Neural Inventory Control in Networks via Hindsight Differentiable Policy Optimization

Matias Alvo
Graduate School of Business, Columbia University, malvo26@gsb.columbia.edu

Daniel Russo
Graduate School of Business, Columbia University, djr2174@gsb.columbia.edu

Yash Kanoria
Graduate School of Business, Columbia University, yk2577@gsb.columbia.edu

We argue that inventory management presents unique opportunities for reliably applying and evaluating deep reinforcement learning (DRL). Toward reliable application, we emphasize and test two techniques. The first is Hindsight Differentiable Policy Optimization (HDPO), which performs stochastic gradient descent to optimize policy performance while avoiding the need to repeatedly deploy randomized policies in the environment—as is common with generic policy gradient methods. Our second technique involves aligning policy (neural) network architectures with the structure of the inventory network. Specifically, we focus on a network with a single warehouse that consolidates inventory from external suppliers, holds it, and then distributes it to many stores as needed. In this setting, we introduce the symmetry-aware policy network architecture. We motivate this architecture by establishing an asymptotic performance guarantee and empirically demonstrate its ability to reduce the amount of data needed to uncover strong policies. Both techniques exploit structures inherent in inventory management problems, moving beyond generic DRL algorithms. Toward rigorous evaluation, we create and share new benchmark problems, divided into two categories. One type focuses on problems with hidden structures that allow us to compute or bound the cost of the true optimal policy. Across four problems of this type, we find HDPO consistently attains near-optimal performance, handling up to 60-dimensional raw state vectors effectively. The other type of evaluation involves constructing a test problem using real time series data from a large retailer, where the optimum is poorly understood. Here, we find HDPO methods meaningfully outperform a variety of generalized newsvendor heuristics. Our code can be found at `https://github.com/MatiasAlvo/Neural_inventory_control`.

Key words: deep reinforcement learning; inventory theory and control; supply chain management

1. Introduction

Inventory management deals with designing replenishment policies that minimize costs related to holding, selling, purchasing, and transporting goods. Such problems are typically modelled as Markov Decisions Processes (MDPs), but the triple curse of dimensionality (Powell 2007) (*i.e.*,

exponentially large state, action, and outcome spaces) usually renders them computationally intractable. Since the 1950s, researchers have attempted to circumvent this curse by formulating specialized models in which the optimal policy has a simple structure (Arrow et al. 1958, Scarf et al. 1960, Clark and Scarf 1960, Federgruen and Zipkin 1984b). These policies then form the basis of simple heuristics that work well in slightly broader – but still very narrow – problem settings (see *e.g.*, Federgruen and Zipkin 1984a, Xin 2021). Practical inventory management problems are often far more complex due to the presence of multiple inventory storage locations, dynamic demand patterns, and complexities in customer behavior during stock-outs.

Deep reinforcement learning (DRL) is a promising alternative for addressing the complexities of practical inventory management problems. These techniques aim to "learn" effective policies through interactions with a simulator grounded in real data. Rather than using artful model abstractions to simplify the optimal policy, practitioners employing DRL would emphasize the fidelity of a simulator. Instead of being limited by the ingenuity of a human policy designer, the solution can be improved by increasing computation – *i.e.*, by increasing network size, training time and data. In recent years, this approach has produced impressive successes in arcade games (Mnih et al. 2015), board games (Silver et al. 2017), and robotics (Levine et al. 2016). Recent works have used DRL in queuing network control (Dai and Gluzman 2022), ride-hailing (Feng et al. 2021, Oda and Joe-Wong 2018, Tang et al. 2019) and inventory management (Gijsbrechts et al. 2021), but applications to logistics still remain limited.

Despite the promises of DRL, two factors have slowed its adoption for addressing complex logistics problems. One factor is the differences in academic culture. Like other areas of machine learning, the DRL literature has typically validated heuristic policies by comparing them against each other, or against human experts. Operations research has traditionally focused on comparison to the true optimum. Furthermore, in contrast to machine learning, where standardized benchmarks such as ImageNet Krizhevsky et al. (2012) serve as widely accepted datasets and tasks for evaluating various approaches, inventory management lacks such universally recognized benchmarks against which numerous policies can be rigorously tested.

A more important factor is that generic DRL techniques can be unreliable. Consider traditional policy gradient methods like the REINFORCE (Williams 1992) algorithm and its variants. These methods adjust the parameters of a neural network policy that maps states to actions, searching for parameters that minimize total cost incurred throughout the planning horizon. The use of randomized policies and a clever log-derivative trick enable the REINFORCE algorithm to conduct unbiased policy gradient estimation even if transition and cost functions are unknown. This trick is no panacea, however, and more advanced policy gradient ideas are commonplace in successful applications; these include natural gradient methods to deal with ill-conditioning (Kakade 2001),

entropy regularization to prevent premature convergence to nearly deterministic policies (Haarnoja et al. 2018), baselines to help reduce variance (Greensmith et al. 2004), and actor-critic methods to further reduce variance (at the expense of bias) (Konda and Tsitsiklis 1999). Still, many "tricks" (Huang et al. 2022) and careful hyperparameter choices (Henderson et al. 2018) are required. Skeptics have even argued that methods like REINFORCE are little better than random search (Mania et al. 2018). A fundamental limitation of these algorithms is that they are entirely generic, exploiting no structure of the problem at hand. The usual challenges of reliably applying DRL are exacerbated in many logistics problems, due to high variance and exponentially large action spaces (see Dai and Gluzman 2022, for a discussion). Therefore, we expect that reliable use of DRL in logistics problems will require some tailoring of generic algorithms to exploit the structure of those problems.

We argue that several problem classes within inventory management present unique opportunities for reliably applying and rigorously evaluating DRL. Toward reliable application, we emphasize and evaluate two techniques. The first is a direct method to optimize the parameters of a neural network policy, which we call *hindsight differentiable policy optimization* (HDPO). The second involves using policy (neural) network architectures that mirror inventory network structure. Focused on a network with a single warehouse that consolidates inventory from external suppliers, holds it, and then distributes it to many stores as needed, we introduce the *symmetry-aware policy network architecture* and demonstrate its ability to reduce the amount of data needed to uncover strong policies. Both techniques move away from completely generic DRL algorithms toward methods that leverage some of the structures that are common in inventory management problems. Toward rigorous evaluation, we create and share new types of benchmark problems, divided into two categories. One type focuses on problems with hidden structures that allow us to compute or bound the cost of the true optimal policy. The other type involves constructing a test problem using real time series data from a large retailer, where the optimum is poorly understood. Below, we provide detailed discussions of these contributions.

1.1. Inventory management with continuous costs

Our model treats ordering quantities as continuous variables and interprets costs as continuous functions of these quantities, facilitating efficient gradient-based policy optimization. This model departs from "textbook" inventory management formulations, which aim to optimize ordering decisions for a single product in settings associated with substantial fixed costs. These fixed costs introduce discontinuities in the overall cost structure, prompting the adoption of classic (s, S) policies that recommend waiting until inventory has significantly depleted before replenishing.

Our modeling is driven by the operational realities of large retailers who manage shipments of a diverse assortment of products on a fixed delivery schedule. Viewing single-product inventory management as a simplification of this complex setting, we naturally exclude the fixed costs associated with overall shipments when optimizing inventory orders for individual products.

1.2. Toward more reliable application of DRL in inventory management

Here we elaborate on two techniques that, based on our experiments, improve the performance of DRL in inventory management problems.

Hindsight differentiable policy optimization (HDPO). HDPO should be viewed as an alternative to the REINFORCE method and its variants, which attempt to optimize policy parameters by stochastic gradient ascent even though the transition dynamics and cost functions are entirely unknown, and costs are not smooth in actions (or actions are discrete). HDPO exploits two model features to optimize policy parameters more effectively and reliably. *First, we are able to backtest the performance of any policy under consideration.* In most inventory models, for instance, it is well understood how states evolve under given realizations of demand and shipping lead times: inventory levels increase with the arrival of goods and decrease when items are sold or shipped. Assuming, as is canonically done in the literature, that demand and lead times are exogenous of ordering decisions, one can evaluate how any policy *would have* performed in a historical scenario (*i.e.*, given a sample path of demands and lead times). This structure is common to many problems arising in operations research (Powell 2007) and, when present, allows practitioners to evaluate policy performance with high fidelity. *Second, under appropriate modeling choices, we can evaluate the gradient of total cost incurred on a given historical scenario with respect to inventory ordering decisions, enabling especially efficient policy search (Glasserman and Tayur 1995).* In doing this we optimize order quantities as if they were continuous; some of our numerical experiments evaluate performance when decisions are rounded to integral values at implementation time. These two structures enable one to optimize inventory control policies by performing direct gradient-based policy search on the average cost incurred in a backtest.

It is worth noting that Madeka et al. (2022) utilizes HDPO (referred to as "DirectBackprop") in their recent study of inventory management at Amazon. We became aware of their work after our numerical experiments were already in progress, and have made complementary contributions to theirs. We discuss the overlap and differences in Section 1.4. We view the success of HDPO in practice, at the scale of Amazon, as a major piece of evidence pointing to the promise of HDPO.

Symmetry aware policy network architectures. Many inventory management problems involve rich network structure. The following situation arises frequently. Inventory can be ordered from manufacturers, but long lead times make forecasting difficult. Inventory can be held at "stores"

that sell directly to customers, but stores have high holding costs (reflecting, *e.g.*, expensive real estate and limited shelf space). Instead, most inventory is ordered to a centralized warehouse(s), which holds inventory at lower cost and, as a result of "pooling" demand across stores, can forecast more accurately. Inventory is shipped from the warehouse to stores as needed. Stores impact each other, since they draw on the same pool of inventory, but the coupling between them is "weak" (Meuleau et al. 1998).

A *vanilla* policy network architecture for such a problem is a fully connected NN that maps the full state of the system – information needed to forecast demand, inventory levels at various locations, inventory ordered but yet to be delivered, etc. – to a complex system-level decision vector. We introduce a natural policy network architecture which better reflects the structure of a class of network inventory problems in which different locations are linked through weak coupling constraints. The architecture we propose has two ingredients: (i) a NN for each location, with sharing of NN weights between "sibling" locations, *e.g.*, in the setting we study, different stores which are supplied by the same warehouse share network weights, (ii) a Context Net that takes in the raw state as input and generates a "context" vector as output. This context acts as a compressed summary of the overall state of the system. Using this context along with the local state of each location, a Warehouse Net and a Store Net (one for each store) generate intermediate outputs that lead to ordering decisions. Note that the overall neural network consisting of the Context Net, Warehouse Net, and Store Nets can be viewed as one single NN with substantial weight-sharing. We motivate this architecture by proving, in a specific setting, that even a small (fixed-size) context dimension enables optimal performance in an asymptotic regime in which the number of stores grows.

1.3. Main empirical findings

The majority of the paper is dedicated to careful, reproducible evaluation of the techniques described above. We have three main findings.

1. ***HDPO recovers near-optimal policies in problems with hidden structure:*** Section 3 evaluates HDPO applied to a set of problems where hidden structure — identified by ingenious researchers — allows the optimal cost to be computed or bounded. Such problems provide a rare opportunity to benchmark DRL methods against the true optimum, rather than against each other. On these problems, we apply HDPO to optimize over NN policies that map "raw" state vectors to (typically vector-valued) actions, without modifications that reflect each problem's unique hidden structure. Nevertheless, HDPO consistently converges to (essentially) optimal policies, achieving average optimality gaps below 0.03%, 0.25%, 0.35%, and 0.15% in each of the four settings we considered (see Table 1 in Section 3.3). Careful comparisons to the

results reported in Gijsbrechts et al. (2021) suggest that generic, black-box DRL algorithms are far less reliable. For instance, they report optimality gaps of 3.0% to 6.7% across six fairly small problems; this level of performance requires selecting the best run among 250 hyperparameter choices for each instance. By contrast, HDPO attains optimality gaps that are as low as detectable[1] across all six instances for each of twelve different combinations of hyperparmeter choices (see Table 3 in Section 3.3.1).

2. ***Symmetry-aware policy network architectures enhance sample efficiency:*** Section 4 focuses on the sample efficiency of HDPO. Our main hypothesis is that reflecting inventory network structure within the policy network design can enhance sample efficiency. We focus on comparisons between the vanilla and symmetry-aware policy networks when in a setting with a single warehouse which ships inventory to many stores. A demand "sample" is a time series of demand observations for a single product at each of the stores. The benefits of the symmetry-aware architecture are striking. A symmetry-aware policy network trained with only 4 samples has stronger out-of-sample performance than its "vanilla" counterpart trained with 256 samples, across store quantities ranging from 20 to 50 (see Figure 4).
3. ***HDPO outperforms generalized newsvendor heuristics with real time series data:*** To further evaluate HDPO, we construct a new benchmark problem out of publicly available sales data shared by Corporación Favorita, one of Ecuador's largest grocery retailers. This setting requires grappling with realistic nonstationarity and the optimal policy cannot easily be characterized. HDPO addresses nonstationarity by optimizing the hindsight performance of policies that map available information — including a historical window of demand observations and outstanding inventory orders — directly to a decision. This approach is compared to a several newsvendor type policies. Our implementations address nonstationarity in the data by training a NN to forecast the distribution of cumulative demand over item's lead time. We implement several methods for making decisions based on the quantiles of the demand forecast, tuning them to minimize the cost they incur. Nevertheless, our findings indicate that HDPO consistently outperforms these generalized newsvendor policies in a lost demand model and points to comprehensible reasons those heuristics may perform poorly.

1.4. Further related literature

Inventory management. The field of inventory management boasts a rich and extensive history. In seminal research, Arrow et al. (1958) demonstrated that in a setting involving a single location with positive vendor lead times and assuming backlogged demand (*i.e.*, customers are

[1] These settings rely on optimal cost values reported in Zipkin (2008), which are rounded to the second decimal place.

willing to wait for unavailable items), an optimal one-dimensional base-stock policy exists. Specifically, they illustrated that the state, which initially has the same dimension as the lead time, can be condensed into a single parameter. Conversely, in situations where demand is considered lost (*i.e.*, customers do not wait for unavailable products), the optimal policy may rely on the entire inventory pipeline, encompassing the quantity of units arriving each day until the lead time. This assumption may better reflect reality, particularly in brick-and-mortar retail, where only 9-22% of customers are inclined to postpone their purchases when a specific item is unavailable in-store (Corsten and Gruen 2004). Zipkin (2008) highlighted that a base-stock policy may perform poorly in such contexts. Given the computational complexity associated with solving this problem, numerous heuristics have been proposed, including myopic policies (Morton 1971, Zipkin 2008), constant-order policies (Reiman 2004), capped base-stock policies (Xin 2021), and linear programming-based techniques (Sun et al. 2014).

In multi-echelon inventory networks, inventory can be held across numerous locations. Clark and Scarf (1960) demonstrated that within a serial system, where inventory progresses linearly through sequential locations, a simple extension of base-stock policies proves to be optimal under a backlogged demand assumption. Similarly, Federgruen and Zipkin (1984a) introduced a well-performing policy for a setting in which a central "transshipment" warehouse, unable to hold inventory, supplies multiple stores. Yet, the task of determining the optimal policy, even in some straightforward inventory network setups, remains daunting. For a comprehensive review of works addressing multi-echelon inventory problems, we direct readers to De Kok et al. (2018).

Deep reinforcement learning algorithms. To describe the literature, we will differentiate between two types of algorithms. Similar to value-iteration methods in dynamic programming, Q-learning (Watkins and Dayan 1992) makes iterative updates to an estimate of the optimal state-action value function (usually denoted as Q^*). DRL implementations employ neural networks to approximate the value function. Influential work by (Mnih et al. 2015) introduces a variant of this idea called deep Q-networks (DQN).

Closer in spirit to policy-iteration methods in dynamic programming, policy gradient methods make iterative improvements to an initial policy. The REINFORCE algorithm by (Williams 1992) relies on the use of randomized policies and inverse propensity weighting to produce unbiased policy gradients. See Appendix A.1 for a discussion. Most successful implementations use additional techniques to reduce the variance of gradient estimates, improve optimization geometry, and prevent convergence to deterministic policies. Actor-critic methods maintain an estimate of the value function under the current policy and leverage this to reduce the variance of gradient estimates. The A3C algorithm by Mnih et al. (2016) is an influential variant that exploits distributed computation. Building off of works by Kakade (2001) and Schulman et al. (2015), the Proximal Policy

Optimization (PPO) algorithm (Schulman et al. 2017) also uses regularization intended to stabilize policy changes and improve optimization geometry. Researchers sometimes achieve fantastic results with these algorithms, but as mentioned in the introduction, critical inspections suggest that subtle implementation details (Huang et al. 2022) and careful hyperparameter choices (Henderson et al. 2018) have an enormous impact on reported results.

As discussed below, we use a policy gradient type algorithm, but use deterministic, rather than randomized policies, and rely on the use of a differentiable simulator to calculate gradients.

HDPO and differentiable simulators. As discussed in the introduction, HDPO relies on two properties: (1) an ability to back-test how a policy would have performed in a historical scenario (See e.g. Sinclair et al. 2022) and (2) smoothness of the total cost incurred in a scenario with respect to small changes in actions. Rather than apply REINFORCE type algorithms, which rely on randomization to evaluate directions of potential policy improvement, these properties allow one to employ deterministic policies and perform model-based computation of gradients with respect to policy parameters. These ideas have old roots in the stochastic simulation literature. See (Glasserman 2004, Chapter 7) for a comparison of likelihood ratio estimates (similar to REINFORCE) and pathwise derivative estimates of gradients. Almost three decades ago, Glasserman and Tayur (1995) proposed a method to estimate gradients of total cost with respect to the parameters of base-stock policies. Today, it is possible to scale that approach to optimize over far more complex policy classes. Software libraries like TensorFlow (Abadi et al. 2015), PyTorch (Paszke et al. 2019), or JAX (Bradbury et al. 2018) now enable one to easily construct rich neural network policies, automatically calculate derivatives with respect to their parameters by backpropagation, and perform rapid parallel computation on graphics processing units (GPUs). Our use of HDPO is partly motivated by the recent impact differentiable physics simulators have had in optimizing robotic control policies (Freeman et al. 2021, Hu et al. 2019) in this manner. Our results suggest this approach holds great promise for tackling difficult inventory management problems.

The recent work of Madeka et al. (2022) also applies HDPO to inventory management. We learned of their work while our numerical experiments were well underway, and believe it is complementary. Madeka et al. (2022) verifies that HDPO can outperform internal baselines at Amazon, providing an exciting case study on the potential real-world impact of the approach. Our work includes three contributions not contained in theirs. First, while their work focuses solely on a single-location problem, our work emphasizes inventory problems in networks with multiple locations. This presents challenges due to the explosion of the state and action spaces, and motivates the symmetry-aware neural network architecture in Section 4. Second, rather than compare the performance of HDPO solely against heuristics, our results in Section 3 show HDPO recovers the actual optimum in problems whose solution structure was uncovered by decades of research.

Finally, rather than compare against heuristic baseline policies that are internal to Amazon, we open-source our code and ensure the performance of all baseline policies is reproducible. Notably, Section 5 includes a new benchmark problem constructed out of real time-series data. That section offers insight into why newsvendor-style methods are outperformed, which also builds trust in the findings.

Other deep reinforcement learning methods in inventory management. To our knowledge, Van Roy et al. (1997) were the first to apply DRL to inventory management. They introduce the notions of a "pre-decision-state" and "post-decision-state", which is the state of the inventory system before and after an order is placed. See Powell (2007) for further discussion. They apply a specialized approximate policy iteration algorithm which approximates the cost-to-go from a "post-decision-state" by applying a two layer neural network and hand-crafted features of the state. Recent successes of DRL in various domains have sparked renewed interest in applications to inventory management.

Recent papers of Oroojlooyjadid et al. (2022) and Shar and Jiang (2023) apply variants of the deep Q-networks algorithm (Mnih et al. 2015) to inventory management problems. Oroojlooyjadid et al. (2022) extends deep Q-networks (Mnih et al. 2015) to address the classical beer game. Their data-driven approach demonstrates the ability to recover near-optimal policies in a simplified setting and significantly outperforms simple heuristics when considering complicating problem features. Shar and Jiang (2023) introduce weakly coupled deep Q-networks, leveraging a Lagrangian-decomposition-based upper bound to accelerate the training of the Q-function approximation. Applied to an inventory control problem with multiple products and exogenous production rates, their approach demonstrates superiority over classical value-based methods. Our work deviates from this literature stream as HDPO focuses on directly approximating a policy rather than a value function.

Several other works apply policy gradient type algorithms to inventory control. With the exception of Madeka et al. (2022), which we discussed above, all works apply advanced variants of REINFORCE algorithms; these are fundamentally different from HDPO. A notable example is the work of Gijsbrechts et al. (2021), which applies the A3C algorithm to address three classical inventory control problems. They put great effort into getting A3C to work well — including tuning hyperparameters separately for each set of problem primitives — and report the resulting performance on well-documented benchmarks. In some problems, they include comparisons against the true optimal cost. For these reasons, their work serves as a crucial benchmark in Section 3.3.1. Unlike A3C, we find HDPO performs robustly well without requiring extensive parameter tuning or other tricks.

Some follow up works have applied the PPO algorithm, another REINFORCE style policy gradient algorithm that is a successor to A3C (Vanvuchelen et al. 2020, van Hezewijk et al. 2023, Vanvuchelen et al. 2023, Kaynov et al. 2024). Unfortunately, it is difficult to rigorously benchmark HDPO's performance by comparing it against our own implementation of PPO; research suggests that hundreds of implementation tricks impact the performance of PPO (Huang et al. 2022). (Note that the importance of these tricks already points to the benefits of HDPO.) We chose not to invest substantial effort in implementing and tinkering with PPO. Similar to A3C, PPO involves consideration of many tunable hyperparameters, contrasting with the simplicity of HDPO. Additionally, the results of (Vanvuchelen et al. 2020, van Hezewijk et al. 2023, Vanvuchelen et al. 2023, Kaynov et al. 2024) all demonstrate cases where it is sometimes surpassed by hand-coded heuristics. For instance, Kaynov et al. (2024) applied PPO to problems involving one warehouse and multiple locations, finding that PPO was often outperformed by an echelon-stock heuristic and emphasized the need for "better DRL algorithms tailored to inventory decision making". In a similar problem setup, HDPO achieved an average optimality gap smaller than 0.15% (refer to Table 4 in Section 3.3.2). Based on this evidence, we do not expect PPO to offer fundamentally different performance from the results in Gijsbrechts et al. (2021), against which we carefully compare.

Multi-agent reinforcement learning approaches in inventory management. In recent efforts to address inventory problems with a network structure, actor-critic multi-agent reinforcement learning (MARL) techniques have been explored (Liu et al. 2022, Ding et al. 2022). Although our work has a loose connection to MARL, we take a distinct approach by considering a "central planner" who optimizes a system comprising many "weakly coupled" components. This approach avoids the need for sophisticated techniques to handle the flow of information among agents.

Other machine learning approaches in inventory management. Alternative machine learning approaches, different from DRL, have also been investigated for inventory problems. Qi et al. (2023) uses supervised learning for a single-location inventory problem with stochastic lead times, employing posterior optimal actions as training labels for a policy NN. Additionally, Harsha et al. (2021) proposed training ReLU-based NNs to approximate value functions, and then utilizing them in a Mixed Integer Program to derive actions.

2. Problem formulation

We begin by presenting a generic formulation of Markov Decision Processes. We then highlight the additional features that make such a problem amenable to HDPO. Finally, we specialize to an inventory control problem, which is our main setting of interest.

2.1. Markov decision processes (MDPs)

In each period t, a decision-maker (who we later call the "central planner") observes a state S_t, and chooses an action a_t from a feasible set $\mathcal{A}(S_t)$. In a T-period problem, a policy π is a sequence of functions $\pi = (\pi_1, \ldots, \pi_T)$ in which each π_t maps a state S_t to a feasible action $\pi_t(S_t) \in \mathcal{A}(S_t)$. Let Π denote the set of feasible policies. The objective of the decision-maker is to solve

$$\min_{\pi \in \Pi} \mathbb{E}^{\pi} \left[\tfrac{1}{T} \textstyle\sum_{t \in [T]} c(S_t, \pi_t(S_t), \xi_t) \big| S_1 \right] \quad \text{subject to} \quad S_{t+1} = f(S_t, \pi_t(S_t), \xi_t) \;\; \forall t \in [T], \tag{1}$$

with $[x] = \{1, \ldots, x\}$, and where the system function f governs state transitions, the cost function c governs per-period costs, and $\xi = (\xi_1, \ldots, \xi_T)$ is a sequence of exogenous random terms. Note that the exogenous random terms may be drawn from an arbitrary stochastic process. This is how the classic textbook of Bertsekas (2011) defines MDPs.[2] We will later approximate an infinite-horizon average-cost objective by choosing the time horizon T to be very large. See Remark 1.

2.2. Hindsight differentiable reinforcement learning and policy optimization

In most RL problems, the decision-maker must learn to make effective decisions despite having very limited knowledge of the underlying MDP. Typically, they learn by, across episodes, applying policies and observing the T-length sequence of states visited and costs incurred. A *hindsight differentiable Reinforcement Learning (HD-RL)* problem has three special properties:

1. **Known system:** The system function f and cost function c are known.
2. **Historical scenarios:** The policy designer observes H historical examples of scenarios $(\bar{S}_1^h, \bar{\xi}_{1:T}^h)$, indexed by $h \in [H]$, with $\bar{S}_1^h$ representing an initial state and $\bar{\xi}_{1:T}^h = (\bar{\xi}_1^h, \ldots, \bar{\xi}_T^h)$ representing a trace of exogenously determined outcomes. We interpret the historical scenarios as independent draws from some population distribution.
3. **Differentiability:** The action space is "continuous" and the system and cost functions are "smooth enough" that the total cost $\sum_{t \in [T]} c(S_t, a_t, \xi_t)$ is an almost everywhere (a.e.) differentiable function of $(a_1, \ldots, a_T)$ for each fixed S_1 and ξ.

While these may seem like special properties, they arise naturally in a broad class of operations problems, including our main setting of interest described in the next subsection.

HD-RL problems allow for efficient policy search via a scheme we call *hindsight differentiable policy optimization* (HDPO), defined as follows. Let $\Theta = \mathbb{R}^b$, for some $b \in \mathbb{N}$, and let each $\theta \in \Theta$ define a smoothly parameterized deterministic policy π_θ. The first two properties allow to estimate

[2] An alternative definition, which may be more familiar to the reader, involves transition probabilities. This is just a stylistic difference and the two ways of defining MDPs are mathematically equivalent. A substantive constraint is that the current state variable must encode all information about past exogenous randomness that is relevant to predicting the next realization. See our discussion of forecasting information in the next subsection.

the policy loss $J(\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{t\in[T]} c(S_t, a_t, \xi_t)\right]$ of policy π_θ based on H historical (initial state, trace) pairs as

$$\bar{J}(\theta) = \tfrac{1}{H}\textstyle\sum_{h\in[H]} \ell(\theta, \bar{S}_1^h, \bar{\xi}_{1:T}^h), \tag{2}$$

where $\ell(\theta, \bar{S}_1^h, \bar{\xi}_{1:T}^h) = \left(T^{-1}\sum_{t\in[T]} c(S_t, \pi_\theta(S_t), \bar{\xi}_t^h)\right)$ is the cost under a scenario with trace $\bar{\xi}_{1:T}^h$ and initial state $S_1 = \bar{S}_1^h$. The third property ensures that $\ell(\theta, \bar{S}_1^h, \bar{\xi}_{1:T}^h)$ is an a.e. differentiable function of θ for a HD-RL problem. These three properties collectively enable the derivation of a low-variance gradient estimator for the policy loss as $\nabla_\theta \bar{J}(\theta) = \frac{1}{H}\sum_{h\in[H]} \nabla_\theta \ell(\theta, \bar{S}_1^h, \bar{\xi}_{1:T}^h)$. This is achieved by computing a sample of the gradient for each scenario $(\bar{S}_1^h, \bar{\xi}_{1:T}^h)$ separately and subsequently averaging them. Utilizing this estimator, one can efficiently perform stochastic gradient descent to search for the policy parameter that approximately minimizes Eq. (2).

The pseudo-code for HDPO is presented in Algorithm 1. In practical terms, one can abstract transition and cost functions f and g into a differentiable simulator, defining a differentiable mapping ℓ from a (parameter vector, initial state, trace) triplet to a cost (see Eq. 2). Modern automatic differentiation libraries, like PyTorch, allow one to obtain low-variance gradient estimates by backpropagating through the simulator (*i.e.*, by repeatedly applying the chain rule through the operations performed in the simulator in a backwards manner) to derive one gradient for each triplet, and subsequently averaging them. To simplify the presentation, Algorithm 1 demonstrates the forward and backward passes being performed one scenario at a time. However, in our implementation, we execute each pass simultaneously across a batch of scenarios. This is achieved by defining a tensor $\tilde{S}_t^H$ that describes the state across all scenarios within a batch and leveraging PyTorch's parallel computing capabilities to perform all operations in a parallel manner (see Appendix B.1 for a detailed explanation).

Remark 1 (On the use of stationary policies). For our experiments in Sections 3 and 4.3, we optimize over stationary policies, which have many practical benefits. It is known that there is a stationary optimal policy for a stationary infinite-horizon average-cost minimization problem under regularity conditions, like the restriction to finite state and action spaces (Bertsekas 2012). But an infinite horizon formulation of our problem is problematic in that it would require access to an infinite-length historical trace of exogenous outcomes $(\xi_1, \xi_2, \xi_3, \ldots)$. We will focus on large T to mimic an infinite horizon, hence we expect a stationary policy to perform very well.

