

DeepStock: Reinforcement Learning with Policy Regularizations for Inventory Management

Yaqi Xie¹, Xinru Hao², Jiayi Liu³, Will Ma⁴, Linwei Xin⁵, Lei Cao², Yidong Zhang²

¹Booth School of Business, University of Chicago, Chicago, USA. yaqi.xie@chicagobooth.edu

²Taobao & Tmall Group, Hangzhou, China. xinru.hxr@taobao.com;huaju.cl@taobao.com;yidongzster@gmail.com

³School of Economics, Sichuan University, Chengdu, China. liujiayi@stu.scu.edu.cn

⁴Graduate School of Business, Columbia University, New York, USA. wm2428@gsb.columbia.edu

⁵School of Operations Research and Information Engineering, Cornell University, Ithaca, USA. lx267@cornell.edu

Deep Reinforcement Learning (DRL) provides a general-purpose methodology for training inventory policies that can leverage big data and compute. However, off-the-shelf implementations of DRL have seen mixed success, often plagued by high sensitivity to the hyperparameters used during training. In this paper, we show that by imposing policy regularizations, grounded in classical inventory concepts such as “Base Stock”, we can significantly accelerate hyperparameter tuning and improve the final performance of several DRL methods. We report details from a full-scale deployment of DRL with policy regularizations on Alibaba’s e-commerce platform, Tmall. We also include extensive synthetic experiments, which show that policy regularizations reshape the narrative on what is the best DRL method for inventory management.

History: This draft: November 22, 2025

Introduction

Inventory control is a foundational problem in the field of Operations Research. Historically, the focus has been on stylized inventory and demand models that allow for the derivation of optimal policies. In modern times, these models can be significantly enhanced by leveraging high-dimensional contextual data. Even though optimal policies are no longer analytically defined, Deep Reinforcement Learning (DRL) offers a promising general-purpose methodology for learning performant policies that act on this high-dimensional data.

⁰YX, XH, and JL are co-first authors with equal contribution. WM and LX provided academic guidance, while LC and YZ offered industry leadership.

However, the success of applying DRL to inventory control has been mixed (see Gijbrecchts et al. 2025). Indeed, a common approach is to frame inventory control as a generic sequential decision-making problem and then apply off-the-shelf DRL methods, which leads to actions that are difficult to interpret. More importantly, a dealbreaker is that the performance of DRL is highly sensitive to hyperparameters, whose tuning is time-consuming.

In this paper, we show how these dealbreakers can be mitigated through policy regularizations, where we encode simple intuitions from inventory theory into the policy learned by DRL. One such regularization incorporates the inventory concept of “Base Stock” into Deep RL, which is why we call our algorithm “DeepStock”. This not only reduces the black-box nature of DRL, but drastically speeds up training and improves final performance. In fact, our policy regularization techniques have enabled the full-scale deployment of DRL at Alibaba. As of October 2025, our algorithm manages inventory replenishment for 100% of the products (both domestic and international) sold by Alibaba on its B2C e-commerce platform, Tmall, covering over 1 million SKU-warehouse combinations.

Description of DRL and Policy Regularizations

A DRL method learns a policy π , represented by a (deep) neural network, that can decide an inventory ordering action from any state. In our problem, the state of an SKU at a time t includes exogenous features $x_t \in \mathbb{R}^m$, which consists of both static attributes (e.g., product category, demand magnitude, supplier, lead time, review period, profit margin) and dynamic attributes that evolve over time (e.g., upcoming promotions, seasonality, recent social media trends). In addition, the state includes endogenous information I_t about upcoming inventory shipments, which depend on prior actions (i.e., previously placed orders). This rich state space allows for meta-learning across SKU’s with diverse characteristics, such as in demand magnitude (e.g., 10 vs. 10,000 weekly sales), lead time (domestic vs.

international), and general demand pattern (e.g., fast-moving vs. long-tail). Neural networks can encode this high-dimensional information into meaningful representations while filtering out noisy signals.

When a DRL algorithm is applied off-the-shelf, the policy π is a typically single neural network that outputs an order quantity $\pi(I_t, x_t)$ based on the input state I_t, x_t .

Our first policy regularization imposes that the order quantity takes the functional form $\pi(I_t, x_t) = \max\{\mu_{\text{BASE}}(I_t, x_t) - \text{tot}(I_t), 0\}$, where μ_{BASE} is the learned neural network, and tot denotes the total inventory (including upcoming shipments) contained in I_t . Here, BASE stands for “base stock”, where $\mu_{\text{BASE}}(I_t, x_t)$ represents the target level for the total inventory, and the order quantity $\pi(I_t, x_t)$ equals this target level minus the total inventory $\text{tot}(I_t)$ we already have. Intuitively, the learned target $\mu_{\text{BASE}}(I_t, x_t)$ should depend mostly on the exogenous features x_t which forecasts the upcoming demand, while the inventory information I_t is incorporated into the decision through the $\text{tot}(I_t)$ term.

Our second policy regularization imposes the functional form $\pi(I_t, x_t) = \mu_{\text{COEFF}}(I_t, x_t)^\top \text{feat}(x_t)$, where μ_{COEFF} is a learned neural network with an m' -dimensional output, providing Coefficients for m' features extracted from $x_t \in \mathbb{R}^m$ by the mapping $\text{feat}(x_t) \in \mathbb{R}^{m'}$. At Alibaba, we have $m' = 5$, with $\text{feat}(x_t)$ consisting of 4 historical and forecasted demand features over the near and distant horizons, along with a constant bias term. Intuitively, the desired order quantity should exhibit a positive relationship with these features.

Finally, we can combine both of our policy regularizations, in which case we impose the order quantity to take the functional form $\pi(I_t, x_t) = \max\{\mu_{\text{BOTH}}(I_t, x_t)^\top \text{feat}(x_t) - \text{tot}(I_t), 0\}$, with μ_{BOTH} being the learned neural network with an m' -dimensional output.

Main Results

Our policy regularizations can be tested in conjunction with any DRL method for training the neural network that defines the policy. We test two traditional DRL methods, DDPG (Lillicrap et al. 2015) and PPO (Schulman et al. 2017), which are representative model-free DRL methods that span different facets of DRL.

DDPG and PPO may only perform well under specific hyperparameter configurations, which change with each dataset, causing a major bottleneck to their deployment in the real world. In fact, other e-commerce firms like Amazon have adopted a different DRL method that directly takes gradients of the total simulated loss over a specified time horizon (Madeka et al. 2022), instead of learning Cost-to-go functions for intermediate states which is key to RL. The performance of this “Differentiable Simulator” (DS) method is much less sensitive to hyperparameter tuning.

We test our policy regularizations with DDPG, PPO, and DS, and report results based on synthetic data, offline data from Alibaba, and real-world deployments at Alibaba.

Takeaway I: Policy regularizations improve the performance of DRL methods, *especially when hyperparameter tuning is limited.*

We demonstrate Takeaway I on Alibaba’s offline data and especially our synthetic data, where we plot the best-so-far performance of DRL methods, with and without regularization, after each hyperparameter trial. The improvement from regularization is more drastic under a limited number of hyperparameter trials, even though the best performance after many trials also improves for all of DDPG, PPO, and DS.

To explain this finding, our regularizations encode into the policy human intuitions that prevent obvious blunders: our BASE regularization discourages large orders when we already have a lot of inventory in I_t , while our COEFF regularization ensures that larger

historical and forecasted demands imply larger orders. This especially improves the policy learned under the untuned hyperparameter configurations, where learning may be hindered by obvious blunders. In the synthetic experiments, we also discover a more nuanced reason why policy regularizations specifically facilitate the learning of a Q-function in RL.

Takeaway II: Policy regularizations *redefine the narrative* on whether DS is actually desirable compared to traditional DRL methods (DDPG,PPO) for inventory.

We demonstrate Takeaway II on Alibaba’s offline data and our synthetic data. As explained earlier, policy regularizations improve both traditional DRL and DS, but the improvement is more significant when hyperparameter tuning is limited and when learning a Q-function, which is more likely to be the case for traditional DRL than for DS.

