Submitted to *Transportation Science*
manuscript (Please, provide the manuscript number!)

Data-Driven Optimization for Meal Delivery: A Reinforcement Learning Approach for Order-Courier Assignment and Routing at Meituan

Ramón Auad
Amazon.com, Bellevue, WA 98004
Universidad Católica del Norte, Antofagasta 1240000, Chile

Felipe Lagos
Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Santiago de Chile 7941169, Chile

Tomás Lagos
Discipline of Business Analytics, The University of Sydney, Sydney, NSW 2000, Australia

The rapid growth of online meal delivery has introduced complex logistical challenges, where platforms must dynamically assign orders to couriers while accounting for demand uncertainty, courier autonomy, and service efficiency. Traditional dispatching methods, often focused on short-term cost minimization, fail to capture the long-term implications of assignment decisions on system-wide performance. This paper presents a novel hybrid framework that integrates reinforcement learning with hyper-heuristic optimization to improve sequential order assignment and routing decisions in meal delivery operations. Our approach combines n-step SARSA with value function approximation and a multi-armed bandit-based hyper-heuristic incorporating seven specialized low-level heuristics. Our approach explicitly models the evolving system state, enabling dispatching policies that balance immediate efficiency with future operational performance. By employing scalable linear value function approximation, we enhance policy learning in high-dimensional environments while maintaining generalization across states and actions. Using real operational data from the food delivery platform Meituan, we develop a comprehensive simulation environment that captures order dynamics, courier behavior, and service times. Through extensive computational experiments, we demonstrate that our framework significantly outperforms traditional benchmark policies, achieving 12% cost reduction through strategic order postponement. Our results reveal that the largest improvements occur during high-demand periods with courier shortages, and that a 10% increase in courier availability yields greater benefits than algorithmic improvements alone. The proposed methodology effectively balances immediate operational efficiency with long-term performance, while providing valuable insights for meal delivery platforms regarding courier fleet management and order assignment strategies.

Key words: Last-Mile Logistics; Meal Delivery; Machine Learning; Reinforcement Learning; Value Function Approximation; Metaheuristics.

1. Introduction

The past decade has witnessed profound transformations in urban logistics, particularly in the food delivery industry, driven by advances in technology, shifting consumer behavior, and the proliferation of gig-economy. Online food delivery services have become an essential component of this evolution, allowing customers to browse menus, place orders, and have meals delivered with unprecedented convenience. In 2015, the US food delivery market was valued at \$11 billion, with a projected annual growth of 16%, significantly outpacing traditional dine-in services (Morgan Stanley Research 2017). Recent projections suggest that the global food delivery market could reach \$365 billion by 2030, fueled by changing lifestyles, urbanization, and a growing preference for convenience over traditional homemade meals (Cheng 2018, Union Bank of Switzerland 2018). The COVID-19 pandemic further accelerated this shift, as lockdowns and social distancing measures made online food delivery indispensable for millions of households (Yeo 2021, Shead 2020). This surge in demand intensified competition among delivery platforms, creating an increasingly crowded and operationally complex landscape.

Despite its rapid growth, the food delivery industry faces significant operational challenges due to the high uncertainty and complexity of its logistics. Orders arrive continuously throughout the day, with demand peaks occurring within narrow time windows, necessitating rapid assignment of orders to couriers while ensuring fast and reliable service. Poor dispatching decisions can lead to increased delivery times, lower courier utilization, and degraded service quality (Auad et al. 2024b, Reyes et al. 2018, Yildiz and Savelsbergh 2019). Adding to this complexity, couriers, who often operate as independent contractors, exhibit a high degree of autonomy, making decisions that may not align with system-wide efficiency. They can reject orders, multitask across competing platforms, or take detours that introduce inefficiencies into the system (Auad et al. 2023, Ausseil et al. 2022, 2024). This decentralized structure, combined with the inherent uncertainty in demand patterns, makes achieving optimal dispatching decisions particularly challenging.

In response to these challenges, Meituan, a leading on-demand service platform in China, collaborated with the INFORMS Transportation Science and Logistics Society (TSL) to launch the First INFORMS TSL Data-Driven Research Challenge (INFORMS TSL 2024). This initiative provides researchers with an exhaustive, real-world dataset, publicly available on the official GitHub repository (Zhao et al. 2024), offering a unique opportunity to develop innovative solutions that bridge academic research and practical application in food delivery logistics.

A fundamental issue in meal delivery logistics is balancing short-term dispatching efficiency with long-term operational performance. Traditional rule-based and myopic heuristics, while computationally inexpensive, often fail to anticipate the impact of current decisions on future system conditions, leading to suboptimal outcomes. To address this challenge, we propose a novel reinforcement learning (RL)-based framework that integrates value function approximation techniques with hyper-heuristic optimization (Kheiri 2020) to guide sequential decision-making in online order assignment and routing. Our approach dynamically learns from the evolving system state, adapting to fluctuations in demand and courier availability while capturing the trade-offs between immediate cost reduction and long-term operational efficiency.

Specifically, our framework employs n-step SARSA (State-Action-Reward-State-Action) to optimize dispatching policies by explicitly considering future system states. SARSA is an on-policy control method used in RL to teach an agent how to make decisions in an environment by learning a policy. This enables assignments that are not only locally optimal, but also beneficial over extended decision horizons. To efficiently learn policies in high-dimensional, large-scale environments, we leverage a linear value function approximation, which allows for generalization across similar states and actions while maintaining computational feasibility. Additionally, we incorporate multi-armed bandit-based hyper-heuristic to further refine decision-making by selecting effective heuristic strategies dynamically.

Prior work has modeled the Restaurant Meal Delivery Problem and related capacity management problems as Markov decision processes (Ulmer et al. 2021, Auad et al. 2024b). We develop a sequential decision framework for courier-order assignment that explicitly incorporates courier autonomy and rejection behavior. This approach allows us to integrate RL (via n-step SARSA and value function approximation) with a bandit-based hyper-heuristic. This framework is specifically designed to capture the behavioral features observed in the Meituan's dataset while remaining compatible with RL methods, and provides a scalable and interpretable decision-support tool for meal delivery platforms. Through extensive computational experiments, we demonstrate that our approach outperforms benchmark policies in key performance metrics, including delivery time, courier utilization, and cost efficiency. In addition, our analysis reveals valuable business insights, highlighting the critical role of order postponement, courier connection probabilities, and dynamic dispatch strategies in optimizing meal delivery logistics.

The remainder of this paper is organized as follows. Section 2 reviews relevant literature on meal delivery and decision-making approaches. Section 3 formulates the problem as a sequential decision-making model. Section 4 details our proposed methodology, covering the value function approximation framework, hyper-heuristic optimization, and benchmark policies. Section 5 describes the data and simulation environment used in our study. Section 6 presents numerical experiments and performance evaluations, analyzing algorithmic choices and providing business insights. Finally, Section 7 discusses key findings and potential directions for future research.

2. Literature review

Meal delivery operations can be classified within the broader framework of dynamic vehicle routing problems, where routing decisions must be made continuously as new orders are placed. More specifically, these operations fall under the scope of dynamic pickup and delivery problems (dPDP), where couriers are dynamically assigned to retrieve orders from different vendors and deliver them to customer locations (Pillac et al. 2013, Psaraftis et al. 2016). The literature on dPDP is extensive (Berbeglia et al. 2010), spanning various applications in last-mile logistics, including meal delivery (Auad et al. 2023, 2024a, Ulmer et al. 2021, Yildiz and Savelsbergh 2019).

The Restaurant Meal Delivery Problem (RMDP), formally introduced by Reyes et al. (2018) and further developed by Ulmer et al. (2021), presents several distinctive features that set it apart from traditional delivery problems. Most notably, these operations face exceptionally narrow time windows and restrictions on combining orders from different restaurants, making route optimization especially complex. The stochasticity in both delivery and meal preparation processes creates unique operational challenges, while the urgency of dynamic requests and perishable nature of the product impose strict time constraints that significantly impact service quality (Mao et al. 2019). Recent work by Hildebrandt and Ulmer (2022) has specifically addressed the challenge of accurate delivery time predictions, which is crucial for customer satisfaction and system efficiency.

A critical aspect of modern meal delivery systems is courier autonomy and workforce management. Unlike traditional vehicle routing problems where drivers follow prescribed routes, meal delivery couriers often operate as independent contractors with the freedom to accept or reject orders (Auad et al. 2023). This has led to significant research in workforce scheduling (Ulmer and Savelsbergh 2020, Dai and Liu 2020) and service area zoning (Ulmer et al. 2022, Auad et al. 2023) to ensure efficient operations while managing labor costs. Recent work by Ausseil et al. (2022) and Ausseil et al. (2024) addresses the fundamental challenge of predicting courier behavior and its integration into order dispatching decisions. Their study demonstrates that incorporating individual courier acceptance patterns into the assignment process can significantly improve platform revenue and service quality, while highlighting important trade-offs between system efficiency and fairness in courier treatment. This research has revealed that while personalized courier behavior modeling can enhance system performance, it may lead to uneven distribution of less desirable orders to more agreeable couriers.

The evolution of solution approaches in meal delivery problems reflects the increasing complexity of operational requirements. Early work focused on developing intuitive heuristic methods for dynamic order assignment (Reyes et al. 2018, Liu 2019), emphasizing the balance between routing efficiency and fleet flexibility. Yildiz and Savelsbergh (2019) provided important insights through analysis of optimal dispatching strategies in deterministic settings, particularly regarding the value of order bundling and guidelines for demand management. More recent approaches have embraced simulation-based methods, with Steever et al. (2019) introducing the Virtual Food Court Delivery Problem and employing mixed-integer linear programming along with an auction-based heuristic to anticipate future system states and account for the dynamic nature of the problem. A notable recent development addresses the emerging concept of ghost kitchens, where multiple virtual restaurants operate from a centralized facility. Neria et al. (2024) propose an innovative approach combining large neighborhood search with value function approximation to jointly optimize food preparation scheduling and vehicle dispatching decisions. Their work demonstrates how centralization can improve both service quality and operational efficiency through better order consolidation and synchronized preparation times, while highlighting important trade-offs between delivery speed and food freshness.

The integration of machine learning (ML) methods in transportation logistics has gained significant momentum, offering new possibilities for solving complex routing and assignment problems (Hildebrandt et al. 2023, Yan et al. 2022). The work by Jahanshahi et al. (2022) introduced RL techniques to determine anticipatory order assignments, postponement, and rejections, focusing on learning expected future revenue from decisions. In a related context, Pham and Kiesmüller (2024) demonstrate the power of combining genetic algorithms with graph neural networks for complex routing problems, developing a hybrid value function approximation method that effectively handles high-dimensional state and decision spaces. Their innovative approach to state encoding through multiattribute graphs and spatial markers removes the need for manually crafted basis functions, making training more efficient. Chen et al. (2022) developed a deep Q-learning approach for managing hybrid fleets of traditional vehicles and drones in same-day delivery operations. Their approach incorporated both resource utilization and action impact features in the value function approximation. The resulting policy was shown to be robust to changes in customer request patterns while effectively balancing the complementary capabilities of vehicles and drones. Auad et al. (2024b) address the challenge of dynamic fleet sizing in rapid delivery operations through a deep Q-learning approach. Their framework enables real-time decisions about when and how to add delivery capacity in response to demand fluctuations, demonstrating significant improvements over traditional methods that rely solely on order-per-courier ratios.

Despite these advances, several important gaps remain in the literature. While existing approaches have addressed specific aspects such as order assignment (Reyes et al. 2018), workforce scheduling (Ulmer and Savelsbergh 2020), or couriers' acceptance probability (Ausseil et al. 2024), few have successfully integrated these elements into a comprehensive framework. Most existing solutions either focus on immediate operational efficiency or long-term performance, but rarely both.

Additionally, while recent work has begun to explore RL approaches (Jahanshahi et al. 2022), the integration of these techniques with traditional routing and scheduling decisions remains limited.

Our work addresses these gaps by proposing a novel RL framework that combines n-step SARSA (Sutton and Barto 2018) with hyper-heuristic optimization (Drake et al. 2020). This approach explicitly considers both immediate operational costs and long-term system performance, while maintaining computational tractability through careful design of the value function approximation. Moreover, our framework explicitly models courier behavior and autonomy, addressing a critical aspect of modern meal delivery operations that has been understudied in the literature.

3. Problem definition

Let $\mathcal{T} \subseteq \mathbb{R}_+$ be the operating period for a meal delivery platform (*e.g.*, a day, sequence of days). A set of restaurants $\mathcal{R}$ offer their food products on the platform. Each restaurant $r \in \mathcal{R}$ has an associated pickup location $\ell_r \in \mathcal{L}_R \subset \mathbb{R}^2$, where $\mathcal{L}_R$ is the set of all restaurant pickup locations.

During $\mathcal{T}$, customers continuously place orders to the restaurants in $\mathcal{R}$. For an order o, we denote the time when it is placed by $t_o \in \mathcal{T}$ and the restaurant where it is placed by $r_o \in \mathcal{R}$. At t_o, the platform promises the customer to deliver its order at a delivery location ℓ_o by deadline d_o, otherwise it is considered late. Each order o undergoes a preparation at its restaurant, and becomes ready for delivery at ready time b_o. All information regarding order o becomes known to the platform at t_o (including t_o).

To complete deliveries, the platform uses a fleet of couriers $\mathcal{C}$. Each courier $c \in \mathcal{C}$ works n_c uninterrupted on-duty periods $\{[\sigma_{c,i}^{start}, \sigma_{c,i}^{end}]\}_{i=1}^{n_c}$ during $\mathcal{T}$, with $[\sigma_{c,i}^{start}, \sigma_{c,i}^{end}] \subseteq \mathcal{T}$, and $\sigma_{c,i}^{start} < \sigma_{c,i}^{end}$, for all $i \in [n_c]$; during each of these periods, the platform may offer orders pending delivery to courier c, who then autonomously decides whether to accept or reject the offer. The platform only learns of the courier's decision after making the offer. If the courier accepts, then we say the order *is assigned* to the courier, namely the pickup and delivery tasks associated with that order are inserted in the courier's ongoing itinerary. At any time, each courier has a maximum capacity of $N_{\max}^{o}$ active orders, meaning their itinerary can contain tasks associated with up to $N_{\max}^{o}$ distinct orders; this constraint must be satisfied when assigning new orders. Couriers' locations are known to the planner at all times. However, the planner does not know in advance when a courier will enter or exit the system, *i.e.*, $\sigma_{c,i}^{start}$ and $\sigma_{c,i}^{end}$ are unknown to the planner at any point before their occurrence. We assume that all assignments are final and cannot be undone; Furthermore, couriers with incomplete assigned tasks do not exit the system, although they will reject order offers that, if accepted, would extend the duration of their routes beyond the time the intend to end their block, $\sigma_{c,i}^{end}$.

The distance between pairs of locations are modeled by a function $D : \mathbb{R}^2_+ \times \mathbb{R}^2_+ \to \mathbb{R}_+$, where $D(\ell, \ell')$ represents the driven distance incurred by a courier when traveling from location $\ell \in \mathbb{R}^2_+$ to location $\ell' \in \mathbb{R}^2_+$. Travel times between pairs of locations are modeled by random variables $T^{\text{travel}}(h; \ell, \ell')$ representing the random travel time incurred by a courier when traveling from location $\ell \in \mathbb{R}^2_+$ to location $\ell' \in \mathbb{R}^2_+$ during the hour of the day $h \in [0, 24)$. A courier incurs a random service time of μ_p when picking up an order, and a random service time of μ_d when dropping it off. Consequently, suppose that at time $t \in \mathcal{T}$, a courier $c \in \mathcal{C}$ at location ℓ_c that previously accepted to deliver an order o starts heading towards ℓ_{r_o} to pick it up. Then courier c finishes the pick-up of o at time $t^p_{c,o} \doteq \max\{b_o, t + T^{\text{travel}}(h; \ell_c, \ell_{r_o}) + \mu_p\}$. Likewise, if at time t' courier c starts heading to the delivery location ℓ_o from a location ℓ_c, then o is effectively delivered at time $t^d_{c,o} := t' + T^{\text{travel}}(h; \ell_c, \ell_o) + \mu_d$; at this time, the courier departs from that location to resume its itinerary. If no pending tasks remain on the itinerary, then courier c repositions itself by heading to the nearest restaurant location and remains there until it accepts a new order or else until the end of its ongoing block.

