Multi-Agent Reinforcement Learning from Delayed Marketplace Feedback for Objective-Weight Adaptation in Three-Sided Dispatch

Haochen Wu [1] **Yi Hou** [1] **Ryan Xie** [1]

Abstract

Dispatch in three-sided marketplaces provides a natural setting for reinforcement learning from world feedback: decisions are evaluated by delayed operational outcomes such as delivery speed, courier utilization, and merchant congestion. We present a deployed reinforcement learning system at DoorDash that adapts dispatch objective weights in a large-scale food-delivery marketplace using delayed signals. Rather than replacing the combinatorial assignment optimizer, a store-level policy learned from logged marketplace data selects a discrete multiplier that shifts the dispatch optimizer's tradeoff between delivery quality and batching efficiency. This interface enables offline policy learning under noisy, delayed, and coupled feedback while preserving production feasibility constraints and operational safeguards. We train a shared value function using centralized offline data and decentralized store-level execution, with Double Q-learning targets and a conservative regularizer to reduce out-of-distribution value overestimation. In a production switchback experiment, the offline-trained policy increases batching and reduces courier-side time costs without degrading customer-facing delivery quality. Results illustrate how world feedback from a live economic and logistics system can be used to safely adapt decision policies online.

1. Introduction

Food-delivery dispatch is a sequential decision problem embedded in a three-sided marketplace. Each decision affects customer delivery quality, merchant congestion, courier availability, batching opportunities, and cost effectiveness, which are evaluated not by human preference labels but by delayed operational outcomes from the marketplace. Dispatch systems must therefore balance competing objectives: batching can improve courier efficiency, while faster execution can reduce lateness and improve customer experience (Ulmer et al., 2021; Agatz et al., 2023). In production, this tradeoff is often governed by static heuristic weights that are tuned globally and updated manually, making them brittle under local, time-varying conditions: over-batching during congestion can increase lateness, while under-batching during slack periods leaves efficiency gains unrealized.

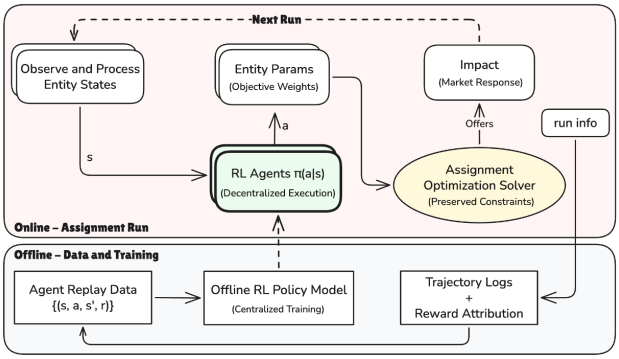

Figure 1. Agentic objective-weight adaptation loop for production dispatch. During online serving, a policy agent observes local states and selects an objective-weight multiplier that parameterizes the assignment optimizer. The optimizer remains responsible for feasible courier-order assignment decisions. During offline learning, logged runs are joined with delayed marketplace outcomes to construct transition data tuples for policy training and deployment.

We present a deployed offline reinforcement learning (RL) system for real-time objective-weight adaptation in DoorDash's large-scale food-delivery dispatch platform. Rather than replacing the combinatorial assignment optimizer, the learned policy controls a narrow objective-weight interface that shifts the optimizer's tradeoff between delivery speed and batching efficiency. Every dispatch run, each store-level RL agent observes local marketplace state and selects a discrete multiplier applied to the optimizer objective. This constrained action space preserves existing optimization and operational safeguards while enabling local adaptation. The learned policy is trained offline from logged marketplace

[1]DoorDash Inc. Correspondence to: Haochen Wu <haochen.wu@doordash.com>.

Accepted at the International Conference on Machine Learning (ICML) Workshop on Reinforcement Learning from World Feedback (RLxF), Seoul, South Korea. 2026. Copyright 2026 by the author(s).

