Les Cahiers du GERAD ISSN: 0711–2440

Learning-to-optimize for consolidation and transshipment in multi-store order delivery

X. Wang, O. Arslan, J.-F. Cordeau, E. Delage

G–2025–40

June 2025

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Learning-to-optimize for consolidation and transshipment in multi-store order delivery

Xin Wang

Okan Arslan

Jean-François Cordeau

Erick Delage

Department of Decision Sciences & GERAD, HEC Montréal, Montréal (Qc), Canada, H3T 2A7

xin.wang@hec.ca
okan.arslan@hec.ca
jean-francois.cordeau@hec.ca
erick.delage@hec.ca

June 2025
Les Cahiers du GERAD
G–2025–40

Abstract : This study investigates multi-store order delivery services where customers can order from multiple stores for home delivery. We first consider separated-order delivery, where orders from different stores are processed and delivered individually. To improve customer convenience and operational efficiency, we introduce consolidated-order delivery, enabling customers to place a single order across stores and receive all items in one combined delivery. While this enhances convenience, it can increase delivery times due to additional routing for visiting multiple stores. To mitigate this shortcoming, we propose a consolidated-order delivery system with transshipment, allowing drivers to transfer orders at transshipment nodes for higher efficiency. We develop a mixed-integer linear program for the multi-store order problem that models different delivery systems, including separated-order delivery and consolidated-order delivery with or without transshipment. Due to computational challenges arising from routing decisions and time variables, we adopt a learning-to-optimize approach that integrates machine learning and optimization. Four methods are implemented for learning driver allocation decisions: Nearest Driver Allocation, Driver Assignment Neural Network (DANN), Driver Classification Neural Network (DCNN), and Graph-based Neural Network (GNN). Our experimental study reveals that GNN consistently performs the best in terms of accuracy, optimality gap, efficiency, and scalability to larger problem instances beyond the training set. The DCNN and DANN are effective with sufficiently large training sets and perform well when the instance scale in the testing set aligns with the training set. We conduct experiments across four U.S. regions using the learning-to-optimize method in a realistic setting with dynamic customer arrivals. We find that consolidated-order delivery with transshipment, coupled with a short-duration waiting strategy, consistently delivers superior performance, yielding shorter order completion times and reduced driver travel times through effective spatial and temporal consolidation. Waiting longer to batch more customers is advantageous under conditions of frequent customer arrivals and limited driver availability.

Keywords : Last-mile delivery, consolidation and transshipment, pickup and delivery, learning-to-optimize, neural network

1 Introduction

Online meal and grocery delivery services emerged in the early 2000s and experienced substantial growth soon thereafter. This expansion accelerated between 2010 and 2020, with a dramatic surge in 2020 due to the global pandemic. By 2024, the online delivery market has reached a volume of USD 1.22 trillion, with projections indicating further growth to USD 1.79 trillion by 2028 (Statista 2024).

To access a wider variety of products or enjoy the convenience of home delivery, customers increasingly rely on online delivery services for items from multiple types of stores, such as meals from restaurants, fresh produce from grocery stores, or emergency medicine from pharmacies. Companies such as UberEats and DoorDash provide a *multi-store order service* that caters to customers who prefer the convenience of home delivery over visiting multiple stores in person, especially when they are pressed for time. These platforms conventionally adopt *separated-order delivery* (SOD), a system where orders from different stores are treated as distinct transactions and delivered individually. Customers place separate orders from each store, incurring separate delivery fees for each, while drivers handle pickups and deliveries for each order independently. Although this model provides convenient access to products from multiple stores, it can be inefficient and costly due to multiple delivery fees and minimum order requirements for each store. Recently, companies such as DoubleDash (DoorDash), Instacart, and Epipresto have started offering what we term *consolidated-order delivery* (COD) services. These services allow customers to order from multiple stores in a single transaction, receiving all items as a combined delivery, typically without an added delivery fee. For instance, DoubleDash (2023) and Instacart (2022) allow customers to add items from nearby retailers to their original order without an additional delivery charge, ensuring that all items are delivered together by the same driver. Similarly, Epipresto (2023) enables customers to shop from multiple stores at in-store prices with a fixed delivery fee, regardless of distance. This consolidated-order model offers multiple benefits: customers save on delivery fees, enjoy the convenience of placing one order for all needed products, and have a single delivery experience. Meanwhile, companies may benefit from a decreased need for drivers, reduced delivery costs, and increased sales. Despite these benefits, consolidated-order delivery can introduce new challenges to delivery efficiency. A single driver responsible for picking up and delivering items from multiple stores may face extended delivery times, particularly when fulfilling orders from far apart store locations to multiple customers. To overcome these limitations and optimize the delivery process, we propose a model that divides multi-store orders among several drivers, who can then transfer orders through transshipment. In the *consolidated-order delivery with transshipment* (CODT) system, a customer's multi-store order is distributed among multiple drivers, each responsible for pickups from different stores. These orders are then transferred at selected transshipment nodes, potentially assigned to a different driver, and delivered as one package to the customer's doorstep. This collaborative approach offers the combined benefits of consolidation with enhanced efficiency, reducing overall delivery time and improving route management for drivers handling multi-store, multi-customer orders.

To quantify the value of enabling order consolidation and transshipment for fulfilling multi-store orders in a combined delivery, we develop a mixed integer linear program (MILP) that models three delivery systems: separated-order delivery, consolidated-order delivery without transshipment, and consolidated-order delivery with transshipment. The MILP for multi-store order services presents substantial computational challenges. From the drivers' perspective, this problem resembles a complex vehicle routing problem involving multiple store and customer locations. Conversely, the journey of each item can be viewed as a shortest path problem within a network defined by the drivers' delivery routes, starting at the store and ending at the customer. This requires precise tracking of each item and driver's arrival and departure times to meet time windows and ensure timely deliveries. To mitigate the significant solution times for solving the MILP directly, we adopt a learning-to-optimize approach. This method replaces the traditional optimization process with a machine learning-based approximation (i.e., an optimization proxy). The learning model estimates allocation decisions, identifying which drivers should pick up items from stores and which should deliver them to customers. These estimated

allocations are subsequently refined through an MILP with a reduced search space, significantly enhancing computational efficiency and enabling practical implementation. We evaluate three delivery systems in a simulated dynamic setting, where both customer arrivals and the optimization process occur in real-time, making time and space consolidation crucial. The system can optimize deliveries immediately as each customer arrives or wait to batch multiple customers for joint optimization. This analysis helps identify the most efficient delivery system under various waiting strategies.

The four main contributions of this paper are as follows:

- We investigate three delivery systems for multi-store order services, where customers place orders across multiple stores for home delivery. Beyond separated-order delivery, in which each order from each store is handled separately, we propose consolidated-order delivery, with or without transshipment. Consolidated-order delivery enables customers to order from multiple stores in a single transaction and receive all items as a combined delivery, while transshipment further allows for driver coordination and order transshipment to enhance delivery efficiency.
- We formulate a mixed-integer linear program to model the multi-store order problem (MSOP) across all three delivery systems. To address the computational demanding problem of optimizing both routing and transfer decisions, we adopt a learning-to-optimize approach that integrates machine learning with optimization. In this framework, the estimated allocation decisions act as a lower bound within the optimization process. This process not only refines the allocation decisions but also optimizes additional decisions that are not directly learned. By offloading a portion of the computational workload to the offline phase, this approach enables the near real-time generation of high-quality solutions.
- We implement four methods to obtain the estimated allocation plans efficiently. Nearest Driver Allocation (NDA) identifies the closest driver for each customer. Driver Assignment Neural Network (DANN) uses binary predictions to assign customers to drivers, treating each driver independently. Driver Classification Neural Network (DCNN) frames the allocation of customers to drivers as a classification problem, with each driver representing a unique class. Finally, Graph-based Neural Network (GNN) models each instance as a graph and treats customer-to-driver allocation as edge labels to be learned.
- We conduct numerical experiments across various U.S. regions with varying customer arrival rates, comparing the performance of our three delivery systems and four learning methods. The following insights are derived:
  - By applying a learning-to-optimize approach, we enhance the solution efficiency of the multi-store order problem and obtain high-quality solutions. This approach involves learning allocation decisions for assigning drivers to visit nodes based on historically optimal samples, which narrows the feasible solution space and facilitates effective decision-making within this refined search area.
  - The GNN outperforms other methods, achieving a better balance between optimality gap and solution time while also scaling well to larger instances beyond the training set, due to its ability to facilitate information exchange. However, it requires significant effort to collect sufficient historical samples, as each instance corresponds to a single graph. The NDA is simple to implement, requiring no training or historical samples, but struggles with clustered customer locations. Both the DCNN and DANN perform well when the test instance scale matches the training data but face challenges adapting to larger-scale instances.
  - We identify a balance between batching efficiency and individual order fulfillment. As more orders are batched for joint optimization, delivery times rise due to longer routes, but wait times initially drop significantly because batching makes better use of drivers. However, beyond a certain point, extending the re-optimization interval leads to increased wait times as customers experience longer delays before being served. Consequently, the total order completion time, which includes both wait and delivery times, initially decreases, reaching a minimum before increasing again. This indicates the existence of an optimal waiting

strategy, influenced by factors such as customer arrival rates and the ratio of drivers to customers.

– Consolidated-order delivery shows increasing efficiency over separated-order delivery as customer scale grows, since consolidating orders from multiple stores effectively reduces both delivery and driver travel times. The COD with transshipment and a short-duration waiting strategy consistently achieves the best performance in delivery and travel times, regardless of customer or driver scales.

The rest of the paper is organized as follows. We review related work in Section 2. Next, we define the problem setting, describe our three delivery systems, and present the models for these systems in Section 3. We develop the learning-to-optimize method in Section 4 and report the results of numerical studies comparing the performance of our proposed models and their solution quality in Section 5. Finally, we conclude with managerial insights in Section 6.

2 Literature review

In this section, we review the main studies relevant to our research from three points of view: last-mile delivery, pickup and delivery problem, and learning-based optimization.

2.1 Last-mile delivery

Last-mile delivery, referring to delivery from distribution points or transshipment nodes to customer locations, is critical in online food and grocery delivery. Savelsbergh and Van Woensel (2016) review various current and anticipated challenges and opportunities in last-mile logistics, highlight the growing customer demands for price, quality, time, and sustainability, and emphasize the importance of batching in city logistics for improvement in the cost-efficiency while acknowledging the risk of delayed fulfillment.

In the field of on-time last-mile delivery for food and groceries, most research focuses on improving service quality through faster delivery or enhancing efficiency by reducing costs. This is achieved through demonstrating the importance of on-demand delivery using empirical evidence, optimizing the assignment of drivers to customers or driver routing for dispatching items, or proposing novel business models and applying learning techniques to improve the overall efficiency of the delivery system. From an empirical perspective, Mao et al. (2022) and Li and Wang (2024), demonstrate the benefits of on-demand platforms for delivery performance and restaurant profitability. From an optimization perspective, Liu et al. (2021) address order assignment problem with travel-time predictors to investigate the impact of delivery data on the on-time performance of food delivery services. Zhang et al. (2023) develop scalable algorithms for optimizing routing operations and vehicle-customer coordination in a dynamic setting to improve profitability, service, and environmental impact. Moreover, Carlsson et al. (2024) propose region partitioning policies to minimize delivery times in a stochastic and dynamic setting by assigning drivers to subregions. In terms of novel business models, Cao and Qi (2023) introduce self-driving mini grocery stores to enhance mobility, proximity, and flexibility of grocery delivery by avoiding the last 100 meters. Additionally, Raghavan and Zhang (2024) explore coordinated logistics by evaluating the value of using aides who can assist drivers in last-mile delivery to reduce service time and enable parallel deliveries. From the learning perspective, Hildebrandt and Ulmer (2022) improve delivery systems by predicting accurate arrival times, enhancing service perception and efficiency. Similarly, Auad et al. (2024) propose a deep reinforcement learning approach for balancing courier capacity with service quality in meal delivery systems.

We aim to provide efficient on-demand, on-time delivery for food and groceries, ensuring fast delivery for customers while minimizing driver travel times. Our proposed setting allows customers to order from multiple stores and receive all items in one delivery for more convenience. Drivers can coordinate at intermediate points to transfer orders, with any driver able to complete the delivery.

However, this setup creates a complex pickup and delivery problem, which we will address using mathematical programming formulations and efficient solution techniques.

2.2 Pickup and Delivery Problem

The Pickup and Delivery Problem (PDP) is a classical combinatorial optimization problem concerned with optimizing vehicle routes for picking up and delivering goods or passengers at various locations. Its primary goal is the efficient movement of vehicles to minimize transportation costs. The PDP is an extension of the Vehicle Routing Problem (VRP), first introduced by Dantzig and Ramser (1959). There are several variants of the PDP, including single-vehicle or multiple-vehicle PDP, dynamic PDP, many-origin-to-many-destination PDP, PDP with time windows, PDP with split delivery, and PDP with transshipment, among others. Berbeglia et al. (2010) provide a general framework for dynamic PDPs, where objects or people need to be collected and delivered in real-time, with requests revealed over time. They also explain how to design waiting strategies and assess their performance. Additionally, Koç et al. (2020) conduct a detailed survey of vehicle routing problems with simultaneous pickup and delivery, where goods must be transported from various origins to different destinations, satisfying both delivery and pickup demands concurrently.

In the context of the many-to-many PDP with transshipment that we are interested in, the pickup loads and delivery processes can be split among drivers, with items exchanged at transshipment points. Specifically, one vehicle picks up the load, delivers it to a transshipment node, and another vehicle completes the delivery. In terms of empirical studies, Mitrović-Minić and Laporte (2006) evaluate the usefulness of transshipment points in PDP with time windows. Nowak et al. (2009) show that splitting loads into multiple trips benefits the PDP and identify favorable real-world environments for this approach through empirical analysis. In terms of optimization, Rieck et al. (2014) address the many-to-many location-routing problem with inter-hub transport and multi-commodity PDP, developing heuristics for medium- and large-scale instances. While this problem is similar to our study, it does not account for the importance of time windows, which further complicates the setting. Rais et al. (2014) and Lyu and Yu (2023) investigate PDP with time windows and transshipments, but they only solve small and medium instances and overlook the consolidated-delivery of orders from multiple pickups and the potential for any visited node to act as a transshipment point under time constraints. Lastly, Su et al. (2023) study PDP with crowdsourced bids and transshipment in last-mile delivery, which optimizes vehicle routes and bid selection but not crowdshipper routing and imposes restrictive service assumptions.

We investigate a many-origin-to-many-destination pickup and delivery problem with time windows, incorporating driver coordination and transshipment to fulfill orders from multiple stores. Our model will account for order allocation to drivers, selection of transshipment nodes for item exchanges, routing decisions for drivers and orders, and time variables for driver and order arrivals and departures.

2.3 Machine Learning-based optimization

Machine learning (ML) has emerged as a transformative tool in operations research (OR) and operations management, enabling innovative approaches to solve complex decision-making problems. The integration of ML in OR can broadly be categorized into two areas: using ML to design and enhance optimization models, and employing optimization proxies to accelerate or approximate solutions.

In the first category, ML is widely used to design optimization models, improve data quality, and enhance solution procedures. Sadana et al. (2024) review contextual stochastic optimization and unify decision-making methods under uncertainty, categorizing approaches into decision rule optimization, sequential learning and optimization, and integrated learning and optimization. Maragno et al. (2023) propose a pipeline where constraints and objectives are learned from data and embedded into optimization models. Similarly, Wang et al. (2024) enhance inputs to vehicle routing problems using techniques like k-nearest neighbor and kernel density estimation. In practical applications, Liu et al.

(2021) integrate ML-based travel-time predictors into order-assignment optimization for food delivery, improving on-time performance by refining parameter estimates. Additionally, Hong et al. (2021) propose a robust optimization framework using ML to enhance parameter estimates and optimization outcomes.

In the second category, optimization proxies use ML to approximate solutions, accelerate computational processes, or enable inverse optimization. Van Hentenryck (2021) distinguish between two main approaches: end-to-end learning, which uses proxies to directly approximate optimal solutions, and learning-to-optimize, which accelerates existing optimization algorithms for enhanced efficiency. Kotary et al. (2021) review hybrid methods that integrate combinatorial solvers with ML for fast, approximate solutions and logical inference. Several studies highlight how learning-to-optimize speeds up solution generation. For example, Ojha et al. (2023) develop ML proxies to predict loads and ensure constraint compliance in dynamic load planning, significantly improving decision-making speed. Similarly, Julien et al. (2024) introduce a machine-learning-based node selection strategy that accelerates robust optimization by identifying high-quality solutions faster. Larsen et al. (2024) use supervised learning to approximate computationally intensive second-stage solutions in two-stage stochastic programming, reducing solution time without compromising accuracy. Kotary et al. (2024) propose a learning-to-optimize-from-features framework, aligning traditional methods with the predict-then-optimize paradigm. Additionally, Qi et al. (2023) demonstrate the impact of end-to-end learning by developing a multi-period inventory replenishment model that directly outputs optimal replenishment decisions from input features. Özarık et al. (2024) apply ML to estimate key parameters for last-mile delivery route optimization in inverse optimization, illustrating how predictive modeling can inform complex decision-making.

