Learning to Optimize the Dispatch Time Interval for On-Demand Food Delivery Service

Jingfeng Yang[1,2], Zhiqin Zhang[2], and Hoong Chuin Lau[2]

Abstract— **In recent years, the rapid advancement of mobile and wireless communication technologies has enabled real-time connectivity for on-demand delivery platforms, facilitating efficient door-to-door services like online food delivery. This study addresses a practical challenge faced by a food delivery platform, where customer orders must be allocated to drivers responsible for collecting food from designated centers and delivering it to customers within specific time windows. This dynamic pickup and delivery problem emphasizes prompt delivery as the critical objective. Our research focuses on optimizing the dispatch intervals for orders on such platforms. We tackle this by formulating the problem as a Markov decision process (MDP) and introducing a two-stage framework that combines a multi-agent reinforcement learning (RL) approach for order dispatching with a heuristic method for driver routing. The RL algorithm determines the optimal timing for each order's entry into the matching pool, while the routing method integrates orders into drivers' delivery routes. Extensive experiments, using real-world data and a simulator, show our results surpass benchmark methods, enhancing the efficiency of order dispatching in on-demand food delivery services.**

I. INTRODUCTION

Recent transformations in the logistics sector include the rise of on-demand service platforms like Uber Eats, DoorDash, Grab Food, and Meituan, which have revolutionized efficient door-to-door food delivery. The on-demand food delivery market consists of customers, restaurants, drivers, and the platform itself. Customers browse and order food from listed restaurants via the platform, which then alerts the restaurant to prepare the food and assigns drivers nearby to deliver the orders. The main challenge lies in effectively dispatching these orders to drivers, ensuring timely pickups and deliveries within strict delivery time constraints (e.g., 45 minutes). Conventional order dispatching strategies implemented in practical settings [1] typically involve fixed time intervals (e.g., every 10 minutes), within which the platform accumulates all incoming orders and consolidates them into a matching pool. Subsequently, the orders are collectively assigned to drivers for simultaneous delivery. However, recent investigations in the realm of ridesourcing [2], [3] have revealed that by extending the dispatching time interval, it is possible to optimize the allocation of drivers to passengers, leading to reduced wait times for both parties involved. Drawing inspiration from the efficacy of extended dispatching time intervals in ridesourcing, our research concentrates on the dynamic optimization of order dispatching time intervals for on-demand food delivery services. Specifically, our focus lies in determining the opportune moment for orders to enter the matching pool to initiate the dispatching process, given their arrival on the platform.

Figure 1 provides an illustrative example that demonstrates the impact of optimizing dispatch times at an order-level on both delivery distance and time. The figure highlights two distinct dispatching times, denoted as t and $t+1$. At $t+1$, three orders are received, with orders 1 and 2 arriving prior to t, and order 3 arriving within the time interval $(t, t+1]$. Initially, we consider dispatching using a fixed time interval approach, wherein all orders arriving in the interval $(t-1, t]$ enter the matching pool and are dispatched to drivers for delivery at time t. As per this strategy, orders 1 and 2 are assigned to drivers A and B respectively at t, while order 3 is dispatched to driver A at $t+1$, who is en route to deliver order 1. The delivery routes are depicted by black dotted lines with arrows. However, if we postpone the inclusion of order 1 in the matching pool until $t+1$, driver A can conveniently pick up both order 1 and 3 for delivery. This adjustment results in a reduced total travel distance, albeit at the expense of a lengthier waiting time for order 1.

Although implementing above postponement strategy at the order level can significantly reduce the total travel distance and decrease delivery times, it is crucial to acknowledge the potential trade-offs involved: the extended dispatching time may elicit impatience among customers, prompting them to cancel their orders. Thus, understanding the delicate balance between customer waiting time, driver travel distance, and order delivery time across various dispatching time intervals (or the number of intervals to wait before entering the matching pool) becomes crucial. Consequently, developing optimal dispatching strategies that yield superior system performance is of paramount importance. A promising approach to this entails dynamically determining the dispatch time for each customer order.

To address the aforementioned challenges, we propose in this paper a model that treats the dynamic order dispatching problem in on-demand food delivery services as a multi-agent Markov decision process. In our model, each customer order is considered as an independent agent with the ability to make decisions regarding its entry into the matching pool for dispatch. Subsequently, the platform employs a driver routing algorithm to optimize the insertion of orders into

*This work was supported by SIMTech, A*STAR

[1]Jingfeng Yang is with the Singapore Institute of Manufacturing Technology (SIMTech), Agency for Science, Technology and Research (A*STAR), Singapore 138634, Republic of Singapore Yang_Jingfeng@simtech.a-star.edu.sg

[2]Jingfeng Yang, Zhiqin Zhang and Hoong Chuin Lau was/are with School of Computing and Information Systems, Singapore Management University, Singapore 188065, Republic of Singapore jfyang.2018, zqzhang.2020@phdcs.smu.edu.sg, hclau@smu.edu.sg

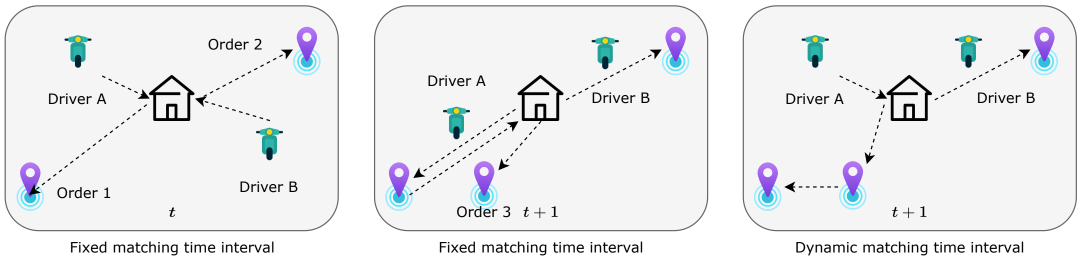

Order 2
Driver A
Order 1
Driver B
t
Fixed matching time interval
Driver A
Driver B
Order 3
$t+1$
Fixed matching time interval
Driver A
Driver B
$t+1$
Dynamic matching time interval