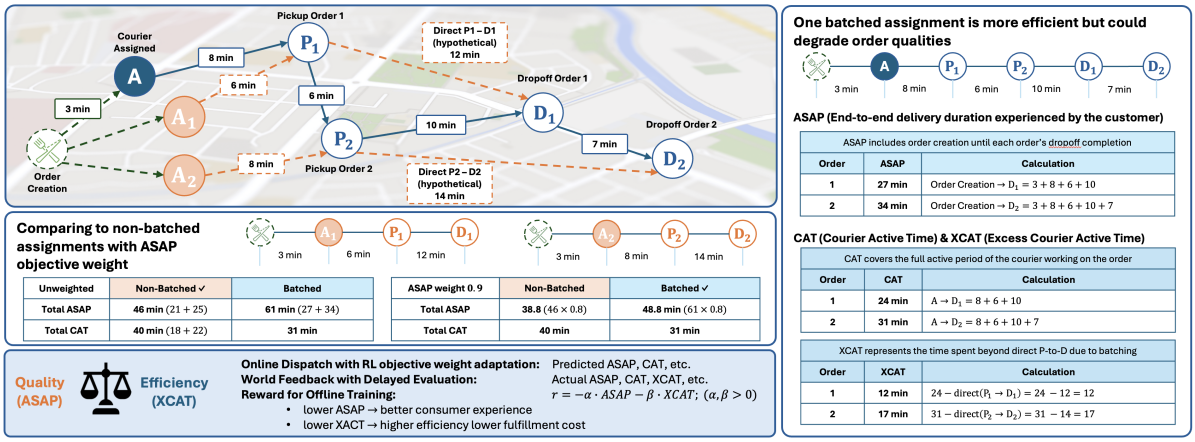

Figure 2. Logistics timing compoents used for delayed reward attribution. ASAP measures customer-facing delivery duration, and XCAT measures excess courier active time (CAT) beyond the direct route. The RL-selected ASAP-weight multiplier steers the optimizer's speed-efficiency tradeoff: higher weights favor faster delivery completion, while lower weights make batching-compatible assignments more attractive. Observed ASAP and XCAT outcomes are joined back to dispatch runs for reward construction.

outcomes and deployed at production scale, serving hundreds of millions of daily inferences at a 20-second cadence.

Prior work has applied RL to food-delivery operations, including dispatching, courier assignment, routing, repositioning, and batching (Chen et al., 2024; Jahanshahi et al., 2022; Zou et al., 2021; Guo et al., 2021; Lu et al., 2024; Cheng & Azadeh, 2025). Most methods learn direct operational decisions and are evaluated in simulated or offline settings. We study a complementary production setting where the RL agent learns from delayed marketplace feedback while modulating the objective of an existing assignment optimizer through a constrained adaptation interface. This setting introduces delayed and coupled feedback, as dispatch decisions jointly affect quality for customers, merchant congestion, and courier utilization.

Our contributions are threefold: 1) Introducing an RL architecture that learns from marketplace feedback while adapting dispatch objective weights through a low-dimensional control interface rather than replacing the production assignment optimizer. 2) Formulating objective-weight adaptation as an offline multi-agent decision-making problem with store-level decentralized execution and delayed regional rewards from the real-world marketplace. 3) Providing production switchback evidence showing increased batching and reduced courier-side time costs without degrading customer-facing delivery quality.

2. Learning from Marketplace Feedback

The deployed system has two nested decision layers. The inner is the optimizer, mapping orders, couriers, constraints, and objective weights to courier-order assignments. The outer is the objective-weight adaptation agent (OWA-RL) that learns from marketplace outcomes and selects a store-level objective-weight multiplier before optimization.

Formulation We formulate this outer adaptation problem as a decentralized multi-agent Markov decision process with centralized offline training. Let $\mathcal{I} = \{1, \ldots, N\}$ denote stores, each treated as an agent. At assignment cycle t, store i observes local state $s_t^i \in \mathcal{S}$ and selects action $a_t^i \in \mathcal{A}$. The action changes the downstream optimizer objective and the resulting assignment decisions induce delayed marketplace outcomes. We write the RL problem as $\mathcal{M} = (\mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \lambda_0)$, where $\mathcal{T}$ denotes transition dynamics induced jointly by marketplace evolution and the production optimizer, R is a delayed multi-objective reward, γ is the discount factor, and λ_0 is the baseline delivery-speed objective weight (ASAP weight).