Remark 2 (Comparison with REINFORCE style policy gradient optimization). It is noteworthy that policy gradient algorithms in the style of REINFORCE (Williams 1992) do not calculate gradients samples as in Algorithm 1, instead attempting to perform gradient based optimization without any knowledge of the model and without assuming any smoothness

Algorithm 1 Hindsight Differentiable Policy Optimization

Require: Historical scenario pairs $(\bar{S}_1^1, \bar{\xi}_{1:T}^1), \ldots, (\bar{S}_1^H, \bar{\xi}_{1:T}^H)$, initial policy parameters $\theta \in \Theta$, cost and transition functions c and f, gradient-based policy update function `UpdatePolicy` (*e.g.*, Adam).

while not reached termination criteria **do**

Randomly partition scenario indexes $[H]$ into n_B index batches $(\hat{H}_1, \ldots, \hat{H}_{n_B})$.

for index batch $\hat{H} \in (\hat{H}_1, \ldots, \hat{H}_{n_B})$ **do**

$J \leftarrow 0$, $\nabla J \leftarrow 0$ ▷ initialize batch cost and gradient

for $h \in \hat{H}$ **do**

$\ell \leftarrow 0$, $S \leftarrow \bar{S}_1^h$ ▷ initialize cost and initial state

for $t \in [T]$ **do** ▷ forward pass to obtain cost on scenario

$\ell \leftarrow \ell + c(S, \pi_\theta(S), \bar{\xi}_t^h)$ ▷ update cost on scenario

$S \leftarrow f(S, \pi_\theta(S), \bar{\xi}_t^h)$ ▷ get new state

end for

$J \leftarrow J + \ell$ ▷ update cost across batch

$\nabla J \leftarrow \nabla J + \nabla_\theta \ell$ ▷ get gradient sample (by backpropagation) and update

end for

$\theta \leftarrow$ `UpdatePolicy`$(\theta, \frac{1}{|\hat{H}|} \nabla J)$ ▷ update parameters based on average sample gradient

end for

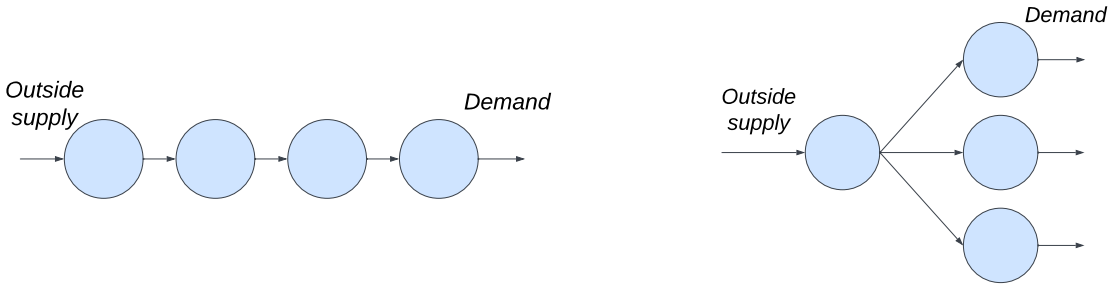
end while

properties of it. These algorithms enforce smoothness by employing randomized policies under which action probabilities are smooth functions of policy parameters. They derive gradient estimates by utilizing T-length sequences of states visited and costs incurred, and differentiating the log-probabilities associated with the sampled action in each observed state. The resulting gradient estimates are unbiased, but can have exceptionally high variance when T is large and policies are nearly deterministic. Further details on the operation of REINFORCE are provided in Appendix A.1.

2.3. Formulation of an HD-RL problem in inventory control

We use the HD-RL problem formulation described above to instantiate inventory control problems. We examine single-location problems and problems concerning two classical inventory network structures within the operations literature: a serial system and a one warehouse multiple store system, illustrated in Figures 1a and 1b, respectively. In the former, only the left-most location is permitted to acquire inventory, while all other locations can solely procure inventory from the preceding location. Additionally, only the right-most location can sell products. This problem

assumes that storing inventory in upstream locations is more cost-effective, making it advantageous to retain it in those locations. In the latter, a single location (referred to as the "warehouse") has access to an external supply source, allowing it to consolidate inventory and distribute it to stores that handle product sales.



(a) Serial. (b) One warehouse, multiple stores.

Figure 1 Diagrams of the network structures for two classical inventory management problems. Arrows indicate the direction in which inventory flows.

These problems may be challenging because the state and action spaces can be quite large due to the presence of many locations at which inventory is held. This is the "network" problem feature stressed in the introduction. For notational convenience, we present a formulation that represents the one warehouse multiple store system. This formulation captures all inventory control problems we consider except for one (the serial system), and we will use it consistently from Section 3 onwards.

The problem we model consists of $K+1$ locations, comprising K heterogeneous stores indexed by $k \in [K]$ and one warehouse indexed as location 0. Each store sells the same identical product (meaning that we do not model cross-product interactions), and we assume that the warehouse has access to an unlimited supply of goods. There are no constraints on storage or transfer quantities. For brevity, we will assume that lead times are strictly positive. It should be mentioned that the following formulation needs slight adjustments when dealing with lead times of zero.

Recall that $[x] = \{1, \ldots, x\}$, and define $[x]_0 = [x] \cup \{0\}$ and $(x)^+ = \max\{0, x\}$.

State and action spaces. In this model, the trace of exogenous outcomes $(\xi_1, \ldots, \xi_T)$, with $\xi_t = (\xi_t^k)_{k \in [K]}$, represents the sequence of (uncensored) demand quantities at each store and across time. The system state is a concatenation $S_t = (S_t^0, \ldots S_t^K)$ of $K+1$ local states

$$S_t^k = (I_t^k, Q_t^k, \mathcal{F}_t^k),$$

where $I_t^k \in \mathbb{R}$ is the current inventory on-hand at location k, Q_t^k represents the vector of outstanding orders and $\mathcal{F}_t^k$ represents information about the past that is relevant to forecasting future demand. More precisely, each location k has a known and fixed[3] lead time $L^k \in \mathbb{N}$. The vector $Q_t^k = (q_{t-L^k+1}^k, \ldots, q_{t-1}^k) \in \mathbb{R}_+^{L^k-1}$ encodes the quantity of goods ordered in each of the previous $L^k - 1$ periods. It is worth noting that Section 5 tests the performance of neural policies that are not supplied with explicit lead time information and instead make inferences based solely on past data.

The forecasting information $\mathcal{F}_t^k$ can encode information about the recent demand, and also exogenous contextual information like the weather. For the state to satisfy the Markov property, the forecasting information should be rich enough[4] to capture information about the past demands that is relevant to predicting the future:

$$\mathbb{P}\left([\xi_t, \ldots, \xi_T] \in \cdot \mid \mathcal{F}_t^1, \ldots \mathcal{F}_t^K\right) = \mathbb{P}\left([\xi_t, \ldots, \xi_T] \in \cdot \mid (\mathcal{F}_i^1, \ldots \mathcal{F}_i^K)_{i\in[t]}, \xi_1, \ldots, \xi_{t-1}\right).$$

After observing S_t, a central planner must jointly define the warehouse's order a_t^0 and the quantity of goods a_t^k allocated to every store from the warehouse, without exceeding the warehouse's inventory on-hand. The action space at state S_t is then given by $\mathcal{A}(S_t) = \{a_t \in \mathbb{R}_+^{K+1} \mid \sum_{k\in[K]} a_t^k \leq I_t^0\}$.

Notice that the dimension of the state space grows with the number of locations, the lead time, and the length of the demand forecast information. The dimension of the action space and the dimension of the demand realizations (*i.e.*, ξ_t) also grows with the number of locations.

Transition functions. Following the literature on inventory control, we separately consider two ways in which inventory on-hand at each store evolves:

$$I_{t+1}^k = I_t^k - \xi_t^k + q_{t-L^k+1}^k \qquad k \in [K], \quad \text{OR} \tag{3}$$

$$I_{t+1}^k = (I_t^k - \xi_t^k)^+ + q_{t-L^k+1}^k \qquad k \in [K] \quad . \tag{4}$$

The case (3) is known as *backlogged demand*, and imagines the customers will wait for the product if not immediately available (though a cost is incurred for the delay). The case (4) is known as a *lost demand* model and imagines that unmet demand disappears. The rest of the transitions are defined as

$$I_{t+1}^0 = I_t^0 + q_{t-L^0+1}^k - \textstyle\sum_{k\in[K]} a_t^k \tag{5}$$

$$Q_{t+1}^k = (q_{t-L^k+2}^k, \ldots, q_{t-1}^k, q_t^k = a_t^k) \qquad k \in [K]_0\,. \tag{6}$$

[3] We further note that random lead times can be modeled by letting each entry in Q_t^k track orders that have not yet arrived and updating I_t^k considering the stochastic realization for the random lead time.

[4] One can always satisfy this property by taking $\mathcal{F}_t^k = (\xi_1, \ldots, \xi_{t-1})$ to include all past demand observations. More concise representations are often possible. For instance, if demand is independent across locations, one can take $\mathcal{F}_t^k = (\xi_1^k, \ldots, \xi_{t-1}^k)$. If instead the demand process is an m^{th} order Markov chain, one can take $\mathcal{F}_t^k = (\xi_{t-m}^k, \ldots, \xi_{t-1}^k)$.

We omit an explicit formula for how forecasting information is updated, as this is case dependent. In some of our examples, demand is independent across time and no forecasting state is used. In another example, $\mathcal{F}_t^k$ contains a window of recent demands that is considered sufficient for predicting the future. Taken together, these equations define the system function f that governs state transitions.

Cost functions. The per-period cost function is defined as

$$c(I_t, a_t, \xi_t) = \underbrace{c^0(I_t^0, a_t)}_{\text{cost at warehouse}} + \sum_{k\in[K]} \underbrace{c^k(I_t^k, a_t^k, \xi_t^k)}_{\text{cost at store } k}. \tag{7}$$

The cost incurred at store k is given by $c^k(I_t^k, a_t^k, \xi_t^k) = p^k(\xi_t^k - I_t^k)^+ + h^k(I_t^k - \xi_t^k)^+$, where the first and second terms correspond to underage and overage costs, respectively. Per-unit underage costs p^k and holding costs h^k may be heterogeneous across stores. The cost incurred at the warehouse is given by $c^0(I_t^0, a_t) = \beta a_t^0 + h^0\left(I_t^0 - \sum_{k\in[K]} a_t^k\right)$. The first and second terms reflect procurement costs at rate β and holding costs at rate h^0, where we assume that $h^0 < \min_{k\in[K]} h^k$. We will restrict attention to zero warehouse procurement costs $\beta = 0$ for simplicity, following Gijsbrechts et al. (2021) and Xin (2021).

Historical scenarios. The policy designer observes H historical scenarios, consisting of independent examples of initial states $\bar{S}_1^h$ and demand traces $\bar{\xi}_{1:T}^h = (\bar{\xi}_1^h, \ldots, \bar{\xi}_T^h)$, indexed by $h \in [H]$. For our experiments with synthetic data in Section 3, we sample these from mathematical generative models (*e.g.*, from a Poisson distribution), as is common in the operations literature. In practice, retailers carry multiple items, and historical observations for each individual item provide a demand trace; these traces can collectively be used for training as in Madeka et al. (2022). We build a semi-realistic simulator from real time-series data in Section 5.

2.4. Feasibility enforcement for neural policy classes

We will apply HDPO to search over the weights θ defining a policy π_θ. For any given θ, π_θ maps a state vector S to a feasible action vector $\pi_\theta(S) \in \mathcal{A}(S) \subset \mathbb{R}^{K+1}$. To apply HDPO, we need a convenient way to ensure a NN outputs a feasible action, *e.g.*, to ensure that total inventory sent to the stores does not exceed inventory at the warehouse. Here, we show an approach to enforce action feasibility for settings involving a warehouse and multiple stores, and defer a definition for the serial system to Appendix B.2.

As motivation, consider the conventional way of applying NNs to classification problems. To enforce that a network output is a probability vector (over possible labels), a typical network outputs first an *intermediate output* which is an unconstrained vector, interpreted as "logits". Applying the softmax function transforms the logits to a feasible probability vector.

Similarly, we consider NN policies that first map a state to a warehouse allocation a^0 and an unconstrained *intermediate store output vector* $(b^1, \ldots, b^K) \in \mathbb{R}^K$, analogous to the logits in classification problems. Applying a pre-specified *feasibility enforcement function* assigns a feasible division of available warehouse inventory based on the intermediate allocation, analogous to the application of a softmax function in classification problems. For our settings involving one warehouse and multiple stores, a *feasibility enforcement function* is any function g that maps the warehouse inventory I_0 and intermediate outputs $(b^1, \ldots, b^K)$ to an allocation of inventory $(a^1, \ldots, a^k) \in \mathbb{R}_+^K$ obeying $\sum_{k=1}^K a^k \leq I^0$. We test three feasiblity enforcement functions defined as

$$g_1(I^0, b^1, \ldots, b^K) = \left[[b^k]^+ \cdot \min\left\{1, \frac{I^0}{\sum_{j\in[K]}[b^j]^+}\right\}\right]_{k\in[K]} \quad \text{(Proportional Allocation)} \tag{8}$$

$$g_2(I^0, b^1, \ldots, b^K) = \left[I^0 \cdot \frac{\exp(b^k)}{1+\sum_{j\in[K]}\exp(b^j)}\right]_{k\in[K]} \quad \text{(Softmax)} \tag{9}$$

$$g_3(I^0, b^1, \ldots, b^K) = \left[I^0 \cdot \frac{\exp(b^k)}{\sum_{j\in[K]}\exp(b^j)}\right]_{k\in[K]} \quad \text{(Softmax without Constant)}. \tag{10}$$

Softmax without Constant, g_3, additionally enforces all inventory at the warehouse to be allocated to stores and will be used only for a specific setting in which the warehouse cannot hold inventory (see Section 3.3.2).

3. HDPO recovers near-optimal policies in problems with hidden structure

This section considers inventory control problems whose hidden structure enables us to either compute or tightly bound the cost attained by an optimal policy. We apply HDPO to optimize over NN policies that map "raw" state vectors to (typically vector-valued) actions, without tailoring the methodology to each problem's special structure. We find that HDPO achieves (essentially) optimal average cost for each of these problems. As highlighted in the introduction, it is quite rare to benchmark deep RL against the global optimum; our ability to do so rests on decades of research in operations that has uncovered the hidden structure of the global optimum in some settings of interest.

In addition to comparing against the true optimum, comparisons with results reported by Gijsbrechts et al. (2021) highlight the benefits of HDPO over generic policy gradient methods that are popular in deep RL. HDPO exhibits much better performance on the problems tested in Gijsbrechts et al. (2021), and does so consistently without extensive tricks or hyperparameter tuning.

This section focuses on "textbook" inventory control problems and focuses solely on issues of optimization. Our solutions to these problems assume access to a very large dataset on historical demand, which, here, is synthetically generated and assumed to be i.i.d. We focus on problems with more limited or complex data in the following sections. Section 4 focuses on sample efficiency

improvements that are possible by reflecting inventory-network structure in the neural policy architecture. Section 5 stresses performance on real time series data.

Short note on terminology. In our experiments, a *setting* denotes a problem defined by a specific network structure and assumption regarding unmet demand (whether it is backlogged or lost). An *instance* refers to a setting with a particular choice of problem primitives, such as costs and lead time for each location. A scenario denotes an initial state and a sequence of demands over T periods across all stores. We will use the term *sample size* to denote the number of scenarios considered. A *run* signifies one execution of HDPO over a specific instance. An *epoch* refers to one pass of the algorithm across all scenarios, while a *gradient step* denotes an update to policy parameters after a pass through a batch of data. Moreover, to streamline notation, we will exclude superscripts associated with location for settings where only one location is considered.

3.1. Settings addressed

In this section, we study the performance of HDPO in four distinct settings. We begin with two settings involving a single store: one where unmet demand is assumed to be backlogged, and the other where it is considered lost. Next, we consider a setting with a serial network structure (see Figure 1a), operating under a backlogged demand assumption. Finally, we analyze a setting involving a warehouse and multiple stores (see Figure 1b), also under the assumption of backlogged demand. In this setting, as the warehouse cannot hold inventory, we refer to it as a *transshipment center*. For detailed definitions of each setting, please refer to Appendix A.2, Section 3.3.1, Appendix A.4, and Section 3.3.2, respectively.

3.2. Overview of implementation details

Precise experiment details for experiments using synthetic data are presented in Appendix B. We developed a differentiable simulator using PyTorch and executed all experiments on an NVIDIA A40 GPU with 48GB of memory. All trainable benchmark policies were also optimized using our differentiable simulator; we restart their training multiple times and report the best results attained. We utilize the Adam optimizer, setting the β parameters to the default values provided by PyTorch, namely $(0.9, 0.999)$. To mimic an infinite horizon stationary setting, we train the parameters of a stationary policy to minimize cost accrued over episodes of long duration (specified below), and report the costs obtained per store-period. We drew initial states randomly, and this appeared to enhance convergence speed compared to utilizing a fixed initial state. To avoid reporting "transient" effects, the metrics we report exclude a given number of early periods, which we specify below.

We utilized three separate datasets: a *train* set for model training, a *dev* set (short for *development*) for hyperparameter tuning and model selection, and a *test* set for evaluating the models' performance over a longer time horizon and estimating per-period costs. For each run, we use early

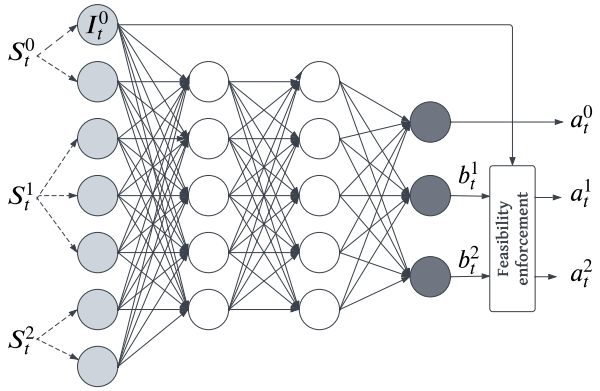


Figure 2 Vanilla NN for a setting with a warehouse and 2 stores.

stopping. That is, we save the model weights and report performance of the model that minimizes costs on the dev set. All metrics reported throughout this section are based on the test set.

Throughout our trials, we consistently utilized a fixed duration of 50 periods, excluding the initial 30 periods when calculating costs for both the train and dev sets. However, in the setting involving a transshipment center (see Section 3.3.2), we extended the duration to 100 periods while excluding the initial 60 periods for both the train and dev sets. This increase in horizon length was necessary in that particular setting, as we observed a significant increase in average costs when simulating over a longer horizon when training with 50 periods. For the test set, we expanded the number of periods to 500 and excluded the initial 300 periods. We utilized a sample size of 32,768 scenarios for each of the training, development, and test sets, unless specified otherwise. As a termination condition, we established a maximum number of gradient steps for each setting. These limits were determined based on prior knowledge of the approximate count required to achieve nearly optimal solutions, which we obtained from early experiments. For the experiments in this section, computational constraints did not pose a limitation. HDPO consistently achieved performance within 1% of optimality within a 10-minute timeframe for instances involving a single location. For instances with up to 10 stores, it typically achieved this within less than 80 minutes, except for a few specific runs.

Neural Network Architectures. We employ a basic NN policy architecture class, which we refer to as the *Vanilla NN*. It consists of a fully-connected Multilayer Perceptron (MLP) as depicted in Figure 2. The raw state is fed to the NN, and its output is used to determine an order quantity for each location.

We employed ELU activation functions for all non-output layers, facilitating faster and more stable learning compared to ReLU and Tanh. In settings involving a single store, we apply a Softplus activation function to each output to ensure non-negative order amounts.[5]

In the one transshipment center many stores setting, the Vanilla NN outputs an allocation for the transshipment center and intermediate store outputs for each store. Subsequently, a *Softmax without Constant* feasibility enforcement function (See Sec. 2.4) is applied across these intermediate store outputs to derive store allocations. Note that both the input and output size scale linearly with the number of locations. In Appendix B.2, we offer an in-depth explanation of the architecture class considered for the serial setting, given the considerable difference in network structure from other settings.

For warehouse orders, we consistently employed one "trick" which allowed to solve large-scale problems in a more stable manner. We fix a very crude upper bound on the amount of inventory the warehouse should order[6] and call this the maximum-allowable order. Then, we obtain warehouse orders by applying a Sigmoid activation function to the NN output and multiplying by the max-allowable order.

3.3. Results

Table 1 summarizes the performance of the Vanilla NN for four settings where we can compare the performance to the optimal cost or a provable lower bound, across multiple instances. For each setting, we maintain consistent hyperparameter values across instances. In settings involving a single location, the cost of a single run is reported for each instance. On the other hand, for settings with multiple locations, we present the best cost achieved out of three runs for each instance. This approach is adopted due to occasional runs where HDPO, while generally reliable, exhibited suboptimal performance, and selecting the best run provides a more robust evaluation. Our findings illustrate that the Vanilla NN consistently and reliably attains near-optimal policies in instances with a state space of up to 63 dimensions, even when utilizing raw state inputs. Moreover, our results demonstrate that when constraints can be imposed in a differentiable manner, HDPO successfully addresses inventory problems with a network structure. This includes settings with a serial network structure (with 4 locations) and one with a warehouse and multiple stores (ranging from 3 to 10 stores).

[5] To avoid beginning on the "flat" regions of the Softplus function, we add 1 to the output of the linear layer preceding the application of the Softplus.

[6] In practice, we set this value to 4 times the average cumulative demand across stores.

Table 1 Vanilla NN performance in settings where the optimal cost can be bounded or computed. We report performance under the best-performing hyperparameter configuration.

Network structure	Unmet demand assumption	Dimension of raw state	Benchmark	Instances tested (#)	Average opt. gap	Max. opt. gap
One store, no warehouse	Backlogged	1 to 20	Analytical optimal cost	24	0.03%	0.05%
One store, no warehouse	Lost	1 to 4	Computable optimal cost for short lead times	16	$< 0.25\%$	$< 0.25\%$
Serial	Backlogged	10 to 13	Optimization over optimal policy class	16	0.35%	0.69%
Many stores, one transship-ment center	Backlogged	9 to 63	Analytical lower bound	24	$\leq$0.15%	$\leq 0.47\%$

3.3.1. On the benefits of HDPO As explained in Section 2.2, HDPO leverages important problem properties (known system, observation of historical scenarios, and differentiability) to enable an efficient search over the parameters of a neural policy. Is this important, or would generic deep RL methods have matched the exceptional performance summarized in Table 1?

Some insight can be gained by comparing to the performance reportd in Gijsbrechts et al. (2021), who tested their A3C approach on 6 out of 16 instances from the classic inventory control paper Zipkin (2008), which considers a single store facing a stationary Poisson demand, under lost demand and discrete allocation (second row in Table 1). We replicated the test-bed of 16 instances from Zipkin (2008) by setting holding cost $h = 1$, underage cost $p = 4, 9, 19, 39$, and lead time $L = 1, 2, 3, 4$. **We trained our model as if actions were continuous, and rounded prescribed allocations to the nearest integer at test time.** To test the robustness to hyperparameter choices, we considered architectures with two and three hidden layers, learning rates of $10^{-4}, 10^{-3}$, and 10^{-2}, and batch sizes of 1024 and 8192, *i.e.*, $2 \times 3 \times 2 = 12$ different hyperparameter combinations, running each of them once on all instances. We quantify performance in terms of the optimality gap with respect costs obtained by solving the actual dynamic program, as reported by Zipkin (2008).

For each instance, we compare the **worst** loss achieved by HDPO across hyperparameter settings with the **best** loss reported by Gijsbrechts et al. (2021). The latter corresponds to the best loss across approximately 250 hyperparameter combinations for each instance. Additionally, we include results from the Capped Base-Stock (CBS) heuristic proposed by Xin (2021), which achieved an average optimality gap below 0.7% in the 16-instance test bed. The optimal parameters reported by Xin (2021) were considered, but we ran the heuristic in our environment.We note that values of the lower bound on the optimal cost are reported in Zipkin (2008) with two decimals, while we

did not round the loss obtained by HDPO or CBS. The smallest optimal average cost is around 4.04, so we consider that gaps below $0.01/4.04 * 100 \approx 0.25\%$ cannot be "detected".

Table 2 summarizes the performance of each approach on the 6-instance test bed. While the approach presented by Gijsbrechts et al. (2021) falls short of surpassing the CBS heuristic in any of these instances, with gaps between 3.0% and 6.7% for the best run, HDPO consistently outperforms the CBS heuristic, demonstrating near-optimal performance across the board. It achieved a minimum detectable optimality gap of 0.25% in each of the 192 runs. In contrast, Gijsbrechts et al. (2021) reported that their A3C algorithm exhibited high sensitivity to hyperparameters, necessitating extensive tuning.

Table 2 Optimality gaps in 6 instances studied in Zipkin (2008) for A3C (costs reported by Gijsbrechts et al. 2021, reflecting best run across around 250 hyperparameters), CBS heuristic (parameters from Xin 2021,), and HDPO (worst run across 12 hyperparameter settings per instance).

Approach	$L=2$ $p=4$	$L=2$ $p=9$	$L=3$ $p=4$	$L=3$ $p=9$	$L=4$ $p=4$	$L=4$ $p=9$
A3C (best run)	3.20%	4.80%	3.00%	3.10%	6.70%	3.40%
CBS	$< 0.25\%$	0.43%	0.67%	1.34%	1.63%	1.04%
HDPO (worst run)	$< 0.25\%$	$< 0.25\%$	$< 0.25\%$	$< 0.25\%$	$< 0.25\%$	$< 0.25\%$

Table 3 showcases the computational efficiency of HDPO and its robustness to hyperparmeter choices. For a range of hyperparmeter choices, it presents the average number of gradient steps and the time required to achieve a 1% optimality gap on the dev set across the 16 instances considered in Zipkin (2008).

Further results regarding the performance of HDPO in this context can be found in Appendix B.5.

Remark 3 (Why we do not implement all competing DRL algorithms ourselves). Unfortunately, benchmarking generic deep RL methods methods is subtle since their performance can be very sensitive to implementation choices, and even the choice of random seed (Huang et al. 2022, Henderson et al. 2018). Because of this challenge, we do not implement and tune generic deep RL methods ourselves, and instead benchmark against the results reported in previous research works which invested subsantial effort into getting them to work well. In particular, we compare against results of Gijsbrechts et al. (2021), who employ the A3C algorithm of Mnih et al. (2016). With enough compute and engineering effort, we expect that a variety of DRL could be used to solve the relatively simple problems instances in Zipkin (2008). In these easier problems, it is the simplicity and reliability of HDPO that stands out.

Table 3 Summary of performance metrics of the Vanilla NN for different hyperparameter settings for the single store with lost demand problem setting, across the 16-instance test-bed in Zipkin (2008) for Poisson demand with mean 5. We consider that an instance is solved to optimality if the gap is smaller than 0.25%. For the last 2 columns we consider the performance of the NN with continuous allocation as proxy for the performance in the discrete-allocation setting, as costs changed, on average, by less than 1% after discretizing the allocation.

Hidden layers	Batch size	Learning rate	Instances solved to optimality (#)	Average gradient steps to 1% dev gap	Average time to 1% dev gap (s)
2	1024	0.0001	16	4660	346
2	1024	0.0010	16	1200	94
2	1024	0.0100	16	670	48
2	8192	0.0001	16	4824	418
2	8192	0.0010	16	998	83
2	8192	0.0100	16	680	65
3	1024	0.0001	16	4660	394
3	1024	0.0010	16	1200	93
3	1024	0.0100	16	670	56
3	8192	0.0001	16	4824	498
3	8192	0.0010	16	998	96
3	8192	0.0100	16	680	63

3.3.2. HDPO can near-optimally solve some problems with network structure. We go into further detail into the results on one setting: the many stores, one transshipment center setting depicted in row 4 of Table 1.

We consider the setting introduced in Federgruen and Zipkin (1984a), where a warehouse operates as a transshipment center (*i.e.*, cannot hold inventory) and there are multiple stores, under a backlogged demand assumption (fourth row in Table 1). Demand is i.i.d. across time but may exhibit correlation across stores. The warehouse has a constant lead time of 3 periods. To apply a known analytical lower bound (see Appendix A.5) on the optimum, we assume uniform per-unit costs and lead time across all stores. We generated 24 instances by fixing holding costs at 1, considering number of stores $K = 3, 5, 10$, lead times $L^k = 2, 6$, underage costs $p^k = 4, 9$, and pairwise correlation in demands of 0.0 and 0.5. The store-level marginal demand distributions are assumed to be normal, with the mean and coefficient of variation (recall that the coefficient of variation is the ratio of the standard deviation to the mean) sampled uniformly between 2.5 to 7.5 and 0.16 to 0.32, respectively. In this setting, demand may take on negative values (corresponding to the possibility of products being returned directly to a store). Three runs were performed for each instance, with a limit of 800 epochs.

In Table 4 we show an upper bound on the optimality gap for the best run on each instance for instances with pairwise correlation in demands of 0.0 (left portion) and 0.5 (right portion) (additional findings are detailed in Table 10 in Appendix A.5). We obtain near-optimal performance across all instances, with a maximum gap of 0.47% and an average gap of 0.15%. These

Table 4 Upper bound on optimality gap for the Vanilla NN in the many stores one transshipment setting, with pairwise correlations in demands of 0.0 (left portion) and 0.5 (right portion).

Number of stores	Store leadtime	Underage cost	Upper bound on test gap
3	2	4	0.07%
3	2	9	0.08%
3	6	4	0.11%
3	6	9	0.14%
5	2	4	0.06%
5	2	9	0.08%
5	6	4	0.16%
5	6	9	0.17%
10	2	4	0.15%
10	2	9	0.19%
10	6	4	0.24%
10	6	9	0.28%

Number of stores	Store leadtime	Underage cost	Upper bound on test gap
3	2	4	0.15%
3	2	9	0.23%
3	6	4	0.14%
3	6	9	0.16%
5	2	4	0.06%
5	2	9	0.06%
5	6	4	0.09%
5	6	9	0.06%
10	2	4	0.13%
10	2	9	0.15%
10	6	4	0.20%
10	6	9	0.47%

results demonstrate the reliability of HDPO in effectively addressing the network problem studied, particularly when the existing constraints can be represented in a differentiable manner.

4. Symmetry-aware policy network architectures enhance sample-efficiency

The previous section studied the ability of HDPO to recover near-optimal performance when applied to a large dataset of historical demand observations. Here we focus on the sample efficiency of HDPO, which we define as the number of scenarios needed in the training data to achieve relatively strong out-of-sample performance. This is likely to be an important consideration in practice. As explained in Section 5, when applying HDPO to real data each training scenario corresponds to a time series of demand observations for a single product. Sample inefficient methods require historical observations of many products, rendering them unusable for inventory management for retailers that carry narrower product ranges.

Our main hypothesis is that reflecting the structure of the inventory network in the NN architecture can greatly enhance sample efficiency. We study this hypothesis in the important special case of the setting depicted in Figure 1b, where the inventory network consists of a single warehouse that consolidates inventory from external supply and distributes it across multiple stores. In Section 4.1, we introduce a "symmetry-aware" policy network architecture, which is designed to reflect the similar and weakly-coupled nature of the inventory decisions made at individual stores. Section 4.2 offers theoretical insight into why this architecture could be expected to work well. Numerical experiments in Section 4.3 show that employing symmetry-aware policies can result in enormous sample efficiency benefits. We find that performance of the Vanilla neural network architecture studied in the previous section degrades substantially both as the number of stores increases and

as the number of training scenarios decreases; performance under the symmetry-aware architecture remains strong in settings with just four scenarios in the training data and up-to fifty stores (see Figure 4).

