Triple-BERT: Do We Really Need MARL for Order Dispatch on Ride-Sharing Platforms?

Anonymous Author(s)
Affiliation
Address
email

Abstract

On-demand ride-sharing platforms, such as Uber and Lyft, face the intricate real-time challenge of bundling and matching passengers—each with distinct origins and destinations—to available vehicles, all while navigating significant system uncertainties. Due to the extensive observation space arising from the large number of drivers and orders, order dispatching, though fundamentally a centralized task, is often addressed using Multi-Agent Reinforcement Learning (MARL). However, independent MARL methods fail to capture global information and exhibit poor cooperation among workers, while Centralized Training Decentralized Execution (CTDE) MARL methods suffer from the curse of dimensionality. To overcome these challenges, we propose Triple-BERT, a centralized method designed specifically for large-scale order dispatching on ride-sharing platforms. Built on TD3, our approach addresses the vast action space through an action decomposition strategy that breaks down the joint action probability into individual driver action probabilities. To handle the extensive observation space, we introduce a novel BERT-based network, where parameter reuse mitigates parameter growth as the number of drivers and orders increases, and the attention mechanism effectively captures the complex relationships among the large pool of driver and orders. We validate our method using a real-world ride-hailing dataset from Manhattan. Triple-BERT achieves approximately an 11.95% improvement over current state-of-the-art methods, with a 4.26% increase in served orders and a 22.25% reduction in pickup times. Our code, trained model parameters, and processed data are publicly available at the anonymous repository `https://anonymous.4open.science/r/Triple-BERT`.

1 Introduction

Ride-sharing platforms, such as Uber and Lyft, face the complex challenge of dynamically matching passengers with distinct origins and destinations to available vehicles in real time. This task must account for significant system uncertainties, including fluctuating demand, varying traffic conditions, and the availability of drivers. As the volume of concurrent ride requests increases, these platforms must efficiently allocate resources to minimize detours, reduce waiting times, and maximize customer satisfaction and platform revenue. However, the inherently large and dynamically changing action and observation spaces make this problem highly challenging for the operation of ride-sharing platforms.

Recently, Reinforcement Learning (RL) methods have shown great potential in addressing the order dispatching problem in ride-sharing platforms. Model-free RL, in particular, enables agents to autonomously learn optimal dispatching policies by interacting with the environment, without requiring complex system modeling. This approach allows platforms to optimize multiple objectives, including platform income, driver payments, and customer satisfaction. Despite these advantages,

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

applying RL to large-scale order dispatching introduces significant challenges. The vast action and observation spaces, stemming from the large number of drivers and orders, make sufficient exploration and efficient training difficult. Multi-Agent Reinforcement Learning (MARL) methods have been widely adopted to address these challenges by decomposing the problem into smaller subproblems for individual agents (drivers). Independent MARL methods, such as IDDQN [1; 2; 3] and ISAC [4], are computationally efficient but fail to capture global information and exhibit limited cooperation among agents. Graph Neural Networks (GNNs) have been introduced to enable the network to capture neighboring information for each agent, alleviating this issue to certain extent [5; 6]. Meanwhile, Centralized Training with Decentralized Execution (CTDE) methods, such as QMIX [7] and CoPO [8], struggle with the curse of dimensionality when applied to large-scale scenarios with thousands of agents, resulting in slow convergence and suboptimal performance.

To address these limitations, this paper proposes a centralized Single-Agent Reinforcement Learning (SARL) method, named Triple-BERT, tailored for large-scale order dispatching in ride-sharing platforms. Triple-BERT introduces an action decomposition method that simplifies the joint action probability into individual driver action probabilities, enabling each driver to make independent decisions while maintaining global coordination. The method leverages TD3 [9] for optimization, with modifications to the actor optimization process via policy gradient [10] to better suit the ride-sharing context. To handle the extensive observation space, we design a novel BERT-based [11] neural network architecture. This network employs bi-directional self-attention to effectively capture complex relationships between drivers and orders, while its parameter reuse mechanism prevents parameter explosion as the number of drivers and orders increases. Additionally, compared to MARL, SARL faces a unique challenge of sample scarcity, as the records of multiple agents are merged into a single training stream. To address this, we propose a two-stage training strategy, where feature extractors are pre-trained using a MARL approach to learn general embedding capabilities, followed by centralized fine-tuning. The main contributions of this paper can be summarized as follows:

- We introduce Triple-BERT, which is the first centralized SARL framework for large-scale order dispatching on ride-sharing platforms. This approach addresses the limitations of the observation space and the inefficiencies in cooperation among agents present in MARL methods. To tackle the large action space inherent in the matching problem of order dispatching tasks, we propose an action decomposition method that breaks down the joint action probability into individual driver action probabilities. Additionally, we propose a two-stage training method to address the sample scarcity issue in SARL, where the feature extractors are first trained using a MARL approach.
- To support the proposed RL framework in a large observation space, we develop a novel neural network architecture based on BERT. This design leverages self-attention mechanisms to effectively capture the relationships between drivers and orders. Furthermore, we incorporate a QK-attention module to reduce computational complexity from multiplication to addition in the order dispatching task, along with a positive normalization method to mitigate parameter redundancy issues.
- We validate the proposed method in the ride sharing scenario, using a real-world dataset of ride-hailing trip records from Manhattan. Our method outperforms the MARL methods reported in previous works, demonstrating approximately a 11.95% improvement over current state-of-the-art methods, with a 4.26% increase in served orders and reductions of about 22.25% in pickup time.

2 Problem Setup

In this paper, we address the order dispatching task within on-demand logistic systems, such as ride hailing, food delivery, and express delivery. We consider a platform managing n drivers (hereafter referred to as workers), represented by the state $W_t = \{w_{1,t}, w_{2,t}, \ldots, w_{n,t}\}$, where $w_{i,t}$ denotes the state of worker i at time t. At each time step, the platform processes a set of orders, including newly arrived orders and any previously unassigned orders, denoted as $O_t = \{o_{1,t}, o_{2,t}, \ldots, o_{m_t,t}\}$, where m_t is the total number of orders at time t. Since real-time performance is crucial in on-demand systems, the platform aims to bundle and assign orders in a way that minimizes delivery time while maximizing the number of served orders. Customers are assumed to be impatient; if an order is not acknowledged within a specified time frame, workers will decline it. Moreover, late deliveries beyond the scheduled time may result in customer complaints, potentially causing losses for the platform. The overall workflow is illustrated in Fig. 1, and the Markov Decision Problem (MDP) is formulated as $< S, A, R, P >$, encompassing the state, action, reward, and transition function, which will be detailed below:

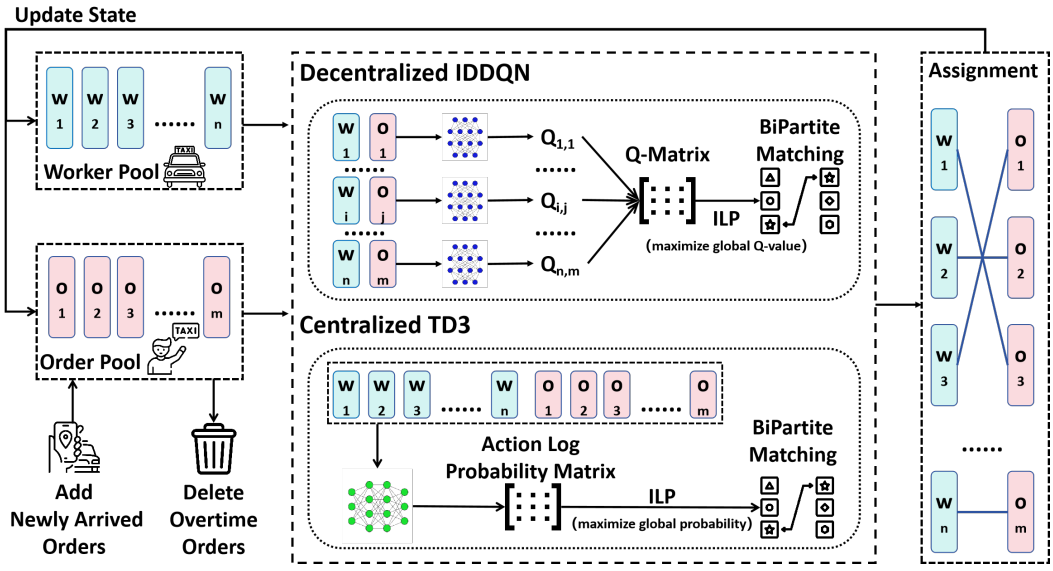


Figure 1: Workflow: At each time step, the worker and order pools update their states based on the assignments made in the previous time step. Specifically, the order pool adds newly arrived orders and removes overdue ones. For IDDQN, the Q-value of each worker-order pair is calculated, and ILP is applied to maximize the global Q-value. For TD3, the probability of each worker-order pair is computed, followed by the application of ILP to maximize the global assignment probability.

(i) State: At timestep t, the state or observation can be represented as $S_t = [W_t, O_t]$, consisting of the states of workers and orders. For the order j to be assigned, the state $o_{i,j}$ includes the order's origin and destination, pickup time, and scheduled arrival time. For each worker i, the state $w_{i,t}$ consists of the onboard orders $H_{i,t}$ that are still unfinished, the current location, the residual capacity, and the estimated time when he/she will be available to accept a new order. (Note that we assume if a worker is en route to pick up a new order or if his/her capacity is full, he/she cannot serve a new order.) Specifically, $H_{i,t}$ is a sequence of orders $H_{i,t} = \{h_{i,1,t}, h_{i,2,t}, \ldots, h_{i,k_{i,t},t}\}$, where $k_{i,t}$ is the number of onboard orders for worker i at time t and each order $h_{i,k,t}$ contains the same information as the orders to be assigned $o_{j,t}$.

(ii) Action: At each time t, the action can be represented as $A_t = \{a_{1,t}, a_{2,t}, \ldots, a_{n,t}\}$, where each $a_{i,t}$ is an m_t-dimensional vector with at most one element set to 1, indicating which order is assigned to worker i. The order dispatching task is particularly challenging due to two main factors: (i) the size of the action space keeps changing over time because the number of orders m_t varies dynamically as new orders arrive and old orders are completed or canceled; (ii) the size of the action space is extremely large for real systems. For instance, considering $n = 1000$ workers and $m_t = 10$ orders, the action space can reach approximately 10^{30}. (A detailed proof is provided in Appendix A.) This combination of an enormous action space and its continuously changing size significantly complicates sufficient exploration and stable network convergence for standard RL methods.