Action $\mathcal{A} = \{0.8, 0.9, 1.0, 1.1, 1.2\}$. The action is a discrete multiplier on the baseline ASAP weight. Store i selects $a_t^i \in \mathcal{A}$, producing the adapted weight $\lambda_t^i = a_t^i \lambda_0$ for the current assignment cycle. Lower weights make batching or more efficient routing more attractive during optimization (the solid line in Figure 2). Higher weights emphasize faster order completion and may lead the optimizer to assign nearby orders to separate couriers rather than batch them (the dashed lines in Figure 2). The neutral action $a_t^i = 1.0$ recovers the static production baseline. Thus, the policy steers the existing optimizer through a constrained objective-weight interface, while the optimizer enforces feasibility and makes the final courier-order assignments.

State $s_t^i = \left[d_t^i, \mathrm{sup}_t^i, \mathrm{cwt}_t^i\right]$. The store-level state is refreshed every assignment run. Here, d_t^i is outstanding delivery count, cwt_t^i is median courier wait time, and sup_t^i is a localized supply-pressure feature. To expose

store-level variation hidden by regional supply signals, we rescale the regional feature by effective courier supply as $\sup_t^i = \sup_t^{g(i)} \cdot \frac{\widetilde{S}_t^{g(i)}}{\widetilde{S}_t^i}$, where $g(i)$ is the region containing store i, $\widetilde{S}_t^i$ is the recent median number of feasible couriers reaching optimization for store i, and $\widetilde{S}_t^{g(i)}$ is the corresponding regional median. When the store has more feasible couriers than the regional median ($\widetilde{S}_t^{g(i)} > \widetilde{S}_t^i$), it results in lower supply pressure.

Reward. Rewards are computed from delayed delivery outcomes and aggregated regionally to capture the network effect across nearby stores and couriers. Figure 2 summarizes the timing quantities used in the reward. For delivery j, with order creation, courier acceptance, and dropoff times t_j^{create}, t_j^{acc}, and t_j^{drop}, define

$$\text{ASAP}_j = t_j^{\text{drop}} - t_j^{\text{create}}, \qquad \text{CAT}_j = t_j^{\text{drop}} - t_j^{\text{acc}}, \quad (1)$$

$$\text{XCAT}_j = \text{CAT}_j - T_j^{\text{direct}}, \quad (2)$$

where T_j^{direct} is direct pickup-to-dropoff travel time if delivery j were served alone. Let $\mathcal{D}_t^{g(i)}$ be deliveries in region $g(i)$ whose outcomes are attributed to decision cycle t. The reward is

$$r_t^{g(i)} = -\frac{1}{|\mathcal{D}_t^{g(i)}|} \sum_{j \in \mathcal{D}_t^{g(i)}} \left(\alpha\, \text{ASAP}_j + \beta\, \text{XCAT}_j\right). \quad (3)$$

The reward is a compact representation of the marketplace feedback. ASAP captures customer-facing latency, XCAT captures excess courier-side effort beyond direct travel, and regional aggregation captures network effects across nearby stores and couriers.

Offline Training Let $\pi_\theta(a \mid s)$ denote the shared store-level policy. Stores execute the policy independently using local state, while training pools experience across stores and uses regional rewards to capture cross-store network effects. A transition pipeline joins logged online decisions with delayed fulfillment outcomes to construct $\mathcal{D} = \{(s_t^i, a_t^i, r_t^{g(i)}, s_{t+1}^i)\}$, where $g(i)$ is the dispatch region containing store i and $r_t^{g(i)}$ is the realized regional reward attributed to decision cycle t. The objective is to learn a policy maximizing expected discounted regional reward, $\pi^* = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{k=0}^{H} \gamma^k r_{t+k}^{g(i)}\right]$.