In this work, we apply learning-to-optimize to address our computationally challenging problem. We first use machine learning techniques, including neural networks and graph-based networks, to estimate allocation decisions. These estimates serve as inputs for an optimization process, which refines the allocation decisions and makes additional decisions, such as optimal routing. Unlike conventional methods, our approach treats estimated decisions as lower bounds, maintaining flexibility and allowing the optimization process to explore better solutions. Our work not only aims for efficiency and high-quality solutions but also evaluates the scalability of different learning algorithms across various cases.

3 The multi-store order delivery problem

In this section, we first introduce our three delivery systems of interest and present the problem definition in Section 3.1. We then formulate the mathematical model for the multi-store order delivery problem under various delivery systems in Section 3.2, followed by the dynamic problem description incorporating waiting strategies in Section 3.3.

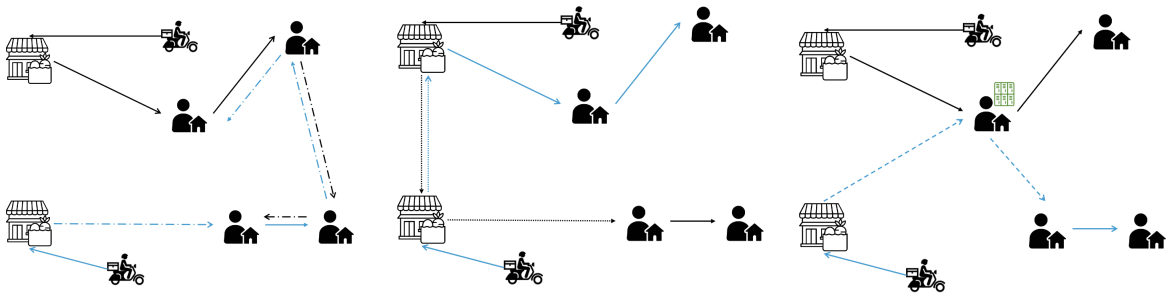
3.1 Delivery system description and problem definition

Due to the variety of store types, product availability, and special pricing, customers often purchase products from multiple stores, such as meals from restaurants, fresh produce from grocery stores, or emergency medicine from pharmacies. To avoid the inconvenience of visiting multiple stores, customers increasingly rely on online ordering and delivery platforms, creating the necessity for multi-store order delivery services. We consider three delivery systems for managing multi-store order deliveries:

1. *Separated-order delivery (SOD):* Customers place separate orders from different stores, pay a delivery fee for each order, and receive individual deliveries. Orders from each store are handled independently, with drivers managing both the pickup and delivery processes (Figure 1a). This system serves as a baseline for comparison in the absence of consolidation.
2. *Consolidated-order delivery (COD):* Customers place a combined order from multiple stores in a single transaction and receive a combined delivery, typically without additional delivery fees.

A single driver collects items from several stores and delivers them to the customer (Figure 1b). This system consolidates orders from different stores and serves as a baseline in the absence of transshipment.

3. *Consolidated-order delivery with transshipment (CODT):* Similar to COD, customers place a consolidated order from multiple stores and receive a combined delivery. However, in this system, different drivers can pick up items from different stores and can transfer them at selected transshipment nodes. A single driver then completes the final delivery to the customer's doorstep (Figure 1c). Transshipment can occur at various locations, such as stores or customer locations, where drivers can drop off or pick up items and items can be temporarily held without the presence of drivers. In some cases, customers acting as transshipment nodes may receive multiple deliveries, while others receive a single delivery.



(a) Separated-order delivery. (b) Consolidated-order delivery. (c) COD with transshipment.

Figure 1: Delivery systems for multi-store order services. This figure illustrates an instance with two drivers, two stores, and four customers. The same set of customers is served by the same set of drivers across all three systems. The route of one driver is represented by a black line, while the route of another driver is shown in blue. The solid lines are the same for each system, while the dashed lines highlight the differences.

We define the problem that accommodates these three delivery systems as follows:

Definition 1. The *Multi-Store Order Problem (MSOP)* is an optimization problem addressing the fulfillment of customer orders placed across multiple stores using a limited number of drivers. The primary objective is to minimize the latest delivery time for serving each order, and the secondary objective is to minimize the total driver travel time for fulfilling all requests. This is achieved by addressing key decisions, including allocating drivers for pickup and delivery, planning driver routes with time and capacity restrictions, and selecting transshipment nodes for transferring items among drivers. In addition to handling orders from different stores separately, MSOP also incorporates consolidation, allowing orders from multiple stores to be grouped and handled together, and transshipment, permitting partial order transfers among drivers at specified nodes.

3.2 Mathematical model

We formulate the multi-store order problem for all three delivery systems using a mixed-integer linear programming model. This MILP is defined on an undirected network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ represents the set of nodes and $\mathcal{A}$ denotes the set of arcs. Customers are denoted by $i \in \mathcal{I}$, stores by $j \in \mathcal{J}$, and drivers by $k \in \mathcal{K}$. The origin node of driver k, representing their starting location, is denoted as o_k, and the set $\mathcal{O}$ comprises the origin nodes of all drivers. The node set $\mathcal{N}$ includes all customer, store, and driver starting locations, i.e., $\mathcal{N} = \mathcal{I} \cup \mathcal{J} \cup \mathcal{O}$. The binary parameter e_{ij} indicates whether customer i orders from store j, and the item ordered by customer i from store j is indexed by ij. The size of item ij is p_{ij}, and the capacity of driver k is q^k. The travel time between nodes n and n' is denoted by $t_{nn'}$, and it satisfies the triangle inequality: $t_{nn'} \leq t_{nn''} + t_{n''n'}, \forall n, n', n''$. The time window for customer i is given by $[\alpha_i, \beta_i]$. The decision variables are as follows. Let z_n^k be 1 if and only if driver k visits node n, $x_{nn'}^k$ be 1 if and only if driver k travels from node n to node n', $y_{nn'}^{ij}$ be 1 if and only if item ij travels from node n to node n' during the trip, v_n^{kij+} be 1 if and only if item ij arrives at node n via driver k, and v_n^{kij-} be 1 if and only if item ij departs from node n via driver k.

The continuous variables are as follows: τ_n^{k+} is the time at which driver k arrives at node n, τ_n^{k-} is the time at which driver k departs from node n, and λ_n^{ij} is the arrival time of item ij at node n. A summary of the notation is provided in Appendix A.

The model for the multi-store order problem is presented as follows:

$$\mathrm{M}(\mathcal{I},\mathcal{J},\mathcal{K}) = \min_{\substack{x,y,z,\\ \tau,v,\lambda}} \quad \rho_1 \max_{\substack{k\in\mathcal{K}\\ i\in\mathcal{I}}} \left\{\tau_i^{k+}\right\} + \rho_2 \sum_{k\in\mathcal{K}}\sum_{n\in\mathcal{N}}\sum_{\substack{n'\in\mathcal{N},\\ n'\neq n}} t_{nn'}x_{nn'}^k \tag{1a}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} v_j^{kij-} \geq e_{ij}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J} \tag{1b}$$

$$\sum_{k\in\mathcal{K}} v_i^{kij+} \geq e_{ij}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J} \tag{1c}$$

$$\sum_{k\in\mathcal{K}} v_n^{kij+} = \sum_{k\in\mathcal{K}} v_n^{kij-}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}: n\neq i,j \tag{1d}$$

$$z_{o_k}^k \geq z_n^k \geq \max\{v_n^{kij+}, v_n^{kij-}\}, \qquad \forall k\in\mathcal{K}, i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N} \tag{1e}$$

$$\sum_{n'\in\mathcal{N},n'\neq n} x_{n'n}^k = z_n^k, \qquad \forall n\in\mathcal{N}, k\in\mathcal{K} \tag{1f}$$

$$\sum_{n'\in\mathcal{N},n'\neq n} x_{nn'}^k = z_n^k, \qquad \forall n\in\mathcal{N}, k\in\mathcal{K} \tag{1g}$$

$$\tau_{n'}^{k+} \geq x_{nn'}^k(\tau_n^{k-} + t_{nn'}), \qquad \forall n,n'\in\mathcal{N}: n\neq n', k\in\mathcal{K} \tag{1h}$$

$$\tau_n^{k-} \geq \tau_n^{k+}, \qquad \forall n\in\mathcal{N}\setminus\mathcal{O}, k\in\mathcal{K} \tag{1i}$$

$$\sum_{n'\in\mathcal{N},n'\neq j} y_{jn'}^{ij} = e_{ij}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J} \tag{1j}$$

$$\sum_{n'\in\mathcal{N},n'\neq j} y_{n'i}^{ij} = e_{ij}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J} \tag{1k}$$

$$\sum_{n'\in\mathcal{N},n'\neq n} y_{nn'}^{ij} = \sum_{n'\in\mathcal{N},n'\neq n} y_{n'n}^{ij}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}, n\neq i,j \tag{1l}$$

$$y_{nn'}^{ij} \leq \sum_{k\in\mathcal{K}} x_{nn'}^k, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n,n'\in\mathcal{N}: n'\neq n \tag{1m}$$

$$y_{no_k}^{ij} \leq 0, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}, k\in\mathcal{K} \tag{1n}$$

$$\lambda_{n'}^{ij} \geq y_{nn'}^{ij}(\lambda_n^{ij} + t_{nn'}), \qquad \forall k\in\mathcal{K}, n\in\mathcal{N}: n\neq n', n'\in\mathcal{N}\cup\{k'\} \tag{1o}$$

$$v_n^{kij+} \geq \min\left\{x_{nn'}^k, y_{nn'}^{ij}\right\}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n,n'\in\mathcal{N}: n\neq n', k\in\mathcal{K} \tag{1p}$$

$$v_{n'}^{kij-} \geq \min\left\{x_{nn'}^k, y_{nn'}^{ij}\right\}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n,n'\in\mathcal{N}: n\neq n', k\in\mathcal{K} \tag{1q}$$

$$\lambda_n^{ij} \geq v_n^{kij+}\tau_n^{k+}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}, k\in\mathcal{K} \tag{1r}$$

$$\tau_n^{k-} \geq v_n^{kij-}\lambda_n^{ij}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}, k\in\mathcal{K} \tag{1s}$$

$$\alpha_i \leq \lambda_i^{ij} \leq \beta_i, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J} \tag{1t}$$

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} p_{ij}v_n^{kij-} \leq q^k, \qquad \forall n\in\mathcal{N}, k\in\mathcal{K} \tag{1u}$$

$$x_{nn'}^k, y_{nn'}^{ij}, z_n^k, v_n^{kij+}, v_n^{kij-} \in \{0,1\}, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n,n'\in\mathcal{N}, k\in\mathcal{K} \tag{1v}$$

$$\tau_n^{k+}, \tau_n^{k-}, \lambda_n^{ij} \geq 0, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}, k\in\mathcal{K}. \tag{1w}$$

The objective function (1a) aims to minimize the weighted average of the latest delivery time to the last customer and the total travel time for fulfilling all orders, where ρ_1 is the weight assigned to the latest delivery time and ρ_2 is the weight assigned to the total travel time. Note that to prioritize on-time delivery, ρ_1 should be set significantly larger than ρ_2, while a relatively small value for ρ_2 is

sufficient to discourage unnecessary long travel times. The term $\max_{k\in\mathcal{K},i\in\mathcal{I}}\{\tau_i^{k+}\}$ can be equivalently replaced by a new variable l, along with the constraints $l \geq \tau_i^{k+}, \forall i \in \mathcal{I}, k \in \mathcal{K}$.

Constraints (1b) and (1c) stipulate that if customer i orders items from store j (i.e., $e_{ij} = 1$), item ij should be picked up from store j by a driver and delivered to customer i by a driver. Constraints (1d) ensure that if item ij arrives at node n (except for destination i and origin j), it must leave this node after being picked up by a driver. Constraints (1e) indicate that if driver k serves item ij by passing through node n, then driver k must leave its origin location o_k and also visit node n. Constraints (1f) and (1g) impose degree constraints on the nodes visited by driver k. Constraints (1h) specify the arrival and departure times of driver k at node n. If driver k travels from node n to node n' (i.e., $x_{nn'}^k = 1$), the arrival time of driver k at node n' must be no earlier than the departure time from the previous node n, accounting for the travel time from node n to node n'. These constraints can be linearized into $\tau_n^{k-} + t_{nn'} - \tau_{n'}^{k+} \leq M_1(1 - x_{nn'}^k)$, where M_1 should be sufficiently large to ensure that $M_1 \geq \tau_n^{k-} + t_{nn'}$. Additionally, constraints (1i) ensure that the departure time at any node of a driver should be no earlier than their arrival time, except for their origin nodes. Constraints (1j) to (1l) denote that item ij should leave store j and arrive at customer i, and that if item ij arrives at any other node, it must also leave this node (i.e., flow balance constraints). Constraints (1m) and (1n) state that item ij can traverse the arc (n, n') only if this arc is visited by a driver, and that arcs for drivers returning to origin nodes cannot be part of the path for items. Constraints (1o) specify the arrival time of item ij at node n. If item ij travels from node n to node n' (i.e., $y_{nn'}^{ij} = 1$), the arrival time of item ij at node n' must be no earlier than its arrival time at the previous node n, while accounting for the travel time from node n to node n'. These constraints can be linearized into $\lambda_n^{ij} + t_{nn'} - \lambda_{n'}^{ij} \leq M_2(1 - y_{nn'}^{ij})$, where M_2 must be large enough to ensure that $M_2 \geq \lambda_n^{ij} + t_{nn'}$. Constraints (1p) and (1q) indicate that if item ij passes the arc (n, n') visited by driver k, then item ij must be picked up by driver k to leave node n and dropped off by driver k at node n'. Since x, y, and v are binary, these constraints can be linearized into $v_n^{kij-} \geq x_{nn'}^k + y_{nn'}^{ij} - 1$ and $v_{n'}^{kij+} \geq x_{nn'}^k + y_{nn'}^{ij} - 1$, respectively. Constraints (1r) and (1s) state that the arrival time of item ij at node n should be no earlier than the arrival time of driver k at node n if this item is served by driver k to arrive at node n. Furthermore, the departure time of driver k should be no earlier than the arrival time of item ij at node n if this item is about to leave node n via driver k. These constraints can be linearized into $\tau_n^{k+} - \lambda_n^{ij} \leq M_3(1 - v_n^{kij+})$ and $\lambda_n^{ij} - \tau_n^{k-} \leq M_4(1 - v_n^{kij-})$, with $M_3 \geq \tau_n^{k+}$ and $M_4 \geq \lambda_n^{ij}$. Constraints (1t) ensure that the time windows are respected. Constraints (1u) represent capacity constraints for drivers at every node. Constraints (1v) and (1w) impose domain restrictions.

This model is capable of accommodating both single and multiple deliveries for multi-store orders placed by customers, while also allowing drivers to coordinate and transfer orders through transshipment, resulting in consolidated-order delivery with transshipment (CODT). To evaluate the value of consolidation and transshipment in multi-store order delivery, we demonstrate ways to customize $\mathrm{M}(\mathcal{I}, \mathcal{J}, \mathcal{K})$ to model the separated-order delivery (SOD) and the consolidated-order delivery (COD) without transshipment.

The model for SOD requires that orders from each store are handled and delivered separately, which can be structured as a series of programs. For all $j' \in \mathcal{J}$,

$$\mathrm{M}_{\mathrm{SOD}}(\mathcal{I}_{j'}, \mathcal{J}_{j'}, \mathcal{K}_{j'}) = \min_{\substack{x,y,z,\\ \tau,v,\lambda}} \quad \rho_1 \max_{\substack{k\in\mathcal{K}_{j'}\\ i\in\mathcal{I}_{j'}}} \{\tau_i^{k+}\} + \rho_2 \sum_{k\in\mathcal{K}_{j'}} \sum_{n\in\mathcal{N}_{j'}} \sum_{\substack{n'\in\mathcal{N}_{j'},\\ n'\neq n}} t_{nn'} x_{nn'}^k \tag{2a}$$

$$\text{s.t.} \quad (1b) - (1w),$$

$$v_j^{kij-} = v_i^{kij+}, \quad \forall i \in \mathcal{I}_{j'}, j \in \mathcal{J}_{j'}, k \in \mathcal{K}_{j'}. \tag{2b}$$

The objective function (2a) specifies that each store optimizes their delivery operations separately to serve its respective customers. Here, $\mathcal{J}_{j'}$ denotes the set containing only store j' (i.e., $\mathcal{J}_{j'} = \{j'\}$). The set $\mathcal{I}_{j'}$ denotes customers who place orders from store j' (i.e., $\mathcal{I}_{j'} = \{i \in \mathcal{I} | e_{ij'} = 1\}$). There may be overlaps between different $\mathcal{I}_{j'}$ since customers can place orders from multiple stores. Similarly, $\mathcal{K}_{j'}$

denotes the drivers who serve orders from store j', and they jointly form a partition of $\mathcal{K}$. Furthermore, $\mathcal{N}_{j'} = \mathcal{I}_{j'} \cup \mathcal{J}_{j'} \cup \mathcal{O}_{j'}$, where $\mathcal{O}_{j'}$ is the set of origin nodes for drivers $k \in \mathcal{K}_{j'}$. To ensure at least one driver is available for each store, we assume that the number of drivers exceeds the number of stores. Note that the pre-assignment of drivers to stores (i.e., the set $\mathcal{K}_{j'}$) affects the $\mathrm{M}_{\mathrm{SOD}}$. To focus solely on the difference between separation and consolidation business models, we evaluate all possible partitions of the driver set $\mathcal{K}$ for the SOD. For each partition, we determine the worst-case performance across all stores by finding the maximum optimal objective value among them. The partition that yields the minimum worst-case performance (i.e., $\min\{\max_{j' \in \mathcal{J}} \mathrm{M}_{\mathrm{SOD}}(\mathcal{I}_{j'}, \mathcal{J}_{j'}, \mathcal{K}_{j'})\}$) is selected. This method works because our primary objective is to ensure on-time delivery for all customers, with the main target being to minimize the latest delivery time. By using this method, we can compare separation and consolidation models fairly without the results being influenced by the initial pre-assignment of drivers. Constraints (1b)–(1w) are applied, with $\mathcal{I}, \mathcal{J}, \mathcal{K}, \mathcal{N}$ replaced by $\mathcal{I}_{j'}, \mathcal{J}_{j'}, \mathcal{K}_{j'}, \mathcal{N}_{j'}$. Constraints (2b) state that item ij is served by the same driver k encompassing both the pickup from store j and the delivery to customer i, implying no transshipment is allowed.