This work addresses the challenge faced by delivery platforms in assigning orders efficiently to their pool of couriers. When customers place delivery orders, the platform must decide which couriers to offer these orders to. If an offered order is rejected by a courier, the platform must find a different courier to offer the order to, potentially leading to longer delivery times. For accepted orders, the platform must integrate the new delivery into the courier's ongoing schedule. The objective is to optimize crucial operational metrics, such as minimizing delivery delays and minimizing the total distance traveled by couriers.

Now we formulate the problem as a sequential decision process, outlining its main components, namely, decision epochs, states, actions, transitions, rewards, exogenous information, and objective.

Decision Epochs. Each decision epoch corresponds to a time point where the planner evaluates whether to insert the tasks associated with each unassigned orders into the itinerary of available couriers. We assume the planner makes a decision every Δ time units (*e.g.*, seconds). Accordingly, we denote the set of decision epochs as $\mathcal{T}_\Delta = \mathbb{N} \subseteq \mathcal{T}$, where each epoch is indexed by a natural number representing the number of Δ-time unit intervals that have elapsed since the start of the timeline.

States. At each decision epoch $t \in \mathcal{T}_\Delta$, the state $s_t \in \mathcal{S}$ captures all relevant features necessary for the planner's decision-making. It is defined as

$$s_t := (\bar{h}_t, w_t, \mathcal{O}_t, \mathcal{C}_t, \tau^{duty}_t, \mathcal{O}^{\text{open}}_t, \phi_t, c_t, \tau^P_t, \tau^D_t, \rho_t, \ell_t)$$

The scalar $\bar{h}_t$ represents the time of day at epoch t, and w_t is a binary indicator that equals 1 if and only if t corresponds to a weekday. The set of all active (undelivered) customer orders at time

t is $\mathcal{O}_t$, and $\mathcal{C}_t$ is the set of couriers on duty at that time. The time elapsed since the start of each courier's most recent block is encoded by $\tau_t^{duty} := \{\tau_{t,c}^{duty}\}_{c\in\mathcal{C}_t}$. The times elapsed since arriving at a pickup and delivery location, respectively, are denoted as $\tau_t^P := \{\tau_{t,c}^P\}_{c\in\mathcal{C}_t}$ and $\tau_t^D := \{\tau_{t,c}^D\}_{c\in\mathcal{C}_t}$, and are set to 0 when the courier is not currently engaged in the corresponding activity. The set $\phi_t := \{\phi_{t,c}\}_{c\in\mathcal{C}_t}$ tracks, for each courier, the active orders that they were previously offered but rejected. The subset $\mathcal{O}_t^{\text{open}} \subseteq \mathcal{O}_t$ includes all active orders that have not yet been assigned to a courier. The mapping $c_t := \{c_{t,o}\}_{o\in\mathcal{O}_t}$ associates each active order $o \in \mathcal{O}_t$ with its assigned on-duty courier, with $c_{t,o} = 0$ indicating that the order remains unassigned (*i.e.*, $o \in \mathcal{O}_t^{\text{open}}$). The mapping $\rho_t := \{\rho_{t,c}\}_{c\in\mathcal{C}_t}$ describes the planned itineraries of on-duty couriers. For each $c \in \mathcal{C}_t$, the sequence $\rho_{t,c}$ consists of tuples (t', ℓ', o, q) specifying that c is scheduled to complete either the pickup $(q = 1)$ or delivery $(q = 2)$ task for order o at location ℓ' at planned time $t' > t$. The location ℓ' corresponds to ℓ_{r_o} for pickups and ℓ_o for deliveries. These tuples are ordered in increasing order of t'. Finally, the location information of on-duty couriers is given by $\ell_t := \{\ell_{t,c}\}_{c\in\mathcal{C}_t}$. If a courier $c \in \mathcal{C}_t$ is currently waiting without an active pickup or delivery task, $\ell_{t,c}$ encodes its present location. Otherwise, if the courier is en route to fulfill a pickup or delivery, $\ell_{t,c}$ specifies the destination toward which it is currently traveling.

Throughout the paper, empty mappings (*i.e.*, a mapping with no elements in its domain) are denoted as $\emptyset$.

Decisions. At each decision epoch $t \in \mathcal{T}_\Delta$, the planner makes a decision $x_t \in \mathcal{X}_t(s_t)$, specifying which unassigned orders to be offered to which on-duty couriers for pickup and delivery, and a reordering of tasks in each courier's schedule at epoch t. For couriers receiving a new order offer, the reordering of tasks includes determining the specific positions where the pickup and delivery tasks of newly offered orders would be inserted if accepted. Specifically, the planner chooses

- A subset of unassigned orders $\mathcal{O}_t^{\text{offered}} \subseteq \mathcal{O}_t^{\text{open}}$ and a corresponding subset of couriers $\mathcal{C}_t^{\text{offered}} \subseteq \mathcal{C}_t$ to receive these offers.
- A newly proposed schedule $\rho_{t,c}^{\text{prop}}$ for each on-duty courier $c \in \mathcal{C}_t$.

Each courier $c \in \mathcal{C}_t^{\text{offered}}$ is offered exactly one order $o \in \mathcal{O}_t^{\text{offered}}$, ensuring a one-to-one mapping where $|\mathcal{C}_t^{\text{offered}}| = |\mathcal{O}_t^{\text{offered}}|$. Although the framework could accommodate multiple simultaneous order offers per courier, we restrict offers to one order per courier per decision epoch. This restriction simplifies the decision-making process, reducing the cognitive burden on couriers who must evaluate route modifications in real time. In addition, to maintain operational stability, we enforce that any tasks in progress (*i.e.*, the first task in a couriers' current sequence, which they are currently completing) remain fixed in their current sequence.

For each on-duty courier $c \in \mathcal{C}_t$, the construction of the proposed schedule $\rho_{t,c}^{\text{prop}}$ involves a reordering of the tasks in the courier's current schedule $\rho_{t,c}$. If in addition $c \in \mathcal{C}_t^{\text{offered}}$, the construction of $\rho_{t,c}^{\text{prop}}$ also involves inserting the pickup and delivery tasks of an order $o \in \mathcal{O}_t^{\text{offered}}$, represented as $(t', \ell_{r_o}, o, 1)$ and $(t'', \ell_o, o, 2)$, ensuring that the pickup of each order precedes its delivery. The mapping with all the proposed courier schedules given by the decision x_t is denoted as $\rho_t^{\text{prop}} := \{\rho_{t,c}^{\text{prop}}\}_{c \in \mathcal{C}_t}$.

Since updating an existing schedule requires specifying the positions of where the new pickup and delivery tasks are inserted, we encode the decision as $x_t = (x_t^{c,o})_{\{c \in \mathcal{C}_t,\, o \in \mathcal{O}_t\}}$, where

$$x_t^{c,o} = \begin{cases} (k_1, k_2), & \text{if courier } c \text{ is offered order } o \in \mathcal{O}_t^{\text{open}} \text{ with the pickup inserted at position} \\ & \quad k_1 \in K_1(c) \text{ and the delivery at position } k_2 \in K_2(k_1, c) \\ (k_1, k_2), & \text{if order } o \in \mathcal{O}_t \setminus \mathcal{O}_t^{\text{open}} \text{ has its position in the route of courier } c = c_{t,o} \text{ changed,} \\ & \quad \text{having the pickup at position } k_1 \in K_1(c) \text{ and the delivery at position} \\ & \quad k_2 \in K_2(k_1, c) \\ (0,0), & \text{otherwise} \end{cases}$$

Here, the tuple (k_1, k_2) specifies the positions of the pickup and delivery tasks within courier c's existing schedule $\rho_{t,c}$. The case $x_t^{c,o} = (0,0)$ indicates that either (i) no insertion (and thus no offer) of order $o \in \mathcal{O}_t^{\text{open}}$ into the route of couriers c occurs, or (ii) no change in the position an already assigned order $o \in \mathcal{O}_t \setminus \mathcal{O}_t^{\text{open}}$ within the route of its assigned courier $c = c_{t,o}$. The set of allowable pickup positions is defined as

$$K_1(c) := \begin{cases} \{2, \dots, |\rho_{t,c}| + 1\}, & \text{if } |\rho_{t,c}| \geq 1, \text{ and } c \in \mathcal{C}_t^{\text{offered}}, \\ \{2, \dots, |\rho_{t,c}| - 1\}, & \text{if } |\rho_{t,c}| \geq 3, \exists o \in \mathcal{O}_t \setminus \mathcal{O}_t^{\text{open}} \colon x_t^{c,o} \neq (0,0), \text{ and } c \in \mathcal{C}_t \setminus \mathcal{C}_t^{\text{offered}}, \\ \{1\}, & \text{if } |\rho_{t,c}| = 0, \text{ and } c \in \mathcal{C}_t^{\text{offered}}, \\ \{0\}, & \text{otherwise.} \end{cases}$$

For each possible value of k_1, the delivery position must be later in the sequence, *i.e.*, $K_2(k_1, c) := \{k_1 + 1, \dots, |\rho_{t,c}| + 2\}$ if $c \in \mathcal{C}_t^{\text{offered}}$, $K_2(k_1, c) := \{k_1 + 1, \dots, |\rho_{t,c}|\}$ if $c \in \mathcal{C}_t \setminus \mathcal{C}_t^{\text{offered}}$, and $K_2(k_1, c) := \{0\}$, otherwise. We give more details on the construction of ρ_t^{prop} in the Appendix A.

Let $\mathcal{P}_t^{\text{offered}} \subseteq \mathcal{O}_t^{\text{offered}} \times \mathcal{C}_t^{\text{offered}}$ denote the set of order-courier pairs (o, c) representing an assignment offer made by the planner. Upon receiving an offer, courier c decides within the time window $(t, t+1]$ whether to accept the updated schedule that includes order o. To mitigate the likelihood of repeated rejections across decision epochs, the planner enforces a rejection policy: if a courier $c \in \mathcal{C}_t$ previously rejected an order $o \in \mathcal{O}_t^{\text{open}}$, that order is not re-offered to the same courier in subsequent epochs. Orders in $\mathcal{O}_t^{\text{open}} \setminus \mathcal{O}_t^{\text{offered}}$ remain unassigned and will be reconsidered in future decision epochs. Furthermore, if an order in $\mathcal{O}_t^{\text{open}}$ remains unassigned for more than a time threshold λ_{lost}, then the order is assumed lost and can no longer be offered or served.

Exogenous Information. At each epoch $t+1$, new information is revealed to the planner, represented by the vector of exogenous information $\omega_{t+1} \in \Omega$, where

$$\omega_{t+1} := (\tilde{\mathcal{O}}_{t+1}^{\text{new}}, \tilde{\mathcal{O}}_{t+1}^{\text{delivered}}, \tilde{\mathcal{O}}_{t+1}^{\text{lost}}, \tilde{\mathcal{P}}_{t+1}^{\text{accepted}}, \tilde{\mathcal{P}}_{t+1}^{\text{rejected}}, \tilde{\mathcal{C}}_{t+1}^{\text{new}}, \tilde{\mathcal{C}}_{t+1}^{\text{out}}, \tilde{\ell}_{t+1})$$

This information comprises several elements that capture system changes occurring within the interval $(t, t+1]$.

New customer orders arrive dynamically, represented by the set $\tilde{\mathcal{O}}^{\text{new}}_{t+1}$, which contains all orders placed within $(t, t+1]$ along with their associated attributes, namely, placement time, ready time, and pickup and delivery locations. The set $\tilde{\mathcal{O}}^{\text{delivered}}_{t+1}$ contains orders that were active at epoch t and successfully delivered during $(t, t+1]$, and the set $\tilde{\mathcal{O}}^{\text{lost}}_{t+1}$ the orders that become lost during $(t, t+1]$.

Courier decisions regarding assignment offers are captured through the sets $\tilde{\mathcal{P}}^{\text{accepted}}_{t+1} \subseteq \mathcal{P}^{\text{offered}}_{t}$ and $\tilde{\mathcal{P}}^{\text{rejected}}_{t+1} \subseteq \mathcal{P}^{\text{offered}}_{t}$, which respectively denote accepted and rejected courier-order pairings proposed at epoch t.

The set $\tilde{\mathcal{C}}^{\text{new}}_{t+1}$ consists of couriers who started their block during $(t, t+1]$, while their start times are recorded as $\tilde{\tau}^{\text{new}}_{t+1} := \{\tilde{\tau}^{\text{new}}_{t+1,c}\}_{c \in \tilde{\mathcal{C}}^{\text{new}}_{t+1}}$, where $\tilde{\tau}^{\text{new}}_{t+1,c}$ represents the time when courier c became available. Similarly, $\tilde{\mathcal{C}}^{\text{out}}_{t+1}$ denotes the set of couriers who ended their duty period and exited the system during the same interval.

Finally, $\tilde{\ell}_{t+1,c}$ provides updated location information for on-duty couriers at epoch $t+1$. For each courier $c \in \mathcal{C}_{t+1}$, $\tilde{\ell}_{t+1,c}$ encodes the courier's current location if they are waiting without any pickup or delivery task, and the location the courier is traveling toward if they are currently en route to fulfill a pickup or delivery.

Rewards. The reward r_t associated with epoch t materializes at time $t+1$, after ω_{t+1} is observed. It is calculated as $r_t := r^{\text{lateness}}_t + r^{\text{dist}}_t + r^{\text{lost}}_t$, where r^{lateness}_t denotes the cost (*i.e.*, negative reward) due to late deliveries of orders during $(t, t+1]$, r^{dist}_t the cost of the total distance to be driven by the courier fleet for tasks initiated during $(t, t+1]$, and r^{lost}_t is the penalty incurred for orders permanently lost during $(t, t+1]$ after remaining unassigned for more than a time λ_{lost} after being placed.

Let K^{late} represent the cost incurred for each unit of time that an order is delivered past its promised delivery time, K^{dist} the cost per unit of distance traveled by couriers, and K^{lost} the penalty paid for each lost order. The costs based on delivery lateness, driven distance, and lost orders associated with epoch $t \in \mathcal{T}_\Delta$ are computed as

$$\begin{aligned}
r^{lateness}_t &:= -K^{\text{late}} \sum_{o \in \tilde{\mathcal{O}}^{\text{delivered}}_{t+1}} \max\{0, t^d_{c_{t,o},o} - d_o\} \\
r^{\text{dist}}_t &:= -K^{\text{dist}} \sum_{c \in \mathcal{C}_t : |\rho_{t,c}| \geq 1} D(\ell_t, \tilde{\ell}_{t+1}) \\
r^{\text{lost}}_t &:= -K^{\text{lost}} |\tilde{\mathcal{O}}^{\text{lost}}_{t+1}|
\end{aligned}$$

Transition. We define the transition as a mapping $S : \mathcal{S} \times \mathcal{X} \times \Omega \to \mathcal{S}$ that transforms a state-action-exogenous information tuple (s_t, x_t, ω_{t+1}) into the state at the next decision epoch, $s_{t+1} := S(s_t, x_t, \omega_{t+1})$.

Time-related attributes h_{t+1} and w_{t+1} are directly determined by the value of $t+1$. The set of active orders is updated to $\mathcal{O}_{t+1} = \mathcal{O}_t \cup \tilde{\mathcal{O}}^{\text{new}}_{t+1} \setminus \tilde{\mathcal{O}}^{\text{delivered}}_{t+1}$, incorporating new orders placed during $(t, t+1]$ and removing those that were delivered. Similarly, the set of couriers on duty is updated to $\mathcal{C}_{t+1} = \mathcal{C}_t \cup \tilde{\mathcal{C}}^{\text{new}}_{t+1} \setminus \tilde{\mathcal{C}}^{\text{out}}_{t+1}$, reflecting couriers that started or ended their duty periods during $(t, t+1]$.