4.1. Symmetry-aware architecture design

As explained in Section 3.2, a Vanilla NN comprises a single fully-connected MLP that directly maps the raw state to intermediate outputs for each location. This class of functions is quite flexible, and could easily express strange ordering policies, like one where inventory orders at store 7 are large whenever inventory is abundant at store 11. As a result, trained Vanilla NNs risk overfitting decisions to spurious patterns in the training data. Our experiments in this section suggest its out-of-sample performance degrades when there are a large number of stores.

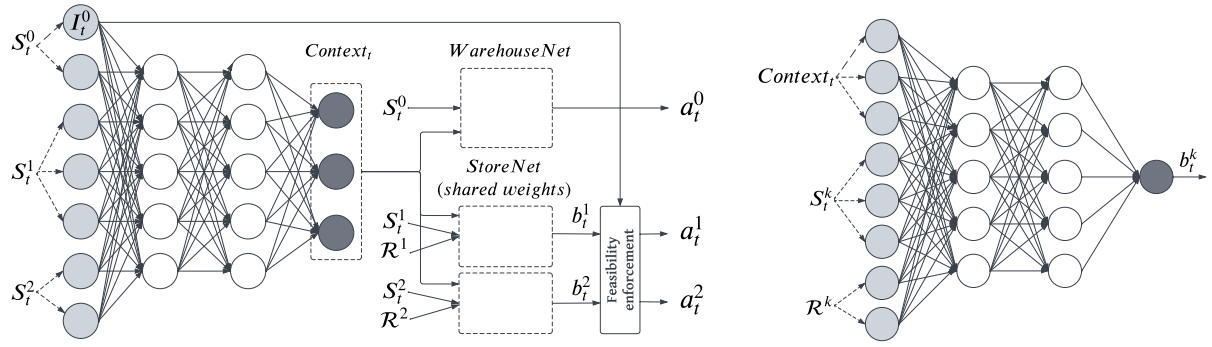
Hoping to improve sample efficiency, we introduce the *Symmetry-aware NN* architecture class, which mimics the physical structure of the inventory network, and leverages weak coupling between locations and the symmetry among stores. The symmetry-aware NN architecture is depicted in Figure 3a. Our implementation maintains three separate neural networks: a context Net, a Warehouse Net and a Store Net (with virtual copies of the latter, with identical weights, for each store). The context network produces a d dimensional embedding of the overall state of the inventory system – termed the current *context*. The Store Nets, depicted in Figure 3b, specify intermediate outputs for a specific store based on the context, the store's local state and store-specific primitives $\mathcal{R}^k$, which might include cost parameters, lead time information, or summary statistics of the demand distribution. The Warehouse net directly specifies a warehouse order as a function of the context and its local state.

The composition of these NNs can be viewed as a single NN, as depicted in Figure 3a. This NN can be viewed as a specific symmetry aware policy, denoted by π in Definition 1. This architecture exploits symmetries across stores by utilizing weight sharing, while allowing for heterogeneous behavior solely via the learned mapping's treatment of local states S_t^k and store-specific primitives $\mathcal{R}^k$. The weights of this NN are trained in an end-to-end fashion by HDPO.

DEFINITION 1. Let $\pi_{\text{context}} : \mathcal{S} \to \mathbb{R}^d$ be a mapping from state to a d-dimensional "context" vector for some $d \in \mathbb{N}$, and $\pi^0 : \mathbb{R}^{d_0} \times \mathbb{R}^d \to \mathbb{R}$ and $\pi^1 : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \times \mathbb{R}^d \to \mathbb{R}$ be mappings for the warehouse and the stores, respectively, with $d_0 \in \mathbb{N}$ the dimension of the local state for the warehouse, and $d_1, d_2 \in \mathbb{N}$ the common dimension of local state and primitives across stores. We say π is a *symmetry-aware policy* with *d-dimensional context* if it is of the form

$$
\begin{aligned}
a_t^{\pi 0} &= \pi^0\left(S_t^0, \pi_{\text{context}}(S_t)\right) \\
b_t^{\pi k} &= \pi^1\left(S_t^k, \mathcal{R}^k, \pi_{\text{context}}(S_t)\right) && k \in [K] \\
a_t^{\pi k} &= [g(I_t^0, b_t^{\pi 1}, \ldots, b_t^{\pi K})]_k && k \in [K],
\end{aligned}
$$

where g is a feasibility enforcement function, defined above. We call π_{context} the *context mapping* and π^0, π^1 the *context-dependent local policies* for the warehouse and the stores, respectively.



(a) Symmetry-aware NN architecture. (b) Store Net.

Figure 3 Symmetry-aware NN (left) and Store Net (right) with a 3-dimensional context for a setting with a warehouse and 2 stores.

4.2. Why should a symmetry-aware policy work well?

Here we offer some theoretical insights into why symmetry-aware policies should work well. We construct a stylized example in which the number of stores K grows. Since the the dimension of the state and action spaces increases as the number of stores grows, one might expect that the structure of an optimal policy grows increasingly complex. Instead, due to increasingly weak coupling between individual stores, the problem simplifies. There is a simple (asymptotically) optimal policy in which each store makes decisions in a symmetric manner based on its local state and summary statistics that give an overall sense of inventory scarcity in the network. Our formal theorem shows this policy can be expressed as a particularly simple instance of a symmetry aware NN.

The formal example we study in our theory is defined below.

EXAMPLE 1. Consider K stores and one warehouse, with backlogged demand (3). The underage and holding costs of each store k satisfy satisfy $p^k \in [\underline{p}, \overline{p}]$ and $h^k \in [\underline{h}, \overline{h}]$ for some $\underline{p}, \overline{p}, \underline{h}, \overline{h} \in \mathbb{R}_+$. Demand is of the form $\xi_t^k = B_t U_t^k$, where $U_t^k \sim \text{Uniform}(\underline{u}^k, \overline{u}^k)$ is drawn independently across stores and time (with $\overline{u}^k \in [1/\kappa, \kappa] \forall k \in [K]$ for some $\kappa \in \mathbb{R}$), and B_t is drawn independently across time; it takes a *high* value $\gamma^{\text{H}} > 0$ with probability q and a *low* value $\gamma^{\text{L}} \in (0, \gamma^{\text{H}})$ otherwise. The warehouse lead time is 1 and store lead time is 0. The system starts with zero inventory. We make two extra assumptions. First, $\gamma^{\text{L}} \underline{u}^k \geq \gamma^{\text{H}} \overline{u}^k - \gamma^{\text{L}} \underline{u}^k$ for every k, which ensures that, if store k starts with an inventory no larger than $\gamma^{\text{H}} \overline{u}^k$, the remaining inventory after demand is realized is below $\gamma^{\text{L}} \underline{u}^k$. (Note that this assumption implies that $\underline{u}^k > 0$ for every k, ensuring $U_t^k > 0$ w.p. 1 for every

k and t.) Second, $q\underline{p} \geq (1-q)h^0$, which ensures that it is worth acquiring an incremental unit of inventory at the warehouse if that is sure to prevent a lost sale in a period of high demand. □

We comment briefly on the structure of this example setting. Here, due to the aggregate demand being either "high" or "low" in each period, and the lead time of the warehouse being a single period, in each period there is either a scarcity or an abundance of inventory in the network, due to the aggregate demand having been "high" or "low", respectively, in the previous period. As a result, both the warehouse and the stores do need some information about the global inventory state to place an appropriate order. We believe it is possible to prove that making local orders without knowledge of the global state necessarily leads to an $\Omega(1)$ percentage increase in the cost incurred, under a range of problem primitives. By contrast, we expect that no information about the global state of the system is needed when demand realizations are independent across stores.

We establish the following guarantee of asymptotic optimality in per-period costs of a symmetry-aware policy with 1-dimensional context, as the number of stores K grows. At least in this asymptotic limit, it is optimal for each store to make decisions in a symmetric manner based on its local state and a summary of the overall system state.

THEOREM 1. *In Example 1, let J_1^π be the expected total cost incurred by policy π from the initial state. There exists $C = C(\overline{p}, \overline{h}, \underline{p}, \underline{h}, q, \kappa, \gamma^{\mathrm{L}}, \gamma^{\mathrm{H}}) < \infty$ such that the following occurs. There exists a stationary symmetry-aware policy $\tilde{\pi}$ with 1-dimensional context which satisfies $\frac{J_1^{\tilde{\pi}}}{\inf_\pi J_1^\pi} \leq 1 + \frac{C}{\sqrt{K}}$.*

The proof is given in Appendix C. It constructs an asymptotically optimal policy $\tilde{\pi}$ in which the warehouse follows an echelon-stock policy (see Equation 19 in Appendix A.4) and each store a base-stock policy (see Equation 15 in Appendix A.2), with the current base-stock level depending only on an estimate of whether the previous system-wide demand was high or low; this is signaled by a context mapping that maps I_t to the sum of its components $\sum_{k\in[K]_0} I_t^k$.

4.3. On the enhanced sample efficiency of Symmetry-aware NNs

To study the sample efficiency of the Symmetry-aware NN, we explored a setting with a single warehouse capable of holding inventory and multiple stores, under the lost demand assumption. These experiments highlight the impact of neural architecture design on an agent's out-of-sample performance under a limited number of training scenarios. Our investigation is motivated by real-world settings in which a retailer operates with a small product range, and hence data is limited since historical data for each product provides a single scenario. Our experiments demonstrate that the Symmetry-aware NN significantly outperforms the Vanilla NN in such a regime.

Neural Network Architectures. For the Vanilla NN, we follow the specifications outlined in Section 3, and employ a Softmax feasibility enforcement function as per Equation (9). For the Symmetry-aware NN, we adopt the previously mentioned "trick" for warehouse orders and utilize

ELU activation functions. We consistently employ a Sigmoid activation function for the outputs of the Context Net, a choice that we observed enhances the stability of the learning process. Additionally, we utilize a Proportional Allocation feasibility enforcement function (Equation (8)).

We note that we assessed different feasibility enforcement functions for the Vanilla and Symmetry-aware NNs and selected the one that demonstrated superior performance for each architecture separately. The superior performance of the Proportional Allocation feasibility enforcement function in the Symmetry-aware NN aligns with the intuition that store actions are "weakly" coupled. In this design, an intermediate store output may influence another store's action only when inventory at the warehouse is fully allocated.

Experiment specifications. For the experiments detailed in this section, we adhere to the terminology and implementation details provided in Section 3. Any additional specifications are highlighted here.

It is important to note that, in this setting, we no longer benefit from an analytical bound on costs. We generate demands by sampling from a multivariate normal distribution that is i.i.d. over time, truncating it at 0 from below. For each store, we sample parameters uniformly at random and independently, including an underage cost between 6.3 and 11.7, holding cost between 0.7 and 1.3, lead time between 2 and 3 periods, mean demand between 2.5 and 7.5, and coefficient of variation of demand between 0.25 and 0.5. The warehouse operates with a lead time of 6 periods and incurs a holding cost of 0.3 per unit. Additionally, the normal distribution from which we sample assumes a fixed correlation of 0.5 across all pairs of stores. All parameters remain time-invariant, and the scenarios are consistent across all runs for a fixed instance.

For the Vanilla NN, we employ three hidden layers and investigate architectures with 128 and 512 neurons per layer. As for the Symmetry-aware NN, we maintain a consistent architecture that incorporates Warehouse and Store Nets with 16 and 32 neurons per layer, respectively, and includes two hidden layers for each. Meanwhile, for the Context Net, we consider one hidden layer and one output layer, each consisting of 256 neurons. While it may be possible to achieve comparable performance with fewer units per layer in the Context Net, theoretical evidence suggests that overparameterized networks enhance generalization in deep learning (Jiang et al. 2019) and enjoy convergence guarantees under mild assumptions when trained with first-order methods (Chen et al. 2019, Allen-Zhu et al. 2019). Therefore, we have opted to utilize wide layers. For each architecture, we examine the learning rates in the set $10^{-5}, 3\times10^{-5}, 10^{-4}, 3\times10^{-4}, 10^{-3}, 3\times10^{-3}, 10^{-2}, 3\times10^{-2}$.

The batch size was determined as the minimum between 1024 and the total number of training scenarios. In our experiments with smaller batch sizes, we observed a decline in performance for the Vanilla NN. Furthermore, in initial experiments, we explored the use of weight decay as a possible

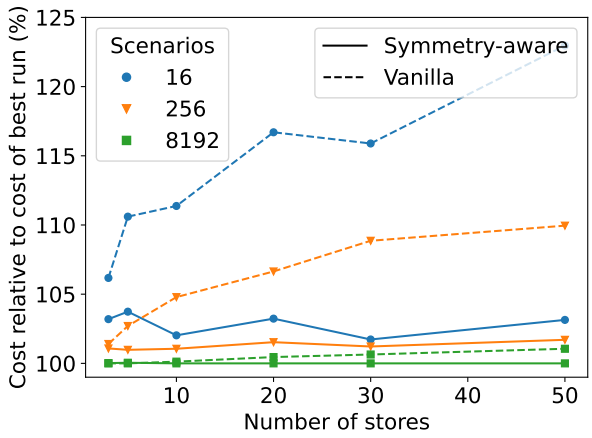
method to address overfitting. However, no additional gains were observed, leading us to decide against its use.

We explore different instances by varying the number of stores within the set $\{3, 5, 10, 20, 30, 50\}$ and evaluate performance with train sample sizes from the set $\{16, 256, 8192\}$ (a scenario consists of an initial state and demands for each store across 50 periods). We conduct 3 runs for each architecture class, layer width, and learning rate combination. For each training run, we use early stopping and save the model weights corresponding to the best performance on a dev set comprising 4096 scenarios. Subsequently, we assess the model's performance on a separate clean test set consisting of 4096 new scenarios. Training is capped at a maximum of 16,000 gradient steps, and we stop training if the performance on the dev set does not improve after 500 epochs.

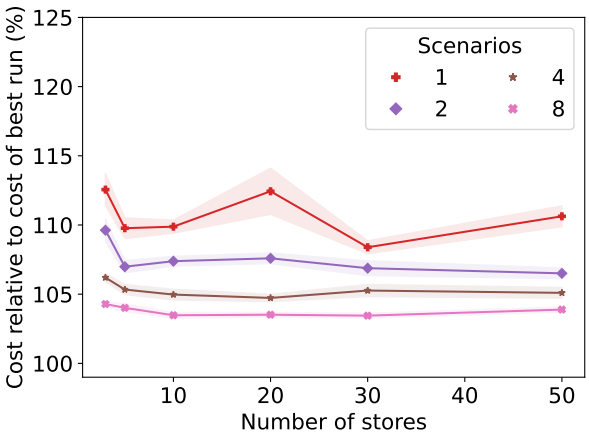
Main findings. Figure 4a depicts the cost attained by Vanilla and Symmetry-aware NNs in their best-performing runs, for a given number of training scenarios and stores. We report the cost achieved relative to the best run for the same number of stores across all architectures and sample sizes. The graph reveals that the performance of the Vanilla NN declines rapidly with an increasing number of stores, even with as many as 8192 scenarios. Moreover, its performance significantly deteriorates with a decrease in the number of scenarios, leading to cost increases exceeding 22% for the case of 16 scenarios and 50 stores. In contrast, the performance of the Symmetry-aware NN does not exhibit a noticeable decline as the number of stores increases and consistently achieves results within 5% relative to the best run. The higher sample efficiency of the Symmetry-aware NN compared to the Vanilla NN may be because the Symmetry-aware NN architecture enforces the underlying symmetry of the problem, eliminating the need for the agent to "learn" that property from the data, and leading to parameter efficiency due to sharing of weights across Store Nets. Furthermore, the Symmetry-aware NN may effectively be able to learn from each of the demand traces seen for individual stores in a single scenario for the overall system. In contrast, the Vanilla NN experiences an increase only in the dimension of the scenario as the number of stores grows.

We delve deeper into this experiment by assessing the performance of the Symmetry-aware NN with smaller training sample sizes of $1, 2, 4$ and 8 scenarios. Recognizing that with such diminutive sample sizes, the quality of the policy learned by our agent may vary with the specific scenarios used for training, we repeat the outlined procedure in the preceding paragraph 12 times for each sample size to estimate the average performance. This involves changing the seeds used to generate scenarios across each iteration. For each of the 12 seeds, we examine the lowest test loss attained across all runs for each number of stores , and again report the cost achieved relative to the best run for the same number of stores across all architectures and sample sizes as described in the preceding paragraph. The average and 95% confidence interval for this quantity are depicted in Figure 4b. Remarkably, the Symmetry-aware NN consistently displays relative performance gaps

of less than 5% for all numbers of stores when the number of scenarios is 8 or greater. Even with just two training scenarios, it exhibits a performance gap below 10% for every number of stores, underscoring its exceptional sample efficiency.



(a) Performance for the Vanilla and Symmetry-aware NNs for 16, 256 and 8192 training scenarios.



(b) Mean and 95% confidence interval for the performance of the Symmetry-aware NN for 1, 2, 4 and 8 training scenarios.

Figure 4 Cost comparison relative to the optimal run in a setting with one warehouse, multiple stores, and a lost demand assumption, for a varying number of stores and training scenarios. We calculate the minimum cost achieved across all runs for a specific architecture class, number of training scenarios, and number of stores. This minimum cost is then divided by the minimum cost achieved across all runs for a given number of stores. For sample sizes of 8 or smaller, we repeat the process 12 times, and report the mean and 95% confidence interval for this quantity.

5. HDPO with real time series data

To exemplify the use of HDPO with real-world data, we utilized sales data from the *Corporación Favorita Grocery Sales Forecasting* competition (Favorita 2017) hosted on Kaggle. Corporación Favorita, one of Ecuador's largest grocery retailers, operates stores in a variety of formats, including hypermarkets, supermarkets, and convenience stores, all with physical locations. The dataset encompasses around 200,000 daily sales time series for over 4,000 products across 54 stores, spanning the years 2013 to 2018. The primary challenge in these test problems lies in addressing the presence of potentially complex nonstationary patterns in the sales data. Building and sharing an open-source set of test problems derived from this dataset is a secondary contribution of the paper.

HDPO addresses nonstationarity by optimizing the hindsight performance of policies that map relevant information — including a historical window of demand observations, information on outstanding inventory orders and other features — to a decision. This approach is compared with

generalized newsvendor policies, which address nonstationarity in the data by initially forecasting the distribution of upcoming cumulative demand over the item's lead time and then making decisions based on that distribution's quantiles. Our findings indicate that HDPO consistently outperforms these generalized newsvendor methods in a lost demand model. Subsection 5.7 provides insights into the drivers of the performance gap.

5.1. Dataset description

We utilize the dataset by aggregating sales data on a weekly basis. Each sales time series, corresponding to a specific product at a particular store, is treated as an independent demand trace. We concentrate on a period of 170 weeks, encompassing sales data from January 2013 to early April 2016. We excluded data from mid-April 2016 onward to mitigate potential distortions in sales distribution caused by a major earthquake in Ecuador. Figure 5 displays the weekly sales for five demand traces (first five plots) and summed across 32,768 demand traces (last plot), selected based on the criteria we now describe. The figure highlights varying orders of magnitude in sales across demand traces and significant heterogeneity in the effects of seasonality. Since our goal is to evaluate data-driven inventory decisions, as an initial pre-processing step we selectively choose demand traces that exhibit at least one sale in any of the first 16 periods.

A limitation of this dataset is that it only includes sales data, which may be censored due to stockouts. In an effort to align sales observations more closely with underlying demand, we try to filter out demand traces that were heavily influenced by stockouts. Interpreting zero-sale periods as proxies for stock-outs, we excluded all demand traces that exhibited no sales in at least 10% of the weeks. Moreover, we omitted all perishable products, anticipating that inventory replenishment would occur more frequently than on a weekly basis for such products. Finally, we randomly selected 32,768 sales demand traces from within the filtered sub-sample, treating them as demand paths for our experiments. It is important to note that in a real-world setting, retailers may have additional information, such as inventory availability or website visits, enabling better estimates of historical demand than raw sales data. Though an imperfect reflection of reality, we believe that the nonstationarity in real-world sales data still provides a good stress test of the performance of inventory policies.

5.2. Problem definition

After this pre-processing, we construct a testbed of inventory control problems using the dataset in the setting of a single store under a lost demand assumption. We draw inspiration from Madeka et al. (2022) in considering that each product provides a sampled scenario and training across products. Due to the absence of unit economics data, we assign random underage costs and lead

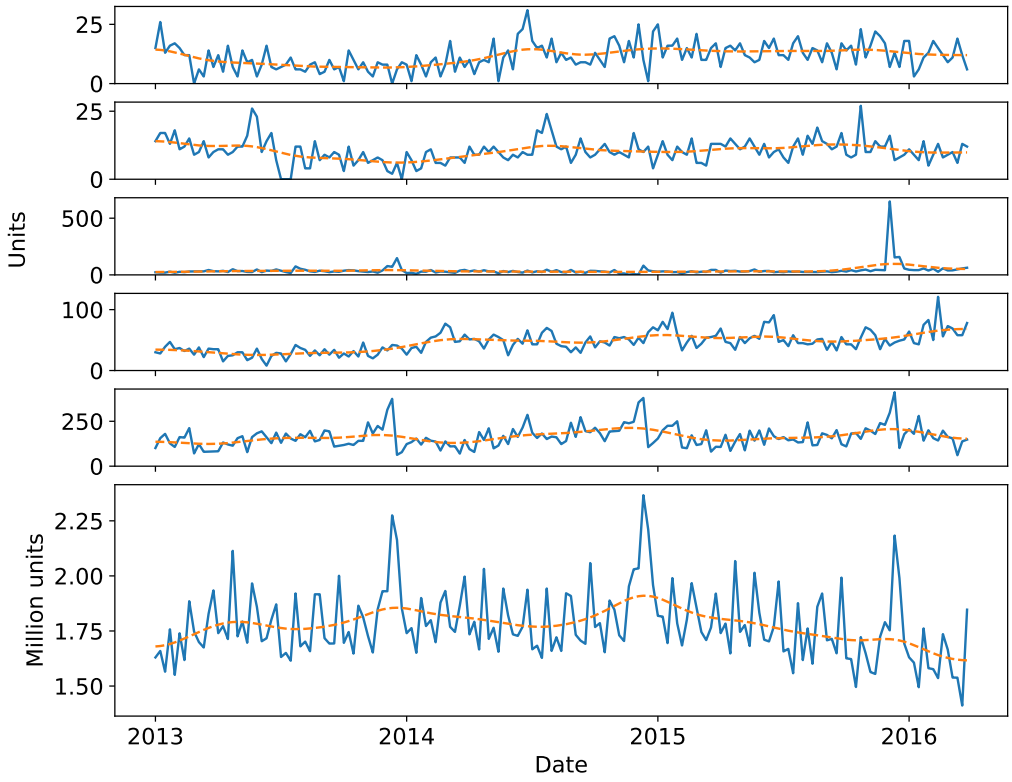


Figure 5 Weekly sales (blue line) and Gaussian smoothing (orange line) for 5 sample paths (first 5 plots) and summed across all 32,768 sample paths (last plot). Data corresponds to a filtered sub-sample of the dataset provided in the Corporación Favorita Grocery Sales Forecasting competition.

times to each demand trace while maintaining a constant holding cost. We generate multiple *meta-instances*, which define the procedure through which problem primitives are sampled (explained in Section 5.5). These meta-instances can represent, *e.g.*, industries with distinct average margins. A *scenario* is defined as a demand trace $\xi_{1:T}^h$ paired with an initial state S_0^h consisting of sampled problem primitives (*i.e.*, specifically fixed values for underage cost and lead time) and initial states of inventory and forecasting information. For simplicity in our terminology, we will say that a scenario relates to a specific product, although the associated demand trace pertains to a given (product, store) pair.

In our setting of interest, the decision-maker possesses information about a product's lead time, holding cost, and underage cost, and observes the current on-hand inventory, outstanding inventory orders, and historical demand observations for the product. The formulation of a HD-RL problem in Section 2.2 can represent the knowledge of problem primitives within each scenario through the state S_t, with transition and cost functions accounting for specific values associated with each set

of problem primitives. A policy, in this context, is a rule that maps the available information to an ordering decision. We can evaluate such a policy by repeatedly sampling scenarios, applying the policy in that scenario, and tracking the average cost incurred, as per Eq. (2). It is important to note that our inventory control formulation in Section 2.3, designed for simplicity, assumes fixed problem primitives, necessitating adjustments for accurate representation. Nonetheless, we will adopt most terminology from that section, excluding store-related superscripts due to the single-store nature of this setting. We represent the state as $S_t = (p, h, L, I_t, Q_t, \mathcal{F}_t)$, where p, h, and L represent underage cost, holding cost, and lead time, respectively, for the given scenario. Additionally, I_t, Q_t, and $\mathcal{F}_t$ denote variables tracking the inventory on-hand, outstanding orders, and forecasting information at time t. These variables evolve according to the equations presented in our instantiation of an inventory control problem in Section 2.3.

The testbed isolates a single challenging problem feature: nonstationarity in real-world demand patterns. By simplifying other problem features, we facilitate the adaptation of ideas from the OR literature to design heuristic policies. Notably, our application of HDPO did not appear to leverage these simplifying assumptions, and we expect that its performance would not undergo substantial changes if, for example, lead times were random.

5.3. State representation and input features

We train a separate NN for each meta-instance. We feed time-invariant features (*i.e.*, item-specific underage and holding cost) into our NNs. To manage the NN input size, we introduce a *lookback window* of length $s_1 \in \mathbb{N}$, denoting the number of preceding periods of demand that our agents utilize. Additionally, we input the count of days remaining until the next Christmas, as it appears to have a significant correlation with demand (see Figure 5).

The heuristics we compare against are assumed to have complete knowledge of lead time values, but we do not input this information directly to the policies trained by HDPO. Instead, we feed the list of all inventory orders and inventory arrivals (*i.e.*, units of inventory that arrived in each period) over a lookback window of the previous $s_2 \in \mathbb{N}$ weeks. This is consistent with how we input demand information to the network. We expect this method is more practical in settings where lead times are uncertain and time-varying, requiring a decision-maker to (at least implicitly) forecast them from observed data.

While supplying the NN with additional product-specific information (*e.g.*, product type) could enhance performance, we choose not to do so to simplify the development of practical heuristics against which we benchmark.

5.4. Benchmarks

We assessed our model by comparing it to several generalized newsvendor heuristics. These heuristics utilize a distribution forecaster trained offline, and aim to "raise" inventory levels to a level dictated by a given quantile. The *inventory position* $X_t = I_t + \sum_{l=1}^{L-1} q_{t-l}$ reflects the sum of inventory on-hand and orders yet to arrive. Denoting the distribution of the sum of the next $L+1$ demands, given lead time L and forecasting information $\mathcal{F}_t$, as $H(L, \mathcal{F}_t)$, a *generalized newsvendor policy* π takes the form

$$\pi(S_t) = \left(H(L, \mathcal{F}_t)^{-1}(\tau^\pi) - X_t\right)^+. \tag{11}$$

Here, τ^π represents a time-invariant quantile, with the flexibility to depend on static product features (*e.g.*, underage and holding costs) to accommodate heterogeneity across products.

The rationale for adopting this class of policies stems from the notion that, assuming that unmet demand is backlogged, the inventory on-hand before the order placed at time t arrives can be expressed as $I_{t+L} = X_t - \sum_{l=0}^{L-1} \xi_{t+l}$. Hence, under the backlogged demand assumption, the action minimizing the expected cost L periods into the future can be obtained as a quantile of the distribution of the sum of the next $L+1$ demands. Adjusting this quantile has the potential to account for variations in product economics (underage and holding costs) and underlying model misspecifications.

We trained a NN to forecast multiple quantiles for this metric using historical features. This allows us to approximate $H(L, \mathcal{F}_t)$ and hence estimate the specified quantiles. Additional information about the implementation and performance evaluation of the quantile forecaster can be found in Appendix B.10. We conducted two separate assessments to validate the good performance of our trained forecaster (see Figures 13a and 13b). We further validate the efficacy of the quantile forecaster by observing satisfactory results achieved by some generalized newsvendor benchmarks (to be introduced shortly) in meta-instances with lost demand and high average unit underage costs (refer to Figure 6a in Section 5.6) and in a backlogged demand setting (refer to Figure 7 in Section 5.7).

We considered several generalized newsvendor policies, which are equipped with the same quantile forecaster and have access to the problem primitives in each scenario. They only differ in the heuristic choice of the quantile τ^π:

- Newsvendor: Places orders up to the newsvendor quantile, given by $\frac{p}{p+h}$.
- Fixed Quantile: Considers a common quantile τ^π for all scenarios, with the quantile being a trainable parameter.
- Transformed Newsvendor: Utilizes a NN to flexibly learn a map from the newsvendor quantile $\frac{p}{p+h}$ to a new quantile.

For the last two heuristics, we optimize the parameters defining the policy by integrating the forecaster (with fixed parameters) into our simulator. Employing linear interpolation across the predicted quantiles enables us to optimize the parameters in a differentiable manner [7]. It is crucial to highlight that our forecaster is not trained at this stage.

We additionally considered two non-admissible policies, to serve as benchmarks and enable a better evaluation of HDPO.

- Returns Newsvendor: Resembles the Newsvendor policy but allows for "negative" orders, simulating the possibility of returning inventory. Conditional on the forecaster outputting the "true" distribution, this policy would lead to a lower bound on costs for settings involving backlogged demand.[8] It outputs the demand quantile that minimizes expected cost in every period, irrespective of the current inventory state.
- Just-in-time: An oracle policy that looks into the future and orders precisely to meet future demand (*i.e.*, sets $a_t = \xi_{t+L}$). The cost under the Just-in-time policy is a lower bound on the minimal achievable cost.

5.5. Parameter setting

Our main setting of interest considers a lost demand assumption. However, to identify the limitations of generalized newsvendor heuristics, we will also incorporate results under the assumption of backlogged demand, despite its departure from real-world settings (Corsten and Gruen 2004). For the backlogged demand setting, we trained our agents to minimize cumulative expected *cost*, defined as $\mathbb{E}\left[\sum_{t\in T}\left[p(\xi_t - I_t)^+ + h(I_t - \xi_t)^+\right]\right]$. In contrast, for the lost demand setting, we let p be the variable profit (excluding holding costs) per unit, and set the objective to maximizing $\mathbb{E}\left[\sum_{t\in T}\left[p\min\{\xi_t, I_t\} - h(I_t - \xi_t)^+\right]\right]$. Note that under a lost demand assumption, maximizing profit considering a variable profit per unit p is equivalent to minimizing cost considering p as the underage cost. We report the percentage of the maximum possible profit accrued by each policy.

