Appendix A: Courier empirical rejection

To obtain a more realistic courier behavioral rule, we fit a logistic regression model on the data provided by INFORMS (2024). The logistic model estimates the probability that a courier-to-order assignment is rejected by the courier. The result of the logistic regression, which can be reproduced from our online repository, is given in Table 5. The logistic regression results show that all included variables are statistically significant predictors of order rejection ($p < 0.001$). Courier workload has the strongest influence: having more accepted orders substantially decreases the likelihood of rejecting an order, whereas having more outstanding invitations greatly increases it. Naturally, every picked-up order is also accepted so taken together, rejection probability tends to increase with increased (accepted) orders to deliver. Spatial factors also matter, with longer courier-to-merchant and courier-to-customer distances increasing rejection probability, while longer customer-to-merchant distances are associated with a lower likelihood of rejection after controlling for other variables. Timing variables have smaller but significant effects, indicating that orders with more time until the promised delivery are more likely to be rejected, whereas having more time remaining until the end of the courier's shift slightly reduces rejection. Overall, the model achieves a McFadden pseudo-(R^2) of 0.44, suggesting good explanatory power for predicting order rejection.

Table 5 Logistic regression predicting order rejection as a function of order-level and courier-level features.

Variable	Estimate	STD. error	z-value	Pr(>|z|)
Intercept	1.135	0.014	79.557	$< 0.001^{***}$
$\lvert O^c_{\text{accepted}} \rvert$	-3.438	0.011	-316.665	$< 0.001^{***}$
$\lvert O^c_{\text{inv}} \rvert$	3.513	0.013	279.171	$< 0.001^{***}$
Courier-merchant distance	0.205	0.005	37.595	$< 0.001^{***}$
Courier-customer distance	0.025	0.005	4.927	$< 0.001^{***}$
Customer-merchant distance	-0.088	0.007	-12.256	$< 0.001^{***}$
Time (s) until promise delivery	3.621e-04	1.228e-05	29.484	$< 0.001^{***}$
Time (s) until end of shift	-1.365e-07	2.565e-08	-5.319	$< 0.001^{***}$

Notes: Estimate column reports the log-odds. Standard errors are reported in the second column. All coefficients are statistically significant at the 0.001 level. Distances are in Manhattan kilometers. The pseudo-R^2 (McFadden) equals 0.44. *p<0.05; **p<0.01; ***p<0.001.

Appendix B: Example illustrating impact of implicit future assignment costs

For an illustrative example of the parameterized courier-to-order assignment problem (1) and the workings of the CFA framework, see Figure 6. At this decision moment, there are three orders $1, 2, 3$ and two couriers 1 and 2 with number of previously assigned orders $I^1 = 0$ and $I^2 = 1$, respectively. In this example, $B = 1$ and we assume that R and V are sufficiently high such that they do not restrict the assignments. Then the direct-cost minimizing assignment (i.e., $f_{ij} \equiv 0$) would have an objective of $d_{11} + d_{22} + d_{03} = 2 + 5 + R$, with order 3 postponed (assigned to a dummy courier).

Imagine that positioning courier 2 towards the vicinity of merchant 1 is extremely valuable for future assignment, for example, because future orders are expected to appear there. In that case, the direct cost of $d_{21} = 15$ can be effectively reduced by an amount, say $f_{21} = -13$. In turn, the total cost-minimizing objective becomes $d_{03} + d_{12} + (d_{21} + f_{21}) = R + 4 + (15 - 13)$, with a solution that assigns courier 1 to order 2, which could be advantageous in the long run. A reverse scenario could be that the merchant of order 1 is very popular, and couriers often start their shift there. Then assigning couriers to that merchant can be disadvantageous because more couriers are expected to appear there soon,

so setting $f_{11}, f_{22} > 0$ and/or $f_{01} < 0$ could be valuable. Similarly, postponement can be stimulated by setting f_{0j} low enough for as long as the platform wishes to postpone the order indexed by j.