The time each courier has spent on duty is updated as follows: for each courier $c \in \mathcal{C}_t \setminus \tilde{\mathcal{C}}^{\text{out}}_{t+1}$, the duty duration increases as $\tau^{\text{duty}}_{t+1,c} = \tau_{t,c} + \Delta$. For newly on-duty couriers $c \in \tilde{\mathcal{C}}^{\text{new}}_{t+1} \setminus \tilde{\mathcal{C}}^{\text{out}}_{t+1}$, the duty time is set as $\tau^{\text{duty}}_{t+1,c} = t + \Delta - \tilde{\tau}^{\text{new}}_{t+1,c}$. Any courier in $\tilde{\mathcal{C}}^{\text{out}}_{t+1}$ is removed from the domain of this mapping.

Let $\mathcal{O}^{\text{accepted}}_{t+1} \subseteq \mathcal{O}^{\text{offered}}_t$ denote the subset of unassigned orders that were offered to couriers at epoch t and subsequently accepted during $(t, t+1]$. The set of unassigned orders is updated as $\mathcal{O}^{\text{open}}_{t+1} = \mathcal{O}^{\text{open}}_t \cup \tilde{\mathcal{O}}^{\text{new}}_{t+1} \setminus \mathcal{O}^{\text{accepted}}_{t+1}$, where the update accounts for (i) unassigned orders that remain after decision x_t (either because they were not offered or were offered and subsequently rejected during $(t, t+1]$), (ii) newly placed orders, and (iii) the removal of orders that were accepted by couriers. The mapping c_{t+1}, which associates orders with assigned couriers, is adjusted by removing assignments for delivered orders and initializing $c_{t+1,o} = 0$ for newly placed order $o \in \tilde{\mathcal{O}}^{\text{new}}_{t+1}$.

The updated mapping with new couriers schedules, $\rho_{t+1} = \{\rho_{t+1,c}\}_{c \in \mathcal{C}_{t+1}}$, is computed as follows. First, for each $c \in \mathcal{C}_{t+1}$, we define an auxiliary schedule,

$$\rho^{\text{aux}}_{t+1,c} = \begin{cases} \rho^{\text{prop}}_{t,c} & \text{if } c \notin \mathcal{C}^{\text{offered}}_t \\ \rho^{\text{prop}}_{t,c} & \text{if } c \in \mathcal{C}^{\text{offered}}_t \text{ and } c \text{ accepted the newly proposed schedule } \rho^{\text{prop}}_{t,c} \\ \rho_{t,c} & \text{if } c \in \mathcal{C}^{\text{offered}}_t \text{ and } c \text{ rejected the newly proposed schedule } \rho^{\text{prop}}_{t,c} \end{cases}$$

Note that only couriers that are offered a new order have the option of rejecting the newly proposed schedule.

Then, for each $c \in \mathcal{C}_{t+1}$, any task $(t', \ell, o, q) \in \rho^{\text{aux}}_{t+1,c}$ with $t' \leq t+1$ is removed if completed by $t+1$; otherwise, its planned completion time t' is updated with a new estimate to account for the delay. The final updated schedule $\rho_{t+1,c}$ is obtained by applying this adjustment to $\rho^{\text{aux}}_{t+1,c}$.

Finally, the courier locations are updated as $\ell_{t+1} = \{\tilde{\ell}_{t,c}\}_{c \in \mathcal{C}_{t+1}}$.

Objective. Let Π be the space of policies π that map each possible state s_t to a feasible action $x_t = \pi(s_t)$, and let $\gamma \in (0,1)$ be a discount factor on future rewards. Given an initial state $s_0 := (0, 0, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$, the goal is to find an optimal policy π^* that maximizes the expected rewards (given by the expected total delay across the delivery of all placed orders, the total distance driven by couriers, and lost orders). Mathematically:

$$\pi^* \in \arg\max_{\pi \in \Pi} \mathbb{E}\left[\sum_{t \in \mathcal{T}_\Delta} \gamma^t R(S_t, \pi(S_t)) \middle| S_0 = s_0 \right]$$

where S_t denotes the random state at decision epoch t, and $R(S, x)$ the reward given state S and action x.

Remark 1 *Some reward components, such as lateness penalties ($r^{lateness}$) or lost orders (r^{loss}), may stem from courier-order assignments made in earlier decision periods. However, the state representation encodes all relevant historical information (*e.g.*, courier positions, pending tasks, order waiting times, and delivery deadlines) needed to capture these past effects. Consequently, the distribution of next states and rewards depends only on the current state and action, ensuring that the sequential decision process maintains the Markov property.*

4. Methodology

In this section, we present an RL-based methodology for order assignment and routing in meal delivery. The proposed framework integrates n-step SARSA with value function approximation and a bandit-based hyper-heuristic to guide decision-making, balancing immediate costs (associated with delivery lateness and driven distance) against long-term costs captured by system features that impact future efficiency. The main procedure jointly trains these components: the n-step SARSA algorithm updates the value function parameters using actions selected by the hyper-heuristic, while the hyper-heuristic adaptively re-estimates its weights to explore the action space efficiently.

We conclude this section by introducing a set of benchmark policies that trade off computational simplicity with model complexity. These policies serve as comparative baselines to evaluate the effectiveness of our proposed approach.

4.1. Value function approximation

We approximate the value function using a linear approximation based on state-action features. The estimated value function is defined as

$$\hat{Q}(s_t, x_t, \theta) = c_0(s_t, x_t) + \sum_{f=0}^{n_{\text{feat}}} \theta_f \phi_f(s_t, x_t) \tag{1}$$

where given a selected action x_t in state s_t, $\hat{Q}(s_t, x_t, \theta)$ represents the estimated value function, and $c_0(s_t, x_t)$ denotes the immediate costs.

The immediate cost term combines delivery lateness and travel distance:

$$c_0(s_t, x_t) \coloneqq \sum_{c \in \mathcal{C}_t^{\text{offered}}} \left(K^{\text{dist}} \left(\delta(\rho_{t,c}^{\text{prop}}) - \delta(\rho_{t,c}) \right) + K^{\text{late}} \left(\zeta(\rho_{t,c}^{\text{prop}}) - \zeta(\rho_{t,c}) \right) \right), \tag{2}$$

where $\zeta(\rho_{t,c})$ quantifies the expected lateness in courier c's route, computed from the estimated completion times of delivery tasks, and $\delta(\rho_{t,c})$ measures the corresponding travel distance, derived from the ordered sequence of locations in the route:

$$\zeta(\rho_{t,c}) \coloneqq \sum_{o:(t',\ell',o,q') \in \rho_{t,c}} \max\{t^o - d^o, 0\},$$

$$\delta(\rho_{t,c}) \coloneqq D(\tilde{\ell}_{t,c}, \ell^1) + \sum_{k=2}^{|\rho_{t,c}|} D(\ell^{k-1}, \ell^k)$$

The second term in (1) considers a linear combination of n_{feat} features, $\phi_f(s_t, x_t)$, with weight parameters θ_f. These features capture longer-term system effects not explicitly modeled in the immediate cost (*e.g.*, workload balance, courier availability), enabling generalization across similar state-action pairs. The objective is to learn the parameter vector θ that minimizes the prediction error of $\hat{Q}$ over future scenarios.

In this linear formulation, the gradient $\nabla_\theta \hat{Q}(s_t, x_t, \theta)$ required for parameter updates (Line 12 in Algorithm 1) simplifies to the feature vector $\phi(s_t, x_t)$. This formulation achieves a balance between expressiveness and computational efficiency: unlike a tabular model, it supports generalization across states while remaining tractable in large-scale, real-time environments. In general, $c_0(s_t, x_t)$ captures the immediate operational cost, while the feature-weighted term learns the cost variations that emerge over longer decision horizons.

In particular, our implementation considers a set of features that capture system congestion, defined as follows.

- **Order-to-courier ratio.** Measures the global workload level in the system:

$$\phi_1(s_t, x_t) = \frac{|\mathcal{O}_t|}{\max\{|\mathcal{C}_t|, 1\}}$$

- **Ratio of unoffered orders to couriers.** Given tentative decision x_t, this feature measures how many unassigned orders remain without being offered relative to the number of active couriers:

$$\phi_2(s_t, x_t) = \frac{|\{o \in \mathcal{O}_t^{\text{open}} : \forall c \in \mathcal{C}_t, x_t^{c,o} = (0,0)\}|}{\max\{|\mathcal{C}_t|, 1\}}$$

- **Number of unoffered orders.** This feature counts the total number of unassigned orders that are not being offered to any courier per decision x_t:

$$\phi_3(s_t, x_t) = |\{o \in \mathcal{O}_t^{\text{open}} : \forall c \in \mathcal{C}_t, x_t^{c,o} = (0,0)\}|$$

- **Deadline distribution of unoffered orders.** To capture the urgency of pending orders, we partition the upcoming time horizon into $m_{\text{feat}} \geq 1$ intervals of $\Delta_{\text{feat}} > 0$ seconds each and count the number of orders not being offered whose delivery deadlines fall within each interval. For $i = 0, \ldots, m_{\text{feat}} - 1$:

$$\phi_{4+i}(s_t, x_t) = \left| \left\{ o \in \mathcal{O}_t^{\text{open}} : \forall c \in \mathcal{C}_t,\, x_t^{c,o} = (0,0),\, t + \frac{i \cdot \Delta_{\text{feat}}}{\Delta} \leq d_o < t + \frac{(i+1) \cdot \Delta_{\text{feat}}}{\Delta} \right\} \right|$$

and for orders whose deadlines fall beyond the last interval:

$$\phi_{4+m_{\text{feat}}}(s_t, x_t) = \left| \left\{ o \in \mathcal{O}_t^{\text{open}} : \forall c \in \mathcal{C}_t,\, x_t^{c,o} = (0,0),\, d_o \geq t + \frac{m_{\text{feat}} \cdot \Delta_{\text{feat}}}{\Delta} \right\} \right|$$

- **Couriers without assigned offers.** This feature counts the number of active couriers who do not receive an order offer under decision x_t:

$$\phi_{5+m_{\text{feat}}}(s_t, x_t) = |\{c \in \mathcal{C}_t \colon \forall o \in \mathcal{O}_t, x_t^{c,o} = (0,0)\}|$$

- **Average restaurant-to-customer distance.** This feature computes the average distance between restaurant and delivery locations for all orders being offered under decision x_t:

$$\phi_{6+m_{\text{feat}}}(s_t, x_t) = \frac{\sum_{o \in \mathcal{O}_t^{\text{open}} \colon \exists c \in \mathcal{C}_t, x_t^{c,o} \neq (0,0)} D(\ell_{r_o}, \ell_o)}{\max\{|\{o \in \mathcal{O}_t^{\text{open}} \colon \exists c \in \mathcal{C}_t, x_t^{c,o} \neq (0,0)\}|, 1\}}$$

Overall, we consider $n_{\text{feat}} = m_{\text{feat}} + 6$ features, plus a bias term $\phi_0(s_t, x_t) = 1$.

4.2. n-step SARSA with value function approximation

Algorithm 1 presents our online implementation of n-step SARSA with value function approximation. The algorithm is designed for a dynamic setting in which decisions and updates occur continuously. At each decision epoch, the decision maker selects the action that minimizes the estimated cost $\hat{Q}(s_t, x_t, \theta^{\text{best}})$, where θ^{best} represents the best set of parameters learned so far.

To facilitate learning, we maintain a memory buffer M of size m storing recent experience tuples. As shown in Algorithm 1, the stored tuple is $(s_{t-n}, x_{t-n}, G - c_0(s_t, x_t))$, where G is the full value function. The value stored $G - c_0(s_t, x_t)$ is then the net return of the immediate cost. This is done to directly estimate the feature-based component of the value function during the parameter update step. At each training iteration t, we extract a batch of size $\min\{b, t\}$ tuples $(b < m)$ from the memory to train the model. Unlike standard one-step SARSA, where parameters update immediately after each decision, our approach revisits stored experiences to enhance stability of learning. The use of n-step updates reduces variance and incorporates longer-term cost information. Additionally, experience replay mitigates instability by allowing the decision maker to learn from past decisions, reducing sensitivity to transient fluctuations in the environment.

At regular intervals, the algorithm evaluates the quality of the learned value function to track progress and preserve the most effective parameters. Let $\hat{V}(s_0, \theta)$ represent the estimated performance of the current parameter vector θ from the initial state s_0, while $\hat{V}(s_0, \theta^{\text{best}})$ corresponds to the best value observed so far. Whenever the current estimate θ yields a lower cost, the best parameter vector θ^{best} is updated. This mechanism ensures that training retains the parameter configuration associated with the highest-performing value function.

4.3. Multi-armed bandit-based hyper-heuristic

In this section, we introduce the multi-armed bandit-based hyper-heuristic that is integrated with the n-step SARSA procedure. At each decision epoch t, given a state s_t, the n-step SARSA procedure invokes the hyper-heuristic to generate an action x_t, which is then used to update

Algorithm 1: n-step SARSA

```
1  Input: Initial parameter vector $\theta_0$, learning rate $\alpha$, discount factor $\gamma$, initial state $s_0$, batch
     size $b$, memory size $m$, number of iterations $t_{\max}$, update frequency parameter $n_{\text{update}}$ ;
2  Initialize: Memory $M$ of size $m$, $\theta^{\text{best}} \leftarrow \theta_0$, $\theta \leftarrow \theta_0$;
3  for $t = 0, 1, \ldots, t_{\max}$ do
4      Select the action (set of offers) $x_t$ according to the current decision policy based on
         current state $s_t$ and parameter vector $\theta^{\text{best}}$ ;
5      Observe exogenous information and reward $r_t$;
6      Determine $s_{t+1}$;
7      if $t \geq n$ then
8          $G \leftarrow \sum_{i=0}^{n-2} \gamma^i r_{t-n+1+i} + \gamma^{n-1}\hat{Q}(s_t, x_t, \theta)$ ;
9          Store experience tuple $M_{t \pmod m} = (s_{t-n}, x_{t-n}, G - c_0(s_{t-n}, x_{t-n}))$;
10         $M^b \leftarrow$ Random sample of size $\min\{b, t\}$ of memory $M$;
11         for $(s', x', G') \in M^b$ do
12             $\theta \leftarrow \theta + \alpha(G' - \hat{Q}(s', x', \theta) + c_0(s', x'))\nabla_\theta \hat{Q}(s', x', \theta)$;
13     if ($t$ mod $n_{update} = 0$ or $t = t_{\max}$) and $\hat{V}(s_0, \theta) < \hat{V}(s_0, \theta^{best})$ then
14         update $\theta^{\text{best}} \leftarrow \theta$;
15 return $\theta^{best}$
```

the value function parameters θ in Equation (1) (Line 12 of Algorithm 1). In this formulation, the hyper-heuristic is not merely a post-training search mechanism, but actively influences the learning process by guiding exploration during training and continuously adapting its selection strategy over time.

To achieve this, we employ a Markov model-based hyper-heuristic, initially introduced in Kheiri (2020). This approach iteratively learns an optimal sequence of simple heuristics (*i.e.*, low-level heuristics) by applying a randomized process to an initial solution.

Hyper-heuristics are a class of metaheuristics that select, generate, or combine low-level heuristics to explore the solution space. A Markov model-based hyper-heuristic leverages an underlying Markov chain, where states correspond to elements from a predefined set of low-level heuristics $\mathcal{H}$. The transition matrix defines how heuristics should be sequenced and when the sequence should terminate. A particular variation of this procedure incorporates ideas from the Exponential Weights for Exploration and Exploitation (EXP3) algorithm from the multi-armed bandit literature (Lagos and Pereira 2024), allowing the transition probabilities to be updated iteratively based on performance. Let $T = [T_{h,g}]_{h,g \in \mathcal{H}} \in [0,1]^{|\mathcal{H}| \times |\mathcal{H}|}$ be a transition matrix, where $T_{h,g}$ represents the probability of transitioning from heuristic h to heuristic g in the resulting heuristic sequence; and $S = [S_{h,i}]_{h \in \mathcal{H}, i \in \{1,2\}} \in [0,1]^{|\mathcal{H}| \times 2}$ be a score matrix, where $S_{h,1}$ and $S_{h,2}$ determine whether the sequence should terminate with heuristic h and be applied to the current solution. These matrices satisfy $\sum_{g\in\mathcal{H}} T_{h,g} = S_{h,1} + S_{h,2} = 1$.