(iii) Reward Function: We split the reward function for each worker, meaning each worker will receive a reward $r_{i,t+1}$ at time step t, and the global reward is the sum of each worker's reward: $R_{t+1} = \sum_{i=1}^{n} r_{i,t+1}$. The reward $r_{i,t+1}$ can be calculated according to the following function:

$$r_{i,t+1} = \mathcal{R}(s_{i,t}, a_{i,t}) = \begin{cases} \beta_1 + \beta_2 p_{i,t}^{in} - \beta_3 p_{i,t}^{out} - \beta_4 \chi_{i,t} - \beta_5 \rho_{i,t}\,, & |a_{i,t}| = 1 \\ 0\,, & |a_{i,t}| = 0 \end{cases} \tag{1}$$

where β_1 to β_5 are non-negative weights representing the platform's valuation of each term, $p_{i,t}^{in}$ and $p_{i,t}^{out}$ represent the income from customers and the payout to workers, respectively. The variables $\chi_{i,t}$ and $\rho_{i,t}$ represent the number of en-route orders that will exceed their scheduled time and the additional travel time of all en-route orders when the assigned order is added to the scheduled route of worker i at time t, respectively. This reward function is designed to comprehensively consider the interests of the platform, workers, and customers, mimicking the operation of a real-world food delivery platform. It is important to emphasize that $p_{i,t}^{in}$ and $p_{i,t}^{out}$ are calculated based on the order

distance and the additional travel distance for the worker, respectively. When calculating travel time, we will utilize the Traveling Salesman Problem (TSP) to optimize the worker's route.

(iv) Transition Function: In our system, the reward is deterministic given the current state and action. Therefore, the transition function is represented by $P(S_{t+1}|S_t, A_t)$. In this study, the transition probabilities are not explicitly modeled; instead, they are inferred through the model-free RL.

3 Methodology

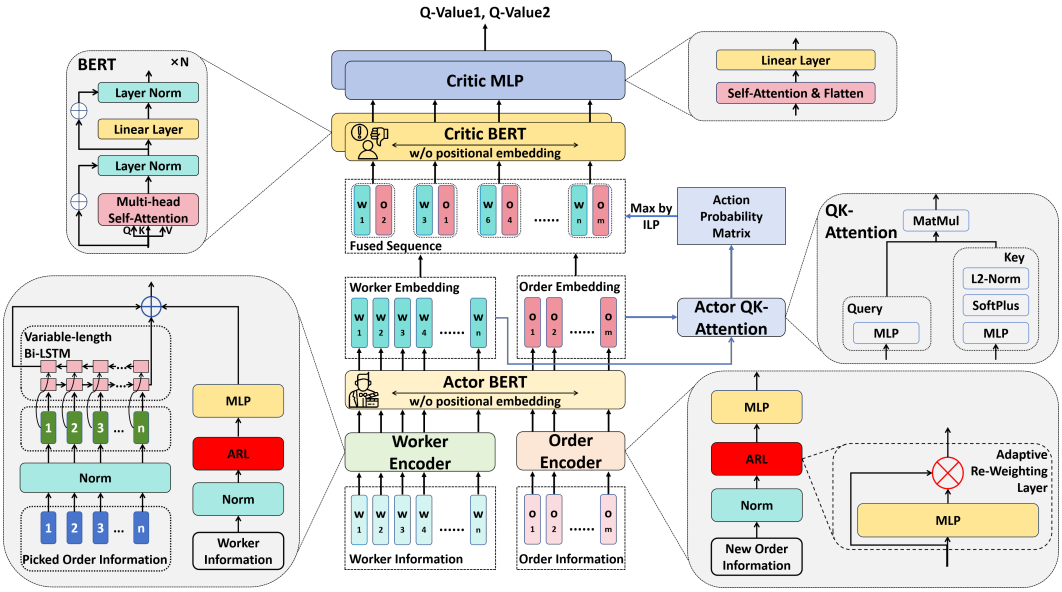


Figure 2: Proposed Network Architecture: In this figure, the fused sequence (input to Critic-BERT) represents workers 1, 3, 6, and n selecting orders 2, 3, 4, and m, respectively.

3.1 Overview

In this work, we aim to utilize centralized SARL to address the large-scale order dispatching task, with the goal of enabling the model to fully leverage global information to enhance cooperation among workers. To tackle the challenges of large action and observation spaces, we propose a novel network architecture, as illustrated in Fig. 2. This architecture employs the BERT model [11] to effectively extract the relationships between workers and orders using the self-attention mechanism. Additionally, an improved QK-attention [12] is implemented to reduce the computational complexity associated with the order dispatching task. Furthermore, we introduce an action decomposition method that breaks down the choice probability of each action within the vast action space into individual action probabilities for each worker selecting each order. Finally, to address the data scarcity challenge in MARL, we propose a two-stage training method, as shown in Fig. 1. In the first stage, we train the upstream layers of the network using the IDDQN approach, allowing them to develop general feature extraction capabilities. Subsequently, we train the entire neural network using centralized TD3 to realize better cooperation between workers.

3.2 Network Architecture

The proposed network structure is shown as Fig. 2, which constists of three parts: encoders (embed the worker and order information to a common feature space), actor sub-network (a BERT to extract the relationship between different workers and orders and a QK-Attention to generate the utility/probability of each worker-order pair), and critic sub-network (two BERT taking output of actor BERT as input and output the Q-value respectively).

3.2.1 Feature Extractors

At each time step, the network takes the entire state $S_t = [W_t, O_t]$ as input. We consider this as a combination of two sequences: W_t and O_t. For each element $w_{i,t}$ and $o_{j,t}$, we employ two distinct encoders, referred to as the "Worker Encoder" and the "Order Encoder", to embed them separately into a feature space of the same dimension, allowing them to be input into a single BERT model.

Each worker state $w_{i,t}$ consists of two parts: an on-board order sequence and other non-sequence information. For the order sequence, a bi-directional LSTM [13] is utilized to extract its features. This approach effectively encodes variable-length sequences into a uniform dimensional feature space, addressing the curse of dimensionality associated with conventional MLP encoders, where the number of parameters increases with sequence length. For non-sequence information, an MLP is employed for feature extraction. Finally, the two features are combined into a primary feature $\tilde{w}_{i,t}$. For the orders to be assigned $o_{j,t}$, an MLP is also used to extract the feature $\tilde{o}_{j,t}$. Notably, the dimensions of $\tilde{w}_{i,t}$ and $\tilde{o}_{j,t}$ are identical, and their information is concatenated into a sequence represented as $\tilde{S}_t = [\tilde{w}_{1,t}, \tilde{w}_{2,t}, \ldots, \tilde{w}_{n,t}, \tilde{o}_{1,t}, \tilde{o}_{2,t}, \ldots, \tilde{o}_{m_t,t}]$.

Additionally, to facilitate network convergence and enhance the extraction of input features, we incorporate a normalization layer and an Adaptive Re-weighting Layer (ARL) [14]. Given that different parts of the input may have varying magnitudes, which can impede model training, the normalization layer effectively addresses this issue. Furthermore, since different parts of the input carry different levels of importance, we utilize the ARL to enable the model to learn these variations, represented as: $y = x \circ \Omega$, where x denotes the input, Ω represents the weight vector, calculated by $\Omega = \text{MLP}(x)$, and $\circ$ indicates the element-wise product.

3.2.2 Actor Sub-Networks

The Actor sub-network consists of a BERT [11] model for feature extraction and a QK-attention module [12] for action decomposition and generation, which we will introduce in turn. In the feature extractors, we have already extracted the primary features from each worker and order state separately. To further explore the relationships between workers and orders, we utilize the BERT model, where the self-attention mechanism can effectively capture these relationships: $\overline{S}_t = [\overline{w}_{1,t}, \overline{w}_{2,t}, \ldots, \overline{w}_{n,t}, \overline{o}_{1,t}, \overline{o}_{2,t}, \ldots, \overline{o}_{m_t,t}] = \text{Actor-BERT}(\tilde{S}_t)$. Specifically, due to the permutation invariance of our input sequence, we omit the positional embedding in BERT, ensuring that the order in S does not influence the encoding result. In contrast to conventional MARL methods like [5; 7], which encode each worker with its neighboring states to gain a broader perspective, our Actor-BERT directly aggregates global worker information, facilitating more effective cooperative dispatching between workers.

In conventional order dispatching tasks, the typical approach to address the dynamic action space (related to the number of orders) involves evaluating each worker-order pair separately and finding the optimal dispatching solution based on these evaluations. However, this approach has two significant shortcomings. First, it neglects the relationships between orders, which we address through the self-attention mechanism in BERT, capturing not only the relationships between workers but also between orders and between orders and workers. Second, evaluating each worker-order pair is time-consuming and resource-intensive: $\text{F}(\overline{w}_{i,t}, \overline{o}_{j,t}; \theta_F) \in \mathbb{R}^1$, where F is the network and θ_F represents its parameters. The complexity can be represented as $O(|\text{F}| \cdot n \cdot m_t)$, where $|\text{F}|$ denotes the complexity of the neural network. To mitigate this issue, we employ a QK-attention module [12], represented as:

$$\text{QK-Attention}(\overline{w}_{i,t}, \overline{o}_{j,t}) := \text{f}(\overline{w}_{i,t}; \theta_f) \cdot \text{g}(\overline{o}_{j,t}; \theta_g)^T \approx \text{F}(\overline{w}_{i,t}, \overline{o}_{j,t}; \theta_F) \,, \tag{2}$$

where f and g are two smaller networks, and θ_f and θ_g are their parameters. The intuition behind QK-attention is to use two smaller networks to approximate a larger network, similar to the motivation behind LoRA [15]. In this way, the complexity of computing all worker-order pairs becomes $O(|\text{f}| \cdot n + |\text{g}| \cdot m_t + d \cdot n \cdot m_t)$, where $|\text{f}|$ and $|\text{g}|$ are the complexities of the two neural networks, d is their output dimension, and $d \cdot n \cdot m_t$ is the complexity of matrix multiplication. Here, d is very small, making $d \cdot n \cdot m_t$ much smaller than the neural network computation complexity, i.e., $d \cdot n \cdot m_t \ll |\text{f}| \approx |\text{g}| < |\text{F}|$. Thus, we have $O(|\text{f}| \cdot n + |\text{g}| \cdot m_t + d \cdot n \cdot m_t) < O(|\text{F}| \cdot (n + m_t)) < O(|\text{F}| \cdot n \cdot m_t)$, indicating that the QK-attention successfully transforms the multiplication complexity of evaluating each worker-order pair into addition complexity.