We train a discrete-action value function $Q_\theta(s, a)$ using an offline Double DQN objective (Mnih et al., 2015; van Hasselt et al., 2016), where the behavior network selects the next action and the target network evaluates it with temporal-difference loss:

$$y_t^{\text{DDQN}} = r_t^{g(i)} + \gamma Q_{\bar{\theta}}\left(s_{t+1}^i, \arg\max_{a' \in \mathcal{A}} Q_\theta(s_{t+1}^i, a')\right). \quad (4)$$

$$\mathcal{L}_{\text{DDQN}}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}}\left[\left(Q_\theta(s,a) - y^{\text{DDQN}}\right)^2\right]. \quad (5)$$

Because the learned policy may assign high values to actions weakly supported by logged data, we add a discrete Conservative Q-Learning regularizer (Kumar et al., 2020):

$$\mathcal{L}_{\text{CQL}}(\theta) = \mathbb{E}_{(s,a) \sim \mathcal{D}}\left[\log \sum_{a' \in \mathcal{A}} \exp Q_\theta(s, a') - Q_\theta(s, a)\right]. \quad (6)$$

The final training objective is

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{DDQN}}(\theta) + \eta \mathcal{L}_{\text{CQL}}(\theta), \quad (7)$$

where η controls the strength of conservatism. Double DQN reduces maximization bias, while the conservative penalty discourages unsupported actions from receiving high values, improving offline training stability before deployment.

Appendix A details the controlled offline data collection, policy architecture, and training hyperparameters. Appendix A.3 compares training curves with a DQN baseline, showing how the conservative regularizer affects offline learning before online deployment.

3. Production Switchback Experiment

Experiment Setup Before deployment, we use offline reward-reweighting diagnostics to test whether the learned policy responds directionally to different definitions of marketplace feedback. We choose $\alpha = 0.9$ because it yields a more balanced action distribution, avoiding collapse toward either aggressive batching or speed prioritization. Full diagnostics are provided in Appendix B.

We evaluate OWA-RL against the DoorDash production baseline using a global switchback experiment. The randomization units are approximately 4,000 geographic regions. At each two-hour switchback interval, regions are randomly assigned to treatment or control, with approximately half of regions exposed to each condition in each interval, over a two-week experiment period. The control uses the baseline static objective weight, while the treatment uses the OWA-RL policy trained with $\alpha = 0.9$. Eligible traffic includes all stores in randomized regions.

Treatment changes only weight selection; the optimizer, constraints, and serving infrastructure remain fixed. We estimate treatment effects using CUPED variance reduction and compute p-values clustered at the region-hour switchback bucket. Following standard switchback-experiment practice, we monitor customer-experience quality, cancellations, and carryover effects as guardrails (Bojinov et al., 2023).

Policy Behavior To verify the deployed policy uses local marketplace state, we inspect predicted actions from the

Table 1. Online experiment results for all day parts and dinner. Baseline and OWA-RL report metric means, while ATE reports the variance-reduction adjusted average treatment effect, with p-values shown in parentheses. CAT, CWT, and ASAP are measured in seconds; % batched and % 20-min late are reported as percentage-point rates. Lower values are better for CAT, CWT, ASAP, and % 20-min late, while higher values are better for % batched. Statistically significant ATEs at $p < 0.05$ are bolded.

		Efficiency Metrics			Quality Metrics	
Scope	Statistic	CAT (sec.)	CWT (sec.)	% batched	ASAP (sec.)	% 20-min late
All Day Parts	Baseline	1163.0	277.1	47.52%	1956.0	2.09%
	OWA-RL	1159.8	275.7	48.14%	1960.0	2.09%
	ATE (p)	$\mathbf{-1.261}$ ($\mathbf{0.019}$)	$\mathbf{-0.856}$ ($\mathbf{0.004}$)	$\mathbf{+0.495}$ ($\mathbf{< 0.001}$)	+0.972 (0.264)	−0.012 (0.237)
Dinner	Baseline	1156.3	262.9	57.99%	2168.3	2.36%
	OWA-RL	1153.1	261.6	58.59%	2173.0	2.34%
	ATE (p)	$\mathbf{-1.289}$ ($\mathbf{0.042}$)	$\mathbf{-1.030}$ ($\mathbf{0.041}$)	$\mathbf{+0.600}$ ($\mathbf{0.010}$)	+0.869 (0.633)	$\mathbf{-0.037}$ ($\mathbf{0.040}$)

