Performance of deep reinforcement learning algorithms in two-echelon inventory control systems

RESEARCH ARTICLE

Performance of Deep Reinforcement Learning Algorithms in Two-Echelon Inventory Control Systems

Francesco Stranieri[a,b], Fabio Stella[a], and Chaaben Kouki[c]

[a]Department of Informatics, Systems, and Communication (DISCo), University of Milano-Bicocca, Viale Sarca, 336, Milan, 20126, Italy; [b]Department of Control and Computer Engineering (DAUIN), Polytechnic of Turin, Corso Duca degli Abruzzi, 24, Turin, 10129, Italy; [c]Department of Operations Management and Decision Science (OMDS), ESSCA School of Management, Rue Joseph Lakanal, 1, Angers, 49100, France;

ARTICLE HISTORY

Compiled 2024-03-04

ABSTRACT

This study conducts a comprehensive analysis of deep reinforcement learning (DRL) algorithms applied to supply chain inventory management (SCIM), which consists of a sequential decision-making problem focused on determining the optimal quantity of products to produce and ship across multiple capacitated local warehouses over a specific time horizon. In detail, we formulate the problem as a Markov decision process for a divergent two-echelon inventory control system characterized by stochastic and seasonal demand, also presenting a balanced allocation rule designed to prevent backorders in the first echelon. Through numerical experiments, we evaluate the performance of state-of-the-art DRL algorithms and static inventory policies in terms of both cost minimization and training time while varying the number of local warehouses and product types and the length of replenishment lead times. Our results reveal that the Proximal Policy Optimization algorithm consistently outperforms other algorithms across all experiments, proving to be a robust method for tackling the SCIM problem. Furthermore, the (s, Q)-policy stands as a solid alternative, offering a compromise in performance and computational efficiency. Lastly, this study presents an open-source software library that provides a customizable simulation environment for addressing the SCIM problem, utilizing a wide range of DRL algorithms and static inventory policies.

KEYWORDS

inventory management; inventory control systems; inventory control policies; artificial intelligence; deep learning; reinforcement learning

CONTACT Francesco Stranieri. Email: francesco.stranieri@polito.it

This is an original manuscript of an article published by Taylor & Francis in International Journal of Production Research on 01 March 2024, available at: https://doi.org/10.1080/00207543.2024.2311180.

1. Introduction

In this study, we address an *optimization problem* that consists of a two-echelon inventory control system with limited capacity at each echelon. The system includes a capacitated source (central warehouse) that receives products from the upstream level (factory). The downstream level consists of multiple retailers (local warehouses) that face stochastic and seasonal demands for a diverse range of product types. Excess demand for each type of product that cannot be directly satisfied from stocks is backordered. Local warehouses place orders at the upstream level, and these orders are delivered after a constant and positive replenishment lead time. The objective is to minimize the total cost of the system under consideration.

This problem can be viewed as a nonlinear stochastic problem whose optimal solution is challenging to find due to the *curse of dimensionality*, which emerges from the number of product types, the number of warehouses, the presence of lead times, and the stochastic nature of the demand (Liu and Tu 2008). The simplest solutions available for one-warehouse-one-product-type inventory control systems or employed in serial supply chains cannot be extended to systems involving multiple warehouses and multiple product types in a multi-echelon structure as the complexity grows with the problem size. By relaxing the assumption of stochastic demand and replacing it with a deterministic value (e.g., with its mean), the system can be modeled as a linear programming problem, which can be technically solved for a reasonable number of warehouses and product types and length of lead times. Nevertheless, for supply chain systems with large-scale instances, even when considering deterministic demand, the problem remains arduous to solve.

Consequently, the goal is to identify an *approximate solution* that provides robust performance in terms of cost minimization or profit maximization, all while maintaining limited computation time. A common approximate approach for dealing with the supply chain inventory management (SCIM) problem involves formulating the system as a Markov decision process (MDP) and solve it approximately. For further information on approximate techniques for solving MDPs, the interested reader can refer to the work of Gordon and Mitchell (1999).

In this context, artificial intelligence (AI) enhances the decision-making capabilities and operational efficiency in supply chains, leading to its *expanding role* in the SCIM field (Sharma et al. 2022). For example, Jackson et al. (2023) demonstrated how recent advances in AI and natural language processing can be leveraged to enable human experts to automatically build simulation models of inventory control systems using verbal descriptions. Additionally, artificial neural networks (ANNs) have been effectively employed to propose a deep learning method for demand forecasting, which has proven robust compared to other benchmarks used in the study (Punia et al. 2020), and to develop a predictive model that determines the likelihood of product backorders in complex scenarios, particularly those characterized by unbalanced training datasets

(Shajalal et al. 2021). According to Rolf et al. (2022), the most common application of reinforcement learning (RL) in supply chains lies in addressing the SCIM problem, with Q-learning emerging as the most utilized algorithm. However, to overcome the limitations of traditional tabular RL algorithms such as Q-learning, deep reinforcement learning (DRL) has recently emerged as a promising approach for efficiently tackling MDP problems.

Given the recent advancements and the increasing application of deep learning in SCIM, this study aims to investigate this trend by comparing different state-of-the-art DRL algorithms in two-echelon inventory control systems. In detail, we focus on evaluating the effectiveness of DRL algorithms by comparing their performance in terms of cost minimization and computation time. It is worth noting that the considered problem remains challenging to solve, even with the implementation of DRL algorithms. Consequently, we assess their performance by simplyfing certain assumptions of the original system. Specifically, we limit the set of experiments by considering one and two product types, a number of local warehouses ranging from one to three, and replenishment lead times of length one and three. These constraints allow all the algorithms used in the benchmark to converge within a reasonable amount of time. The primary objective of this study is thus to evaluate the effectiveness of DRL algorithms and compare their performance with static inventory policies. Through this effort, we aim to provide insights into the relative strengths of each approach and contribute valuable information for decision-makers in selecting appropriate SCIM strategies that balance performance and computational efficiency.

In what follows, Section 2 will offer a review of recent literature on SCIM, specifically focusing on the use of DRL algorithms in this domain. Then, in Section 3, we will outline the general MDP framework of DRL algorithms and detail their application within our assumed inventory control system. In Section 4, we will conduct numerical experiments to assess the performance of the different DRL algorithms and static inventory policies. Finally, Section 5 will conclude the paper, summarizing our findings and discussing their implications for both researchers and practitioners in the field of SCIM.

2. Literature Review

The SCIM problem is a *sequential decision-making problem* that involves determining both the optimal production quantities at the factory and the optimal shipment quantities from the central warehouse to local warehouses over a specified time horizon (either finite or infinite). As evidenced by the roadmap provided by Boute et al. (2022), DRL algorithms are rarely applied to the SCIM field. Nevertheless, they hold the potential to develop near-optimal policies that are challenging, if not impossible, to achieve using traditional methods. Indeed, the uncertain and stochastic nature of product demand combined with lead times presents significant challenges for math-

ematical programming approaches to be effective. Furthermore, when the inventory control system accounts for limited storage capacity, identifying effective SCIM strategies becomes even more complex. As observed by de Kok et al. (2018), the optimal policy for capacitated two-echelon inventory control systems is multidimensional and infeasible for real-world scenarios. In what follows, we review studies that investigate solutions to inventory control systems through RL and DRL algorithms.