Fig. 1: Delivery distance/time reduction with dynamic dispatching time interval.

drivers' delivery routes. In summary, our study makes the following key contributions:

- We explore the dynamic order dispatching problem in on-demand food delivery services, formulating it as a multi-agent MDP. Specifically, we examine how each order determines its entry into the matching pool at subsequent dispatch times, followed by the implementation of the driver routing algorithm.
- We propose a two-stage optimization framework using learning techniques for dynamic order dispatching. The first stage employs an end-to-end reinforcement learning (RL) approach, utilizing a centralized multi-agent deep Q-network to streamline order dispatching and entry into the matching pool. In the second stage, we apply an efficient driver routing algorithm that considers driver behavior and supports order re-dispatch.
- We conduct extensive numerical experiments with a real-world dataset to evaluate our proposed framework's performance. The results demonstrate that our framework surpasses benchmark methods, effectively enhancing order dispatching efficiency in on-demand food delivery services.

Our research provides a comprehensive model and a learning-based optimization framework that significantly enhance the order dispatching process in on-demand food delivery services, leading to improved operational efficiency and customer satisfaction.

II. Related Works

Our review centers on recent studies pertinent to order dispatching and driver routing in the context of food delivery services. Specifically, this aspect of food delivery is defined as the Meal Delivery Routing Problem (MDRP) in Reyes et al. [4], representing a specialized form of the Dynamic Pickup and Delivery Problem (DPDP), which has garnered significant scholarly attention over the past decades, as noted by Psaraftis et al. [5]. In a recent study on MDRP, [6] formulate the problem assuming that the platform has perfect information about order arrivals and solve it using a combined column and row generation approach. [7] formulate the problem as a stochastic dynamic pickup and delivery problem using a route-based Markov decision process (MDP). An anticipatory customer assignment policy is proposed for order dispatching and vehicle routing. [8] aim to improve the working conditions of drivers by order dispatching algorithm design. A queuing-model-based algorithm is proposed with the objective of minimizing the waiting time of drivers while ensuring a good user experience. Two fast heuristic algorithms have been proposed and deployed on Meituan, the largest takeaway platform in China by [1] with the objective to maximize the order dispatched rate. However, many of the aforementioned studies employ conventional operations research techniques to formulate the problem as either a Mixed-integer programming problem or an MDP problem, subsequently applying deterministic (e.g., column generation) or stochastic algorithms with heuristics to find solutions.

Finally, we summarize the most relevant literature that focuses on optimizing the dispatching time of orders and couriers (vehicles). [2] proposed a reinforcement learning model to determine the delay time for each order, which aligns with our settings. However, their method is applicable only to ridesourcing platforms, whereas our model specifically caters to the food delivery service, considering constraints on order delivery time and driver routing decisions. Another related work by [9] also employs reinforcement learning to address the dynamic pickup and delivery problem (DPDP). They propose a hierarchical framework with an upper agent that dynamically partitions the DPDP into a series of subproblems to optimize vehicle routing. Similar to our model, they also take into account soft time window constraints and vehicle routing decisions. The main difference lies in the fact that in our model, each order is treated as an independent agent capable of deciding when to enter the matching pool, whereas they only permit the release of all accumulated orders at once. Furthermore, by allowing order re-dispatch, we have developed a novel heuristic-based driver routing algorithm for order pickups and deliveries, while they employ a reinforcement learning-based neighborhood search algorithm.

III. Problem Description

We are given a fleet of drivers $\mathcal{D} = \{d_1, d_2, \cdots, d_m, \cdots\}$ to deliver a set of orders $\mathcal{O} = \{o^1, o^2, ..., o^n, ...\}$ that arrive dynamically in the planning horizon $\mathcal{T} = \{1, 2, ..., T\}$. Each order $o \in \mathcal{O}$ can be represented as a tuple (o_t, o_+, o_-, p^o) that contains the arrival time, pickup and delivery locations and the promised delivery time. Each driver $d \in \mathcal{D}$ has a service capacity cap (i.e., the maximum number of orders that can be carried) and drivers are distributed around the

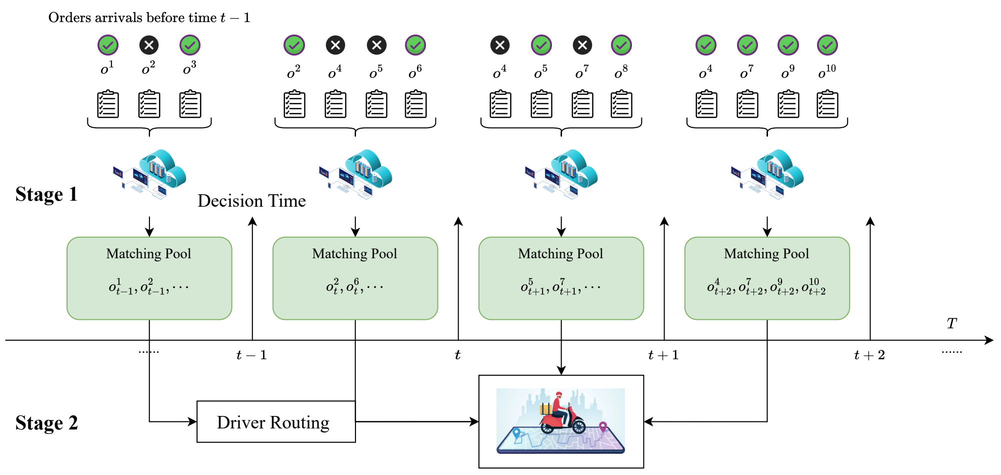


Fig. 2: A two stage optimization framework for on-demand food delivery service.