4.3.1. Algorithm overview Algorithm 2 runs until a predefined time limit t_{limit} (in seconds) is reached. Each iteration generates a new solution while updating two parameter matrices, $P = [P_{h,g}]_{(h,g)\in\mathcal{H}^2}$ and $L = [L_{h,u}]_{(h,u)\in\mathcal{H}\times\{1,2\}}$. These matrices are used to compute the transition probabilities that govern heuristic selection. The final values of P and L are preserved as initial parameters (P^0 and L^0) for subsequent executions of the algorithm, ensuring continuity across runs.

During execution, the algorithm monitors the elapsed time t_{elapsed} since initialization. At each iteration, it either (i) extends the current sequence by appending another heuristic ($u_{\text{next}} = 1$), or (ii) evaluates the current sequence by applying it to the incumbent solution $\hat{x}$ ($u_{\text{next}} = 2$). When a heuristic sequence is evaluated, up to three updates may occur:

- **Transition probability update.** If the new solution improves upon $\hat{x}$, the transition probabilities T are adjusted to increase the likelihood of repeating the corresponding sequence (Line 11). The update is performed using a learning rate parameter $\eta > 0$.
- **Solution acceptance.** Given a tolerance threshold $\varepsilon > 0$ and temperature parameter $\vartheta > 0$, the new solution x is accepted if its cost is lower than a factor of $\left(1+\varepsilon+\vartheta\left(1-\frac{t_{\text{elapsed}}}{t_{\text{limit}}}\right)\right)$ times the cost of the incumbent best solution x^*. If this condition holds (Line 12), x becomes the new starting solution for subsequent iterations.
- **Best solution update.** If x outperforms the current best solution x^*, then x^* is updated accordingly (Line 14).

4.3.2. Low-level heuristics The low-level heuristics in $\mathcal{H}$ are simple procedures that modify the current solution locally or partially destruct and reconstruct it through local search operations. In Algorithm 2, we assume that all low-level heuristics always produce feasible solutions. If any heuristic were capable of generating infeasible solutions, additional refinements would be required in the update steps (Lines 12 and 14). However, because all heuristics implemented in this study maintain feasibility by design, no further adjustments are required. We now describe in detail the specific heuristics considered in our framework, beginning with a set of auxiliary operators that support their construction.

(h_1^{aux}) Insert unassigned orders. This operator allocates a sequence of orders to couriers using a greedy insertion strategy. Given a current allocation x_t and two sequences, $\mathcal{O}^{\text{unassigned}}$, containing orders from $\mathcal{O}_t^{\text{open}}$, and $\mathcal{C}^{\text{unassigned}}$, containing couriers from $\mathcal{C}_t$, the procedure iterates through each order $o \in \mathcal{O}^{\text{unassigned}}$ and assigns it to the first eligible courier $c \in \mathcal{C}^{\text{unassigned}}$ that minimizes the incremental cost of inserting o into $\rho_{t,c}$, subject to predefined acceptance criteria. Formally, for each order $o \in \mathcal{O}^{\text{unassigned}}$, the operator performs the following steps:

Algorithm 2: Multi-armed bandit hyper-heuristic

```
1  Input: Current solution $\hat{x}$, time limit $t_{\text{limit}}$, learning rate $\eta$, threshold $\varepsilon$, temperature
     parameter $\vartheta$, set of low-level heuristics $\mathcal{H}$, initial transition matrices $P^0$ and $L^0$;
2  Initialize: Set $P_{h,g} \leftarrow P^0_{h,g}$, $L_{h,1} \leftarrow L^0_{h,1}$, $L_{h,2} \leftarrow L^0_{h,2}$ for all $h, g \in \mathcal{H}$. Set $x^* \leftarrow \hat{x}$. Sample
     $h_{\text{last}} \in \mathcal{H}$ uniformly at random. Initialize $\kappa$ as an empty sequence. Initialize $t_{\text{elapsed}} \leftarrow 0$;
3  while $t_{elapsed} < t_{limit}$ do
4      Set $T_{h,g} \leftarrow \frac{\exp(\eta P_{h,g})}{\sum_{g' \in \mathcal{H}} \exp(\eta P_{h,g'})}$, for $h, g \in \mathcal{H}$;
5      Set $S_{h,1} \leftarrow \frac{\exp(\eta L_{h,1})}{\exp(\eta L_{h,2}) + \exp(\eta L_{h,2})}$, and $S_{h,2} \leftarrow 1 - S_{h,1}$, for $h \in \mathcal{H}$;
6      Sample the next low-level heuristic $h_{\text{next}} = g \in \mathcal{H}$ with probability $T_{h_{\text{last}},g}$, append $h_{\text{next}}$
         to $\kappa$;
7      Sample $u_{\text{next}} = u \in \{1,2\}$ with probability $S_{h,u}$;
8      if $u_{next} = 2$ then
9          Set $x \leftarrow$ Solution resulting from applying heurisics in $\kappa$ to $\hat{x}$;
10         if $\hat{Q}(s_t, x, \theta) < \hat{Q}(s_t, \hat{x}, \theta)$ then
11             Set $P_{\kappa_k,\kappa_{k+1}} \leftarrow P_{\kappa_k,\kappa_{k+1}} + \frac{1}{T_{\kappa_k,\kappa_{k+1}}}$, for $k \in \{1, \dots, |\kappa| - 1\}$, and set
               $L_{\kappa_k,u} \leftarrow L_{\kappa_k,u} + \frac{1}{S_{\kappa_k,u}}$ for $k \in \{1, \dots, |\kappa|\}$ and $u \in \{1,2\}$;
12         if $\hat{Q}(s_t, x, \theta) < \left(1 + \varepsilon + \vartheta \left(1 - \frac{t_{elapsed}}{t_{limit}}\right)\right) \hat{Q}(s_t, x^*, \theta)$ then
13             $\hat{x} \leftarrow x$;
14         if $\hat{Q}(s_t, x, \theta) < \hat{Q}(s_t, x^*, \theta)$ then
15             $x^* \leftarrow x$;
16     $h_{\text{last}} \leftarrow h_{\text{next}}$;
17     Update $t_{\text{elapsed}}$ with the current computational time;
18 return $x^*$
```

1. Compute, for each $c \in \mathcal{C}^{\text{unassigned}}$, the insertion position $(k_1^*, k_2^*, x_{t,c}^{\text{new}})$ that minimizes $\hat{Q}(s_t, \hat{x}, \theta)$.
2. Select $c^* \in \arg\min_c \hat{Q}(s_t, x_{t,c}^{\text{new}}, \theta)$ with lowest position index in the sequence $\mathcal{C}^{\text{unassigned}}$ that satisfies the following conditions:
   C1) Courier c^* satisfies $D(\ell_{c^*}, \ell_{r_o}) < \delta^{\text{dist}}$, for given distance threshold $\delta^{\text{dist}} > 0$.
   C2) Courier c^* has not previously rejected o.
   C3) Courier c^* is not currently being offered any order in the incumbent solution x_t, *i.e.*, satisfy $x_t^{c^*,o'} = (0,0)$ for all unassigned orders $o' \in \mathcal{O}^{\text{unassigned}}$.
   C4) Courier c^* has strictly fewer than $N^{\text{o}}_{\text{max}}$ orders currently assigned for delivery.
   C5) $\hat{Q}(s_t, x_{t,c^*}^{\text{new}}, \theta) \leq 0$.
3. If the above criteria are satisfied, then $x_t = x_{t,c^*}^{\text{new}}$.

(h_2^{aux}) Insert not offered orders with randomization. This operator performs randomized insertion of unoffered orders into courier routes. Given a current allocation x_t and two sequences,

$\mathcal{O}^{\text{unassigned}}$, containing orders from $\mathcal{O}_t^{\text{open}}$, and $\mathcal{C}^{\text{unassigned}}$, containing couriers from $\mathcal{C}_t$; both sequences are randomly permuted. The operator then applies h_1^{aux} using the permuted sequences to modify x_t accordingly.

(h_3^{aux}) Remove offers with the furthest deadlines. This procedure removes proposed offers associated with new orders that have the most distant promised delivery times. Given a current allocation x_t, all order-courier pairs $\{(o,c) \in \mathcal{O}_t^{\text{open}} \times \mathcal{C}_t : x_t^{c,o} \neq (0,0)\}$ are sorted in descending order of their delivery deadline d_o. The top k^{remove} offers in the resulting list are then removed by setting $x_t^{c,o} = (0,0)$.

(h_4^{aux}) Randomly remove offers. This procedure randomly removes a subset of proposed offers. Given a proposed allocation x_t, a set of k^{remove} order-courier pairs is randomly selected from $\{(o,c) \in \mathcal{O}_t^{\text{open}} \times \mathcal{C}_t : x_t^{c,o} \neq (0,0)\}$. For each selected pair (c,o), the corresponding assignment offer is removed by setting $x_t^{c,o} = (0,0)$.

We now define the low-level heuristics that compose the set $\mathcal{H}$. All heuristics take as input a current allocation x_t and the sets of unassigned orders $\mathcal{O}_t^{\text{open}}$ and available couriers $\mathcal{C}_t$.

(h_1) Reinsert orders by promised delivery time. This heuristic aims to improve order offers by prioritizing those with the earliest delivery deadlines. The procedure is as follows:

1. Apply h_3^{aux} to remove k^{remove} offers in x_t based on their delivery deadlines.
2. Construct a sequence of orders $\mathcal{O}^{\text{reinsert}}$ containing the removed orders, sorted in ascending order of d_o (earliest deadlines first).
3. Randomly permute the sequence of available couriers $\mathcal{C}_t$.
4. Apply h_1^{aux} to iteratively reinsert the orders in $\mathcal{O}^{\text{reinsert}}$ into the routes of couriers in $\mathcal{C}_t$.

(h_2) Remove orders by distance and reinsert. This heuristic improves order allocation by removing geographically close orders and reinserting them to enhance route efficiency. It proceeds as follows:

1. Select a random order $\bar{o}$ from the set $\{o \in \mathcal{O}_t^{\text{open}} : \exists c \in \mathcal{C}_t : x_t^{c,o} \neq (0,0)\}$.
2. For each order to be offered in x_t (*i.e.*, each o such that $x_t^{c,o} \neq (0,0)$ for some $c \in \mathcal{C}_t$), compute the proximity measure given by $D_o = D(\ell_{\bar{o}}, \ell_o) + D(\ell_{r_{\bar{o}}}, \ell_{r_o})$.
3. Sort all to be offered orders in non-decreasing order of D_o, and remove the k^{remove} offers with the smallest values by setting $x_t^{c,o} = (0,0)$. Let $\mathcal{O}^{\text{removed}}$ denote the removed orders and $\mathcal{C}^{\text{removed}}$ the couriers from whose schedules these orders were removed.
4. For each courier $c \in \mathcal{C}^{\text{removed}}$, locally improve their route as follows. Iterate through all orders in c's route that have not yet been picked up and whose pickup location is not the courier's

next stop. For each such order o, temporarily remove its pickup and delivery tasks and reinsert them in the positions that minimize the estimated cost $\hat{Q}(s_t, \hat{x}, \theta)$. Once all insertions are complete, update $x_t = x_t^*$.

5. Apply h_2^{aux} to reinsert the removed orders $\mathcal{O}^{\text{removed}}$ into the routes of couriers in $\mathcal{C}^{\text{removed}}$.

(h_3) Randomly remove and reinsert orders. This heuristic randomly removes and reinserts orders, promoting solution diversity. It proceeds as follows:

1. Apply h_4^{aux} on x_t to remove k^{remove} offers at random.
2. Reinsert the removed orders by applying h_2^{aux}, which selects couriers at random during reinsertion.

(h_4) Greedy route restructuring. This heuristic refines individual courier routes by greedily reinserting allocated orders not yet picked up.

1. Randomly select k^{restruct} couriers and denote the resulting sequence as $\mathcal{C}_{\text{restruct}}$.
2. Perform the local route improvement procedure described in step 4 of h_2 to the sequence $\mathcal{C}_{\text{restruct}}$, replacing, for each courier $c \in \mathcal{C}_{\text{restruct}}$, its current route $\rho_{t,c}$ with the updated route $\rho_{t,c}^{\text{prop}}$.

(h_5) Exchange orders allocations between couriers. This heuristic balances workloads by exchanging orders between couriers:

1. Randomly select $k^{\text{interchange}}$ pairs of couriers $c_1, c_2 \in \mathcal{C}_t$ who both have allocated orders.
2. For each selected pair (c_1, c_2):
   - Let o_1 and o_2 be the orders to be offered to c_1 and c_2, respectively. Remove these offers by setting $x_t^{c_1,o_1} = x_t^{c_2,o_2} = (0,0))$.
   - Apply h_1^{aux} to insert the removed orders, using $\mathcal{O}^{\text{unassigned}} = (o_1, o_2)$ and $\mathcal{C}^{\text{unassigned}} = (c_1, c_2)$, then proceed to the next pair.

(h_6) Random removal, local optimization, and reinsertion. This heuristic combines random removal with local route optimization and reinsertion to explore diverse solution neighborhoods.

1. Remove k^{remove} offers from x_t using h_4^{aux}. Let $\mathcal{O}^{\text{removed}}$ be a random sequence of removed orders, and $\mathcal{C}^{\text{removed}}$ a random sequence of the couriers whose routes were affected.
2. Perform step 4 of h_2 on the courier set $\mathcal{C}^{\text{removed}}$.
3. Reinsert Apply h_2^{aux} to reinsert the removed orders $\mathcal{O}^{\text{removed}}$ into the corresponding routes of couriers $\mathcal{C}^{\text{removed}}$.

(h_7) Remove by delivery time, improve routes, and reinsert. This heuristic enhances courier schedules by removing orders with latest delivery deadlines, locally optimizing affected routes and then reinserting the removed orders in deadline order.

- Apply h_3^{aux} to remove from x_t the offers involving the k^{remove} orders with the latest promised delivery times. Let $\mathcal{C}^{\text{removed}}$ be a random sequence of couriers whose routes were affected by these removals.
- Perform step 4 of h_2 on the courier set $\mathcal{C}^{\text{removed}}$.
- Create the sequence $\mathcal{O}^{\text{reinsert}}$ by sorting all unassigned orders $\{o \in \mathcal{O}_t^{\text{open}} : x_t^{c,o} = (0,0), \forall c \in \mathcal{C}_t\}$ in ascending order of d_o. Then generate a random permutation of the couriers in $\mathcal{C}^{\text{remove}}$.
- Apply h_1^{aux} using the sets $\mathcal{O}^{\text{reinsert}}$ and $\mathcal{C}^{\text{remove}}$ to reinsert removed orders.

Using the definitions above, we define the set of heuristics as $\mathcal{H} = \{h_1, h_2, h_3, h_4, h_5, h_6, h_7\}$.

4.4. Policies

This section presents four benchmark policies used to evaluate our proposed framework, followed by the main policy developed in this work. The first two are simple, fast heuristics that approximate "business-as-usual" dispatching strategies commonly observed in practice. The remaining two incorporate learning-based and metaheuristic components, while the final policy combines these elements into a unified RL and hyper-heuristic framework.

4.4.1. Fastest policy. This policy prioritizes computational speed. It offers orders to couriers solely based on proximity, with each new order appended to the end of a courier's current route. The objective is to minimize delivery lateness penalties without global route optimization.