Reinforcement Learning Algorithms

RL algorithms have recently achieved remarkable results in the field of AI. Essentially, RL adopts the MDP framework to represent the interactions between a learning agent and an environment (Sutton and Barto 2018), as depicted in Figure 1. To define this interaction, at each time step t, the agent observes the current state of the environment, s_t, chooses an action, a_t, and receives a reward, $r_{t+1} \in \mathbb{R}$; then, the environment transitions into a new state, s_{t+1}. The primary objective of RL is to find an optimal policy that maximizes the sum of *expected discounted rewards*, $\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$, where $0 \leq \gamma \leq 1$ represents a hyperparameter known as the discount rate.

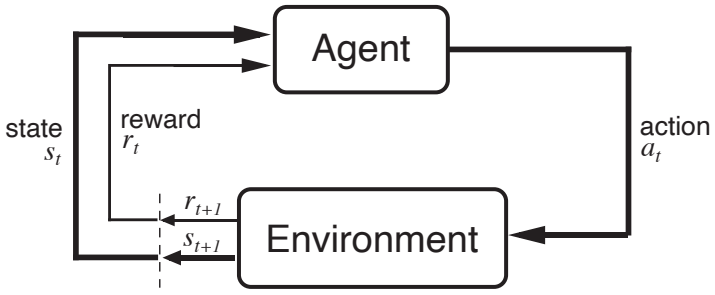


Figure 1. Agent-environment interaction in an MDP (taken from Sutton and Barto (2018)).
Figure 1 Alt Text. Illustration of agent-environment interaction in a Markov decision process.

One of the most common approaches to addressing the SCIM problem through RL algorithms is Q-learning (Yan et al. 2022; Rolf et al. 2022). This approach is rooted in a tabular and temporal-difference algorithm that learns to determine the *value* of an action a_t in a state s_t, referred to as the Q-value and denoted $Q(s_t, a_t)$. The Q-values for each state-action pair are stored in a table known as the Q-table, where each state corresponds to a row and each action to a column. Through the Q-learning algorithm, the Q-values associated with each state-action pair are estimated according to an updating rule. Once convergence has been achieved – which is guaranteed under certain conditions (Jaakkola et al. 1994) – an optimal policy can be derived by identifying the action with the highest Q-value for each state.

Chaharsooghi et al. (2008) proposed a method based on Q-learning to tackle the beer game problem, which consists of a serial supply chain structure with four participants

and specific assumptions. In particular, they defined the current system state using a vector that consists of the current four stock levels. However, since the stock levels thus defined could potentially take infinite values, directly applying this strategy appears unfeasible since the Q-table would become infinitely large. Consequently, the authors *discretized* the state space into nine intervals, resulting in 9^4 possible state values. Regarding actions, their method determines the number of products to order using the $d+x$ policy (precisely, if a participant in the previous time step received a request for d units from the downstream level, this policy requires ordering $d+x$ units to the upstream level in the current time step). The primary goal of the learning process is to determine the optimal value of the unknown variable x based on the given system state. To keep the Q-table's size manageable, the authors *constrained* x to values within the range [0, 3], resulting in 4^4 possible actions.

By defining limited state and action spaces, the resulting Q-table becomes more manageable. However, it becomes evident that the Q-tables implemented are typically huge and, thus, *unscalable*. For example, the Q-table adopted by Chaharsooghi et al. (2008) contains ($9^4 \cdot 4^4 =$) 1679616 cells, equivalent to the number of states multiplied by the number of actions. Consequently, expanding the size of the state or action spaces might not be feasible, as the Q-tables can no longer be handled. Geevers et al. (2023) also considered the beer game problem and demonstrated that the method proposed by Chaharsooghi et al. (2008) is unable to learn an effective policy since the impact of the variable x is so limited that even *random values* yield basically the same outcomes.

In conclusion, tabular RL approaches are feasible only when state and action spaces are discretized or constrained. However, such limitations can lead to a *loss of critical information*, in addition to being inappropriate for real-world scenarios. As a result, enhanced RL methods are necessary for effectively addressing multi-echelon inventory control systems.

Deep Reinforcement Learning Algorithms

DRL combines RL with *deep learning*, enabling it to tackle decision-making problems previously considered intractable. DRL is particularly well-suited for environments with high-dimensional state and action spaces, such as video games and gaming in a general sense (Mnih et al. 2015; Silver et al. 2017; Vinyals et al. 2019). Deep learning is rooted in ANNs, which are capable of providing optimal approximations for highly nonlinear functions. Basically, function parameters are adjusted during the learning process to minimize or maximize a specific objective function.

The DRL algorithms we used belong to a class called *policy-based methods*. These methods can learn a parameterized and stochastic policy to select actions directly, in contrast to value-based methods like Q-learning. Under this class of algorithms, policy gradient methods offer a significant theoretical advantage due to the *policy gradient*

theorem, and the vanilla policy gradient (PG) algorithm (Williams 1992) is a natural result of this theorem. However, the high variance of gradient estimates can lead to policy update instabilities (Wu et al. 2018). To mitigate this issue, Schulman et al. (2015) introduced the trust region policy optimization (TRPO) algorithm, an actor-critic method that simultaneously learns a policy and a value function while bounding the difference between the new and the old policies in a trust region. Proximal policy optimization (PPO) (Schulman et al. 2017) shares the same background as TRPO but is simpler to implement and tune while demonstrating comparable or superior performance. Mnih et al. (2016) proposed the asynchronous advantage actor-critic (A3C) algorithm, which involves multiple agents interacting with different instances of the environment, each with its parameters. These agents periodically and asynchronously update a global ANN that incorporates shared parameters. For a more in-depth and rigorous discussion of various DRL algorithms, interested readers can refer to François-Lavet et al. (2018).

The implementation of DRL algorithms in inventory control systems has been limited, but some promising results have been reported in recent studies. Oroojlooyjadid et al. (2022) used an extension of the deep Q-network (DQN) proposed by Mnih et al. (2015) for the beer game problem. Their DQN agent, which uses an ANN instead of a Q-table to return Q-values for state-action pairs, learned a near-optimal policy while other supply chain participants followed a base-stock policy. Due to the DQN's requirement for a limited action space cardinality, the authors conducted numerical experiments using a $d+x$ policy imposing different constraints on the variable x.