We employ a Vanilla NN architecture with 2 hidden layers, each containing 64 neurons. The learning rate is set to 0.003, and the batch size is fixed at 8192. A lookback window of length 16 is utilized for the demand, and we monitor the 8 preceding orders and inventory arrivals to construct a data-driven representation of the inventory state. At the start of the planning horizon, we assume

[7] When predicting values outside the range of predicted quantiles, we employ linear interpolation using the slope of the nearest quantile range. For example, to predict an extremely low quantile, we interpolate with the same slope as the line connecting the first and second quantiles. It is worth noting that the vast majority of predicted quantiles fall within the range of predicted quantiles.

[8] Since the quantile forecaster is not specifically trained to minimize costs in the downstream objective, this policy does not necessarily lead to a lower bound on costs. Nevertheless, one may expect that it is challenging to outperform this benchmark.

zero on-hand inventory and no outstanding orders.[9] Additionally, we initialize previous orders and inventory arrivals to zero, incorporating an initial sequence of previous demands derived from real data.

Training and model selection occur during weeks 17-120, which we refer to as *train* set, allowing weeks 1-16 to be used as the initial sequence of previous demands. Performance evaluation takes place during weeks 121-170 (considering weeks 105-120 as the initial sequence of previous demands), which we refer to as *dev* set. Again, all quantities in the dev set are initialized according to the values explained in the preceding paragraph, ensuring independence between the train and dev sets. To mitigate the impact of the initial inventory state on performance, we exclude the first 16 periods when presenting costs (*i.e.*, weeks 17-32 and 121-136 in the train and dev sets, respectively). All metrics presented subsequently are based on performance in the dev set, unless explicitly stated otherwise. Although training and selecting models on the same dataset might result in overfitted models, we choose not to introduce an additional "test" set to leverage a larger set of weeks for training our agents.

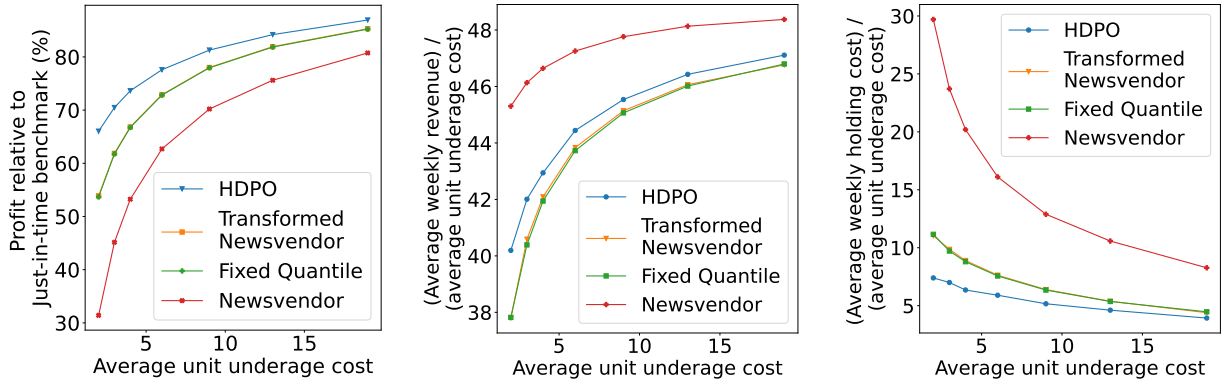
We set holding costs to 1 in all our experiments, creating seven meta-instances that differ solely in the *average unit underage cost* $\hat{p}$. We specified $\hat{p}$ values as $2, 3, 4, 6, 9, 13$, and 19. For a given meta-instance, the underage cost for each scenario was derived by multiplying $\hat{p}$ by a uniformly random number ranging from 0.7 to 1.3, independently drawn for each scenario. Simultaneously, lead times were uniformly drawn at random from 4 to 6 periods, maintaining independence across scenarios. The range of lead times remained constant across meta-instances, and every policy was retrained for each setting of $\hat{p}$.

5.6. Numerical results

Figure 6a presents the outcomes for the lost demand setting, across the 7 meta-instances detailed in Section 5.5. It depicts the total profit as a percentage of the value achieved by the Just-in-time benchmark. The figure illustrates that HDPO consistently outperforms all heuristics across instances, achieving over 80% of the hindsight optimal profit when $\hat{p}$ takes a value of 9 or larger. Our agent surpasses the best generalized newsvendor policy by up-to 22%, and the performance gap diminishes as the average unit underage cost increases—an aspect we scrutinize further in the subsequent subsection. Additional details on the performance of HDPO in this setting can be seen in Table 13 in Appendix B.9.

[9] Optimizing over generalized newsvendor policies with a time invariant target quantile may learn a smaller-than-ideal quantile if stock-outs in earlier periods are typically large, given that we assume zero on-hand inventory and no outstanding orders at the start of the planning horizon. To address this, we excluded the cost of the first 16 periods during quantile training (*i.e.*, we "disable" the gradient of the first 16 ordering actions) and the first 16 periods for reporting cost in the dev set, which allowed this influence to become negligible.

For further analysis, we compute the *average weekly revenue* and *average weekly holding cost* by summing the revenue $p\min\{\xi_t, I_t\}$ and holding cost $h(I_t - \xi_t)^+$ across all scenarios and weeks, then dividing by the total number of scenario-weeks. Figures 6b and 6c depict the average weekly revenue and holding cost, respectively, normalized by the average unit underage cost in the setting with lost demand. The figures illustrate that HDPO consistently achieves higher revenues and lower holding costs compared to competitive benchmarks. The exception is the newsvendor policy, which severely over-orders and incurs very high costs, but as a consequence does have higher revenues than HDPO. This emphasizes that adjusting a quantile alone cannot systematically compensate for a policy that merely tracks the inventory position (see Equation 11), which is further analyzed in Section 5.7.



(a) Profit relative to Just-in-time policy. Orange and green significantly lines overlap. **Higher is better.**

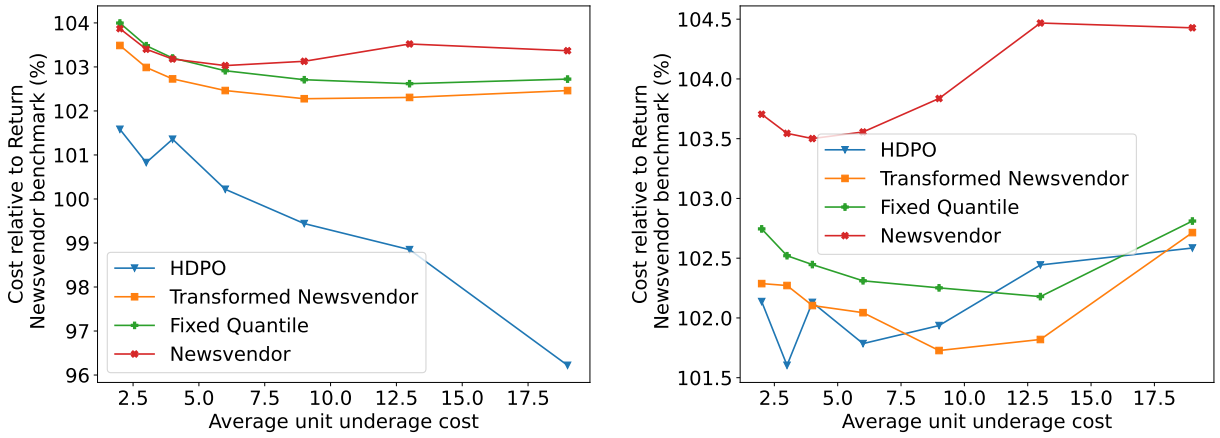
(b) Average weekly revenue, normalized by the average unit underage cost. **Higher is better.**

(c) Average weekly holding cost, normalized by the average unit underage cost. **Lower is better.**

Figure 6 Relative performance for setting with one store, considering $h = 1$, realistic demand data and a lost demand assumption, for different average unit underage costs.

5.7. What do generalized newsvendor policies miss?

Quantile policies are designed to be effective in a backlogged demand setting, where all incoming demand depletes the existing inventory. To empirically validate this assertion, Figures 7a and 7b depict the cost comparisons of various policies against the Return Newsvendor benchmark in the train and dev sets, respectively, assuming backlogged demand. These figures demonstrate the favorable performance of all generalized newsvendor policies evaluated, consistently achieving costs within 5% of those attained by the benchmark. Notably, the Transformed Newsvendor policy performs comparably to HDPO, and even outperforms it in certain meta-instances, although this might be attributed to overfitting (particularly evident in the low relative costs attained by HDPO in the train set for scenarios with high average unit underage costs).



(a) Relative cost on the train set.

(b) Relative cost on the dev set.

Figure 7 Cost relative to the Return Newsvendor benchmark for various policies under different average unit underage costs, assuming backlogged demand and a unit holding cost of 1.

In contrast, in a lost demand setting demand does not deplete inventory upon a stock-out, posing a challenge for policies optimized for backlogged demand settings. While adjusting the target quantile can partially address this issue, the underestimation of inventory may depend on the current inventory state in a non-trivial manner. As anticipated, in the lost demand setting, the performance of generalized newsvendor policies improves with an increase in average underage cost (see Figure 6a in Section 5.6), as unmet demand is expected to decrease. This observation aligns with previous analyses in stylized settings considering a lost demand assumption, where base-stock policies become asymptotically optimal as the underage cost grows large (Sun et al. 2014).

To conduct a more detailed analysis, we calculate the *implied quantile* that the agent orders up to by "inverting" the target level $(a_t + X_t)$ using our quantile forecaster. In other words, we find the τ^π that solves $H(L, \mathcal{F}_t)(\tau^\pi) = (a_t + X_t)$ following Equation (11) (with the implied quantile set to 0 when $a_t = 0$ since we cannot solve for τ^π in that case). We standardize this quantity by subtracting the mean and dividing by the standard deviation for each scenario. Subsequently, we define the *stock-out ratio* as the cumulative unmet demand until the order at time t arrives (*i.e.*, L periods into the future), divided by the scenario's average cumulative demand across L periods.

In Figure 8, we group the standardized implied quantile into buckets of width 0.1 and compute the sample average stock-out ratio for each bucket, for the meta-instance with $\hat{u} = 9$, and under a lost demand assumption. We interpret this as the expected stock-out ratio observed by the agent at time t when defining each quantile. The analysis reveals that our agent tends to place orders corresponding to lower quantiles when predicting significant stock-outs in subsequent periods, highlighting the potential underestimation of future inventory by a generalized newsvendor policy in such cases.

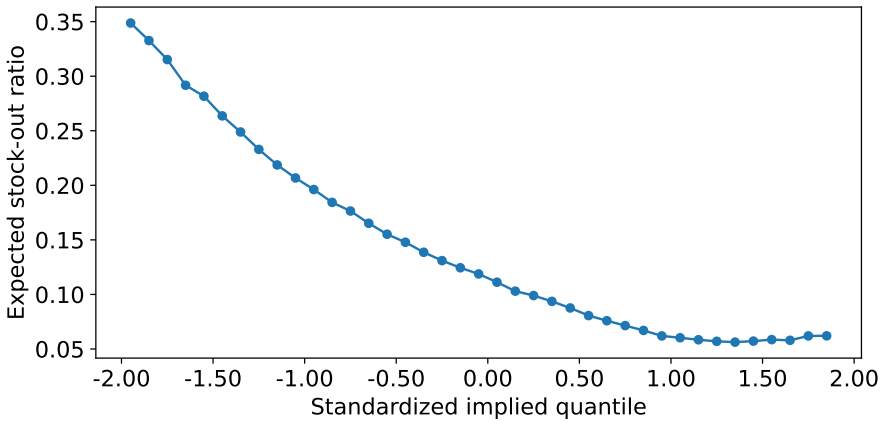


Figure 8 Standardized implied quantile versus expected stock-out ratio, considering an average unit underage cost of 9 and a lost demand assumption. The standardized implied quantile is binned with a width of 0.1, and the expected stock-out ratio is computed by averaging the sample stock-out ratio within each bin.

An additional drawback of generalized newsvendor policies lies in their failure to incorporate value functions. For products with pronounced seasonality, such as Christmas trees, excessive ordering may lead to incurring additional holding costs across multiple periods beyond the one in which orders arrive. In our experiments involving a backlogged demand setting (refer to Figure 7), the over-ordering indicated by the Newsvendor quantile can be partially mitigated by statically adjusting the target quantile, as demonstrated by the favorable performance of the Transformed Newsvendor policy. However, it is reasonable to anticipate that in a scenario featuring predictable short-term demand surges (such as planned promotions or national holidays), the effectiveness of generalized newsvendor policies might decline.

Finally, it is crucial to emphasize that constructing a generalized newsvendor policy, which involves building and training a forecaster for multiple time horizons and quantiles, requires significantly more computational resources and time compared to building and training an agent directly on the downstream task. Furthermore, determining how to divide a task into prediction and optimization stages becomes increasingly challenging as the problem incorporates more realistic features, such as the inclusion of random lead times. This underscores additional advantages of an end-to-end approach over strategies that separate prediction and decision-making.

6. Conclusion and future research

Previous research applying deep reinforcement learning to inventory managmenet problems has typically applied generic RL methods, like REINFORCE and its variants, and used generic neural network architectures. While such methods can often be made to work, (Gijsbrechts et al. 2021) reports that "initial tuning of the hyperparameters is computationally and time intensive and

requires both art and science." We systematically study two approaches to enhancing the reliability of DRL methods. The first is HDPO, a way of performing stochastic gradient optimization over the parameters of neural network policies that leverages important problem structure. The second is a symmetry-aware policy network architecture, which we find greatly enhances sample efficiency in a natural inventory network consisting of a single warehouse and a single store. Armed with these approaches, we consistently solve a range of inventory problems to optimality by searching over neural network policies that map "raw" high-dimensional state vectors to actions. Experiments with real time-series data point to substantial benefits of optimizing over a flexible policy class in an end-to-end fashion, rather than employ newsvendor type policies which are common in practice.

This paper suggests many promising future directions. First, one might seek out other problems in operations that are amenable to HDPO. Second, as discussed in the introduction, HDPO does not address problems with discrete actions and highly discontinuous costs, like those that arise when deciding whether to order a single shipment with a large bundle of goods or to delay that decision. It would be interesting to explore differentiable approximations to some these decision problems, or hybrid approaches which use REINFORCE to optimize over discrete decisions and HDPO to optimize over continuous ones. Lastly, it would be interesting to tackle problems with a more complex inventory network structure. Building on the intuition that inventory network should be reflected in the policy network, the use of graph neural networks as a policy architecture appears promising.

Acknowledgement

After beginning our numerical experiments, we learned of the work of Madeka et al. (2022). We thank them for their open and collaborative outlook, and their very helpful feedback.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. URL `https://www.tensorflow.org/`, software available from tensorflow.org.

Allen-Zhu Z, Li Y, Song Z (2019) A convergence theory for deep learning via over-parameterization. *International conference on machine learning*, 242–252 (PMLR).

Arrow KJ, Karlin S, Scarf HE, et al. (1958) Studies in the mathematical theory of inventory and production .

Bertsekas D (2012) *Dynamic programming and optimal control: Volume I*, volume 1 (Athena scientific).

Bertsekas DP (2011) Dynamic programming and optimal control 3rd edition, volume ii .

Bradbury J, Frostig R, Hawkins P, Johnson MJ, Leary C, Maclaurin D, Necula G, Paszke A, VanderPlas J, Wanderman-Milne S, Zhang Q (2018) JAX: composable transformations of Python+NumPy programs. URL `http://github.com/google/jax`.

Chen Z, Cao Y, Zou D, Gu Q (2019) How much over-parameterization is sufficient to learn deep relu networks? *arXiv preprint arXiv:1911.12360* .

Clark AJ, Scarf H (1960) Optimal policies for a multi-echelon inventory problem. *Management science* 6(4):475–490.

Corsten D, Gruen TW (2004) Stock-outs cause walkouts. *Harvard Business Review* 82(5):26–28.

Dai JG, Gluzman M (2022) Queueing network controls via deep reinforcement learning. *Stochastic Systems* 12(1):30–67.

De Kok T, Grob C, Laumanns M, Minner S, Rambau J, Schade K (2018) A typology and literature review on stochastic multi-echelon inventory models. *European Journal of Operational Research* 269(3):955–983.

Ding Y, Feng M, Liu G, Jiang W, Zhang C, Zhao L, Song L, Li H, Jin Y, Bian J (2022) Multi-agent reinforcement learning with shared resources for inventory management. *arXiv preprint arXiv:2212.07684* .

Favorita C (2017) Favorita grocery sales forecasting. URL `https://www.kaggle.com/competitions/favorita-grocery-sales-forecasting/overview`.

Federgruen A, Zipkin P (1984a) Approximations of dynamic, multilocation production and inventory problems. *Management Science* 30(1):69–84.

Federgruen A, Zipkin P (1984b) Computational issues in an infinite-horizon, multiechelon inventory model. *Operations Research* 32(4):818–836.

Feng J, Gluzman M, Dai JG (2021) Scalable deep reinforcement learning for ride-hailing. *2021 American Control Conference (ACC)*, 3743–3748 (IEEE).

Freeman CD, Frey E, Raichuk A, Girgin S, Mordatch I, Bachem O (2021) Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281* .

Gijsbrechts J, Boute RN, Van Mieghem JA, Zhang D (2021) Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems. *Manufacturing & Service Operations Management* .

Glasserman P (2004) *Monte Carlo methods in financial engineering*, volume 53 (Springer).

Glasserman P, Tayur S (1995) Sensitivity analysis for base-stock levels in multiechelon production-inventory systems. *Management Science* 41(2):263–281.

Greensmith E, Bartlett PL, Baxter J (2004) Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research* 5(9).

Haarnoja T, Zhou A, Hartikainen K, Tucker G, Ha S, Tan J, Kumar V, Zhu H, Gupta A, Abbeel P, et al. (2018) Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* .

Harsha P, Jagmohan A, Kalagnanam J, Quanz B, Singhvi D (2021) Math programming based reinforcement learning for multi-echelon inventory management. *Available at SSRN 3901070* .

Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D (2018) Deep reinforcement learning that matters. *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Hu Y, Anderson L, Li TM, Sun Q, Carr N, Ragan-Kelley J, Durand F (2019) Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935* .

Huang S, Dossa RFJ, Raffin A, Kanervisto A, Wang W (2022) The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023* .

Jiang Y, Neyshabur B, Mobahi H, Krishnan D, Bengio S (2019) Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178* .

Kakade SM (2001) A natural policy gradient. *Advances in neural information processing systems* 14.

Kaynov I, van Knippenberg M, Menkovski V, van Breemen A, van Jaarsveld W (2024) Deep reinforcement learning for one-warehouse multi-retailer inventory management. *International Journal of Production Economics* 267:109088.

Konda V, Tsitsiklis J (1999) Actor-critic algorithms. *Advances in neural information processing systems* 12.

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25.

Levine S, Finn C, Darrell T, Abbeel P (2016) End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17(1):1334–1373.

Liu X, Hu M, Peng Y, Yang Y (2022) Multi-agent deep reinforcement learning for multi-echelon inventory management. *Available at SSRN* .

Madeka D, Torkkola K, Eisenach C, Foster D, Luo A (2022) Deep inventory management. *arXiv preprint arXiv:2210.03137* .

Mania H, Guy A, Recht B (2018) Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055* .

Meuleau N, Hauskrecht M, Kim KE, Peshkin L, Kaelbling LP, Dean TL, Boutilier C (1998) Solving very large weakly coupled markov decision processes. *AAAI/IAAI*, 165–172.

Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, 1928–1937 (PMLR).

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. *nature* 518(7540):529–533.

Morton TE (1971) The near-myopic nature of the lagged-proportional-cost inventory problem with lost sales. *Operations Research* 19(7):1708–1716.

Oda T, Joe-Wong C (2018) Movi: A model-free approach to dynamic fleet management. *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2708–2716 (IEEE).

Oroojlooyjadid A, Nazari M, Snyder LV, Takáč M (2022) A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management* 24(1):285–304.

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, et al. (2019) Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32.

Powell WB (2007) *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703 (John Wiley & Sons).

Qi M, Shi Y, Qi Y, Ma C, Yuan R, Wu D, Shen ZJ (2023) A practical end-to-end inventory management model with deep learning. *Management Science* 69(2):759–773.

Reiman MI (2004) A new and simple policy for the continuous review lost sales inventory model. *Unpublished manuscript* .

Scarf H, Arrow K, Karlin S, Suppes P (1960) The optimality of (s, s) policies in the dynamic inventory problem. *Optimal pricing, inflation, and the cost of price adjustment* 49–56.

Schulman J, Levine S, Abbeel P, Jordan M, Moritz P (2015) Trust region policy optimization. *International conference on machine learning*, 1889–1897 (PMLR).

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .

Shar IE, Jiang DR (2023) Weakly coupled deep q-networks. *arXiv preprint arXiv:2310.18803* .

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, et al. (2017) Mastering the game of go without human knowledge. *nature* 550(7676):354–359.

Sinclair SR, Frujeri F, Cheng CA, Swaminathan A (2022) Hindsight learning for mdps with exogenous inputs. *arXiv preprint arXiv:2207.06272* .

Sun P, Wang K, Zipkin P (2014) Quadratic approximation of cost functions in lost sales and perishable inventory control problems. *Fuqua School of Business, Duke University, Durham, NC* .

Tang X, Qin Z, Zhang F, Wang Z, Xu Z, Ma Y, Zhu H, Ye J (2019) A deep value-network based approach for multi-driver order dispatching. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 1780–1790.

van Hezewijk L, Dellaert N, Van Woensel T, Gademann N (2023) Using the proximal policy optimisation algorithm for solving the stochastic capacitated lot sizing problem. *International Journal of Production Research* 61(6):1955–1978.

Van Roy B, Bertsekas DP, Lee Y, Tsitsiklis JN (1997) A neuro-dynamic programming approach to retailer inventory management. *Proceedings of the 36th IEEE Conference on Decision and Control*, volume 4, 4052–4057 (IEEE).

Vanvuchelen N, De Moor B, Boute R (2023) The use of continuous action representations to scale deep reinforcement learning for inventory control .

Vanvuchelen N, Gijsbrechts J, Boute R (2020) Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry* 119:103239.

Watkins CJ, Dayan P (1992) Q-learning. *Machine learning* 8:279–292.

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning* 5–32.

Xin L (2021) Understanding the performance of capped base-stock policies in lost-sales inventory models. *Operations Research* 69(1):61–70.

Zipkin P (2008) Old and new methods for lost-sales inventory systems. *Operations research* 56(5):1256–1263.

Appendix

Outline of the appendix

We divide the Appendix into 3 large sections. In Appendix A, we provide an overview of the computation of zeroth-order gradient estimators and offer a detailed description of the settings discussed in Section 3.3 that were not explicitly detailed in the main body. Additionally, we show how we obtained the optimal cost or lower bound, and detail well-performing heuristics. In Appendix B, we provide details on the implementation of our models, the process of hyperparameter tuning and numerical results. Finally, in Appendix C we provide a full proof of Theorem 1.

A. Description of problem settings and benchmarks for the numerical experiments in Section 3.3

In this Section, we provide an overview of the computation of zeroth-order gradient estimators and offer an overview of the settings discussed in Section 3.3 that were not explicitly detailed in the main body. Additionally, we describe the optimal policy structures for these settings whenever a straightforward one is known; otherwise, we provide a description of well-performing heuristics. Recall that, to streamline notation, we will exclude superscripts associated with location for settings where only one location is considered.

A.1. Zeroth-order estimators for inventory control

In this section, we briefly explain how zeroth-order estimators of gradients can be computed, which are commonly used in the REINFORCE algorithm and its variants. Let π_θ be a randomized policy such that, for each $S \in \mathcal{S}$, $\pi_\theta(\cdot|S)$ defines a probability distribution over actions $a \in \mathcal{A}(S)$. Define the policy loss $J(\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{t\in[T]} c(S_t, a_t, \xi_t)\right]$, representing the expected cumulative cost when following policy π_θ. The REINFORCE algorithm uses the following policy gradient formula

$$\nabla_\theta J(\theta) = \mathbb{E}^{\pi_\theta}\left[\left(\sum_{t\in[T]} G_t \frac{\nabla_\theta \pi_\theta(a_t|S_t)}{\pi_\theta(a_t|S_t)}\right)\right], \tag{12}$$

where $G_t = \sum_{u=t,\dots,T} c(S_u, a_u, \xi_u)$.

To obtain an estimator for the gradient of the policy loss, a collection of N trajectories, denoted as $\{\tau^n = (S_1^n, a_1^n, c_1^n, \dots, S_T^n, a_T^n, c_T^n)\}_{n\in[N]}$, must be generated via policy rollouts following each gradient update. Here, S_t^n represents the state, a_t^n signifies the action sampled from $\pi_\theta(\cdot|S_t^n)$, and c_t^n denotes the corresponding incurred cost at time step t within policy rollout n. Subsequently, the gradient is estimated as:

$$\frac{1}{N}\sum_{n\in[N]}\left[\left(\sum_{t\in[T]} G_t^n \frac{\nabla_\theta \pi_\theta(a_t^n|S_t^n)}{\pi_\theta(a_t^n|S_t^n)}\right)\right], \tag{13}$$

with $G_t^n = \sum_{u=t,\dots,T} c_u^n$ representing the cumulative cost from time period t onwards for rollout n.

This method has at least two substantial drawbacks. First, estimating the policy gradient requires employing that policy for many periods to gather data. This is feasible when a high fidelity simulator is available, but may be impractical otherwise. Second, while the estimator (13) is unbiased, its variance grows with the time horizon T and can explode if policies become nearly deterministic, since the inverse propensity weights $1/\pi_\theta(a_t^n|S_t^n)$ explode. The structure of hindsight differentiable RL problems alleviates these challenges, making it easy to backtest the performance of new polices without additional data gathering and enabling gradient estimation without inverse propensity weighting.

A.2. One store, no warehouse. Backlogged demand assumption.

This setting consists of a single store with known per-unit underage and holding costs and a deterministic lead time L. In each period, the store faces an independent, identically distributed (i.i.d.) demand with distribution F, and has access to a supplier with infinite inventory. Under linear costs, a base-stock policy is optimal (Arrow et al. 1958). We define the store's inventory position at time t as

$$X_t = I_t + \sum_{l=1}^{L-1} q_{t-l}. \tag{14}$$

The optimal policy then takes the form

$$\pi(S_t) = \left(\hat{S} - X_t\right)^+, \tag{15}$$

for some constant *base-stock level* $\hat{S} \in \mathbb{R}$. Furthermore, as there are no procurement costs, the optimal base stock level can be calculated as

$$\hat{S} = \left(\hat{F}\right)^{-1} \left(\frac{p}{p+h}\right), \tag{16}$$

where $\hat{F}$ is the distribution of the cumulative demand in $L+1$ periods.

A.3. One store, no warehouse. Lost demand assumption.

We consider the setting defined in Section 3.3.1. This setting is identical to that in Appendix A.2, except that demand is assumed to be lost if not satisfied immediately (*e.g.*, consumers may buy from the competition if a product is not available). This assumption may better represent reality especially in brick and mortar retail, as depending on product category, only $9-22\%$ of customers are willing to delay their purchase when not finding a specific item at a store (Corsten and Gruen 2004).

A base-stock policy is no longer optimal in this setting under a lead time of at least of 1 period, and its performance deteriorates as the lead time grows. Furthermore, the optimal policy does not appear to have a simple structure, and may depend on the complete inventory pipeline. Finding an exact solution to this problem is computationally intractable, even for instances of moderate size. Nevertheless, Zipkin (2008) managed to solve the underlying dynamic program for smaller instances, serving as the benchmark against which we evaluate our algorithms in both Section 3.3.1 and Appendix B.5.

Many heuristics have been suggested for this setting. One of the top-performing heuristics in this context is based on the class of Capped Base-Stock (CBS) policies, as discussed in (Xin 2021). These policies demonstrate an average optimality gap of 0.71% across the 32-instance test-bed introduced in (Zipkin 2008). A CBS policy is defined by a base stock level $\hat{S}$ and an order cap r, as per

$$\pi(S_t) = \min\left\{\left(\hat{S} - X_t\right)^+, r\right\}. \tag{17}$$

A.4. Serial network structure. Backlogged demand assumption.

In this setting, the system consists of K echelons, each comprising a single location. Locations are sequentially numbered from 1 to K, starting from the upstream and progressing towards the downstream. Demand arises solely at the most downstream location (location K), which we refer to as the *store*, and any unfilled demand is backlogged. Inventory costs, denoted as h^k, are incurred at each location, and there is an underage cost

of p per unit at the most downstream location. Each location has a positive lead time denoted as $L^k \in \mathbb{N}$. It is assumed that demand is i.i.d. over time.

The sequence of events is as follows: A central planner observes the state S_t at time t and jointly determines the order a_t^1 that the first location places with an external supplier with unlimited inventory and, for $k = 2, \ldots, K$, specifies the quantity a_t^k to transfer from location $k-1$ to location k, ensuring that it does not exceed the current inventory level at location $k-1$ (*i.e.*, , $a_t^k \leq I_t^{k-1}$).

We define the *echelon inventory position* Y_t^k of location k at time t as

$$Y_t^k = \sum_{j=k}^{K} X_t^k, \tag{18}$$

with X_t^k the inventory position of location k as defined in (14). That is, it represents the sum over all inventory positions of downstream locations and its own. As shown by Clark and Scarf (1960), the optimal policy π takes the form

$$\begin{aligned} [\pi(S_t)]_1 &= [\hat{S}^1 - Y_t^1]^+ \\ [\pi(S_t)]_k &= \min\{I_t^{k-1}, [\hat{S}^k - Y_t^k]^+\} \quad k = 2, \ldots, K \end{aligned} \tag{19}$$

for some *echelon base-stock levels* $\hat{S}^1, \ldots, \hat{S}^K$. We note that we will use the terms echelon base-stock levels and echelon-stock levels interchangeably. We refer to (19) as an *Echelon-stock policy.*

A.5. Many stores, one transshipment center. Backlogged demand assumption.

We consider the setting described in Section 4.3, where a warehouse operates as a transshipment center (*i.e.*, cannot hold inventory) and there are multiple stores, under a backlogged demand assumption. The demand is i.i.d. across time but may be correlated across stores. The costs incurred solely consider holding and underage costs at the stores.

Federgruen and Zipkin (1984a) provides a clever way to obtain a lower bound when demand at the stores follows a jointly normal demand, and store costs and lead times are identical. To leverage their results, we assume that all stores have the same underage costs denoted as p^1 and lead time denoted as L^1, while the transshipment center has a lead time of L^0. For each store k, we consider a marginal demand that follows a normal distribution with mean μ^k and standard deviation σ^k. Furthermore, we represent the covariance matrix as Σ, where $[\Sigma]_{ij}$ denotes the covariance $\mathrm{Cov}(\xi_1^i, \xi_1^j)$ between the demands of stores i and j.