entire area. The platform must make two decisions: (1) when to release the orders to the matching pool; and (2) how to dispatch the orders in the matching pool for available drivers for delivery. More specifically, at each dispatch time $t \in \mathcal{T}$, the platform could determine which orders with $o_t \leq t$ can enter the matching pool, followed by specific order dispatching in batches. Those decisions are made sequentially over time until it reaches an ending time of T. An example of the proposed two-stage framework is demonstrated in Figure 2. As shown in the figure, the order dispatching decisions are made at time $t-1$, t, $t+1$ and $t+2$, e.g., every 5 minutes, and between consecutive decision times, we have a matching pool to hold all orders are allowed to dispatch to available drivers for pickup and delivery. For example, before time $t-1$, there are three orders, o^1, o^2, o^3 arrive in the platform and waiting for dispatching. The platform decide order o^1 and o^3 can enter the matching pool, while order o^2 should waiting for the next decision time at t. And there are three new orders (o^4, o^5, o^6) arriving between $t-1$ and t. Order o^4 is delayed again for dispatch and only order o^6 is allowed to enter the matching pool. All orders including order o^4 which arrive in the platform before time t are allowed for dispatch at time $t+2$. In this case, a total of four orders are delayed for dispatch, with orders o^2, o^5 and o^7 waited for one more time interval, while order o^4 waited two time intervals. Following the determination of dispatchable orders, the platform will allocates each order in the dispatch pool to available nearby drivers and subsequently updates the driver's route for pickup and delivery.

Here, we are interested in optimization from a platform's perspective, i.e. we aim to minimize the total cost with the delivery constraints. To be specific, as given in the objective function below, the platform aims to minimize the total travel distance while ensuring a service level of served orders as well as order delivery time.

$$\min \quad \sum_{d \in \mathcal{D}} c_d + \lambda \cdot \sum_{o \in \mathcal{O}} \max(0, f^o - p^o) + \mu \cdot \sum_{o \in \mathcal{O}} u^o \quad (1)$$

In this context, we use c_d to denote the total travel distance of driver d within the entire planning horizon $\mathcal{T}$. Furthermore, we define variable f^o to represent the time when customer o receives their order, and u^o as a binary variable with a value of 1 if order o is cancelled by the customer, and 0 otherwise. The first component computes the total travel distance for all drivers. We impose penalties for served orders with delays or orders cancelled by customers through the assignment of large values to parameters λ and μ, respectively.

IV. METHODOLOGY

A. MDP Formulation

In the context of our problem, it seems most logical to conceive of the platform as the decision-maker. The platform, acting as a single centralized agent, determines which orders are allowed to enter the matching pool. However, the nature of this modeling approach significantly complicates reinforcement learning tasks due to the dynamic nature of the actions on orders. In an on-demand food delivery service, the number of orders arriving in each dispatching time interval (e.g., $(t-1, t]$) varies over time. Correspondingly, we need to maintain a "dynamic" action space, $\mathcal{A}_t$. However, representing a dynamic action space in the learning algorithm can be challenging. Unlike standard RL algorithms which are typically designed for static action spaces, we need to take actions which are contingent on orders arriving dynamically. More precisely, given the set of orders waiting to enter the matching pool at time t is $O_t = \{o_t^j, o_t^{j+1}, \cdots, o_t^{j+k}\}$, at the decision step t, the action of the agent is to select a subset of orders a_t from all k orders. Accordingly, we define the action space $\mathcal{A}_t = \{a_t | a_t \in O_t\}$, which contains all possible subsets of orders. Note that the size of the combinatorial space $\mathcal{A}_t$ is 2^k, which grows exponentially with the number of accumulated orders k in time interval $(t-1, t]$. This limitation hinders the application of general reinforcement learning algorithms to large-scale problems, as these algorithms necessitate an explicit representation of all actions and exploration across a large action space. To accommodate a large action space while maintaining tractable learning, we factorize the combinatorial action space $\mathcal{A}_t$ into elementary actions along each order dimension. Specifically, we conceptualize each order as an individual agent. The elementary action for order i at decision time t is denoted as a_t^i. This implies that order i has two options: either enter the matching pool for dispatching a driver for delivery, or wait until the next decision time. Using this representation, any subset of orders can be expressed as $a_t = \cup_{i=j}^{i=j+k} a_t^i$. Consequently, our task transforms into learning policies for binary decisions corresponding to each order. This approach allows for the exploration of a large action space through the traversal of binary action spaces, which increase linearly with the number of orders. Our multi-agent MDP model is defined as follows:

Agent. We consider each customer order as an individual agent, endowed with the decision-making capability to opt for entry into the matching pool. Agent activation occurs upon receiving a customer request from a restaurant and is randomly initiated by the environment, in accordance with the underlying demand distribution. The count of agents in each time interval, represented as N_t for $t \in \mathcal{T}$, varies across different time intervals.