We also dig deeper in our synthetic experiments to explain why traditional DRL can potentially beat DS. We find that DS has a tendency to overfit, because it does not explicitly learn intermediate structure in the form of a Cost-to-go function from a state or a Q-function from a state-action pair. However, given enough IID trajectories describing the evolution of SKU features and demands, the overfitting effect diminishes, and DS performs no worse than traditional DRL (with or without regularization). That being said, Alibaba still found DDPG to outperform DS when learning from 55,000 90-day SKU trajectories on its offline data, suggesting that even this sample size may be insufficient for DS to effectively learn from such high-dimensional trajectories.

Takeaway III: Policy regularizations enable a *unified, full-scale* deployment of DRL.

We demonstrate Takeaway III by reporting the impact of policy regularizations on Alibaba’s deployment of DRL in the real world. Alibaba proceeded to deploy DDPG with our policy regularizations, which generally achieved the best performance in its routine

offline tests, as exemplified by the results on Alibaba’s offline data presented in this paper. Although Alibaba has previously tinkered with DRL as reported in Liu et al. (2023), that work primarily addressed uncertain order yields arising from supply shortages during the COVID-19 pandemic. In addition, that DRL version was not general-purpose: it required grouping similar products and training separate models per group (see more details in Appendix B). By contrast, our policy regularizations have allowed for a full-scale deployment of DRL that is *unified*, where we can meta-learn a single policy across all SKU’s by leveraging their features, instead of separating them into groups such as fast-moving vs. long-tail SKU’s.

It is worth emphasizing that both the dynamic attributes in x_t and the inventory state I_t are normalized across time for each SKU before being fed into the neural network. In conjunction with our COEFF regularization, this facilitates meta-learning across SKU’s. To explain why, the normalization allows us to interpret patterns in dynamic attributes and inventory states in a consistent manner across SKU’s with different sales magnitudes, while the multiplication by $\text{feat}(x_t)$ in our COEFF regularization ensures that the resulting actions are properly denormalized with respect to the demand magnitudes. This allows unified meta-learning across all SKU’s, without requiring explicit clustering.

Paper outline. This rest of paper contains the following sections: State of the Literature on RL for Inventory; Inventory Model, Metrics, and Policies; Training using DRL; Experiments on Synthetic Data; Experiments on Alibaba’s Offline Data; Real-world Deployment and Impact at Alibaba; Conclusion.

State of the Literature on RL for Inventory

Policy regularizations. To the best of our knowledge, simple forms of policy regularizations such as the “base stock” structure have not been previously explored in the RL literature for

inventory management. Our high-level approach is to minimize an empirical loss function while imposing a policy structure that is satisfied by the optimal policy under stylized models, which does align with the sample complexity theory developed in Xie et al. (2025).

Many previous works however have encoded inventory policy structures into RL. As examples, De Moor et al. (2022) modify the reward function by embedding structure from heuristic inventory policies, using those as teacher policies; Qi et al. (2023) use a labeling method that captures the behavior of the optimal dynamic programming solution under historical observations, and then use that as a regularizer on the learned policy; Maggiar et al. (2025) impose a penalty term in the objective when the learned policy violates certain structural properties known to hold for optimal policies (as proved in the literature). These works generally differ from ours by incorporating penalty terms into the objective function, rather than directly restricting the policy space.

Finally, we note that our policy regularizations are problem-specific and should not be confused with generic regularization techniques commonly used in RL, such as entropy regularization (Haarnoja et al. 2018) or those used in trust-region policy optimization (Schulman et al. 2015a).

DS vs. traditional DRL for inventory. Although Madeka et al. (2022) and concurrently Alvo et al. (2023) have been promoting DS when doing DRL for inventory management, our paper rigorously compares DS to traditional DRL with proper hyperparameter tuning, finding that traditional DRL (with policy regularizations) can actually outperform. Our synthetic experiments reveal a new shortcoming of DS: its inability to learn across time when there are not enough parallel trajectories. That being said, DS performs excellently in synthetic settings given enough IID trajectories, which is consistent with the experimental findings in Alvo et al. (2023).

Large-scale deployments of DRL for inventory. There has been increasing documentation of deep learning deployments in inventory management at leading e-retailers, including Alibaba (Liu et al. 2023), Amazon (Madeka et al. 2022), and JD.com (Qi et al. 2023). A distinction of our work is that we achieve 100% coverage of all products sold by Alibaba on its Tmall e-commerce platform, which is only possible by training a unified DRL policy. Although all these works (including ours) omit many implementation details due to corporate confidentiality, our work also includes synthetic experiments that aim to academically illustrate our main findings inside the company.

Inventory Model, Metrics, and Policies

Inventory dynamics. Let T , P , and L be positive integers. We consider a discrete-time horizon of $t = 1, \dots, T$ days, where inventory orders are placed every P days (the “Period”), specifically at the beginnings of days $1, P + 1, 2P + 1, \dots$. Ordered inventory arrives L days later (the “Lead time”), i.e., at the beginnings of days $1 + L, P + 1 + L, \dots$. We let $I_t = (I_t^0, I_t^1, \dots, I_t^{L-1})$ denote the inventory state vector at the beginning of day t , where I_t^0 denotes the inventory on hand, and I_t^ℓ denotes the inventory that will arrive ℓ days into the future, for all $\ell = 1, \dots, L - 1$. We initialize $I_1^0 = I_1^1 = \dots = I_1^{L-1} = 0$.

If inventory is to be ordered on day t , we decide the quantity I_t^L at the beginning of day t ; otherwise we set $I_t^L = 0$. Over the course of the day, d_t consumers each attempt to purchase one unit of inventory, where d_t is called the “demand”. The actual sales made are $\min\{I_t^0, d_t\}$, and the inventory vector for the next day is updated as

$$I_{t+1}^0 = I_t^0 - \min\{I_t^0, d_t\} + I_t^1 = \max\{I_t^0 - d_t, 0\} + I_t^1; \quad I_{t+1}^1 = I_t^2, \dots, I_{t+1}^{L-1} = I_t^L. \quad (1)$$

Performance metrics. In inventory theory, the standard loss objective to minimize is

$$\ell_{\text{bh}} := \sum_{t=1}^T (b \cdot \max\{d_t - I_t^0, 0\} + h \cdot \max\{I_t^0 - d_t, 0\}).$$

That is, at the end of each day t : if $d_t \geq I_t^0$ (i.e., the inventory stocked out), then we are penalized b times the unmet demand $d_t - I_t^0$; otherwise, we are penalized h times the leftover inventory $I_t^0 - d_t$.

However, ℓ_{bh} is not a practical evaluation metric, as quantifying b and h is difficult in practice, particularly since unmet demand during stockouts cannot be directly observed when d_t is unknown. Therefore, at Alibaba, we evaluate instead the following two metrics:

$$\ell_{\text{SR}} := \frac{1}{T} \sum_{t=1}^T \mathbb{1}(d_t \geq I_t^0) \quad \text{and} \quad \ell_{\text{TT}} := \frac{\frac{1}{T} \sum_{t=1}^T \max\{I_t^0 - d_t, 0\}}{\frac{1}{T} \sum_{t=1}^T \min\{I_t^0, d_t\}}.$$

Here, the ‘‘Stockout Rate’’ loss ℓ_{SR} measures the % of days ending in stockout, penalizing having too little inventory. On the other hand, the ‘‘Turnover Time’’ loss ℓ_{TT} computes the average end-of-day inventory divided by average sales, which measures the average duration that a unit of inventory stays on hand, penalizing having too much inventory.

In our synthetic experiments, we train and evaluate policies using the standard objective ℓ_{bh} . We note that the loss objective ℓ_{bh} is naturally additive over time t , providing a reward signal for each action taken during RL training. At Alibaba, the policies are trained and evaluated using a weighted combination of ℓ_{SR} and ℓ_{TT} . We can adjust the relative weight to trade off between ℓ_{SR} and ℓ_{TT} in practice, ideally getting a Pareto improvement where both $\ell_{\text{SR}}, \ell_{\text{TT}}$ decrease. Although ℓ_{TT} cannot be exactly decomposed into per-period losses, Alibaba approximates the immediate loss of an action by applying the Turnover Time loss over the periods from its arrival until the arrival of the next action.