Peng et al. (2019) proposed the PG algorithm to address a two-echelon inventory control system characterized by stochastic and seasonal demand while considering storage capacity constraints. Their experiments demonstrated that the PG agent outperformed the (s, Q)-policy used as a baseline in all three experiments. Gijsbrechts et al. (2022) applied and tuned the A3C algorithm to a similar supply chain structure, showing that A3C achieved performance comparable to state-of-the-art heuristics and approximate dynamic programming algorithms despite its initial tuning being computationally intensive. Alternatively, van Hezewijk et al. (2022) investigated the capacitated single-echelon lot-sizing problem – a type of production planning problem – proposing modifications to the standard PPO algorithm, which matched state-of-the-art benchmarks. These modifications included the use of a feasibility mask and the scaling of states and rewards, aimed at reducing the action space size in large-scale instances.

For more extended inventory control systems, Hubbs et al. (2020) studied a four-echelon structure and employed the PPO algorithm in two different environments (i.e., without and with backorders), demonstrating that PPO outperformed the base-stock policy in both cases. Finally, Alves and Mateus (2020) considered a similar structure and proposed the PPO algorithm to deal with the problem, while a deterministic linear programming model (i.e., considering deterministic demand) is employed as a

baseline. Results of numerical experiments showed that PPO still achieved satisfactory performance.

While these studies highlight the potential of DRL algorithms in multi-echelon inventory control systems, more research is needed to further explore their abilities in this domain.

Contributions to Literature

The current state of DRL algorithms applied in multi-echelon inventory control systems presents several *limitations*. These include insufficient testing across various supply chain typologies (e.g., by varying the number of local warehouses), lack of extensive experiments with different configurations (e.g., number of product types, length of lead times, and associated costs), and an absence of comprehensive comparisons between different state-of-the-art DRL algorithms.

Moreover, relevant aspects of the SCIM problem have not yet been efficiently addressed. Notable gaps include the lack of a well-established open-source library for reproducing and validating simulation models, insufficient performance benchmarking against an oracle, and an absence of multiple product types in multi-echelon inventory control systems, which results in a high-dimensional action space that necessitates a more interconnected and complex set of constraints. In this context, our work contributes to the existing literature by avoiding *infeasible actions* through specific constraints and employing an allocation rule designed to prevent backorders in the first echelon. This contrasts with approaches that combine RL with operations research methods (Wang et al. 2022) or rely on knowledge of the optimal solution in small-scale instances to model the action space in larger-scale ones (van Hezewijk et al. 2022).

In an attempt to address these shortcomings, this paper offers the following *contributions to the literature* on multi-product types and multi-echelon inventory control systems:

- Design and formulation of a simulation environment representing a divergent two-echelon inventory control system under stochastic and seasonal demand, avoiding central warehouse backorders and allowing for an arbitrary number of warehouses and product types to be managed.
- Comparison of a set of state-of-the-art DRL algorithms in terms of their ability to find an optimal policy, i.e., a policy that minimizes the supply chain costs as determined by an oracle.
- Evaluation of performance achieved by DRL algorithms and comparison with selected static inventory policies, such as the base-stock policy and the (s, Q)-policy, with their optimal parameters determined through a data-driven approach.
- Design and execution of a rich experimental setup involving different system

configurations (i.e., considering different lead time lengths and varying numbers of local warehouses and product types) to assess the performance of DRL algorithms and static inventory policies in a wide range of experiments.
- Design and development of an open-source library [1] for addressing the presented SCIM problem, thus embracing open science principles and guaranteeing reproducible results.

Through addressing these gaps and offering these contributions, this paper aims to advance the understanding and application of DRL algorithms in multi-echelon inventory control systems and provides a foundation for future research and advancements in this domain.

3. Problem Definition and Modelling

We consider a finite-horizon, periodic-review, two-echelon inventory control system composed of a factory and its central warehouse (first echelon), denoted by the index zero, along with J local warehouses (second echelon). The structure of the system under consideration is depicted in Figure 2 and is defined as *divergent* because, while the first level consists of a single central warehouse, the second level can include multiple local warehouses. Each local warehouse $j, j \in \{1, \ldots, J\}$, faces, for each time step $t, t \in \{1, \ldots, T\}$, and each product type $i, i \in \{1, \ldots, I\}$, a stochastic and seasonal demand. Any unsatisfied unit of demand that exceeds the on-hand stocks is backordered. Contextually, the central warehouse produces and ships products to local warehouses throughout the finite episode horizon of length T. Products produced by the factory are available to the central warehouse at the same time step, while products shipped to the local warehouse j are received after a positive and constant lead time L_j.

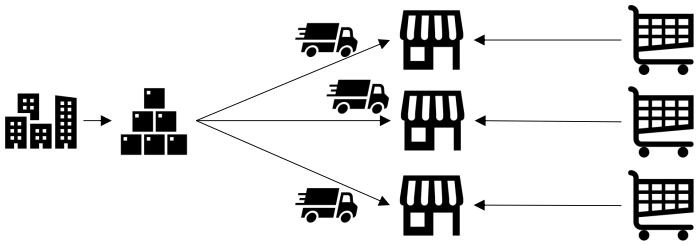


Figure 2. A divergent two-echelon supply chain consisting of a factory and its central warehouse (first echelon) and three local warehouses (second echelon). Shopping carts represent customer demands.
Figure 2 Alt Text. Diagram of a divergent two-echelon supply chain. It consists of a factory connected to a central warehouse, representing the first echelon. From this central warehouse, lines split into three local warehouses, representing the second echelon. Shopping cart icons symbolize customer demands.

The inventory control system proposed in this study primarily draws inspiration

[1] Our open-source library is available at `https://github.com/frenkowski/SCIMAI-Gym`.

from the models presented in Kemmer et al. (2018); Peng et al. (2019); Stranieri and Stella (2022). However, we introduce two critical distinctions to more accurately characterize complex inventory control systems. First, we extended the model to consider a *multi-product type* case in a multi-echelon structure rather than focusing exclusively on a single product type. This extension not only prevents central warehouse backorders but also enables an assessment of the performance in challenging and heterogeneous configurations where multiple product types are managed simultaneously. Secondly, we incorporated *positive lead times* between the central and local warehouses. By accounting for lead times, we capture the inherent uncertainty and delay in the replenishment strategy, which can significantly influence SCIM decisions and the overall system process, allowing for a more in-depth exploration of how lead times impact performance. These distinctions to the original models aim to offer a more comprehensive understanding of the effectiveness and applicability of the DRL algorithms and static inventory policies in managing complex inventory control systems.

For every time step t and each product type i, the factory determines its production level $a^i_{0,t}$, which represents the number of units to produce, considering a fixed production cost of p^i_0 per unit (as mentioned before, we assume $j = 0$ for the central warehouse and $1 \leq j \leq J$ for local warehouses). The central warehouse can store a maximum of c^i_0 units for product type i, implying that the overall capacity is $\sum_{i=1}^{I} c^i_0 = c_0$. The cost of storing one unit of product type i at the central warehouse is h^i_0 per time step, while the corresponding stocks on hand, also referred to as *stock level*, at time step t equals $q^i_{0,t}$.