The fastest policy operates as follows:

1. Initialize x_t such that $x_t^{c,o} = (0,0)$ for all $c \in \mathcal{C}_t$ and $o \in \mathcal{O}_t$.
2. Randomly sample $|\mathcal{O}_t^{\text{open}}|$ orders (with replacement) from $\mathcal{O}_t^{\text{open}}$.
3. For each sampled order o:
    - If the set of courier candidates to have this order offered to, $\mathcal{C}_o^{\text{sample}}$, is uninitialized or empty:
        - —Randomly sample $|\mathcal{C}_t|$ couriers (with replacement) from $\mathcal{C}_t$ and include them in $\mathcal{C}_o^{\text{sample}}$.
        - —Remove any courier $c \in \mathcal{C}_o^{\text{sample}}$ who has previously rejected o or currently has $N_{\max}^{\text{o}}$ or more assigned orders.
    - For each remaining courier $c \in \mathcal{C}_o^{\text{sample}}$, compute the incremental distance $D^{incr}(\rho_{t,c}, o)$ incurred by appending order o to the end of $\rho_{t,c}$.
    - Sort $\mathcal{C}_o^{\text{sample}}$ in non-decreasing order of $D^{incr}(\rho_{t,c}, o)$.

- Offer o to the first courier in $\mathcal{C}_o^{\text{sample}}$ by inserting its pickup and delivery tasks at the end of their route, *i.e.*, let such a courier be c, set $x_t^{c,o} = (|\rho_{t,c}|+1, |\rho_{t,c}|+2)$.
- Remove that courier from C_o^{sample}.

4.4.2. Simple policy. This policy selects the insertion points for each order's pickup and delivery tasks that minimize the increment of the immediate cost c_0, which combines both travel distance and expected lateness. Specifically, the procedure is as follows:

1. Initialize x_t so that $x_t^{c,o} = (0,0)$ for all $c \in \mathcal{C}_t$ and $o \in \mathcal{O}_t$.
2. Generate a random permutation of $\mathcal{O}_t^{\text{open}}$.
3. For each order o in the permutation:
   - Determine the set of active couriers C_o^{feas} that satisfy all of the following:
     Ca) Have fewer than $N_{\max}^{\text{o}}$ assigned orders.
     Cb) Are not currently being offered any order in the incumbent solution x_t, *i.e.*, couriers c satisfying $x_t^{c,o'} = (0,0)$ for all unassigned orders $o' \in \mathcal{O}_t^{\text{open}}$.
     Cc) Have not previously rejected order o.
   - If $C_o^{\text{feas}} = \emptyset$, continue to the next order.
   - Otherwise, determine the courier c^* and insertion positions (k_1^*, k_2^*) for the pickup and delivery tasks of order o of minimum incremental cost among feasible couriers, which is done by solving:

$$\begin{aligned}(c^*, k_1^*, k_2^*, x_t^*) \in \; & \underset{\substack{c \in \mathcal{C}_t^{\text{feas}},\\ k_1 \in K_1(\rho_{t,c}),\\ k_2 \in K_2(k_1, \rho_{t,c}),\\ \hat{x}}}{\arg\min} \quad c_0(\hat{x}, s_t) \\ \text{s.t.} \quad & \hat{x}^{c',o'} = \begin{cases} x_t^{c',o'}, & \text{if } o' \neq o \text{ or } c' \neq c, \\ (k_1, k_2), & \text{otherwise.} \end{cases}\end{aligned} \tag{3}$$

   - Set $x_t = x_t^*$.
4. Return x_t as the final set of order offers.

Although more computationally demanding than the fastest policy, this approach yields more cost-efficient assignments. Note that it does not modify existing routes with accepted orders, focusing instead on constructing offers for currently unassigned orders.

4.4.3. Hyper-policy. This policy uses only the metaheuristic component of our framework, described in Section 4.3, focusing exclusively on short-term cost minimization while retaining the structured exploration of the heuristic sequence search. It does not incorporate the learning-based value function. Specifically, during the search process (*e.g.*, in Lines 10, 12, 14 of Algorithm 2, and in heuristics h_1^{aux}, h_2, h_4, h_5, h_6, h_7), candidate solutions are evaluated using the immediate cost function $c_0(s_t, x_t)$ instead of the learned approximation $\hat{Q}(s_t, x_t, \theta^{\text{best}})$. Additionally, this variant removes condition C5), thereby disabling order postponement.

4.4.4. n-step SARSA policy. This policy extends the simple policy by incorporating a learned value function to account for the long-term effects of current decisions. While the simple policy focuses on minimizing the immediate cost $c_0(s_t, x_t)$, the n-step SARSA policy instead minimizes the approximate expected future cost $\hat{Q}(s_t, x_t, \theta^{\text{best}})$, where θ^{best} is the parameter vector learned via Algorithm 1. This enables the policy to balance short-term efficiency with long-term operational performance.

This policy operates similarly to the simple policy described in Section 4.4.2, iterating through a random permutation of unassigned orders and evaluating feasible order-courier assignments. However, it introduces two key extensions. First, the immediate cost function $c_0(s_t, x_t)$ used in the simple policy is replaced with the learned value function approximation $\hat{Q}(s_t, x_t, \theta^{\text{best}})$, allowing decisions to incorporate expected future cost. Second, a postponement condition is added: an offer is made only if $\hat{Q}(s_t, x_t, \theta^{\text{best}}) \leq 0$; otherwise, the order is deferred to a later decision epoch. This enables the policy to skip low-value offers and prioritize those expected to improve long-term performance.

Because this policy relies on a greedy construction, it iterates through a random permutation of unassigned orders. While deterministic sorting rules (*e.g.*, earliest deadline) could be adopted, randomization avoids systematic biases, ensuring each order has an equal chance of being evaluated first. This design encourages exploration during training and contributes to a more robust learned policy.

4.4.5. n-step hyper-policy. This is the main policy proposed in this work, combining value function approximation, n-step SARSA, and the hyper-heuristic within a unified learning framework. In this joint scheme, SARSA updates the value function parameters based on the actions generated by the hyper-heuristic, while the hyper-heuristic adaptively adjusts its transition weights based on the evolving value estimates. This interaction allows the framework to jointly learn effective search behaviors and value function representations, combining the exploration capacity of metaheuristics with long-term predictive capabilities of RL.

5. Data processing

The primary data source for this study is the publicly available dataset from the First INFORMS TSL Data-Driven Research Challenge, developed in collaboration with Meituan. The dataset is documented on the official INFORMS TSL website and hosted in the official GitHub repository in Zhao et al. (2024). It spans eight days of operations and provides detailed records of key operational variables, including order dynamics (*i.e.*, preparation times, restaurant and delivery locations), completion times (sum of the travel and service durations), and courier-level data such as order acceptance, rejection, and working hours.

Despite its richness, the dataset presents two main challenges for training robust learning models. First, it is not large enough to train complex models effectively. Second, it only contains information on actions that were actually taken, providing no direct observations of the outcomes of alternative (counterfactual) decisions.

To mitigate these limitations, we employ two complementary strategies. To address data scarcity, we generate additional samples by drawing order and courier realizations from the empirical distribution observed in the dataset, ensuring realistic variability across simulated days. To handle the lack of counterfactual outcomes, we develop a simulator that estimates delivery completion times and courier acceptance probabilities for given offers and route configurations. This approach allows us to train and test our simulator and learning framework independently, enhancing our approach's generalization.

The remainder of this section introduces the main components of the simulator and presents the results from training and testing its predictive modules. We conclude by outlining the overall data flow and interaction between the simulator and the sequential decision-making model.

5.1. Simulator

The simulator consists of four core components: order arrivals, courier connections and disconnections, completion time generation, and couriers' acceptance of order offers. Order and courier dynamics are modeled using empirical distribution-based sampling. For predicting completion times, we compare a linear regression approach with boosting regression trees. To predict courier order acceptance decisions, we evaluate logistic regression and boosting classification trees. All methods correspond to standard ML techniques as described in Hastie et al. (2009). To ensure robust evaluation and prevent overfitting, we employ an 80-20 split, reserving 80% of the observations for model training and 20% for testing. The following subsections describe the implementation and results of each component in detail.

5.1.1. Orders generation. For each order in the dataset, we record its placement time of day ($\bar{h}:\bar{m}:\bar{s}$), where $\bar{h}$, $\bar{m}$ and $\bar{s}$ represent the hour (0-23), minute (0-59), and second (0-59), respectively. These timestamps serve as potential order arrival times in the simulation. During each simulated day, when the simulation clock reaches a historical timestamp, each order whose placement time matches the timestamp is generated with probability p^{activ}. This mechanism reproduces the empirical temporal distribution of order placements while introducing stochastic variability across simulated runs.

5.1.2. Couriers connections. Since the dataset does not explicitly record courier work schedules, we first reconstruct historical working blocks by analyzing activity patterns. For each courier,

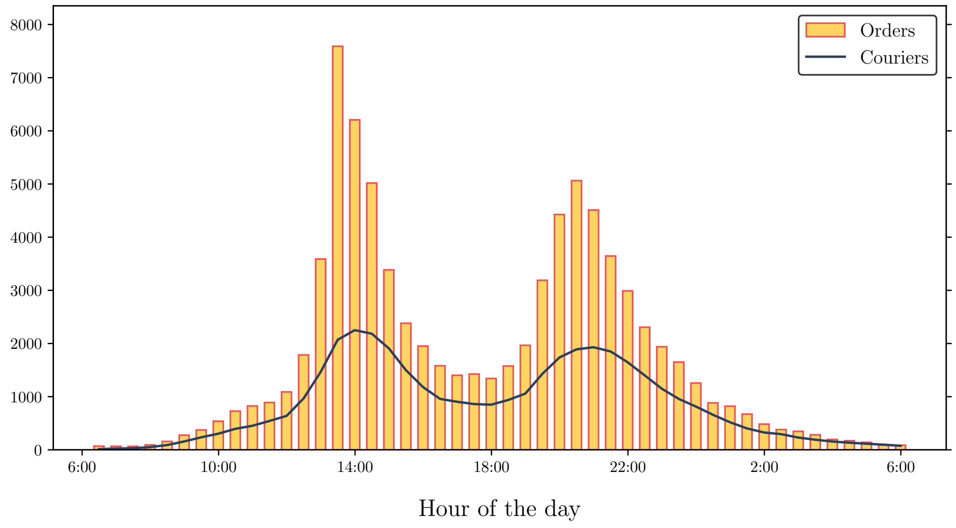

Figure 1 **Temporal distribution of daily order arrivals and courier availability across hours of operation.**

the first observed activity (*i.e.*, initial order acceptance) marks the start of the first block. Subsequent activities are considered part of the same block if they occur within one hour of the previous activity. When the time gap between consecutive activities exceeds one hour, we interpret this as a block transition: the earlier activity marks the end of the one block, and the next marks the beginning of another.

Using these reconstructed patterns, we simulate courier availability as follows. For each courier, we extract all times of day $(\bar{h}:\bar{m}:\bar{s})$ at which the courier was observed to connect on any day in the dataset. During simulation, these historical connection times act as potential entry points. At each such timestamp, a courier enters the system with probability p^{conn}, provided that they are not already active at that moment. Otherwise, the connection event is ignored and the courier continues their ongoing duties. Figure 1 illustrates the hourly average number of generated orders and courier connections produced by the simulator.

5.1.3. Completion times. We evaluate two modeling approaches for predicting travel plus service times: linear regression and boosting regression trees.

The linear regression model assumes that completion times depend on the distance $D(\ell,\ell')$ between locations ℓ and ℓ', with an additional bias term for restaurant destinations:

$$\tau_{\ell,\ell',t} = \beta_0^c + \beta_1^c D(\ell,\ell') + \beta_2^c \mathbb{1}_{[\ell'\in\mathcal{L}_R]} + \varepsilon_t^c,$$

where ε_t^c is a zero-mean Gaussian noise term.

Regression trees partition the feature space into disjoint regions R_m, each associated with a constant value ζ_m representing the average completion time of observations within the region. If the loss function is the sum of squared errors, the solution is a set of regions R_m partitioning the

Metric	**Linear regression**	**Regression trees**
Mean Squared Error (MSE)	68141.14	46024.76
Root Mean Squared Error (RMSE)	261.04	214.53
Mean Absolute Error (MAE)	155.47	146.30
R^2	0.2921	0.5093
Adjusted R^2	0.2921	0.2965
Weighted Mean Absolute Percentage Error (WMAPE)	39.35%	35.49%

Table 1 Comparison of predictive performance on testing data between linear regression and regression trees for estimating order completion times.

independent variables in the training data, and each constant ζ_m is the average of the observations of the dependent variable, associated with observations contained in region R_m.

Boosting regression trees are the result of successive iterations of this procedure in the residuals that result from each fitting of regression trees. The parameters of the j-th iteration of this procedure are denoted as $\{R_{m,j}, \zeta_{m,j}\}_{m=1}^{M_j}$, where $M_1 = 1$, $R_{1,1}$ is the domain for the independent variables and $\zeta_{1,1}$ is the average of the dependent variable across all observations.

Let $\varsigma(t)$ represent the number of seconds elapsed since the start of the workday at 6:00:00 AM, measured at time t. The workday spans from 6:00:00 AM to 5:59:30 AM the following day, meaning $\varsigma(t)$ resets at the start of each new workday. The feature set used for partitioning includes the distance between locations $D(\ell, \ell')$, courier and restaurant attributes, and temporal components represented as:

$$TC_t = \left\{ \left(\cos\left(2\pi(i+1)\frac{\varsigma(t)}{3600 \cdot 24}\right), \sin\left(2\pi(i+1)\frac{\varsigma(t)}{3600 \cdot 24}\right) \right) \right\}_{i=0}^{5}.$$

where $3600 \cdot 24$ represents the number of seconds in a day. Thus, the completion time model employed by the simulator is formulated as:

$$\tau_{\ell,\ell',t} = \sum_{j=1}^{J} \sum_{m=1}^{M_j} \zeta_{m,j} \mathbb{1}_{\left[\left(c, D(\ell,\ell'), \ell' \mathbb{1}_{[\ell' \notin \mathcal{L}_R]}, r(\ell'), TC_t\right) \in R_{m,j}\right]},$$

where location indicators are defined as follows: $\ell' \mathbb{1}_{[\ell' \notin \mathcal{L}_R]}$ equals ℓ' for delivery locations and -1 for pickup locations, while $r(\ell')$ returns the restaurant identifier for pickup locations and -1 otherwise. The model employs $J = 200$ boosting iterations with a step size shrinkage parameter of 0.025 after each iteration.

Table 1 compares several metrics for predicting completion times using linear regression and boosting regression trees. Notably, regression trees achieve lower mean squared error (MSE), higher R^2, and lower weighted mean absolute percentage error (WMAPE), which weights the errors by the average times to avoid large values in cases the target and prediction are small values. These results highlight the effectiveness of more complex methods to predict completion times across all metrics.

5.1.4. Orders acceptance and rejection. At each decision epoch t, offered order-courier pairs $\mathcal{P}_t^{\text{offered}}$ are classified as either accepted ($\tilde{\mathcal{P}}_{t+1}^{\text{accepted}}$) or rejected ($\tilde{\mathcal{P}}_{t+1}^{\text{rejected}}$). For each pair $(o,c) \in \mathcal{P}_t^{\text{offered}}$, the offer is automatically rejected if the estimated delivery completion time t_k^{prop} (see Appendix A) exceeds the courier's block end time $\sigma_{c,i}^{end}$. Otherwise, couriers' acceptance decisions are modeled using a predictive machine learning model. Importantly, $\sigma_{c,i}^{end}$ is not known to the planner and is never used in the decision-making process; it is used solely by the simulator to determine whether a courier would accept or reject an offer. We evaluate two predictive models for order acceptance: logistic regression and boosting classification trees.

Logistic regression model. In this model, the probability that a courier c accepts order o at time t is

$$\mathbb{P}(y_{c,o,t} = 1) = \frac{\exp(U_{c,o,t})}{1 + \exp(U_{c,o,t})}$$

where $U_{c,o,t}$ denotes the utility function and is given by:

$$\begin{aligned} U_{c,o,t} = {} & \beta_1^a D(\ell_{r_o}, \ell_o) + \beta_2^a D(\ell^{t,c}, \ell_{r_o}) + \beta_3^a N_{t,c}^{\text{on-hand}} + \beta_4^a \max_{o' \in \mathcal{O}_{t,c}^{\text{on-hand}}} D(\ell_{r_{o'}}, \ell_{o'}) \\ & + \beta_5^a \mathbb{1}_{[o \text{ is pre-booked}]} + \beta_6^a \mathbb{1}_{[w_t=1]} + \beta_7^a \tau_{t,c}^{\text{duty}} + \varepsilon_{c,o,t}^a, \end{aligned}$$

Here, $\mathcal{O}_{t,c}^{\text{on-hand}} := \{o : \exists (t', \ell, q) : (t', \ell, o, q) \in \rho_{t,c}\}$ is the set of orders that by time t are assigned to courier c, $N_{t,c}^{\text{on-hand}} := | \mathcal{O}_{t,c}^{\text{on-hand}} |$, and $\varepsilon_{c,o,t}^a$ is a random error term capturing unobserved variation.