Contexts and policies. For all $t = 1, \dots, T$, we let $x_t \in \mathbb{R}^m$ denote a context vector available at the beginning of day t , that may be used to inform the inventory ordering decision. We assume that x_t contains all the up-to-date information and repeats static information such as P and L . An inventory policy π outputs an order quantity given any inventory

vector I_t and context vector x_t , setting the decision variable I_t^L to $\pi(I_t, x_t)$ (i.e., π does not depend on x_{t-1} , as that information is already outdated). We assume without loss that π is stationary, i.e., does not depend on t , because any time-specific information (e.g., holidays or seasonality) can be encoded into the context vector x_t .

Discussion of model assumptions. We will train inventory policies π by simulating the above inventory dynamics using historical trajectories $\xi = (x_t, d_t)_{t=1}^T$. We emphasize that the trained policies will be able to output decisions in real-world environments even if these simplified model dynamics are violated. For example, if we want to make an emergency order on a day $t \neq 1, P + 1, 2P + 1, \dots$, then we can still evaluate $\pi(I_t, x_t)$ based on the current I_t and x_t to get an ordering decision; meanwhile, if there are unexpected shipping delays, then the inventory update rule would deviate from (1) but the policy would proceed all the same. Of course, the resulting decisions may perform poorly under model violations—for example, if the actual lead time is highly random, then a policy calibrated for a deterministic L may fail to maintain sufficient inventory on hand. That being said, our success in real-world environments suggests that the policy trained on this simplified model is sufficient, and our general experience is that being able to train better because the model is simpler outweighs the benefits of modeling more complex inventory dynamics.

Training using Deep Reinforcement Learning (DRL)

We generally let \mathcal{D} denote a dataset of trajectories $\xi = (x_t, d_t)_{t=1}^T$, which can be synthetically generated, or based on historical data internal to Alibaba. In Alibaba’s data, d_t must be inferred if there was a stockout during day t , and this can be done by extrapolating d_t based on time of stockout. We train using these inferred demands, and again, the success of the trained policy after deployment suggests that the extrapolation procedure was appropriate.

We distinguish between datasets used for training, validation, and testing, denoting them using $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{validate}}$, and $\mathcal{D}^{\text{test}}$, respectively. $\mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{validate}}$ is the data available for

learning π , where during training π is updated iteratively based on the trajectories in $\mathcal{D}^{\text{train}}$, and a separate dataset $\mathcal{D}^{\text{validate}}$ is used to evaluate intermediate policies learned during the training and determine when training should stop. Finally, a third dataset $\mathcal{D}^{\text{test}}$ is used to evaluate out-of-sample performance, although in deployment we would evaluate based on the actually observed ℓ_{SR} and ℓ_{TT} metrics.

For any $\text{set} \in \{\text{train}, \text{validate}, \text{test}\}$ and $\text{metric} \in \{\text{bh}, \text{SR}, \text{TT}\}$, we define

$$L_{\text{metric}}^{\text{set}}(\pi) := \frac{1}{|\mathcal{D}^{\text{set}}|} \sum_{\xi \in \mathcal{D}^{\text{set}}} \ell_{\text{metric}}(\pi, \xi),$$

which is the average of loss $\ell_{\text{metric}}(\pi, \xi)$ over the trajectories $\xi \in \mathcal{D}^{\text{set}}$, for a policy π .

Overview of DRL Training Methods

A DRL method learns a policy π based on $\mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{validate}}$. While there is an enormous variety of DRL methods (see e.g., Achiam 2018), we focus on *model-free* methods which directly minimize the loss of π on trajectories in $\mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{validate}}$, instead of trying to learn a stochastic model for the evolution of $\xi = (x_t, d_t)_{t=1}^T$ and then using that to optimize π . Off-policy model-free DRL can learn purely by observing historical policies; meanwhile, on-policy model-free DRL requires simulating the current policies, which we can do, by replaying historical trajectories $\xi = (x_t, d_t)_{t=1}^T$ with the inferred demands d_t on our inventory model. Key concepts in model-free DRL include taking policy gradients and learning Q-functions; the methods we implement consider both of these concepts.

In this paper we consider Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO), which we believe to be appropriate for our inventory problem, and also span these distinctions between DRL methods. In particular, PPO simulates a randomized π and reinforces actions that generate high rewards under on-policy trajectories, being a modern policy gradient method that “clips” updates to prevent π from changing too quickly. PPO can include additional tricks such as generalized advantage

estimation (Schulman et al. 2015b) and an entropy bonus to encourage exploration. Meanwhile, DDPG learns a Q-function from a buffer of off-policy observations, which is used to optimize a (deterministic) policy by taking gradients of Q in the continuous action space (as is the case in our inventory problem). Given that on-policy simulation is also possible in our inventory problem, we always add the observations from the most recent policy to the buffer for learning the Q-function. DDPG can include additional tricks such as using a target network to stabilize the policy and Q-function updates.

Finally, we consider DS, the differentiable simulator method which takes policy gradients based on loss on the entire trajectory. It fully exploits the fact that any counterfactual inventory policy can be simulated on historical trajectories while allowing gradients.

Hyperparameter tuning. For any DRL method, the specific update rule for π depends on hyperparameters (e.g., “learning rate”, which determines the step size of each update). A DRL method may perform well only under carefully tuned hyperparameter configurations, which can vary across different datasets. Finding the optimal configuration is challenging, as both DDPG and PPO have at least 10 hyperparameters that may need to be adjusted to achieve good performance, and a configuration can only be evaluated after the training run converges, which can take many back-and-forths between training and validation. Exacerbating the challenge, even for a fixed hyperparameter configuration the performance of a training run is random, depending on the seed. We follow a standard hyperparameter tuning procedure that first uses Bayesian optimization to search over configurations, and then runs the most promising configurations with multiple seeds each to determine a winner.

Further details about our implementations of DDPG, PPO, and DS, including hyperparameter search ranges, can be found in Appendix A.

Combining DRL Methods with Policy Regularizations

A DRL method specifies how to train π , which can be used in conjunction with any of our policy regularizations (detailed in the Introduction) that impose a functional form on π . For each combination of $\text{DRL} \in \{\text{DDPG}, \text{PPO}, \text{DS}\}$ and $\text{REG} \in \{\text{NONE}, \text{BASE}, \text{COEFF}, \text{BOTH}\}$ (where **NONE** refers to “no regularization”, i.e., π directly outputs an order quantity), we let $\pi^{\text{DRL,REG}}$ denote the final policy learned.

Experiments on Synthetic Data

We fix $P = 2, L = 1, m = 1$, and set $T = 29$ unless stated otherwise. While the lost-sales dynamics introduced in (1) are common in practice (e.g., at Alibaba), we consider the backlogged dynamics $I_{t+1}^0 = I_t^0 - d_t + I_t^1$ (i.e., I_{t+1}^0 can be negative) in the synthetic experiments, as the optimal policy is relatively easy to compute. This formulation is also standard in many stylized inventory models and complements the lost-sales dynamics.

For each setting, we construct datasets $\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{validate}}, \mathcal{D}^{\text{test}}$ by drawing $\xi = (x_t, d_t)_{t=1}^T$ from trajectory distributions. In our first set of trajectory distributions which we call **INDEP**, for all $t = 1, \dots, T$, the feature x_t is the time t , and d_t is drawn from a scalar distribution that depends on t but is independent across t . We also consider trajectory-generation schemes in which the demand distribution for each SKU in $\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{validate}}$, or $\mathcal{D}^{\text{test}}$ is determined by parameters initialized randomly drawn from a prior, capturing the varying demand magnitudes observed across SKU’s. Indeed, in our second set of trajectory distributions, d_t is correlated over time following an **AR(1)** stochastic process with the drawn parameters, and the feature is $x_t = d_{t-1}$ which determines the state of the stochastic process (if the parameters were known). In our third set of trajectory distributions, d_t is **IID** across t following a distribution determined by a single parameter, and the feature is set exactly to this parameter. Details of these trajectory distributions can be found in Appendix A. We construct datasets for the following settings:

1. INDEP distribution with 20 sample trajectories, split so that $|\mathcal{D}^{\text{train}}| = |\mathcal{D}^{\text{validate}}| = 10$;
2. AR(1) distribution with 20 sample trajectories, split so that $|\mathcal{D}^{\text{train}}| = |\mathcal{D}^{\text{validate}}| = 10$;
3. AR(1) distribution with 10 sample trajectories, split so that $|\mathcal{D}^{\text{train}}| = |\mathcal{D}^{\text{validate}}| = 5$;
4. IID distribution with sample sizes $|\mathcal{D}^{\text{train}}| = |\mathcal{D}^{\text{validate}}| \in \{5, 10, 20\}$ and horizon lengths $T \in \{5, 17, 33, 65, 129\}$.