However, we observe a parameter redundancy issue in Equation 2, which can lead to potential instability during training. This redundancy arises because there are actually infinite solutions for f and g, as $f' = \alpha f$ and $g' = \frac{g}{\alpha}$ is also a valid solution for any non-zero real vector α. Inspired by Dueling DQN [16], we propose a positive normalization method:

$$\text{QK-Attention-Norm}(\overline{w}_{i,t}, \overline{o}_{j,t}) := \text{f}(\overline{w}_{i,t}; \theta_f) \cdot \frac{\text{Softplus}(\text{g}(\overline{o}_{j,t}; \theta_g))^T}{||\text{Softplus}(\text{g}(\overline{o}_{j,t}; \theta_g))||_2} \ . \tag{3}$$

This normalization ensures that the elements in $\frac{\text{Softplus}(\text{g}(\overline{o}_{j,t};\theta_g)^T)}{||\text{Softplus}(\text{g}(\overline{o}_{j,t};\theta_g)^T)||_2}$ are always non-negative, with an L2 norm of 1. This guarantees a unique solution. In our task, the output of the QK-attention is a matrix $M_t \in \mathbb{R}^{n,m_t}$, representing the utility of each worker choosing each order, which will be detailed in Section 3.3.2.

3.2.3 Critic Sub-Networks

The role of the critic is to evaluate the quality of actions, with the detailed action generation method introduced in Section 3.3.2. We first define an action function $\mathcal{A}$:

$$\mathcal{A}(w_{i,t}) = \begin{cases} (\overline{w}_{i,t}, \overline{o}_{j,t}) & \text{if order } j \text{ is assigned to worker } i \text{ at time } t \\ \emptyset & \text{if no order is assigned to worker } i \text{ at time } t \end{cases} \tag{4}$$

where $\overline{w}_{i,t}$ and $\overline{o}_{j,t}$ are the outputs of Actor-BERT, and $(\overline{w}_{i,t}, \overline{o}_{j,t})$ represents the combination of the two vectors into a single feature vector. We then construct a new sequence: $\dot{S}_t = [\mathcal{A}(w_{1,t}), \mathcal{A}(w_{2,t}), \ldots, \mathcal{A}(w_{i,t})]$. Another BERT network, referred to as "Critic-BERT", is used to further extract features from $\dot{S}_t$, represented as $\ddot{S}_t = \text{Critic-BERT}(\dot{S}_t)$. A self-attention mechanism and a linear layer (collectively named Critic-MLP) are then utilized to estimate the Q-value from $\ddot{S}_t$ (for detailed processing methods, refer to [17]). Furthermore, as TD3 [9] requires two critics, we employ two distinct Critic-BERT and Critic-MLP networks. These share the input features from Actor-BERT but process them separately.

3.3 Training Process

3.3.1 Stage 1: Decentralized IDDQN Training

In this stage, we aim to first train the feature-extracting capacity of the worker encoder and order encoder using a substantial number of samples. To obtain sufficient samples, we view the dispatching problem as a multi-agent scenario, where at each time step, each agent can access its own record. We adopt the independent assumption that all agents share the same policy, allowing for the sharing of records between agents and leading to a large experience replay buffer.

Since our goal in this stage is not to train a powerful model but rather to enable the feature extractor to learn its general feature-extracting capabilities, we select the simplest yet efficient method for order dispatching, namely, the IDDQN. Each worker is treated as an independent agent with the state defined as $s_{i,t} = [w_{i,t}, O_t]$ at time t. We employ a neural network to estimate the Q-value at each step as $\text{Q}^{DQN}_{\pi^Q_\Phi}(s_{i,t}, a_{i,t})$, where Φ represents the network parameters and π^Q_Φ denotes the strategy.

To construct the network, we utilize QK-attention to process the outputs of the worker encoder and order encoders to estimate the Q-value for each worker-order pair, represented as $\text{QK-Attention-Norm}(\tilde{w}_{i,t}, \tilde{o}_{j,t})$ (denoted as $y_{i,j,t}$). Although the state space encompasses the entire order state from $o_{1,t}$ to $o_{m_t,t}$, we focus on a single order $o_{j,t}$ when computing the Q-value for choosing order j. This approach aligns with previous work such as [5; 18], as the entire order state can be excessively large for a simple network to learn (our Triple-BERT effectively addresses this issue) and many networks struggle to process variable dimensional inputs (with order amounts varying at each time step). Consequently, we can compute a Q-matrix $Y_t \in \mathbb{R}^{n,m_t}$, where the element in the i-th row and j-th column, $y_{i,j,t}$, represents the Q-value of assigning order j to worker i at time t. The core strategy of IDDQN is to maximize the global Q-value, expressed as $\text{Q}(S_t, A_t) = \sum_{i=1}^{n} \text{Q}(s_{i,t}, a_{i,t})$ at each time step. To achieve this, we construct a bipartite graph where each worker and order is represented as a node. An arbitrary worker i and order j are linked by an edge weighted by the Q-value of this worker selecting this order at the current time, i.e., $y_{i,j,t}$. We then utilize Integer

Linear Programming (ILP) to solve this maximizing bipartite matching problem. (To avoid assigning orders to unavailable workers—those at full capacity or on their way to pick up an assigned order—we set the Q-value of all actions for such workers in the Q-matrix Y_t to $-\infty$.) A detailed construction of the problem is provided in Appendix B.1. For the training of IDDQN, it follows the same process of previous work [5]. Due to page limitation, we detailed it in Appendix D.1.

3.3.2 Stage 2: Centralized TD3 Training

In the standard AC framework, the process can be summarized as follows: an actor network generates actions based on the current state, represented as $A_t = \text{Actor}(S_t; \theta_A)$, while a critic network evaluates these actions using $\hat{Q}_t = \text{Critic}(S_t, A_t; \theta_C, \pi^T_{\theta_A})$. Here, θ_A and θ_C are the parameters of the actor and critic networks, respectively, and $\pi^A_{\theta_A}$ denotes the strategy of AC. During training, the critic network is updated using TD-error, similar to Q-learning, and the actor network is updated to maximize $\hat{Q}$. However, a challenge mentioned in Section 2 is that the action space is too large for the order dispatching scenario. Additionally, the actions in order dispatching are discrete, complicating optimization using TD3. To address these issues, we propose an action decomposition method along with a policy gradient-style optimization method.

Before delving into the details, we denote both θ_A and θ_C with the parameters Θ, as in our network (Fig. 2), the actor and critic share the same architecture. The trained network parameters from Stage 1, Φ, are part of Θ. Moreover, the policy of TD3 is represented as π^T_Θ.

(i) Actor: To address the large action space, we propose an action decomposition method that separates the probability of selecting each worker-order assignment combination into the probabilities of each worker choosing their respective orders. First, we expand the utility matrix M_t output by the Actor QK-Attention to $\mathcal{M}_t = [M_t, N_t] \in \mathbb{R}^{n, m_t+1}$, where N_t is an n-dimensional vector representing the utility of each worker choosing no order. This vector can be obtained by processing the output of Actor-BERT with a MLP, i.e., $N_t = \text{MLP}([\overline{w}_{1,t}, \overline{w}_{2,t}, \ldots, \overline{w}_{n,t}])$. This allows us to compute the probability of each worker choosing each action using a logit model [19], *if the actions among workers are independent*, i.e., $\mathscr{P}_t = \text{Softmax}(\mathcal{M}_t, \text{dim=-1})$. According to this independent assumption, the joint action probability can be expressed as: $\prod_{i,j \in \text{h}(A_t)} \mathscr{P}_{i,j,t}$, where h() is defined as the function $\text{h}(A_t) = \{(i,j) | a_{i,j,t} = 1\}$. Similar to stage 1, we set the probability of those unavailable workers choosing no order to 1 and all other actions to 0 in $\mathscr{P}_t$. *(Remark: We consider this independent assumption can be approximately realized after the network is well-trained, as BERT has already captured the relationships among workers, including their strategic interactions.)*

However, since an order cannot be assigned to different workers repeatedly, the actions among workers are actually not independent. Intuitively, if a worker is more willing to choose a particular action, this action should have a higher probability of being selected by this worker in the joint action. Based on this intuition, the action choosing probability can be defined as:

$$\pi^T_\Theta(A_t|S_t) = \text{z}\Big(\prod_{i,j \in \text{h}(A_t)} \mathscr{P}_{i,j,t}\Big)\,, \tag{5}$$

where $\text{z}(\cdot)$ is an increasing function that also depends on the current state S_t (which we omit for simplicity). This equation implies that if an action A_t has a higher value of $\prod_{i,j \in \text{h}(A_t)} \mathscr{P}_{i,j,t}$, it will have a higher probability of being chosen.

However, defining and computing such a function $\text{z}(\cdot)$ is challenging due to the vast action space, complicating the sampling of an action from the strategy $\pi^T_\Theta(A_t|S_t)$. We define an efficient approach to address this. First, during inference, we can greedily select the action with the maximum probability, as this action should theoretically have the highest utility:

$$\arg\max_{A_t \in \psi(S_t)} \pi^T_\Theta(A_t|S_t) = \arg\max_{A_t \in \psi(S_t)} \text{z}\Big(\prod_{i,j \in \text{h}(A_t)} \mathscr{P}_{i,j,t}\Big) = \arg\max_{A_t \in \psi(S_t)} \sum_{i,j \in \text{h}(A_t)} \log \mathscr{P}_{i,j,t}\,, \tag{6}$$

where $\psi(S_t)$ is the set of all possible actions under the current state S_t. This holds because both $\text{z}(\cdot)$ and $\log(\cdot)$ are increasing functions. We can construct a bipartite graph similar to Stage 1, where each available worker and order is represented as a node, and the link between each worker i and order j at time t is weighted by their log probability $\log \mathscr{P}_{i,j,t}$. By utilizing ILP, we can find the action A_t that maximizes $\pi^T_\Theta(A_t|S_t)$. The bipartite graph construction process is detailed in Appendix B.2. During

training, we introduce random noise to the probability matrix $\mathscr{P}_t$ and the model selects actions using the same method as in Eq. 6. When the noise is sufficiently large, the policy degrades to a totally random policy, and when the noise is zero, the policy converges to a greedy strategy. Although we cannot directly express the function $\mathrm{z}(\cdot)$, it must ensure that the function is a increasing function (since the noise is totally random). More details about the noise can be found at Appendix C.