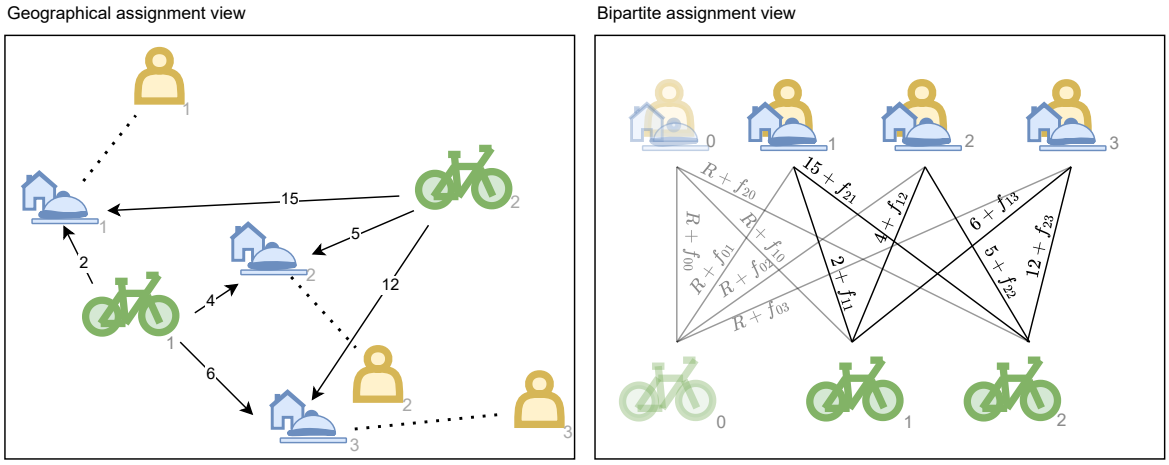


Figure 6 Example of courier-to-order assignment problem at decision moment $t \in \mathcal{T}$.

Note. On the left: top-down geographic view with direct costs indicating the cheapest insertion distances. On the right: bipartite assignment view including dummy order/courier (indexed by zeros), and direct and implicit future costs.

Appendix C: GNN Actor/Critic architecture

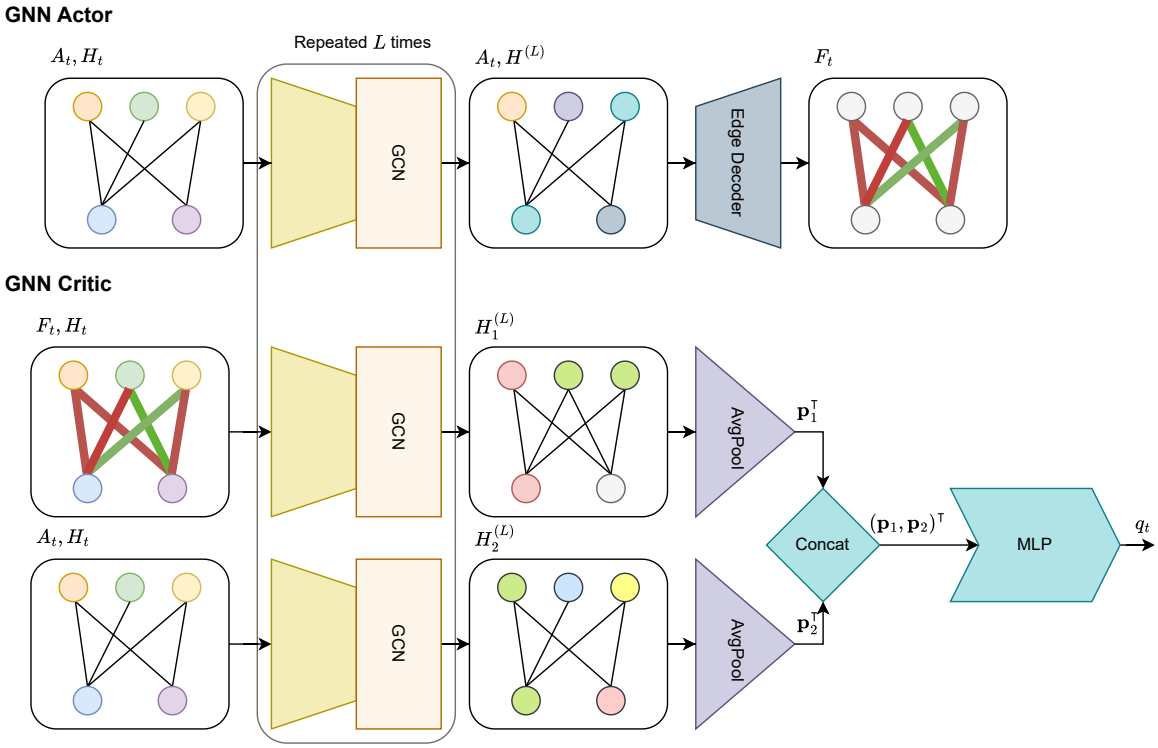


Figure 7 Architecture diagram of GNN Actor and GNN critic networks.