The *echelon inventory position* at the transshipment center takes the form

$$Y_t^0 = \sum_{k \in [K]_0} X_t^k, \tag{20}$$

where X_t^k denotes the inventory position of location k as defined in (14). That is, Y_t^0 represents the sum of the inventory position across stores plus its own. The authors consider a relaxation of the problem by allowing inventory to flow from stores to the transshipment center and to other stores. The authors demonstrate that it is possible to reformulate the problem using the echelon inventory position Y_t^0 as the state variable. They establish that the Bellman Equation in this relaxed setting can be expressed as a single-location inventory problem with convex costs. Consequently, they deduce that an optimal policy for the transshipment center, in the relaxed problem, is an echelon-stock policy of the form

$$[\pi(S_t)]_1 = [\hat{S}^0 - Y_t^0]^+ \tag{21}$$

for some echelon base-stock level $\hat{S}^0 \in \mathbb{R}_+$. Finally, as we do not consider purchase costs, the optimal base-stock level $\hat{S}^0$ can be calculated analytically via the newsvendor-type formula

$$\hat{S}^0 = F_G^{-1}\left(\frac{p^1}{p^1 + h^1}\right), \tag{22}$$

where G is a normal distribution with mean $\hat{\mu}_G$ and standard deviation $\hat{\sigma}_G$ given by

$$\hat{\mu}_G = (L^0 + L^1 + 1) \sum_{k\in[K]} \mu^k$$

$$\hat{\sigma}_G = \sqrt{L^0 \sum_{i\in[K]} \sum_{j\in[K]} [\Sigma]_{ij} + (L^1 + 1)\left(\sum_{i\in[K]} \sigma^i\right)^2}.$$

Now, let $\hat{s} = \frac{\hat{S}^0 - \hat{\mu}_G}{\hat{\sigma}_G}$ be a standardized version of $\hat{S}$. The lower bound on costs per-period is finally given by

$$p^1(\hat{\mu}_G - \hat{S}^0) + (p^1 + h^1)\hat{\sigma}_G\left(\hat{s}\Phi(\hat{s}) + \phi(\hat{s})\right),$$

with $\Phi(\cdot)$ and $\phi(\cdot)$ the Cumulative Distribution Function and Probability Density Function, respectively, of a standard Normal distribution.

We note that the previous quantity is not normalized by the number of stores.

B. Implementation details and numerical experiments

This section offers a comprehensive account of our implementation and numerical experiments. In Appendix B.1, we define terminology, outline training procedures, and provide details of our PyTorch implementation. Appendix B.2 details the feasibility enforcement functions used for settings considering multiple locations. Appendix B.3 specifies the best-performing hyperparameter setting and initialization procedure for each problem setting. In appendices B.4 to B.9, we present all information required to replicate our experiments accurately, including experimental setup, results, and analysis. Finally, Appendix B.10 provides details on our quantile forecaster and its performance.

B.1. Implementation details

Terminology and reporting conventions

We briefly introduce the terminology required to follow our experiments and setup

- Hyperparameter: A parameter that is set before the learning process begins and affects the model's behavior or performance.
- NN architecture class: Overall topology and organization of the network, which determines how information flows through the network and how computations are performed.
- Epoch: A complete pass through the entire training dataset during the training process.
- Batch: A subset of the training dataset used for updating model parameters.
- Batch size: The number of scenarios included in each batch.
- Gradient step: A step taken to update the model's parameters based on the gradient computed over one batch.
- Learning rate: A hyperparameter that determines the step size of the gradient step.
- Hidden layer: A layer in a NN that sits between the input and output layers and performs intermediate computations.
- Unit/neurons: Fundamental unit of a NN that receives input, performs a computation, and produces an output.

We outline the reporting convention that will be followed throughout this section.

- Units per layer: The count of units or neurons present in each hidden layer.
- Loss: The average cost incurred per period and per store in a given dataset. Calculated by dividing the total cost accumulated across the periods considered by the number of stores and periods considered.
- Gap: The percentage difference between the cost incurred by a policy and a predefined benchmark cost. The benchmark cost can take the form of the optimal cost, a lower bound on costs, or the cost incurred by a heuristic approach.

Initialization

We analyzed two different procedures to initialize the on-hand inventories I_1^k and vectors of outstanding orders Q_1^k for stores.

- Uniform: letting $\hat{\mu}^k$ be the sample mean demand for store k, we initialized I_1^k and every entry in Q_t^k by sampling independently (across Q and I, and across (k,t)) from Uniform$(0,\hat{\mu}^k)$.
- Set to 0: we simply set I_1^k and every entry in Q_t^k to 0.

In the experiments outlined in Sections 3.3 and 4.3, we utilized the uniform initialization procedure, selected for its tendency to expedite learning. Nevertheless, in the setting involving real data discussed in Section 5, we choose to set the initial inventory to 0 to mitigate any additional impact of the algorithm's performance associated with the chosen initialization procedure. For the warehouse's inventory on-hand I_1^0 and vector of outstanding orders Q_1^0, we set each entry to 0 across all experiments.

Hyperparameter tuning

We found that our method exhibited robustness to the choice of hyperparameters (especially for single-location settings), requiring minimal tuning effort in most cases. Initially, we conducted experiments to explore various combinations of hyperparameters such as learning rates, batch sizes, number of hidden layers, and number of neurons per layer. Through this process, we identified a set of hyperparameter combinations that yielded satisfactory performance. However, when dealing with settings involving multiple locations, we performed some ad-hoc tuning by searching for hyperparameter values on a grid.

Technical implementation specifications

We developed a differentiable simulator using PyTorch and conducted all experiments on an NVIDIA A40 GPU with 48GB of memory. To expedite the training process, we implemented an efficient parallel computation scheme. For a given mini-batch of H scenarios, we simultaneously executed the forward pass first, followed by the backward pass, across all scenarios. To achieve this, we utilized an initial mini-batch "state" matrix denoted as $\tilde{S}_1^H$. This matrix was obtained by stacking the initial states $\tilde{S}_1^h$ for each scenario $h \in H$. At each time period, we input the matrix $\tilde{S}_t^H$ into the NN, enabling us to obtain the outputs for every scenario in a highly parallelizable manner. Subsequently, we computed the costs c_t and updated the mini-batch state matrix $\tilde{S}_{t+1}^H$ through efficient matrix computations. This approach allowed PyTorch to efficiently estimate the gradients during the backward pass.

B.2. Feasibility enforcement functions for settings in Sections 3.3 and 4.3

Table 5 provides an overview of the feasibility enforcement functions employed in settings with multiple locations for each architecture class. Subsequently, we offer a comprehensive description of the architecture adopted for the serial system, along with brief commentary on the selection of feasibility enforcement functions for each specific setting. For the serial system, we assume that NNs output allocation a^1 for the most upstream location, and intermediate outputs b^k for each $k \in \{2,\ldots,K\}$. Subsequently, the feasibility enforcement function g, which is applicable for $k \in \{2,\ldots,K\}$, takes two inputs: the current inventory level I^{k-1} at location $k-1$, and the intermediate output b^k corresponding to the immediately downstream location k. It then generates a feasible action for location k.

For the serial network structure, we initially experimented with a feasibility enforcement function analogous to proportional allocation. In this setup, for $k = 2,\ldots,K$, the k-th intermediate output of the NN was initially considered as a tentative allocation for ordering from the parent location in the supply chain. These tentative

Table 5 Feasibility enforcement functions used for each setting with a network structure and for each architecture class. For the serial system, we outline the feasibility enforcement functions considered for each location, except the most upstream one.

Setting description	Architecture class	Feasibility enforcement function
Serial system. Backlogged demand (B.6)	Vanilla	Softmax
Many stores, one transshipment center. Backlogged demand (B.7)	Vanilla	Softmax without constant
Many stores, one warehouse. Lost demand (B.8)	Vanilla	Softmax
	Symmetry-aware	Proportional allocation

allocations were capped at the inventory available at the parent location. However, we encountered situations where this policy became trapped in what we believe were suboptimal local optima. To address this issue, we utilized a Sigmoid function, noting its equivalence to applying Softmax to a single input. Consequently, if b^k represents the k-th output and I^{k-1} denotes the inventory on-hand at the preceding location, the final allocations were calculated as $a^k = \frac{\exp(b^k)}{1+\exp(b^k)} \times I^{k-1}$. This modified approach consistently yielded near-optimal performance in our experiments. For the first location, we employed the "trick" described in Section 3, in which we apply a Sigmoid activation function and multiply by a crude upper bound.

In the setting with one transshipment center and multiple stores under backlogged demand (second row in Table 5), we exclusively assessed Softmax without Constant as a feasibility enforcement function. This choice stems from the fact that the warehouse lacks the capacity to store inventory, and utilizing this feasibility enforcement function inherently enforces this constraint directly.

In the setting with one warehouse and multiple stores under lost demand (third and fourth rows in Table 5), we examined various functions for the Vanilla and Symmetry-aware NNs. Notably, Softmax yielded best performance for the former, while Proportional Allocation proved most effective for the latter. This aligns with the nature of weak coupling among store orders in the Symmetry-aware NN, where a proportional allocation allows for only a "weak" relationship between intermediate store outputs.

B.3. Hyperparameter settings for numerical experiments

Table 6 presents well-performing hyperparameter combinations for each setting in Section 3.3 for the Vanilla NN. The reported results for fixed hyperparameters are based on these hyperparameters. When assessing different hyperparameters, we choose the ones that minimize cost on the dev set. It is worth noting that a wide range of hyperparameters generally yield satisfactory performance for experiments in Section 3.3. Unless explicitly stated, we employed 32,768 scenarios for the training, dev, and test sets in all experiments. Results demonstrating robustness to hyperparameter choices in a single-store lost-demand setting were presented in Section 3.3.1 in Table 3.

The hyperparameters used in the experiments in Section 4.3 are documented in Tables 11 and 12 in Appendix B.8. We note that in the experiments conducted in this section, the extensive exploration of learning rates is motivated by empirical evidence suggesting that larger learning rates contribute to improved

generalization (Jiang et al. 2019). Indeed, particularly for the Vanilla NN in low-data regimes, the test set loss was typically minimized by one of the largest learning rates among those leading to low training losses.

Hyperparameter choices for experiments involving real data were already detailed in the main body of the paper. See Section 5.5.

Table 6 Hyperparameter settings for the Vanilla NN for each setting in Section 3.3. For the second row, we consider the best-performing set of hyperparameters among the ones assessed (see table 3).

Setting description	Hidden layers (#)	Units per layer	Learning rate	Batch size
One store, no warehouse. Backlogged demand (B.4)	3	32	0.001	8192
One store, no warehouse. Lost demand (B.5)	3	32	0.01	1024
Serial network structure. Backlogged demand (B.6)	2	32	0.01	8192
Many stores, one transshipment center. Backlogged demand. (B.7)	3	256	0.0001	1024

B.4. One store, no warehouse. Backlogged demand assumption.

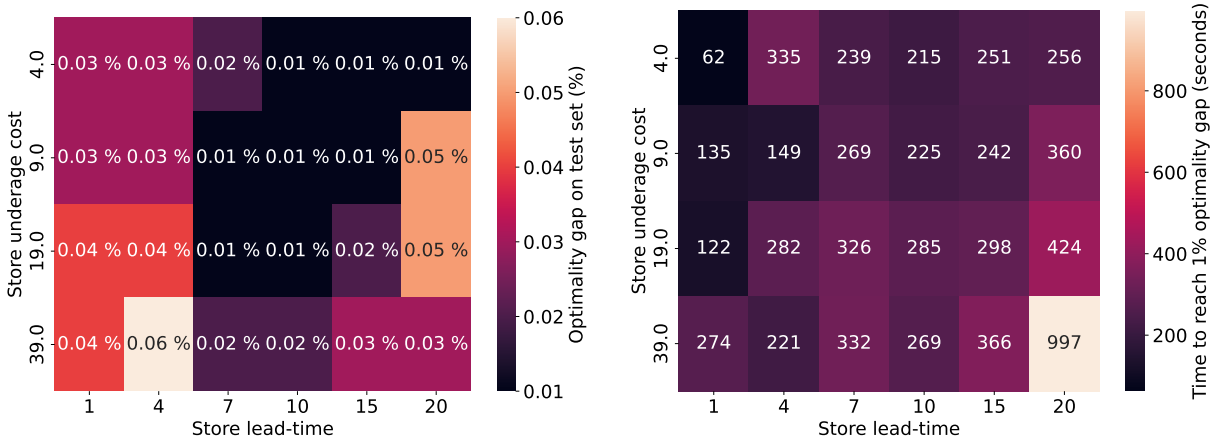
We generate demand traces by sampling from a normal distribution with a mean of $\mu = 5.0$ and a standard deviation of $\sigma = 1.6$, truncating it from below at 0. We created 24 instances by setting $h = 1$, $p = 4, 9, 19, 39$ and $L = 1, 4, 7, 10, 15, 20$. We compare our model with the optimal base-stock policy computed according to (15) and (16) in Appendix A.3.

Figures 9a and 9b, respectively, show the optimality gap on the test set and time to reach 1% of optimality gap on the dev set. Our approach achieves an average gap of 0.03% across instances and takes less than 4 minutes, on average, to obtain a 1% gap. Further, gaps are consistently below 0.1% across instances, even for long lead times of up-to 20 periods. We report additional performance indicators in Table 7.

B.5. One store, no warehouse. Lost demand assumption.

The details of the test-bed considered for this setting can be found in Section 3.3.1. In Table 8, we report the performance under the hyperparameter setting that minimizes loss on the dev set (see Table 6 in Appendix B.3). This table illustrates the reliability of HDPO, achieving results within 1% of optimality in under 70 seconds for all but one instance. Moreover, Figures 10a and 10b show the learning curves on one instance for two "extreme" choices of hyperparameters, revealing stable learning and rapid convergence to near-optimal solutions across different hyperparameter settings (note that an epoch corresponds to 4 and 32 gradients steps for the plots on the left and right, respectively).

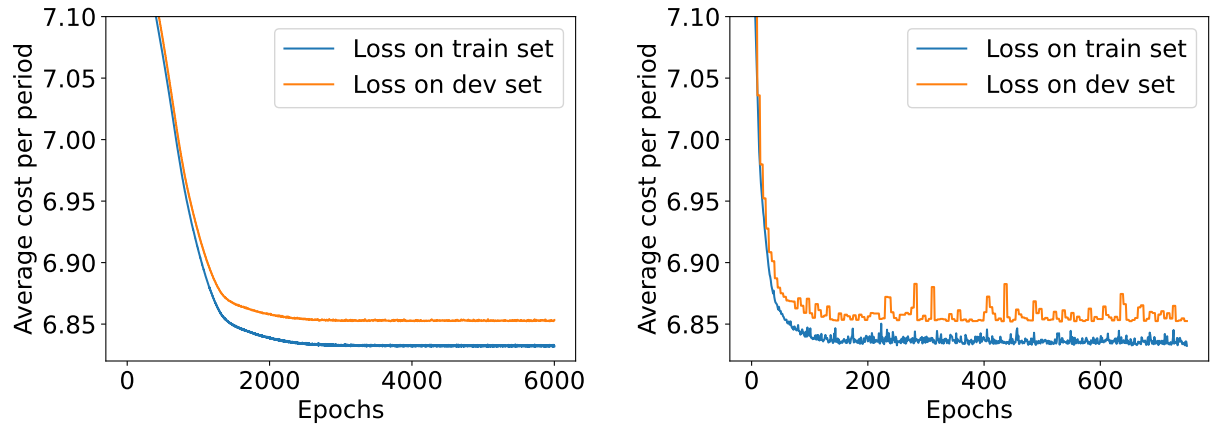
In Figure 11 we plot the inventory position (see Eq. (14)) and allocation under the Vanilla NN policy for 2 settings and compare it to the allocation under the optimal CBS policy (red line). We randomly jitter points for visibility, and color points according to the current inventory on-hand. We observe that the structure of the policy learned by our Vanilla NN somewhat resembles a CBS policy, but our learned policy is able to



(a) Optimality gap on the test set.

(b) Time to reach 1% opt. gap on the dev set.

Figure 9 Optimality gap on the test set and time to reach 1% of optimality gap on the dev set for the one store under backlogged demand setting for different underage costs and lead times. Training was halted after 4, 000 epochs, which took around 30 minutes for every run.



(a) 2 hidden layers, learning rate of 10^{-4} and batch size of 8192.

(b) 3 hidden layers, learning rate of 10^{-2} and batch size of 1024.

Figure 10 Epochs vs average cost per period on train and dev set considering continuous allocation, for different choice of hyperparameters. Setting of one store under a lost demand assumption, with unit underage cost of 9 and lead time of 4. Note that an epoch corresponds to 4 and 32 gradients steps for the plots on the left and right, respectively.

use additional information in the state space to achieve lower costs. For a fixed inventory position, the NN tends to order less for lower inventory on hand, as a stock-out is more likely under such a scenario in which case less inventory will actually be depleted.

B.6. Serial network structure. Backlogged demand assumption.

We analyzed a serial network structure with 4 echelons. We considered Normal demand with mean $\mu = 5.0$ and standard deviation $\sigma = 2.0$, and truncated it at 0 from below. We fixed holding costs as $h^1 = 0.1$, $h^2 = 0.2$,

Table 7 Performance metrics of the Vanilla NN for each instance of the setting with one store under a backlogged demand assumption.

Store lead-time	Store under-age cost	Train loss	Dev loss	Test loss	Train gap (%)	Dev gap (%)	Test gap (%)	Time to 1% dev gap (s)
1	4	3.17	3.17	3.17	0.02	0.03	0.03	62
1	9	3.97	3.97	3.97	0.03	0.03	0.03	135
1	19	4.66	4.68	4.67	0.04	0.04	0.04	122
1	39	5.29	5.29	5.29	0.04	0.04	0.04	274
4	4	5.03	5.02	5.01	0.02	0.02	0.03	335
4	9	6.28	6.30	6.28	0.03	0.02	0.03	149
4	19	7.36	7.37	7.38	0.04	0.03	0.04	282
4	39	8.36	8.36	8.36	0.06	0.04	0.06	221
7	4	6.33	6.33	6.34	0.02	0.02	0.02	239
7	9	7.93	7.94	7.93	0.02	0.01	0.01	269
7	19	9.41	9.39	9.34	0.07	0.01	0.01	326
7	39	10.56	10.61	10.56	0.03	0.01	0.02	332
10	4	7.45	7.41	7.43	0.04	0.01	0.01	215
10	9	9.32	9.29	9.31	0.07	0.01	0.01	225
10	19	10.92	11.00	10.95	0.05	0.01	0.01	285
10	39	12.45	12.42	12.42	0.07	0.01	0.02	269
15	4	8.97	8.94	8.94	0.08	0.00	0.01	251
15	9	11.48	11.22	11.23	0.04	0.01	0.01	242
15	19	13.27	13.20	13.21	0.02	0.02	0.02	298
15	39	14.91	14.89	14.94	0.06	0.03	0.03	366
20	4	10.28	10.29	10.24	0.09	0.00	0.01	256
20	9	12.98	12.86	12.87	0.30	0.06	0.05	360
20	19	15.01	15.19	15.13	0.05	0.12	0.05	424
20	39	17.37	17.12	17.14	0.01	0.12	0.03	997

$h^3 = 0.5$ and $h^4 = 1.0$. The lead times for the first 3 echelons are fixed as $L^1 = 2$, $L^2 = 4$ and $L^3 = 3$. We created 16 instances by setting $p = 4, 9, 19, 39$ and $L^4 = 1, 2, 3, 4$.

Within this setting, there exists an optimal echelon-stock policy (see Eq. (19) in Appendix A.4). We employ our differentiable simulator to search for the best-performing base-stock levels $\hat{S}^1, \ldots, \hat{S}^K$ through multiple runs. We then compare our NNs with the best-performing echelon-stock policy obtained, which we take as the optimal cost.

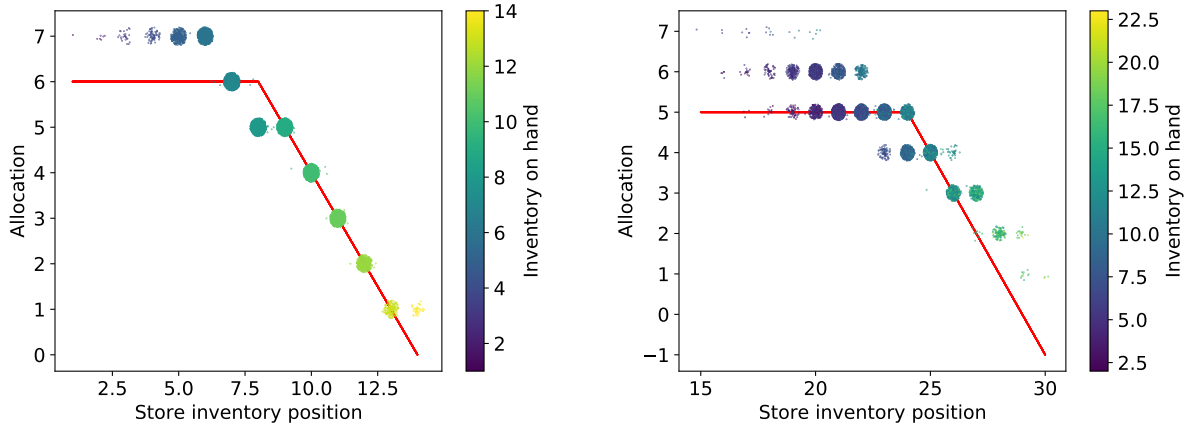
Table 9 describes the performance of the Vanilla NN for each instance of the serial network structure described in Appendix A.4, for the hyperparameters in Table 6. The Vanilla NN achieved an average optimality gap of 0.35%, and needed around 1500 gradient steps, on average, to achieve a gap smaller than 1% in the dev set.

B.7. Many stores, one transshipment center. Backlogged demand assumption.

Table 10 summarizes the performance of the Vanilla NN for the setting described in Appendix A.5, considering the experimental setup described in Section 3.3.2, for the hyperparameters in Table 6, and reveals a consistent near-optimal performance across instances. HDPO required approximately 8,000 gradient steps and 53 minutes, on average, to achieve 1% of optimality gap on the dev set.

Table 8 **Performance metrics of the Vanilla NN for each instance of the setting with one store under the lost demand assumption for the best hyperparameter setting. For the last two columns we consider the performance of the NN with continuous allocation as proxy for the performance in the discrete-allocation setting, as costs changed, on average, by less than 1% after discretizing the allocation.**

Store lead time	Store underage cost	Test loss	Test gap (%)	Gradient steps to 1% dev gap	Time to 1% dev gap (s)
1	4	4.04	<0.25	160	10
1	9	5.44	<0.25	320	19
1	19	6.67	<0.25	160	11
1	39	7.84	<0.25	480	69
2	4	4.40	<0.25	3680	247
2	9	6.09	<0.25	160	10
2	19	7.67	<0.25	640	45
2	39	9.10	<0.25	480	74
3	4	4.60	<0.25	480	67
3	9	6.53	<0.25	320	24
3	19	8.36	<0.25	160	11
3	39	10.04	<0.25	1120	79
4	4	4.73	<0.25	480	79
4	9	6.84	<0.25	960	72
4	19	8.88	<0.25	480	34
4	39	10.79	<0.25	640	45



(a) Lead time of 1 period (b) Lead time of 4 periods

Figure 11 **Inventory position vs allocation under Vanilla NN for setting with one store under lost demand assumption, Poisson demand with mean 5 and unit underage cost of 9. Points are randomly jittered for clarity. Point colors correspond to current inventory on-hand, and the red line captures the allocation under the optimal CBS policy (reported in Xin (2021)).**

Table 9 Performance metrics for the Vanilla NN for each instance of the serial network structure.

Store lead-time	Store under-age cost	Train loss	Dev loss	Test loss	Train gap (%)	Dev gap (%)	Test gap (%)	Gradient steps to 1% dev gap	Time to 1% dev gap (s)
1	4	6.91	6.91	6.91	0.42	0.23	0.26	1912	497
1	9	8.36	8.39	8.38	0.25	0.22	0.24	900	218
1	19	9.62	9.63	9.63	0.48	0.21	0.24	2012	494
1	39	10.71	10.73	10.72	0.44	0.17	0.19	644	159
2	4	7.62	7.61	7.61	0.61	0.28	0.31	1408	324
2	9	9.26	9.28	9.28	0.41	0.24	0.27	1168	287
2	19	10.68	10.70	10.70	0.44	0.31	0.35	1780	442
2	39	11.96	11.98	11.99	0.80	0.60	0.69	4020	1004
3	4	8.21	8.22	8.22	0.40	0.30	0.34	1008	263
3	9	10.05	10.06	10.06	0.56	0.27	0.31	1492	370
3	19	11.59	11.61	11.61	0.42	0.21	0.27	1044	262
3	39	13.00	13.01	13.01	0.72	0.33	0.42	2040	515
4	4	8.80	8.78	8.78	0.85	0.44	0.48	1596	433
4	9	10.74	10.76	10.76	0.48	0.35	0.37	896	224
4	19	12.41	12.45	12.45	0.38	0.36	0.38	912	233
4	39	13.98	13.96	13.97	0.87	0.40	0.48	1272	327

B.8. Many stores, one warehouse. Lost demand assumption.

The details of the experimental setup for this setting are outlined in Section 4.3. Table 11 presents various metrics for the first experimental setup, with Figure 4a in 4.3 serving as a summary of results. The findings indicate that, for training scenarios of 16 and 256, the Vanilla NN struggles to generalize effectively, evidenced by significant gaps between losses in the train and dev sets. These gaps, as well as the test loss, escalate as the number of stores increases, reaching disparities of approximately 8% between performance in the train and dev sets. This leads to relative test losses of up to 123%. On the other hand, the Symmetry-aware NN showcases robust generalization with 256 training scenarios, consistently maintaining a difference in train and dev losses below 2% and relative test losses below 102%.

Table 12 presents various metrics for the second experimental setup, with Figure 4b in Section 4.3 serving as a summary of results. It highlights the good performance overall across all training sample sizes, achieving mean relative test losses below 113% even for a single training scenario.

B.9. Real demand data

We consider the setting and experiment specifications detailed in Section 5. Tables 13 and 14 showcase the performance of our approach across different average unit underage costs, accounting for lost and backlogged demand assumptions, respectively. It is important to note that, despite including the elapsed time and the number of gradient steps required to achieve a 1% performance gap relative to a benchmark, our approach was not specifically optimized for speed. These metrics would likely be considerably smaller if the hyperparameters were selected with speed optimization as the primary objective.

In Table 13, it is evident that our approach demonstrates robust generalization under a lost demand assumption, as indicated by the minimal difference in profits relative to the Just-in-time benchmark between

Table 10 Performance metrics of the Vanilla NN for each instance of setting with one "transshipment" warehouse under backlogged demand assumption. Results consider sampling means and coefficients of variation uniformly between $2.5 - 7.5$ and $0.16 - 0.32$, respectively.

Number of stores	Store lead-time	Store under-age cost	Pairwise correla-tion	Lower bound	Train loss	Dev loss	Test loss	Test gap $\leq$ (%)	Gradient steps to 1% dev gap	Time to 1% dev gap (s)
3	2	4	0.0	3.30	3.30	3.30	3.30	0.07	4000	1494
3	2	4	0.5	3.68	3.69	3.69	3.69	0.15	4960	1851
3	2	9	0.0	4.13	4.14	4.14	4.14	0.08	3840	1427
3	2	9	0.5	4.62	4.62	4.62	4.63	0.23	5920	3906
3	6	4	0.0	4.66	4.67	4.66	4.66	0.11	4800	1827
3	6	4	0.5	4.94	4.95	4.94	4.95	0.14	4960	1890
3	6	9	0.0	5.84	5.85	5.85	5.85	0.14	6880	2641
3	6	9	0.5	6.19	6.20	6.20	6.20	0.16	7360	2814
5	2	4	0.0	2.94	2.94	2.94	2.94	0.06	5440	2042
5	2	4	0.5	3.39	3.39	3.39	3.39	0.06	6080	2281
5	2	9	0.0	3.68	3.68	3.68	3.69	0.08	7360	3479
5	2	9	0.5	4.25	4.25	4.25	4.25	0.06	6720	2519
5	6	4	0.0	4.26	4.27	4.27	4.27	0.16	7360	2885
5	6	4	0.5	4.59	4.59	4.59	4.59	0.09	8800	3452
5	6	9	0.0	5.35	5.35	5.35	5.36	0.17	10080	3994
5	6	9	0.5	5.75	5.76	5.76	5.76	0.06	9760	3852
10	2	4	0.0	2.99	2.99	3.00	3.00	0.15	7840	2774
10	2	4	0.5	3.54	3.54	3.55	3.55	0.13	8480	2993
10	2	9	0.0	3.75	3.75	3.76	3.76	0.19	8960	3136
10	2	9	0.5	4.44	4.45	4.45	4.45	0.15	10400	3671
10	6	4	0.0	4.44	4.45	4.45	4.45	0.24	11360	4283
10	6	4	0.5	4.83	4.84	4.84	4.84	0.20	11520	4340
10	6	9	0.0	5.57	5.58	5.58	5.58	0.28	18080	6903
10	6	9	0.5	6.05	6.08	6.08	6.08	0.47	12960	4929

the train and dev sets. Moreover, our approach consistently achieves a profit within 1% of the Transformed Newsvendor benchmark in less than 90 seconds for each instance. It is noteworthy that, as depicted in Figure 6a in Section 5.6, the Transformed Newsvendor policy attains profits within 5% of those obtained by our approach in instances with unit underage costs of 9 or greater. This observation underscores that our approach achieves commendable performance within a brief time frame.

Conversely, as shown in Table 14, our approach displays noticeable overfitting when considering that unmet demand is backlogged, which intensifies with higher unit underage costs. Remarkably, our approach attains costs on the train set lower than the formidable Returns Newsvendor benchmark for unit underage costs of 9 or higher. Nevertheless, the approach consistently achieves costs within 1% of those incurred by the Transformed Newsvendor policy in less than 30 minutes on average, showcasing rapid learning overall.

Lastly, Figure 12 displays the weekly profit, averaged across scenarios, for each week and policy in the development set. The illustration highlights that our approach consistently achieves higher profits almost every week, emphasizing the reliability of the Vanilla NN.

Table 11 Metrics for the setting of one warehouse and many stores under a lost demand assumption. For each number of stores, training scenarios, and architecture classes, we report the metrics corresponding to the run that minimizes the dev loss among the three runs for each learning rate and layer width. Numbers in bold represent the best test loss obtained for a fixed number of stores. Relative test loss is calculated by dividing the test loss by the loss obtained by the best run for a given number of stores and multiplying by 100. The number of units per layer for the Symmetry-aware NN is fixed, as detailed in Section 4.3.