Optimizing this probability using vanilla TD3 is challenging due to the variable action space and the gap between action probabilities and the selected action (the gradient cannot propagate through them). To address this, we employ an approximate policy gradient optimization method [10]:

$$\nabla_\Theta \mathrm{J}(\Theta) \propto \mathbb{E}_{\pi_\Theta^T}\left[(\mathrm{Q}_{\pi_\Theta^T}^{TD3}(S_t, A_t) - B)\nabla_\Theta \sum_{i,j\in \mathrm{h}(A_t)} \log \mathscr{P}_{i,j,t}\right] , \tag{7}$$

where $\mathrm{J}(\Theta)$ is the optimization objective (long-term cumulative reward), B is a baseline independent of state (we simplify by setting it to 0), and $\mathrm{Q}_{\pi_\Theta^T}^{TD3}(S_t, A_t)$ is the Q-value under the policy π_Θ^T, which can be estimated by $\mathrm{Q}_{\pi_\Theta^T,i}^{TD3}(S_t, A_t; \Theta)$ using our proposed network ($i = 1, 2$, as there are two estimated Q-values in TD3). Detailed derivations can be found in Appendix C. We then use gradient ascent to maximize $\mathrm{J}(\Theta)$, thus the loss function for the actor can be expressed as $L_A = -\nabla \mathrm{J}(\Theta)$.

(ii) Critic: For the critic, it can be updated in a manner similar to vanilla TD3, where the loss function can be expressed as:

$$\begin{aligned} L_C &= \sum_{i=1,2} \mathbb{E}_{\pi_\Theta^T}\left[\mathcal{Q}_{\pi_{\Theta^-}^T}^{TD3}(S_{t+1}, R_{t+1}; \Theta^-) - \mathrm{Q}_{\pi_\Theta^T,i}^{TD3}(S_t, A_t; \Theta)\right] , \\ \mathcal{Q}_{\pi_{\Theta^-}^T}^{TD3}(S_{t+1}, R_{t+1}; \Theta^-) &= R_{t+1} + \gamma \min_{i=1,2} \mathrm{Q}_{\pi_{\Theta^-}^T,i}^{TD3}(S_{t+1}, \mathrm{Actor}(S_{t+1}; \Theta^-, \xi); \Theta^-) , \end{aligned} \tag{8}$$

where $\mathcal{Q}_{\pi_{\Theta^-}^T}^{TD3}$ is the learning target function, Θ^- represents the parameters of the target network, which updates more slowly than the policy network Θ to provide a stable target, and ξ is a small random noise applied in the probability matrix $\mathscr{P}$. More details can be viewed in Appendix D.2.

4 Experiment

Table 1: Comparison of Different Ride Sharing Methods

Method	**DeepPool** [1]	**BMG-Q** [5]	**HIVES** [7]	**Enders et al.** [20]	**CEVD** [21]	**Triple-BERT**
Type	Independent		CTDE		Centralized	
RL Algorithm	IDDQN [22]	IDDQN [22]	QMIX [23]	MASAC [24]	VD[1] [25]	TD3 [9]
Multi-Agent	✓	✓	✓	✓	✓	×
Network Backbone	MLP	GAT [26]	GRU [27]	MLP+Attention	MLP	BERT [11]
Model Size	**20K**	117K	16M	118K	23K	16M
GPU Occupation (GB)	**3.97**	4.28	6.01	8.19	21.45	8.03
Average Reward (10^3)	12.72	13.04	12.37	12.04	13.16	**14.73**

To validate the proposed method, we evaluate its performance in the ride sharing dispatching task using real-world yellow ride-hailing data from Manhattan, New York City[2] [28]. To illustrate the efficiency and superiority of our proposed Triple-BERT, we compare it with several previous ride sharing methods of different types, including Independent MARL, CTDE MARL, and Centralized MARL, as shown in Table 1. Detailed information regarding our experiment configuration, simulator setup, and a comprehensive description of the comparative experiment can be found in Appendix E.

As shown in Fig. 3, we first illustrate the training process of different models by evaluating their performance in the training scenario every 10 episodes. The six sub-figures depict the cumulative reward, the number of orders served, and the average delivery time, detour time, pickup time, and confirmation time for each order. Additionally, the Greedy method serves as a baseline, where orders

[1]The original VD is a CTDE method. However, the CEVD variant modifies it to a centralized version.

[2]`https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`

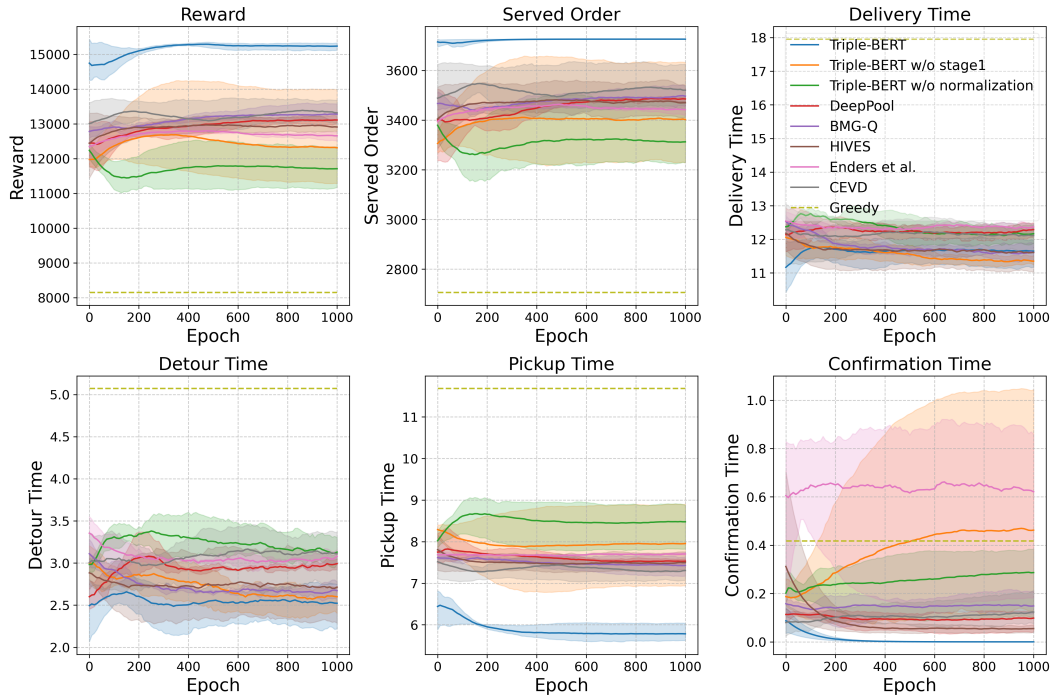


Figure 3: Training Process: Each method is trained three times, and the curve is smoothed using Exponential Moving Average (EMA) with $\alpha = 0.1$. The shaded area represents the range of fluctuations, while the solid line indicates the average value. (Here, for delivery time and detour time, only completed orders are counted, as these metrics are uncertain for unfinished orders.)

are assigned to the nearest worker. It is evident that our method outperforms the other models in most metrics, with the cumulative reward exceeding that of the best alternative method by approximately 15%. The highest number of served orders indicates that our method achieves better cooperation among workers. We then evaluate these methods over different periods, and the average rewards are shown in Table 1, where our method also demonstrates the best performance. More details about the experimental results can be found in Appendix E.4.

To further demonstrate the model's efficiency, we conduct a series of ablation studies. In terms of model training, we compare the performance of the model with and without stage 1 pre-training. Regarding the network structure, we primarily compare the QK-Attention mechanism with and without the proposed positive normalization module. The detailed results are shown in Fig. 3. We observe that without stage 1 pre-training, the model fails to converge and exhibits significant fluctuations. Particularly in the later stages, the reward begins to decrease, which can be attributed to the lack of samples. Additionally, without the proposed normalization in QK-Attention, the model performs poorly, underperforming compared to all other methods. This is due to parameter redundancy, which leads to substantial fluctuations and hinders efficient learning.

5 Conclusion

In this work, we propose the first centralized SARL method, Triple-BERT, for large-scale order dispatching in ride-hailing platforms. Our method successfully addresses the challenge of large action spaces through an action decomposition technique and tackles the issue of sample scarcity with a proposed two-stage training method. The novel network also addresses the large observation space challenge by leveraging the self-attention mechanism of BERT. Additionally, we introduce an improved QK-Attention mechanism to reduce the computational complexity of order dispatching. Through experiments on real-world ride sharing data, we demonstrate that our method significantly outperforms conventional MARL methods, achieving better cooperation among drivers.

References

[1] A. O. Al-Abbasi, A. Ghosh, and V. Aggarwal, "Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4714–4727, 2019.

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

[3] D. Wang, Q. Wang, Y. Yin, and T. Cheng, "Optimization of ride-sharing with passenger transfer via deep reinforcement learning," *Transportation Research Part E: Logistics and Transportation Review*, vol. 172, p. 103080, 2023.

[4] Z. Zhang, L. Yang, J. Yao, C. Ma, and J. Wang, "Joint optimization of pricing, dispatching and repositioning in ride-hailing with multiple models interplayed reinforcement learning," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[5] Y. Hu, S. Feng, and S. Li, "Bmg-q: Localized bipartite match graph attention q-learning for ride-pooling order dispatch," *arXiv preprint arXiv:2501.13448*, 2025.

[6] Y. Wang, J. Wu, H. Sun, Y. Lv, and J. Zhang, "Promoting collaborative dispatching in the ride-sourcing market with a third-party integrator," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 7, pp. 6889–6901, 2024.

[7] J. Hao and P. Varakantham, "Hierarchical value decomposition for effective on-demand ride-pooling," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 580–587, 2022.