State. At each decision time $t \in \mathcal{T}$, the state $\mathbf{s}_t$ is defined. we distinguish between two types of states: the global states $s_{t,\text{global}}$ which provide comprehensive demand and supply information for the entire environment and are shared by all agents at dispatch time t, and the local states $s_{t,\text{local}}$ which are specific to each individual agent and vary among them at the same dispatch time. Assuming the study area is divided into $|\mathcal{M}|$ small hexagons, the global states in our model encapsulate four types of spatio-temporal supply-demand features: (1) the count of pending orders in the matching pool within each hexagon; (2) the available service capacity in each hexagon; (3) the origin-destination distribution of all pending orders within the study area; and (4) the pair of current and destination hexagons for each driver operating within the study area. Consequently, the dimension of the global state vector is $|\mathcal{M}| \times (2|\mathcal{M}| + 2)$. The local states, on the other hand, comprise three elements: (1) the origin (restaurant) and destination (customer's location) of each order; (2) the estimated increase in travel distance for dispatched drivers, as calculated by the routing algorithm; and (3) the time elapsed since the customer placed the order. It is evident that the local-view state varies among agents, each characterized by a dimensionality of $2|\mathcal{M}| + 2$. Therefore, the state for agent i at the decision time t can be expressed as $s_t^i = [s_{t,\text{global}}, s_{t,\text{local}}] \in \mathbb{R}^{(|\mathcal{M}|+1)\times(2|\mathcal{M}|+2)}$.

Action. At each decision time t, every agent i possesses two feasible actions represented as $a_t^i = \{0, 1\}$. Specifically, $a_t^i = 1$ signifies that agent i opts to join the matching pool, thereby becoming eligible for dispatch to a driver. Conversely, $a_t^i = 0$ indicates that agent i chooses not to participate in the matching pool and instead awaits the subsequent matching time $t + 1$.

Reward. The reward function is specifically designed to ensure alignment with the established optimization objective function, which aims to balance the trade-offs among driver travel distance, order delays, and the number of unserved orders. Let e_i denote the last dispatch time for agent i in the queue (i.e., orders that have not been picked up or cancelled). For $t < e_i$, the reward for agent i at time t is denoted as r_t^i. We consider: (1) If agent i decides not to enter the matching pool, a constant penalty is incurred: $r_t^i = -\Delta t$. This penalty is imposed due to the existence of a promised delivery time for each order, and customers generally prefer to receive their orders promptly; (2) If agent i is cancelled by the customer after a prolonged waiting time, the reward is calculated as: $r_t^i = -\Delta t - r_{\text{cancel}}$. Here, $-r_{\text{cancel}}$ represents a constant negative penalty reward assigned to cancelled orders; (3) If agent i enters the matching pool and is subsequently picked up by a driver, the reward is determined as: $r_t^i = r_{\text{match}} - \frac{d_{i,j}}{v}$. In this case, r_{match} represents a constant positive reward for successful matching, $d_{i,j}$ denotes the estimated increase in travel distance for the matched driver j during the order pickup and delivery process, and v represents the driver speed.

B. Reinforcement Learning Algorithm

We initialize a neural network with parameter $\boldsymbol{\theta}$ to approximate the action value function $Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta})$. For every decision time $t \in \mathcal{T}$, we gather the transitions $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ for each agent i and store them within agent-specific replay buffers denoted as $\mathcal{B} = \{\mathcal{B}^1, \mathcal{B}^2, \cdots, \mathcal{B}^{|N|}\}$, where $\mathcal{B}^i$ signifies the replay buffer for agent i. The training process aims to iteratively update the action-value function $Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta})$ in order to find the optimal action-value function $Q_*(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^*)$ that maximize the expected long-term reward. The details are summarized in Algorithm 1.

C. Driver Routing

This study primarily centers around order dispatching, yet routing decisions play a vital role in the overall service process at a lower level. In recent years, there has been extensive attention given to the problem of route optimization in food delivery. The routing problem is first formalized as meal delivery routing problem in [4], which is a variant of dynamic pickup and delivery problem and has been studied in recent decades [5]. Given the high demand for timely food delivery, exact solution methods [6] are often computationally intensive and impractical for addressing driver routing problems. Consequently, conventional heuristic algorithms [1], [10] are typically employed in prominent food delivery platforms (e.g., Meituan) as a means of swiftly obtaining approximate solutions. In this subsection, we present a comprehensive account of our driver routing algorithm, as depicted in Algorithm 2.

Dispatching rules and assumptions for service operations are as follows: (1) *Dispatch fairness*: If the number of orders carried by driver d is large/small (i.e., ω_d is large/small), s/he will be less/more likely dispatched new orders; (2) *Order re-dispatch*: Orders that have not been picked up but have been dispatched a driver can be re-dispatched to a new

Algorithm 1 Dynamic Multi-agent Deep Q Learning

```
1: procedure DYAMIC-M-DQN
2:     Initialize $\mathcal{B} = \{\mathcal{B}^1, \mathcal{B}^2, \cdots, \mathcal{B}^{|N|}\}$, $Q(\boldsymbol{\theta})$;
3:     for $e = 1 \to |E|$ do
4:         Initialize $s_0$, $\mathcal{B}^i$;
5:         for $t \in \mathcal{T}$ do
6:             for $i = 1 \to N_t$ do
7:                 Get $s_t^i$;
8:                 Take $a_t^i = \text{argmax} Q(s_t^i, a_t^i|\boldsymbol{\theta})$;
9:             end for
10:            Get $\mathbf{s}_t = \{s_t^1, s_t^2, \cdots, s_t^{N_t}\}$;
11:            Get $\mathbf{a}_t = \{a_t^1, a_t^2, \cdots, a_t^{N_t}\}$;
12:            Observe $\mathbf{r}_t = \{r_t^1, r_t^2, \cdots, r_t^{N_t}\}$;
13:            Observe $\mathbf{s}_{t+1} = \{s_{t+1}^1, s_{t+1}^2, \cdots, s_{t+1}^{N_t}\}$;
14:            for agent $i = 1 \to N_t$ do
15:                Append $(s_t^i, a_t^i, r_t^i, s_{t+1}^i, d_t^i)$ to $B^i$;
16:                Append $\mathcal{B}^i$ to $\mathcal{B}$;
17:            end for
18:        end for
19:        if $|\mathcal{B}| \geq b_{\min}$ then
20:            Randomly sample $M$ complete trajectories
               $(s_t^i, a_t^i, r_t^i, s_{t+1}^i, e_t^i)$ from $\mathcal{B}$ from all agents;
21:            Update $\boldsymbol{\theta}$:
22:            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \mathcal{L}(\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} Q(s_t^i, a_t^i|\boldsymbol{\theta})$
23:        end if
24:    end for
25: end procedure
```