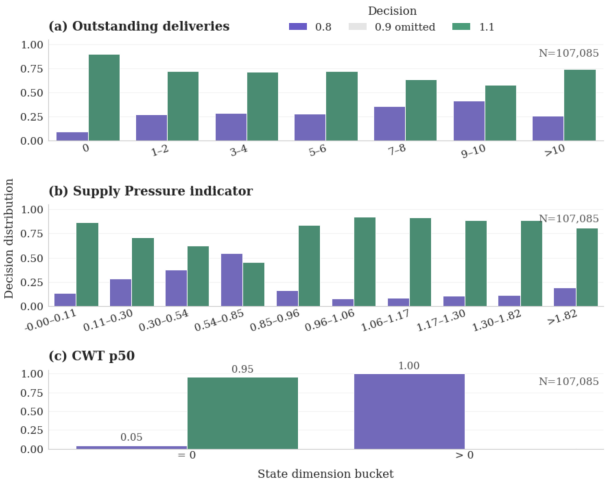

Figure 3. Empirical action distributions from DoorDash production logs during a Friday dinner peak, showing state-dependent ASAP-weight multipliers across backlog, supply pressure, and CWT.

San Francisco-Bay Area market during Friday dinner peak. Figure 3 shows that the policy shifts probability mass across ASAP-weight multipliers as outstanding deliveries, supply pressure, and courier wait time change. The dominant action is often the higher ASAP-weight multiplier, but the lower multiplier becomes more likely in specific backlog and supply regimes. We monitor how decisions and supply pressure states change week over week in Appendix C, indicating state-dependent adaptation rather than fixed global retuning.

Online Impact Table 1 reports how a policy trained from logged world feedback transfers to online marketplace operation and summarizes results for all day parts and dinner, including baseline and OWA-RL means, CUPED-adjusted average treatment effects (ATE), and clustered p-values. Across all day parts, OWA-RL improves efficiency without degrading delivery quality: CAT and CWT decrease significantly, batching increases by 0.495 percentage points, and ASAP and 20-minute lateness remain statistically unchanged. CWT is important because courier wait time at merchant pickup contributes to courier utilization and indicates non-optimal arrival timing relative to order readiness; it complements CAT, which primarily captures active travel and service time. During dinner, the policy again reduces CAT and CWT and increases batching, while ASAP remains unchanged, and 20-minute lateness improves slightly. These results show that adaptive objective-weight selection increases batching and reduces courier-side time costs without degrading customer-facing quality at production scale.

4. Conclusion

We presented a production case study of reinforcement learning from delayed marketplace feedback for real-time objective-weight adaptation in a three-sided dispatch system at DoorDash. By exposing a decision layer before the existing combinatorial assignment optimizer, online experiments show the learned policy increases batching, reduces courier-side time costs, preserves global delivery quality and operational safeguards, and improves dinner-hour lateness.

The current approach only acts through a low-dimensional objective-weight interface, which ensures serving reliability, scalability, and latency. Rewards are attributed using delayed regional outcomes, better reflecting the marketplace network effect, but introducing noisy credit assignment for individual store-level actions. Also, the reliability of the offline-trained policy heavily depends on the stability of marketplace dynamics after deployment.

Future work includes introducing different decision layers to the dispatch system. This would require studying systematic methods for detecting distribution shift and RL decision interactions in the multi-agent setting. We have extended the monitoring of state, action, and reward distributions, with examples shown in Appendix C. In addition, interpretability tools based on large-language-models that link RL policy decisions to dynamic states could support automated hypothesis generation, debugging, and policy retraining.

Impact Statement

This paper presents work with the goal to advance reinforcement learning methods for large-scale logistics and marketplace dispatch. Since dispatch decisions affect customers, merchants, and couriers, production use should include monitoring of service-quality guardrails, courier wait time, and workload effects, regional heterogeneity, distribution shift, and rollback criteria. This work also emphasizes the importance of having a constrained control layer, offline RL safeguards, and online experimentation when deploying reinforcement learning in systems with real-world operational and labor impacts.

References

Agatz, N., Fan, Y., and Stam, D. Crowdsourced on-demand food delivery: An order batching and assignment algorithm. *Transportation Research Part C: Emerging Technologies*, 149:104055, 2023. doi: 10.1016/j.trc.2023.104055.

Bojinov, I., Simchi-Levi, D., and Zhao, J. Design and analysis of switchback experiments. *Management Science*, 69 (7):3759–3777, 2023. doi: 10.1287/mnsc.2022.4583.