[8] J. Wang, Q. Hao, W. Huang, X. Fan, Z. Tang, B. Wang, J. Hao, and Y. Li, "Dyps: Dynamic parameter sharing in multi-agent reinforcement learning for spatio-temporal resource allocation," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3128–3139, 2024.

[9] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[10] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

[12] Z. Zhao, T. Chen, Z. Cai, X. Li, H. Li, Q. Chen, and G. Zhu, "Crossfi: A cross domain wi-fi sensing framework based on siamese network," *IEEE Internet of Things Journal*, 2025.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] T. Chen, Y. Wang, H. Chen, Z. Zhao, X. Li, N. Piovesan, G. Zhu, and Q. Shi, "Modelling the 5g energy consumption using real-world data: Energy fingerprint is all you need," *arXiv preprint arXiv:2406.16929*, 2024.

[15] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, *et al.*, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, vol. 1, p. 3, 2022.

[16] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, pp. 1995–2003, PMLR, 2016.

[17] Y.-H. Chou, I.-C. Chen, J. Ching, C.-J. Chang, and Y.-H. Yang, "Midibert-piano: Large-scale pre-training for symbolic music classification tasks," *Journal of Creative Music Systems*, vol. 8, no. 1, 2024.

[18] Y. Hu, T. Dong, and S. Li, "Coordinating ride-pooling with public transit using reward-guided conservative q-learning: An offline training and online fine-tuning reinforcement learning framework," *arXiv preprint arXiv:2501.14199*, 2025.

[19] D. McFadden, "Conditional logit analysis of qualitative choice behavior," 1972.

[20] T. Enders, J. Harrison, M. Pavone, and M. Schiffer, "Hybrid multi-agent deep reinforcement learning for autonomous mobility on demand systems," in *Learning for Dynamics and Control Conference*, pp. 1284–1296, PMLR, 2023.

[21] A. Bose, H. Jiang, P. Varakantham, and Z. Ge, "On sustainable ride pooling through conditional expected value decomposition," in *ECAI 2023*, pp. 295–302, IOS Press, 2023.

[22] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[23] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *Journal of Machine Learning Research*, vol. 21, no. 178, pp. 1–51, 2020.

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, Pmlr, 2018.

[25] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2085–2087, 2018.

[26] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48550, 2017.

[27] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[28] N. Y. C. Taxi and L. Commission, "Nyc taxi and limousine commission-trip record data nyc.."

[29] 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.

[30] D. Luxen and C. Vetter, "Real-time routing with openstreetmap data," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, (New York, NY, USA), pp. 513–516, ACM, 2011.

Appendix Contents

A Action Space Size

The action space in our order dispatching task is given by:

$$|A_t| = \sum_{k=0}^{m_t} \mathrm{C}(m_t, k)\mathcal{P}(n, k) = \sum_{k=0}^{m_t} \frac{m_t!}{k!(m_t-k)!}\frac{n!}{(n-k)!}\ , \tag{9}$$

where $\mathcal{P}(n, k)$ represents the permutations of assigning k orders to n workers and $\mathrm{C}(m_t, k)$ represents the combinations of selecting k orders from the total m_t orders. This equation is based on two assumptions: (i) the platform will assign an arbitrary number of orders at each step (some orders yielding negative income will be declined by the platform) and (ii) the number of orders m_t is less than the number of workers n, which can always be satisfied since m_t represents the order count at only one timestep. Then we can derive the lower bound of $|A_t|$ as:

$$\begin{aligned} |A_t| &= \sum_{k=0}^{m_t} \mathrm{C}(m_t, k)\frac{n!}{(n-k)!} \geq \sum_{k=0}^{m_t} \mathrm{C}(m_t, k)(n-k+1)^k \\ &\geq \sum_{k=0}^{m_t} \mathrm{C}(m_t, k)(n-m_t+1)^k = (n-m_t+2)^{m_t} \geq 2^{m_t} \quad (n \geq m_t \geq 0)\ . \end{aligned} \tag{10}$$

As a result, the action space has a lower bound with the exponent to m_t. Consider the example in Section 2 where the number of workers n is 1000 and the number of orders m_t is 10. In this case, the expression $(n - m_t + 2)^{m_t}$ evaluates to $992^{10} \approx 10^{30}$.

B BiParite Graph Construction

B.1 IDDQN Bipartite Graph

The bipartite graph in the IDDQN-based order dispatching method is constructed as follows:

$$\max_{A_t} \sum_{i\in\mathcal{I}} a_{i,j,t} \cdot y_{i,j,t}, \tag{11a}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{I}} a_{i,j,t} \leq 1, \quad \forall j \in \mathcal{J}_t, \tag{11b}$$

$$\sum_{j\in\mathcal{J}_t} a_{i,j,t} \leq 1, \quad \forall i \in \mathcal{I}, \tag{11c}$$

$$a_{i,j,t} \in \{0, 1\}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}_t, \tag{11d}$$

where $a_{i,j,t}$ is the action representing whether worker i is assigned order j at time t (with 1 indicating assignment and 0 indicating no assignment), $y_{i,j,t}$ denotes the Q-value of worker i choosing order j at time t (with $y_{i,j,t} = -\infty$ for all unavailable workers at time t), $\mathcal{I}$ is defined as $\{1, 2, \ldots, n\}$, and the set $\mathcal{J}_t$ is defined as $\{1, 2, \ldots, m_t\}$. Constraint 11b ensures that an order can be assigned to at most one worker, while constraint 11c guarantees that each worker is assigned at most one order.

B.2 TD3 Bipartite Graph

The bipartite graph in our proposed TD3-based order dispatching method is constructed as follows:

$$\max_{X_t} \sum_{i\in\mathcal{I}_t^w} x_{i,j,t} \cdot \log \mathscr{P}_{i,j,t}, \tag{12a}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{I}} x_{i,j,t} \leq 1, \quad \forall j \in \mathcal{J}_t, \tag{12b}$$

$$\sum_{j\in\mathcal{J}_t} x_{i,j,t} = 1, \quad \forall i \in \mathcal{I}_t^w, \tag{12c}$$

$$x_{i,j,t} \in \{0, 1\}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}_t \cup \{m_t + 1\}, \tag{12d}$$

where $\mathcal{I}_t^w$ represents the set of available workers at time t. Here, constraint 12b does not apply in the last column, as it represents declining all orders, an action that can be chosen by any worker. Constraint 12c requires each row to equal 1, ensuring that each worker must either take an order or reject all, without other choices. We can then convert X_t to action A_t as follows:

$$a_{i,t} = \begin{cases} x_{i,j,t} & \text{if } i \in \mathcal{I}_t^w \text{ and } x_{i,m_t+1,t} = 0 \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{13}$$

C Policy Gradient Proof

According to the policy gradient theory [10], we have:

$$\begin{aligned}
&\nabla_\Theta \mathrm{J}(\Theta) \\
&\propto \mathbb{E}_{\pi_\Theta^T}\left[\left(\mathrm{Q}_{\pi_\Theta^T}^{TD3}(S_t, A_t) - B\right)\nabla_\Theta \log \pi_\Theta^T(A_t|S_t)\right] \\
&= \mathbb{E}_{\pi_\Theta^T}\left[\left(\mathrm{Q}_{\pi_\Theta^T}^{TD3}(S_t, A_t) - B\right)\nabla_\Theta \log \mathrm{z}\left(\prod_{i,j\in \mathrm{h}(A_t)} \mathscr{P}_{i,j,t}\right)\right] \\
&= \mathbb{E}_{\pi_\Theta^T}\left[\left(\mathrm{Q}_{\pi_\Theta^T}^{TD3}(S_t, A_t) - B\right)\frac{d\mathrm{z}(\prod_{i,j\in \mathrm{h}(A_t)} \mathscr{P}_{i,j,t})}{d\prod_{i,j\in \mathrm{h}(A_t)} \mathscr{P}_{i,j,t}}\frac{\prod_{i,j\in \mathrm{h}(A_t)} \mathscr{P}_{i,j,t}}{\mathrm{z}(\prod_{i,j\in \mathrm{h}(A_t)} \mathscr{P}_{i,j,t})}\nabla_\Theta \log \prod_{i,j\in \mathrm{h}(A_t)} \mathscr{P}_{i,j,t}\right] \\
&= \mathbb{E}_{\pi_\Theta^T}\left[\left(\mathrm{Q}_{\pi_\Theta^T}^{TD3}(S_t, A_t) - B\right)\mathcal{E}_{\mathrm{z}(x),x}|_{x=\prod_{i,j\in \mathrm{h}(A_t)} \mathscr{P}_{i,j,t}}\nabla_\Theta \sum_{i,j\in \mathrm{h}(A_t)} \log \mathscr{P}_{i,j,t}\right],
\end{aligned} \tag{14}$$

where $\mathcal{E}$ denotes elasticity, which measures the sensitivity of one variable to changes in another, and is defined as:

$$\mathcal{E}_{y,x} = \frac{d\log y}{d\log x} = \frac{dy}{dx}\frac{x}{y}. \tag{15}$$

Since $\mathrm{z}(x)$ is an increasing function, the elasticity is always non-negative. Here, we assume that the elasticity of $\mathrm{z}(x)$ with respect to x can be approximately viewed as a positive constant. Thus, we have: $\nabla_\Theta \mathrm{J}(\Theta) \propto \mathbb{E}_{\pi_\Theta^T}\left[\left(\mathrm{Q}_{\pi_\Theta^T}^{TD3}(S_t, A_t) - B\right)\nabla_\Theta \sum_{i,j\in \mathrm{h}(A_t)} \log \mathscr{P}_{i,j,t}\right]$, corresponding to Eq. 7.

We acknowledge that the elasticity may not be a positive constant in practice (this requires that $z(x)$ has the same form as ax^b $(a, b > 0)$). However, we consider this a reasonable approximation; otherwise, optimizing the actor would not be feasible, as obtaining a closed-form solution for $z(x)$ is impossible. Additionally, the final form of the equation aligns with the intuition that if an action has a higher Q-value, we should increase its probability, whereas we should decrease its probability if the Q-value is lower. While this approach may impede the model's convergence to the optimal solution, experimental results demonstrate the effectiveness of this formula, showing that it significantly outperforms other MARL methods.