This formulation captures key decision factors, including:

- Order and courier distances: Distance between the restaurant and delivery location $D(\ell_{r_o}, \ell_o)$, and between the courier's next destination and the restaurant $D(\ell_{t,c}, \ell_{r_o})$.
- Workload: The courier's number of ongoing orders $N_{t,c}^{\text{on-hand}}$ and the longest active delivery distance, $\max_{o' \in \mathcal{O}_{t,c}^{\text{on-hand}}} D(\ell_{r_{o'}}, \ell_{o'})$.
- Contextual factors: Binary indicators for pre-booked orders, $\mathbb{1}_{[o \text{ is pre-booked}]}$, and weekend activity, $\mathbb{1}_{[w_t=1]}$.
- Fatigue: Time elapsed since the courier began working, $\tau_{t,c}^{\text{duty}}$.

Boosting classification trees. These trees partition the feature space into decision regions based on learned rules. The acceptance probability is estimated as:

$$\mathbb{P}(y_{c,o,t} = 1) = \sum_{j=1}^{J} \frac{1}{|R_{m,j}|} \sum_{OC_{c',o',t'} \in R_{m,j}} \mathbb{1}_{[y_{c',o',t'}=1]}$$

where $R_{m,j}$ denote the partition regions and the feature vector is

$$OC_{c,o,t} = (c, r_o, \bar{h}_t, D(\ell_{r_o}, \ell_o), D(\ell^{t,c}, \ell_{r_o}), N_{t,c}^{\text{on-hand}}, \max_{o' \in \mathcal{O}_{t,c}^{\text{on-hand}}} D(\ell_{r_{o'}}, \ell_{o'}), \mathbb{1}_{[o \text{ is pre-booked}]}, \mathbb{1}_{[w_t=1]}, \tau_{t,c}^{\text{duty}})$$

Metric	**Logistic regression**	**Classification trees**
Accuracy	0.748	0.877
True positive rate (sensitivity)	0.827	0.873
True negative rate (specificity)	0.670	0.881
F1-score	0.765	0.877
ROC AUC	0.826	0.952
Log loss (cross-entropy loss)	0.506	0.283
Gini index	0.652	0.903
Prevalence	0.497	0.270
Matthews correlation coefficient	0.502	0.754

Table 2 Comparison of predictive performance on testing data between logistic regression and classification trees for courier-order acceptance.

Unlike logistic regression, boosting classification trees can directly handle categorical variables such as courier and restaurant identifiers, as well as temporal features like hour of the day $\bar{h}_t$. The model is trained with $J = 600$ iterations and a learning rate parameter for the step size shrinkage of 0.05.

Table 2 reports performance metrics for both models. Boosting classification trees outperform logistic regression across all metrics, including accuracy, sensitivity, specificity, F1-score, and ROC AUC, while also achieving lower log-loss and higher Gini coefficients. Although the tree model predicts slightly fewer accepted orders (lower prevalence, see Table 2), metrics insensitive to class imbalances (*e.g.*, Mathews correlation coefficient) confirm its superior performance.

While tree-based models (regression trees for travel times and classification trees for order acceptance/rejection) exhibit higher predictive capabilities, the simpler linear and logistic regression models achieve competitive results at a fraction of the computational cost. Given the trade-off between model complexity, predictive power and computational efficiency, we adopt the simpler models for the simulator, enabling faster training iterations and improved stability within the learning framework. Figure 2 illustrates the simulator's data flow and its training-testing pipeline.

5.2. Flow of data for the learning framework

The RL framework is trained using a combination of simulated data and a subset of real observations from the Meituan dataset. Figure 3 illustrates the overall flow of information, highlighting how the simulator, policy and environment interact throughout the learning process.

Each training cycle begins by sampling orders and courier connections for a given time period t. The system then updates its state by adjusting the sets of active couriers and unassigned orders. Couriers who complete their working block within the interval are removed from the system. Based on the updated state, the policy prescribes order offers and route adjustments for available couriers.

The simulator subsequently generates acceptance outcomes and travel times for the proposed offers. Using this information, the system transitions to the next state, updating order statuses (assigned, unassigned, or rejected), courier routes and locations, and other time-dependent state

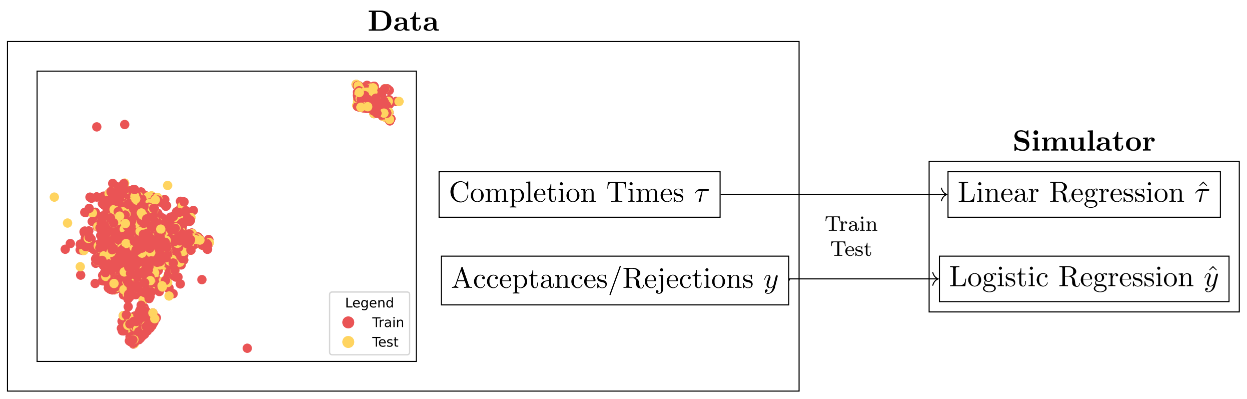
Data
Legend
Train
Test
Completion Times τ
Acceptances/Rejections y
Train
Test
Simulator
Linear Regression $\hat{\tau}$
Logistic Regression $\hat{y}$

Figure 2 **Illustration of the simulator's data flow and training process. Observations are randomly partitioned into training (red dots) and testing (yellow dots) sets, with the split performed independently of geographical locations.**

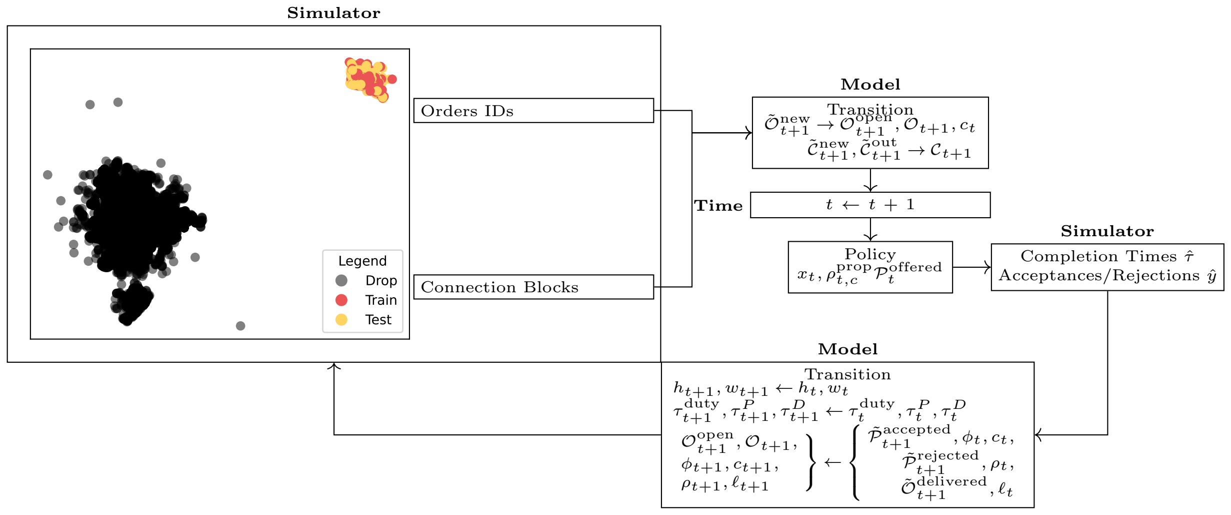
Simulator
Legend
Drop
Train
Test
Orders IDs
Connection Blocks
Time
Model
Transition
$\tilde{\mathcal{O}}^{\text{new}}_{t+1} \to \mathcal{O}^{\text{open}}_{t+1}, \mathcal{O}_{t+1}, c_t$
$\tilde{\mathcal{C}}^{\text{new}}_{t+1}, \tilde{\mathcal{C}}^{\text{out}}_{t+1} \to \mathcal{C}_{t+1}$
$t \leftarrow t+1$
Policy
$x_t, \rho^{\text{prop}}_{t,c} \mathcal{P}^{\text{offered}}_t$
Simulator
Completion Times $\hat{\tau}$
Acceptances/Rejections $\hat{y}$
Model
Transition
$h_{t+1}, w_{t+1} \leftarrow h_t, w_t$
$\tau^{\text{duty}}_{t+1}, \tau^P_{t+1}, \tau^D_{t+1} \leftarrow \tau^{\text{duty}}_t, \tau^P_t, \tau^D_t$
$\left.\begin{array}{l}\mathcal{O}^{\text{open}}_{t+1}, \mathcal{O}_{t+1},\\ \phi_{t+1}, c_{t+1},\\ \rho_{t+1}, \ell_{t+1}\end{array}\right\} \leftarrow \left\{\begin{array}{r}\tilde{\mathcal{P}}^{\text{accepted}}_{t+1}, \phi_t, c_t,\\ \tilde{\mathcal{P}}^{\text{rejected}}_{t+1}, \rho_t,\\ \tilde{\mathcal{O}}^{\text{delivered}}_{t+1}, \ell_t\end{array}\right.$

Figure 3 **Flow of information and interactions with the sequential decision making framework.**

variables. Orders that remain unassigned for more than λ_{lost} hours past their delivery deadlines are removed and penalized as lost orders. The process then repeats for the next time interval, continuously simulating the dynamic environment under which the RL framework is trained.

6. Computational study

In this section, we present the computational results evaluating the performance of our proposed framework. The experiments assess both the quality of the value function approximation and the effectiveness of the integrated n-step hyper-heuristic procedure (Section 4.3) for sequential decision-making in the online order assignment and routing problem. We benchmark our approach against the rest of the policies introduced in Section 4.4.

6.1. Experimental setup

Decision epochs occur every $\Delta = 30$ seconds, resulting in $24 \cdot 3600/\Delta = 2880$ epochs per 24-hour period. The value function uses a discount factor of $\gamma = 0.999$. The reward structure comprises penalty values $K^{\text{dist}} = 1$, $K^{\text{late}} = 4$, and $K^{\text{lost}} = 4 \cdot K^{\text{late}} = 16$. Orders are considered lost if they remain unassigned for more than $\lambda_{\text{lost}} = 2$ hours past their delivery deadline. The maximum courier capacity is set to $N^{\text{o}}_{\text{max}} = 4$, consistent with the dataset, where 98.2% of couriers handle four or fewer concurrent orders.

For the value function approximation, we test learning rates $\alpha \in \{10^{-7}, 10^{-6}, 10^{-5}\}$, memory sizes $m \in \{3000, 5000, 9000\}$, and a fixed batch size $b = 20$. All learning-based policies, *i.e.*, n-step SARSA and the n-step hyper-heuristic, are trained and tested with look-ahead steps $n \in \{2, 1000, 3000\}$, update frequency $n_{\text{update}} = 200$, and initial parameter vector $\theta_0 = \mathbf{0}$. The employed number of features is $n_{\text{feat}} = 14$. Features $\phi_i, i \in \{4, 5, \ldots, 11\}$ are constructed using $m_{\text{feat}} = 8$ intervals of $\Delta_{\text{feat}} = 1200$ seconds. Training runs for $t_{\max} = 10^5$ decision epochs.

Hyper-heuristic parameters for both the hyper-policy and the n-step hyper-policy are provided in Appendix B. After each execution, the updated transition matrix parameters P and L are saved and reused as initial matrices P^0 and L^0 in the next run to accelerate convergence.

Each policy is evaluated over 50 independent test episodes, each representing a 24-hour operating period (6:00:00 AM to 5:59:59 AM of the following day). Performance metrics in Algorithm 1 (*i.e.*, $\hat{V}(s_0, \theta)$ and $\hat{V}(s_0, \theta^{\text{best}})$) are estimated using Monte Carlo simulation over a single 24-hour episode. Specifically, given an initial state s_0 and a parameter vector θ, the system is simulated under the corresponding policy to completion, and the total accumulated cost across decision epochs is recorded. The resulting cost serves as the empirical estimate of $\hat{V}(s_0, \theta)$. Results from policies that fail to complete all 50 test episodes within a four-day runtime limit are discarded.

Route construction ($\rho^{\text{prop}}_{t,c}$) assumes an average urban travel speed of $v = 18$ km/h. The simulator operates at a 1:10 scale relative to real operations, processing approximately 600 orders and 55 couriers per day. Specifically, for both order generation and courier connections, we set $p^{activ} = p^{conn} = 0.1/N_{days}$, where $N_{days} = 7.96$ corresponds to the number of observed operational days in the dataset, used to sample orders and connection blocks empirically (see further details in Section 5.1). Task completion times in proposed routes (t^k_{prop}) use fixed pickup and delivery service time estimates $\hat{\mu}_p = \hat{\mu}_d = 3.5$ minutes. These are planner approximations used for efficient planning; in contrast, the simulation environment treats these service times as random variables (μ_p and μ_d, defined in Section 3) to represent real-world stochasticity. The route construction procedure is detailed in Appendix A.

We empirically validate the contribution of each low-level heuristic for the n-step hyper-policy in Appendix C.

All experiments were conducted in Java 17 on a CentOS Linux computing cluster with four 8-core Intel Xeon-2670 processors and 128 GB of RAM.

6.2. Learning hyper-parameters selection

Table 3 reports the cost and runtime performance of all strategies under different hyper-parameter configurations. With the selected discount factor, higher values of n ($n \in \{1000, 3000\}$) dilute the bootstrapping effect ($\gamma^{999} \approx 0.37$, $\gamma^{2999} \approx 0.049$). In contrast, when $n = 2$, the algorithm is equivalent to Q-learning (*i.e.*, takes the best action solely based on the $\hat{Q}$ estimate, see Algorithm 1), where bootstrapping has a strong influence.

The sensitivity analysis confirms that value function approximation policies are highly sensitive to hyper-parameter settings. Larger n values ($n \in 1000, 3000$) generally outperform Q-learning ($n = 2$), suggesting that bootstrapping can be detrimental in this environment. Interestingly, however, when $n = 2$, the value function approximation achieves the largest transportation cost reductions, implying that the rewards structure (K^{dist}, K^{late} and K^{lost}) may bias learning towards service quality rather than routing efficiency. Overall, the results suggest that variance reduction strategies (*e.g.*, avoiding bootstrapping, using small learning rates α, and larger memory sizes m), tend to perform better when service quality is the dominant objective. From a computational perspective, the last two columns of Table 3 show that while training times vary considerably across policies, inference times per episode remain low, supporting the practical feasibility of offline training with real-time deployment.

For n-step SARSA, the best configuration is $(n, m, \alpha) = (3000, 3000, 10^{-6})$, which achieves the lowest total cost among tested settings. For n-step hyper-policy, the best configuration is $(n, m, \alpha) = (1000, 9000, 10^{-5})$. These results highlight the sensitivity of the proposed approaches to hyper-parameter tuning. Cost reductions are primarily driven by improvements in either transportation or lateness penalties; note that certain parameter combinations may favor one type of cost reduction over the other, with only a few configurations effectively balancing both.