Algorithm 2 Driver Routing Algorithm

```
Input: $\mathcal{O}_{\text{pool}}$, $\mathcal{D}$, $\mathcal{R}$, $\omega_d$
Output: $\text{RT}'_d$
1: procedure ROUTING($\mathcal{O}_{\text{pool}}, \mathcal{D}$)
2:     Initialize $\mathcal{O}_{\text{pick}} = \emptyset$
3:     for $d \in \mathcal{D}$ do
4:         if $c$ is going to pick up order $o \in \mathcal{O}_{\text{pool}}$ then
5:             Update $\mathcal{O}_{\text{pick}} \leftarrow \mathcal{O}_{\text{pick}} \cup o$
6:         end if
7:     end for
8:     for $o \in \mathcal{O}_{\text{pool}}$ do                    ▷ Greedy insertion
9:         if $o \in \mathcal{O}_{\text{pick}}$ then
10:            Continue
11:        end if
12:        Initialize $\mathcal{D}_{\text{avail}} = \emptyset$
13:        for $d \in \mathcal{D}$ do
14:            if $|\omega_d| < cap$ then
15:                $\mathcal{D}_{\text{avail}} \leftarrow \mathcal{D}_{\text{avail}} \cup c$
16:            end if
17:        end for
18:        Assign $o$ to $d = \text{argmax}\mathcal{D}_{\text{avail}}$
19:        do greedy insertion
20:        Update $\text{RT}_d \leftarrow \text{RT}_d \cup (o_+, o_-)$
21:    end for
22:    for $d \in \mathcal{D}$ do          ▷ Driver's delivery route reconstruct
23:        if $\omega_d \neq \emptyset$ then
24:            Get $l_d$ and $o_-$ for order $o \in \omega_d$
25:            Solve a open TSP problem with $\omega_d$
26:            Update $\text{RT}_d$
27:        end if
28:    end for
29:    return new delivery plan route $\text{RT}'_d$ for $d \in \mathcal{D}$
30: end procedure
```

driver; (3) *driver routing behavior*: Once order dispatching is done, drivers deliver their carried orders by solving an open Traveling Salesman Problem (TSP) to optimize their route; (4) *Order cancellation*: If a newly generated order cannot find an available driver over a period of time (e.g., 20 minutes), we assume that the customer become impatient and cancel the order. The first rule is aimed at ensuring fairness in dispatching by preventing certain drivers from being overloaded with a disproportionately large number of orders, while others only receive a few. The second rule serves to enhance the flexibility of the dispatch algorithm by allowing for order re-dispatch, which has the potential to reduce order pickup time. Third, we assume that drivers always follow the shortest one-way delivery travel route from the restaurant to one of the customer locations. This route is derived by solving an open TSP, where the driver does not return to the starting location. Lastly, customers have the option to cancel orders if the platform fails to find a driver within a specific time period.

V. SIMULATOR AND DATASET

Simulator. To facilitate the training of Dynamic-M-DQN (given in Algorithm 1) and to generate representative samples that mimic real-world scenarios in food delivery service, we designed a simulator calibrated with real data. The simulator consists of multiple modules: order generation, information collection, the order dispatching algorithm, and updates for both order and driver statuses. Five distinct states have been defined for each agent and order, specifically denoted as "Generated", "Dispatched", "Ongoing", "Completed", and "Cancelled". The state denoted as "Generated" signifies that the customer has initiated an order placement, yet it remains pending for dispatch to a driver for pickup. "Dispatched" indicates that the order has entered the matching pool, though pickup by a driver has not transpired as of yet. "Ongoing" signifies that a driver has collected the order and is presently in the process of delivery. Ultimately, "Completed" denotes successful delivery to the customer, while "Cancelled" denotes the customer's decision to rescind the order. The comprehensive simulation framework is depicted in Figure 3, comprising five key constituents: input data, agent properties, platform operational procedures, optimization algorithms, and output outcomes.

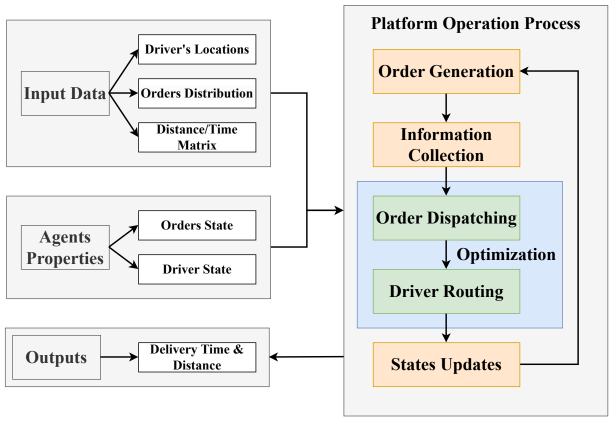


Fig. 3: Simulation Framework.