For the first three settings, we fix $|\mathcal{D}^{\text{test}}| = 100$ for evaluation. In the fourth setting, we assume there are $|\mathcal{D}^{\text{train}}|$ SKU’s (with distinct parameters) in total, and we set $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{validate}}$ and $\mathcal{D}^{\text{test}}$ to include 1, 1, and 100 trajectories per SKU, respectively (i.e., $|\mathcal{D}^{\text{test}}| = 100 \times |\mathcal{D}^{\text{train}}|$).

We compare policies $\pi^{\text{DRL,REG}}$ for combinations of $\text{DRL} \in \{\text{DDPG, PPO, DS}\}$ and $\text{REG} \in \{\text{NONE, BASE}\}$, omitting our COEFF regularizations because x_t is one-dimensional. We train, validate, and test using the same ℓ_{bh} metric, with $b = 0.9$ and $h = 0.1$.

We consider the performance of $\pi^{\text{DRL,REG}}$ relative to π^* , which we use to denote the optimal policy knowing the true trajectory distribution. For our AR(1) distribution, we approximate π^* by optimizing over the $\mathcal{D}^{\text{test}}$ dataset. In either case, we report

$$\text{Validation Loss Gap} := \frac{L_{\text{bh}}^{\text{validate}}(\pi^{\text{DRL,REG}})}{L_{\text{bh}}^{\text{test}}(\pi^*)} - 1 \quad \text{and} \quad \text{Testing Loss Gap} := \frac{L_{\text{bh}}^{\text{test}}(\pi^{\text{DRL,REG}})}{L_{\text{bh}}^{\text{test}}(\pi^*)} - 1.$$

The Testing Loss Gap is our main metric of interest, which is out-of-sample and always non-negative, although we also include the Validation Loss Gap, which is in-sample and can be negative. For each combination of DRL and REG, we report averages from running the hyperparameter tuning procedure three times, for each of Settings 1–3, in Figure 1.

Results. As seen in Figure 1, the BASE regularization improves the Testing Loss of the DRL method in 8 out of 9 cases (all except for DS in Setting 3), often significantly. Furthermore, Figure 2 shows the best-so-far performance¹ in Setting 1 as hyperparameter trials

¹ Technically this is Validation not Testing Loss, because we do not compute the latter during hyperparameter tuning.

However, Figure 1 showed that the distinction between Validation and Testing Loss is unimportant in Setting 1.

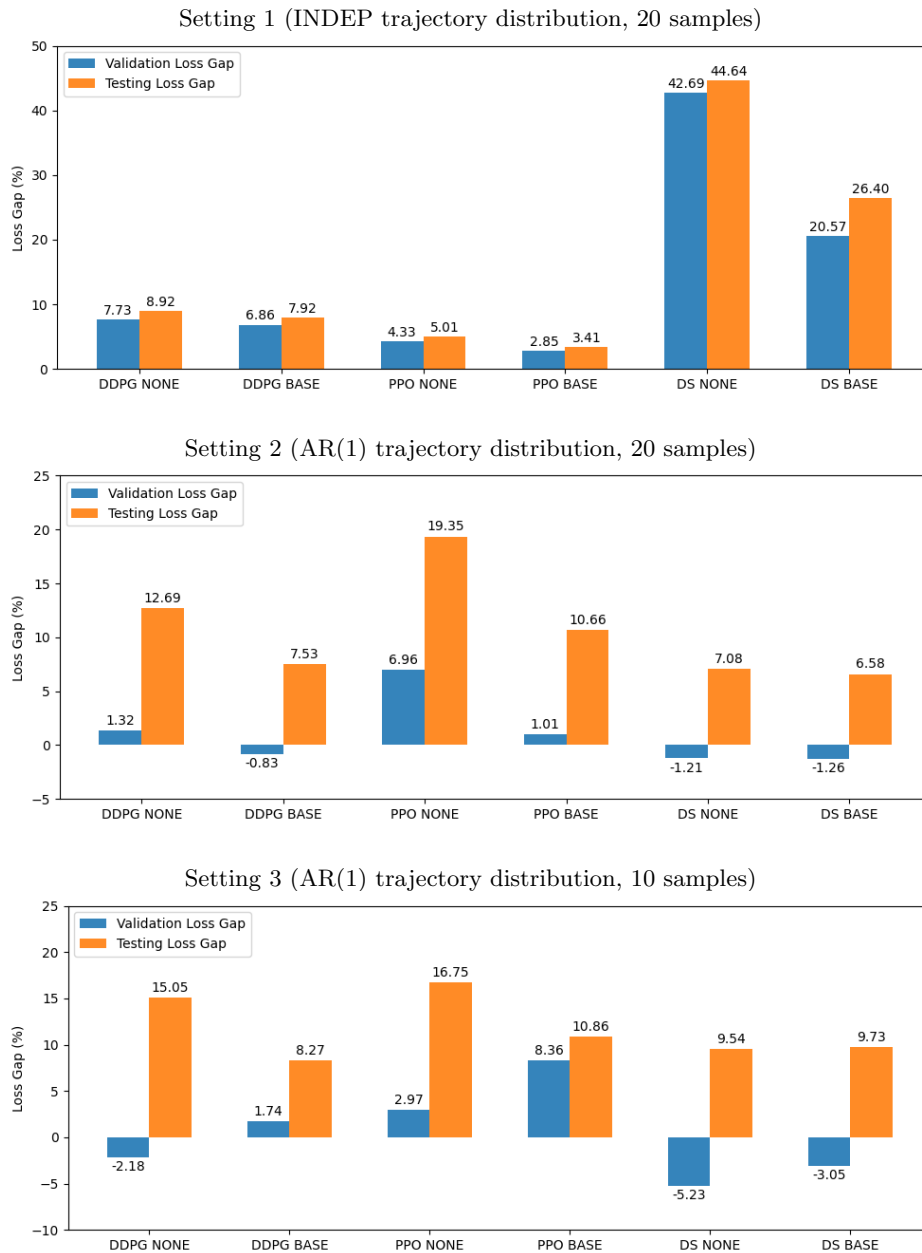


Figure 1 Validation and Testing Loss Gaps for the 6 combinations of DRL Method and Policy Regularization.

are being conducted, for each $\pi^{\text{DRL,REG}}$. We can see for all three DRL methods that the improvement from REG = NONE to REG = BASE is greater under limited hyperparameter search, corroborating our Takeaway I from the Introduction.

Still on Figure 2, we observe that without regularization, DDPG needs to search 17 configurations to surpass DS, while PPO needs to search 32. By contrast, with the BASE

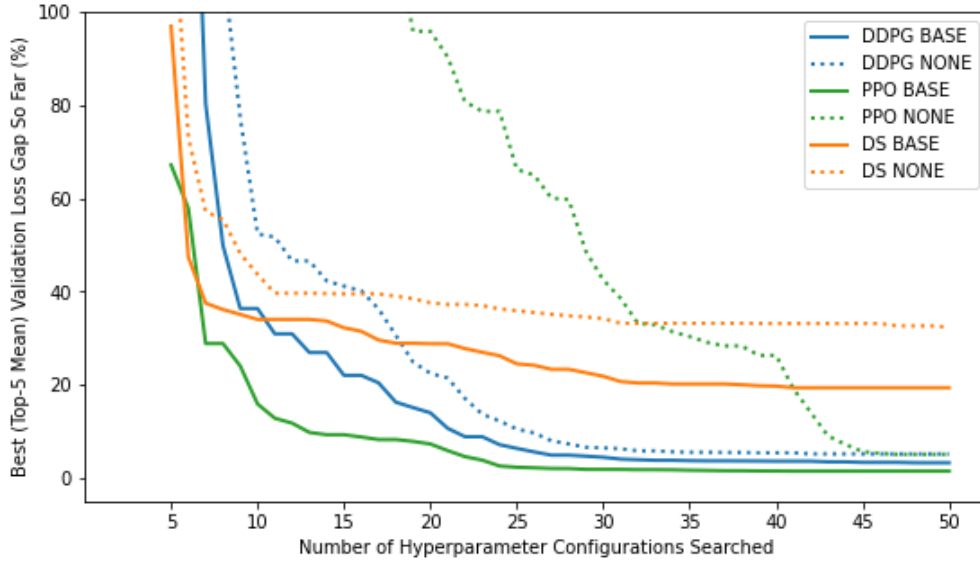


Figure 2 Validation Loss Gaps for the top-5 hyperparameter configurations, shown for each combination of DRL Method and Policy Regularization, in Setting 1.

regularization, these numbers reduce to 11 and 7, respectively. This suggests that policy regularization greatly reduces the hyperparameter tuning effort required for traditional DRL methods (DDPG, PPO) to outperform DS. Returning to Figure 1, we observe that regularization also improves the final Testing Loss (after hyperparameter search is finished) of DDPG and PPO by a greater amount than DS, and this can change whether DDPG or DS is best, in Setting 3. This corroborates our Takeaway II from the Introduction.