During testing, we observed that in some configurations the n-step hyper-policy, guided by its learned parameter vector θ, exhibited excessive postponement of orders. This behavior led to an accumulation of orders in the system, which in turn caused the low-level heuristics to experience polynomial growth in runtime (see, *e.g.*, the **NStepHyper** cases with $n = 2$). In other words, as more orders were postponed, the enlarged action space prolonged heuristic search and significantly increased solution times. Despite these effects, all optimized policies in their best hyper-parameter configurations outperform the fastest policy baseline.

6.3. Business operational insights

Cost improvement across days. Figure 4 shows the relative cost improvements of each policy compared with the simple policy baseline. We observe that performance differences are most pronounced during high-cost days, where the proposed approaches achieve substantial savings under

Policy	n	**batch**	α	**Objective (%)**	**Transp. (%)**	**Penalty (%)**	**Runtime (hours)**	**Training (hours)**
FastestPolicy				33.40	61.79	-13.85	0.00	-
SimplePolicy				0.00	0.00	0.00	0.00	-
HyperPolicy				-10.29	-15.33	-1.90	0.43	-
NStepSarsa	2	3000	1.0E-05	1168.78	1925.80	-91.40	1.02	0.76
			1.0E-06	990.41	1632.67	-78.74	1.55	1.11
			1.0E-07	695.46	1145.23	-53.25	1.73	1.13
		5000	1.0E-05	1238.47	2039.88	-95.60	1.35	1.12
			1.0E-06	980.45	1616.62	-78.56	1.70	1.32
			1.0E-07	797.31	1314.55	-63.72	2.02	1.43
		9000	1.0E-05	528.76	870.06	-39.40	3.48	2.64
			1.0E-06	536.89	884.80	-42.25	3.17	2.46
			1.0E-07	973.66	1605.66	-78.39	2.31	1.88
	1000	3000	1.0E-05	8.72	8.76	8.65	0.50	0.48
			1.0E-06	31.98	48.60	4.31	0.49	0.48
			1.0E-07	178.48	289.80	-6.83	0.40	0.37
		5000	1.0E-05	10.46	15.24	2.49	0.84	0.82
			1.0E-06	44.84	73.53	-2.92	0.85	0.84
			1.0E-07	144.51	237.04	-9.52	0.74	0.69
		9000	1.0E-05	34.28	51.15	6.20	1.48	1.47
			1.0E-06	77.24	129.61	-9.94	1.48	1.39
			1.0E-07	13.83	15.72	10.69	1.34	1.34
	3000	3000	1.0E-05	21.60	32.86	2.86	0.55	0.52
			1.0E-06	-12.21	-19.92	0.64	0.44	0.43
			1.0E-07	13.34	13.42	13.22	0.41	0.40
		5000	1.0E-05	51.16	86.28	-7.31	0.84	0.75
			1.0E-06	519.23	854.10	-38.23	0.95	0.78
			1.0E-07	27.14	39.90	5.90	0.86	0.85
		9000	1.0E-05	9.17	13.65	1.72	1.35	1.34
			1.0E-06	10.08	18.98	-4.73	1.39	1.35
			1.0E-07	185.56	304.43	-12.33	1.47	1.42
NStepHyper	2	3000	1.0E-05	1141.49	1880.87	-89.31	13.60	5.81
			1.0E-06	899.54	1483.05	-71.81	19.98	7.49
			1.0E-07	637.10	1050.98	-51.89	24.07	8.32
		5000	1.0E-05	1158.16	1908.24	-90.47	24.93	15.40
			1.0E-06	1033.22	1703.01	-81.75	19.05	9.27
			1.0E-07	732.33	1207.48	-58.62	19.98	9.45
		9000	1.0E-05	-	-	-	-	-
			1.0E-06	1069.31	1762.80	-85.12	18.83	8.05
			1.0E-07	830.95	1370.39	-67.04	18.68	8.42
	1000	3000	1.0E-05	35.27	52.43	6.71	3.95	2.35
			1.0E-06	268.90	440.69	-17.07	12.57	1.97
			1.0E-07	25.67	41.50	-0.68	5.86	2.71
		5000	1.0E-05	14.32	19.56	5.60	2.79	2.10
			1.0E-06	6.63	4.07	10.90	3.20	2.52
			1.0E-07	15.01	24.83	-1.34	2.46	1.74
		9000	1.0E-05	**-13.04**	-20.40	-0.80	3.17	2.78
			1.0E-06	46.32	77.76	-6.03	5.71	2.83
			1.0E-07	-5.25	-16.14	12.88	3.26	2.73
	3000	3000	1.0E-05	57.32	85.70	10.07	3.72	2.23
			1.0E-06	-1.12	-1.53	-0.45	3.15	2.63
			1.0E-07	455.49	741.09	-19.93	9.48	1.53
		5000	1.0E-05	1182.77	1948.78	-92.38	6.93	2.91
			1.0E-06	76.99	130.07	-11.37	5.84	2.46
			1.0E-07	22.38	30.30	9.21	2.74	2.09
		9000	1.0E-05	67.62	113.56	-8.86	6.59	3.39
			1.0E-06	19.79	24.18	12.47	3.76	2.70
			1.0E-07	3.27	1.54	6.14	2.99	2.55

Table 3 Percentage cost reduction achieved by different policies relative to the simple policy. Results are decomposed into total cost (column Objective), transportation cost (column Transp.), and penalty cost from delays and lost orders (column Penalty). The last two columns report the total runtime and the training time of the value function approximation scheme. Entries marked with "-" indicate that the policy did not complete the required 50 testing episodes within the 4-day computational time limit.

challenging operational conditions. Conversely, during low-cost days, all policies perform similarly, suggesting that simpler approaches suffice when operational complexity is low. Statistical analysis of policy performance reveals a clear and consistent ranking, confirming that our methods remain robust across both complex and routine operating conditions.

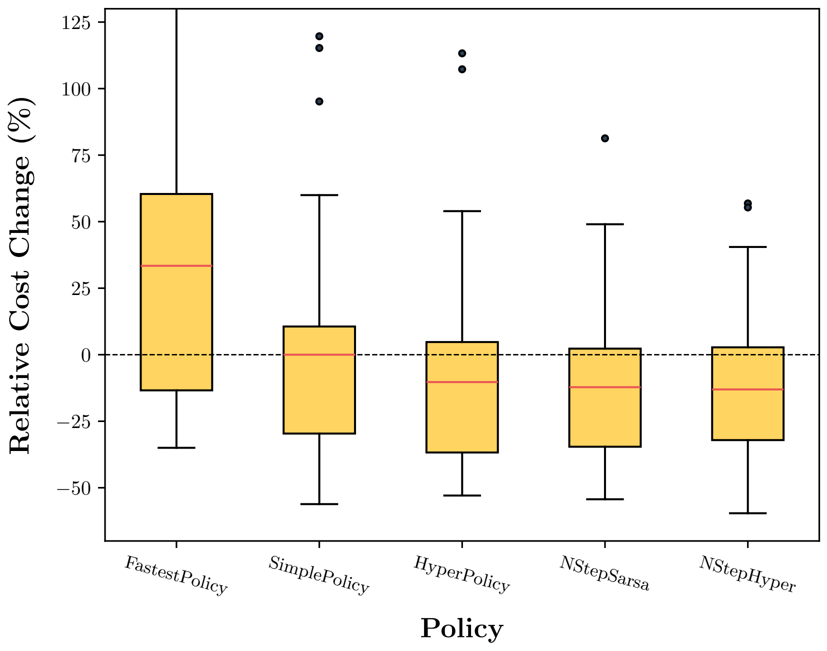

Figure 4 **Box plot of total cost reduction of the considered policies. Used parameters for n-step SARSA and n-step hyper-heuristic are** $(n = 3000, m = 3000, \alpha = 10^{-6})$ **and** $(n = 1000, m = 9000, \alpha = 10^{-5})$**, respectively.**

Performance of the fastest policy. Table 4 shows that the fastest policy achieves the lowest traveled distance per completed order among all approaches. Specifically, its ratio of total distance to completed orders is approximately 0.57 km per order ($\approx 323.66/(600 \cdot (100 - 5.08)\%)$)), representing a 10-14% improvement relative to other policies. This demonstrates that the fastest policy is highly efficient in terms of routing, achieving lower distance costs per order despite its simplicity.

Performance of value function approximation policies. Although the fastest policy achieves the lowest travel distance, Table 4 shows that the n-step SARSA and n-step hyper-heuristic policies achieve comparable routing efficiency (only 13% and 14% higher distance per order than the fastest policy, respectively), while reducing delivery lateness by 74% and 73%. Both also handle nearly twice as many orders. Compared to the simple policy, improvements are less pronounced but still significant: the n-step hyper-policy reduces traveled distance per order by 1% (none for n-step SARSA) and lateness per order by 34% (31% for the n-step SARSA). These results highlight the trade-off between routing efficiency and service quality, with value-based approaches exhibiting better delay management without excessive degratadion of distance-based metrics.

Benefit of order postponement. As discussed in Section 4, only value function-based policies allow order postponement; among them, the n-step SARSA provides a clear measure of its impact given that it does not rely on hyper-heuristics. By expanding the action space of the simple policy to include postponement, this policy learns to delay unpromising offers and prioritize future opportunities, achieving an average cost reduction of approximately 12% relative to the simple policy.

KPI	**SimplePolicy**	**HyperPolicy**	**FastestPolicy**	**NStepSarsa**	**NStepHyper**
Total cost	1001.12	898.12	1335.54	878.92	870.52
Penalty cost	625.41	529.57	1011.88	500.82	497.82
Transportation cost	375.70	368.56	323.66	378.10	372.71
Percentage of canceled orders	3.30	3.00	5.08	2.82	2.88
Percentage of orders attended on time	74.20	76.96	42.33	78.83	79.70
Average delivery time (seconds)	1684.87	1616.59	2474.22	1426.87	1421.45
Average delay (seconds)	361.17	311.34	889.01	250.12	238.89
Running time per episode (seconds)	0.20	30.63	0.14	0.49	27.83

Table 4 Key performance metrics for selected policies. Used parameters for n-step SARSA and n-step hyper-heuristic are $(n = 3000, m = 3000, \alpha = 10^{-6})$ and $(n = 1000, m = 9000, \alpha = 10^{-5})$, respectively.

Lateness costs and lost orders. Figure 5 compares system performance on a typical day and a high-cost episode under the n-step hyper-heuristic policy with parameters $(n, m, \alpha) = (1000, 9000, 10^{-5})$. The high-cost episode exhibits increased order volume and courier shortages during peak demand periods, resulting in higher lateness and more lost orders. These findings confirm that operational stress arises primarily from concurrent demand surges and limited courier capacity. Advanced policies (hyper, n-step SARSA, and n-step hyper) achieve the largest cost savings in these high-pressure scenarios through better management of delays and lost orders, while simpler policies (simple and fastest) remain competitive in stable, low-stress conditions.

Impact of courier availability. Table 5 reports the cost sensitivity of the simple policy under increased courier connection probabilities. A 10% increase in this probability yields a cost reduction that exceeds that of any policy without such an increase, indicating that courier availability is a key limiting factor. Further doubling the connection probability produces only an additional 5% reduction, revealing diminishing returns beyond the initial 10% increase.

KPI	**Base**	**+10%**	**+100%**
	(p^{conn})	$(1.1 \cdot p^{conn})$	$(2 \cdot p^{conn})$
Total cost	1001.12	556.47	511.60
Penalty cost	625.41	200.24	158.32
Transportation cost	375.70	356.23	353.28
Percentage of canceled orders	3.30	1.06	0.96
Percentage of orders attended on time	74.20	92.73	94.41
Average delivery time (seconds)	1684.87	1155.56	1104.24
Average delay (seconds)	361.17	78.19	62.44

Table 5 Sensitivity analysis of key performance metrics for the simple policy under varying courier connection probabilities. Results show average values across 50 episodes for three scenarios: base case (no change in probability), 10% increase in connection probability, and 100% increase in connection probability.

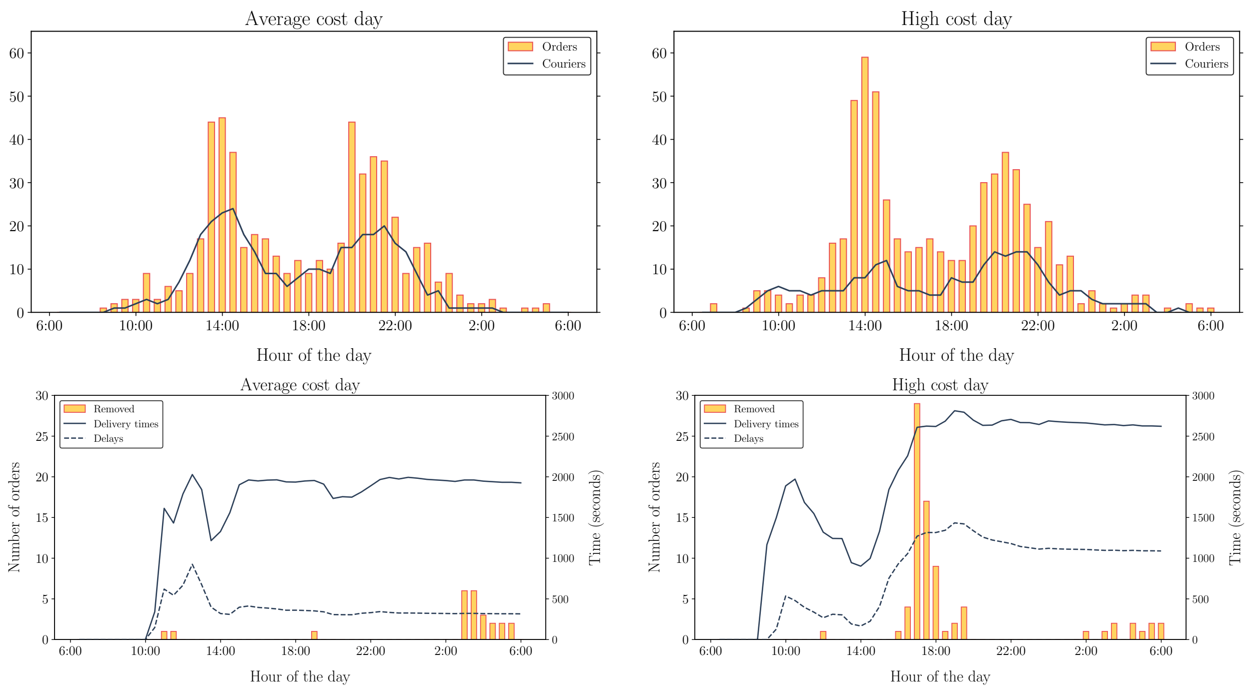

Figure 5 **Comparison between an average-cost day (left) and a high-cost day (right). Upper panels display temporal patterns of order arrivals and courier connections. Lower panels show the evolution of key performance metrics: accumulated average delivery time, average delay time, and number of lost orders (order unassigned beyond the maximum waiting threshold λ_{lost}). Results are shown for the n-step hyper-heuristic policy with parameters** $(n = 1000, m = 9000, \alpha = 10^{-5})$.

7. Conclusions

This paper addresses the First INFORMS TSL Data-Driven Research Challenge by developing a novel framework for modeling and optimizing Meituan's large-scale meal delivery operations. We first model the problem as a sequential decision process. Then using real operational data from Meituan, we design a simulator and decision-making environment that captures the dynamic, stochastic, and combinatorial nature of the problem.

We employ ML tools to build predictive models for key operational components, including travel times and order acceptance. We observe that more advanced ML methods, such as gradient boosting with trees, yield superior predictive accuracy over linear and logistic regression models; however, given the complexity of the problem, the latter are adopted in our simulations for computational efficiency. Furthermore, we propose simple sampling methods for generating order arrivals and courier connections that successfully preserve empirical correlations observed in the data.