At every time step t, $a^i_{j,t}$ units of product type i are shipped from the central to local warehouse j, with an associated transportation cost of z^i_j per unit. For each product type i, each local warehouse j has a maximum storage capacity of c^i_j (resulting in an overall capacity of $\sum_{i=1}^{I} c^i_j = c_j$), a storage cost of h^i_j per unit and time step, and a stock level at time t equal to $q^i_{j,t}$. The demand for product type i at local warehouse j during time step t is indicated by $d^i_{j,t}$.

Products in the considered inventory control system are assumed to be non-perishable and provided in discrete quantities. In addition, we assume that each local warehouse is legally obligated to satisfy all submitted demands. Unsatisfied demands are designed as a negative stock level (corresponding to *backorders*) and maintained over time. Consequently, if a demand for a specific time step exceeds the corresponding stock level, a backorder cost, denoted by b^i_j, is applied for each time step and unit of negative stocks. This assumption implies that when the backorder cost is specifically high, the agent may find it challenging to develop cost-effective solutions if it incurs backorders. As a result, an agent should prefer a policy that leads to preserving stocks in advance, even if it means incurring storage costs. By including backorder costs in our model, we offer a more complex and practical scenario for evaluating the DRL algorithms and static inventory policies, enhancing our understanding of the trade-offs between satisfying customer demands and reducing supply chain costs. A summary of

the SCIM notation is available in Table 1.

Parameter	Explanation	Parameter	Explanation
I	Number of Product Types	p_0^i	Production Cost (per unit)
J	Number of Local Warehouses	z_j^i	Transportation Cost (per unit)
T	Episode Horizon (time step)	$q_{j,t}^i$	Stock Level (units)
L_j	Length of Lead Times (time step)	c_j^i	Storage Capacity (units)
$d_{j,t}^i$	Demand (units)	h_j^i	Storage Cost (per unit and time step)
$a_{j,t}^i$	Production and Shipping Level (units)	b_j^i	Backorder Cost (per unit and time step)

Table 1. The adopted SCIM notation accompanied by a relative explanation (and units of measure).

3.1. *Markov Decision Process Formulation*

In this subsection, we formalize the considered SCIM problem using the previously described MDP framework. More precisely, we introduce and define the main components related to the MDP formulation, that is, the *state vector*, the *action vector*, and the *reward function*.

State Vector

Considering time step t, we denote the stocks on hand at warehouse $j, j \in \{0, \dots, J\}$, by:

$$\mathbf{q}_{j,t} := \begin{pmatrix} q_{j,t}^1 \\ q_{j,t}^2 \\ \vdots \\ q_{j,t}^I \end{pmatrix}, \tag{1}$$

where $q_{j,t}^i$ represents the stock level for product type i, $i \in \{1, \dots, I\}$. It is important to note that, due to backorders, the stock level can assume negative values.

The quantity of product type i already ordered from local warehouse j but not yet delivered at time step $t-1, \dots, t-L_j+1$ is represented by $x_{j,t}^i$. In this respect, we express the vector of total *in-transit orders* as follows:

$$\mathbf{x}_{j,t} := \begin{pmatrix} x_{j,t-1}^1 & x_{j,t-2}^1 \cdots & x_{j,t-L_j+1}^1 \\ x_{j,t-1}^2 & x_{j,t-2}^2 \cdots & x_{j,t-L_j+1}^2 \\ \vdots & \vdots & \vdots \\ x_{j,t-1}^I & x_{j,t-2}^I \cdots & x_{j,t-L_j+1}^I \end{pmatrix}. \tag{2}$$

The state vector of the MDP can then be defined as:

$$\mathbf{s}_t := (\mathbf{q}_{0,t}, \mathbf{q}_{1,t}, \mathbf{x}_{1,t}, \ldots, \mathbf{q}_{J,t}, \mathbf{x}_{J,t}). \tag{3}$$

Action Vector

Regarding the actions, the quantities produced and shipped at time step t are denoted by:

$$\mathbf{a}_{j,t} := \begin{pmatrix} a_{j,t}^1 \\ a_{j,t}^2 \\ \vdots \\ a_{j,t}^I \end{pmatrix}, \tag{4}$$

where $j = 0$ indicates the production level and $1 \leq j \leq J$ denotes the shipping level, respectively. In our formulation, we assume that for every warehouse j, $j \in \{0, \ldots, J\}$, and every product type i, $i \in \{1, \ldots, I\}$, the quantities shipped or produced cannot exceed their respective storage capacities, formally $a_{j,t}^i \leq c_j^i$.

The action vector of the MDP is consequently defined as:

$$\mathbf{a}_t := (\mathbf{a}_{0,t}, \ldots, \mathbf{a}_{J,t}), \tag{5}$$

with a size of $I(J + 1)$.

Reward Function

The goal of our study is to minimize the total cost associated with the assumed inventory control system. Consequently, the optimization problem we aim to address is a multi-variable, non-linear, non-convex, mixed-integer programming problem (MINPP).

Mathematically, this optimization problem can be defined as follows:

$$\min \sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{i=1}^{I} z_j^i a_{j,t}^i + h_j^i (q_{j,t}^i)^+ + b_j^i (-q_{j,t}^i)^+ \quad + \sum_{t=1}^{T}\sum_{i=1}^{I} p_0^i a_{0,t}^i + h_0^i q_{0,t}^i \tag{6}$$

$$\text{s.t. } a_{j,t}^i \leq c_j^i \qquad \forall\, j \in \{0,\dots,J\}, \forall\, i \in \{1,\dots,I\} \tag{7}$$

$$a_{j,t}^i + q_{j,t}^i \leq c_j^i \qquad \forall\, j \in \{0,\dots,J\}, \forall\, i \in \{1,\dots,I\} \tag{8}$$

$$\sum_{j=1}^{J} a_{j,t}^i \leq q_{0,t}^i \qquad \forall\, i \in \{1,\dots,I\} \tag{9}$$

$$q_{0,t}^i = q_{0,t-1}^i + a_{0,t}^i - \sum_{j=1}^{J} a_{j,t-1}^i \qquad \forall\, j \in \{0\}, \forall\, i \in \{1,\dots,I\} \tag{10}$$

$$q_{j,t}^i = q_{j,t-1}^i + x_{j,t-L_j}^i - d_{j,t}^i \qquad \forall\, j \in \{1,\dots,J\}, \forall\, i \in \{1,\dots,I\} \tag{11}$$

$$a_{j,t}^i \in \mathbb{N} \qquad \forall\, j \in \{0,\dots,J\}, \forall\, i \in \{1,\dots,I\} \tag{12}$$

$$q_{j,t}^i \in \mathbb{N} \qquad \forall\, j \in \{0\}, \forall\, i \in \{1,\dots,I\} \tag{13}$$

$$q_{j,t}^i \in \mathbb{Z} \qquad \forall\, j \in \{1,\dots,J\}, \forall\, i \in \{1,\dots,I\}, \tag{14}$$

where $(\cdot)^+$ denotes $max(0,\cdot)$.