Number of stores	Training scenarios (#)	Architecture Class	Learning rate	Units per layer (#)	Train loss	Dev loss	Test loss	Relative test loss (%)
3	16	Symmetry-aware	0.0300	-	6.01	5.79	5.79	103.20
3	16	Vanilla	0.0030	128	5.70	5.94	5.95	106.17
3	256	Symmetry-aware	0.0300	-	5.59	5.66	5.67	101.08
3	256	Vanilla	0.0010	128	5.55	5.69	5.68	101.40
3	8192	Symmetry-aware	0.0010	-	5.60	5.61	5.61	100.02
3	8192	Vanilla	0.0010	128	5.61	5.61	**5.61**	100.00
5	16	Symmetry-aware	0.0100	-	5.35	5.45	5.44	103.74
5	16	Vanilla	0.0030	128	5.41	5.79	5.80	110.60
5	256	Symmetry-aware	0.0100	-	5.36	5.29	5.29	100.98
5	256	Vanilla	0.0010	128	5.36	5.38	5.38	102.70
5	8192	Symmetry-aware	0.0100	-	5.25	5.25	5.24	100.05
5	8192	Vanilla	0.0003	512	5.25	5.25	**5.24**	100.00
10	16	Symmetry-aware	0.0100	-	5.80	5.85	5.83	102.03
10	16	Vanilla	0.0003	512	6.01	6.37	6.36	111.37
10	256	Symmetry-aware	0.0100	-	5.71	5.80	5.77	101.06
10	256	Vanilla	0.0003	512	5.82	6.00	5.99	104.79
10	8192	Symmetry-aware	0.0100	-	5.74	5.73	**5.71**	100.00
10	8192	Vanilla	0.0003	512	5.72	5.74	5.72	100.12
20	16	Symmetry-aware	0.0100	-	5.48	6.04	6.01	103.23
20	16	Vanilla	0.0003	512	5.68	6.84	6.79	116.70
20	256	Symmetry-aware	0.0100	-	5.93	5.93	5.91	101.53
20	256	Vanilla	0.0001	512	6.08	6.23	6.21	106.64
20	8192	Symmetry-aware	0.0030	-	5.85	5.84	**5.82**	100.00
20	8192	Vanilla	0.0030	128	5.85	5.87	5.85	100.46
30	16	Symmetry-aware	0.0030	-	5.19	5.66	5.65	101.73
30	16	Vanilla	0.0003	512	5.46	6.46	6.43	115.89
30	256	Symmetry-aware	0.0100	-	5.62	5.63	5.62	101.22
30	256	Vanilla	0.0001	512	5.79	6.07	6.04	108.86
30	8192	Symmetry-aware	0.0030	-	5.56	5.57	**5.55**	100.00
30	8192	Vanilla	0.0030	128	5.58	5.60	5.59	100.64
50	16	Symmetry-aware	0.0030	-	5.48	5.52	5.53	103.14
50	16	Vanilla	0.0030	128	6.09	6.57	6.59	122.97
50	256	Symmetry-aware	0.0030	-	5.31	5.44	5.45	101.70
50	256	Vanilla	0.0003	512	5.61	5.87	5.89	109.95
50	8192	Symmetry-aware	0.0030	-	5.35	5.34	**5.36**	100.00
50	8192	Vanilla	0.0003	512	5.41	5.40	5.42	101.06

Table 12 Metrics for the Symmetry-aware NN for the setting of one warehouse and many stores under a lost demand assumption. For every combination of store count and training sample size, we examine the metrics associated with the run that minimizes the dev loss among three runs for each learning rate. This procedure is repeated 12 times, with the seed varied in each of the 12 iterations, and we present aggregated metrics across all iterations. Relative test loss is calculated by dividing the test loss by the loss obtained by the best run for a given number of stores and multiplying by 100. The number of units per layer for the Symmetry-aware NN is fixed, as detailed in Section 4.3.

Number of stores	Training scenarios (#)	Mean of train loss	Mean of dev loss	Mean of test loss	Mean of relative test loss (%)	Standard dev. of relative test loss (%)
3	1	3.73	6.32	6.31	112.56	6.07
3	2	4.33	6.14	6.15	109.62	4.21
3	4	4.78	5.95	5.95	106.20	1.61
3	8	5.25	5.85	5.85	104.29	0.80
5	1	3.16	5.75	5.76	109.77	3.91
5	2	3.65	5.61	5.61	106.98	2.30
5	4	4.22	5.52	5.52	105.33	1.89
5	8	4.52	5.45	5.45	104.01	1.47
10	1	3.71	6.28	6.28	109.88	2.47
10	2	4.28	6.14	6.13	107.38	1.83
10	4	4.66	6.00	6.00	104.97	2.00
10	8	5.13	5.92	5.91	103.48	1.06
20	1	4.01	6.54	6.54	112.44	8.70
20	2	4.46	6.27	6.26	107.59	2.01
20	4	4.86	6.10	6.10	104.73	1.39
20	8	5.27	6.03	6.02	103.52	0.88
30	1	3.86	6.02	6.02	108.39	2.43
30	2	4.27	5.93	5.93	106.88	2.78
30	4	4.77	5.84	5.84	105.26	2.27
30	8	4.97	5.74	5.74	103.45	0.88
50	1	4.28	5.94	5.93	110.63	3.94
50	2	4.69	5.72	5.71	106.51	2.03
50	4	4.76	5.64	5.63	105.10	2.05
50	8	4.95	5.57	5.57	103.89	0.97

B.10. Quantile forecaster

In this subsection, we provide detailed information regarding the offline training and performance assessment of the quantile forecaster employed by the generalized newsvendor policies in Section 5.4. The quantile forecaster is designed to estimate the distribution of the sum of m demand terms (considering various values of m simultaneously), given the sequence of the previous $s_1 \in \mathbb{N}$ demands and the number of days until the next occurrence of Christmas, denoted as $d_{\text{Christmas}} \in \mathbb{N}$. Specifically, given a tuple $(\xi_1, \ldots, \xi_{s_1}, d_{\text{Christmas}})$, the quantile forecaster predicts the value of the τ-quantile of the sum of the next m demands concurrently

Table 13 Performance metrics of the Vanilla NN for each instance of the setting with one store considering sales data from the Corporación Favorita Grocery Sales Forecasting competition, under a lost demand assumption. Evaluation on dev set was performed every 5 epochs (equivalent to 20 gradient steps), so time and gradient steps represent upper bounds.

Average unit underage cost	Train profit	Dev profit	Train profit relative to Just-in-time (%)	Dev profit relative to Just-in-time (%)	Time to 1% gap from Transformed Newsvendor (s)	Gradient steps to 1% gap from Transformed Newsvendor
2	69.77	65.60	65.2	66.0	28	100
3	112.92	105.02	70.4	70.5	33	120
4	157.66	146.35	73.7	73.7	34	120
6	250.31	231.23	78.0	77.6	35	120
9	394.10	363.37	81.9	81.3	44	160
13	590.35	543.64	84.9	84.2	82	280
19	890.83	820.42	87.7	86.9	60	220

Table 14 Performance metrics of the Vanilla NN for each instance of the setting with one store considering sales data from the Corporación Favorita Grocery Sales Forecasting competition, under a backlogged demand assumption. Evaluation on dev set was performed every 5 epochs (equivalent to 20 gradient steps), so time and gradient steps represent upper bounds.

Average unit underage cost	Train loss	Dev loss	Train cost relative to Returns Newsvendor (%)	Dev cost relative to Returns Newsvendor (%)	Time to 1% gap from Transformed Newsvendor (s)	Steps to 1% gap from Transformed Newsvendor
2	80.93	78.80	101.6	102.1	898	3160
3	97.88	96.50	100.8	101.6	747	2660
4	112.44	111.74	101.4	102.1	1195	4220
6	133.13	134.97	100.2	101.8	1697	5960
9	157.00	162.84	99.4	101.9	2374	8340
13	181.67	193.18	98.8	102.4	2727	9620
19	206.00	228.14	96.2	102.6	2010	6900

for each τ and m within specified sets $\mathcal{Q}$ and $\mathcal{M}$, respectively. This is, for one input tuple, it produces $|\mathcal{Q}||\mathcal{M}|$ terms.

We consider a dataset comprising N pairs in the form of (x_i, y_i), where $x_i \in \mathbb{R}^{s_1+1}$ represents a feature vector, and $y_i \in \mathbb{R}^{|\mathcal{M}|}$ is a target vector. Given the prediction $\hat{y}_i \in \mathbb{R}^{|\mathcal{Q}||\mathcal{M}|}$ for feature vector x_i, we define the multi-horizon loss for the τ-quantile as

$$\ell_\tau(y_i, \hat{y}_i) = \sum_{m \in \mathcal{M}} \max\{\tau(y_{im} - \hat{y}_{im\tau}), (1-\tau)(\hat{y}_{im\tau} - y_{im})\}, \tag{23}$$

and the multi-horizon multi-quantile loss as

$$\ell(y_i, \hat{y}_i) = \sum_{\tau \in \mathcal{Q}} \ell_\tau(y_i, \hat{y}_i). \tag{24}$$

The objective is to minimize the sample average of $\ell(y_i, \hat{y}_i)$, given by $1/N \sum_{i=1}^{N} \ell(y_i, \hat{y}_i)$.

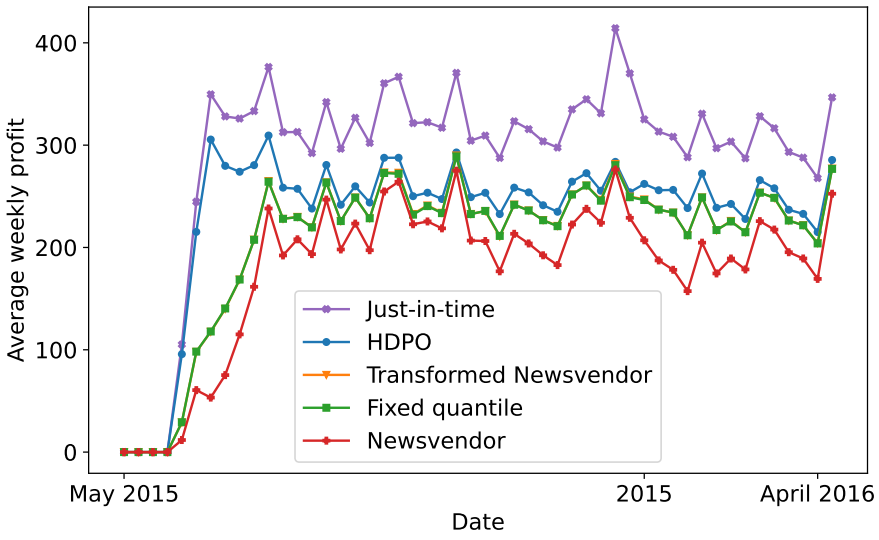


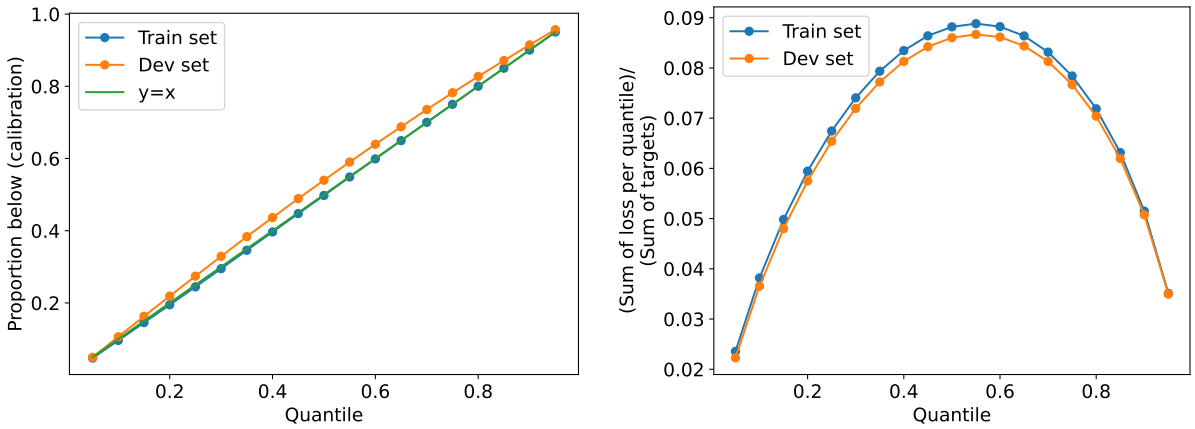
Figure 12 Average weekly profit on the dev set obtained by each policy in the setting with one store and realistic demand, for an average unit underage cost of 6 and under a lost demand assumption. Orange and green lines significantly overlap.

We implemented the quantile forecaster using PyTorch, considering quantiles ranging from 0.05 to 0.95 in steps of 0.05. Given that the numerical experiments in Section 5.6 consider lead times of 4, 5, and 6, we set $\mathcal{M} = \{5, 6, 7\}$ (recall that generalized newsvendor policies take into account the distribution of the sum of $L+1$ demands). The train set and dev set sizes were approximately 4 million and 1.5 million samples, respectively. These sets were generated by extracting subsequences of demands with a length of 16 from the datasets outlined in Section 5.6.

The architecture of the implemented Multilayer Perceptron (MLP) includes 2 hidden layers, each with 128 neurons. We considered a batch size of $1,048,576$ samples and a learning rate of 0.01. In practice, input features for all samples within a batch are fed in a compact matrix form, and the model outputs a matrix of predictions with dimensions $(n_{\text{batch}}, |\mathcal{Q}|, |\mathcal{M}|)$, where n_{batch} is the batch size. This design allows us to leverage the parallel computing capabilities of GPUs and take advantage of an efficient implementation in PyTorch.

We are able to validate the efficacy of the quantile forecaster by observing remarkable results achieved by the non-admissible Return Newsvendor benchmark in the backlogged demand setting (refer to Figure 7 in Section 5.7), as well as the satisfactory performance of the Transformed Newsvendor policy in the lost demand setting with high average unit underage costs (refer to Figure 6a in Section 5.6). To further assess the performance of our quantile forecaster, we conducted two separate analyses.

First, we evaluated the model's calibration, which measures the agreement between the predicted values for each quantile and the empirical probability of a target lying below each of them. Ideally, for each $\tau \in \mathcal{Q}$, we would observe an empirical proportion of τ of the samples lying below the predicted τ-quantile. Therefore, in Figure 13a, we plotted each $\tau \in \mathcal{Q}$ against the proportion of targets lying below the predicted τ-quantile, calculated as $\sum_{i\in[N]} \sum_{m\in\mathcal{M}} \mathbb{1}\left(y_{im} \leq \hat{y}_{im\tau}\right) / (N|\mathcal{M}|)$. The figure demonstrates that our forecaster is nearly



(a) Calibration for each quantile, calculated as the proportion of targets that lie below the quantile predicted by the forecaster.

(b) Sum of multi-horizon quantile loss (see Equation 23) divided by the sum of the targets, for each quantile.

Figure 13 Performance indicators for the quantile forecaster.

perfectly calibrated in the train set. However, in the dev set, there is a slight overestimation, resulting in larger-than-ideal proportions of observed targets lying below each predicted quantile value, although the calibration remains generally good.

Next, our objective is to provide an estimation of the magnitude of prediction errors relative to the targets. Illustrated in Figure 13b, we depict, for each $\tau \in \mathcal{Q}$, the cumulative multi-horizon loss (see Eq. 23) divided by the sum of targets, given by $\sum_{i\in[N]} \ell_\tau(y_i, \hat{y}_i) / \sum_{i\in[N]} \sum_{m\in\mathcal{M}} y_{im}$. To enhance comprehension, consider that when $\tau = 0.5$, the multi-horizon quantile loss is equivalent to one half of the sum of mean absolute errors (MAEs) across time horizons $m \in \mathcal{M}$. The sum of MAEs is therefore less than 18% of the sum of the targets, indicating a relatively low value. As the analyzed ratio is nearly maximized for $\tau = 0.5$, this suggests that the quantile forecaster adeptly predicts the conditional distribution of cumulative demands.

While we recognize the importance of comparing our forecaster with other methodologies to evaluate its performance, the three evaluations conducted in this section collectively indicate its effectiveness.

C. Proof of Theorem 1

In this section, we prove Theorem 1. To begin, we review our existing notation and introduce some supplementary notation that will aid us in defining a *relaxed* version of our problem setting. We note that, given that stores have zero lead time, the form of transition and cost functions will be somewhat different to that presented in Section 2.3.

State and action spaces. The state of the system is given by $S_t = I_t = (I_t^0, \ldots, I_t^K)$. We emphasize that I_t^k represents the inventory on-hand for location $k \in [K]_0$ **before** orders are placed. As store lead times are zero, there are no outstanding orders. Further, for the warehouse, we can update inventory by replacing $q^k_{t-L^0+1}$ by a_t^0 in (5), avoiding the need for keeping track of Q_t^0. We further let $Z_t = \sum_{k\in[K]_0} I_t^k$ track the system-wide inventory in the system. The action space at state S_t is given by $\mathcal{A}(I_t) = \{a_t \in \mathbb{R}_+^{K+1} \mid \sum_{k\in[K]} a_t^k \leq I_t^0\}$.

Random variables. As described in Section 4.2, demand takes the form $\xi_t^k = B_t U_t^k$, where $U_t^k \sim \text{Uniform}(\underline{u}^k, \overline{u}^k)$ and B_t takes a *high* value $\gamma^{\text{H}} > 0$ with probability q and a *low* value $\gamma^{\text{L}} \in (0, \gamma^{\text{H}})$. We denote the mean $\mathbb{E}[U_t^k]$ of each uniform as $\mu^k = (\overline{u}^k + \underline{u}^k)/2$, and denote the aggregate demand by $\Xi_t = \sum_{k\in[K]} \xi_t^k$. Further, let $\hat{U}_t = \sum_{k\in[K]} U_t^k$ be the sum of the K uniforms at time t, $\hat{\mu} = \sum_{k\in[K]} \mu^k$ its mean, and $\tilde{U}_t^k = \sum_{k\in[K]}(U_t^k - \mu^k)$ be the sum of zero-mean uniforms. Note that we can represent aggregate demand as $\Xi_t = B_t \hat{U}_t$.

We define $D^{\text{H}} = \mathbb{E}[\Xi_t | B_t = \gamma^{\text{H}}] = \gamma^{\text{H}}\hat{\mu}$ and $D^{\text{L}} = \mathbb{E}[\Xi_t | B_t = \gamma^{\text{L}}] = \gamma^{\text{L}}\hat{\mu}$ as the expectation of aggregate demand conditional on B_t taking *high* and *low* values, respectively, and let D_t be a random variable such that $D_t = D^H$ if $B_t = \gamma^H$ and $D_t = D^L$ otherwise. Finally, let $W_t = \mathbb{1}(B_{t-1} = \gamma^{\text{L}})$ be an indicator that the demand was low in the previous period, and $\tilde{W}_t(Z_t)$ be an estimator of the previous quantity, which will be defined shortly. For brevity, we define the shorthand notation $\tilde{W}_t \equiv \tilde{W}_t(Z_t)$.

Transition functions

Given that stores have no lead time and that the warehouse lead time is equal to 1, inventory evolves as

$$I_{t+1}^k = I_t^k + a_t^k - \xi_t^k \qquad k \in [K] \tag{25}$$

$$I_{t+1}^0 = I_t^0 + a_t^0 - \textstyle\sum_{k\in[K]} a_t^k. \tag{26}$$

Even though (25) would be identical for store lead times of one period, there is a distinction in the cost function between lead times of zero and one, as shown later on.

Meanwhile, the system-wide level of inventory evolves as

$$Z_{t+1} = Z_t + a_t^0 - \Xi_t\,, \tag{27}$$

and note that Z_t does not depend on $a_t^1, \ldots, a_t^K$.

Cost functions. *The case when store lead times are zero is an edge-case where the cost incurred at stores is different from what was written in the generic problem formulation.* In particular, the cost at store k in some period is given by $c^k(I_t^k, a_t^k, \xi_t^k) = p^k(\xi_t^k - (I_t^k + a_t^k))^+ + h^k((I_t^k + a_t^k) - \xi_t^k)^+$, where the first and second terms correspond to underage and overage costs, respectively. Note that, as store lead times are zero, the inventory a_t^k allocated in the current period arrives to the store *before* demand occurs, so the store incurs

the same overage/underage costs it would if lead time were nonzero and it had inventory $I_t^k + a_t^k$ on hand at the start of the period.

The cost at the warehouse is solely composed of holding costs and is given by $c^0(I_t^0, a_t) = h^0\left(I_t^0 - \sum_{k\in[K]} a_t^k\right)$. Total costs in period t are finally $c(I_t, a_t, \xi_t) = c^0(I_t^0, a_t) + \sum_{k\in[K]} c^k(I_t^k, a_t^k, \xi_t^k)$. We will define costs in a more convenient way by defining

$$v(Z, y^1, \ldots, y^K) = h^0 Z + \sum_{k\in[K]} \mathbb{E}_{\xi_t^k}\left[\left(p^k(\xi_t^k - y^k)^+ + h^k(y^k - \xi_t^k)^+ - h^0 y^k\right)\right] \tag{28}$$

and noting that $c(I_t, a_t, \xi_t) = v\left(\sum_{k=0}^K I_t^k, I_t^1 + a_t^1, \ldots, I_t^K + a_t^K\right)$.

Cost-to-go.

We define $J_t^\pi(I_t) = \mathbb{E}^\pi\left[\sum_{s=t}^T c(I_s, \pi_s(I_s), \xi_s)\middle| I_t\right]$ as the expected cost-to-go under policy $\pi \in \Pi$ from period t and initial state I_t, and $J_t(I_t)$ as the optimal cost-to-go, that is, $J_t(I_t) = \inf_{\pi\in\Pi} J_t^\pi(I_t)$.

Recall that the setting in Example 1 makes two assumptions.

Assumption 1. *$\gamma^{\mathrm{L}}\underline{u}^k \geq \gamma^{\mathrm{H}}\overline{u}^k - \gamma^{\mathrm{L}}\underline{u}^k$ for every k.*

Assumption 2. *$q\underline{p} \geq (1-q)h^0$.*

These assumptions will facilitate our analysis. Assumption 1 ensures that, if store k starts with an inventory no larger than $\gamma^{\mathrm{H}}\overline{u}^k$, the remaining inventory after demand is realized is below $\gamma^{\mathrm{L}}\underline{u}^k$. This condition can be slightly relaxed relative to the form stated, but we avoid presenting the most general form in the interest of simplicity. Assumption 2 ensures that it is worth acquiring an incremental unit of inventory at the warehouse if that is sure to prevent a lost sale in a period of high demand.

C.1. Proof overview

We now present a detailed version of Theorem 1. Let

$$\kappa \equiv \max_{k\in[K]} \max(\overline{u}^k, 1/\overline{u}^k) \quad \Rightarrow \quad \overline{u}^k \in [1/\kappa, \kappa]\, \forall k \in [K]$$

and recall the definition of the Proportional Allocation feasibility enforcement functions as $g_1(I^0, b^1, \ldots, b^K) = \left[[b^k]^+ \cdot \min\left\{1, \frac{I^0}{\sum_{j\in[K]}[b^j]^+}\right\}\right]_{k\in[K]}$. Moreover, denote $\mathcal{R}^k = (\underline{u}^k, \overline{u}^k, p^k, h^k)$ as the store-specific primitives of location k, and let $\mathcal{H} \subset \mathbb{R}^4$ denote the set of possible values for store-specific primitives according to the assumptions outlined in Example 1.

Theorem 2. *Under the Proportional Allocation feasibility enforcement function, there exists a constant $C = C(\overline{p}, \overline{h}, \underline{p}, \underline{h}, q, \kappa, \gamma^{\mathrm{L}}, \gamma^{\mathrm{H}}) < \infty$, $y^0 \in \mathbb{R}$, continuous functions $\underline{G}, \overline{G} : \mathcal{H} \to \mathbb{R}_+$ and a stationary symmetry-aware policy $\tilde{\pi}$ with*

$$\begin{aligned}
\tilde{\pi}_{\text{context}}(S_t) &= Z_t \\
\tilde{\pi}^0\left(I_t^0, Z_t\right) &= y^0 - Z_t \\
\tilde{\pi}^1\left(I_t^k, \mathcal{R}^k, Z_t\right) &= \max(0, \left[\underline{G}(\mathcal{R}^k) + \tilde{W}_t(Z_t)(\overline{G}(\mathcal{R}^k) - \underline{G}(\mathcal{R}^k))\right] - I_t^k) \quad k \in [K],
\end{aligned} \tag{29}$$

where $\tilde{W}_t(Z) = \mathbb{1}\left\{y^0 - Z_t < (D^{\mathrm{H}} + D^{\mathrm{L}})/2\right\}$, such that, its expected cumulative cost $J_1^{\tilde{\pi}}(I_1)$ satisfies

$$\frac{J_1^{\tilde{\pi}}(I_1)}{J_1(I_1)} \leq 1 + \frac{C}{\sqrt{K}}\,.$$

Throughout this section, we utilize the notation $\underline{y}^k = \underline{G}(\mathcal{R}^k)$ and $\overline{y}^k = \overline{G}(\mathcal{R}^k)$. The construction of the functions $\underline{G}$ and $\overline{G}$ is demonstrated within the proof of Lemma 4 in Appendix C.9.

For this asymptotically optimal policy $\tilde{\pi}$, the context mapping signals the current system-wide level of inventory Z_t to the stores. Then, the warehouse follows an echelon-stock policy (see Equations 21 and 22 in Appendix A.5), this is, it observes Z_t and places an order to "raise" the system-wide level of inventory up-to the target level y^0. Meanwhile, stores use the context signal to compute the estimator $\tilde{W}_t(Z)$ of whether the demand in the previous period was low and, consequently, propose a tentative allocation to raise their local inventory to one base-stock level among $\overline{y}^k$ and $\underline{y}^k$. The intuition behind the form of $\tilde{W}_t(Z)$ is the following. Given that under $\tilde{\pi}$ the system-wide inventory is raised to y^0 before demand is realized, the quantity $(y^0 - Z_t)$ is equal to cumulative demand in the previous period. Then, if the previous demand is below the midpoint $(D^{\mathrm{H}} + D^{\mathrm{L}})/2$ of the conditional expectations for *high* and *low* demand periods, the store estimates that demand in the previous period was low.

Our proof of Theorem 2 consists of two main parts, which we now summarize:

1. **Converse bound**: We start by constructing a relaxed setting in two steps. First, we allow inventory to "flow back" from stores to the warehouse, and show that the cost-to-go only depends on the current state through the system-wide level of inventory. We then analyze a setting, which we refer to as the *fully relaxed setting*, in which the system-wide level of inventory evolves as if $\hat{U}_t = \hat{\mu}$, but in which the costs incurred at stores do account for the randomness coming from U_t^k, and show that the expected cost in this relaxed system is a lower bound on that of the original one.

2. **Achievability**: Our strategy for establishing achievability begins with a detour in which we consider the fully relaxed setting and we characterize the optimal policy for that system: We will demonstrate the optimality of a stationary policy for the fully relaxed setting, where the warehouse follows an echelon stock policy and the stores follow a base-stock policy with one of two base-stock levels. Additionally, we will establish that the inventory at the stores prior to demand realization is kept below both base-stock levels. This indicates that the optimal policy for the fully relaxed setting does not involve transferring inventory from the stores to other stores or the warehouse. We will then derive an asymptotically optimal policy for the original system by following the warehouse's echelon stock level and store's base-stock levels from the optimal policy in the relaxed setting, and employing a Proportional Allocation feasibility enforcement function (Eq. 8 in Section 2.4). We will use a simple upper bound on the the per-unit cost of "scarcity" (*i.e.*, units of unsatisfied store's orders) and show that scarcity is upper bounded by the deviation of the sum of Uniforms $\hat{U}_t$ from its mean. We will conclude that per-period scarcity costs scale as $\sqrt{K}$, implying a relative excess cost of order $1/\sqrt{K}$ as compared to that incurred in the fully relaxed setting.

C.2. Converse bound

We begin by presenting the structure of the optimal cost-to-go in the original setting, and then introduce a series of relaxations that will allow us to obtain a lower bound on costs.

In Appendix C.4, we show the following expression for the cost-to-go.

LEMMA 1. *The cost-to-go (for the original setting) in period t takes the form*

$$J_t(I_t) = \inf_{a_t \in \mathcal{A}(I_t)} \left\{ \mathbb{E}_{\xi_t} \left[J_{t+1}(f(I_t, a_t, \xi_t)) \right] + v \left(\sum_{k \in [K]_0} I_t^k, I_t^1 + a_t^1, \ldots, I_t^K + a_t^K \right) \right\}, \tag{30}$$

where the immediate cost $v(\cdot)$ was defined in (28).

We construct a relaxed system, which we denote as the *partially relaxed system*, following a procedure similar to that of Federgruen and Zipkin (1984a). We will allow $a^1, \ldots, a^K$ to be negative, thus allowing inventory to flow from stores to the warehouse and be re-allocated to other stores. The action space for the partially relaxed system is given by $\hat{\mathcal{A}}(I_t) = \{a_t \in \mathbb{R}^+ \times \mathbb{R}^K \mid \sum_{k \in [K]} a_t^k \leq I_t^0\}$. As formalized in Appendix C.5, since we allow inventory to flow back from stores, the cost-to-go for the partially relaxed system $\breve{J}_{t+1}(I_{t+1})$ depends on I_{t+1} solely through the sum of its components $Z_{t+1} = \sum_{k \in [K]_0} I_{t+1}^k = Z_t + a_t^0 - \Xi_t$ (from (27)). This leads to the following Bellman equation for the partially relaxed cost-to-go

$$\breve{J}_t(I_t) = \inf_{a_t \in \hat{\mathcal{A}}(I_t)} \left\{ \mathbb{E}_{\xi_t} \left[\breve{J}_{t+1}(Z_t + a_t^0 - \Xi_t) \right] + v \left(\sum_{k \in [K]_0} I_t^k, I_t^1 + a_t^1, \ldots, I_t^K + a_t^K \right) \right\}, \tag{31}$$

$$= \inf_{a_t^0 \in \mathbb{R}^+} \mathbb{E}_{\xi_t} \left[\breve{J}_{t+1}(Z_t + a_t^0 - \Xi_t) \right] + \inf_{\sum_{k \in [K]} a_t^k \leq I_t^0} v \left(\sum_{k \in [K]_0} I_t^k, I_t^1 + a_t^1, \ldots, I_t^K + a_t^K \right). \tag{32}$$

Here, (32) captures that the minimization over a_t is separable across the warehouse action a_t^0 versus the store allocations $(a_t^k)_{k \in [K]}$, because the cost-to-go term in (31) does not depend on store allocations, and the immediate cost term $v(\cdot)$ in (31) does not depend on the warehouse order a_t^0. (We abuse notation in writing $\breve{J}_t(I_t) = \breve{J}_t(Z_t)$.)