The model for COD without transshipment is:

$$\mathrm{M}_{\mathrm{COD}}(\mathcal{I}, \mathcal{J}, \mathcal{K}) = \min_{\substack{x, y, z, \\ \tau, w, v, \lambda}} \quad \rho_1 \max_{\substack{k \in \mathcal{K} \\ i \in \mathcal{I}}} \left\{\tau_i^{k+}\right\} + \rho_2 \sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}} \sum_{\substack{n' \in \mathcal{N}, \\ n' \neq n}} t_{nn'} x_{nn'}^k \tag{3a}$$

$$\text{s.t.} \qquad (1b) - (1w),$$

$$v_j^{kij-} = v_i^{kij+}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}, \tag{3b}$$

$$\sum_{k \in \mathcal{K}} z_i^k = 1, \qquad \forall i \in \mathcal{I}. \tag{3c}$$

Constraints (3b) imply that item ij is assigned to the same driver for both pickup and delivery, and Constraints (3c) enforce a single delivery, ensuring that each customer is served exactly once for delivery, resulting in a consolidated-order delivery system without transshipment.

3.3 Dynamic problem and waiting strategy

The MSOP addresses the problem of serving customers who place orders in a specified time period, with transshipment facilitating spatial consolidation at both the store and customer levels. Since customer orders arrive dynamically over time in realistic settings, temporal consolidation can further improve operations by implementing an effective waiting strategy.

Assuming that customer arrivals follow a stochastic process, such as a Poisson process, the delivery system can be optimized by determining an appropriate *re-optimization interval*, defined as the time period between two consecutive optimizations. For each optimization, the system collects information on new customer orders that arrive within the interval, updates driver availability, allocates orders to available drivers, and plans the routing accordingly.

A longer waiting strategy can reduce the total travel time but may increase delivery delays. Conversely, an event-triggered strategy, which re-optimizes the system upon each order arrival without waiting, accelerates the delivery speed but may increase the total travel time. The objective of the dynamic problem is to identify the waiting strategy that balances delivery time and waiting delays by determining the re-optimization interval, which also corresponds to the customer batching size during that interval.

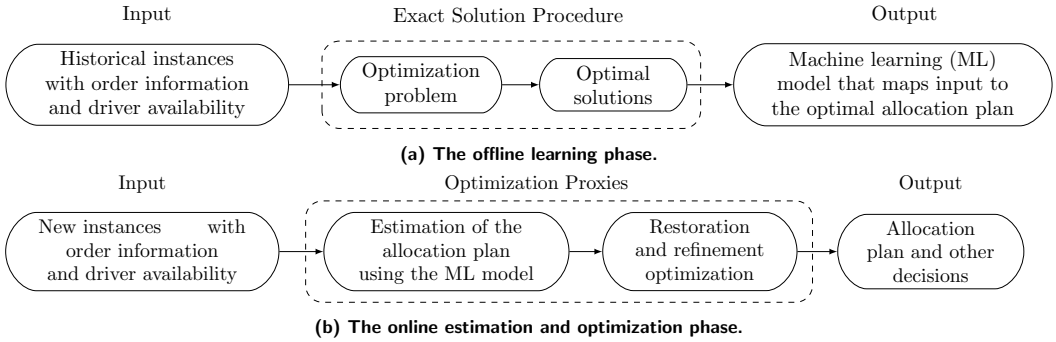
4 Solution procedure

Solving the model for MSOP to optimality is computationally demanding due to the large number of constraints and variables arising from routing and time considerations. To address this, we adopt a

learning-to-optimize approach that combines an offline learning phase with an online estimation and optimization phase. We next present the details of this framework in Section 4.1, introduce three learning methods in Section 4.2, and explain the optimization process in Section 4.3.

4.1 Learning-to-optimize method

The learning-to-optimize method is illustrated in Figure 2 and consists of two key steps. The first is the *offline learning phase*, as shown in Figure 2a, where a mapping is trained using machine learning models on historical instances. This step is conducted only once. Specifically, historical instances, which are solved to optimality using exact optimization techniques, are used to train a neural network model that learns the mapping from instance data to optimal allocation plans. The instance data includes order details, store and customer locations, initial driver positions, travel times between locations, and driver availability. The allocation plan determines the nodes visited by drivers, ensuring that each order is picked up from stores and delivered to customers. The second step is the *online estimation and optimization phase*, as shown in Figure 2b. For each new instance, the estimated allocation plans are obtained from the learning model and then serve as a lower bound for the corresponding variables in the optimization problem. This optimization problem is a mixed-integer linear program, where restoration and refinement work together to ensure feasibility and improve the solution: restoration addresses potential infeasibility caused by the estimated allocation (i.e., the optimization input) underestimating time constraints and capacity limitations, while refinement further improves the estimated allocation decisions, refining the initial solutions to optimal ones through optimization. This process narrows the feasible solution space, provides flexibility to improve the estimated allocation plans, and completes the optimal driver routes and efficient paths for the MSOP. This approach can offload a portion of the computational workload to the offline phase, facilitating the near real-time generation of high-quality solutions.



(a) The offline learning phase.

(b) The online estimation and optimization phase.

Figure 2: Learning-to-optimize Method.

4.2 Learning methods

Three neural network models are implemented to train the mapping from instance information to allocation plans. Let $\mathcal{N}$ denote the set of nodes, including driver origins, stores, and customers that drivers may visit, $\mathcal{K}$ denote the set of drivers, and $\mathcal{L}$ denote the feature space. We use bold symbols to represent vectors. We define the neural network model g, parameterized by θ, that maps the set of input features $\boldsymbol{f_n^k} \in \mathbb{R}^{|\mathcal{L}|}$, for each node $n \in \mathcal{N}$ and driver $k \in \mathcal{K}$ to a binary allocation plan $\tilde{\boldsymbol{z}} \in \{0,1\}^{|\mathcal{N}|\times|\mathcal{K}|}$, which represents the decision of assigning driver k to visit node n. The function is presented by

$$g_\theta : \mathbb{R}^{|\mathcal{N}|\times|\mathcal{K}|\times|\mathcal{L}|} \to \{0,1\}^{|\mathcal{N}|\times|\mathcal{K}|}. \tag{4}$$

All information related to orders, customers, stores, and drivers constitutes the feature set. These include store locations for order pickup, customer locations for delivery, driver initial locations, travel

times between nodes, and nearest driver allocation to each node. Specifically, the features $\boldsymbol{f_n^k}$ for node n that can be served by driver k include: (1) node latitude; (2) node longitude; (3) distance between the driver and the node; (4) nearest driver allocation indicator; and (5) the ratio of the number of customers to the number of available drivers. Additionally, we generate additional features based on a *connected node* (CN) set for each node and driver. For store nodes, this set comprises the customers ordering from that store; for customer nodes, it includes the stores from which they have ordered; and for drivers, it consists of nodes that would be assigned to each driver based on the nearest allocation method. The additional features related to the CN set are: (6) the number of nodes in the CN union sets of both the node and the driver; (7) the Traveling Salesman Problem (TSP) cost for visiting nodes in this union CN set; and (8) the convex hull area enclosing the nodes in this union CN set.

4.2.1 Driver Assignment Neural Network (DANN).

This method uses a neural network for binary prediction to determine whether driver k is assigned to node n. A training sample consists of a node-driver pair, where the input features are $\boldsymbol{f_n^k}$ and the label is z_n^{k*} for node n and driver k. Each driver is considered independently of all other drivers.

We implement a multi-layer neural network to learn the function g_θ, capturing the relationships between the input features and the binary decision outputs. The model predicts the probability of assigning node n to driver k:

$$Pr\left(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n^k}\right) = \sigma\left(\Theta(\boldsymbol{f_n^k})\right), \qquad \forall n \in \mathcal{N}, k \in \mathcal{K},$$

where $\Theta(\boldsymbol{f_n^k}) : \mathbb{R}^{|\mathcal{L}|} \to \mathbb{R}$ represents the neural network's output after multiple layers of computation on the input features $\boldsymbol{f_n^k} \in \mathbb{R}^{|\mathcal{L}|}$. Since $\tilde{z}_n^k$ is binary, the final layer of the neural network $\sigma(\cdot)$ applies the sigmoid activation function, $\sigma(x) = \frac{1}{1+e^{-x}}$, ensuring that the output is a probability between 0 and 1. If $Pr(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n^k}) > 0.5$, then $\tilde{z}_n^k = 1$, indicating that driver k is assigned to visit node n.

4.2.2 Driver Classification Neural Network (DCNN).

This method treats driver assignment as a classification problem, where each driver is considered a distinct class. The neural network classifies each node into one of the available "driver classes", ensuring that at least one driver is assigned to visit each node. A training sample consists of a node with multiple drivers as potential choices, where the input features are $\left\{\boldsymbol{f_n^k}\right\}_{k\in\mathcal{K}}$ and the labels are $\left\{z_n^{k*}\right\}_{k\in\mathcal{K}}$ for node n.

We again implement a multi-layer neural network to learn the function g_θ. For node n, the model outputs a probability distribution over classes k as follows:

$$Pr\left(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n}\right) = \left[\text{softmax}\left(\Theta\left(\boldsymbol{f_n}\right)\right)\right]_k, \qquad \forall n \in \mathcal{N}, k \in \mathcal{K},$$

where $\Theta(\boldsymbol{f_n}) : \mathbb{R}^{|\mathcal{K}|\times|\mathcal{L}|} \to \mathbb{R}^{|\mathcal{K}|}$ represents the neural network's computation through multiple layers on the features $\boldsymbol{f_n} \in \mathbb{R}^{|\mathcal{L}|\times|\mathcal{K}|}$ for node n and all drivers. The softmax function outputs a probability distribution over the $|\mathcal{K}|$ classes, ensuring that $\sum_{k=1}^{|\mathcal{K}|} Pr(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n}) = 1$. The driver k with the highest probability $Pr(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n})$ is selected. Thus, $\tilde{z}_n^k = 1$ for the driver with the highest score, and $\tilde{z}_n^k = 0$ for all other drivers.

4.2.3 Graph-based Neural Network (GNN).

This method employs a graph neural network to learn driver allocation plans by predicting edge labels that indicate whether a driver should visit a node within the graph structure, as illustrated in Figure 3, depicting an instance or area to be served.

Unlike traditional neural networks, the graph structure connects all nodes and includes edge features, global features, and labels. The *node features* $\boldsymbol{f_n} \in \mathbb{R}^{|\mathcal{L}|}$ include: (1) driver type: 1 if the

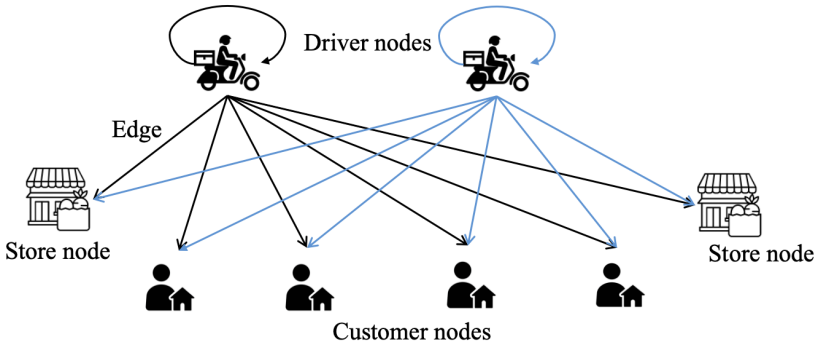


Figure 3: Graph-based Neural Network structure for the case with 2 drivers, 2 stores, and 4 customers.

node represents a driver, 0 otherwise; (2) store type: 1 if the node represents a store, 0 otherwise; (3) customer type: 1 if the node represents a customer, 0 otherwise; (4) node latitude; (5) node longitude; (6) the number of nodes in the CN set; (7) the TSP cost of visiting the nodes in the CN set; and (8) the convex hull area enclosing the nodes in the CN set. The *edge features* $\boldsymbol{e_{nn'}} \in \mathbb{R}^{|\mathcal{M}|}$ of all edges from nodes to drivers consist of (1) distance between the driver and the nodes; and (2) nearest driver allocation to the node. The *global feature* $g \in \mathbb{R}$ is the ratio of the number of customers to the number of available drivers. These features are similar to those used in DANN and DCNN, but are now represented in a graph structure. The *edge label* to be learned represents the optimal allocation of drivers to nodes. The GNN model leverages node features, edge features, and global features through multiple layers to capture complex relationships within the graph. A training sample consists of all node with all drivers as potential choices, where the input features are $\left(\{\boldsymbol{f_n}\}_{n\in\mathcal{N}}, \{\boldsymbol{e_{nn'}}\}_{n,n'\in\mathcal{N}}, g\right)$ and the labels are $\left\{z_n^{k*}\right\}_{k\in\mathcal{K},n\in\mathcal{N}}$ for the whole graph.

At each layer t, the embedding of node n, denoted $\boldsymbol{h_n^{(t)}}$, is updated based on information from its neighboring nodes. The update is computed as:

$$\boldsymbol{h_n^{(t+1)}} = \phi\left(\boldsymbol{h_n^{(t)}}, \sum_{n'\in \text{Neight}(n)} \psi\left(\boldsymbol{h_n^{(t)}}, \boldsymbol{h_{n'}^{(t)}}, \boldsymbol{e_{nn'}}\right), g\right),$$

where $\boldsymbol{h_n^{(t)}}$ is the embedding of node n at layer t with $\boldsymbol{h_n^{(0)}} = \boldsymbol{f_n}$ and $\boldsymbol{f_n} \in \mathbb{R}^{|\mathcal{L}|}$, Neight$(n)$ denotes the set of neighbors of node n, $\boldsymbol{e_{nn'}} \in \mathbb{R}^{|\mathcal{M}|}$ is the feature vector for the edge between node n and its neighbor n', g is the global feature, and ϕ and ψ are neural network layers.

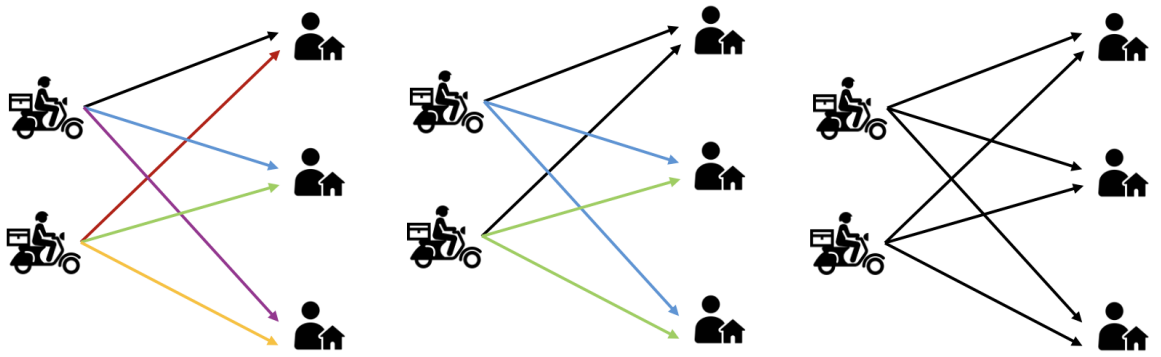
After T layers, the GNN produces final embeddings $\boldsymbol{h_n^{(T)}}$ for each node n. For all edges connecting node n, the final layer applies a softmax function to output a probability distribution over all possible drivers. The probability of assigning driver k to node n is given by:

$$Pr\left(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n}, \boldsymbol{e_{nn'}}, g\right) = \left[\text{softmax}\left(\varphi\left(\boldsymbol{h_n^{(T)}}, \boldsymbol{h_{n'}^{(T)}}, \boldsymbol{e_{nn'}}, g\right)\right)\right]_k, \qquad \forall n \in \mathcal{N}, k \in \mathcal{K},$$

where $\varphi : \mathbb{R}^{|\mathcal{N}|\times|\mathcal{L}|} \times \mathbb{R}^{|\mathcal{K}|\times|\mathcal{N}|\times|\mathcal{M}|} \times \mathbb{R} \to \mathbb{R}^{|\mathcal{K}|\times|\mathcal{N}|}$ is a neural network layer that combines the final node embeddings, edge features, and global features. The softmax function normalizes the edge outputs from each node n to all driver nodes, providing a probability distribution over drivers for node n, ensuring that $\sum_{k=1}^{|\mathcal{K}|} Pr\left(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n}, \boldsymbol{e_{nn'}}, g\right) = 1$. The driver k with the highest probability $Pr\left(\tilde{z}_n^k = 1 \mid \boldsymbol{f_n}, \boldsymbol{e_{nn'}}, g\right)$ is selected to serve node n. This GNN framework effectively captures node interactions, edge features, and global features to predict the optimal driver-to-node allocations within the graph.