For a more nuanced reason why traditional DRL methods enjoy greater benefit from regularization, we consider DDPG as an example. DDPG minimizes the Bellman error of the fitted Q-function over historical transitions (s, a, r, s') , where s , a , and r denote the current state, action, and immediate loss, and s' is the next state. The Bellman error is the gap between $Q(s, a)$ and the target value, defined as the sum of r and the future loss $\min_{a'} Q(s', a')$. We note that the action a stored in each transition is the output of the policy network μ . That is, with the BASE regularization, the action produced by μ_{BASE} represents the target level of total inventory, whereas without regularization the action

corresponds to the final order quantity. Thus, under the BASE regularization, it becomes easier to approximate $\min_{a'} Q(s', a')$, because the best action a' is identical across different inventory states s' , leading to more stable learning of the Q-target.

To explain when DDPG and PPO beat DS, we compare results across different settings. In Setting 3 (with the BASE regularization), we can see that DS has better Validation Loss than DDPG but worse Testing Loss, suggesting that DS is overfitting. This occurs because DS takes policy gradients directly with respect to the trajectory losses in $\mathcal{D}^{\text{train}}$, forcing in-sample minimization. Moreover, since DS does not learn from (s, a, r, s') transitions, it lacks the cross-temporal learning that is central to traditional DRL methods and therefore DS is less sample-efficient. This overfitting seems to subside when the sample size increases from 10 to 20, as DS performs best in Setting 2.

To further investigate the learning behavior of DS, we conclude by training DS (with the BASE regularization) under different sample sizes and horizon lengths. In Figure 3, when the sample size is small ($|\mathcal{D}^{\text{train}}| = 5$), the testing-loss gap remains about 8% even at $T = 129$ under the simple IID distribution. This shows that increasing the horizon length T yields only limited improvement, highlighting the insufficiency of cross-temporal learning within DS. When $|\mathcal{D}^{\text{train}}|$ increases from 5 to 10 to 20, the testing losses improve overall across all horizon lengths, indicating that DS does benefit from meta-learning across trajectories (i.e., across SKU's), but nonetheless we do not see convergence for longer horizon lengths. As a final remark about overfitting, we find that with a limited number of samples, DS exhibits a large gap between validation and testing losses, demonstrating its tendency to overfit to idiosyncrasies in the observed trajectories.

Experiments on Alibaba's Offline Data

We show one edition of Alibaba's internal testing results on offline data. Here, $\mathcal{D}^{\text{train}}$ consists of 50,000 randomly-selected SKU trajectories from July to September 2024, while

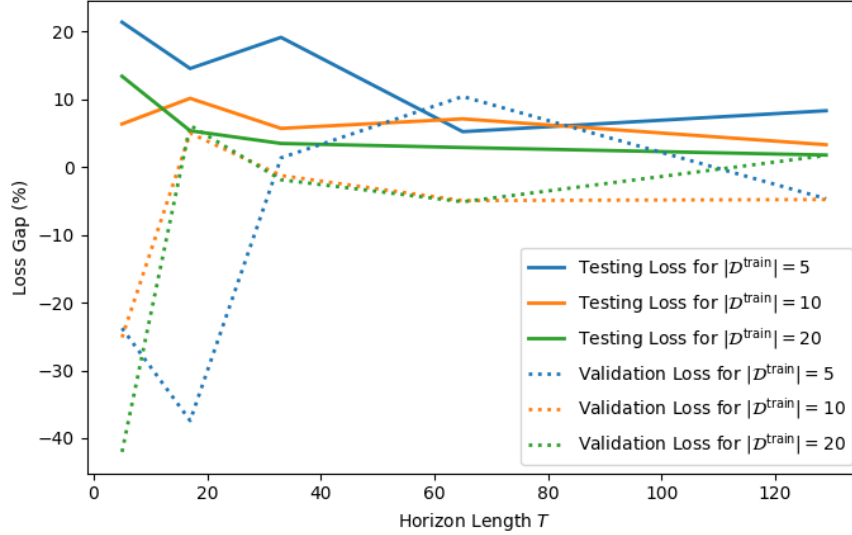


Figure 3 Testing and Validation Loss Gaps for DS with Base Regularization in Setting 4.

DRL Method	DDPG			DS			
	NONE	BASE	COEFF	NONE	BASE	COEFF	BOTH
$L_{SR}^{\text{test}}(\pi^{\text{DRL,REG}}) - L_{SR}^{\text{test}}(\pi^{\text{DDPG,BOTH}})$ (%)	10.10	6.03	4.41	2.10	2.18	1.74	1.91
$L_{TT}^{\text{test}}(\pi^{\text{DRL,REG}}) - L_{TT}^{\text{test}}(\pi^{\text{DDPG,BOTH}})$ (days)	6.13	6.46	-0.41	-1.25	-2.81	3.80	0.23

Table 1 Stockout Rates (reported in % of days) and Turnover Times (reported in days) on the test data, comparing all policies to $\pi^{\text{DDPG,Both}}$. Positive values mean worse than $\pi^{\text{DDPG,Both}}$; negative values mean better.

$\mathcal{D}^{\text{validate}}$ consists of 5,000 other SKU trajectories from the same horizon of $T = 90$ days. $\mathcal{D}^{\text{test}}$ consists of 5,000 randomly-selected SKU trajectories from February to April 2025, which is a chronologically-later horizon. SKU’s in the datasets have varying values for review period P and lead time L that are encoded into their context vectors x_t , which lie in \mathbb{R}^{190} .

We compare policies $\pi^{\text{DRL,REG}}$ for combinations of $\text{DRL} \in \{\text{DDPG}, \text{DS}\}$ and $\text{REG} \in \{\text{NONE}, \text{BASE}, \text{COEFF}, \text{BOTH}\}$, noting that Alibaba has stopped testing PPO at scale due to it generally performing worse than DDPG. The policies were trained using an objective where a 1% decrease in Stockout Rate is worth roughly a 2-day decrease in Turnover Time, and we report the metrics L_{SR}^{test} and L_{TT}^{test} in Table 1.

Results. As shown in Table 1, having BOTH regularizations produces by far the best version of DDPG, while having the BASE regularization produces the best version of DS. This supports our Takeaway I from the Introduction. To see the result of Takeaway II, note that without regularizations, $\pi^{\text{DDPG,NONE}}$ is much worse than $\pi^{\text{DS,NONE}}$; with regularizations, $\pi^{\text{DDPG,BOTH}}$ is comfortably better than $\pi^{\text{DS,BASE}}$ (and $\pi^{\text{DS,BOTH}}$), recalling that Stockout Rate is twice as important as Turnover Time.

Finally, related to Takeaway I, one may wonder whether the results in Table 1 are from “limited hyperparameter search”. To give some perspective, one hyperparameter trial for DDPG in this setting (with data size $\approx 50,000$ trajectories \times 90 days \times 190-dimensional features) takes about 24 wall-clock hours internally at Alibaba. Table 1 shows test results for every $\pi^{\text{DRL,REG}}$ after 10 sequential trials, with each trial choosing new hyperparameters based on the outcome of the previous trial, over 10 physical days. Although Alibaba could have potentially improved the hyperparameter search with some parallelization, which may decrease the improvement of $\pi^{\text{DDPG,BOTH}}$ over $\pi^{\text{DDPG,NONE}}$, this would only increase the advantage of DDPG over DS. Regardless, training is expensive and slow at this scale, and the results in Table 1 depict a realistic outcome for hyperparameter tuning in practice at Alibaba.