Note. The GNN actor takes as input the bipartite assignment graph with node features H_t (indicated with color) and adjacency matrix A_t (see edges). It non-linearly transforms the node-features L times until $H^{(L)}$ is obtained. The edge decoder then transforms the node features into implicit future costs (F_t) per edge. The GNN critic first processes the estimated implicit future costs by the actor and the state in parallel. The resulting node features are averaged across the nodes and concatenated. An MLP transforms the concatenated vector into a single scalar quality (q_t). Best viewed in color.

Appendix D: TD3 algorithm

We find the weights of our actor and critic models, using the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm (Fujimoto, van Hoof, and Meger 2018). This algorithm, outlined in Algorithm 1, iteratively updates the actor and critic network weights over many *episodes*, where each episode consists of multiple simulated days of the MDRP.

Because overestimation of the quality function by the critic can be rapidly exploited by the actor, actor-critic methods can be quite unstable to train. To circumvent this, the TD3 algorithm, applies five tricks to stabilize and speed up convergence of training. First, to reduce overestimation of the quality value by the critic, TD3 uses two critic GNNs and at every decision moment only the smallest predicted quality value is used in the temporal difference error. Second, copies of the actor and critic networks, called *target* GNNs, are created and the actor/critic weights are slowly lagged behind the actual networks, further stabilizing training. Third, instead of only learning from the last decision, a replay experience buffer stores 'memories' of past states, implicit future costs, and costs to provide the networks with a more

diverse set of learning experiences. Fourth, the update frequency of the actor weights is delayed compared to the critic's update frequency. For example, we update the critics twice as often as the actor. Fifth, to encourage exploration and simultaneously make it harder for the actor to precisely target weaknesses in the critic quality prediction, some small amount of random noise is added to the output predicted by the actor.

For the TD3 algorithm training, we use the following settings. We randomly initialize all our actor and critic trainable weights using $\mathcal{N}(0, 0.1)$. The experience replay buffer contains $|\mathcal{B}| = 2048$ experiences and every 32 decision moments we sample $N = 16$ experiences to train our networks. We use the Adam optimizer with default parameters to update the actor and critic weights on the sampled experiences from the replay buffer (Kingma and Ba 2014). Furthermore, we update our critics' weight twice as frequently as our actor weights. We add mean-centered Gaussian exploration noise $\mathcal{N}_t(0, 0.1)$ to each implicit future cost estimated by the actor. We set the target networks' update parameter equal to $\tau = 0.001$.