Real-world Dataset. The dataset employed in the numerical experiments originates from a crowd-sourced food delivery platform in Singapore [1]. The dataset comprises approximately $80,000$ order records and more than $30,000$ customers spanning 8 months (October 2020 to May 2021). Each delivery record consists of a wealth of order and driver details, encompassing order pickup and delivery locations, delivery distance, order acceptance time, driver identification, and the corresponding fee. Evidently, the order volume exhibits substantial augmentation during the peak intervals at $11:00$ and $18:00$. For the purpose of enabling in-depth exploration, we have proactively provided access to the tailored simulator and exemplar test data utilized within this research to the wider community[2].

VI. EXPERIMENTS

In this section, we present a set of experiments to evaluate the performance of our two-stage learning-based optimization approach. We generate restaurants and customer orders to be delivered following the patterns of our partner dataset. We also account for the random ready time for orders in our simulations. We make the assumption that the food preparation time for each restaurant follows distinct gamma distributions, characterized by different shape parameters α and a scalar parameter β. This choice is motivated by the common use of gamma distributions for modeling waiting times in on-demand food delivery platforms [7], [11]. We set the promised delivery time to 45 minutes and the velocity of the drivers is established at 20 km/h. The simulations are executed employing Python version 3.9 and leveraging the PyTorch library [12].

Parameters Settings. Table I presents the hyper-parameter configurations for our Dynamic-M-DQN algorithm. An epoch is delineated as the complete temporal span $\mathcal{T}$, ranging from $6:00$ AM to $10:00$ PM. This epoch is partitioned into 96 dispatching time intervals, each lasting 10 minutes, predicated on the order arrival rate within our dataset.

TABLE I: Hyper-parameter settings for Dynamic-M-DQN.

Parameters	Values	Parameters	Values
Learning rate	$2e-3$	Hidden layer neurons	$512, 256, 128$
Discount γ	0.99	Epochs	100
Epsilon ϵ	$0.1 \sim 0.9$	Dispatches in one episode	96
Buffer size $\lvert\mathcal{B}\rvert$	500	Batch size	64
Target update	10	State dimension	$2 \times \lvert\mathcal{M}\rvert^2 + 2\lvert\mathcal{M}\rvert + 2$

Evaluation Metrics. Our experimental focus is on the minimization of a weighted sum that encompasses the total travel distance of drivers, cumulative order delays, and the count of customer-cancelled orders. To effectively portray delivery performance, we employ three evaluation metrics: (1) total travel distance of drivers (C_{total}) in kilometers; (2) cumulative order delays (T_{delay}) in seconds; and (3) count of customer-initiated cancellations (N_{cancel}) stemming from extended waiting periods. The objective is defined as follows:

$$\text{Obj} = \frac{D_{\text{total}}}{|D|} + \lambda \cdot \frac{T_{\text{delay}}}{3600} + \mu \cdot N_{\text{cancel}} \qquad (2)$$

Our research focuses on a multi-objective optimization challenge, aiming to achieve a balance among three critical elements: the delivery cost for drivers, as measured by the average travel distance; the platforms' revenue, reflected by the number of unserved orders; and the user experience, demonstrated through the extent of order delays. Importantly, platforms have the flexibility to select different values for λ and μ, allowing them to tailor the evaluation of model performances based on their prioritization of these two objectives. Here, a penalty value of $\lambda = 100$ is employed for delays, as customers frequently hold elevated delivery time expectations. Extended delays may prompt customers to depart the platform in search of alternatives. Furthermore, a significant penalty value of $\mu = 500$ is administered to mitigate the concern of customer cancellations arising from extended waiting durations for available drivers.

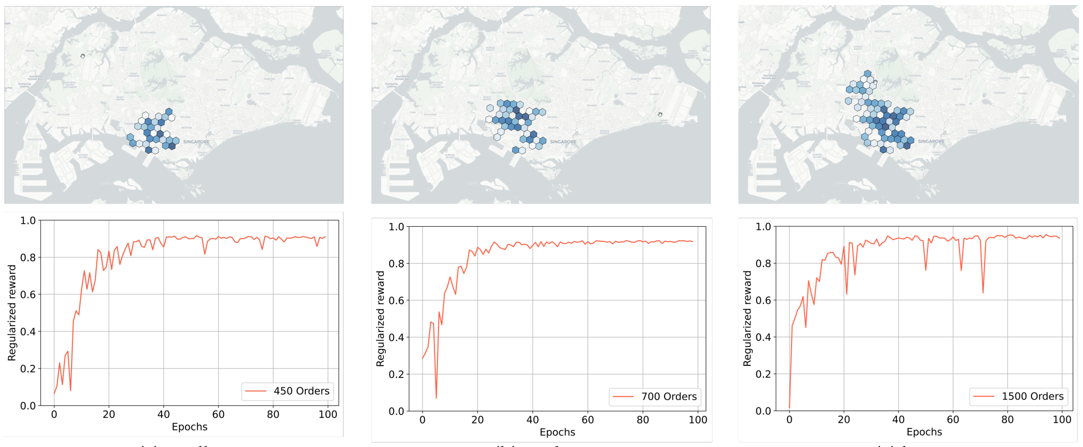
Main Results. Our investigation encompasses three datasets varying in scale: a small dataset, comprising a single restaurant with more than 450 orders and 15 drivers; a medium dataset, encompassing two restaurants with a total of over 700 orders and 40 drivers; and a large dataset, involving three restaurants and exceeding 1500 orders, with a driver pool of 100. Figure 4 presents a visual representation of customer order locations for each dataset. The degree of shading in the hexagonal areas corresponds to the number of orders received, with darker shades indicating higher order volumes. In this study, we analyze the performance of the proposed Dynamic-M-DQN coupled with the DR algorithm. A comparative evaluation is conducted between this approach and the following commonly employed dispatch strategies and routing algorithms:

- FIX: Dispatch strategy. In this approach, the order dispatching time interval remains constant for all orders (e.g., 5 minutes), mirroring the default dispatching mode found on numerous food delivery platforms like Meituan [1] and Grubhub [6];
- MGI: Routing Algorithm. We use the Greedy heuristic, as described in [13], for our routing algorithm. This heuristic is known for effectively handling the dynamic pickup and delivery problem common in the industry. In particular, MGI stands for Modified Greedy Insertion, a variation used by Meituan-Dianping [1], the largest food delivery platform in China; [3];
- H-RL: Dispatch strategy. This strategy employs a hierarchical reinforcement learning based optimization framework, as proposed by [9]. In this approach, the upper-level agent dynamically decomposes the dynamic pickup and delivery problem (DPDP) into a sequence of sub-problems for optimization. Meanwhile, the lower-level agent selects operators to enhance the resultant

[1]The dataset used in this study was contributed by an industry collaborator who has chosen to remain anonymous.