As mentioned in Section 3.3.2, during training, we add random noise to $\mathscr{P}_t$ and then choose the action that maximizes $\sum_{i,j\in \mathrm{h}(A_t)} \log \mathscr{P}_{i,j,t}$. Currently, the mapping from $\sum_{i,j\in \mathrm{h}(A_t)} \log \mathscr{P}_{i,j,t}$ to the choosing probability π_Θ^T corresponds to $\mathrm{z}(\cdot)$. To further illustrate the robustness of our method, we compare the performance of our model using Gaussian noise, uniform noise, and binary symmetric channel (BSC) noise, where the noise follows a Bernoulli distribution and has been widely utilized in previous work [5; 18]. During training, we gradually reduce the noise to make the policy more deterministic. The experimental results are shown in Fig. 4, where we observe that, despite certain performance differences between the various types of noise, they all outperform conventional MARL methods. This suggests the efficiency and high robustness of our proposed method, indicating that the detailed expression of $\mathrm{z}(x)$ does not significantly influence the validation of the method based on Eq. 7, even if it may cause some performance gaps. The optimal noise for our task may require further exploration. *For fairness, we choose to use BSC noise when comparing with other methods, even though it appears to perform the worst among the three types of noise. We aim to demonstrate that our results are robust and superior, not relying on a particular choice of hyper-parameters or experiment scenarios.*

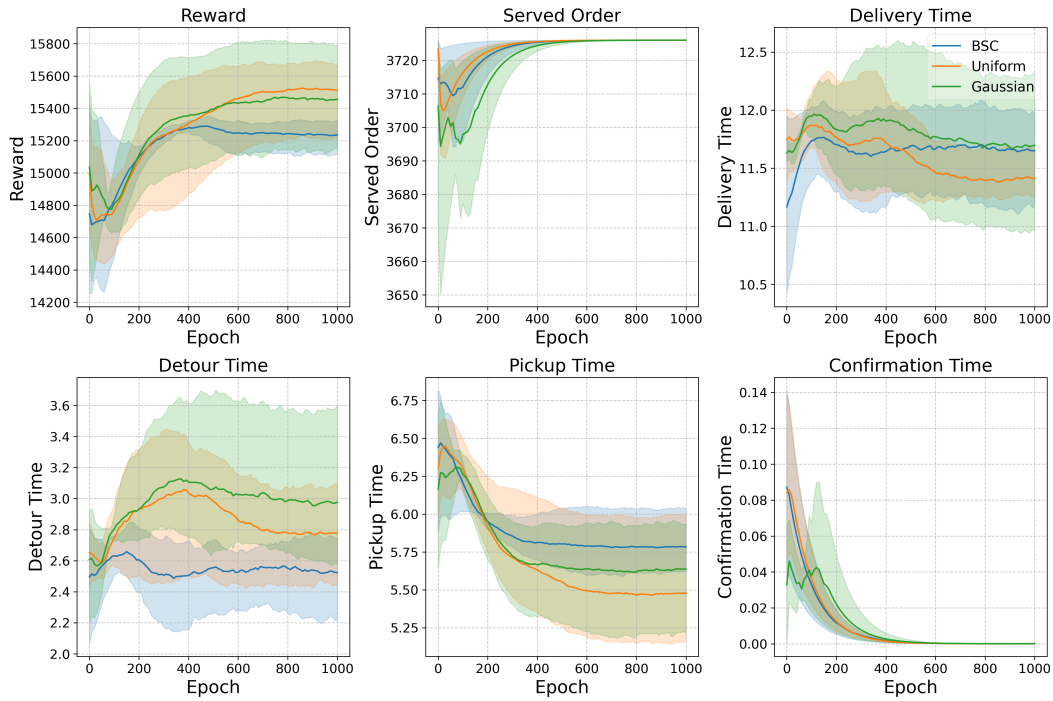


Figure 4: Comparison Between Different Noise Methods

D Training Process

D.1 Stage 1: IDDQN Algorithm

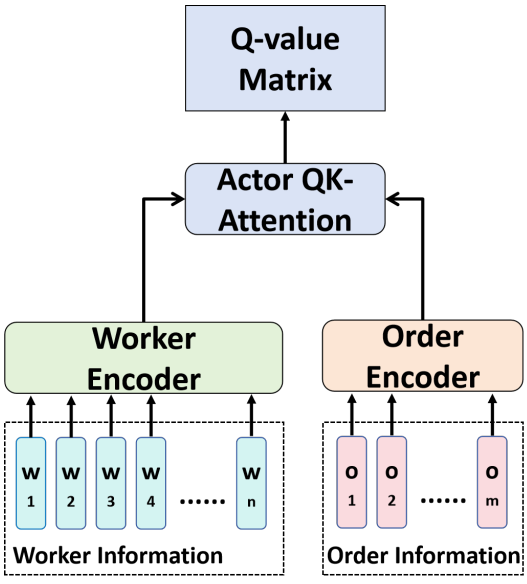


Figure 5: Network Structure in Stage 1

In stage 1, the network structure is shown as Fig. 5, which is consisted by the encoders and the QK-Attention module of proposed network in Fig. 2. *Remark: Although the model takes the entire worker and order sequence as input, it primarily aims to utilize parallel computation to enhance computational efficiency. In the encoders, each worker and order's information is processed separately. Similarly, in the QK-Attention module, the Q-value for each worker-order pair is computed*

independently. It is also feasible to input only a single worker-order pair into this network, computing the Q-value exclusively for that pair; however, this would increase the computation time.

During IDDQN training, we need to introduce some noise into the Q-matrix Y_t to facilitate sufficient exploration. Specifically, for the ϵ-greedy strategy, we randomly select a proportion ϵ of non-$-\infty$ elements in Y_t and set them to a large positive number $\overline{Y}$ to enhance their likelihood of being selected. We then update the neural network by minimizing the TD-error, expressed as:

$$\begin{aligned} L_Q &= \mathbb{E}_{\pi_\Phi^Q}\left[\mathcal{Q}^{DQN}_{\pi^Q_{\Phi^-}}(s_{i,t+1}, r_{i,t+1}; \Phi^-) - \mathrm{Q}^{DQN}_{\pi^Q_\Phi}(s_{i,t}, a_{i,t}; \Phi)\right], \\ \mathcal{Q}^{DQN}_{\pi^Q_{\Phi^-}}(s_{i,t+1}, r_{i,t+1}; \Phi^-) &= r_{i,t+1} + \gamma \mathrm{Q}^{DQN}_{\pi^Q_{\Phi^-}}(s_{i,t+1}, \kappa_{i,t+1}; \Phi^-), \\ \kappa_{i,t+1} &= \arg \max_{\kappa_{i,t+1} \in \psi_{i,t+1}} \mathrm{Q}^{DQN}_{\pi^Q_\Phi}(s_{i,t+1}, \kappa_{i,t+1}; \Phi), \end{aligned} \tag{16}$$

where $\mathcal{Q}^{DQN}_{\pi^Q_{\Phi^-}}$ is the learning target function, γ is the discount factor, $\psi_{i,t+1}$ is the possible action space for worker i at time $t+1$, and Φ^- represents the parameters of the target network, which are updated at a slower pace compared to the policy network to provide a stable target for training. After each training iteration, the target network is updated in a soft manner: $\Phi^- := \tau\Phi + (1-\tau)\Phi^-$, where τ is the update rate.

The detailed process is illustrated in Algorithm 1, where $\mathbf{1}_j$ represents the vector that only the j^{th} position is 1 and other positions are 0.

Algorithm 1 IDDQN Training Process

Require: Number of training episodes E, number of training steps T, mini-batch size m, target update rate τ, exploration noise ϵ, final exploration ϵ_f, exploration decay δ, discount factor γ, model parameters Φ

Initialize target networks $\Phi^- \leftarrow \Phi$
Initialize replay buffer $\mathcal{B}$
for $k = 1$ to E **do**
  for $t = 1$ to T **do**
    Calculate Q-value matrix Y_t: $y_{i,j,t} = \mathrm{Q}^{DQN}_{\pi^Q_\Phi}(s_{i,t}, \mathbf{1}_j; \Phi)$
    Select action with exploration noise:$A_t = \mathrm{ILP}(Y_t, \epsilon)$
    Observe reward $r_{i,t+1}$ and new state $s_{i,t+1}$ for each worker i
    Store transition $(s_{i,t}, a_{i,t}, r_{i,t+1}, s_{i,t+1})$ in $\mathcal{B}$
    Sample mini-batch of m transitions (s, a, r, s') from $\mathcal{B}$
    Compute target Q-value:
    $y \leftarrow r + \gamma \mathrm{Q}^{DQN}_{\pi^Q_{\Phi^-}}(s_{i,t+1}, \arg\max_{\kappa_{i,t+1} \in \psi_{i,t+1}} \mathrm{Q}^{DQN}_{\pi^Q_\Phi}(s_{i,t+1}, \kappa_{i,t+1}; \Phi); \Phi^-)$
    Update critics: $\Phi \leftarrow \arg\min_\Phi \frac{1}{m}\sum(y - \mathrm{Q}^{DQN}_{\pi^Q_\Phi}(s, a; \Phi))^2$
    Update target networks: $\Phi^- \leftarrow \tau\Phi + (1-\tau)\Phi^-$
  end for
  Decay exploration: $\epsilon \leftarrow \max(\epsilon_f, \epsilon\delta)$
end for

D.2 Stage 2: TD3 Algorithm

The process of our Stage 2 - TD3 training is illustrated in Algorithm 2. In experiment, we follow the vanilla TD3 approach of updating the actor once after updating the critic twice.