Appendix E: Order release time

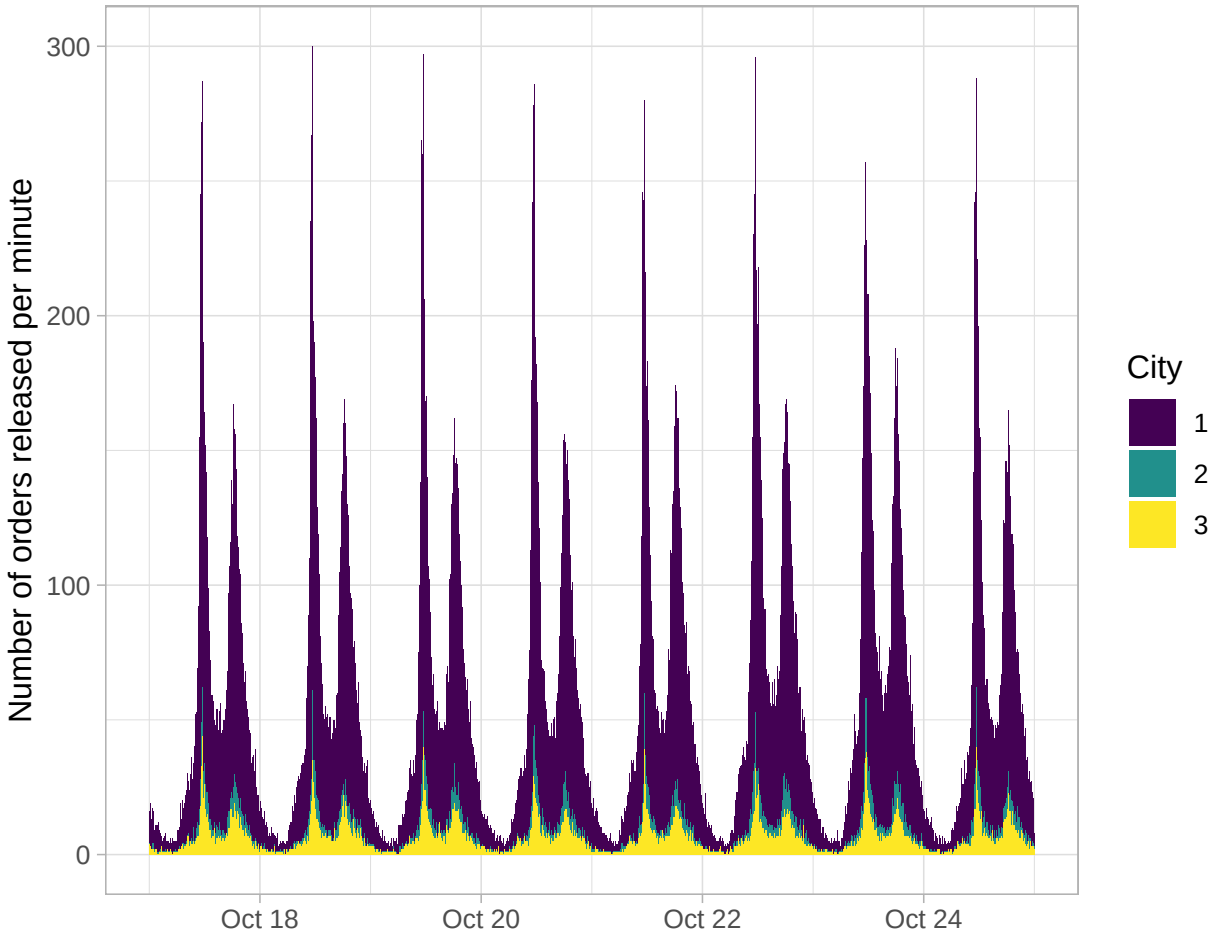


Figure 8 Order release time.

Note. The dinner peak and lunch peak are clearly distinguishable.

Algorithm 1 TD3 algorithm with GNN

Randomly initialize weights of actor θ, and critic networks ω_1, ω_2

Randomly initialize weights of target networks θ' and ω'_1, ω'_2

Initialize experience replay buffer $\mathcal{B}$

for each episode = $1, 2, \ldots,$ max episode **do**

Initialize DES, load in orders and shifts events from the *calibration dataset*

Initialize $S_0 = (A_0, H_0)$

Estimate implicit future costs $F_t = \mu_\theta(A_t, H_t) + \mathcal{N}_t$ with exploration noise $\mathcal{N}_t$

Solve the courier-to-order assignment problem given F_t, implement assignment decision x_t

Simulate state transition until next decision moment, $t \rightarrow t + \Delta t$

Observe next state $S_{t+\Delta t} = (A_{t+\Delta t}, H_{t+\Delta t})$ and cost R_t

Store the transition $(S_t, F_t, R_t, S_{t+\Delta t})$ in the replay buffer $\mathcal{B}$

if time to update **then**

Sample a random batch $\bar{\mathcal{B}}$ of N transitions $(S_t, F_t, R_t, S_{t+\Delta t})$ from $\mathcal{B}$

for each transition i in $\bar{\mathcal{B}}$ **do**

Calculate the shared temporal difference target:

$\alpha_i = R_i + \gamma \min_{j=1,2} Q_{\omega'_j}(A_{i+\Delta t}, H_{i+\Delta t} \mu_{\theta'}(S_{i+\Delta t}))$ ▷ Set $\alpha_i = R_i$ if $S_{t+\Delta t}$ terminal

end for

Update critic network weights by minimizing the critic batch loss using Adam:

$L(\omega_j) = \frac{1}{N} \sum_{i=1}^{N} \left(\alpha_i - Q_{\omega_j}(A_i, H_i, F_i)\right)^2$, for $j = 1, 2$.

if time to update actor **then**

Update actor network weights by maximizing the random batch quality using Adam:

$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} Q_1(A_i, H_i, \mu_\theta(A_i, H_i))$

Update target networks weights:

$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$

$\omega'_j \leftarrow \tau\omega_j + (1 - \tau)\omega'_j$, for $j = 1, 2$.