To address the sequential and stochastic decision-making aspects of the problem, we introduce an integrated RL and hyper-heuristic framework. The n-step SARSA method enables continuous learning in a non-terminal environment, while the hyper-heuristic efficiently explores the combinatorial action space. A linear value function approximation is adopted for scalability, though future work could explore more involved approximations such as neural networks. Moreover, more low-level heuristics could be added to the pool, to check whether this results in better solutions.

Our computational study demonstrate that the proposed methods obtain achieve high-quality solutions and outperform simpler baselines. The results highlight the value of combining RL with metaheuristic search for operational decision-making under uncertainty. However, due to computational constraints experiments were conducted on scaled-down instances; extending the framework to full-scale Meituan operations (with up to 260 orders per minute) remains a promising direction for future research.

References

Auad, R., Erera, A., and Savelsbergh, M. (2023). Courier satisfaction in rapid delivery systems using dynamic operating regions. *Omega*, 121:102917.

Auad, R., Erera, A., and Savelsbergh, M. (2024a). Capacity requirements and demand management strategies in meal delivery. *EURO Journal on Transportation and Logistics*, page 100135.

Auad, R., Erera, A., and Savelsbergh, M. (2024b). Dynamic courier capacity acquisition in rapid delivery systems: A deep q-learning approach. *Transportation Science*, 58(1):67–93.

Ausseil, R., Pazour, J. A., and Ulmer, M. W. (2022). Supplier menus for dynamic matching in peer-to-peer transportation platforms. *Transportation Science*, 56(5):1304–1326.

Ausseil, R., Ulmer, M. W., and Pazour, J. A. (2024). Online acceptance probability approximation in peer-to-peer transportation. *Omega*, 123:102993.

Berbeglia, G., Cordeau, J.-F., and Laporte, G. (2010). Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15. .

Chen, X., Ulmer, M. W., and Thomas, B. W. (2022). Deep Q-learning for same-day delivery with vehicles and drones. *European Journal of Operational Research*, 298(3):939–952.

Cheng, A. (2018). Millennials are ordering more food delivery, but are they killing the kitchen, too? Retrieved from https://www.forbes.com/sites/andriacheng/2018/06/26/millennials-are-ordering-food-for-delivery-more-but-are-they-killing-the-kitchen-too/#247cfd8a393e. Accessed October 29, 2025.

Dai, H. and Liu, P. (2020). Workforce planning for O2O delivery systems with crowdsourced drivers. *Annals of Operations Research*, 291:219–245.

Drake, J. H., Kheiri, A., Özcan, E., and Burke, E. K. (2020). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer New York, New York, NY.

Hildebrandt, F. D., Thomas, B. W., and Ulmer, M. W. (2023). Opportunities for reinforcement learning in stochastic dynamic vehicle routing. *Computers & Operations Research*, 150:106071.

Hildebrandt, F. D. and Ulmer, M. W. (2022). Supervised learning for arrival time estimations in restaurant meal delivery. *Transportation Science*, 56(4):1058–1084.

INFORMS TSL (2024). The first informs tsl data-driven research challenge (2024-2025 tsl-meituan). Retrieved from https://connect.informs.org/tsl/tslresources/datachallenge. Accessed October 29, 2025.

Jahanshahi, H., Bozanta, A., Cevik, M., Kavuk, E. M., Tosun, A., Sonuc, S. B., Kosucu, B., and Başar, A. (2022). A deep reinforcement learning approach for the meal delivery problem. *Knowledge-Based Systems*, 243:108489.

Kheiri, A. (2020). Heuristic sequence selection for inventory routing problem. *Transportation Science*, 54(2):302–312.

Lagos, F. and Pereira, J. (2024). Multi-armed bandit-based hyper-heuristics for combinatorial optimization problems. *European Journal of Operational Research*, 312(1):70–91.

Liu, Y. (2019). An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones. *Computers & Operations Research*, 111:1–20. .

Mao, W., Ming, L., Rong, Y., Tang, C. S., and Zheng, H. (2019). Faster deliveries and smarter order assignments for an on-demand meal delivery platform. *Journal of Operations Management*.

Morgan Stanley Research (2017). Is online food delivery about to get "amazoned?". Retrieved from https://www.morganstanley.com/ideas/online-food-delivery-market-expands/. Accessed October 29, 2025.

Neria, G., Hildebrandt, F. D., Tzur, M., and Ulmer, M. W. (2024). The restaurant meal delivery problem with ghost kitchens. *Transportation Science*.

Pham, D. T. and Kiesmüller, G. P. (2024). Hybrid value function approximation for solving the technician routing problem with stochastic repair requests. *Transportation Science*, 58(2):499–519.

Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11. .

Psaraftis, H. N., Wen, M., and Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31.

Reyes, D., Erera, A., Savelsbergh, M., Sahasrabudhe, S., and O'Neil, R. (2018). The meal delivery routing problem. *Optimization Online*, 6571.

Shead, S. (2020). Covid has accelerated the adoption of online food delivery by 2 to 3 years, deliveroo ceo says. Retrieved from https://www.cnbc.com/2020/12/03/deliveroo-ceo-says-covid-has-accelerated-adoption-of-takeaway-apps.html. Accessed October 29, 2025.

Steever, Z., Karwan, M., and Murray, C. (2019). Dynamic courier routing for a food delivery service. *Computers & Operations Research*, 107:173–188.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

Ulmer, M. W., Erera, A., and Savelsbergh, M. (2022). Dynamic service area sizing in urban delivery. *Or Spectrum*, 44(3):763–793.

Ulmer, M. W. and Savelsbergh, M. (2020). Workforce scheduling in the era of crowdsourced delivery. *Transportation Science*, 54(4):1113–1133.

Ulmer, M. W., Thomas, B. W., Campbell, A. M., and Woyak, N. (2021). The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. *Transportation Science*, 55(1):75–100.

Union Bank of Switzerland (2018). Is the kitchen dead? Retrieved from https://www.ubs.com/global/en/investment-bank/in-focus/2018/dead-kitchen.html. Accessed October 29, 2025.

Yan, Y., Chow, A. H., Ho, C. P., Kuo, Y.-H., Wu, Q., and Ying, C. (2022). Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review*, 162:102712.

Yeo, L. (2021). Which company is winning the restaurant food delivery war? Retrieved from https://secondmeasure.com/datapoints/food-delivery-services-grubhub-uber-eats-doordash-postmates. Accessed October 29, 2025.

Yildiz, B. and Savelsbergh, M. (2019). Provably high-quality solutions for the meal delivery routing problem. *Transportation Science*, 53(5):1372–1388.

Zhao, L., Wang, H., Liang, Y., Li, D., Zhao, J., Ding, X., Hao, J., and He, R. (2024). The first informs tsl data-driven research challenge (tsl-meituan 2024): Background and data description. Retrieved from https://github.com/meituan/Meituan-INFORMS-TSL-Research-Challenge.git. Accessed October 29, 2025.

Appendix

A. Construction of $\rho_{t,c}^{\mathbf{prop}}$

To construct the proposed route ρ_t^{prop} from decision x_t and current route ρ_t, we perform the following steps for each courier $c \in \mathcal{C}_t$ with at least one order assignment ($\exists o \in \mathcal{O}_t : x_t^{c,o} \neq (0,0)$):

1. For each $o \in \mathcal{O}_t$ where we have $(k_1, k_2) = x_t^{c,o} \neq (0,0)$ set $\ell_{\text{prop}}^{k_1} = \ell_{r_o}$, $\ell_{\text{prop}}^{k_2} = \ell_o$ $o_{\text{prop}}^{k_1} = o_{\text{prop}}^{k_2} = o$, $q_{\text{prop}}^{k_1} = 1$, $q_{\text{prop}}^{k_2} = 2$.

2. Let $k' \leftarrow k \leftarrow 1$. Denote the sequence $((t^k, \ell^k, o^k, q^k))_{k \in \{1,\dots,|\rho_{t,c}|\}} = \rho_{t,c}$. While $k \leq \max(K_2(k_1, c))$:
   - If $k \notin \{\hat{k} \in x_t^{c,o} \colon \hat{k} \neq 0, o \in \mathcal{O}_t\}$ and $o \notin \{\hat{o} \colon (\hat{t}, \hat{\ell}, \hat{o}, \hat{q}) \in \rho_{t,c}, x_t^{c,\hat{o}} \neq (0,0)\}$, then set $\ell_{\text{prop}}^k \leftarrow \ell^{k'}$, $o_{\text{prop}}^k \leftarrow o^{k'}$, and $q_{\text{prop}}^k \leftarrow q^{k'}$. Set $k' \leftarrow k' + 1$ and $k \leftarrow k + 1$.
   - Else, if $k \notin \{\hat{k} \in x_t^{c,o} \colon \hat{k} \neq 0, o \in \mathcal{O}_t\}$ and $o \in \{\hat{o} \colon (\hat{t}, \hat{\ell}, \hat{o}, \hat{q}) \in \rho_{t,c}, x_t^{c,\hat{o}} \neq (0,0)\}$, then set $k' \leftarrow k' + 1$.
   - Else, if $k \in \{\hat{k} \in x_t^{c,o} \colon \hat{k} \neq 0, o \in \mathcal{O}_t\}$, then set $k \leftarrow k + 1$.

3. If $|\rho_{t,c}| = 0$, then set $t_{\text{prop}}^1 \leftarrow t + \Delta + D(\tilde{\ell}_{t,c}, \ell_{r_o})/v$ and $t_{\text{prop}}^2 \leftarrow t_{\text{prop}}^1 + \hat{\mu}_p + D(\ell_{r_o}, \ell_o)/v$, where v is the average speed of the drivers in the city.

4. Otherwise, if $|\rho_{t,c}| > 0$, let $t_{\text{prop}}^1 = t^1$, where $\{(t^k, \ell^k, o^k, q^k)\}_{k \in \{1,\dots,|\rho_{t,c}|\}} = \rho_{t,c}$. Then for $k \in \{2, \dots, \max(K_2(k_1, c))\}$ do
$$t_{\text{prop}}^k \leftarrow \begin{cases} \max\{b_{o_{\text{prop}}^k}, t_{\text{prop}}^{k-1} + \mu(q_{\text{prop}}^{k-1}) + D(\ell_{\text{prop}}^{k-1}, \ell_{\text{prop}}^k)/v\}, & \text{if } q_{\text{prop}}^k = 1, \\ t_{\text{prop}}^{k-1} + \mu(q_{\text{prop}}^{k-1}) + D(\ell_{\text{prop}}^{k-1}, \ell_{\text{prop}}^k)/v, & \text{if } q_{\text{prop}}^k = 2, \end{cases}$$
where $\mu(1) = \hat{\mu}_p$ and $\mu(2) = \hat{\mu}_d$ are estimated pickup and delivery service times, respectively.

5. $\rho_t^{\text{prop}} \leftarrow \{(t_{\text{prop}}^k, \ell_{\text{prop}}^k, o_{\text{prop}}^k, q_{\text{prop}}^k)\}_{k \in \{1,\dots,\max(K_2(k_1,c))\}}$.

B. Meta-parameters set-up for the hyper- and n-step hyper-policy

Parameter	Description	Value
t_{limit}	Time limit (in seconds) used for each call to the procedure	30
$P^0_{h,g}$, $L^0_{h,1}$, $L^0_{h,2}$	Initial values of the transition matrices	0
η	Learning rate used to update the transition matrix probabilities	0.25
ε	Threshold parameter used to accept a candidate solution as new temporal solution	10^{-5}
ϑ	Temperature parameter used to accept a candidate solution as new temporal solution	0.01
k^{remove}	Number of offers removed from current solution x_t in low-level heuristics h_1, h_2, h_3, h_6, h_7	$\min\{\lvert\mathcal{O}^{assigned}(x)\rvert, \max\{\lceil 12\%\lvert\mathcal{C}_t\rvert\rceil, 24\}\}$
k^{restruct}	Number of couriers selected to improve their current route in the low-level h_4	$\max\{10\%\lvert\mathcal{C}_t\rvert, 5\}$
$k^{\text{interchange}}$	Number of pairs of couriers selected to swap order offers in low-level h_5	$\min\{\lvert\mathcal{C}_t\rvert/2, \max\{10\%\lvert\mathcal{O}^{assigned}(x)\rvert, 5\}\}$
$N^{\text{o}}_{\max}$	Maximum number of concurrent orders a courier can handle	4
N_{days}	Number of days used to normalize sampling probabilities	7.96
v	Average courier travel speed (km/h)	18
$\hat{\mu}_p, \hat{\mu}_d$	Estimated pickup and delivery service times for policy computation (minutes)	3.5
γ	Discount factor in value function	0.999
K_{dist}	Penalty weight for travel distance	1
K_{late}	Penalty weight for late delivery	4
K_{lost}	Penalty weight for lost orders	16
λ_{lost}	Maximum tolerated lateness before order is lost (hours)	2
Δ	Decision epoch interval (seconds)	30
b	Batch size used in SARSA training	20
n_{update}	Update frequency of parameter vector in SARSA	200
θ_0	Initial parameter vector for the value-function approximation	0
$t_{\max}$	Number of decision epochs used for training (per run)	10^5
m_{feat}	Number of time intervals into which the future horizon is divided to quantify the urgency distribution of pending (unoffered) orders in VFA	8
n_{feat}	Number of features used to approximate the value function through a weighted linear combination	14
δ^{dist}	Distance to the restaurant threshold to bound the eligible couriers to offer a corresponding order (km)	10

Table 6 **Meta-parameters set-up. We define $\mathcal{O}^{assigned}(x_t) := \{(o,c)\colon o \in \mathcal{O}^{\textsf{open}}_t, c \in \mathcal{C}_t, x^{c,o}_t \neq (0,0)\}$.**

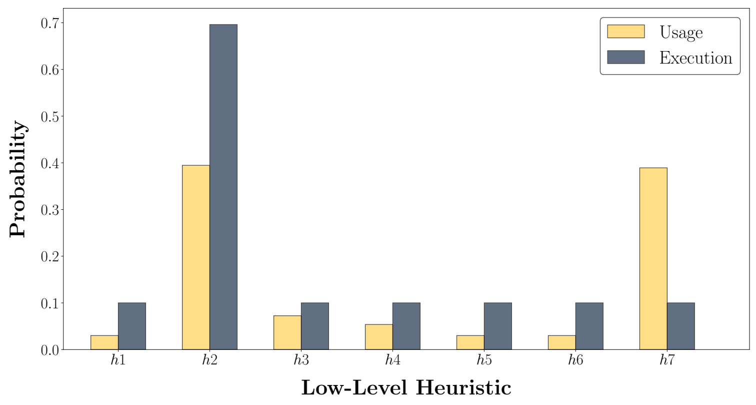

Figure 6 **Usage probability $\sum_{g\in\mathcal{H}} T_{g,h}$ and termination probability $S_{h,2}$ for each low-level heuristic $h\in\mathcal{H}$.**

C. Validation of low-level heuristics

We evaluate whether the seven low-level heuristics in the pool are effectively and actively utilized by the framework. For each heuristic $h\in\mathcal{H}$, we compute two measures: (i) its overall usage probability, $\sum_{g\in\mathcal{H}} T_{g,h}$, which indicates how often h is selected following another heuristic; and (ii) the probability that a heuristic sequence terminates with h, denoted $S_{h,2}$. Figure 6 reports both measures for the n-step hyper-policy.

Results show that all heuristics exhibit strictly positive usage and termination probabilities of comparable magnitudes. Notably, heuristic h_2, which removes geographically close allocated orders and reinserts them after local route improvements, is among the most frequently applied and has a high probability of terminating heuristic sequences. This indicates that h_2 effectively exploits spatial proximity to generate localized route improvements.

Heuristic h_7, which removes orders with the latest delivery deadlines, improves routes, and reinserts them in ascending deadline order, also displays a high usage probability but a lower likelihood of terminating sequences. Intuitively, h_7 often produces partially improved allocations that benefit from subsequent refinement by other heuristics.

Overall, these findings demonstrate that the framework leverages all seven heuristics, each fulfilling distinct and complementary roles. The observed balance between their usage and termination probabilities suggests that the heuristic pool is diverse and non-redundant, with its elements jointly contributing to the exploration and improvement of candidate solutions.