[2]https://anonymous.4open.science/r/Food-Delivery-Simulator-37DC/

[3]Meituan-Dianping is a Chinese shopping platform in China for locally found consumer products and retail services including entertainment, dining, delivery, travel and other services. https://www.meituan.com/



(a) small (b) medium (c) large

Fig. 4: Density of customer orders and reward convergence curves on small, medium and large instances.

solution. We utilize the term H-RL to represent the DQN model situated in the upper-level for the purpose of order dispatch.

TABLE II: Results summary of the three approaches on different sized datasets.

Instance	Approach	C_{total}	T_{delay}	N_{cancel}	$\mathbb{M}_{\text{Rat}}$
Small	Dynamic-M-DQN + DR	**2,130.37**	**191**	0	**0.825**
450 orders	H-RL + DR	2,186.85	896	0	0.243
15 drivers	FIX + DR	2228.31	1,113	0	0.064
	FIX + MGI	2226.75	1,191	0	0
Instance	**Approach**	C_{total}	T_{delay}	N_{cancel}	$\mathbb{M}_{\text{Rat}}$
Medium	Dynamic-M-DQN + DR	3,091.55	**562**	0	**0.758**
700 orders	H-RL + DR	**3,036.23**	1,221	0	0.474
40 drivers	FIX + DR	3,107.87	1,783	0	0.231
	FIX + MGI	3,118.85	2,320	0	0
Instance	**Approach**	C_{total}	T_{delay}	N_{cancel}	$\mathbb{M}_{\text{Rat}}$
Large	Dynamic-M-DQN + DR	**6,666.08**	**10,645**	**5**	**0.153**
1,500 orders	H-RL + DR	6,682.91	11,016	6	0.123
100 drivers	FIX + DR	6,692.72	11,136	6	0.114
	FIX + MGI	6,752.39	12,581	6	0

Our comparative analysis includes our proposed approach Dynamic-M-DQN + DR alongside the following policies: (1) H-RL + DR; (2) FIX + DR; and (3) FIX + MGI. As a baseline, we will employ the FIX + MGI approach. We quantify several metrics, specifically the total travel distance C_{total}, cumulative delays T_{delay}, and the count of canceled orders N_{cancel}, for each policy, correspondingly. These measurements are employed to ascertain the effectiveness of our proposed model. The comprehensive experimental results in Table II illustrate that employing the Dynamic-M-DQN + DR approach results in a substantial reduction in cumulative delays, along with a moderate decrease in total travel distance for drivers. Interestingly, the weighted sum objective value, as computed using equation (2), exhibited reductions of 82.5%, 24.3%, and 6.4% relative to the baseline approach for Dynamic-M-DQN + DR, H-RL + DR, and FIX + DR, respectively. In cases of both small and medium-sized instances, all customer orders can be fulfilled without any cancellations. In the case of large-sized instances, our Dynamic-M-DQN + DR approach stands out among all methods, demonstrating the lowest count of canceled orders.

Next, we define $\mathbb{M}_{\text{approach}}$ as the changing ratio for a given evaluation metric between the selected approach and the baseline approach FIX + MGI as follows:

$$\mathbb{M}_{\text{Rat}} = \frac{\mathbb{M}_{\text{approach}} - \mathbb{M}_{\text{FIX + MGI}}}{\mathbb{M}_{\text{FIX + MGI}}} \tag{3}$$

Our approach resulted in a reduction in the rate of total travel distance for drivers, with decreases ranging from 4.3% to 0.9% to 1.3%. Additionally, substantial reductions of 84.0%, 75.8%, and 15.4% were observed in total order delays for the small, medium, and large instances respectively. The findings suggest that dynamic optimization of the dispatching time interval for order dispatch can yield substantial enhancements in service. These enhancements manifest as noteworthy reductions in delays (up to 84.0%), accompanied by a slight decrease in delivery distances (up to 4.3%).

Figure 4 depicts the convergence patterns of the regularized mean rewards attained by our Dynamic-M-DQN model across datasets of varying sizes. Our observations suggest that our approach can establish a high-quality policy within around 40 epochs across all datasets, as substantiated by the converging training process. These outcomes underscore both the effectiveness and efficiency of our policy for dynamic dispatching time intervals, which effectively optimizes the order dispatching process within on-demand food delivery services.

Effect of different reward. Next we present the main idea in reward design and look at the affects among different objectives. In the objective function, we consider the trade-offs among driver travel distance, order delays, and the number of unserved orders under two scenarios: (1) If an order is eligible for the matching pool, the driver's travel distance is still a crucial factor. To account for this, we set $r_t^i = r_{\text{match}} - \frac{d_{i,j}}{v}$. Here, if $d_{i,j}$ is significantly large, the benefit of dispatching order i reduces, potentially becoming negative. (2) If an order is postponed, meaning it does not enter the matching pool, we consider the trade-off between postponement and cancellation. In this scenario, we define $r_t^i = -\Delta t$ for postponement and $r_t^i = -\Delta t - r_{\text{cancel}}$ for cancellation. A large value of $-r_{\text{cancel}}/\Delta t$ suggests that the platform is striving to delay the allocation of orders to maximize potential benefits on saving travel distance and order delays, while also attempting to minimize the likelihood of orders being cancelled by customers.