The objective function expressed in Equation (6) consists of several terms representing various costs. The first term accounts for the transportation cost of quantity $a_{j,t}^i$ from the central to local warehouses. The second term quantifies the storage cost at local warehouses considering the current stock level. This latter is updated through Equation (11) by adding incoming orders $x_{j,t-L_j}^i$ and subtracting the demand $d_{j,t}^i$ from the on-hand stocks $q_{j,t-1}^i$ at the time step $t-1$. The third term, analogous to the second, represents the backorder cost. Clearly, it is possible to incur either storage or backorder costs, but not both, depending on whether the stock level, $q_{j,t}^i$, is positive or negative. Lastly, the final two terms delineate the production and storage costs at the central warehouse, respectively.

Moving on to the constraints, Equations (7) and (8) address the capacity bounds, c_j^i, of each warehouse j and product type i. Specifically, Equation (7) ensures that neither the production nor the shipping level can exceed the warehouse's maximum capacity. Furthermore, Equation (8) also considers the current stock level in a more restrictive form. Equation (9) specifies that the total sum of products shipped to all local warehouses of a specific product type i cannot exceed the on-hand stocks in the central warehouse, $q_{0,t}^i$. This constraint is crucial to guarantee that the central warehouse does not fall into backorders. Finally, Equations (10) and (11) delineate the stock dynamics for the central and local warehouses, respectively, while Equations (12)

to (14) define the domains for the variables, ensuring that production or shipment of a negative number of products is not allowable.

It should be noted that our MINPP formulation is challenging to solve as it is an *NP-hard problem*. When considering a deterministic version of the model (for example, assuming demand as deterministic or known in advance), the MINPP becomes a linear programming problem solvable for a proper number of warehouses and product types and length of lead times. However, with stochastic and seasonal demand distributions, even stochastic programming approaches struggle to find an effective solution within a reasonable amount of time, mainly due to the curse of dimensionality (Stranieri et al. 2024). In what follows, we consequently focus on a simplified versions of our MINPP. Specifically, we address the problem under the conditions of $I = \{1, 2\}$, $J = \{1, 2, 3\}$, and $L = \{1, 3\}$. This implies that only one or two product types are sold by all local warehouses, which can range in number from one to three, with lead times of either one or three. These lead times apply uniformly to all product types and local warehouses. Consequently, we removed the subscript j from L_j. Although we have defined specific values for I and J in our assumptions, we use the notation I and J in a more general sense. In the end, the size of the MDP state vector reduces to $I + IJ(L + 1)$.

Based on our MDP formulation and exploiting the MINPP problem, we can now define the reward at time t when taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$ as:

$$r_t(\mathbf{s}_t, \mathbf{a}_t) = -\left(\sum_{j=1}^{J}\sum_{i=1}^{I} z_j^i a_{j,t}^i + h_j^i (q_{j,t}^i)^+ + b_j^i (-q_{j,t}^i)^+ + \sum_{i=1}^{I} p_0^i a_{0,t}^i + h_0^i q_{0,t}^i\right). \quad (15)$$

The objective of the MDP is thus to determine an optimal policy that maximizes the sum of expected discounted rewards through a mapping between states and actions. Specifically, our approach employs DRL algorithms trained to minimize the supply chain costs (thereby defining the reward as the negative cost).

System Dynamics

Let us define the demand vector for all product types and each local warehouse j, $j \in \{1, \ldots, J\}$, as:

$$\mathbf{d}_{j,t} := \begin{pmatrix} d_{j,t}^1 \\ d_{j,t}^2 \\ \vdots \\ d_{j,t}^I \end{pmatrix}. \quad (16)$$

Then, as illustrated in Figure 3, at each time step t, $t \in \{1, 2, \ldots, T\}$, a sequence of events characterizing our system dynamics occurs in the following order (Huh et al. 2009; Chao et al. 2015):

(1) New orders, $\mathbf{x}_{j,t-L_j}$, are received and added to the stocks on hand, $\mathbf{q}_{j,t}$, for every warehouse j, $j \in \{0, \ldots, J\}$, and product type i, $i \in \{1, \ldots, I\}$.
(2) Based on the actual state, $\mathbf{s}_t$, ordering decisions, $\mathbf{a}_t$, are made.
(3) Demands, $\mathbf{d}_{j,t}$, are then realized for every local warehouse j, $j \in \{1, \ldots, J\}$, and product type i, $i \in \{1, \ldots, I\}$, and satisfied using the current stock level, accounting for potential backorders.
(4) Cost, r_t, is computed.
(5) The next state, $\mathbf{s}_{t+1}$, containing new stocks on hand and in transit is determined.

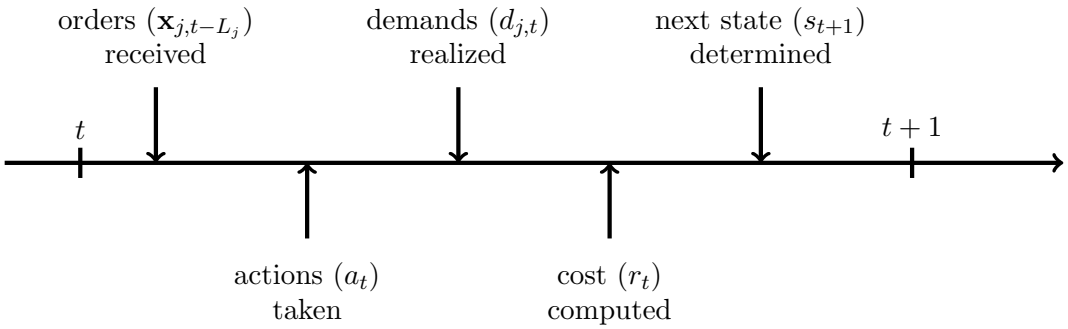


Figure 3. Representation of the dynamics of the inventory control system considered.
Figure 3 Alt Text. Graphical representation of the dynamics within the inventory control system considered.

To determine the next state $\mathbf{s}_{t+1}$, the vector representing the total in-transit orders for each local warehouse is updated by shifting existing values to the right and appending the new ordering decisions $\mathbf{a}_{j,t}$ to the first column on the left:

$$\mathbf{x}_{j,t+1} = \begin{pmatrix} a^1_{j,t} & x^1_{j,t-1} \cdots & x^1_{j,t-L_j+2} \\ a^2_{j,t} & x^2_{j,t-1} \cdots & x^2_{j,t-L_j+2} \\ \vdots & \vdots & \vdots \\ a^I_{j,t} & x^I_{j,t-1} \cdots & x^I_{j,t-L_j+2} \end{pmatrix}. \tag{17}$$

Finally, the stocks dynamics as delineated in Equations (10) and (11) are defined as:

$$\begin{cases} q^i_{0,t+1} = \min\left[\left(q^i_{0,t} + a^i_{0,t} - \sum_{j=1}^{J} a^i_{j,t}\right), c^i_0\right] & \forall\, j \in \{0\}, \forall\, i \in \{1, \ldots, I\} \\ q^i_{j,t+1} = \min\left[\left(q^i_j + x^i_{j,t-L_j} - d^i_{j,t}\right), c^i_j\right] & \forall\, j \in \{1, \ldots, J\}, \forall\, i \in \{1, \ldots, I\}. \end{cases} \tag{18}$$

This means that, at the beginning of the next time step $t+1$, the central warehouse stock level equals the initial stocks, plus the units produced, minus the units shipped. Similarly, the local warehouses' stock levels are equal to the initial stocks, plus the units received, minus the current demands. To ensure compliance with Equation (8), the system prohibits storing products exceeding storage capacities, directly discarding

any units in excess. Clearly, if the stock level is positive, a storage cost is incurred; otherwise, a backorder cost applies.