Algorithm 2 TD3 Training Process

Require: Number of training episodes E, number of training steps T, mini-batch size m, policy delay d, target update rate τ, exploration noise ϵ, final exploration ϵ_f, exploration decay δ, target policy smoothing noise ξ, discount factor γ, model parameters Θ

Initialize target networks $\Theta^- \leftarrow \Theta$
Initialize replay buffer $\mathcal{B}$
for $k = 1$ to E **do**
  for $t = 1$ to T **do**
    Select action with exploration noise: $A_t = \text{Actor}(S_t; \Theta, \epsilon)$
    Observe reward R_{t+1} and new state S_{t+1}
    Store transition $(S_t, A_t, R_{t+1}, S_{t+1})$ in $\mathcal{B}$
    Sample mini-batch of N transitions (S, A, R, S') from $\mathcal{B}$
    Compute target action with smoothing noise: $A' \leftarrow \text{Actor}(S; \Theta^-, \xi)$
    Compute target Q-value: $y \leftarrow r + \gamma \min_{i=1,2} \mathrm{Q}^{TD3}_{\pi^T_{\Theta^-},i}(S', A'; \Theta^-)$
    Update critics: $\Theta \leftarrow \arg\min_\Theta \frac{1}{m} \sum[(y - \mathrm{Q}^{TD3}_{\pi^T_\Theta,1}(S, A; \Theta))^2 + (y - \mathrm{Q}^{TD3}_{\pi^T_\Theta,2}(S, A; \Theta))^2]$
    if $t \mod d == 0$ **then**
      Update actor using deterministic policy gradient:
      $\nabla J(\Theta) = \frac{1}{m} \sum(\mathrm{Q}^{TD3}_{\pi^T_\Theta,1}(S, A; \Theta) - B) \nabla_\Theta \log \pi^T_\Theta(A_t|S_t), \quad (A = \text{Actor}(S; \Theta))$
      Update target networks: $\Theta^- \leftarrow \tau\Theta + (1 - \tau)\Theta^-$
    end if
  end for
  Decay exploration: $\epsilon \leftarrow \max(\epsilon_f, \epsilon\delta)$
end for

E Experiment Details

E.1 Experiment Configurations

Our model was trained using the PyTorch framework [29] on a workstation running Windows 11, equipped with an Intel(R) Core(TM) i7-14700KF processor and an NVIDIA RTX 4080 graphics card. The detailed model configurations are shown as Table 2. During the training phase, the model utilized approximately 8.03 GB of GPU memory. For optimization, we employed the Adam optimizer with an initial learning rate of 10^{-4} and a decay rate of 0.99. In Stage 1, the batch size was set to 256, while in Stage 2, it was reduced to 16, due to a sharp decrease in sample amount. Additionally, optimization was performed once every 4 time steps, and in Stage 2, the actor was updated once for every two updates of the critic.

Table 2: Model Configurations

Configuration	**Our Setting**
Hidden Dimension	64 (Actor) / 128 (Critic)
Attention Heads	4
BERT Layers	3 for Each
Dropout Rate	0.1
Optimizer	Adam
Learning Rate	10^{-4}
Scheduler	ExponentialLR
Learning Rate Decay	0.99
Batch Size	256 (Stage 1) / 16 (Stage 2)
Exploration Rate	$0.99 \rightarrow 0.0005$
Updating Rate of Target Network	0.005
Discount Factor	0.99

E.2 Simulation Setup

In the simulation, we set the total number of drivers to 1,000, with each car having a capacity of 3 passengers. Each episode lasts 30 minutes, divided into 30 time steps, where each step determines the operations for the subsequent minute. For the TSP route optimization and time estimation, we utilize the OSRM simulator [30], with a default traveling speed of 60 km/h. We train the model using data from 19:00 to 19:30 on July 17, 2024, which includes 3,726 valid orders, and we test the trained model during other time periods on July 17, 2024, including 14:00-14:30 (2,850 valid orders), 17:00-17:30 (3,577 valid orders), 20:00-20:30 (3,114 valid orders), 21:00-21:30 (4,264 valid orders), and 22:00-22:30 (4,910 valid orders), where the order amount range from 2,850 to 4,264.

E.3 Introduction of Comparative Methods

The methods using in our comparative experiment can be mainly divided into three categories:

- **Independent MARL:** The DeepPool [1] and BMG-Q [5] utilize a similar IDDQN method as described in Section 3.3.1, with BMG-Q employing GAT [26] to capture the relationships among neighboring agents. Additionally,in the original paper fo DeepPool, the authors used CNN. However, due to differences in the observation space of our task, we replaced it with MLP.
- **Centralized Training Decentralized Execution (CTDE):** The HIVES [23] framework introduces a QMIX [23] based method to address the shortcomings of IDDQN, specifically the inadequacy of treating the global Q-value as a simple summation of the individual Q-values of each agent. Enders et al. [20] propose a MASAC [24] based approach, allowing each driver to choose whether to accept an order, thereby preventing low-profit orders from negatively impacting the global income.
- **Centralized Training and Centralized Execution (CTCE):** CEVD [21], based on VD [25], innovatively combines the Q-values of each agent with those of their neighbors to create a new type of Q-value, akin to the motivation behind BMG-Q.

Overall, most of these methods attempt various strategies to enhance each agent's awareness of the global state, facilitating better cooperation. In contrast, our method directly transforms the formulation into a centralized single-agent reinforcement learning approach.

It is noteworthy that these Independent and CTDE MARL dispatching methods differ slightly from general MARL methods. In order dispatching, one order cannot be assigned to multiple workers, making it necessary to employ some centralized mechanism to achieve this. We refer to them as independent MARL and CTDE methods because they can directly calculate their own Q-values or action probabilities using their own or neighboring states. Conversely, CEVD must calculate the primary Q-value of each agent separately and then combine those primary Q-values with their neighbors to obtain a final Q-value for each agent.

Through the experimental results in Fig. 3, we observe that DeepPool [1], serving as one of the earliest benchmarks, demonstrates relatively stable and good performance, suggesting the simplicity and effectiveness of IDDQN features. In contrast, BMG-Q [5] significantly improves performance by utilizing FAT to capture neighboring information. As for HIVES [23] and CEVD [21], while they exhibit relatively good performance in the early stages of training—likely due to their hierarchical structure and centralized training methods—their performance becomes unstable in later stages, with rewards even starting to decrease. This instability may stem from the hierarchical approach not adequately addressing the large network input of the mixture network in QMIX and the lazy agent problem in VD. Additionally, their centralized training approach faces the same data scarcity issues as our method, making convergence more challenging. For Enders et al. [20], we note that their method shows worse performance than others. This may be related to their state processing method during training, where they replace the next state in the replay buffer with the request state from the current state to maintain a consistent agent count across two successive time steps, which appears to be a strong assumption. Finally, for the last three methods, their original papers primarily focus their reward functions on the serving order amount, without incorporating additional terms like ours (which also considers income, outcome, and user satisfaction levels). This makes our scenario more complex and may further reduce the performance of their methods in our setting.

E.4 Additional Experiment Result

The detailed experimental results across different time periods are shown in Fig. 6, while the weighted average numerical results are presented in Table 3. For each model in each scenario, we repeat the experiment three times, and the error bars in the figure represent the standard deviation. We observe that our Triple-BERT achieves the highest reward across all scenarios, with the advantage becoming more pronounced as the order volume increases. Triple-BERT primarily optimizes the service rate and pickup time, significantly outperforming other methods.

For delivery time and detour time, the figures only account for completed orders, as the status of unfinished orders is uncertain, which may introduce some bias in the detailed values. In terms of these two metrics, Triple-BERT clearly performs better in high order volume scenarios, but not in low order volume scenarios. This may be due to the relatively low conflict caused by MARL in low order scenarios, while in high order scenarios, both the observation and action spaces increase sharply, making it challenging for MARL to find optimal solutions.

Lastly, we note that our method and the approach by Enders et al. [20] exhibit higher confirmation times. This may be attributed to both methods having an explicit rejection action (i.e., choosing no order), unlike the other methods. While this mechanism can lead to higher confirmation times, it also enables the model to discard negative profit orders and reserve some orders for currently unavailable workers.

Table 3: Average Performance under Multiple Periods

Method	Reward	Service Rate	Delivery Time	Detour Time	Pickup Time	Confirmation Time
DeepPool [1]	12723.85	0.91	11.53	2.47	7.77	0.06
BMG-Q [5]	13036.29	0.92	**10.57**	**1.90**	7.61	0.10
HIVES [7]	12365.11	0.89	11.04	2.28	7.99	**0.03**
Enders et al. [20]	12041.62	0.90	12.28	2.90	7.94	0.80
CEVD [21]	13157.96	0.94	11.36	2.31	7.37	0.06
Triple-BERT	**14730.48**	**0.98**	11.53	2.52	**5.73**	0.13
w/o stage 1	10665.02	0.87	11.92	2.72	9.36	0.68
w/o normalization	10839.33	0.88	12.50	2.85	9.10	0.24

F Discussions

F.1 Limitations and Future Works

The limitations of this paper can be mainly categorized into two parts.

First, regarding the theoretical aspect, the current policy gradient formula is an approximation where we assume that the probability mapping function $\mathrm{z}(x)$ has a nearly constant elasticity with respect to the independent variable x. Since obtaining a closed-form solution or elasticity for $\mathrm{z}(x)$ is impossible, we must make certain assumptions for optimization. Although we have demonstrated the efficiency of Eq. 7 through intuition and experiments, there may still be a gap between the model's performance and the optimal solution. In future work, it would be valuable to explore an action strategy that can be proven to have elasticity to $\mathrm{z}(x)$ close to a constant.

Second, concerning the experimental aspect, due to limitations of the experimental setup, we currently train and evaluate the model within a 30-minute simulation window. For 1,000 episodes, we can collect only 30,000 samples in a single-agent setting, which takes about a whole day to train a single method. This is why we designed the two-stage training method; otherwise, the model would struggle to converge with the limited samples. Future exploration should address whether stage 1 training is still necessary when the sample size increases. We also intend to investigate the model's performance in more diverse transportation scenarios, such as food delivery.

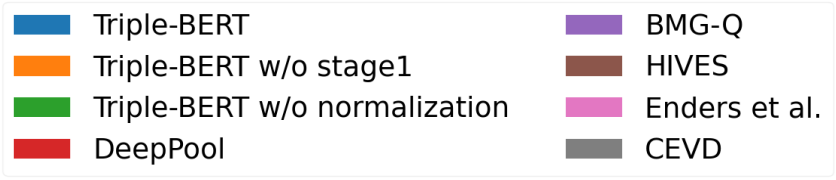
Finally, to better align with practical application scenarios and conditions, we plan to further develop the method to jointly optimize repositioning, payment, and price-setting tasks, making it more feasible for real-world use.