Letting $y_t^k = I_t^k + a_t^k$ for $k \in [K]$, the second minimization in (32) can be rewritten as a minimization over $y_t = (y_t^k)_{k \in [K]}$ as follows:

$$\begin{aligned} \hat{R}(Z_t) \equiv \inf_{y_t \in \mathbb{R}^K} & \; v\left(Z_t, y_t^1, \ldots, y_t^K\right) \\ \text{s.t.} & \sum_{k \in [K]} y_t^k \leq Z_t \,. \end{aligned} \tag{33}$$

As a result, the Bellman equation (32) for the partially relaxed system can be written as

$$\breve{J}_t(Z_t) = \hat{R}(Z_t) + \min_{a_t^0 \in \mathbb{R}^+} \mathbb{E}_{\Xi_t} \left[\breve{J}_{t+1}(Z_t + a_t^0 - \Xi_t) \right]. \tag{34}$$

We introduce an additional relaxation referred to as the *fully relaxed system*, where the state evolves as if $\hat{U}_t = \hat{\mu}$, while maintaining the same cost incurred in each period as in the partially relaxed system. In the fully relaxed setting, we denote the system-wide inventory level by $\hat{Z}_t$, and define that it evolves according to $\hat{Z}_{t+1} = \hat{Z}_t + a_t^0 - D_t$. It is important to note that this quantity is, in general, different from the system-wide inventory levels in the original and partially relaxed systems, even when fixing a policy and demand trace. However, we will demonstrate later that the optimal expected cost in all three systems are close to each other. Let $\hat{J}_t(\hat{Z}_t)$ represent the cost-to-go for the fully relaxed system, starting from period t with a system-wide inventory level of $\hat{Z}_t$. The Bellman equation for the fully relaxed system is

$$\hat{J}_t(\hat{Z}_t) = \hat{R}(\hat{Z}_t) + \min_{a_t^0 \in \mathbb{R}^+} \mathbb{E}_{D_t} \left[\hat{J}_{t+1}(\hat{Z}_t + a_t^0 - D_t) \right]. \tag{35}$$

We consider $\hat{Z}_1 = Z_1$ as the initial state for the fully relaxed system.

In Appendix C.6, we show the following key convexity property.

LEMMA 2. *$\hat{R}(\cdot)$ is a convex function. For all $t = 1, 2, \ldots, T$, $\breve{J}_t(\cdot)$ and $\hat{J}_t(\cdot)$ are convex functions.*

We use this convexity to prove that our relaxations indeed lead to a lower bound on costs for the original setting.

PROPOSITION 1. *For any starting inventory state $I_1 \in (\mathbb{R}^+)^{K+1}$ and $Z_1 = \sum_{k \in [K]_0} I_1^k$, we have*

$$\hat{J}_1(Z_1) \leq \breve{J}_1(Z_1) \leq J_1(I_1)\,,$$

i.e., *the expected cost in the fully relaxed system is weakly smaller than the expected cost in the partially relaxed system, which, in turn, is weakly smaller than the expected cost in the original system.*

We show Proposition 1 in Appendix C.7. The first inequality is obtained by using the convexity of $\hat{R}$ and applying Jensen's inequality on $\hat{U}_t$. The second inequality is immediate since $\breve{J}$ is obtained by relaxing the original setting.

C.3. Achievability

We begin by unveiling the structure of the optimal policy for the fully relaxed setting which, in turn, will allow us to construct an asymptotically optimal policy for the original setting.

Let $y_t^k = I_t^k + a_t^k$, for every $k \in [K]$, be the inventory level at store k before demand is realized.

LEMMA 3. *There exists an optimal policy (which is stationary) for the fully relaxed setting such that the warehouse follows an echelon-stock policy. Furthermore, the optimal echelon-stock level $\hat{S}$ satisfies*

$$\hat{S} \in \arg\min_{S \in \mathbb{R}} \left[q\hat{R}(S - D^{\mathrm{H}}) + (1-q)\hat{R}(S - D^{\mathrm{L}}) \right], \tag{36}$$

where $\hat{R}(\cdot)$ is as defined in (33).

We prove Lemma 3 in Appendix C.8. The policy for the warehouse relies on the fact that (35) takes the form of a single-location inventory problem with inventory level $\hat{Z}_t$ and convex cost function $\hat{R}(\hat{Z}_t)$. Following Federgruen and Zipkin (1984a), an echelon-stock policy must be optimal. Since there are no procurement costs involved, we demonstrate that the echelon-stock level in each period should minimize the expected cost incurred in the subsequent period, as indicated in (36). Consequently, the echelon-stock level remains constant across all periods.

Starting from $t = 2$ onwards, it is important to observe that $\hat{Z}_t$ can only assume one of two distinct values under the optimal policy: either $\hat{S} - D^{\mathrm{H}}$ or $\hat{S} - D^{\mathrm{L}}$. Since store inventory levels solve (33) in each period, at each store the inventory level before the actual demand occurs will take one of two values (corresponding to the two aforementioned solutions of (33)). These inventory levels will be interpreted as store base-stock levels in subsequent analysis.

LEMMA 4. *Suppose Assumption 2 holds, and let $\underline{y}$ and $\overline{y}$ be optimal solutions for problem* (33) *with total inventory $Z_t = \hat{S} - D_t^{\mathrm{H}}$ and $\hat{S} - D_t^{\mathrm{L}}$, respectively. Then, $\gamma^{\mathrm{H}}\overline{u}^k \geq \overline{y}^k \geq \underline{y}^k \geq \gamma^{\mathrm{L}}\underline{u}^k$ for each $k \in [K]$. In particular, an optimal policy for the fully relaxed system does not move inventory from a store to other stores or to the warehouse. Additionally, there exist continuous functions $\underline{G}, \overline{G} : \mathcal{H} \to \mathbb{R}_+$ (where $\mathcal{H}$ was defined in Appendix C.1), such that $\underline{y}^k = \underline{G}(\mathcal{R}^k)$ and $\overline{y}^k = \overline{G}(\mathcal{R}^k)$ for each $k \in [K]$.*

We prove Lemma 4 in Appendix C.9. The argument leading to the assertion that an optimal policy for the fully relaxed system does not move inventory from a store to other stores or to the warehouse is as follows: Recall that Assumption 1 ensures that if store k starts with an inventory no larger than $\gamma^{\mathrm{H}}\overline{u}^k$, the remaining inventory after demand is realized is below $\gamma^{\mathrm{L}}\underline{u}^k$. Hence, Lemma 4 allows us to conclude that the base-stock levels $\overline{y}^k, \underline{y}^k$ for each store are always above the store's inventory level after demand occurs under our assumptions. Therefore, an optimal policy for the fully relaxed system does not move inventory from a store to other stores or to the warehouse. Moreover, in the Lemma we show how to compute $\underline{y}^k$ and $\overline{y}^k$ from the optimality conditions of problem (33) with initial inventories $\hat{S} - D_t^{\mathrm{H}}$ and $\hat{S} - D_t^{\mathrm{L}}$, respectively, which permits the construction of the aforementioned continuous functions $\underline{G}$ and $\overline{G}$.

We derive a feasible policy $\tilde{\pi}$ to the original problem by following the structure in (29), and setting $y^0 = \hat{S}$ for $\hat{S}$ as defined in Lemma 3, and $\overline{y}^k$ and $\underline{y}^k$ for $k \in [K]$ as per the definitions in Lemma 4. That is, the warehouse will follow an echelon-stock policy with level $\hat{S}$ and stores will follow a base-stock policy, with level $\overline{y}^k$ (or $\underline{y}^k$) whenever the demand in the previous period is estimated as being low (high). We derive feasible actions by using the Proportional Allocation feasibility enforcement function $g_1(I^0, b^1, \ldots, b^K) = [b^k]^+ \cdot \min\left\{1, \frac{I^0}{\sum_{j\in[K]}[b^j]^+}\right\}$. We can therefore represent tentative allocations and actions under $\tilde{\pi}$ by

$$
\begin{aligned}
a_t^{\tilde{\pi}0} &= \hat{S} - Z_t \\
b_t^{\tilde{\pi}k} &= \max(0, \left[\underline{y}^k + \tilde{W}_t(Z_t)(\overline{y}^k - \underline{y}^k)\right] - I_t^k) && k \in [K] \\
a_t^{\tilde{\pi}k} &= \left[g_1(I_t^0, b_t^{\tilde{\pi}1}, \ldots, b_t^{\tilde{\pi}K})\right]_k && k \in [K].
\end{aligned}
$$

We provide a brief explanation regarding the role of $\tilde{W}_t$ and its definition $\tilde{W}_t(Z) \equiv \mathbb{1}\left\{y^0 - Z_t < (D^{\mathrm{H}} + D^{\mathrm{L}})/2\right\}$. Recall our assumption that only the observed demand is available to us, without knowledge of the underlying high or low state. Therefore, an estimate is required based on the current system-wide inventory levels. We compare the realized total demand in the previous period $y^0 - Z_t$ with the midpoint between D^{H} and D^{L} to estimate whether the demand in the previous period was high or low B_{t-1}. As the number of stores K increases, the demand fluctuations caused by the sum of uniform random variables $\hat{U}_t$ scale by approximately $\sqrt{K}$, while the difference between the midpoint and both D^{H} and D^{L} scales linearly with K. Consequently, as K grows larger, the probability of incorrect estimation $\tilde{W}_t \neq B_{t-1}$ decreases exponentially, as demonstrated in Appendix C.11.

Recall that $\tilde{U}_t^K = \sum_{k\in[K]}(U_t^k - \mu^k)$ represents the centered sum of the K uniforms at time t, *i.e.*, the sum minus its mean. Let $\alpha = |\hat{R}(\hat{S} - D^{\mathrm{L}}) - \hat{R}(\hat{S} - D^{\mathrm{H}})| + \max\{\overline{p}, \overline{h}\}(|D^{\mathrm{H}} - D^{\mathrm{L}}|)$.

Proposition 2. *The total expected cost in the original system under the policy $\tilde{\pi}$ is bounded above as*

$$
J_1^{\tilde{\pi}}(I_1) \leq \hat{J}_1(Z_1) + T\left[\max\{\overline{p}, \overline{h}\}\gamma^{\mathrm{H}}\mathbb{E}\left[\left|\tilde{U}_1^K\right|\right] + \alpha\mathbb{P}\left(\left|\tilde{U}_1^K\right| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right)\right]. \tag{37}
$$

(Recall that $\hat{J}_1(Z_1)$ is the total expected cost in the fully relaxed system, with the same starting inventory.)

We prove Proposition 2 in Appendix C.10. The proposition bounds the additional expected cost per period in the real system relative to that under the fully relaxed system. The $T\max\{\overline{p}, \overline{h}\}\gamma^{\mathrm{H}}\mathbb{E}[|\tilde{U}_1^K|]$ term bounds the impact of the aggregate demand deviating from D^{L} or D^{H}, and the $T\alpha\mathbb{P}(|\tilde{U}_1^K| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}})$ term bounds

the impact of occasionally getting W_t wrong $W_t \neq B_{t-1}$. In short, the additional expected cost per period in the real system is at most proportional to deviation of the sum of uniforms from its mean. We show the latter quantity to be "small", specifically we show that it is $O(\sqrt{K})$ by the central limit theorem, in Appendix C.11. Plugging this bound into Proposition 2, together with the converse bound in Proposition 1 and 2 leads us to a proof of Theorem 2 in Appendix C.12.

C.4. Bellman Equation for Optimal Cost-to-go

LEMMA 5. *The Bellman equation for the original system can be written as*

$$J_t(I_t) = \inf_{a_t \in \mathcal{A}(I_t)} \left\{ \mathbb{E}_{\xi_t} \left[J_{t+1}(f(I_t, a_t, \xi_t)) \right] + v\left(\sum_{k \in [K]_0} I_t^k, I_t^1 + a_t^1, \ldots, I_t^K + a_t^K \right) \right\}. \tag{38}$$

Recall that $J_t(I_t)$ represents the optimal cost-to-go for the original system from period $t \in [T]$ when starting at state I_t. The Bellman equation is

$$\begin{aligned}
J_t(I_t) &\overset{(i)}{=} \inf_{a_t \in \mathcal{A}(I_t)} \left\{ \mathbb{E}_{\xi_t} \left[J_{t+1}(f(I_t, a_t, \xi_t)) + \sum_{k \in [K]} \left(p^k(\xi_t^k - (I_t^k + a_t^k))^+ + h^k(I_t^k + a_t^k - \xi_t^k)^+ \right) \right. \right. \\
&\qquad \left. \left. + h^0(I_t^0 - \sum_{k \in [K]} a_t^k) \right] \right\} \\
&\overset{(ii)}{=} \inf_{a_t \in \mathcal{A}(I_t)} \left\{ \mathbb{E}_{\xi_t} \left[J_{t+1}(f(I_t, a_t, \xi_t)) + h^0 I_t^0 \right. \right. \\
&\qquad \left. \left. + \sum_{k \in [K]} \left(p^k(\xi_t^k - (I_t^k + a_t^k))^+ + h^k(I_t^k + a_t^k - \xi_t^k)^+ - h^0 a_t^k \right) \right] \right\} \\
&\overset{(iii)}{=} \inf_{a_t \in \mathcal{A}(I_t)} \left\{ \mathbb{E}_{\xi_t} \left[J_{t+1}(f(I_t, a_t, \xi_t)) + h^0 \sum_{k \in [K]_0} I_t^k \right. \right. \\
&\qquad \left. \left. + \sum_{k \in [K]} \left(p^k(\xi_t^k - (I_t^k + a_t^k))^+ + h^k(I_t^k + a_t^k - \xi_t^k)^+ - h^0(I_t^k + a_t^k) \right) \right] \right\} \\
&\overset{(iv)}{=} \inf_{a_t \in \mathcal{A}(I_t)} \left\{ \mathbb{E}_{\xi_t} \left[J_{t+1}(f(I_t, a_t, \xi_t)) \right] + v\left(\sum_{k \in [K]_0} I_t^k, I_t^1 + a_t^1, \ldots, I_t^K + a_t^K \right) \right\},
\end{aligned}$$

where (ii) is obtained by reorganizing $h^0(I_t^0 - \sum_{k \in [K]} a_t^k)$, (iii) by adding and subtracting $h^0 \sum_{k \in [K]} I_t^k$, and (iv) from the definition of $v(\cdot)$ (see (28)).

C.5. Cost-to-go of relaxed systems depends solely on Z_t

LEMMA 6. *For any $I_t \in \mathbb{R}^{K+1}$, the cost-to-go for the partially relaxed system (and the fully relaxed system) depends on I_t solely through the total inventory on hand $Z_t = \sum_{k \in [K]_0} I_t^k$.*

We prove the result by backward induction in t. Let $\breve{J}_t(I_t)$ be the optimal cost-to-go for the partially relaxed setting when starting from period t at state I_t. Clearly, $\breve{J}_T(I_T) = \hat{R}_T(Z_T)$ depends only on Z_T. Suppose $\breve{J}_{t+1}(I_{t+1})$ depends only on $Z_{t+1} = Z_t + a_t^0 - \Xi_t$. As deduced in (34), $\breve{J}_t(I_t)$ satisfies the following Bellman Equation

$$\breve{J}_t(I_t) = \hat{R}(Z_t) + \min_{a_t^0 \in \mathbb{R}^+} \mathbb{E}_{\Xi_t} \left[\breve{J}_{t+1}(Z_t + a_t^0 - \Xi_t) \right], \tag{39}$$

for $\hat{R}(\cdot)$ defined in (33). (We abuse notation in writing $\breve{J}_{t+1}(I_{t+1}) = \breve{J}_{t+1}(Z_{t+1})$, since by induction hypothesis $\breve{J}_{t+1}(I_{t+1})$ depends only on Z_{t+1}.) Since the right-hand side depends only on Z_t so does the left-hand side. Induction on $t = T-1, T-2, \ldots, 1$ completes the proof.

C.6. Convexity of $\hat{R}(Z)$, $\breve{J}_t$ and $\hat{J}_t$

[Proof of Lemma 2] We first show the convexity of $\hat{R}(\cdot)$. Note that for every demand trace, $\sum_{k\in[K]}[p^k(\xi_t^k - y^k)^+ + h^k(y^k - \xi_t^k)^+ - h^0 y^k]$ is convex in y^k, as linear functions are convex, the positive part operator preserves convexity (as it is the maximum of linear functions), and the sum of convex functions is also convex. Further, as expectation preserves convexity, we have that $\sum_{k\in[K]}\mathbb{E}_{\xi_t^k}[(p^k(\xi_t^k - y^k)^+ + h^k(y^k - \xi_t^k)^+ - h^0 y^k)]$ is also convex in y^k.

Consider arbitrary $\underline{Z} \in \mathbb{R}^+$ and $\overline{Z} \in \mathbb{R}^+$ and let $\underline{y}$, $\overline{y}$ be the store inventory level vectors for which the minima of the problem (33) (which defines $\hat{R}(\cdot)$) with initial total inventory $\underline{Z}$ and $\overline{Z}$, respectively, are attained. For any $\lambda \in (0,1)$, letting $y = \lambda\underline{y} + (1-\lambda)\overline{y}$, we have that

$$\begin{aligned}\sum_{k\in[K]} y^k &= \sum_{k\in[K]} (\lambda \underline{y}^k + (1-\lambda)\overline{y}^k) \\ &= \lambda \sum_{k\in[K]} \underline{y}^k + (1-\lambda)\sum_{k\in[K]} \overline{y}^k \\ &\le \lambda\underline{Z} + (1-\lambda)\overline{Z},\end{aligned}$$

where we used the feasibility of $\underline{y}$ (resp. $\overline{y}$) for problem (33) with $\underline{Z}$ (resp. $\overline{Z}$). Hence, y is a feasible solution to the problem (33) with initial total inventory $Y = \lambda\underline{Z} + (1-\lambda)\overline{Z}$. Further, note that

$$\begin{aligned}v(Y, y^1, \ldots, y^K) &= h^0(\lambda\underline{Z} + (1-\lambda)\overline{Z}) + \sum_{k\in[K]} \mathbb{E}_{\xi_t^k}\left[\left(p^k(\xi_t^k - y^k)^+ + h^k(y^k - \xi_t^k)^+ - h^0 y^k\right)\right] \\ &\overset{(i)}{\le} h^0(\lambda\underline{Z} + (1-\lambda)\overline{Z}) + \lambda \sum_{k\in[K]} \mathbb{E}_{\xi_t^k}\left[\left(p^k(\xi_t^k - \underline{y}^k)^+ + h^k(\underline{y}^k - \xi_t^k)^+ - h^0 \underline{y}^k\right)\right] \\ &\quad + (1-\lambda)\sum_{k\in[K]} \mathbb{E}_{\xi_t^k}\left[\left(p^k(\xi_t^k - \overline{y}^k)^+ + h^k(\overline{y}^k - \xi_t^k)^+ - h^0 \overline{y}^k\right)\right] \\ &= \lambda\hat{R}(\underline{Z}) + (1-\lambda)\hat{R}(\overline{Z}),\end{aligned}$$

where (i) follows from the convexity of $\sum_{k\in[K]}\mathbb{E}_{\xi_t^k}[(p^k(\xi_t^k - y^k)^+ + h^k(y^k - \xi_t^k)^+ - h^0 y^k)]$ in y^k. Thus, clearly, $\hat{R}(\lambda\underline{Z} + (1-\lambda)\overline{Z}) = \hat{R}(Y) \le v(Y, y^1, \ldots, y^k) \le \lambda\hat{R}(\underline{Z}) + (1-\lambda)\hat{R}(\overline{Z})$. Since this holds for arbitrary arbitrary $\underline{Z} \in \mathbb{R}^+$ and $\overline{Z} \in \mathbb{R}^+$, we conclude that $\hat{R}(\cdot)$ is convex.

The convexity of $\breve{J}_t(\cdot)$ and $\hat{J}_t(\cdot)$ follows by backward induction in t. We provide the argument for $\breve{J}_t(\cdot)$, and the argument for $\hat{J}_t(\cdot)$ is similar. Note that $\breve{J}_T(\cdot) = \hat{R}(\cdot)$ is convex as shown above. Suppose $\breve{J}_{t+1}(\cdot)$ is convex. Recall the Bellman equation (34) for the partially relaxed system. We have shown above that the first term on the right $\hat{R}(\cdot)$ is convex. We will now argue that the second term on the right of (34) is also convex. Let $y_t^0 \equiv a_t^0 + Z_t$ and $\breve{\ell}(y_t^0) \equiv \mathbb{E}_{\Xi_t}\left[\breve{J}_{t+1}(y_t^0 - \Xi_t)\right]$. Since $\breve{J}_{t+1}(\cdot)$ is convex, so is $\breve{\ell}(\cdot)$. The second term on the right of (34) is nothing but

$$\min_{y_t^0 \ge Z_t} \breve{\ell}(y_t^0)\,.$$

Let $y^* \equiv \arg\min_{y_t^0 \geq \mathbb{R}} \breve{\ell}(y_t^0)$ be the minimizer of $\breve{\ell}(\cdot)$. Then,

$$\min_{y_t^0 \geq Z_t} \breve{\ell}(y_t^0) = \begin{cases} \breve{\ell}(y^*) \text{ for } Z_t \leq y^* \\ \breve{\ell}(Z_t) \text{ for } Z_t > y^* \end{cases},$$

and its convexity follows from the convexity of $\breve{\ell}(y_t^0)$ for $y_t^0 \geq y^*$. Since the right-hand side of (34) is the sum of convex functions, we deduce that $\breve{J}_t(\cdot)$ is convex. Induction on $t = T-1, T-2, \ldots, 1$ completes the proof.

C.7. Proof of Proposition 1

[Proof of Proposition 1.]

We will first show the inequality $\hat{J}_1(Z_1) \leq \breve{J}_1(Z_1)$. We will show by induction for $t = T, T-1, \ldots, 1$ that for any total inventory level Z_t at time t, the optimal cost-to-go in the fully relaxed setting is weakly smaller than that of the partially relaxed setting $\hat{J}_t(Z_t) \leq \breve{J}_t(Z_t)$. The desired result $\hat{J}_1(Z_1) \leq \breve{J}_1(Z_1)$ will follow for the cost-to-go from the initial period.

We clearly have $\hat{J}_T(Z_T) = \breve{J}_T(Z_T) = \hat{R}(Z_T)$ for every Z_T. This forms our induction base. As our induction hypothesis, let us assume that $\hat{J}_{t+1}(Z_{t+1}) \leq \breve{J}_{t+1}(Z_{t+1})$ for all $Z_{t+1} \in \mathbb{R}_+$. We now show that $\hat{J}_t(Z_t) \leq \breve{J}_t(Z_t)$ for all $Z_t \in \mathbb{R}_+$. Let $\hat{\pi}$ and $\breve{\pi}$ be optimal policies for the partially relaxed and the fully relaxed systems, respectively. Recall that $\hat{U}_t = \sum_{k \in [K]} U_t^k$ is the sum of the K uniforms at time t and that $\breve{J}_{t+1}(Z)$ is convex in Z by Lemma 2. Therefore, for any Z_t, we have

$$\begin{aligned}
\breve{J}_t(Z_t) &= \hat{R}(Z_t) + \mathbb{E}_{B_t}\left[\mathbb{E}_{\hat{U}_t}\left[\breve{J}_{t+1}(Z_t + a_t^{\breve{\pi}0} - B_t\hat{U}_t)\right]\right] \\
&\overset{(i)}{\geq} \hat{R}(Z_t) + \mathbb{E}_{B_t}\left[\breve{J}_{t+1}(Z_t + a_t^{\breve{\pi}0} - B_t\hat{\mu})\right] \\
&\overset{(ii)}{\geq} \hat{R}(Z_t) + \mathbb{E}_{B_t}\left[\hat{J}_{t+1}(Z_t + a_t^{\breve{\pi}0} - B_t\hat{\mu})\right] \\
&\overset{(iii)}{\geq} \hat{R}(Z_t) + \mathbb{E}_{B_t}\left[\hat{J}_{t+1}(Z_t + a_t^{\hat{\pi}0} - B_t\hat{\mu})\right] \\
&\overset{(iv)}{=} \hat{J}_t(Z_t),
\end{aligned}$$

where (i) follows by applying Jensen's inequality to $\breve{J}(\cdot)$, (ii) by hypothesis, and (iii) by the fact that $\hat{\pi}$ minimizes $\hat{J}_{t+1}(Z_t + a^{\pi 0} - B_t\hat{\mu})$, and (iv) is just the Bellman equation for the fully relaxed system. Therefore, we can conclude that $\hat{J}_t \leq \breve{J}_t$ for every $t \in [T]$. In particular, it holds for $t = 1$ and $\hat{Z}_t = Z_t$.

We clearly have the second inequality $\breve{J}_1(Z_1) \leq J_1(Z_1)$ given that the action space in the partially relaxed setting contains that of the original setting.

C.8. Proof of Lemma 3

Equation (35) takes the form of an inventory problem for a single location under convex costs. Following Federgruen and Zipkin (1984a), the optimal policy is given by a base-stock policy (we refer to it as an echelon-stock policy given that it considers the system-wide level of inventory).

To show that the policy is stationary, let $\hat{S}_t$ for every $t \in [T]$ be the optimal echelon-stock level at period t. Note that for all $t \in [T]$,

$$\hat{S} = \arg\min_{S \in \mathbb{R}^+} \mathbb{E}_{B_t}\left[\hat{R}(S - B_t\hat{\mu})\right] \tag{40}$$

by definition of $\hat{S}$ and the fact that B_t is i.i.d. Consequently, $\inf_{a_t^0 \geq 0} \left\{ \mathbb{E}_{B_t} \left[\hat{R}(Z_t + a_t^0 - B_t\hat{\mu}) \right] \right\} = \left\{ \mathbb{E}_{B_t} \left[\hat{R}(\max(\hat{S}, Z_t) - B_t\hat{\mu}) \right] \right\}$, where we further used convexity of $\hat{R}(\cdot)$ (Lemma 2).

We will show that $\hat{S}_t = \hat{S}$ for all $t \in [T-1]$ by backward induction in $t = T-1, T-2, \ldots, 1$. Plugging in the echelon stock policy, the Bellman Equation at period $T-1$ takes the form

$$\hat{J}_{T-1}(Z_{T-1}) = \hat{R}(Z_{T-1}) + \mathbb{E}_{B_{T-1}} \left[\hat{R}(\max(\hat{S}_{T-1}, Z_{T-1}) - B_{T-1}\hat{\mu}) \right],$$

and hence $\hat{S}_{T-1} = \hat{S}$ using (40).

Suppose $\hat{S}_t = \hat{S}$. We will show that $\hat{S}_{t-1} = \hat{S}$. For period $t-1$, the Bellman Equation takes the form

$$\begin{aligned}
\hat{J}_{t-1}(Z_{t-1}) &= \hat{R}(Z_{t-1}) + \mathbb{E}_{B_{t-1}} \left[\hat{J}_t(\hat{S}_{t-1} - B_{t-1}\hat{\mu}) \right] \\
&= \hat{R}(Z_{t-1}) + \mathbb{E}_{B_{t-1}} \left[\hat{R}(\hat{S}_{t-1} - B_{t-1}\hat{\mu}) \right] \\
&\quad + \underbrace{\mathbb{E}_{B_{t-1}} \left[\inf_{a_t^0 \geq \hat{S}_{t-1} - B_{t-1}} \left\{ \mathbb{E}_{B_t} \left[\hat{J}_{t+1} \left((a_t^0 + \hat{S}_{t-1} - B_{t-1}\hat{\mu}) - B_t\hat{\mu} \right) \right] \right\} \right]}_{A(\hat{S}_{t-1})}.
\end{aligned}$$

Note that $\hat{S}_{t-1}$ affects the second and third terms on the right. We will show that $\hat{S}_{t-1} = \hat{S}$ causes each of the second and third terms to be individually minimized. For the second term, this follows from (40). Consider the third term. Setting $\hat{S}_{t-1} = \hat{S}$ implies that $\hat{S}_{t-1} - B_{t-1}\hat{\mu} \leq \hat{S}$ for both possible values of B_{t-1}. Consequently,

$$A(\hat{S}) = \mathbb{E}_{B_t} \left[\hat{J}_{t+1}(\hat{S} - B_t\hat{\mu}) \right] = \min_{S \in \mathbb{R}} \mathbb{E}_{B_t} \left[\hat{J}_{t+1}(S - B_t\hat{\mu}) \right]$$

by our induction hypothesis that $\hat{S}_t = \hat{S}$, *i.e.*, setting $\hat{S}_{t-1} = \hat{S}$ also causes the third term to be minimized. We deduce that $\hat{S}_{t-1} = \hat{S}$. Induction completes the proof that the optimal echelon stock level is time invariant and equal to $\hat{S}$ for $t = T-1, T-2, \ldots, 1$.

C.9. Proof of Lemma 4

In Appendix C.8 we showed that the optimal base-stock level $\hat{S}$ for the fully relaxed system must minimize $q\hat{R}(\hat{S} - D^{\mathrm{H}}) + (1-q)\hat{R}(\hat{S} - D^{\mathrm{L}})$. Recall that $\underline{y}$ and $\overline{y}$ denote the optimal solutions for problem (33) with total inventory $\hat{S} - D_t^{\mathrm{H}}$ and $\hat{S} - D_t^{\mathrm{L}}$, respectively.

We will first prove that $\hat{S}$ is such that $\underline{y}^k \geq \gamma^{\mathrm{L}} a^k$ for every k. Suppose the opposite, *i.e.*, $\hat{S}$ is such that $\underline{y}^k < \gamma^{\mathrm{L}} a^k$ for some k. Let $\delta^k > 0$ be such that $\underline{y}^k + \delta^k < \gamma^{\mathrm{L}} a^k$, and consider a base-stock level $\hat{S} + \delta^k$. In the problem (33) with total inventory $(\hat{S} + \delta^k - D_t^{\mathrm{H}})$, we can construct a feasible solution $\underline{y}(\delta^k)$ such that $\underline{y}(\delta^k)^k = \underline{y}^k + \delta^k$ and $\underline{y}(\delta^k)^j = \underline{y}^j$ for all $j \neq k$, reducing cost D by $p^k\delta^k$ as the extra δ_k will be sold immediately w.p. 1 given that the realized demand will be at least $\gamma^{\mathrm{L}} a^k$. We can keep $\overline{y}$ unperturbed, increasing cost by $h^0\delta^k$, as there are an additional δ^k units being held at the warehouse. Therefore, the expected cost is reduced by at least $qp^k\delta^k - (1-q)h^0\delta^k > 0$, so $\hat{S}$ could not have been optimal. We thus conclude that $\underline{y}^k \geq \gamma^{\mathrm{L}} a^k$ for every $k \in [K]$.