Chen, J., Wang, L., Pan, Z., Wu, Y., Zheng, J., and Ding, X. A matching algorithm with reinforcement learning and decoupling strategy for order dispatching in on-demand food delivery. *Tsinghua Science and Technology*, 29(2): 386–399, 2024. doi: 10.26599/TST.2023.9010069.

Cheng, J. and Azadeh, S. S. Real-time integrated dispatching and idle fleet steering with deep reinforcement learning for a meal delivery platform, 2025. URL https://arxiv.org/abs/2501.05808.

Guo, B., Wang, S., Ding, Y., Wang, G., He, S., Zhang, D., and He, T. Concurrent order dispatch for instant delivery with time-constrained actor-critic reinforcement learning. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2021.

Jahanshahi, H., Bozanta, A., Cevik, M., Kavuk, E. M., Tosun, A., Sonuc, S. B., Kosucu, B., and Başar, A. A deep reinforcement learning approach for the meal delivery problem. *Knowledge-Based Systems*, 243:108489, 2022. doi: 10.1016/j.knosys.2022.108489.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1179–1191, 2020.

Lu, M., Yan, X., Azadeh, S. S., and Wang, P. An adaptive agent-based approach for instant delivery order dispatching: Incorporating task buffering and dynamic batching strategies. *International Journal of Transportation Science and Technology*, 13:137–154, 2024. doi: 10.1016/j.ijtst.2023.12.006.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015. doi: 10.1038/nature14236.

Ulmer, M. W., Thomas, B. W., Campbell, A. M., and Woyak, N. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. *Transportation Science*, 55(1):75–100, 2021. doi: 10.1287/trsc.2020.1000.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Zou, G., Tang, J., Yilmaz, L., and Kong, X. Online food ordering delivery strategies based on deep reinforcement learning. *Applied Intelligence*, 2021. doi: 10.1007/s10489-021-02750-3.

A. Offline Training

A.1. Data Collection

Training data is collected over two iterations using a controlled regional rollout. In each iteration, data collection lasts approximately two days. To limit potential user-facing impact, we randomly select only 0.5% of global regions every two hours for data collection. This rollout design keeps the intervention localized while allowing us to measure cumulative impact within each two-hour window.

In the first iteration, data is collected using pure exploration, where actions are sampled uniformly at random from the action space. After training an initial policy on this dataset, we conduct a second data-collection iteration using a mixture policy with 50% exploitation and 50% exploration. Specifically, with probability 0.5, the learned policy is used to select the action, and with probability 0.5, a random action is sampled from the action space.

Each logged transition is represented as

$$(s, a, s', r^g), \forall \texttt{store_id}, \texttt{run_id},$$

where s and s' denote the current and next states, a is the selected action, and r^g is the region-level reward. Since each store in each run contributes one data point, we construct transition tuples (s, a, s') by pairing consecutive runs.

The system executes approximately three runs per minute, yielding $24 \times 60 \times 3 = 4{,}320$ runs per day. With roughly 10 stores per run, this corresponds to $43{,}200$ store-level data points per region per day. Across $4{,}000$ regions, full rollout would produce approximately $4{,}000 \times 43{,}200 = 172.8$ million data points per day. Under the controlled 0.5% rollout, only about 20 regions are selected in each two-hour window, resulting in approximately $864{,}000$ data points per day. Therefore, each two-day iteration produces approximately 1.73 million transition samples, and the two iterations together yield approximately 3.46 million samples.

A.2. Policy Model and Parameters

We parameterize the policy as a lightweight neural network that maps each store-level state to a discrete action. The input state is a three-dimensional feature vector $s \in \mathbb{R}^3$, and the action space contains $|\mathcal{A}| = 5$ candidate actions. The policy network is implemented as a two-layer multilayer perceptron with hidden dimension 16 and ReLU activations:

$$Q_\theta(s) = \mathrm{Linear}(3, 16) \to \mathrm{ReLU} \to \mathrm{Linear}(16, 16) \to \mathrm{ReLU} \to \mathrm{Linear}(16, |\mathcal{A}|).$$

At serving time, the policy selects the action with the largest predicted score:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q_\theta(s)_a.$$

The model is trained offline using logged transition tuples (s, a, s', r) collected from the controlled rollout. Training is initialized from a previously trained checkpoint and continued for 30 epochs with mini-batches of size 32. We use the Adam optimizer with learning rate 10^{-3}, discount factor $\gamma = 0.99$, gradient clipping with maximum norm 10, and a target network that is refreshed every 2 epochs. The final checkpoint is exported for offline evaluation and deployment.