end if

end if

end for

Appendix F: Order topology

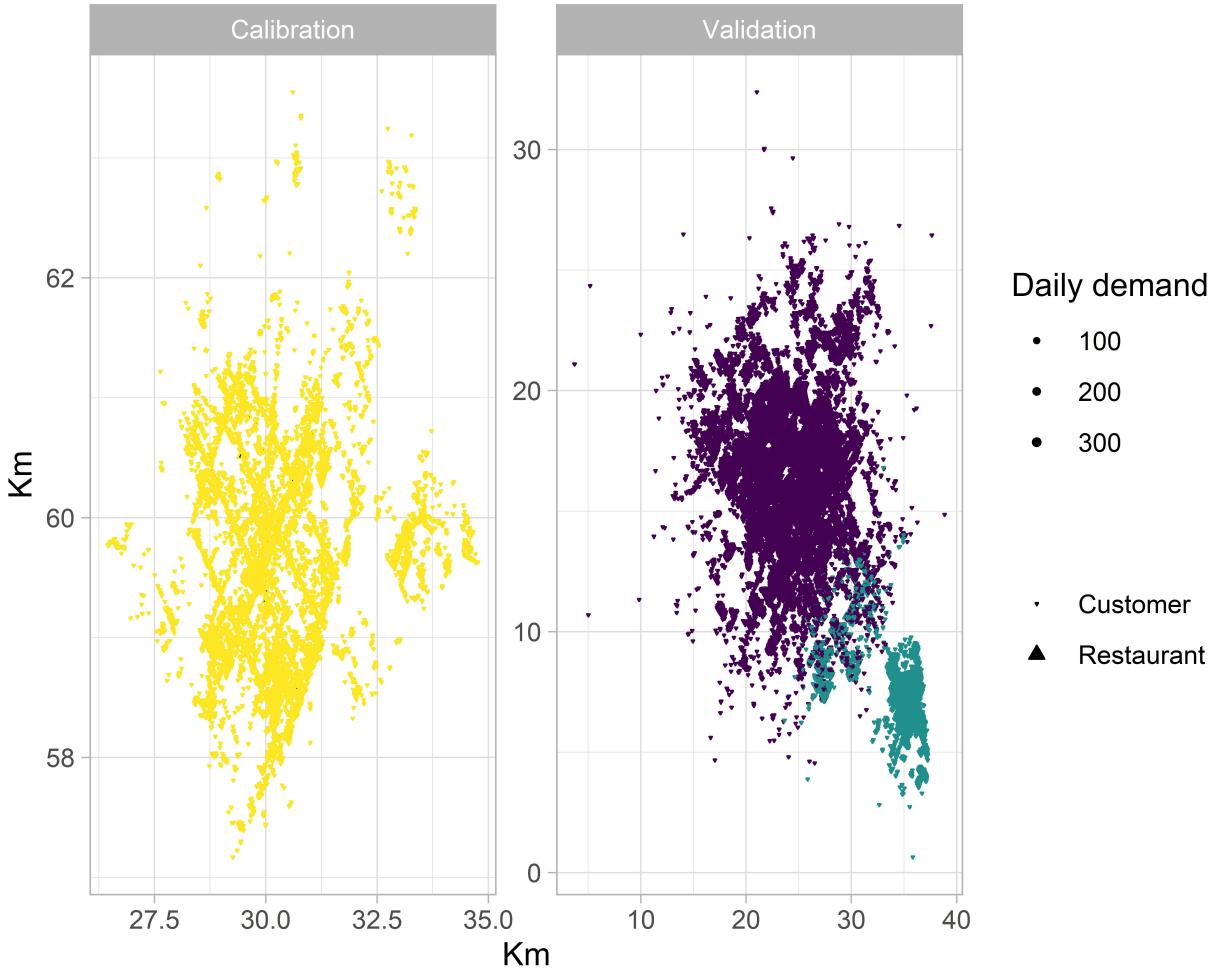


Figure 9 Order topology.

Note. Orders are clustered using k-means into three main cities. Two cities are used for validation, the other city is used for calibration. On the left, a grid-like street network can be observed.

References

Fujimoto S, van Hoof H, Meger D, 2018 *Addressing function approximation error in actor-critic methods*. *International Conference on Machine Learning (ICML)*, 1587–1596 (PMLR), URL `http://proceedings.mlr.press/v80/fujimoto18a.html`.

INFORMS, 2024 *TSL Data-Driven Research Challenge (TSL-Meituan 2024)*. URL `https://connect.informs.org/tsl/tslresources/datachallenge`, last accessed: 2026-01-24.

Kingma DP, Ba J, 2014 *Adam: A method for stochastic optimization*. URL `https://arxiv.org/abs/1412.6980`, arXiv preprint arXiv:1412.6980.