Now, note that the KKT conditions of $\hat{R}(Z)$ can be written as:

$$\begin{aligned}
-p^k\mathbb{P}(\xi_t^k \geq y^k) + h^k\mathbb{P}(\xi_t^k \leq y^k) - h^0 + \lambda = 0 \quad \forall y \in [K] \\
\lambda \cdot (\sum_{k \in [K]} y^k - Z) = 0, \\
\sum_{k \in [K]} y^k - Z \leq 0, \\
\lambda \geq 0.
\end{aligned} \tag{41}$$

It can be easily verified that the optimal y^k for a problem $\hat{R}(Z)$ are weakly increasing in Z by the optimality conditions (41) and the convexity of costs. Therefore, $\overline{y} \geq \underline{y}$. Clearly, as $h^0 < h^k$, a solution such that $\overline{y}^k > \gamma^{\mathrm{H}} b^k$ cannot be optimal, as the units above $\gamma^{\mathrm{H}} b^k$ will not be sold immediately w.p. 1. We can hence conclude that $\gamma^{\mathrm{H}} b^k \geq \overline{y}^k \geq \underline{y}^k \geq \gamma^{\mathrm{L}} a^k$ for each $k \in [K]$.

The proof for the fact that an optimal policy for the fully relaxed system does not move inventory from a store to other stores or to the warehouse was provided in the paragraph immediately after the lemma statement.

Additionally, the first condition in (41) implies that $y^k = F^{-1}_{\xi_t^k}\left(\frac{p^k+h^0-\lambda}{p^k+h^k}\right)$ for every $k \in [K]$. This allows us to write $\underline{y}^k = \underline{G}(\mathcal{R}^k) = F^{-1}_{\xi_t^k}\left(\frac{p^k+h^0-\underline{\lambda}}{p^k+h^k}\right)$ and $\overline{y}^k = \overline{G}(\mathcal{R}^k) = F^{-1}_{\xi_t^k}\left(\frac{p^k+h^0-\overline{\lambda}}{p^k+h^k}\right)$, with $\underline{\lambda}$ and $\overline{\lambda}$ the optimal values of dual variable λ for problem (33) with initial total inventories $\hat{S} - D_t^{\mathrm{H}}$ and $\hat{S} - D_t^{\mathrm{L}}$, respectively. Note that $\underline{G}$ and $\overline{G}$ are continuous functions of problem primitives $\mathcal{R}^k$, since the assumptions outlined in Example 1 imply that $p^k + h^k \geq \underline{p} + \underline{h} > 0$ for every $k \in [K]$.

C.10. Proof of Proposition 2

To prove this proposition, we define some specialized notation. Let $c^{\tilde{\pi}}(I_t) = v\left(Z_t, a_t^{\tilde{\pi}1}(I_t), \ldots, a_t^{\tilde{\pi}K}(I_t)\right)$ be the immediate cost under $\tilde{\pi}$. Recall that $Z_{t+1} = \hat{S} - \Xi_t$ under $\tilde{\pi}$ and that $\hat{Z}_{t+1} = \hat{S} - D_t$ following the optimal policy for the fully relaxed setting. We will use the following two lemmas, which we will prove at the end of this section.

Let $\alpha = |\hat{R}(\hat{S} - D^{\mathrm{L}}) - \hat{R}(\hat{S} - D^{\mathrm{H}})| + \max\{\overline{p}, \overline{h}\}(|D^{\mathrm{H}} - D^{\mathrm{L}}|)$.

The next lemma bounds the immediate cost under $\tilde{\pi}$ by $\hat{R}(Z_t)$, which plays the role of the immediate cost in the partially relaxed system (See (34)) plus two extra terms. The term $\max\{\overline{p}, \overline{h}\}(|\hat{Z}_t - Z_t|)$ penalizes deviations in overall system inventory from the level $\hat{Z}_t$ observed in the fully relaxed system and the term $\mathbb{1}(\tilde{W}_t(Z_t) \neq W_t)\alpha$ penalizes cases when the the "estimate" $\tilde{W}_t$ of whether demand was high or low in the previous period was incorrect.

Lemma 7. *In any period t, with probability 1,*

$$c^{\tilde{\pi}}(I_t) \leq \hat{R}(\hat{Z}_t) + \max\{\overline{p}, \overline{h}\}(|\hat{Z}_t - Z_t|) + \mathbb{1}(\tilde{W}_t \neq W_t)\alpha. \tag{42}$$

The next lemma bounds key terms in (42) in terms of the deviations of the sum of zero-mean uniforms random variables from the sum's mean. Recall that $\tilde{U}_1^K = \sum_{k\in[K]}(U_1^k - \mu^k)$ is the sum of K zero-mean uniforms.

Lemma 8. *For $t \in \{2, \ldots, T\}$,*

$$\mathbb{E}\left[\left|\hat{Z}_t - Z_t\right|\right] \leq \gamma^{\mathrm{H}} \mathbb{E}\left[\left|\tilde{U}_1^K\right|\right]. \tag{43}$$

and

$$\mathbb{P}\left(\tilde{W}_t \neq W_t\right) \leq \mathbb{P}\left(\left|\tilde{U}_1^K\right| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right). \tag{44}$$

We now prove Proposition 2.

⌊ Proof of Proposition 2] By taking expectation on (42), using (43) and (44), we get that for $t=2,\ldots,T$

$$\begin{aligned}\mathbb{E}[c^{\tilde{\pi}}(I_t)] &\leq \mathbb{E}\left[\left(\hat{R}(\hat{Z}_t)+\max\{\overline{p},\overline{h}\}(|\hat{Z}_t-Z_t|)\right)+\alpha\mathbb{1}(\tilde{W}_t\neq W_t)\right]\\ &\leq \mathbb{E}[\hat{R}(\hat{Z}_t)]+\max\{\overline{p},\overline{h}\}\gamma^{\mathrm{H}}\mathbb{E}\left[\left|\tilde{U}_1^K\right|\right]+\alpha\mathbb{P}\left(\left|\tilde{U}_1^K\right|\geq\hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right).\end{aligned}$$

Using this inequality, we conclude that

$$\begin{aligned}J_1^{\tilde{\pi}}(I_t) &= c^{\tilde{\pi}}(I_1)+\mathbb{E}\left[\sum_{t=2}^{T}c^{\tilde{\pi}}(I_t)\right]\\ &\leq c^{\tilde{\pi}}(I_1)+\mathbb{E}\left[\sum_{t=2}^{T}\hat{R}(\hat{Z}_t)\right]+(T-1)\left[\max\{\overline{p},\overline{h}\}\gamma^{\mathrm{H}}\mathbb{E}_{\tilde{U}_1^K}\left[\left|\tilde{U}_1^K\right|\right]+\alpha\mathbb{P}\left(\left|\tilde{U}_1^K\right|\geq\hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right)\right]\\ &= \hat{R}(Z_1)+\mathbb{E}\left[\sum_{t=2}^{T}\hat{R}(\hat{Z}_t)\right]+(T-1)\left[\max\{\overline{p},\overline{h}\}\gamma^{\mathrm{H}}\mathbb{E}_{\tilde{U}_1^K}\left[\left|\tilde{U}_1^K\right|\right]+\alpha\mathbb{P}\left(\left|\tilde{U}_1^K\right|\geq\hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right)\right]\\ &= \hat{J}_1(Z_1)+(T-1)\left[\max\{\overline{p},\overline{h}\}\gamma^{\mathrm{H}}\mathbb{E}_{\tilde{U}_1^K}\left[\left|\tilde{U}_1^K\right|\right]+\alpha\mathbb{P}\left(\left|\tilde{U}_1^K\right|\geq\hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right)\right]\\ &\leq \hat{J}_1(Z_1)+T\left[\max\{\overline{p},\overline{h}\}\gamma^{\mathrm{H}}\mathbb{E}_{\tilde{U}_1^K}\left[\left|\tilde{U}_1^K\right|\right]+\alpha\mathbb{P}\left(\left|\tilde{U}_1^K\right|\geq\hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right)\right].\end{aligned}$$

The first and third equalities apply the definitions of the cost-to-go functions under the two systems. The second equality uses that $c^{\tilde{\pi}}(I_1)=v\left(Z_1,a_1^{\tilde{\pi}1}(I_1),\ldots,a_1^{\tilde{\pi}K}(I_1)\right)=\hat{R}(Z_1)=\hat{R}(\hat{Z}_1)$ because of how the policy $\tilde{\pi}$ is constructed and because the initial inventory levels (Z_1 and $\hat{Z}_1$) in the two systems are equal by construction.

We now return to prove the lemmas stated above.

Lemma 7 follows from two facts: (i) Whenever the policy $\tilde{\pi}$ correctly "estimated" whether the demand in the previous period was low (*i.e.*, $\tilde{W}_t=W_t$) we can bound the difference in the costs incurred in the fully relaxed and original systems by a quantity proportional to the difference in system-wide inventories among systems. Here we make use of Lemma 4 which assures us that we can do as well in the original system as in a fully relaxed system with the same system-wide inventory. (ii) Whenever the estimation is wrong (*i.e.*, $\tilde{W}_t\neq W_t$), we incur an additional cost of at most $|\hat{R}(\hat{S}-D^{\mathrm{H}})-\hat{R}(\hat{S}-D^{\mathrm{L}})|$.

⌊ Proof of Lemma 7.]

Let $r^k(y^k)=\mathbb{E}_{\xi_t^k}\left[(p^k(\xi_t^k-y^k)^+ + h^k(y^k-\xi_t^k)^+ - h^0y^k)\right]$, so that $v(Z,y^1,\ldots,y^K)=h^0Z+\sum_{k\in[K]}r^k(y^k)$. It is clear that

$$r^k(y^k-x)-r^k(y^k)\leq\overline{p}x \tag{45}$$

for all $k\in[K]$ as, in the worst case, x fewer units can cause an additional underage cost of $\overline{p}x$.

Now, recall that, given the definition of $\tilde{W}_t$, we will have that $\tilde{\pi}$ will follow the base-stock levels $\overline{y}$ whenever $\hat{S}-\frac{D^{\mathrm{L}}+D^{\mathrm{H}}}{2}\leq Z_t$. Let us first address two separate cases in which $\tilde{\pi}$ follows $\overline{y}$. These differ on whether $Z_t\leq\hat{S}-D^{\mathrm{L}}$ or $Z_t>\hat{S}-D^{\mathrm{L}}$.

Let us first consider the case $Z_t>\hat{S}-D^{\mathrm{L}}$. We have

$$\begin{aligned}R^{\tilde{\pi}}(I_t) &= h^0Z_t+\sum_{k\in[K]}r^k(a_t^{\tilde{\pi}k}(I_t))\\ &\overset{(i)}{=} h^0Z_t+\sum_{k\in[K]}r^k(\overline{y}^k)\end{aligned}$$

$$\overset{(ii)}{=} h^0(Z_t - (\hat{S} - D^{\mathrm{L}})) + h^0(\hat{S} - D^{\mathrm{L}}) + \sum_{k\in[K]} r^k(\overline{y}^k)$$
$$\overset{(iii)}{=} h^0(Z_t - (\hat{S} - D^{\mathrm{L}})) + \hat{R}(\hat{S} - D^{\mathrm{L}}),$$

where (i) follows by the fact that, as implied by Lemma 4, $a_t^{\tilde{\pi}k}(I_t) = \overline{y}^k$ whenever $Z_t > \hat{S} - D^{\mathrm{L}}$, (ii) by adding and subtracting $h^0(\hat{S} - D^{\mathrm{L}})$ and (iii) by the definition of $\hat{R}(\hat{S} - D^{\mathrm{L}})$

Similarly, if $\hat{S} - \frac{D^{\mathrm{L}}+D^{\mathrm{H}}}{2} \leq Z_t \leq \hat{S} - D^{\mathrm{L}}$, we have

$$\begin{aligned}
R^{\tilde{\pi}}(I_t) &= h^0 Z_t + \sum_{k\in[K]} r^k(a_t^{\tilde{\pi}k}(I_t)) \\
&\overset{(i)}{\leq} h^0(\hat{S} - D^{\mathrm{L}}) + \sum_{k\in[K]} r^k(a_t^{\tilde{\pi}k}(I_t)) \\
&\overset{(ii)}{=} h^0(\hat{S} - D^{\mathrm{L}}) + \sum_{k\in[K]} r^k(\overline{y}^k) + \sum_{k\in[K]} (r^k(a_t^{\tilde{\pi}k}(I_t)) - r^k(\overline{y}^k)) \\
&\overset{(iii)}{\leq} h^0(\hat{S} - D^{\mathrm{L}}) + \sum_{k\in[K]} r^k(\overline{y}^k) + \overline{p}(\hat{S} - D^{\mathrm{L}} - Z_t) \\
&\overset{(iv)}{=} \hat{R}(\hat{S} - D^{\mathrm{L}}) + \overline{p}(\hat{S} - D^{\mathrm{L}} - Z_t),
\end{aligned}$$

where (i) follows from $Z_t \leq \hat{S} - D^{\mathrm{L}}$, (ii) by adding and subtracting $\sum_{k\in[K]} r^k(\overline{y}^k)$, (iii) from (45) and the fact that, using Lemma 4, $\sum_{k\in[K]}(\overline{y}^k - a_t^{\tilde{\pi}k}(I_t)) = (\hat{S} - D^{\mathrm{L}} - Z_t)$ whenever $Z_t \leq \hat{S} - D^{\mathrm{L}}$, and (iv) from the definition of $\hat{R}(\hat{S} - D^{\mathrm{L}})$.

Let $\overline{c} = \max\{\overline{p}, \overline{h}\}$. Given that $\overline{h} > h^0$, we obtain that whenever $\hat{S} - \frac{D^{\mathrm{L}}+D^{\mathrm{H}}}{2} \leq Z_t$,

$$\begin{aligned}
R^{\tilde{\pi}}(I_t) &\leq \hat{R}(\hat{S} - D^{\mathrm{L}}) + \overline{c}(|\hat{S} - D^{\mathrm{L}} - Z_t|) \\
&= \mathbb{1}(\tilde{W}_t = W_t)[\hat{R}(\hat{S} - D^{\mathrm{L}}) + \overline{c}(|\hat{S} - D^{\mathrm{L}} - Z_t|)] \\
&\quad + \mathbb{1}(\tilde{W}_t \neq W_t)[\hat{R}(\hat{S} - D^{\mathrm{L}}) + \overline{c}(|\hat{S} - D^{\mathrm{L}} - Z_t|)]
\end{aligned} \tag{46}$$

Note that under $\{\tilde{W}_t = W_t\}$, we have that $\hat{S} - D^{\mathrm{L}} = \hat{Z}_t$. Conversely, if $\{\tilde{W}_t \neq W_t\}$, we have that $\hat{S} - D^{\mathrm{H}} = \hat{Z}_t$. Under $\{\tilde{W}_t \neq W_t\}$, we have

$$\begin{aligned}
\hat{R}(\hat{S} - D^{\mathrm{L}}) + \overline{c}(|\hat{S} - D^{\mathrm{L}} - Z_t|) &= \hat{R}(\hat{S} - D^{\mathrm{L}}) - \hat{R}(\hat{S} - D^{\mathrm{H}}) + \overline{c}(|\hat{S} - D^{\mathrm{L}} - Z_t| - |\hat{S} - D^{\mathrm{H}} - Z_t|) \\
&\quad + [\hat{R}(\hat{Z}_t) + \overline{c}(|\hat{Z}_t - Z_t|)] \\
&\overset{(i)}{\leq} |\hat{R}(\hat{S} - D^{\mathrm{L}}) - \hat{R}(\hat{S} - D^{\mathrm{H}})| + \overline{c}(|D^{\mathrm{H}} - D^{\mathrm{L}}|) \\
&\quad + [\hat{R}(\hat{Z}_t) + \overline{c}(|\hat{Z}_t - Z_t|)] \\
&\leq \alpha + [\hat{R}(\hat{Z}_t) + \overline{c}(|\hat{Z}_t - Z_t|)],
\end{aligned}$$

with $\alpha = |\hat{R}(\hat{S} - D^{\mathrm{L}}) - \hat{R}(\hat{S} - D^{\mathrm{H}})| + \overline{c}(|D^{\mathrm{H}} - D^{\mathrm{L}}|)$, where (i) follows by the triangle inequality.

Applying our previous identities on (46) and rearranging terms, we obtain

$$R^{\tilde{\pi}}(I_t) \leq \left(\hat{R}(\hat{Z}_t) + \max\{\overline{p}, \overline{h}\}(|\hat{Z}_t - Z_t|)\right) + \mathbb{1}(\tilde{W}_t \neq W_t)\alpha \tag{47}$$

The procedure for the cases in which $\tilde{\pi}$ follows $\underline{y}$ are analogous, so we will omit them. We therefore conclude the desired result.

[Proof of Lemma 8.]

Let $\tilde{U}^k_{t-1} = \sum_{k\in[K]}(U^k_{t-1} - \mu^k)$ be the sum of the zero-mean uniforms. Then, following $\tilde{\pi}$, we have $Z_t = \hat{S} - \Xi_{t-1}$. Therefore,

$$\begin{aligned}\left|\hat{Z}_t - Z_t\right| &\overset{(i)}{=} \left|\hat{S} - D_t - (\hat{S} - \Xi_{t-1})\right| \\ &\overset{(ii)}{=} \left|\hat{S} - B_{t-1}\hat{\mu} - (\hat{S} - B_{t-1}\sum_{k\in[K]} U^k_{t-1})\right| \\ &\overset{(iii)}{=} \left|B_{t-1}\sum_{k\in[K]}(U^k_{t-1} - \mu^k)\right| \\ &\overset{(iv)}{\leq} \gamma^{\mathrm{H}}\left|\tilde{U}^K_{t-1}\right|\end{aligned}$$

where (i) and (ii) follow from the definition of Z_t and $\hat{Z}_t$ under their respective policies, (iii) by $\hat{\mu} = \sum_{k\in[K]}\mu^k$ and (iv) by $B_t \leq \gamma^{\mathrm{H}}$. Similarly, noting that the distance between D_t and the "crossing point" $\frac{D^{\mathrm{H}}-D^{\mathrm{L}}}{2}$ is given by

$$\left|D_t - \frac{D^{\mathrm{H}} - D^{\mathrm{L}}}{2}\right| = \hat{\mu}\frac{\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}}}{2}$$

and recalling that the cumulative demand in the original system can be written as $\Xi_{t-1} = B_{t-1}\hat{\mu} + B_{t-1}\tilde{U}^K_{t-1}$, we have that

$$\left(\tilde{W}_t \neq W_t\right) \quad \text{implies} \quad \left(\left|B_{t-1}\tilde{U}^K_{t-1}\right| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}}}{2}\right).$$

As $B_{t-1} \geq \gamma^{\mathrm{H}}$ and $\tilde{U}^K_{t-1}$ are i.i.d, we deduce that

$$\mathbb{P}\left(\tilde{W}_t \neq W_t\right) \leq \mathbb{P}\left(\left|\tilde{U}^K_1\right| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right).$$

C.11. Bounding $\mathbb{E}\left[\left|\tilde{U}^K_1\right|\right]$ and $\mathbb{P}\left(\left|\tilde{U}^K_1\right| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right)$

Recall the definition of the primitive $\kappa \equiv \max_{k\in[K]}\max(\overline{u}^k, 1/\overline{u}^k)$.

Lemma 9. *We have*

$$\mathbb{E}\left[\left|\tilde{U}^K_1\right|\right] \leq \kappa\sqrt{K}. \tag{48}$$

Let $\tilde{U}^k_1 = U^k_1 - \mu^k$ be the zero-mean uniform corresponding to store k. Clearly, $\tilde{U}^k_1 \sim \mathrm{Uniform}(\underline{u}^k - \mu^k, \overline{u}^k - \mu^k)$, and has variance $\mathrm{Var}(\tilde{U}^k_1) = (\overline{u}^k - \underline{u}^k)^2/12$. Thus, $\mathrm{Var}(\tilde{U}^K_1) = \sum_{k\in K}(\overline{u}^k - \underline{u}^k)^2/12$. Let $\Delta \equiv \max_{k\in[K]}(\overline{u}^k - \underline{u}^k) \leq \kappa$.

Now, notice that

$$\begin{aligned}\mathbb{E}\left[\left|\tilde{U}^K_1\right|\right] &\leq \left(\mathbb{E}\left[\left|\tilde{U}^K_1\right|^2\right]\right)^{1/2} = (\mathrm{Var}(\tilde{U}^K_1))^{1/2} \\ &\leq \sqrt{K}\Delta/\sqrt{12} \leq \kappa\sqrt{K/12}.\end{aligned}$$

We drop the $\sqrt{12}$ to reduce the notational burden, leading to the lemma.

LEMMA 10. *We have*

$$\mathbb{P}\left(\left|\tilde{U}_1^K\right| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right) \leq \frac{4\gamma^{\mathrm{H}}\kappa^2}{(\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}})\sqrt{K}}\,. \tag{49}$$

U sing the definition of $\hat{\mu}$ as the total expected sum of uniforms and $\overline{u}^k \geq 1/\kappa\,\forall k \in [K]$, we have the lower bound

$$\hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}} \geq d_1 K\,, \qquad \text{for } d_1 = \frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{4\gamma^{\mathrm{H}}\kappa}\,. \tag{50}$$

By Markov's inequality, we have

$$\mathbb{P}\left(\left|\tilde{U}_1^K\right| \geq \hat{\mu}\frac{\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}}}{2\gamma^{\mathrm{H}}}\right) \leq \mathbb{E}[|\tilde{U}_1^K|]\frac{2\gamma^{\mathrm{H}}}{\hat{\mu}(\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}})} \leq \frac{4\gamma^{\mathrm{H}}\kappa^2}{(\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}})\sqrt{K}}\,, \tag{51}$$

using Lemma 9 and (50) to get the second inequality.

C.12. Proof of Theorem 2

We will use the following lemmas, which we will prove at the end of this subsection.

LEMMA 11. *There exists a constant $C_3 = C_3(q, \underline{p}, \underline{h}, \kappa) > 0$ such that $\hat{J}_1(I_1) \geq C_3 T\sqrt{K}$ for every I_1 and K.*

LEMMA 12. *There exists a constant $C_4 = C_4(\overline{p}, \overline{h}, \gamma^{\mathrm{H}}, \gamma^{\mathrm{L}}, \kappa) < \infty$ such that $\alpha \leq C_4 K$ for every K.*

[Proof of Theorem 2.]

Using the bounds in Lemmas 9 and 10 in Proposition 2 (achievability) we obtain

$$\begin{aligned} J_1^{\tilde{\pi}}(I_1) &\leq \hat{J}_1(Z_1) + T\sqrt{K}\left[\max\{\overline{p},\overline{h}\}\gamma^{\mathrm{H}}\kappa + \alpha\frac{4\gamma^{\mathrm{H}}\kappa^2}{(\gamma^{\mathrm{H}}-\gamma^{\mathrm{L}})}\right] \\ &\leq J_1(I_1) + C_5 T\sqrt{K} \qquad \text{for } C_5 = C_5(\gamma^{\mathrm{H}}, \gamma^{\mathrm{L}}, \overline{p}, \overline{h}, \kappa)\,, \end{aligned} \tag{52}$$

where the second inequality uses $\hat{J}_1(Z_1) \leq J_1(I_1)$ from Proposition 1 (converse) and Lemma 12.

By Lemma 11, there exists $C_3 = C_3(q, \underline{p}, \underline{h}, \kappa) > 0$ such that $J_1(I_1) \geq C_3 KT$. Dividing (52) by $J_1(I_1)$, we obtain

$$\begin{aligned} \frac{J_1^{\tilde{\pi}}(I_1)}{J_1(I_1)} &\leq 1 + \frac{C_5 T\sqrt{K}}{J_1(I_1)} \\ &\leq 1 + \frac{C_5}{C_3\sqrt{K}}\,. \end{aligned}$$

Defining $C \equiv C_5/C_3 < \infty$ yields the theorem.

We now prove Lemmas 11 and 12.

[Proof of Lemma 11.] Let $w^k(y^k) = \mathbb{E}_{\xi_t^k}\left[(p^k(\xi_t^k - y^k)^+ + h^k(y^k - \xi_t^k)^+)\right]$ be the expected cost incurred at period t at store k given that the inventory level before demand is realized is y^k. Define $\overline{\nu}^k = \gamma^{\mathrm{H}}(\underline{u}^k + \overline{u}^k)/2$ and $\underline{\nu}^k = \gamma^{\mathrm{L}}(\underline{u}^k + \overline{u}^k)/2$, the "mid-points" of the support of the demand distribution conditioned on the general demand being *high* and *low*, respectively. Let $\beta = (\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})/(4\kappa) > 0$. For any choice of y^k, we have that the larger distance $\varepsilon(y^k)$ between y^k and $\underline{\nu}^k, \overline{\nu}^k$, defined as

$$\varepsilon(y^k) = \max\{\ |\underline{\nu}^k - y^k|, |\overline{\nu}^k - y^k|\},$$

satisfies $\varepsilon(y^k) \geq (\overline{\nu} - \underline{\nu})/2 = (\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})(\underline{u}^k + \overline{u}^k)/4 \geq (\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})(\overline{u}^k)/4 \geq (\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})/(4\kappa) = \beta$. Without loss of generality, let us assume $|\overline{\nu}^k - y^k| \geq \beta$ and that $y^k < \overline{\nu}^k$ (all other cases are analogous). Now, $\mathbb{P}(\xi_t^k \geq \overline{\nu}^k) \geq q/2 \geq q(1-q)/2$. Further, whenever $\xi_t^k \geq \overline{\nu}^k$, a cost of at least $\min\{\underline{h}, \underline{p}\}\beta$ is incurred. Therefore, we must have that the expected cost per store for each period satisfies $w^k(y^k) \geq \frac{\min\{\underline{h}, \underline{p}\}q(1-q)\beta}{2}$ for every possible y^k. Defining $C_3 \equiv \frac{\min\{\underline{h}, \underline{p}\}q(1-q)\beta}{2} = \frac{\min\{\underline{h}, \underline{p}\}q(\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})}{8\kappa}$, we immediately infer that $\hat{J}_1(I_1) \geq C_3 TK$ for all I_1.

⌊ Proof of Lemma 12.] Recall that $\alpha = |\hat{R}(\hat{S} - D^{\mathrm{L}}) - \hat{R}(\hat{S} - D^{\mathrm{H}})| + \max\{\overline{p}, \overline{h}\}(|D^{\mathrm{H}} - D^{\mathrm{L}}|)$. First, given that $\overline{u}^k < \kappa$ for every $k \in [K]$, we must have that

$$D^H - D^L = (\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})\hat{\mu} \leq (\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})K\kappa. \tag{53}$$

Now, for the problem (33) with inventory $\hat{S} - D^H$, consider its minimizer $\underline{y}$. We can build a feasible solution y to the problem with inventory $\hat{S} - D^L$ by setting $y^k = \underline{y}^k$ for every k. Therefore,

$$\hat{R}(\hat{S} - D^L) - \hat{R}(\hat{S} - D^H) \leq h^0(D^H - D^L), \tag{54}$$

as following y with initial inventory $\hat{S} - D^L$ leads to holding an additional $(D^H - D^L)$ units of inventory in the warehouse, as compared to that by following $\underline{y}$ with initial inventory $\hat{S} - D^H$.

On the other hand, for the problem (33) with inventory $\hat{S} - D^L$, consider its minimizer $\overline{y}$. Let us first consider the case that $\sum_{k\in[K]} \overline{y}^k > D^H - D^L$. We can build a feasible solution $\tilde{y}$ to the problem with inventory $\hat{S} - D^H$ by setting $\tilde{y}^k = \overline{y}^k \frac{\sum_{k\in[K]} \overline{y}^k - (D^H - D^L)}{\sum_{k\in[K]} \overline{y}^k}$. Note that $\sum_{k\in[K]} \overline{y}^k - \sum_{k\in[K]} \tilde{y}^k = D^H - D^L$. Note that the warehouse inventory is the same in the two cases, hence the warehouse holding costs are the same, and each store has lower inventory under $\tilde{y}$ and hence store holding costs are weakly lower under $\tilde{y}$. Recalling that the per-unit cost of "scarcity" is upper bounded by $\overline{p}$ (see (45)), we can conclude that

$$\hat{R}(\hat{S} - D^H) - \hat{R}(\hat{S} - D^L) \leq \overline{p}(D^H - D^L)\,. \tag{55}$$

Now consider the case that $\sum_{k\in[K]} \overline{y}^k < D^H - D^L$. We build a feasible solution to the problem with inventory $\hat{S} - D^H$ by setting $\tilde{y}^k = 0$ for every k. First, clearly, $\sum_{k\in[K]} \overline{y}^k - \sum_{k\in[K]} \tilde{y}^k = \sum_{k\in[K]} \overline{y}^k < D^H - D^L$, so we can bound the excess underage costs incurred across stores by $\overline{p}(D^H - D^L)$, and stores incur no holding costs. Furthermore, the inventory at the warehouse when following $\overline{y}$ for the problem with inventory $\hat{S} - D^L$ is given by

$$\overline{y}^0 = \hat{S} - D^L - \sum_{k\in[K]} \overline{y}^k > \hat{S} - D^H = \tilde{y}^0\,.$$

Hence, warehouse holding costs are smaller in the case of $\tilde{y}$. Therefore, the upper bound in (55) still applies.

Using (53), (54) and (55), and given that $\overline{h} > h^0$, we conclude that

$$\begin{aligned}
\alpha &= |\hat{R}(\hat{S} - D^{\mathrm{L}}) - \hat{R}(\hat{S} - D^{\mathrm{H}})| + \max\{\overline{p}, \overline{h}\}(|D^{\mathrm{H}} - D^{\mathrm{L}}|) \\
&\leq \max\{\overline{p}, \overline{h}\}(D^H - D^L) + \max\{\overline{p}, \overline{h}\}(D^H - D^L) \\
&= 2\max\{\overline{p}, \overline{h}\}(D^H - D^L) \\
&\leq 2\max\{\overline{p}, \overline{h}\}(\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})\kappa K \\
&\leq C_4 K,
\end{aligned}$$

where we set $C_4 = 2\max\{\overline{p}, \overline{h}\}(\gamma^{\mathrm{H}} - \gamma^{\mathrm{L}})\kappa$.