A.3. Training Curves

Figure 4 shows offline training MSE loss across epochs for the OWA-RL learner (two iterations) and a DQN baseline without CQL regularization. The DQN baseline reaches a lower MSE more quickly, while OWA-RL maintains a higher training loss because the conservative regularizer penalizes high values on unsupported actions. This behavior is expected: the goal of the conservative objective is not to minimize Bellman error alone, but to improve offline deployment stability by discouraging overestimated values for actions weakly supported by logged data.

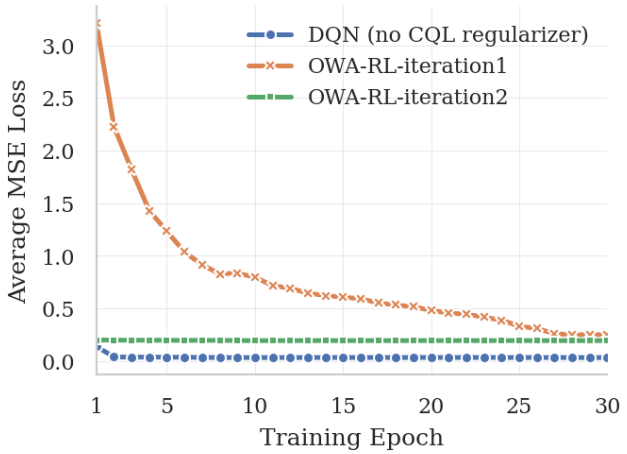

Figure 4. Offline training MSE loss across epochs. The DQN baseline without CQL regularization minimizes MSE faster, while OWA-RL maintains a higher loss due to the conservative penalty used to reduce unsupported-action overestimation.

B. Policy Behavior Under Reward Reweighting

Before deployment, we evaluate whether the learned policy responds directionally to reward design. Figure 5 shows predicted action distributions under different reward-weight settings. Increasing the efficiency weight shifts probability mass toward lower ASAP-weight multipliers, making batch-compatible assignments more attractive to the optimizer; increasing the speed weight shifts mass toward higher multipliers, increasing the penalty on delay. This sensitivity check suggests that the policy uses the constrained action space to express the intended speed-efficiency tradeoff rather than collapsing to a single static action.

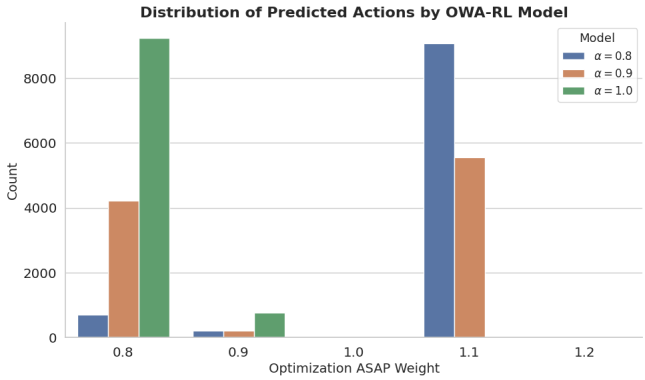

Figure 5. Predicted action distributions under different reward-weight settings. Changing the reward weights shifts the selected objective-weight multipliers, showing that the learned policy adapts its behavior to the desired speed-efficiency tradeoff.

C. Examples of Drift-Monitoring Diagnostics

Figure 6 shows the daily distribution of selected ASAP-weight multipliers, while Figure 7 shows the daily distribution of the supply indicator over the same period. These diagnostics are intended for production monitoring: large shifts in action distributions may indicate policy-behavior drift, while shifts in state-feature distributions may indicate marketplace drift that can affect policy reliability.