F.2 Societal Impacts

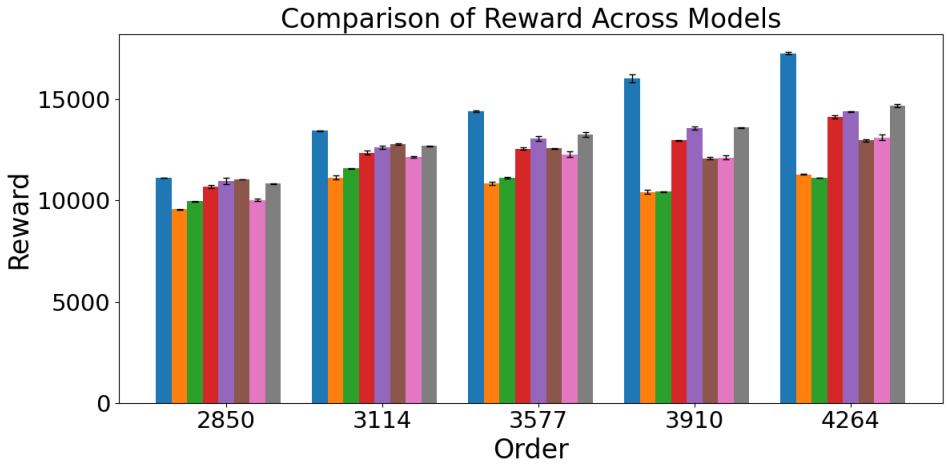
This work has potential value for both academic research and practical applications in the transportation field, particularly for large-scale order dispatching tasks. By shifting from the conventional MARL paradigm to a SARL approach, we significantly improve model performance. This technology holds promise for enhancing daily travel and logistics transport.

However, the issue of algorithmic discrimination has received widespread attention over time. Closed-box management algorithms, including those for order dispatching, have been shown to create discriminatory scenarios for workers, as reinforcement learning methods primarily aim to maximize rewards without considering ethical implications. For example, algorithms may set different payment structures or order assignment preferences based on individual features or geographical locations of workers.

We hope that our method will not exacerbate these issues and can be further developed to include constraints that promote fairness. Our goal is to strike a balance between profit and ethics, fostering a win-win situation for platforms, workers, and customers.

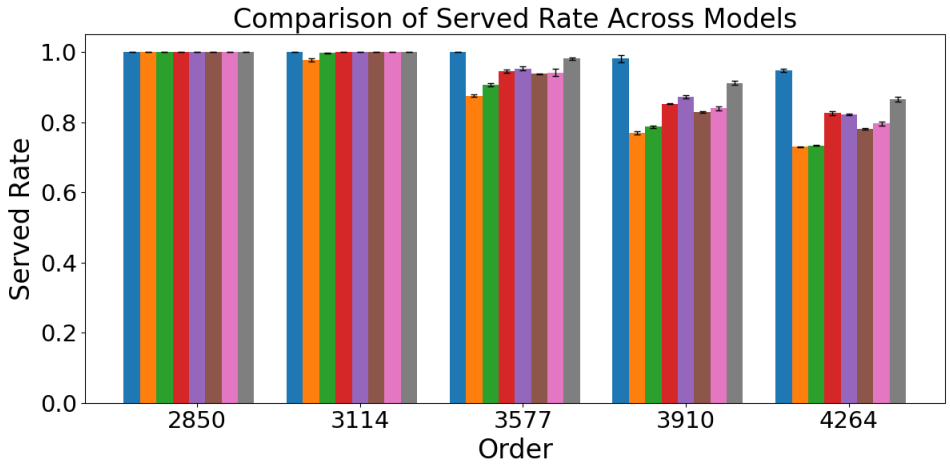


(a) Legend

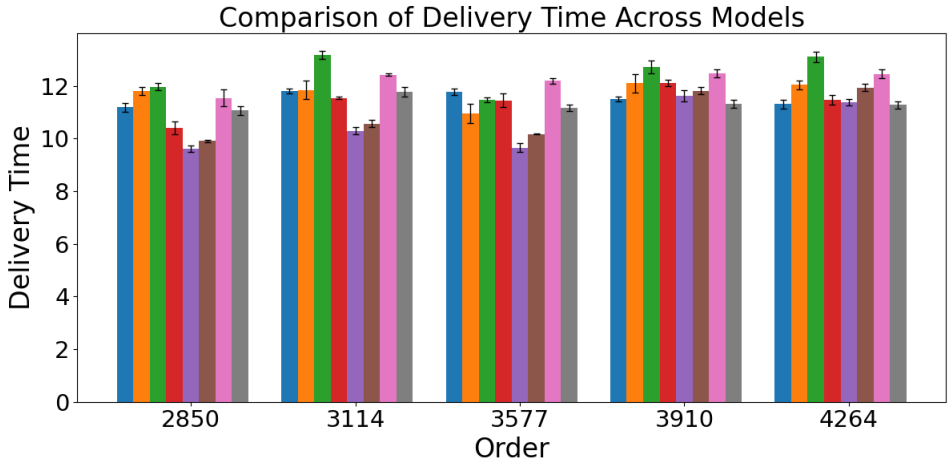


(b) Accumulative Reward

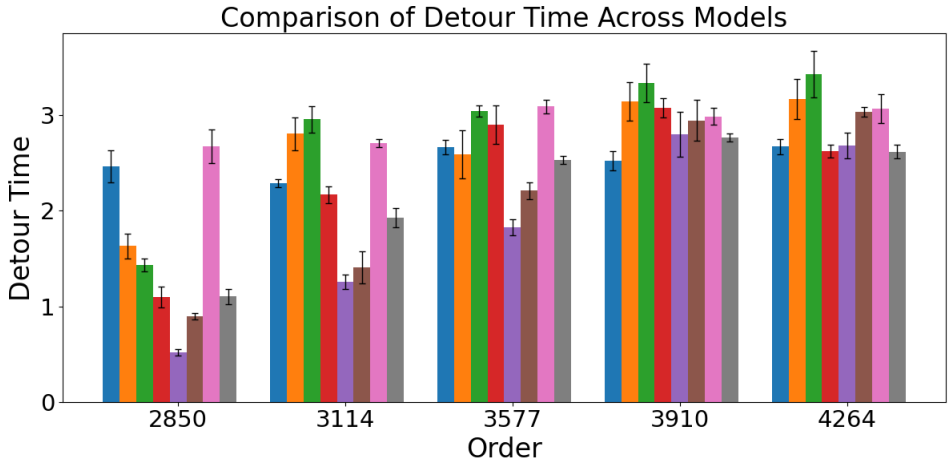
Figure 6: Detailed Evaluation Results



(c) Service Rate

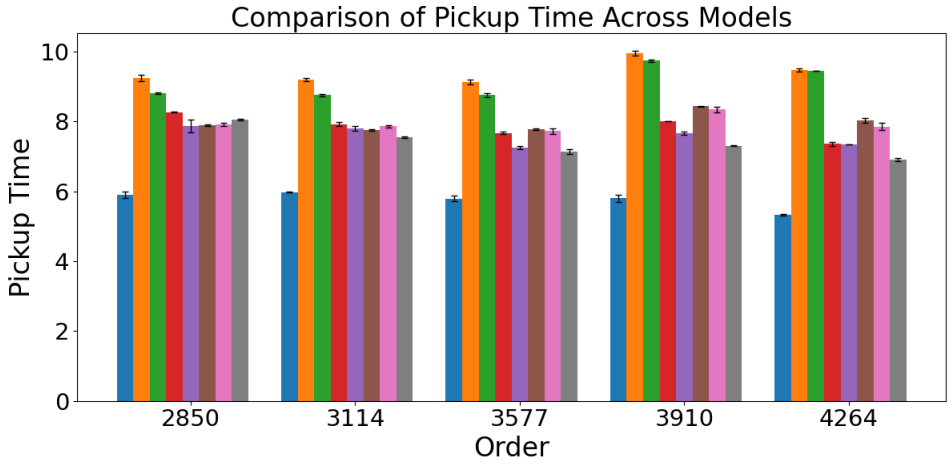


(d) Delivery Time

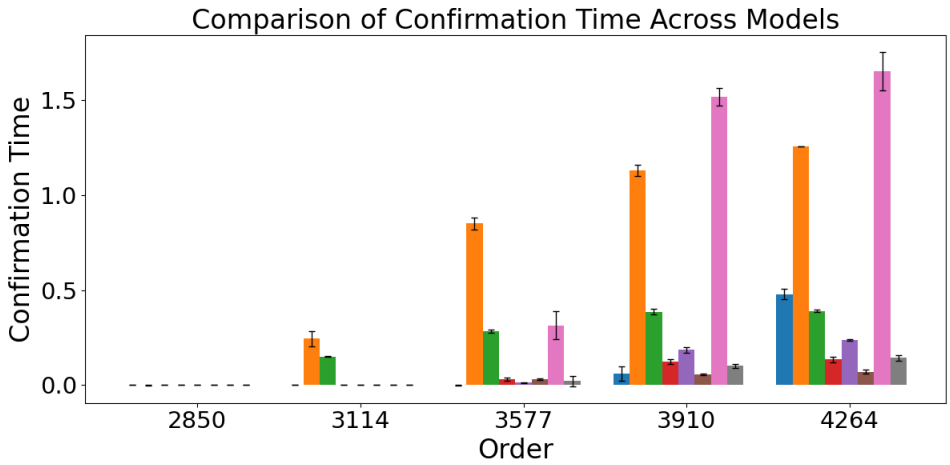


(e) Detour Time

Figure 6: Detailed Evaluation Results



(f) Pickup Time



(g) Confirmation Time

Figure 6: Detailed Evaluation Results

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions by proposing a novel centralized single-agent reinforcement learning framework for large-scale order dispatching (Section 3). The claims of its superior performance compared to conventional multi-agent methods are substantiated by the experimental results presented in Section 4.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.

- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Please refer to Appendix F.1.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All assumptions and proofs are included in Appendix A and Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

Justification: The method and implementation details are provided in Section 3, as well as in Appendix B and Appendix D. The experimental settings and environment details are described in Appendix E.1 and Appendix E.2.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code and trained model parameters are provided in an anonymous repository at `https://anonymous.4open.science/r/Triple-BERT`. The data used is from the public New York taxi dataset, available at `https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental settings and environment details are described in Appendix E.1 and Appendix E.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We repeat the experiment 3 times and report the fluctuations using shadows in Fig. 3 and Fig. 4, and the error bars in Fig. 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The compute resources are described in Appendix E.1, and the computation time and GPU usage are provided in Table 1.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: [NA]

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Please refer to Appendix F.2.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The papers corresponding to the environments used are cited in Section 4, and the toolkit employed is cited in Appendix E.1 and Appendix E.2.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: This paper uses LLMs solely to check grammar and spelling.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.