4.2.4 Summary.

Overall, DANN, DCNN, and GNN use the same features but adopt different graphical structures for generating data samples to learn the allocation decisions. DANN treats each driver-node pair independently (see Figure 4a). DCNN processes each node independently while incorporating driver information (see Figure 4b). In contrast, GNN considers each instance as an interconnected network, allowing both driver and node information to be transferable during the learning process (see Figure 4c). In addition to these three learning methods, we also include Nearest Driver Allocation (NDA) as a benchmark, which assigns the closest driver to visit each store and customer node.



(a) DANN. (b) CDCNN. (c) GNN.

Figure 4: Learning models for allocation decisions with different ways of generating data samples. The lines represent allocation decisions for a driver visiting a node. Lines of the same color indicate a single sample, while different colors correspond to different samples. In DANN, each driver-node pair is represented by a unique color; in DCNN, each node is assigned a color; and in GNN, the entire instance uses the same color.

4.3 MILP-based restoration and refinement problem

In the optimization phase, we formulate the restoration and refinement problem (RRP) as an MILP to restore feasibility, refine allocation decisions, and derive optimal solutions for other decisions. For decision refinement, the estimated allocation $\tilde{z}_n^k$ serves as a lower bound for the allocation decision z_n^k, allowing flexibility in assigning drivers to additional nodes and enabling the optimization of other decisions. However, given the lower bounds of allocation plans, the constraints in Equations (1t) and (1u) may be violated. It is essential to ensure that all items ordered by customer i from store j are delivered within the specified time window while respecting capacity limitations. In other words, the machine learning output may underestimate the time constraints and capacity limitations, which could render Problem 1 infeasible given the estimated $\tilde{z}_n^k$. To address this, we apply soft time windows and capacity constraints while also minimizing the slackness to restore feasibility. The model for the RRP is formulated as follows:

$$\mathrm{M}_{\mathrm{RRP}}(\mathcal{I},\mathcal{J},\mathcal{K}) = \min_{\substack{x,y,z,\\ \tau,w,v,\lambda,s}} \quad \rho_1 \max_{\substack{k\in\mathcal{K}\\ i\in\mathcal{I}}} \left\{\tau_i^{k+}\right\} + \rho_2 \sum_{\substack{k\in\mathcal{K}\\ n\in\mathcal{N}}} \sum_{\substack{n'\in\mathcal{N},\\ n'\neq n}} t_{nn'} x_{nn'}^k + \rho_3 \sum_{\substack{n\in\mathcal{N}\\ k\in\mathcal{K}}} \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}}} (s_n^{1ij} + s_n^{2ij} + s_n^{3k}) \tag{5a}$$

$$\text{s.t.} \quad (1b)-(1s), (1v)-(1w),$$

$$z_n^k \geq \tilde{z}_n^k, \qquad \forall k\in\mathcal{K}, n\in\mathcal{N}, \tag{5b}$$

$$\alpha_i - s_i^{1ij} \leq \lambda_i^{ij} \leq \beta_n + s_i^{2ij}, \quad \forall i\in\mathcal{I}, j\in\mathcal{J}, \tag{5c}$$

$$\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} p_{ij} v_n^{kij-} \leq q^k + s_n^{3k}, \qquad \forall n\in\mathcal{N}, k\in\mathcal{K}, \tag{5d}$$

$$s_i^{1ij}, s_i^{2ij}, s_n^{3k} \geq 0, \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}, k\in\mathcal{K}. \tag{5e}$$

The objective function (5a) states that, in addition to minimizing the weighted average of the latest delivery time and the total travel time, we also minimize the penalty for violating time window and capacity constraints, which cannot be avoided due to the input allocation. Note that ρ_3 represents the penalty weight, and it should be set to a large value relative to ρ_1 and ρ_2 to avoid violations as much as possible. Constraints (5b) ensure that the estimated $\tilde{z}$ serves as a lower bound, while Constraints (5c) and (5d) represent soft time windows and capacity constraints with slack variables s_i^{1ij}, s_i^{2ij}, and s_n^{3k} to maintain feasibility. Finally, Constraints (5d) impose domain restrictions on the slack variables.

5 Numerical study

In this section, we first introduce a real-world dataset, performance metrics, and implementation details. We then compare the performance of three systems to evaluate the benefits of offering consolidation and transshipment for the multi-store order delivery. Due to the computational challenges involved in solving the problem, we employ learning-to-optimize techniques to accelerate the solution process through various learning methods and compare the effectiveness of different learning-based optimization proxies. Finally, we conduct dynamic experimentations in a practical setting to serve customers in areas with varying customer locations and arrival rates, aiming to identify the most efficient delivery system and waiting strategy.

5.1 Dataset and implementation details

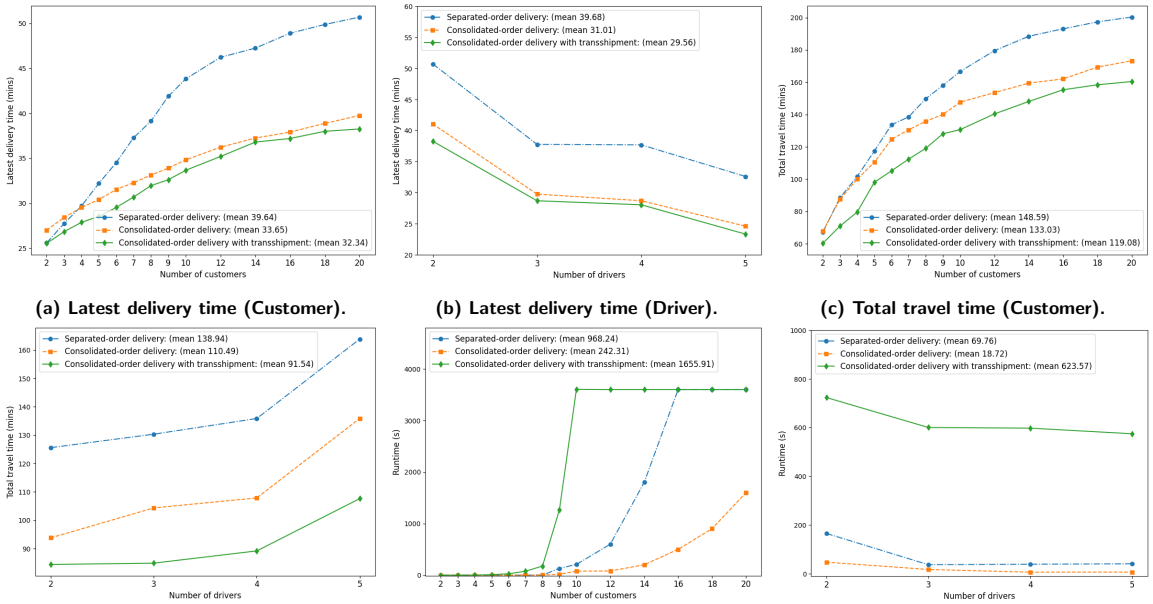
We use a customer location dataset from four regions in the U.S. (Los Angeles, Seattle, Tacoma, and Orange) provided by Amazon (Merchan et al. 2021), which contains the perturbed locations of customers. We obtain the expected travel times using the Google API. Instances with varying scales are created from the dataset, with customer numbers ranging from 2 to 20 and driver numbers ranging from two to five. To assess the benefits of consolidating orders from multiple stores, we assume customers place orders from at least two stores and up to four stores, and whether each customer places orders from each store is randomly generated. Driver initial locations and store locations are fixed in each region. Customer locations are sampled uniformly from the dataset, with each location having an equal probability of selection, referred to as *uniformly sampled customers*. We also consider *clustered customers*, where a central point is pre-selected, and locations closer to this center have a higher probability of being chosen as customer locations. The start time windows are uniformly generated within a range of 0 to 20, and the end time windows are within a range of 20 to 70. Driver capacity is set to 100, with item sizes randomly generated between 0 and 10 as integers. Our primary objective is to ensure on-time delivery for all customers by minimizing the latest delivery time, so the weight ρ_1 is set to 1. A smaller weight $\rho_2 = 0.01$ is used to discourage unnecessary long total travel times. To minimize constraint violations, a penalty weight $\rho_3 = 100$ is applied.

To compare the performance of three delivery systems and four learning methods, we define the following metrics. (1) Delivery time is the duration between each order pickup and delivery, with the latest delivery time being the time to serve the last order arriving within the re-optimization interval. A lower value indicates faster overall delivery. (2) Total travel time is the total time drivers spend traveling, including the time from their starting location to pick up orders and deliver them. The travel time for serving one more customer is calculated as the ratio of total travel time to the number of customers. A lower value indicates reduced overall costs. (3) Wait time is the duration between order placement and pickup. A lower value means quicker driver assignment and faster availability for pickup. (4) Order completion time is the duration from order placement to delivery, including both wait time and delivery time. (5) Runtime is the time required to find final solutions, whether or not learning is used. A lower runtime indicates a more efficient solution method. (6) Learning accuracy is the percentage of correct driver-to-customer allocations. Higher accuracy indicates better learning performance. (7) Allocation percentage (pct.) is the percentage of estimated allocation decisions estimated as 1, indicating that the driver will visit that location. A lower value offers more flexibility

in optimizing the solution. (8) Allocation standard deviation (std.) is the standard deviation of allocation decisions, which indicates the balance of driver workloads. A lower value signifies a more balanced distribution of work. (9) Gap is the difference between the estimated and exact optimal values for key metrics such as objective value, latest delivery time, and total travel time. A lower gap means the learning-based solution is closer to the true optimal solution.

We implement our algorithms using Python 3.10 and Gurobi 10.0.2 on a local computer equipped with a 2 GHz Quad-Core Intel Core i5 processor and 16 GB of RAM, supplemented by resources from Compute Canada's Graham cluster, which includes Multi-Core Intel Xeon processors (20 to 36 cores per node) and standard compute nodes with 64 GB of RAM. The optimization time limit for the exact solution procedure is set to 3600 seconds, while the time limit for learning-to-optimize during the comparison of learning methods is set to 600 seconds. For implementations in dynamic environments, this limit is further reduced to 300 seconds when applying learning-to-optimize with GNN, ensuring efficient real-time decision-making.

5.2 Comparison of delivery systems



(a) Latest delivery time (Customer). **(b) Latest delivery time (Driver).** **(c) Total travel time (Customer).**

(d) Total travel time (Driver). **(e) Runtime (Customer).** **(f) Runtime (Driver).**

Figure 5: Comparison of delivery systems with various number of customers and drivers.

To compare the three systems, including separated order delivery (SOD), consolidated order delivery (COD), and consolidated order delivery with transshipment (CODT), we solve the models M_{SOD}, M_{COD}, and M using Gurobi within the specified time limit across various instances.

Figures 5a and 5b plot the latest delivery time, defined as the maximum duration between order pickup and delivery across customers. The SOD is more efficient than the COD when there are few customers to serve, as the benefits of consolidating requests are minimal. However, as more customers join the system and place orders, the COD begins to dominate the SOD, and this dominance increases with the number of customers. Additionally, the COD consistently yields a shorter total travel time, which represents the total time drivers take to complete deliveries for all requests (see Figures 5c and 5d). It also converges to optimality faster (see Figures 5e and 5f).

In contrast, the CODT is always the most efficient among all three systems in terms of delivery time for serving each customer and the total travel time for fulfilling all requests, regardless of the

instance scale (see Figures 5a–5d). However, due to the complexity of coordination and the flexibility of transshipment at any location, solving the CODT using exact methods requires substantially more computational time (see Figures 5e and 5f). Instances with more than 12 customers are unlikely to converge to optimality within the 3600-second time limit, and a lower bound may not be found to produce a feasible solution within 600 seconds. To address this challenge, we implement learning-to-optimize techniques to explore whether high-quality solutions can be achieved more efficiently, mitigating this computational drawback.

Insight 1. *Consolidated-order delivery with transshipment is the most efficient of the three delivery systems considered, achieving the shortest delivery time for serving each customer and the lowest total travel time for fulfilling all requests, though it requires more time to solve using exact methods.*

Insight 2. *Without transshipment, consolidated-order delivery outperforms separated-order delivery in delivery time, total travel time, and exact solution time, except when the customer number is very small.*

5.3 Comparison of learning algorithms

We compare four methods, including NDA, DANN, DCNN, and GNN, that map instance information to allocation decisions. We implement the training process under three different instance scales: small-scale instances, where the number of customers ($|\mathcal{I}|$) ranges from two to seven and the number of drivers ($|\mathcal{K}|$) ranges from two to five; medium-scale instances, where $|\mathcal{I}|$ ranges from eight to ten and $|\mathcal{K}|$ ranges from two to five; and large-scale instances that fail to converge to optimality, where $|\mathcal{I}|$ ranges from 12 to 20 and $|\mathcal{K}|$ ranges from two to five. First, we train using both small and medium-scale instances and test on small, medium, and large-scale instances. In the second set of experiments, we train solely on small-scale instances but test on all three scales. This approach allows us to assess the model's ability to generalize and predict decisions for larger-scale instances that may not have been included in the training set.

Table 1: Best Learning Method for Uniformly Sampled Customers in the Learning Process.

Training Scale	Testing Type	Testing Scale			Best Method	Accuracy	Allocation Pct. (%)	Allocation Std.
		$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{K}\rvert$	NoI				
$\lvert\mathcal{I}\rvert$: [2, 10]; $\lvert\mathcal{K}\rvert$: [2, 5]; NoI: 3600	T1	[2, 7]	[2, 3]	1200	GNN	0.83	41.67	0.31
	T1	[2, 7]	[4, 5]	1200	GNN	0.92	22.50	0.43
	T1	[8, 10]	[2, 3]	600	DCNN	0.84	41.67	0.30
	T1	[8, 10]	[4, 5]	600	DCNN	0.90	22.50	0.04
	T5	[12, 20]	[2, 5]	400	GNN	0.74	32.08	0.55
	T6	[2, 20]	[2, 5]	4000	GNN	0.88	32.08	0.15
$\lvert\mathcal{I}\rvert$: [2, 7]; $\lvert\mathcal{K}\rvert$: [2, 3]; NoI: 1200	T1	[2, 7]	[2, 3]	1200	GNN	0.84	41.67	0.30
	T2	[2, 7]	[4, 5]	1200	GNN	0.92	22.50	0.70
	T3	[8, 10]	[2, 3]	600	GNN	0.86	41.67	0.35
	T4	[8, 10]	[4, 5]	600	GNN	0.90	22.50	0.09
	T5	[12, 20]	[2, 5]	400	GNN	0.73	32.08	0.40
	T6	[2, 20]	[2, 5]	4000	GNN	0.87	32.08	0.15

Note. $|\mathcal{I}|$: range of customer numbers; $|\mathcal{K}|$: range of driver numbers; NoI: number of instances. T1 corresponds to testing and training with the same scale, while T2−T5 all involve larger scales. Specifically, T2 has a larger customer scale, T3 a larger driver scale, and T4 both scales increased. T5 represents overall large-scale instances, and T6 includes all testing instances.

To simplify and clarify the results, we present the best-performing learning method and its learning performance for uniformly sampled customers in Table 1, with its re-optimization performance in Table 2. For clustered customers, the best learning method and its performance are shown in Table 3, and the corresponding optimization performance is provided in Table 4. To differentiate between the training and testing scales, we consider six testing types, each comprising distinct testing instances. T1 corresponds to the scenario where the testing scale aligns with the training scale. T2 pertains to cases with a larger customer scale, while T3 refers to instances with a larger driver scale. T4 encompasses

Table 2: Best Learning Method for Uniformly Sampled Customers in the Optimization Process.