3.2. *Deep Reinforcement Learning Formulation*

In this section, we explain how we adapt DRL algorithms to fit the previously described MDP formulation.

State Vector

The state vector defined in Section 3.1 contains the current stocks on hand and in transit for each warehouse and product type. In addition, we also include the demand values from the last τ time steps, defining it as:

$$\left(\mathbf{s}_t, \hat{\mathbf{d}}_{1,t} \ldots, \hat{\mathbf{d}}_{J,t}\right), \tag{19}$$

where:

$$\hat{\mathbf{d}}_{j,t} := \begin{pmatrix} d^1_{j,t-\tau} & d^1_{j,t-\tau+1} \cdots & d^1_{j,t-1} \\ d^2_{j,t-\tau} & d^2_{j,t-\tau+1} \cdots & d^2_{j,t-1} \\ \vdots & \vdots & \vdots \\ d^I_{j,t-\tau} & d^I_{j,t-\tau+1} \cdots & d^I_{j,t-1} \end{pmatrix}. \tag{20}$$

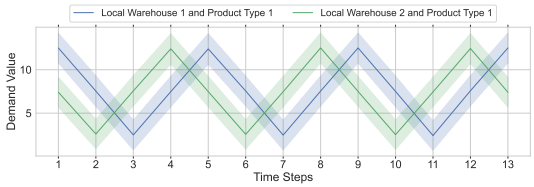
It is crucial to highlight that the size of the newly defined state vector is $I + IJ(L + \tau + 1)$.

To simulate a stochastic and seasonal behavior, drawing inspiration from Kemmer et al. (2018); Peng et al. (2019); Stranieri and Stella (2022), we represent the demand as a sinusoidal function augmented with a stochastic component, as defined by the following equation:

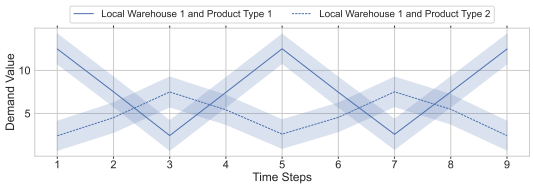
$$d^i_{j,t} = \left\lfloor \frac{d^i_{max}}{2} \left[1 + \sin\left(\frac{\pi(2i + j + t)}{2}\right)\right] + \mathcal{U}\left(0, d^i_{var}\right) \right\rfloor, \tag{21}$$

where $\lfloor \cdot \rfloor$ denotes the floor function, d^i_{max} is the maximum demand value for each product type i, and $\mathcal{U}$ is a uniformly distributed random variable on the support $(0, d^i_{var})$ representing demand variations (i.e., the *uncertainty*). At each time step, the demand can vary for each local warehouse and product type while maintaining the same general behavior, as depicted in Figure 4.

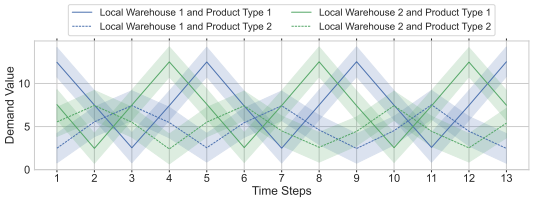
To provide DRL agents with limited knowledge about the demand history, we include the most recent demand values within the state vector. This approach benefits the agent in developing a basic understanding of demand fluctuations. It is important to note that the actual demand d_t will not be included within the state vector $\mathbf{s}_t$ until



(j) $I = 1$ and $J = 2$ with $L = 3$.



(k) $I = 2$ and $J = 1$ with $L = 1$.



(l) $I = 2$ and $J = 2$ with $L = 3$.

Figure 4. Different demands behaviors generated according to Section 3.2 fixing $d^1_{max} = 10$, $d^1_{var} = 5$ and $d^2_{max} = 5$, $d^2_{var} = 10$ for distinct typologies and configurations of the SCIM problem. Specifically, (j) represents one product type and two local warehouses with lead times of three; (k) shows two product types and one local warehouse with a lead time of one; and (l) depicts two product types and two local warehouses with lead times of three.

Figure 4 Alt Text. Set of three plots demonstrating different demand behaviors for the supply chain inventory management problem assumed, generated according to the referenced equation with specific demand parameter values. Figure (a) displays the demand pattern for one product type and two local warehouses with lead times of three. Figure (b) illustrates the demands for two product types and one local warehouse with a lead time of one. Figure (c) presents the demands for two product types and two local warehouses with lead times of three.

the subsequent time step $t+1$. This design choice enables DRL agents to benefit from learning demand patterns, integrating a sort of *demand forecasting* directly into their policy.

Action Vector

Regarding the action vector $\mathbf{a}_t$ represented in Section 3.1, we opted for a *continuous action space*, which means that the ANN generates action values directly, as it scales linearly with the size of the problem. Specifically, each value $\mathbf{a}_{j,t}$ represents the number of units produced at the factory and the number of units shipped to each local warehouse for each product type.

In practical terms, a relatively small and identical upper bound is commonly adopted for all action values to minimize computational effort. However, this method can lead to a significant drop in terms of performance. Indeed, if the upper bound is too small, the agent may choose an inefficient action since the optimal one lies outside the admissible range. On the other hand, if the upper bound is too high, the agent may repeatedly select an incoherent action that falls within the admissible range but exceeds a specified maximum capacity, thus slowing down the training process. Accordingly, our

implementation provides a continuous action space with *independent bounds* for each action value, ensuring a balance between computational efficiency and DRL agents' performance.

Consequently, we set the lower bound for each value at zero, as producing or shipping negative quantities of products would be not allowed. Conversely, the upper bound for each warehouse corresponds to its maximum capacity for each product type. Formally, these bounds are represented as $0 \leq a^i_{j,t} \leq c^i_j$, reflecting our assumption of a specific storage capacity for each warehouse with respect to each product type. Accordingly, we have a clearly defined and coherent action space that adheres to the constraint described in Equation (7). This approach is expected to enhance training times and performance since the action space is bounded (and therefore limited) but contains only coherent actions.