Real-world Deployment and Impact at Alibaba

We report deployment results observed in the real world. Unlike the offline evaluations presented in Table 1, in actual deployment we may have non-deterministic review periods and lead times. Another difference is that we encounter true demands in the real world, allowing us to stress-test the robustness of our offline training using inferred demands when there was a stockout. Finally, we note that the horizon length T used for training can be arbitrary, where each trained policy is executed in the real world until the next retraining. Alibaba aims to retrain every 1.5 months, using data from the most recent 90 days.

To get a sense of real-world scale, Alibaba takes full ownership of inventory and sales for over 100,000 SKU’s on its Tmall e-commerce platform. Inventory for these SKU’s is controlled separately at each of Tmall’s ≈ 20 primary warehouses, leading to over 1 million SKU-warehouse pairs that require periodic inventory decisions.

We distinguish between SKU’s sourced from domestic suppliers vs. SKU’s imported from international suppliers, because the incumbent inventory policies for domestic vs. international SKU’s were different. The incumbent inventory policies deployed by Alibaba at that time consisted of either DRL-based algorithms or the more traditional predict-then-optimize approach, in which a demand model is estimated from recent demand sequences and the optimal order quantity for each SKU is subsequently computed using standard optimization tools. We defer the details of the history of DRL implementation at Alibaba to Appendix B. The same predict-then-optimize approach can also differ in the depth of its search to manage computational cost. The incumbent international policies adopted more sophisticated search procedures that are computationally intensive, whereas the domestic counterparts used more naive but efficient search schemes to accommodate the much higher volume of domestic products.

Results 1: Difference-in-Differences (DiD) study in July 2024. An inventory policy trained through DRL with our policy regularizations was piloted on 10% of international SKU’s, cherry-picked to be problematic with great room for inventory improvement. We compare this 10% of SKU’s to the other 90% of international SKU’s before and after the deployment of our policy, and a simple DiD analysis reveals that the piloted SKU’s saw a 0.83% reduction in average Stockout Rate and a 9.53-day reduction in average Turnover Time. We note that this is a significant, Pareto improvement in both of our metrics.

SKU Category	A+	A	B	C	D	Z
International SKU's (change in days)	-1.36	-0.82	-1.02	-1.27	-	-
Domestic SKU's (change in days)	-4.04	-3.81	-2.93	-2.23	-2.13	-0.64

Table 2 Changes in average Turnover Time without changing average Stockout Rate. SKU's are categorized by sales volume from A+ (fastest-moving) to Z (long-tail). The time period is April 2025.

Results 2: major rollout in April 2025. An inventory policy trained through our DRL was rolled out to 100% of international SKU's and 87% of domestic SKU's. Although this major rollout impedes comparison to the incumbent algorithm via DiD, Alibaba conducted a careful counterfactual analysis that simulated the incumbent algorithm with the non-deterministic review periods and lead times actually encountered during April 2025.

We compare the actual metrics observed by our deployed policy to this counterfactual execution of the incumbent algorithm during April 2025. Averaging over international SKU's, our policy achieved a 1-day reduction in Turnover Time compared to the incumbent algorithm, with no distinguishable difference in Stockout Rate. Averaging over domestic SKU's, our policy achieved a 2-day reduction in Turnover Time, also with no distinguishable difference in Stockout Rate. A further breakdown by sales volume is shown in Table 2, where we note that these simple averages undersell our improvement in Turnover Time because the improvement is greatest on the A+ (highest-volume) SKU's. We also note that the improvement in domestic SKU's is greater partly because the incumbent algorithm is more naive, even though the improvement is not to the level of the international SKU's from the July 2024 DiD study that were cherry-picked.

This Pareto improvement of reducing Turnover Time without changing Stockout Rate also allows to estimate the financial benefit from the inventory reduction. Assuming an average inventory value of 5 billion RMB, Alibaba estimates an inventory reduction worth

350 million RMB, leading to annual savings in the cost of capital that amounts to approximately 13.3 million RMB.

Results 3: turnover during July–August 2025. Alibaba continued monitoring inventory performance after our major rollout of DRL, especially when the inventory policy has to be retrained. To this end, we show the average Turnover Time of international SKU’s during July–August 2025, a period that captures retrainings of the DRL policy, and compare it to the average Turnover Time of international SKU’s during the same period in 2024, before our major rollout of DRL. As seen from Figure 4, there is a massive 20% reduction in Turnover Time in 2025; it was also reported to us that Stockout Rates did not get noticeably worse. Although this comparison does not rigorously control for many other factors² that could have caused the change from 2024 to 2025, it provides high-level managerial confidence that our deployment of DRL is stable over time, even when the policy is retrained. As a result, our policy has been fully deployed as of October 2025 to 100% of the SKU-warehouse pairs (over 1 million) managed by Alibaba.

Conclusion

All in all, our paper demonstrates the power of a hybrid approach that integrates classical inventory heuristics from Operations Research to regularize policies learned by powerful modern DRL methods. Performance aside, we believe that these policy regularizations are essential for operationalizing DRL in practice, as they greatly reduce the computational burden of training and hyperparameter tuning, factors we have observed to be dealbreakers within companies. Our paper also presents a more detailed comparison of DRL algorithms with and without policy regularizations on synthetic data. Implementing a regularized

² We note that July and August are relatively calm months for Tmall without major promotional events that could have differential effects between 2024 and 2025, which is why this period was chosen for comparison.

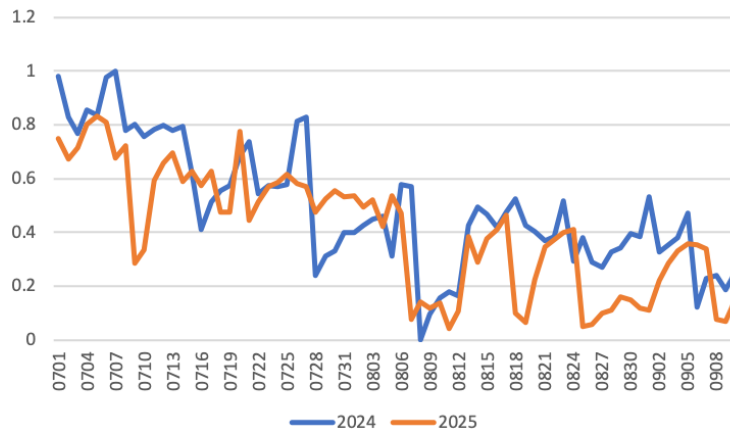


Figure 4 Evolution of average Turnover Time for international SKU's during July–August, in 2024 and 2025. All numbers are normalized relative to the maximum average Turnover Time encountered in either year.

DRL algorithm at Alibaba has had a tremendous impact, allowing inventory replenishment decisions to automatically respond to dynamic, high-dimensional demand signals while remaining interpretable and consistently achieving strong performance.

References

- J. Achiam. Spinning up in deep reinforcement learning, 2018. URL https://spinningup.openai.com/en/latest/spinningup/rl_intro.html.
- M. Alvo, D. Russo, and Y. Kanoria. Neural inventory control in networks via hindsight differentiable policy optimization. *arXiv preprint arXiv:2306.11246*, 2023.
- B. J. De Moor, J. Gijbrecchts, and R. N. Boute. Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. *European Journal of Operational Research*, 301(2):535–545, 2022.
- J. Gijbrecchts, R. N. Boute, J. A. Van Mieghem, and D. Zhang. Ai in inventory management: The disruptive era of drl and beyond. *Available at SSRN 5199616*, 2025.
- T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

- R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.
- T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- J. Liu, S. Lin, L. Xin, and Y. Zhang. Ai vs. human buyers: A study of alibaba’s inventory replenishment system. *INFORMS Journal on Applied Analytics*, 53(5):372–387, 2023.
- D. Madeka, K. Torkkola, C. Eisenach, A. Luo, D. P. Foster, and S. M. Kakade. Deep inventory management. *arXiv preprint arXiv:2210.03137*, 2022.
- A. Maggiar, S. Andaz, A. Bagaria, C. Eisenach, D. Foster, O. Gottesman, and D. Perrault-Joncas. Structure-informed deep reinforcement learning for inventory management. *arXiv preprint arXiv:2507.22040*, 2025.
- M. Qi, Y. Shi, Y. Qi, C. Ma, R. Yuan, D. Wu, and Z.-J. Shen. A practical end-to-end inventory management model with deep learning. *Management Science*, 69(2):759–773, 2023.
- A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015a.
- J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.
- J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- L. V. Snyder and Z.-J. M. Shen. *Fundamentals of supply chain theory*. John Wiley & Sons, Hoboken, NJ, second edition, 2019.
- Y. Xie, W. Ma, and L. Xin. Vc theory for inventory policies. *available on SSRN 4794903*, 2025.