Training Scale	Testing Type	Testing Scale			Best Method	Objective Gap(%)	Delivery Time Gap(%)	Travel Time Gap(%)	Runtime (s)
		$\|\mathcal{I}\|$	$\|\mathcal{K}\|$	NoI					
$\|\mathcal{I}\|$: [2, 10]; $\|\mathcal{K}\|$: [2, 5]; NoI: 3600	T1	[2, 7]	[2, 3]	1200	DCNN	2.10	2.10	1.19	5
	T1	[2, 7]	[4, 5]	1200	DCNN	0.94	0.94	0.15	5
	T1	[8, 10]	[2, 3]	600	DCNN	1.66	1.65	4.92	14
	T1	[8, 10]	[4, 5]	600	DCNN	1.66	1.68	10.42	30
	T5	[12, 20]	[2, 5]	400	GNN	1.19	1.23	16.76	457
	T6	[2, 20]	[2, 5]	4000	DCNN	1.56	1.56	2.75	40
$\|\mathcal{I}\|$: [2, 7]; $\|\mathcal{K}\|$: [2, 3]; NoI: 1200	T1	[2, 7]	[2, 3]	1200	DCNN	2.07	2.07	1.67	6
	T2	[2, 7]	[4, 5]	1200	NDA	1.25	1.25	0.37	6
	T3	[8, 10]	[2, 3]	600	GNN	3.19	3.17	5.30	21
	T4	[8, 10]	[4, 5]	600	GNN	1.89	1.92	12.13	33
	T5	[12, 20]	[2, 5]	400	GNN	1.73	1.77	17.56	406
	T6	[2, 20]	[2, 5]	4000	GNN	2.78	2.78	3.80	38

Table 3: Best Learning Method for Clustered Customers in the Learning Process.

Training Scale	Testing Type	Testing Scale			Best Method	Accuracy	Allocation Pct. (%)	Allocation Std.
		$\|\mathcal{I}\|$	$\|\mathcal{K}\|$	NoI				
$\|\mathcal{I}\|$: [2, 10]; $\|\mathcal{K}\|$: [2, 5]; NoI: 3600	T1	[2, 7]	[2, 3]	1200	GNN	0.80	41.67	0.57
	T1	[2, 7]	[4, 5]	1200	GNN	0.90	22.50	0.09
	T1	[8, 10]	[2, 3]	600	GNN	0.73	41.67	0.29
	T1	[8, 10]	[4, 5]	600	GNN	0.88	18.71	0.45
	T5	[12, 20]	[2, 5]	400	GNN	0.69	32.08	0.76
	T6	[2, 20]	[2, 5]	4000	GNN	0.84	32.08	0.51
$\|\mathcal{I}\|$: [2, 7]; $\|\mathcal{K}\|$: [2, 3]; NoI: 1200	T1	[2, 7]	[2, 3]	1200	GNN	0.81	41.67	0.19
	T2	[2, 7]	[4, 5]	1200	GNN	0.89	22.50	0.35
	T3	[8, 10]	[2, 3]	600	GNN	0.70	41.67	0.76
	T4	[8, 10]	[4, 5]	600	GNN	0.86	22.50	0.30
	T5	[12, 20]	[2, 5]	400	GNN	0.69	32.08	0.17
	T6	[2, 20]	[2, 5]	4000	GNN	0.83	32.08	0.94

Table 4: Best Learning Method for Clustered Customers in the Optimization Process.

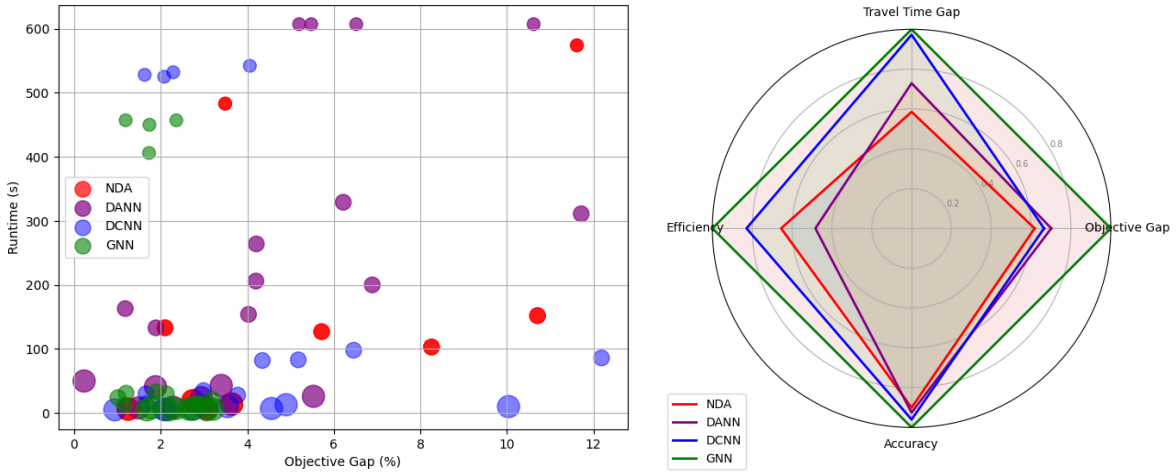
Training Scale	Testing Type	Testing Scale			Best Method	Objective Gap(%)	Delivery Time Gap(%)	Travel Time Gap(%)	Runtime (s)
		$\|\mathcal{I}\|$	$\|\mathcal{K}\|$	NoI					
$\|\mathcal{I}\|$: [2, 10]; $\|\mathcal{K}\|$: [2, 5]; NoI: 3600	T1	[2, 7]	[2, 3]	1200	DANN	1.88	1.86	2.08	6
	T1	[2, 7]	[4, 5]	1200	DANN	0.23	0.23	4.34	5
	T1	[8, 10]	[2, 3]	600	GNN	1.16	1.12	11.79	12
	T1	[8, 10]	[4, 5]	600	GNN	1.01	0.97	21.48	24
	T5	[12, 20]	[2, 5]	400	GNN	1.74	1.69	16.76	450
	T6	[2, 20]	[2, 5]	4000	DANN	1.93	1.90	1.47	111
$\|\mathcal{I}\|$: [2, 7]; $\|\mathcal{K}\|$: [2, 3]; NoI: 1200	T1	[2, 7]	[2, 3]	1200	GNN	2.35	2.33	4.30	7
	T2	[2, 7]	[4, 5]	1200	GNN	1.79	1.73	24.09	8
	T3	[8, 10]	[2, 3]	600	GNN	2.90	2.89	3.57	15
	T4	[8, 10]	[4, 5]	600	GNN	1.20	1.18	7.55	31
	T5	[12, 20]	[2, 5]	400	GNN	2.36	2.30	16.76	457
	T6	[2, 20]	[2, 5]	4000	GNN	2.07	2.07	9.01	38

situations where both customer and driver scales are increased. T5 includes instances characterized by larger overall scales, and T6 represents all testing instances. Both the training scale and testing scale clearly show the range of customer and driver numbers. For the learning process, we present the best method with the highest accuracy, along with its percentage and standard deviation of allocation plans. For the optimization process, we display the best method with the lowest objective gap, including its delivery time gap, total travel time gap, and optimization runtime. More details on the metrics can be found in Section 5.1. Detailed performance results for each method under both uniformly sampled customer and clustered customer cases are provided in Appendix B.

According to Tables 1 and 3, we find that GNN demonstrates the overall best performance in the learning process with the highest accuracy under most scenarios, regardless of whether the testing scale is included in the training set. In instances where both small and medium-scale datasets are included in the training set, DCNN can achieve performance comparable to GNN for those scales. In other words, GNN excels at generalizing and predicting decisions for larger-scale instances that may not have been part of the training data. Both GNN and DCNN perform well if the scale of future instances matches that of historical instances. This difference in performance arises because, in DANN, the assignment decision for each driver-customer pair is considered independently, which means there is no guarantee that each customer will be served by exactly one driver. This is evident in the unstable allocation percentages observed with DANN (see Tables B2 and B4 in Appendix B). In contrast, NDA, DCNN, and GNN ensure that at least one driver is allocated to serve each customer. However, in NDA, the allocation of drivers to customers is based solely on the distances between them. DCNN incorporates more features, leading to higher accuracy compared to NDA. Nevertheless, DCNN still treats customers independently, as its goal is to find the best driver for each individual customer. GNN, on the other hand, connects all customers and drivers through edges, ensuring that every customer is served and allowing for control over the number of customers each driver serves by summing edge labels.

Tables 2 and 4 display the optimal outputs when the estimated allocation decisions are used as input for optimizing the restoration and refinement problem. First, we observe that higher accuracy does not always correspond to a lower optimality gap. This is because there may be multiple good solutions that yield low delivery times during the optimization process. Second, no single method consistently outperforms the others in all cases. Overall, GNN has a smaller optimality gap, along with lower delivery time and travel time, particularly in the clustered customer cases when the testing scale exceeds the training scale. When the training scale includes the testing scale, DCNN tends to have a smaller gap in scenarios with uniformly sampled customers, along with a lower allocation standard deviation. In contrast, DANN exhibits a smaller gap in cases with clustered customers, accompanied by a lower allocation percentage. A lower allocation standard deviation indicates a more balanced workload among drivers, while a lower allocation percentage provides greater flexibility in finding optimal allocation decisions during the optimization process.

A summary of the re-optimization results across both scenarios, based on different training processes, is provided in Figure 6. Figure 6a displays the trade-off between expected runtime and optimality gap for the four methods across all scenarios, with circle size indicating sample size. In terms of runtime for the optimization process, GNN proves to be the most efficient method, while NDA and DANN are much more time-intensive. A lower runtime for optimization reflects that the input-estimated allocation decisions provide a stronger lower bound. By using learning-to-optimize techniques, we achieve optimal solutions in 30 seconds for medium-scale instances, and in 450 seconds for large-scale instances. In contrast, pure optimization methods may take up to 3600 seconds for medium-scale instances and may fail to converge within 3600 seconds for large-scale cases. Figure 6b presents relative performance metrics, including accuracy, objective gap, travel time gap, and runtime efficiency, relative to the best approach among all methods. A value of 1 indicates the best performance: highest accuracy, lowest objective gap, lowest travel time gap, and highest efficiency. We conclude that GNN delivers the most comprehensive performance across metrics and scenarios,



(a) Trade-off between optimality gap and runtime. **(b) Relative performances of different metrics.**

Figure 6: Comparison of performances in optimization of four learning methods.

consistently achieving the lowest optimization runtime and smallest optimality gap by providing an effective lower bound.

Insight 3. *In a learning-to-optimize framework, we can use learning to obtain a lower bound for the allocation decision, which helps reduce the search space for optimization, thereby accelerating the solution procedure.*

Insight 4. *Overall, GNN performs the best with the highest accuracy, the smallest optimality gap, efficient runtime, and superior scalability for larger instances not included in the training set.*

Insight 5. *In the learning-to-optimize framework, higher accuracy during the learning process does not necessarily lead to a smaller optimality gap in the optimization process. A lower allocation percentage allows for greater flexibility and may lead to a smaller gap, but at the cost of a longer optimization runtime. A lower allocation standard deviation, indicating a balanced workload, may also imply a shorter delivery time and lead to a smaller gap.*

The four learning methods each have their own advantages and disadvantages. (1) NDA is simple to implement and performs well without historical data, effectively generalizing with an increasing number of drivers. However, it struggles with complex customer-driver interactions and performs poorly with clustered customer distributions, as well as struggling to generalize with an increasing number of customers. (2) DANN achieves high accuracy with sufficient training data and works well for instances of similar scale, especially in cases with clustered customer distributions. However, it is computationally inefficient, struggles to generalize with larger customer sets, and yields unstable allocation percentages that can lead to longer travel times. (3) DCNN also achieves high accuracy with sufficient data and works well for instances of similar scale, performing effectively with uniform customer distributions. However, it requires extensive training data and careful tuning, and it struggles to generalize with larger driver sets. (4) GNN excels at modeling complex interactions between customers and drivers using structured graphs, achieving high accuracy across different scenarios while being computationally efficient. It also generalizes well to instances with more customers and drivers. However, it faces challenges in collecting sufficient historical samples, as each instance corresponds to a single graph, and its highest accuracy may not always correlate with the lowest optimality gap. The detailed evidence can be found in Tables B2 to B5 in Appendix B.

5.4 Experimentation in a dynamic environment

In the dynamic delivery problem, the waiting strategy, which groups customer orders based on arrival time, can reduce total driver travel times for serving all customers but may increase order completion

times (the duration from order placement to delivery). To evaluate the efficiency of different waiting strategies and delivery systems, we simulate the dynamic experimentation process under various waiting strategies with limited driver availability.

As shown in Figure 7, customers continuously arrive, and delivery can be optimized either immediately upon order arrival to ensure the fastest delivery or after a fixed interval, optimizing delivery for all batched customers. The simulation models customer arrivals as a Poisson process, with rates varying from 4 to 10 customers every 10 minutes. Simulations are conducted over durations of one and two hours, with the number of drivers ranging from 10 to 20. Each setup is repeated 40 times to ensure robust results, simulating real-world conditions while keeping other parameters consistent with the static model. For a given re-optimization interval, the system optimizes driver assignments and routes to serve orders arriving during the interval and updates availability based on task completion for the next interval. This process repeats with new customer arrivals at each interval. Performance metrics such as completion time, wait time, delivery time, and travel time are recorded. The detailed experimentation process is presented in Table C6 in Appendix C.1. Given limited drivers, two re-optimization strategies are considered when no drivers are available within the rolling horizon. The first strategy always fixes re-optimization time points and assigns customers to the earliest available drivers if no driver is free, with results shown below. The second strategy delays optimization until enough drivers are available, with details and a comparison of these strategies provided in Appendix C.

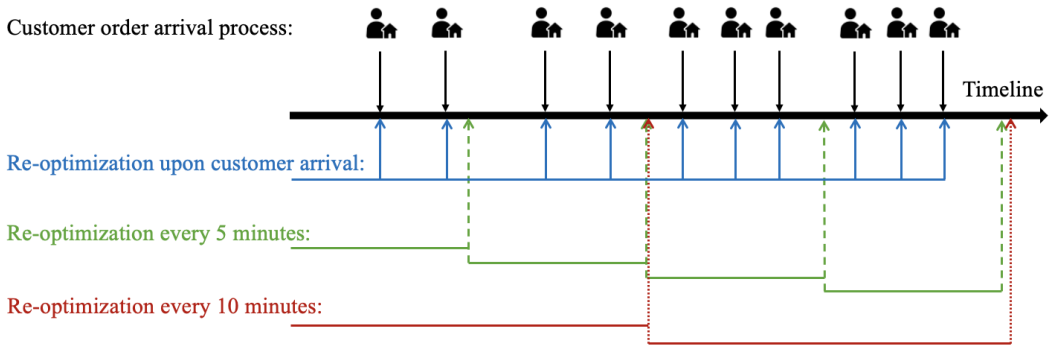
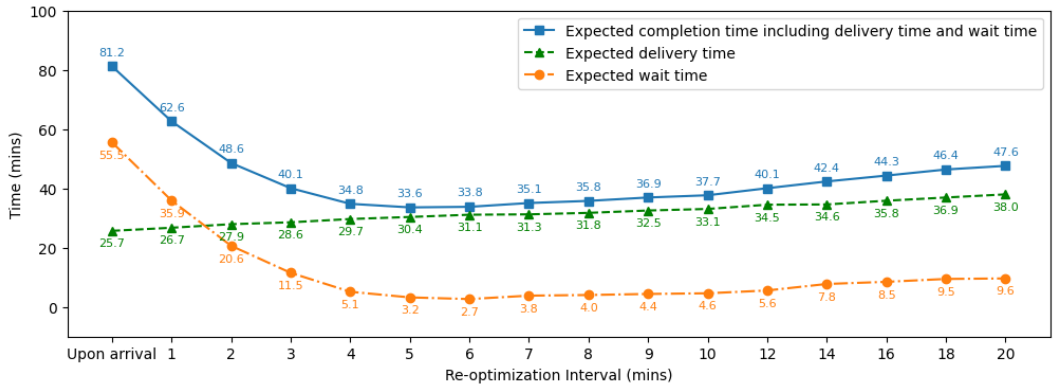


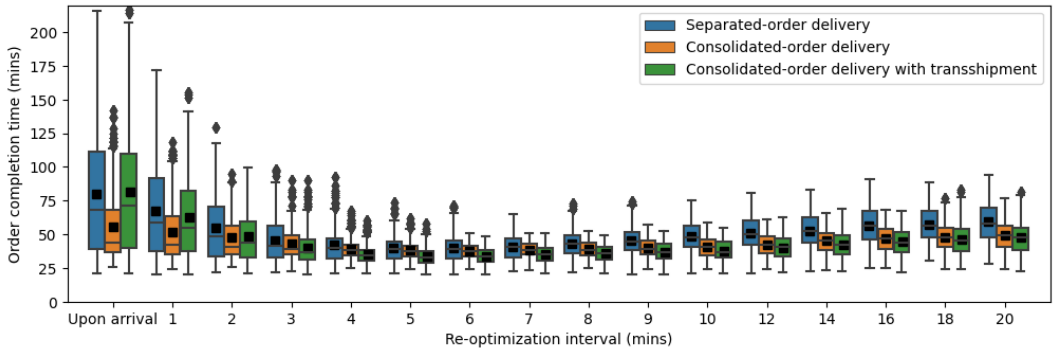
Figure 7: Customer arrival process and dynamic experimentation process.