Balanced Allocation Rule

When the stocks on hand in the central warehouse are insufficient to satisfy orders from all local warehouses, the central warehouse must allocate the available stocks according to a specific decision rule. In the work of Kaynov et al. (2024), two different allocation rules for a multi-discrete action space were proposed for selecting feasible actions. The first one is the proportional allocation. However, this does not provide the DRL agents with an incentive during training to determine feasible and effective actions. The second one is the sequential allocation rule, which randomizes the sequence of the local warehouses at every time step and executes the determined actions sequentially. Nevertheless, following this rule, the last local warehouses in the sequence could receive an amount of product close to zero, making it harder to recover from possible backorders.

To address this issue and respect the constraint detailed in Equation (9), we propose a new *balanced allocation rule* for continuous action spaces. This rule ensures a fair distribution of products among the local warehouses while maintaining the stocks on hand in the central warehouse as non-negative. Please note that the proposed allocation rule should also be applicable to multi-discrete action spaces.

As described in Algorithm 1, for each product type, if the total number of products to ship exceeds the available products at the central warehouse, proceed with the following steps:

(1) Randomly select one local warehouse.
(2) If the determined quantity to ship to the selected local warehouse is greater than zero, then decrease this quantity by one.
(3) Repeat steps 1 and 2 until the total number of products to ship reaches the available products at the central warehouse.

This allocation rule guarantees that the central warehouse does not ship more products than those available while maintaining a balanced distribution among local ware-

Algorithm 1 The proposed balanced allocation rule for continuous action spaces.

for all $i \in \{1, \ldots, I\}$ **do**
  if $\sum_{j=1}^{J} a_{j,t}^{i} > q_{0,t}^{i}$ **then**
    while $\sum_{j=1}^{J} a_{j,t}^{i} > q_{0,t}^{i}$ **do**
      $x \leftarrow \mathcal{U}(1, J)$
      if $a_{x,t}^{i} > 0$ **then**
        $a_{x,t}^{i} \leftarrow a_{x,t}^{i} - 1$

houses and preventing the issue of some warehouses receiving close to zero products. It is important to note that DRL agents do not explicitly consider on-hand stocks when selecting an action. However, due to Section 3.1, the system constraint expressed in Equation (8) is satisfied, and producing or shipping more products than those that can be stored leads to a cost, resulting in an *implicit penalty* for the agents. An alternative formulation involves considering the action masking technique to prevent agents from taking invalid actions, as proposed for a discrete action space by Peng et al. (2019).

Reward Function and System Dynamics

The *reward function* for each time step t is detailed in Section 3.1. Also, the state update rule follows the system dynamics of the MDP presented in Section 3.1. It is worth highlighting that demand values within the state vector are also updated, discarding the oldest value and appending the most recent one.

4. Numerical Experiments

In this section, we conduct a numerical analysis to assess the performance of various agents. First, we detail the parameters of the environment, followed by an explanation of the DRL algorithms' hyperparameters and static inventory policies' parameters. Then, we present and discuss the results.

4.1. *Environment Parameters*

We designed and conducted a comprehensive set of *numerical experiments* to compare the performances of DRL algorithms and static inventory policies under two distinct scenarios, specifically with two different values of replenishment lead times (i.e., $L = \{1, 2\}$). To thoroughly assess the adaptability and robustness of DRL algorithms, each scenario encompasses a varying number of product types (i.e., $I = \{1, 2\}$) and local warehouses (i.e., $J = \{1, 2, 3\}$), resulting in a total of 12 experiments. Moreover, each scenario is associated with its specific environment parameters, as detailed in Table 2. It is crucial to highlight that the capacity of the central warehouse is adjusted based on the number of local warehouses to ensure it can fully *absorb* the demand for each

product type. In detail, it is set to 15 when there is one local warehouse, 25 for two local warehouses, and 35 for three, respectively. In the authors' opinion, this experimental setup facilitated an extensive evaluation of DRL algorithms and static inventory policies under different conditions, providing valuable insights into their effectiveness and adaptability.

Parameter	Values
Episode Horizon	{9, 13}
Maximum Demand Value	{10, 5}
Maximum Demand Variation	{5, 10}
Production Cost	1
Transportation Cost	0.05
Storage Capacity Local Warehouses	15
Storage Cost Central Warehouse	0.01
Storage Cost Local Warehouses	0.1
Backorder Cost	{10, 20}

Table 2. The environment parameters associated with the experiments conducted. For episode duration, the first value pertains to the experiments with a lead time of one, while the second is for the experiments with lead times of three. Concerning backorder costs, the two values represent experiments with one and two product types, respectively. Finally, for maximum demand value and variation, the first value corresponds to the first product type, while the second value relates to the second product type.

For the first product type, the demand value is upper-bounded by 10 units. However, at each time step, it can stochastically increase by up to 5 additional units, leading to a maximum potential demand of 15 units. For the second product type, these values are inverted with an upper bound of 5 units and a potential increase of up to 10 units. Each local warehouse has a capacity of 15 units and can store stocks as needed at a unit storage cost of 0.01 per time step. Similarly, the cost of storing each unit at the central warehouse is 0.1 per time step. The cost to produce a single product unit is set at 1. For every time step and every unit of unsatisfied demand, the backorder cost is 10 with one product type and 20 with two. Finally, the transportation cost from the central warehouse to a local one is 0.03 per unit. It is worth noting that we focus on a *symmetrical environment*, which means that all local warehouses share identical values for the specified parameters. This simplification allows for a more precise comparison of the performance of different agents while still maintaining a level of complexity that is representative of real-world supply chain systems.

For each episode, we selected different time horizons, T, based on lead times, L. Specifically, we evaluated seven time steps. To account for the lead times effect, we added L time steps at the beginning of each episode, during which the demand is set to zero, as this demand cannot be satisfied. Similarly, we added L time steps at the end of the episode, in which the value of the actions is set to zero. We made this decision because the corresponding orders for the L time steps after T will never arrive, and the optimal actions would inherently be zero, providing no additional information for

evaluation purposes. This method enables us to consider the lead time effects while ensuring a proper evaluation of the agents.

4.2. *Implemented Algorithms*

After specifying the environment parameters, we implemented the DRL agents using three different state-of-the-art DRL algorithms, namely A3C, PG, and PPO, which were briefly introduced in Section 2. For our implementation, we relied on Ray (Moritz et al. 2018), an open-source Python framework that includes RLib, a scalable RL library, and Tune, a scalable hyperparameter tuning library. A significant advantage of Ray is its native support for the OpenAI Gym APIs (Brockman et al. 2016) that we utilized to develop the *simulator* representative of the environment and used for the agents' training process. We chose the hyperparameters for the DRL algorithms to tune based on the Ray documentation and discussions in the papers of Alves and Mateus (2020); Gijsbrechts et al. (2022); Stranieri and Stella (2022), as reported in Table A1 of the supplementary material.

