Foundation of Jiangsu Province under Grant No. BK20190336, China National Key R&D Program 2018YFB2100302 and Fundamental Research Funds for the Central Universities under Grant No. 2242021R41068, Key Laboratory of Computer Network and Information Integration, Ministry of Education.

References

[1] InstaCart, "Instacart," Webpage, 2021.
[2] Ubereats, "Ubereats," https://www.ubereats.com/hk, 2020.
[3] Ele.me, "Ele.me website." http://www.ele.me/, 2020.
[4] "Consulting statistics," https://www.chyxx.com/industry/202003/840068.html, 2020, online; accessed 29 January 2020.
[5] Amazon Prime Now, "Amazon prime now," https://primenow.amazon.com/, 2020.
[6] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, "Price-aware real-time ride-sharing at scale: an auction-based approach," *SIGSPATIAL/GIS*, pp. 3:1–3:10, 2016.
[7] S. He and K. G. Shin, "Spatio-temporal capsule-based reinforcement learning for mobility-on-demand network coordination," in *The World Wide Web Conference, WWW 2019, San Francisco, USA*, 2019, pp. 2806–2813.
[8] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye, "Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. ACM, 2018, pp. 905–913.
[9] L. Kaixiang, Z. Renyu, X. Zhe, and Z. Jiayu, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery*, ser. KDD '18, 2018, pp. 1774–1783.
[10] G. Wang, X. Xie, F. Zhang, Y. Liu, and D. Zhang, "bcharge - data-driven real-time charging scheduling for large-scale electric bus fleets," *RTSS*, pp. 45–55, 2018.
[11] G. He, Z. Chai, X. Lu, F. Kong, and B. Sheng, "Admm-based decentralized electric vehicle charging with trip duration limits," *RTSS*, pp. 107–119, 2019.
[12] F. Kong, Q. Xiang, L. Kong, and X. Liu, "On-line event-driven scheduling for electric vehicle charging via park-and-charge," *RTSS*, pp. 69–78, 2016.
[13] Z. Dong, C. Liu, Y. Li, J. Bao, Y. Gu, and T. He, "Rec: Predictable charging scheduling for electric taxi fleets," *RTSS*, pp. 287–296, 2017.
[14] J. Jin, M. Zhou, W. Zhang, M. Li, Z. Guo, Z. T. Qin, Y. Jiao, X. Tang, C. Wang, J. Wang, G. Wu, and J. Ye, "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, CIKM 2019, Beijing, China, November 3-7, 2019*. ACM, 2019, pp. 1983–1992.
[15] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *Proc. VLDB Endow.*, vol. 10, no. 11, p. 1334–1345, Aug. 2017.
[16] X. Xie, F. Zhang, and D. Zhang, "Privatehunt: Multi-source data-driven dispatching in for-hire vehicle systems." vol. 2, pp. 45:1–45:26, 2018.
[17] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. Ye, "A taxi order dispatch model based on combinatorial optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 2151–2159.
[18] Y. Zhang, Y. Liu, G. Li, Y. Ding, N. Chen, H. Zhang, T. He, and D. Zhang, "Route prediction for instant delivery," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 3, no. 3, Sep. 2019.
[19] Y. Ding, L. Liu, Y. Yang, Y. Liu, D. Zhang, and T. He, "From conception to retirement: a lifetime story of a 3-year-old wireless beacon system in the wild," in *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*, 2021, pp. 859–872.
[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 2067–2075.
[21] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.
[22] R. Sutton and A. Barto, *Reinforcement Learning:An Introduction*, 1998.
[23] Z. Wang, Z. Qin, X. Tang, J. Ye, and H. Zhu, "Deep reinforcement learning with knowledge transfer for online rides order dispatching," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 617–626.
[24] R. Salakhutdinov and E. G. Hinton, "Replicated softmax: an undirected topic model," *NIPS*, pp. 1607–1614, 2009.
[25] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov 2017.
[26] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, and A. Bolton, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359.
[27] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, no. 1-2, pp. 83–97.
[28] "Odps," https://cn.aliyun.com/product/odps, 2020, accessed 20 September 2020.
[29] "Open street map," https://www.openstreetmap.org, 2019, accessed 27 June 2019.
[30] J. McCann and R. Chatley, "Fleet management in on-demand transportation networks : Using a greedy approach," 2018.
[31] L. Zheng, L. Chen, and J. Ye, "Order dispatch in price-aware ridesharing," *Proc. VLDB Endow.*, vol. 11, no. 8, p. 853–865, Apr. 2018.
[32] S. Ji, Y. Zheng, Z. Wang, and T. Li, "A deep reinforcement learning-enabled dynamic redeployment system for mobile ambulances," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 3, no. 1, Mar. 2019.
[33] H. Zheng and J. Wu, "Online to offline business: Urban taxi dispatching with passenger-driver matching stability," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 816–825.
[34] Z. Steever, M. H. Karwan, and C. C. Murray, "Dynamic courier routing for a food delivery service." vol. 107, pp. 173–188, 2019.
[35] B. Yildiz and M. W. P. Savelsbergh, "Provably high-quality solutions for the meal delivery routing problem." vol. 53, pp. 1372–1388, 2019.
[36] 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 *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 773–783.
[37] X. Chen, S. Xu, J. Han, H. Fu, X. Pi, C. Joe-Wong, Y. Li, L. Zhang, H. Y. Noh, and P. Zhang, "Pas: Prediction-based actuation system for city-scale ridesharing vehicular mobile crowdsensing," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 3719–3734, 2020.
[38] S. Xu, X. Chen, X. Pi, C. Joe-Wong, P. Zhang, and H. Noh, "ilocus: Incentivizing vehicle mobility to optimize sensing distribution in crowd sensing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 08, pp. 1831–1847, aug 2020.
[39] Y. Li, Y. Zheng, and Q. Yang, "Efficient and effective express via contextual cooperative reinforcement learning," pp. 510–519, 2019.
[40] T. Oda and C. Joe-Wong, "Movi: A model-free approach to dynamic fleet management," in *IEEE International Conference on Computer Communications*, vol. abs/1804.04758, 2018, pp. 2708–2716.