Figure 8a illustrates the trade-off between delivery time and wait time across re-optimization intervals ranging from event-triggered to 20 minutes, resulting in a function that initially decreases and then increases for order completion time, which includes both delivery and wait times. Under the event-triggered strategy, delivery times are shorter, but wait times are significantly longer. This occurs because drivers are immediately assigned to serve individual customers, resulting in fast delivery but inefficient use of drivers. As a result, drivers are occupied with deliveries before they can serve the next customer, and customers must wait longer for the earliest available driver, reducing overall driver availability and increasing wait times. In contrast, the 20-minute re-optimization interval results in longer delivery times but more moderate wait times. This is because more customers are batched together and served by the same driver, improving the utilization of available drivers. However, since each driver must serve more customers once dispatched, delivery times are longer. Additionally, as customers must wait for batching, the wait time before re-optimization (i.e., before drivers are assigned) increases as the re-optimization interval lengthens. Overall, the five-minute re-optimization interval, which minimizes completion time, strikes a favorable balance between wait time and delivery time. This interval ensures that customers spend minimal time waiting for an available driver while also allowing for prompt delivery once the orders are assigned. Figure 8b displays the completion times for three systems under varying re-optimization intervals. For all three systems, order completion time initially decreases before increasing as the re-optimization interval extends. Notably, all systems achieve the lowest order completion time with the 5-minute re-optimization interval among all waiting

strategies. COD outperforms the other systems in extremely short re-optimization intervals, such as the event-triggered and one-minute or two-minute intervals, while CODT excels across all other re-optimization intervals. The detailed distribution of delivery time, wait time, travel time for serving one additional customer, and customer scale for the three systems under different re-optimization intervals is presented in Figure C1 in Appendix C.1.



(a) Trade-off between wait time and delivery time for consolidated-order delivery with transshipment.



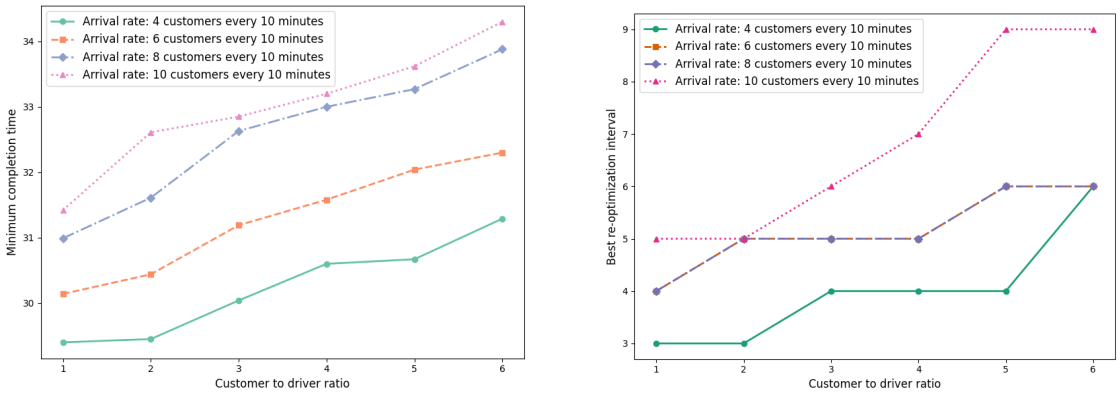
(b) Relative performances of different metrics.

Figure 8: Completion time including wait time and delivery time.

Insight 6. *Consolidated-order delivery is the most efficient system regarding order completion time and average travel time for serving each customer under the event-triggered strategy or a short re-optimization interval. This suggests that when there are few customers to serve, it is advantageous for drivers to pick up all requests from different stores without transshipment and deliver them as a combined order, as the savings in wait time outweigh the costs of delivery time.*

Insight 7. *Consolidated-order delivery with transshipment with a five-minute re-optimization interval is the optimal choice, yielding the lowest order completion time among all delivery systems across different waiting strategies. This approach entails connecting orders that arrive within each five-minute interval, assigning the earliest available drivers to visit different stores, meeting at a transshipment node to exchange items, and making a single delivery to fulfill all requests for each customer.*

The best waiting strategy may vary depending on whether it is a busy or leisurely time for customer arrivals and changes in driver availability. Figure 9 illustrates the best re-optimization interval that results in the minimum order completion time across various customer arrival rates and different ratios of the customer number to available driver number. As the customer arrival rate increases, the best re-optimization interval also becomes larger, leading to a longer completion time. This indicates that when customers arrive more frequently, it is beneficial to wait longer to optimize and fulfill orders. Despite the increased completion time, this strategy still remains more efficient than the others. This is because the reduction in wait time for available drivers outweighs the increase in delivery times.



(a) Minimum completion time under the best re-optimization interval. **(b) Best re-optimization interval.**

Figure 9: Best re-optimization interval for varying ratios of customer number to driver number.

This principle holds true regardless of the customer arrival rate, but it becomes more pronounced when customers arrive more frequently. Moreover, as the ratio of customers to drivers increases, indicating a more limited number of drivers available to serve the same number of customers at a consistent arrival rate, it is advisable to wait longer. This is because once a driver is engaged in the early period, it takes more time for them to become available to serve subsequent customers arriving later.

6 Conclusion

The multi-store consolidated-order delivery service allows customers to purchase products from multiple stores in a single transaction without additional delivery fees, ensuring all items are consolidated and delivered in one combined delivery for greater convenience. This service provides customers with the flexibility to shop across stores, compare prices, save on delivery fees, and receive all items in a single delivery. For stores, it can boost sales by encouraging larger orders, while delivery platforms benefit from reduced driver demand and lower travel times. However, challenges arise in managing longer routes and extended delivery times when a single driver must detour to multiple stores for pickups and then deliver items to various locations. Transshipment, which enables drivers to coordinate and transfer items effectively at transshipment nodes, can mitigate this issue and further improve efficiency.

We develop a mixed-integer linear program for the multi-store order problem with consolidation and transshipment, which can also be adapted to variants without consolidation or transshipment. Solving the model that incorporates complex routing and time variables using exact methods is computationally intensive, particularly for large-scale instances. To overcome this problem, we implement a learning-to-optimize framework that combines neural network-based learning methods with a restoration and refinement optimization process. Using the learning-to-optimize method, the multi-store order problem can achieve high-quality solutions efficiently. In the learning process for allocation plans, we implement four methods, each with its own distinct advantages and disadvantages. The graph-based neural network generally shows superior performance, achieving a better trade-off between optimality gap and solution time while adapting well to larger scales not represented in the training set. This adaptability is due to its ability to exchange information among nodes, edges, and globally. The nearest driver allocation is the simplest to implement, requiring no training or historical samples. Both the driver classification neural network and driver assignment neural network perform well when the testing scale matches the training scale.

Through experiments conducted in various U.S. regions with varying customer locations and arrival frequencies, we find that there is a trade-off between delivery time, which is the duration from order pickup to delivery, and wait time, which is the duration from order placement to pickup. This indicates

the existence of an optimal waiting strategy, influenced by factors such as customer arrival rates and the ratio of drivers to customers. It is beneficial to wait longer to batch more customers in cases of frequent customer arrivals and limited driver availability. Overall, the consolidated-order delivery system with transshipment and a five-minute waiting strategy consistently outperforms the others in terms of order completion time and driver travel time, regardless of customer arrival frequency or driver availability relative to customer demand, due to its superior spatial and temporal consolidation.

Our work has some limitations that can be addressed in future research. In a rolling horizon implementation, earlier-arriving orders should be prioritized over later ones when driver availability is limited, and a piecewise linear function that increases with delivery time for each order could be applied. Future research can also explore stochastic and dynamic programming approaches to address uncertain and time-dependent travel conditions in consolidated delivery services. In our learning process, classification neural networks and graph neural networks are applied independently. However, other efficient machine learning algorithms may yield better performance. For instance, embedding algorithms, which train models independently before combining them into a stronger overall model, or ensemble learning algorithms, which sequentially train multiple models and aggregate their outputs, can enhance performance beyond that of any single model.

Appendix A Summary of notation

The notation is presented in Table A1.

Table A1: Problem notation.

Index	**Description**
$\mathcal{I}$	Set of customers
$\mathcal{J}$	Set of stores
$\mathcal{K}$	Set of drivers
$\mathcal{N}$	Set of nodes, including customers, stores, and driver initial locations
Parameters	**Description**
e_{ij}	Binary parameter indicating if customer i orders from store j
ij	Index of item ordered by customer i from store j
p_{ij}	Size of item ij
q^k	Capacity of driver k
$t_{nn'}$	Travel time between nodes n and n'
$[\alpha_i, \beta_i]$	Time window for customer i
Decisions	**Description**
z_n^k	Binary variable indicating if driver k visits node n
$x_{nn'}^k$	Binary variable indicating if driver k travels from node n to node n'
$y_{nn'}^{ij}$	Binary variable indicating if item ij travels from node n to node n' during the trip
v_n^{kij+}	Binary variable indicating if item ij arrives at node n via driver k
v_n^{kij-}	Binary variable indicating if item ij departs from node n via driver k
τ_n^{k+}	Continuous variable specifying the time driver k arrives at node n
τ_n^{k-}	Continuous variable specifying the time driver k departs from node n
λ_n^{ij}	Continuous variable specifying the arrival time of item ij at node n

Appendix B Detailed metrics of learning and optimization processes

We present the detailed performances of each method under both uniformly sampled customers and clustered customers in this section. The learning performance for the uniformly sampled customers is shown in B2, and the re-optimization performance is presented in Table B3. The learning performance for the clustered customers is detailed in Table B4, with the corresponding optimization performance displayed in Table B5.

Table B2: Comparison of learning methods for uniformly sampled customers in the learning process.

Training Scale	Testing Scale	Method	Accuracy	Allocation Percentage	Allocation Std.
$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	0.79	41.67%	0.26
		DANN	0.82	44.02%	0.00
		DCNN	0.82	41.67%	0.18
		GNN	**0.84**	41.67%	0.30
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	NDA	0.88	22.50%	0.39
		DANN	0.82	29.95%	0.59
		DCNN	0.82	22.50%	0.61
		GNN	**0.92**	22.50%	0.70
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	0.78	41.67%	0.94
		DANN	0.81	52.24%	0.28
		DCNN	0.84	41.67%	0.56
		GNN	**0.86**	41.67%	0.35
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	0.85	22.50%	0.02
		DANN	0.78	37.67%	0.42
		DCNN	0.81	22.50%	0.43
		GNN	**0.90**	22.50%	0.09
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	0.71	32.08%	0.97
		DANN	0.69	44.32%	0.50
		DCNN	0.69	32.08%	0.84
		GNN	**0.73**	32.08%	0.40
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	0.83	32.08%	0.07
		DANN	0.81	39.09%	0.22
		DCNN	0.82	32.08%	0.59
		GNN	**0.87**	32.08%	0.15
$\|\mathcal{I}\|$: (2, 10); $\|\mathcal{K}\|$: (2, 5); NoI: 3600	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	0.79	41.67%	0.26
		DANN	0.81	40.24%	0.04
		DCNN	0.81	41.67%	0.09
		GNN	**0.83**	41.67%	0.31
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	NDA	0.88	22.50%	0.39
		DANN	0.89	20.55%	0.40
		DCNN	0.91	22.50%	0.24
		GNN	**0.92**	22.50%	0.43
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	0.78	41.67%	0.81
		DANN	0.82	49.18%	0.26
		DCNN	**0.84**	41.67%	0.30
		GNN	0.82	41.67%	0.70
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	0.85	22.50%	0.02
		DANN	0.85	28.75%	0.34
		DCNN	**0.90**	22.50%	0.04
		GNN	0.89	22.50%	0.05
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	0.71	32.08%	0.20
		DANN	0.69	37.09%	0.04
		DCNN	0.70	32.08%	0.59
		GNN	**0.74**	32.08%	0.55
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	0.83	32.08%	0.06
		DANN	0.84	32.38%	0.15
		DCNN	0.86	32.08%	0.05
		GNN	**0.88**	32.08%	0.15

Table B3: Comparison of learning methods for uniformly sampled customers in the optimization process.

Training Scale	Testing Scale	Method	Objective Gap (%)	Delivery Time Gap (%)	Travel Time Gap (%)	Runtime (s)
$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	3.06	3.06	1.18	7
		DANN	2.97	2.96	4.31	11
		DCNN	**2.07**	2.07	1.67	6
		GNN	3.21	3.2	1.55	6
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	**NDA**	**1.25**	1.25	0.37	6
		DANN	5.53	5.48	43.79	26
		DCNN	10.04	10.01	11.9	10
		GNN	2.73	2.72	1.49	6
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	8.26	8.24	2.77	103
		DANN	6.89	6.89	11.04	200
		DCNN	3.78	3.78	2.19	28
		GNN	**3.19**	3.17	5.3	21
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	2.1	2.14	11.94	133
		DANN	4.2	4.24	23.79	206
		DCNN	12.19	12.28	27.11	86
		GNN	**1.89**	1.92	12.13	33
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	3.49	3.47	3.74	483
		DANN	5.48	5.47	1.84	607
		DCNN	2.29	2.33	18.38	532
		GNN	**1.73**	1.77	17.56	406
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	2.81	2.8	2.32	50
		DANN	4.54	4.56	19.98	92
		DCNN	6.01	6.03	9.9	53
		GNN	**2.78**	2.78	3.8	38
$\|\mathcal{I}\|$: (2, 10); $\|\mathcal{K}\|$: (2, 5); NoI: 3600	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	3.06	3.06	1.18	7
		DANN	2.28	2.28	0.33	9
		DCNN	**2.10**	2.10	1.19	5
		GNN	3.07	3.07	0.77	5
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	NDA	1.25	1.25	0.37	6
		DANN	1.49	1.48	4.36	8
		DCNN	**0.94**	0.94	0.15	5
		GNN	2.19	2.19	3.22	5
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	8.26	8.24	2.77	103
		DANN	4.03	4.01	6.57	154
		DCNN	**1.66**	1.65	4.92	14
		GNN	2.82	2.81	4.46	12
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	2.1	2.14	11.94	133
		DANN	1.89	1.91	7.17	133
		DCNN	**1.66**	1.68	10.42	30
		GNN	2.13	2.15	7.4	25
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	3.49	3.47	3.74	483
		DANN	5.20	5.18	2.99	607
		DCNN	1.63	1.67	17.51	528
		GNN	**1.19**	1.23	16.76	457
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	2.81	2.8	2.32	50
		DANN	2.32	2.31	2.76	67
		DCNN	**1.56**	1.56	2.75	40
		GNN	2.52	2.51	1.39	35

Table B4: Comparison of learning methods for clustered customers in the learning process.

Training Scale	Testing Scale	Method	Accuracy	Allocation Percentage	Allocation Std.
$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3)); NoI: 1200	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	0.64	41.67%	0.30
		DANN	0.73	37.43%	0.01
		DCNN	0.77	41.67%	0.58
		GNN	**0.81**	41.67%	0.19
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	NDA	0.78	22.50%	0.90
		DANN	0.76	23.11%	0.51
		DCNN	0.82	22.50%	0.26
		GNN	**0.89**	22.50%	0.35
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	0.58	41.67%	0.7
		DANN	0.63	63.14%	0.12
		DCNN	0.63	41.67%	0.67
		GNN	**0.70**	41.67%	0.76
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	0.8	22.50%	1.36
		DANN	0.70	27.04%	0.67
		DCNN	0.80	22.50%	0.58
		GNN	**0.86**	22.50%	0.30
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	0.60	32.08%	0.91
		DANN	0.63	68.31%	0.566
		DCNN	0.63	32.08%	0.76
		GNN	**0.69**	32.08%	0.17
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	0.70	32.08%	0.54
		DANN	0.73	33.81%	0.27
		DCNN	0.78	32.08%	0.55
		GNN	**0.83**	32.08%	0.94
$\|\mathcal{I}\|$: (2, 10); $\|\mathcal{K}\|$: (2, 5); NoI: 3600	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	0.64	41.67%	0.30
		DANN	0.72	28.49%	0.51
		DCNN	0.77	41.67%	0.93
		GNN	**0.80**	41.67%	0.57
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	NDA	0.78	22.50%	0.90
		DANN	0.86	17.48%	1.37
		DCNN	0.90	22.50%	0.03
		GNN	**0.90**	22.5%	0.09
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	0.58	41.67%	0.11
		DANN	0.65	38.15%	0.19
		DCNN	0.72	41.67%	0.21
		GNN	**0.73**	41.67%	0.29
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	0.80	18.71%	0.35
		DANN	0.84	18.53%	0.92
		DCNN	0.86	18.71%	0.07
		GNN	**0.88**	18.71%	0.45
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	0.60	32.08%	0.28
		DANN	0.66	31.11%	0.59
		DCNN	0.66	32.08%	0.30
		GNN	**0.69**	32.08%	0.76
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	0.70	32.08%	0.61
		DANN	0.78	29.31%	0.88
		DCNN	0.82	32.08%	0.63
		GNN	**0.84**	32.08%	0.51

Table B5: Comparison of learning methods for clustered customers in the optimization process.