Appendix A: Synthetic Experiments Implementation and Hyperparameter Details

Our DDPG and PPO code is based on the open-source Stable-Baselines3 implementation (Raffin et al. 2021), and our DS code is based on the open-source implementation of Alvo et al. (2023). Both are implemented in PyTorch. The synthetic experiments were conducted on a rented Runpod instance equipped with one NVIDIA RTX A4500 GPU, 12 vCPU’s (AMD EPYC 7352), and 62 GB of RAM.

During training, DDPG and PPO are evaluated on $\mathcal{D}^{\text{validate}}$ every `eval_freq` steps. Training is stopped if the validation loss $L_{\text{bh}}^{\text{validate}}$ does not improve over the last `patience` evaluations or if the total number of training steps exceeds `total_steps`; in either case, the most recent policy is saved. For DS, evaluation on $\mathcal{D}^{\text{validate}}$ is performed once per epoch, and training is stopped if $L_{\text{bh}}^{\text{validate}}$ does not improve over the last `patience` evaluations or if the total number of training epochs exceeds `epochs`.

All algorithms are trained using the Adam optimizer with an initial learning rate `learning_rate`. To improve convergence, we adopt a learning-rate scheduler for each algorithm. For DDPG and PPO, the scheduler follows the Stable-Baselines3 function `get_linear_fn(start, end, end_fraction)` with `start = learning_rate`, `end = lr_min`, and `end_fraction = lr_fraction`. We also schedule the clip range for PPO using `get_linear_fn(start, end, end_fraction)`, where `start = clip_range`, `end = 0.5 × clip_range`, and `end_fraction = lr_fraction`. For DS, we use the PyTorch `ReduceLROnPlateau` scheduler with `patience = 3` and `factor = scheduler_factor`, keeping all other inputs at their default values.

For consistency, we use the following procedure to tune hyperparameters in all experiments on synthetic data. We use the Ray Tune package (Liaw et al. 2018) to intelligently search over a hyperparameter grid (see details in Tables 3 to 5) and identify the configuration that minimizes validation loss. The search uses Bayesian optimization with a

Tree-structured Parzen Estimator, evaluating 50 configurations in the first stage. In the second stage, the five best configurations from stage one are each re-run across five random seeds, and the final winner is selected based on median performance across these seeds, using the stage-one configuration with the lowest average validation loss. This two-stage procedure helps mitigate overfitting and ensures robust hyperparameter selection for evaluation.

We specify the hyperparameter values or search ranges used in each setting in Tables 3 to 5, where *tune.choice*, *tune.uniform*, and *tune.loguniform* are Ray Tune sampling functions that draw uniformly from a discrete set, a continuous interval, and a continuous interval on the log scale, respectively. We use bold text to indicate hyperparameters that are built-in inputs to the Stable-Baselines3 implementations of DDPG and PPO, while all remaining built-in hyperparameters follow their default values (see <https://stable-baselines3.readthedocs.io/en/master/>). In addition to `eval_freq`, `patience`, `total_steps`, `epochs`, `learning_rate`, `lr_min`, `lr_fraction`, and `scheduler_factor`, our self-defined hyperparameters also include `max_replenish` and `initial_action_bias`. They ensure that both the action space and the final order quantity are upper bounded by `max_replenish`, and that the initial bias of the policy network equals `initial_action_bias` \times `max_replenish`.

Trajectory Distributions and Details of Synthetic Results

We describe the three types of trajectory distributions as follows:

1. Independent but non-identically distributed demands (INDEP distribution): for all $(|\mathcal{D}^{\text{train}}| + |\mathcal{D}^{\text{validate}}| + |\mathcal{D}^{\text{test}}|)$ SKU's, demand trajectories are IID drawn from the same distribution. At time t , the demand d_t is independently drawn from a discrete uniform distribution $U\{b_t, b_t + 1, \dots, b_t + 4\}$. Here, the lower bound $b_t \in \mathbb{Z}_{\geq 0}$ is generated from a truncated normal distribution $\lfloor \max\{0, \mathcal{N}(\mu, \sigma^2)\} \rfloor$ with $\mu = 10$ and $\sigma = 4$.

	INDEP		AR(1)
policy	'MlpPolicy'		
learning_rate = <i>get_linear_fn</i>	learning_rate	<i>tune.loguniform</i> (3e-4, 9e-3)	<i>tune.loguniform</i> (6e-4, 9e-3)
	lr_min	<i>tune.loguniform</i> (6e-5, 3e-4)	<i>tune.loguniform</i> (9e-5, 6e-4)
	lr_fraction	0.95	
batch_size	<i>tune.choice</i> ([128, 256, 512])		
tau	<i>tune.loguniform</i> (1e-3, 1e-2)		
gamma	<i>tune.uniform</i> (0.98, 1)		<i>tune.uniform</i> (0.96, 1)
policy_kwargs = dict[net_arch]	'pi': [64, 64], 'qf': [64, 64]		'pi': [32, 32], 'qf': [32, 32] for Setting 2 'pi': [32, 32], 'qf': [16, 16] for Setting 3
buffer_size	<i>tune.choice</i> ([20,000, 50,000, 100,000])		
train_freq	<i>tune.choice</i> ([1, 14])		
learning_starts	<i>tune.choice</i> ([100, 500, 1000])		
total_steps	200,000		
eval_freq	<i>tune.choice</i> ([512, 1024])		
patience	10		
max_replenish	100		
initial_action_bias	<i>tune.uniform</i> (0.2, 0.7)		<i>tune.uniform</i> (0, 0.6)

Table 3 Hyperparameter values or ranges of DDPG.

- Autoregressive demands (AR(1) distribution): for each SKU $i \in \{1, \dots, |\mathcal{D}^{\text{train}}| + |\mathcal{D}^{\text{validate}}| + |\mathcal{D}^{\text{test}}|\}$, the demands $(d_t^i)_{t=1}^T$ follow a truncated AR(1) process. Specifically, $d_t^i = \max\{0, \phi^i d_{t-1}^i + (1 - \phi^i) \bar{d}^i + \epsilon_t^i\}$, where the initial demand d_0^i is drawn from $\max\{0, \mathcal{N}(\bar{d}^i, (\sigma^i)^2)\}$. Here, the coefficient ϕ^i , base demand \bar{d}^i , and noise standard deviation σ^i are drawn from uniform distributions $U[0.3, 0.9]$, $U[5, 10]$ and $U[2, 8]$, respectively, and ϵ_t^i follows $\mathcal{N}(0, (\sigma^i)^2)$.
- Independent and identically distributed demands (IID distribution): for each SKU $i \in \{1, \dots, |\mathcal{D}^{\text{train}}|\}$, the demand d_t^i at every time period t is independently drawn from

	INDEP		AR(1)
policy	‘MlpPolicy’		
learning_rate = <i>get_linear_fn</i>	learning_rate	<i>tune.loguniform</i> (5e-5, 9e-3)	<i>tune.loguniform</i> (6e-4, 1e-2)
	lr_min	<i>tune.loguniform</i> (1e-5, 5e-5)	<i>tune.loguniform</i> (9e-5, 6e-4)
	lr_fraction	0.9	
clip_range = <i>get_linear_fn</i>	clip_range	<i>tune.uniform</i> (0.1, 0.3)	
	lr_fraction	0.9	
batch_size	<i>tune.choice</i> ([128, 256, 512])		
n_steps	<i>tune.choice</i> ([1024, 2048, 4096])		<i>tune.choice</i> ([512, 1024, 2048])
ent_coef	<i>tune.uniform</i> (1e-4, 5e-1)		
gamma	<i>tune.uniform</i> (0.98, 1)		<i>tune.uniform</i> (0.96, 1)
gae_lambda	<i>tune.uniform</i> (0.5, 1)		<i>tune.uniform</i> (0, 0.5) for BASE <i>tune.uniform</i> (0.5, 1) for NONE
policy_kwargs = dict[net_arch]	‘pi’: [256, 256], ‘vf’: [256, 256]		‘pi’: [128, 128], ‘vf’: [64, 64] for Setting 2 ‘pi’: [64, 64], ‘vf’: [64, 64] for Setting 3
vf_coef	0.4		
total_steps	200,000		
eval_freq	<i>tune.choice</i> ([512, 1024, 2048])		<i>tune.choice</i> ([1024, 2048])
patience	10		
max_replenish	100		
initial_action_bias	<i>tune.uniform</i> (0.2, 0.7)		<i>tune.uniform</i> (0, 0.6)

Table 4 Hyperparameter values or ranges of PPO.

a truncated normal distribution $\max\{0, \mathcal{N}(100, (\sigma^i)^2)\}$, where σ^i is drawn from a uniform distribution $U[5, 20]$. Note that the feature x_t^i of SKU i here is set to be the parameter σ^i for each t .