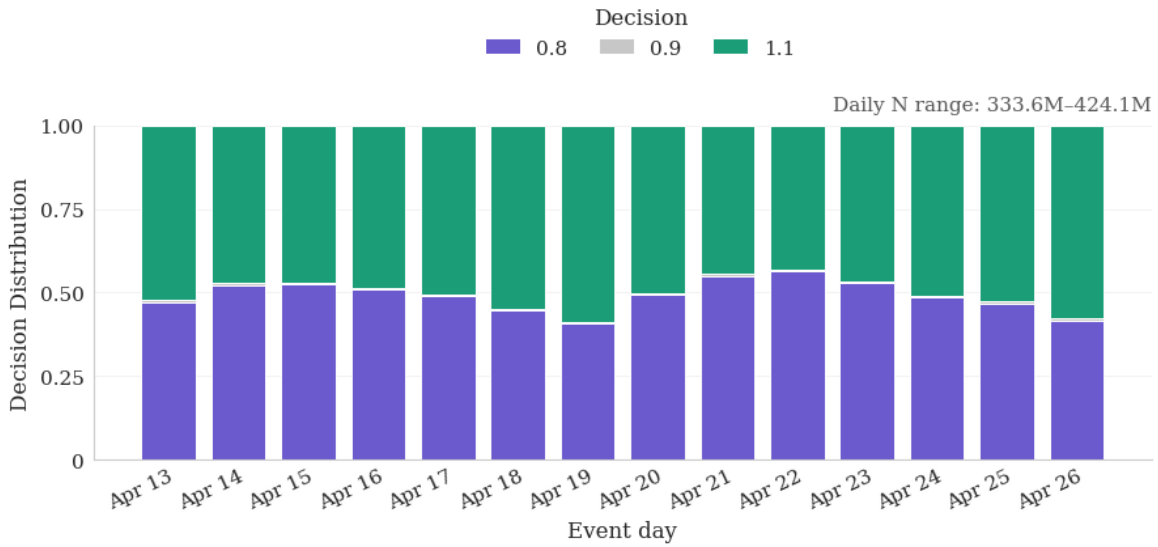

Figure 6. Daily distribution of selected ASAP-weight multipliers in production. Monitoring action distributions helps detect policy-behavior drift after deployment.

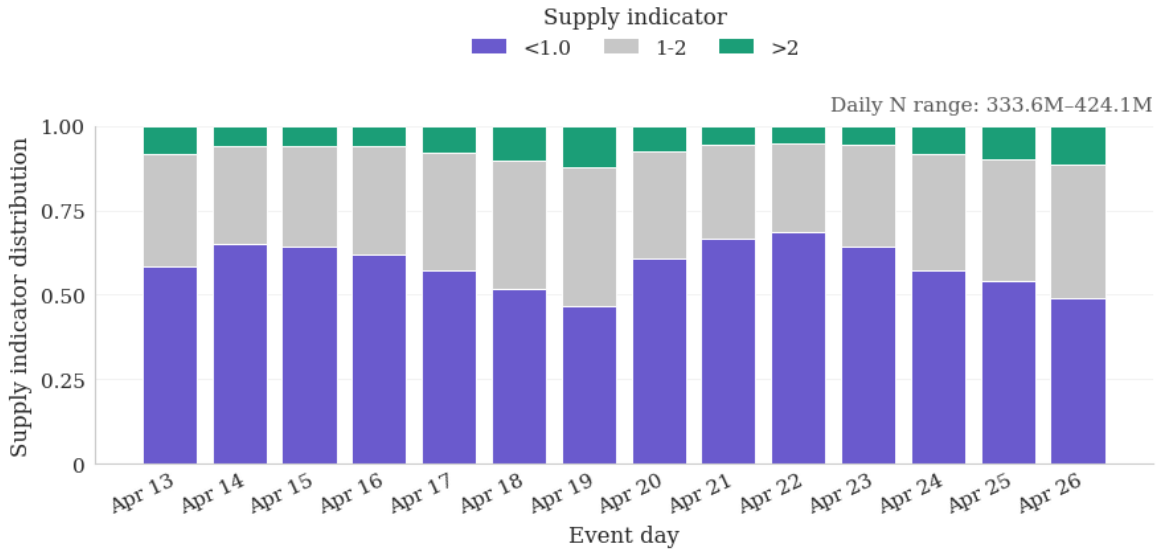

Figure 7. Daily distribution of the supply-pressure indicator in production. Monitoring state-feature distributions helps detect marketplace drift that may affect policy reliability.