Training Scale	Testing Scale	Method	Objective Gap(%)	Delivery Time Gap(%)	Travel Time Gap(%)	Runtime (s)
$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	3.64	3.58	25.77	14
		DANN	2.94	2.92	8.61	24
		DCNN	3.54	3.53	9.96	10
		GNN	**2.35**	2.33	4.3	7
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	NDA	2.74	2.62	62.36	19
		DANN	3.40	3.28	58.73	43
		DCNN	4.90	4.87	9.08	13
		GNN	**1.79**	1.73	24.09	8
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	10.71	10.69	12.31	152
		DANN	11.72	11.69	20.47	311
		DCNN	4.35	4.30	14.55	82
		GNN	**2.90**	2.89	3.57	15
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	5.72	5.63	39.31	127
		DANN	4.21	4.11	41.43	264
		DCNN	6.46	6.38	23.95	98
		GNN	**1.20**	1.18	7.55	31
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	11.62	11.57	3.4	574
		DANN	10.62	10.59	12.5	607
		DCNN	4.06	4.00	17.51	542
		GNN	**2.36**	2.30	16.76	457
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	4.65	4.58	36.19	47
		DANN	4.53	4.48	29.79	109
		DCNN	4.39	4.36	12.09	55
		GNN	**2.07**	2.07	9.01	38
$\|\mathcal{I}\|$: (2, 10); $\|\mathcal{K}\|$: (2, 5); NoI: 3600	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (2, 3); NoI: 1200	NDA	3.64	3.58	25.77	14
		DANN	**1.88**	1.86	2.08	41
		DCNN	4.56	4.55	9.97	7
		GNN	2.65	2.64	3.17	6
	$\|\mathcal{I}\|$: (2, 7); $\|\mathcal{K}\|$: (4, 5); NoI: 1200	NDA	2.74	2.62	62.36	19
		DANN	**0.23**	0.23	4.34	50
		DCNN	2.75	2.72	4.14	6
		GNN	1.68	1.64	23.44	5
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (2, 3); NoI: 600	NDA	10.71	10.69	12.31	152
		DANN	6.22	6.22	5.96	329
		DCNN	5.18	5.15	11.74	83
		GNN	**1.16**	1.12	11.79	12
	$\|\mathcal{I}\|$: (8, 10); $\|\mathcal{K}\|$: (4, 5); NoI: 600	NDA	5.72	5.63	39.31	127
		DANN	1.18	1.13	12.16	163
		DCNN	2.99	2.97	11.53	35
		GNN	**1.01**	0.97	21.48	24
	$\|\mathcal{I}\|$: (12, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 400	NDA	11.62	11.57	3.40	574
		DANN	6.52	6.50	10.02	607
		DCNN	2.08	2.03	17.51	525
		GNN	**1.74**	1.69	16.76	450
	$\|\mathcal{I}\|$: (2, 20); $\|\mathcal{K}\|$: (2, 5); NoI: 4000	NDA	4.65	4.58	36.19	47
		DANN	**1.93**	1.90	1.47	111
		DCNN	3.60	3.57	8.89	46
		GNN	1.96	1.95	11.10	34

Appendix C Dynamic experiments

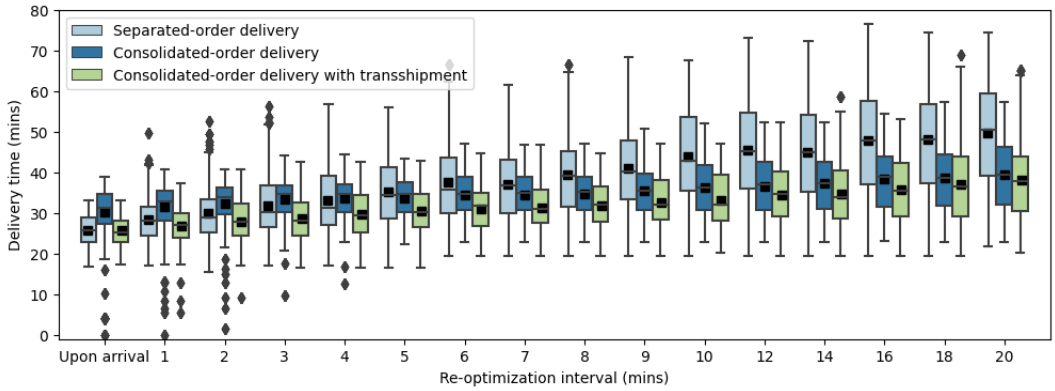
C.1 Dynamic experiments with earliest-available-driver-assignment strategy

The detailed experimentation process for earliest-available-driver-assignment strategy is shown in Table C6.

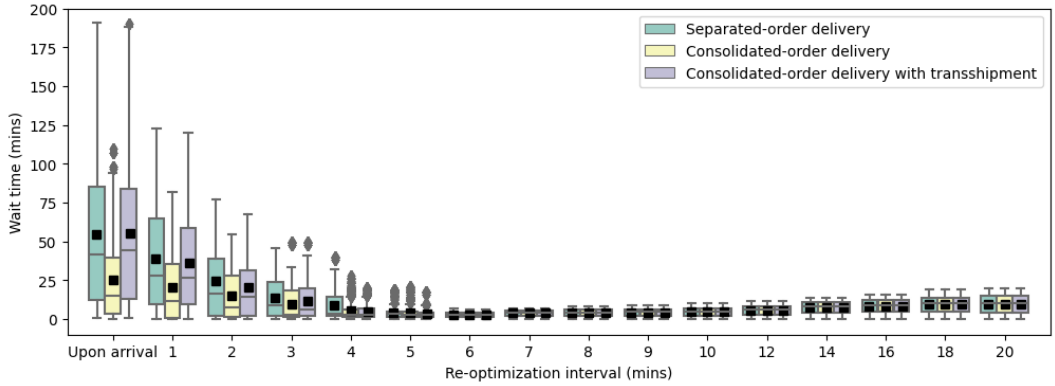
Table C6: Dynamic Experimentation Process.

Step	Description
Initialization	Initialize the experimentation with the following elements: • Define the customer arrival process as a Poisson process with a specified arrival rate. Set service duration $\|\mathcal{T}\|$ to 1 or 2 hours. Generate customer arrival times a_i for each customer $i \in \mathcal{I}$ who arrives within this duration. • Let $\|\mathcal{K}\|$ available drivers serve customers who order from store $j \in \|\mathcal{J}\|$. For driver $k \in \mathcal{K}$, generate their origin node and set availability time $\tau_{o_k} = 0$. Additionally, determine the locations of both customers and stores. • Define the re-optimization intervals, which can vary from event-triggered (e.g., upon a new customer arrival) to a fixed interval (e.g., 10 minutes). Obtain the set of optimization time points $\{p_0, p_1, p_2, \ldots, p_T\}$, and define the final time P for the experimentation.
Step 1	For the initial period $t = 0$, corresponding to the time interval $[p_0, p_1]$, optimize the system at the time point p_1, • Input the information, including the customers that arrive within this period (i.e., $i \in \mathcal{I}_0$ where $\mathcal{I}_0 = \{i \in \mathcal{I} \| p_0 \leq a_i \leq p_1\}$) and place orders from stores $j \in \mathcal{J}$, as well as the drivers $k \in \mathcal{K}$ with their available times to serve customers being $\tau_{o_k} = 0$. • Run the optimization problem to assign drivers to customers and plan their routes. If no drivers are currently available, customers are assigned to the earliest available drivers who can serve the orders once they complete their assigned tasks. • Update the drivers' availability times based on the completion time of their last served customer (i.e., $\tau_{o_k} \leftarrow \max_{i \in I_0} \left\{\tau_i^{k-}\right\}$), and set the driver location to the last served customer location. • Record the number of customers served in the interval $[p_0, p_1]$, as well as the completion time, wait time, delivery time, and expected travel time for each customer.
Step 2	While $t \leq \|\mathcal{T}\| - 1$, repeat the following steps: Increment t and update the time interval to $[p_t, p_{t+1}]$. Optimize the system at the time point p_{t+1}, • Input the information, including the customers that arrive within this period (i.e., $i \in \mathcal{I}_t$ where $\mathcal{I}_t = \{i \in \mathcal{I} \mid p_t \leq a_i \leq p_{t+1}\}$) and place orders from stores $j \in \mathcal{J}$, as well as the drivers $k \in \mathcal{K}$ with their earliest available times to serve customers being τ_{o_k}. • Run the optimization problem to assign drivers to customers and plan their routes. If no drivers are currently available, customers are assigned to the earliest available drivers who can serve the orders once they complete their assigned tasks. • Update the drivers' availability times based on the completion time of their last served customer (i.e., $\tau_{o_k} \leftarrow \max_{i \in I_t} \{\tau_i^{k-}\}$), and set the driver location to the last served customer location. • Record the number of customers served in the interval $[p_t, p_{t+1}]$, as well as the completion time, wait time, delivery time, and expected travel time for each customer.
Output	The experimentation outputs include the total number of customers served during each time interval, and for each customer, the order completion time, wait time, delivery time, and expected travel time.

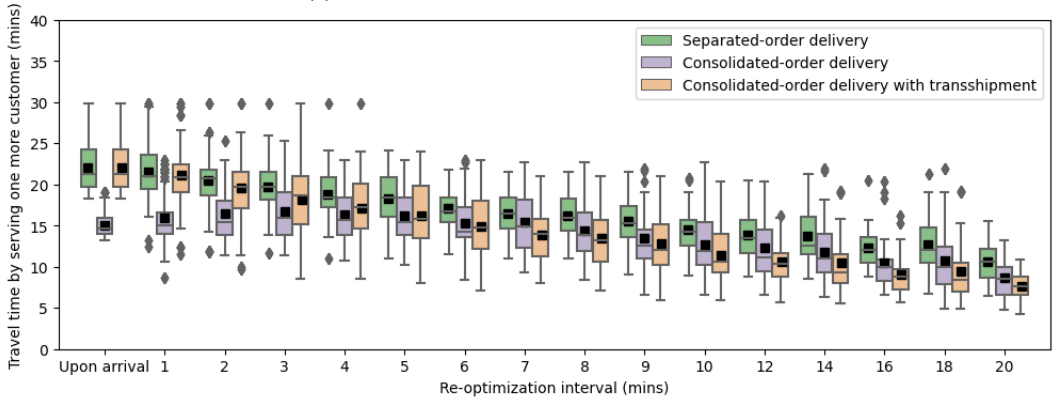
The dynamic experimentation results for earliest-available-driver-assignment strategy, including delivery time, wait time, travel time per customer, and customer scale under varying re-optimization intervals, are shown in Figure C1.



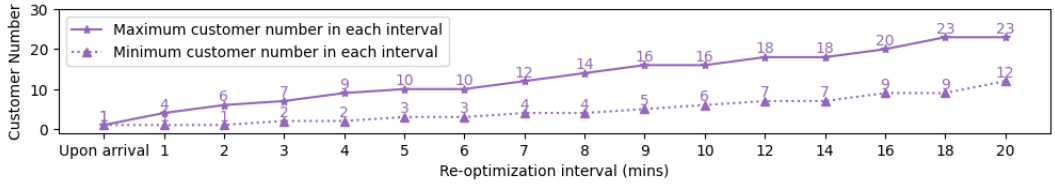
(a) Delivery time between order pickup and arrival.



(b) Wait time between order placement and pickup.



(c) Driver travel time by serving one more customer.



(d) Number of customers in each re-optimization interval.

Figure C1: Delivery time, wait time, travel time, and customer number under varying re-optimization intervals.

C.2 Dynamic experiments with driver-availability-triggered strategy

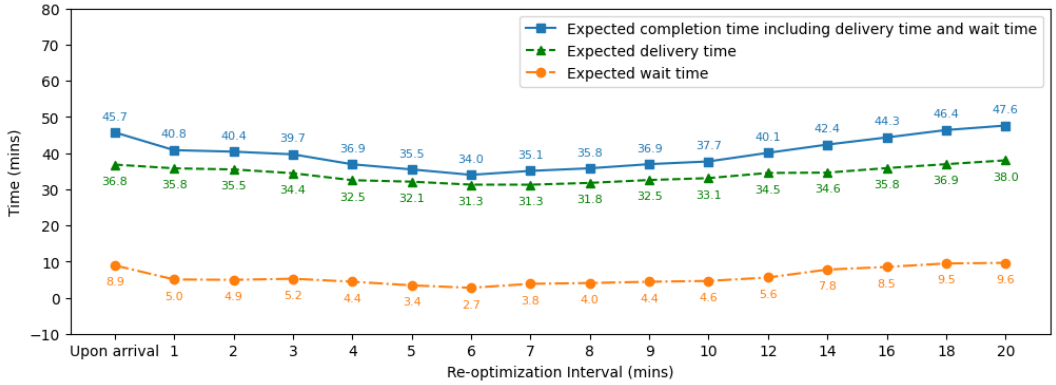
We present the dynamic experimentation process with driver-availability-triggered optimization in Table C7.

Table C7: Dynamic experimentation process with driver-availability-triggered optimization.

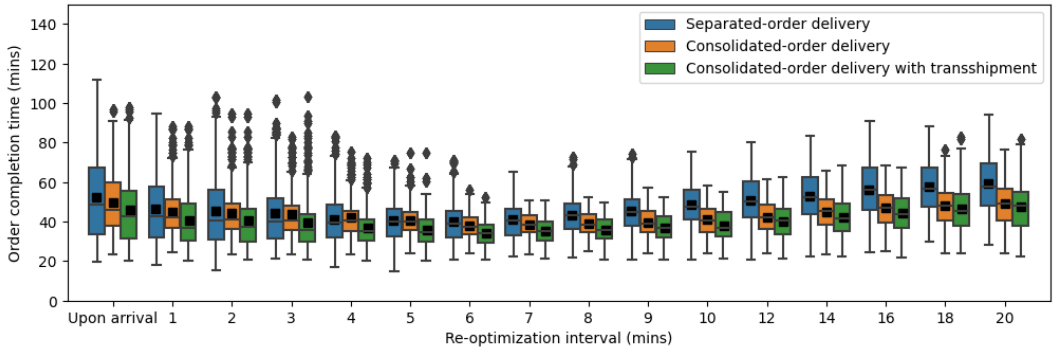
Step	Description
Initialization	Initialize the experimentation with the following elements: • Define the customer arrival process as a Poisson process with a specified arrival rate. Set service duration $\|\mathcal{T}\|$ to 1 or 2 hours. Generate customer arrival times a_i for each customer $i \in \mathcal{I}$ who arrives within this duration. • Let $\|\mathcal{K}\|$ available drivers serve customers who order from store $j \in \|\mathcal{J}\|$. For driver $k \in \mathcal{K}$, generate their origin node and set availability time $\tau_{o_k} = 0$. Additionally, determine the locations of both customers and stores. • Define the re-optimization intervals, which can vary from event-triggered (e.g., upon a new customer arrival) to a fixed interval (e.g., 10 minutes). Obtain the set of optimization time points $\{p_0, p_1, p_2, \ldots, p_T\}$, and define the final time P for the experimentation.
Step 1	For the initial period $t = 0$, corresponding to the time interval $[p_0, p_1]$, if there are available drivers $k \in \{k \in \mathcal{K} \| o_k \leq p_1\}$, optimize the system at the time point p_1, • Input the information, including the customers that arrive within this period (i.e., $i \in \mathcal{I}_0$ where $\mathcal{I}_0 = \{i \in \mathcal{I} \| p_0 \leq a_i \leq p_1\}$) and place orders from stores $j \in \mathcal{J}$, as well as the drivers $k \in \mathcal{K}$ with their available times to serve customers being $\tau_{o_k} = 0$. • Run the optimization problem to assign available drivers to customers and plan their routes. • Update the drivers' availability times based on the completion time of their last served customer (i.e., $\tau_{o_k} \leftarrow \max_{i \in I_0} \left\{\tau_i^{k-}\right\}$), and set the driver location to the last served customer location. • Record the number of customers served in the interval $[p_0, p_1]$, as well as the completion time, wait time, delivery time, and expected travel time for each customer.
Step 2	While $t \leq \|\mathcal{T}\| - 1$, repeat the following steps: Increment t and update the time interval to $[p_t, p_{t+1}]$. If there are available drivers $k \in \{k \in \mathcal{K} \| o_k \leq p_{t+1}\}$, optimize the system at the time point p_{t+1}, • Input the information, including the customers that arrive within this period (i.e., $i \in \mathcal{I}_t$ where $\mathcal{I}_t = \{i \in \mathcal{I} \mid p_t \leq a_i \leq p_{t+1}\}$) and place orders from stores $j \in \mathcal{J}$, as well as the drivers $k \in \mathcal{K}$ with their earliest available times to serve customers being τ_{o_k}. • Run the optimization problem to assign available drivers to customers and plan their routes. • Update the drivers' availability times based on the completion time of their last served customer (i.e., $\tau_{o_k} \leftarrow \max_{i \in I_t} \{\tau_i^{k-}\}$), and set the driver location to the last served customer location. • Record the number of customers served in the interval $[p_t, p_{t+1}]$, as well as the completion time, wait time, delivery time, and expected travel time for each customer.
Output	The experimentation outputs include the total number of customers served during each time interval, and for each customer, the order completion time, wait time, delivery time, and expected travel time.