The absolute values of the validation and testing losses corresponding to the percentage loss gaps in Figures 1 and 3 are reported in Tables 6 to 9. The loss gaps are computed

	INDEP	AR(1)	IID
policy	'MlpPolicy'		
neurons_per_hidden_layer	tune.choice([[32, 32], [64, 64], [128, 128]])	tune.choice([[16, 16], [32, 32], [64, 64]])	
inner_layer_activation	'elu'		
output_layer_activation	'relu'		
batch_size	<i>tune.choice</i> ([5, 10])	<i>tune.choice</i> ([5, 10, 20])	
learning_rate	<i>tune.loguniform</i> (4e-4, 1e-1)		
scheduler_factor	<i>tune.uniform</i> (0.5, 0.95)	0.5	
epochs	3000		
patience	30		
max_replenish	100	500	
initial_action_bias	<i>tune.uniform</i> (0.2, 0.7)	<i>tune.uniform</i> (0, 0.6)	

Table 5 Hyperparameter values or ranges of DS.

relative to the benchmark $L_{\text{bh}}^{\text{test}}(\pi^*)$ obtained by solving or approximating the optimal policy π^* with knowledge of the true demand distribution. For the INDEP distribution, it is well known that the optimal policy belongs to the class of time-dependent base-stock policies (see, e.g., Snyder and Shen 2019), and we compute it exactly via dynamic programming. For the AR(1) distribution, the optimal loss is approximated by training DS on the testing dataset $\mathcal{D}^{\text{test}}$. For the IID distribution, it is well known that the optimal policy lies within the class of stationary base-stock policies (see, e.g., Snyder and Shen 2019), and we compute the optimal base-stock level exactly via the critical fractile formula.

DRL Method	DDPG		PPO		DS	
Policy Regularization	NONE	BASE	NONE	BASE	NONE	BASE
Validation Loss	0.9340	0.9264	0.9045	0.8917	1.2371	1.0454
Validation Loss Gap (%)	7.73	6.86	4.33	2.85	42.69	20.57
Testing Loss	0.9443	0.9356	0.9104	0.8966	1.2540	1.0959
Testing Loss Gap (%)	8.92	7.92	5.01	3.41	44.64	26.40

Table 6 Validation and Testing Loss and corresponding Loss Gaps relative to the benchmark loss of 0.8670 in Setting 1.

DRL Method	DDPG		PPO		DS	
Policy Regularization	NONE	BASE	NONE	BASE	NONE	BASE
Validation Loss	2.3139	2.2647	2.4428	2.3069	2.2561	2.2550
Validation Loss Gap (%)	1.32	-0.83	6.96	1.01	-1.21	-1.26
Testing Loss	2.5735	2.4557	2.7257	2.5273	2.4455	2.4342
Testing Loss Gap (%)	12.69	7.53	19.35	10.66	7.08	6.58

Table 7 Validation and Testing Loss and corresponding Loss Gaps relative to the benchmark loss of 2.2838 in Setting 2.

Appendix B: History of DRL Implementations at Alibaba

Alibaba has been implementing RL methods on a subset of its product portfolio since 2020, as documented in Liu et al. (2023). The earlier version of the RL algorithm was not general-purpose, requiring similar products to be grouped first before training separate models for each group. This grouping was necessary because different products exhibit significant variations in business objectives, demand patterns, and supply constraints. For example, in terms of business objectives, bestsellers prioritize stock availability, whereas long-tail products emphasize inventory turnover; on the supply side, fulfillment capabilities

DRL Method	DDPG		PPO		DS	
Policy Regularization	NONE	BASE	NONE	BASE	NONE	BASE
Validation Loss	2.2341	2.3236	2.3515	2.4747	2.1643	2.2141
Validation Loss Gap (%)	-2.18	1.74	2.97	8.36	-5.23	-3.05
Testing Loss	2.6275	2.4727	2.6664	2.5319	2.5017	2.5060
Testing Loss Gap (%)	15.05	8.27	16.75	10.86	9.54	9.73

Table 8 Validation and Testing Loss and corresponding Loss Gaps relative to the benchmark loss of 2.2838 in Setting 3.

vary across merchants, with substantial differences in restocking frequency, delivery times, and minimum order quantities. Moreover, cross-group transfer from bestsellers to long-tail products is challenging, because the long-tail group suffers from data sparsity and poor generalization. These challenges necessitate maintaining independent feature pipelines, actor/critic networks, and hyperparameter tuning processes for each group, which significantly increases engineering overhead and limits the scalability of RL across all products. Therefore, a significant portion of the portfolio at the time relied on a more traditional predict-then-optimize approach, which was substantially easier to maintain. While an RL model only required retraining every one to two months, each retraining cycle was time-consuming especially in a highly dynamic environment where feature updates were frequent. In contrast, predict-then-optimize models were more lightweight operationally. To illustrate the difference in maintenance burden: RL deployment required a dedicated team of algorithm scientists, whereas predict-then-optimize models could be maintained by a team of engineers plus one or two scientists.

Compared to the earlier version, the new version of RL (as discussed in this paper) is general-purpose: a single model applies across all products, eliminating the need to

		$T = 5$	$T = 17$	$T = 33$	$T = 65$	$T = 129$
$ \mathcal{D}^{\text{train}} = 5$	Benchmark Loss	2.3669	2.3726	2.3792	2.3884	2.3764
	Validation Loss	1.8045	1.4850	2.4117	2.6374	2.2657
	Validation Loss Gap (%)	-23.76	-37.41	1.37	10.43	-4.66
	Testing Loss Gap	2.8733	2.7180	2.8351	2.5135	2.5745
	Testing Loss Gap (%)	21.40	14.56	19.16	5.24	8.33
$ \mathcal{D}^{\text{train}} = 10$	Benchmark Loss	2.2569	2.2672	2.2712	2.2728	2.2640
	Validation Loss	1.6897	2.3809	2.2431	2.1605	2.1559
	Validation Loss Gap (%)	-25.13	5.02	-1.23	-4.94	-4.77
	Testing Loss	2.4006	2.4975	2.4011	2.4349	2.3389
	Testing Loss Gap (%)	6.37	10.16	5.72	7.13	3.31
$ \mathcal{D}^{\text{train}} = 20$	Benchmark Loss	2.0932	2.1064	2.1011	2.0999	2.1003
	Validation Loss	1.2135	2.2363	2.0620	1.9926	2.1363
	Validation Loss Gap (%)	-42.02	6.17	-1.86	-5.11	1.72
	Testing Loss	2.3741	2.2194	2.1747	2.1607	2.1378
	Testing Loss Gap (%)	13.42	5.37	3.50	2.90	1.79

Table 9 Validation and Testing Loss and corresponding Loss Gaps for DS with Base regularization in Setting 4.

train separate models. This is enabled by the idea of policy regularizations, supported by several engineering components, including (1) integrating with the operational system so that business objectives (e.g., service levels, inventory turnover) set on the system side are automatically reflected as reward weights, (2) enriching state features to capture product, supplier, and sales channel diversity, (3) scaling to larger-scale deep networks (up to billions of parameters) for robust high-dimensional mappings, and (4) shifting from manual hyperparameter tuning to automated Taguchi experiments.