TABLE III: Evolution of the performance with different ρ.

Δt	r_{cancel}	ρ	C_{total}	T_{delay}	N_{cancel}
1	-100	100	3,095.00	638	0
10	-100	10	**3,091.55**	**562**	0
20	-100	5	3,143.02	1,963	0
50	-100	2	3,135.33	2,859	0

In light of these findings, we conducted additional experiments to refine the design of the reward function. Specifically, a sensitivity analysis was performed focusing on the ratio $\rho = -r_{\text{cancel}}/\Delta t$, which is pivotal in training our Dynamic-M-DQN method. According to the results presented in Table III, our approach achieves the lowest total travel distance (C_{total}) and delay times (T_{delays}) when $\rho = 10$. Conversely, both the total travel distance and delays increase with a smaller ρ (e.g., $\rho = 5, 2$). This trend can be attributed to the fact that a lower ρ places greater emphasis on postponing rather than cancelling orders. Consequently, with a lower ρ, orders tend to enter the matching pool sooner, as postponement incurs a substantial negative reward, almost equivalent to cancellation. However, a very high ρ value (such as $\rho = 100$) does not necessarily equate to better performance. This is attributed to the tendency of orders to be excessively postponed with a high ρ, potentially missing optimal entry times into the matching pool, which could result in lower travel distances and delays. With a given λ and μ by the platform, if the ratio μ/λ is large, it indicates that the platform prioritizes serving more orders over minimizing order delays. To align with this objective function, a larger ρ should be adopted to discourage order cancellation.

Effect of different dispatch time interval. At last, we evaluated the performance of the DR algorithm using a newly assembled dataset. This assessment encompassed various fixed time intervals between consecutive dispatch decisions. The results of this comparative analysis are presented in Table IV. Our observations suggest that extending the dispatching time interval correlates with an increment in the cumulative delivery distance covered by the drivers. Conversely, this adjustment leads to a substantial decrease in order delay times. Moreover, it is worth noting that all orders can be fulfilled without any cancellations in all scenarios.

TABLE IV: Effect of different dispatching time intervals.

	Metrics	2 mins	5 mins	10 mins	15 mins
DR	D_{total}	2,275.86	2,681.60	3,040.71	3,111.33
	T_{delay}	58,916	3,873	755	371
	N_{cancel}	0	0	0	0

VII. CONCLUSIONS

We introduce a novel two-stage framework to optimize order dispatching and driver routing in on-demand food delivery services. The upper stage features a multi-agent reinforcement learning approach to dynamically determine the best timing for including orders in the matching pool. The lower stage involves a simple yet efficient new routing algorithm. We conducted extensive testing through a customized simulator, comparing our approach against benchmarks that use fixed dispatch intervals, the commonly used MGI in driver routing, and a hierarchical RL-based method. Our proposed approach significantly enhances performance metrics, with improvements ranging from 15.3% to 82.5% across various dataset sizes. Results demonstrate a marked reduction in average travel distance and notable alleviation of delivery delays. Future work will explore the application of RL methodologies to develop a comprehensive end-to-end routing policy.

REFERENCES

[1] Q. Zhou, H. Zheng, S. Wang, J. Hao, R. He, Z. Sun, X. Wang, and L. Wang, "Two fast heuristics for online order dispatching," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[2] J. Ke, F. Xiao, H. Yang, and J. Ye, "Learning to delay in ride-sourcing systems: a multi-agent deep reinforcement learning framework," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 5, pp. 2280–2292, 2020.

[3] H. Yang, X. Qin, J. Ke, and J. Ye, "Optimizing matching time interval and matching radius in on-demand ride-sourcing markets," *Transportation Research Part B: Methodological*, vol. 131, pp. 84–105, 2020.

[4] D. Reyes, A. Erera, M. Savelsbergh, S. Sahasrabudhe, and R. O'Neil, "The meal delivery routing problem," *Optimization Online*, vol. 6571, 2018.

[5] H. N. Psaraftis, M. Wen, and C. A. Kontovas, "Dynamic vehicle routing problems: Three decades and counting," *Networks*, vol. 67, no. 1, pp. 3–31, 2016.

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

[7] 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," *Transportation Science*, vol. 55, no. 1, pp. 75–100, 2021.

[8] W. Weng and Y. Yu, "Labor-right protecting dispatch of meal delivery platforms," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 1349–1355.

[9] Y. Ma, X. Hao, J. Hao, J. Lu, X. Liu, T. Xialiang, M. Yuan, Z. Li, J. Tang, and Z. Meng, "A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems," *Advances in Neural Information Processing Systems*, vol. 34, pp. 23 609–23 620, 2021.

[10] X. Wang, L. Wang, S. Wang, J.-f. Chen, and C. Wu, "An xgboost-enhanced fast constructive algorithm for food delivery route planning problem," *Computers & Industrial Engineering*, vol. 152, p. 107029, 2021.

[11] C. Gao, F. Zhang, Y. Zhou, R. Feng, Q. Ru, K. Bian, R. He, and Z. Sun, "Applying deep learning based probabilistic forecasting to food preparation time for on-demand delivery service," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2924–2934.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[13] S. Mitrović-Minić and G. Laporte, "Waiting strategies for the dynamic pickup and delivery problem with time windows," *Transportation Research Part B: Methodological*, vol. 38, no. 7, pp. 635–655, 2004.