Under the dynamic experimentation with the driver-availability-triggered strategy, Figure C2a presents the order completion time, including delivery and wait times, for consolidated-order delivery with transshipment (CODT). Although the trade-off between delivery and wait times disappears under this strategy, the order completion time still follows a smooth pattern, with trends that initially decrease and then increase. The five-minute re-optimization interval yields the lowest delivery time, while the six-minute interval achieves the lowest wait time and overall order completion times. Figure C2b compares completion times across the three systems under varying intervals, while Figure C3

provides detailed distributions of delivery time, wait time, travel time per customer, and customer scale. CODT consistently outperforms the other systems in order completion time, delivery time, and driver travel time per customer. Figure C4 highlights the best re-optimization interval that minimizes order completion time across varying customer arrival rates and customer-to-driver ratios. As arrival rates or customer-to-driver ratios increase, longer re-optimization intervals are optimal, reflecting the benefit of waiting longer to batch and fulfill orders. These findings align with insights from the earliest-available-driver-assignment strategy.



(a) Completion time, wait time, and delivery time for consolidated-order delivery with transshipment.



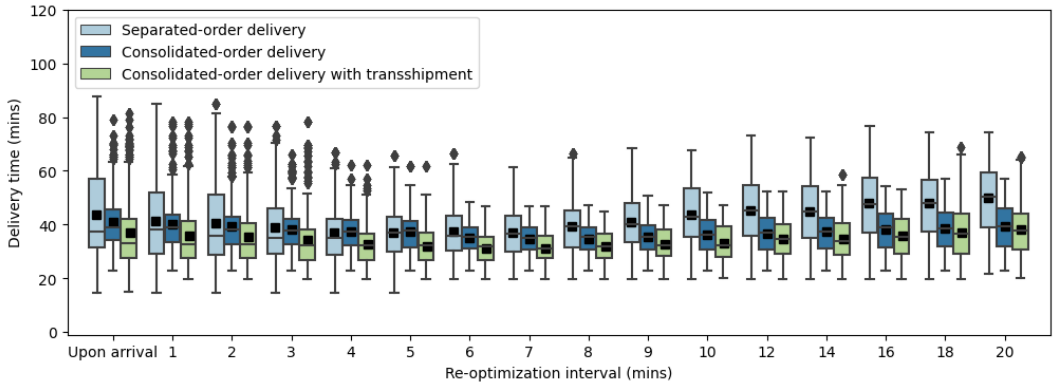
(b) Completion time including wait time and delivery time.

Figure C2: Completion time under varying re-optimization intervals for the driver-availability-triggered strategy.

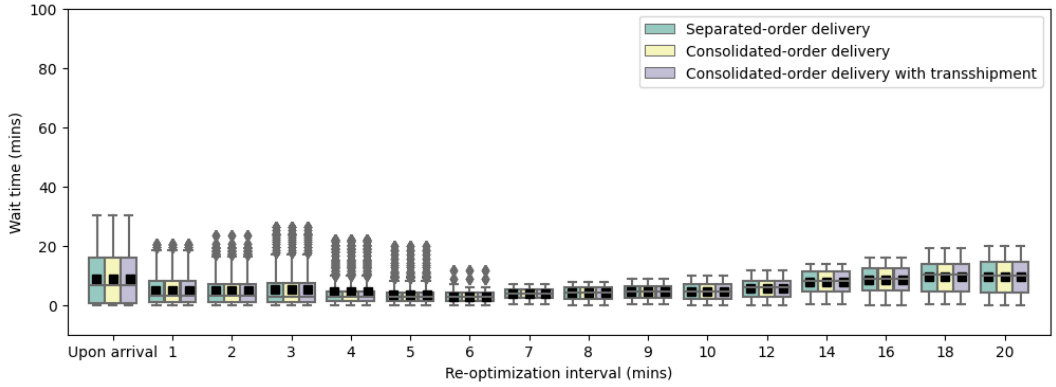
C.3 Comparison between earliest-available-driver-assignment strategy and driver-availability-triggered strategy

We compare the earliest-available-driver-assignment strategy with the driver-availability-triggered strategy in this section. As shown in Figures 8a and C2a, the best completion time under the driver-availability-triggered strategy, at 34.0 minutes, is higher than the best time of 33.6 minutes under the earliest-available-driver-assignment strategy. Therefore, we conclude that fixed-interval optimization with the earliest-available-driver-assignment strategy is slightly superior to the driver-availability-triggered strategy.

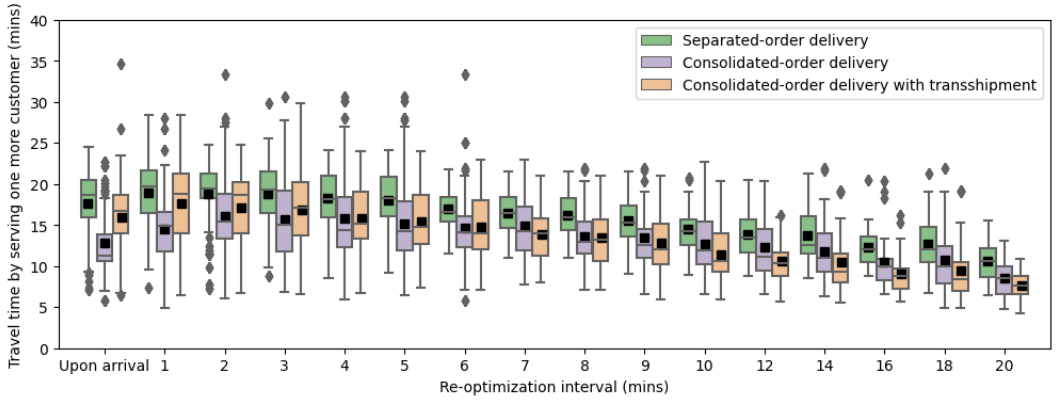
Table C8 presents the best re-optimization intervals and corresponding minimum completion times across different customer arrival rates and customer-to-driver ratios, highlighting how the optimal waiting strategy varies. Overall, the earliest-available-driver-assignment strategy consistently outperforms the driver-availability-triggered strategy, though the gap is small. The best re-optimization interval for the driver-availability-triggered strategy is approximately one minute longer, as both shorter and longer intervals can result in larger batching sizes and longer completion times in this approach. In summary, a shorter-duration waiting strategy is optimal for the consolidated-order delivery system with transshipment.



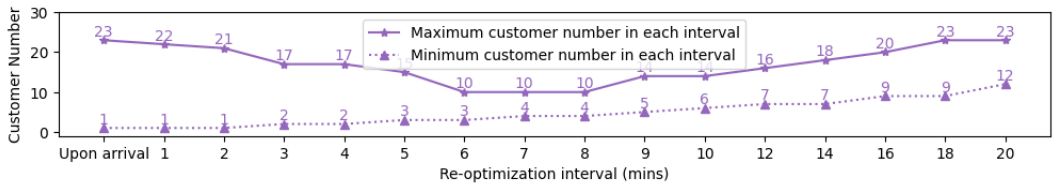
(a) Delivery time between order pickup and arrival.



(b) Wait time between order placement and pickup.

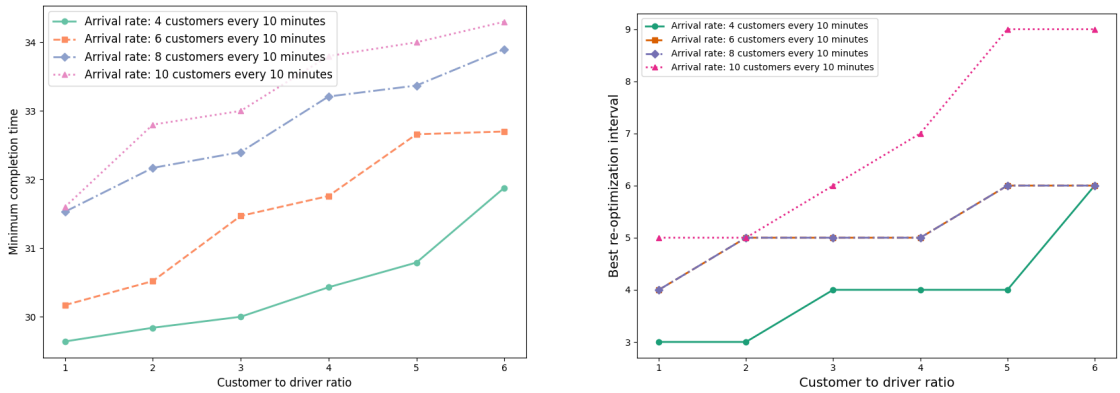


(c) Driver travel time by serving one more customer.



(d) Number of customers in each re-optimization interval.

Figure C3: Delivery time, wait time, travel time, and customer number under varying re-optimization intervals for the driver-availability-triggered strategy.



(a) Minimum completion time under the best re-optimization interval. **(b) Best re-optimization interval.**

Figure C4: Best re-optimization interval for varying ratios of customer number to driver number.

Table C8: Comparison between earliest-available-driver-assignment strategy and driver-availability-triggered strategy.

Customer to Driver Ratio	Arrival Rate (Num. of customers every 10 mins)	Earliest-available-driver-assignment Strategy		Driver-availability-triggered Strategy	
		Best Re-optimization Interval (mins)	Minimum Completion Time (mins)	Best Re-optimization Interval (mins)	Minimum Completion Time (mins)
1	4	3	29.40	3	29.64
1	6	4	30.14	4	30.17
1	8	4	30.99	4	31.53
1	10	4	31.42	5	31.60
2	4	3	29.45	3	29.84
2	6	4	30.44	5	30.52
2	8	4	31.61	5	32.17
2	10	6	32.61	5	32.80
3	4	4	30.04	4	30.00
3	6	5	31.19	5	31.47
3	8	5	32.63	5	32.40
3	10	6	32.85	6	33.00
4	4	4	30.60	4	30.43
4	6	5	31.58	5	31.76
4	8	5	33.00	5	33.21
4	10	6	33.20	7	33.80
5	4	4	30.67	4	30.79
5	6	5	32.04	6	32.66
5	8	5	33.27	6	33.37
5	10	9	33.62	9	34.00
6	4	4	31.29	6	31.88
6	6	6	32.30	6	32.70
6	8	6	33.88	6	33.90
6	10	9	34.30	9	34.30

References

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

Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. European Journal of Operational Research, 202(1):8–15, 2010.

Junyu Cao and Wei Qi. Stall economy: The value of mobility in retail on wheels. Operations Research, 71(2): 708–726, 2023.

John Gunnar Carlsson, Sheng Liu, Nooshin Salari, and Han Yu. Provably good region partitioning for on-time last-mile delivery. Operations Research, 72(1):91–109, 2024.

George B Dantzig and John H Ramser. The truck dispatching problem. Management Science, 6(1):80–91, 1959.

DoubleDash. Introducing doubledash, a new way to shop multiple stores in one order, 2023. URL https://about.doordash.com/en-us/news/introducing-doubledash-a-new-way-to-shop-multiple-stores-in-one-order. Last accessed on Nov 01, 2023.

Epipresto. The place to order from all your specialized and independent stores!, 2023. URL https://epipresto.ca/en. Last accessed on Nov 01, 2023.

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

L Jeff Hong, Zhiyuan Huang, and Henry Lam. Learning-based robust optimization: Procedures and statistical guarantees. Management Science, 67(6):3447–3467, 2021.

Instacart. Introducing orderup, a new way to save time and money on instacart with orders from two retailers–with just one delivery fee, 2022. URL https://www.instacart.com/company/updates/introducing-orderup-a-new-way-to-save-time-and-money-on-instacart-with-orders-from-two-retailers-with-just-one-delivery-fee/. Last accessed on Nov 01, 2023.

Esther Julien, Krzysztof Postek, and Ş İlker Birbil. Machine learning for k-adaptability in two-stage robust optimization. INFORMS Journal on Computing, Forthcoming, 2024. URL https://doi.org/10.1287/ijoc.2022.0314.

Çağrı Koç, Gilbert Laporte, and İlknur Tükenmez. A review of vehicle routing with simultaneous pickup and delivery. Computers & Operations Research, 122:104987, 2020.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In International Joint Conference on Artificial Intelligence, pages 4475–4482. IJCAI, 8 2021.

James Kotary, Vincenzo Di Vito, Jacob Christopher, Pascal Van Hentenryck, and Ferdinando Fioretto. Learning joint models of prediction and optimization. In ECAI 2024, pages 2476–2483. IOS Press, 2024.

Eric Larsen, Emma Frejinger, Bernard Gendron, and Andrea Lodi. Fast continuous and integer l-shaped heuristics through supervised learning. INFORMS Journal on Computing, 36(1):203–223, 2024.

Zhuoxin Li and Gang Wang. On-demand delivery platforms and restaurant sales. Management Science, Forthcoming, 2024. URL https://doi.org/10.1287/mnsc.2021.01010.

Sheng Liu, Long He, and Zuo-Jun Max Shen. On-time last-mile delivery: Order assignment with travel-time predictors. Management Science, 67(7):4095–4119, 2021.

Zefeng Lyu and Andrew Junfang Yu. The pickup and delivery problem with transshipments: Critical review of two existing models and a new formulation. European Journal of Operational Research, 305(1):260–270, 2023.

Wenzheng Mao, Liu Ming, Ying Rong, Christopher S Tang, and Huan Zheng. On-demand meal delivery platforms: Operational level data and research opportunities. Manufacturing & Service Operations Management, 24(5):2535–2542, 2022.

Donato Maragno, Holly Wiberg, Dimitris Bertsimas, Ş İlker Birbil, Dick den Hertog, and Adejuyigbe O Fajemisin. Mixed-integer optimization with constraint learning. Operations Research, Forthcoming, 2023. URL https://doi.org/10.1287/opre.2021.0707.

Daniel Merchan, Julian Pachon, Jatin Arora, Karthik Konduri, Matthias Winkenbach, Steven Parks, and Joseph Noszek. Amazon last mile routing research challenge dataset, 2021. URL https://registry.opendata.aws/amazon-last-mile-challenges. Accessed January 6, 2022.

Snežana Mitrović-Minić and Gilbert Laporte. The pickup and delivery problem with time windows and transshipment. INFOR: Information Systems and Operational Research, 44(3):217–227, 2006.

Maciek Nowak, Ozlem Ergun, and Chelsea C White III. An empirical study on the benefit of split loads with the pickup and delivery problem. European Journal of Operational Research, 198(3):734–740, 2009.

Ritesh Ojha, Wenbo Chen, Hanyu Zhang, Reem Khir, Alan Erera, and Pascal Van Hentenryck. Optimization-based learning for dynamic load planning in trucking service networks. arXiv preprint arXiv:2307.04050, 2023.

Sami Serkan Özarık, Paulo da Costa, and Alexandre M Florio. Machine learning for data-driven last-mile delivery optimization. Transportation Science, 58(1):27–44, 2024.

Meng Qi, Yuanyuan Shi, Yongzhi Qi, Chenxin Ma, Rong Yuan, Di Wu, and Zuo-Jun Shen. A practical end-to-end inventory management model with deep learning. Management Science, 69(2):759–773, 2023.

S Raghavan and Rui Zhang. The driver-aide problem: Coordinated logistics for last-mile delivery. Manufacturing & Service Operations Management, 26(1):291–311, 2024.

Abdur Rais, Filipe Alvelos, and Maria Sameiro Carvalho. New mixed integer-programming model for the pickup-and-delivery problem with transshipment. European Journal of Operational Research, 235(3): 530–539, 2014.

Julia Rieck, Carsten Ehrenberg, and Jürgen Zimmermann. Many-to-many location-routing with inter-hub transport and multi-commodity pickup-and-delivery. European Journal of Operational Research, 236(3): 863–878, 2014.

Utsav Sadana, Abhilash Chenreddy, Erick Delage, Alexandre Forel, Emma Frejinger, and Thibaut Vidal. A survey of contextual optimization methods for decision-making under uncertainty. European Journal of Operational Research, 2024.

Martin Savelsbergh and Tom Van Woensel. 50th anniversary invited article—city logistics: Challenges and opportunities. Transportation Science, 50(2):579–590, 2016.

Statista. Market insights into quick commerce of online food and grocery delivery (worldwide), 2024. URL https://www.statista.com/outlook/dmo/online-food-delivery/worldwide. Last accessed on Oct 15, 2024.

E Su, Hu Qin, Jiliu Li, and Kai Pan. An exact algorithm for the pickup and delivery problem with crowdsourced bids and transshipment. Transportation Research Part B: Methodological, 177:102831, 2023.

Pascal Van Hentenryck. Machine learning for optimal power flows. Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications, pages 62–82, 2021.

Shanshan Wang, Erick Delage, and Leandro C Coelho. Data-driven stochastic vehicle routing problems with dead-lines. Les Cahiers du GERAD ISSN, 711:2440, 2024.

Wei Zhang, Alexandre Jacquillat, Kai Wang, and Shuaian Wang. Routing optimization with vehicle–customer coordination. Management Science, 69(11):6876–6897, 2023.