To evaluate and compare the performance of the DRL algorithms, we implemented two distinct static inventory policies, the well-known base-stock policy (referred to as the BS policy) and the (s, Q)-policy (referred to as the sQ policy). The BS policy requires ordering up to S units whenever the stock level falls below this threshold. In contrast, the sQ policy involves ordering Q units whenever the stock level drops below s.

In our implementation, we opted to make reordering decisions independently, meaning that the policy parameter values (i.e., S_j^i for the BS policy and s_j^i and Q_j^i for the sQ policy) can differ based on the specific warehouse and product type. Despite this flexibility, these policies are still considered *static* because their values remain constant over time. To determine the optimal parameter values that minimize Equation (6), we adopted a data-driven method using Bayesian optimization. This method does not necessitate any assumptions or simplifications. As a result, it can be applied to any supply chain typology or configuration, sharing the same simulator as the DRL algorithms. It is crucial to note that our experimental setup employs data-driven approaches that interact with the simulator under the assumption that they have no access to *additional information*, such as forecasts.

We also implemented an additional model called EVP (an acronym that means "expected value problem"). In the EVP model, the demand variables $d_{j,t}^i$ in Equation (6) are replaced at each time step t with the *expected demand value* for each product type i and local warehouse j. To compare DRL agents with static inventory policies, we finally implemented an oracle, which consists of a baseline that knows at each time step t the *exact demand value* $d_{j,t}^i$ in Equation (6) for each product type i and local warehouse j. As a result, it can select the optimal actions to take a priori (for this reason, we refer to this model as PI, which stands for "perfect information"). It is

important to note that the formulation for the EVP and PI models remains the same as expressed in Equations (6) to (14). However, when the EVP agent takes an action and solves the problem, it replaces the demand at time step t with its expected value (rounded up) over the episode horizon T. In contrast, in the PI model, the agent possesses additional information and knows the exact demand realization at time step t, which leads it to take optimal actions at each time step t.

4.3. *Results*

To compare the performance of different agents (i.e., DRL algorithms and static inventory policies), we simulated 1000 testing episodes for each experiment, reporting the expected value of perfect information (*EVPI gap*) achieved (calculated as the average of the difference between the profits of a specific agent and the profits of the oracle, divided by the profits of the oracle). This comparison highlights the effectiveness of the agents as opposed to an *optimal solution*, which remains not feasible in real-world deployments because the PI model assumes complete knowledge of future demand realizations.

All experiments were run on a machine equipped with an Apple M2 Pro chip with 10 cores and 16 GB of RAM, ensuring consistent computing resources across all tests. By evaluating the performance of DRL algorithms and static inventory policies under diverse conditions, we can gain valuable insights into their effectiveness, adaptability, and potential for application in different supply chain typologies and configurations.

One Product Type

The results of numerical experiments under one product type are summarized in Figure 5.

In the first experiment conducted within the context of a single local warehouse with lead times of one, the PPO and A3C algorithms significantly outperform other agents, with the BS policy performing poorly even when compared to the EVP model. When the lead times increase to three, the performance of A3C and the sQ policy become more balanced, while PPO continues to demonstrate superior results, and both the BS policy, PG, and EVP underperform.

With two local warehouses and lead times of one, PPO still performs better than all other agents, while the performance of the sQ policy falls between that of the A3C and PG algorithms. Meanwhile, the BS policy performs worse than EVP. Extending lead times to three further accentuates the excellent performance of PPO, with the sQ policy following closely behind, while the BS policy slightly surpasses both A3C and PG. In this experiment, static inventory policies tend to perform marginally better than A3C and PG, with the former having an advantage over the latter.

Upon introducing a third local warehouse to the experimental setup, PPO consistently maintains its superior level of performance compared to other agents, albeit in

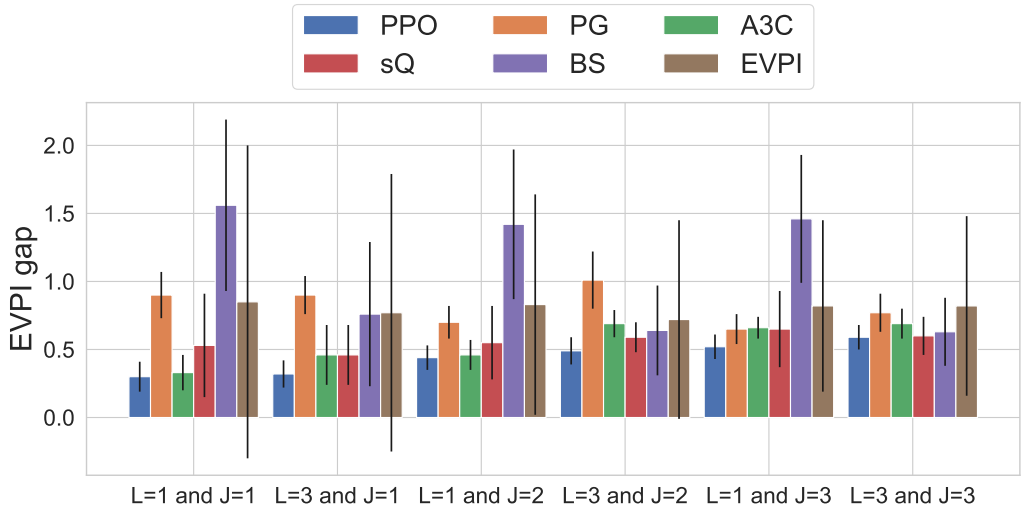


Figure 5. EVPI gap, grouped by experiment, for the experimental setup with one product type. For each number of local warehouses and length of lead times, the figure compares the performance of DRL algorithms, static inventory policies, and EVP. The lower the gap, the better the agent.

Figure 5 Alt Text. Bar chart comparing the expected value of perfect information gap for different agents, grouped by experiment, within a setup involving one product type. The chart shows bars for each combination of the number of local warehouses and length of lead times. Each bar represents the gap of deep reinforcement learning algorithms, static inventory policies, and the expected value problem. A lower bar indicates a smaller gap and, thus, better performance.

a less marked manner. With $L = 1$, the sQ policy achieves results comparable to PG and A3C, while the BS policy continues to perform behind EVP. Conversely, when $L = 3$, static inventory policies slightly outperform both A3C and PG.

Two Product Types

Figure 6 summarizes the results under two product types.

In the experiment with a single local warehouse and $L = 1$, the PPO algorithm demonstrates a clear advantage over other agents. A3C and PG both surpass the BS policy, whereas the sQ policy performance closely aligns with PPO. When $L = 3$, the overall relative performance essentially persists, although the BS policy now outperforms the EVP agent.

When the experimental setup includes two local warehouses and lead times of one, PPO still maintains its advantage over others, followed by the sQ policy, while A3C and PG performance become more comparable in this experiment. Extending the lead times to three reveals a substantial equilibrium in performance across all agents except for PG, which slightly underperforms.

Lastly, in scenarios with three local warehouses, DRL algorithms surpass the sQ policy when $L = 1$, although the performance gap with PPO remains consistent, and

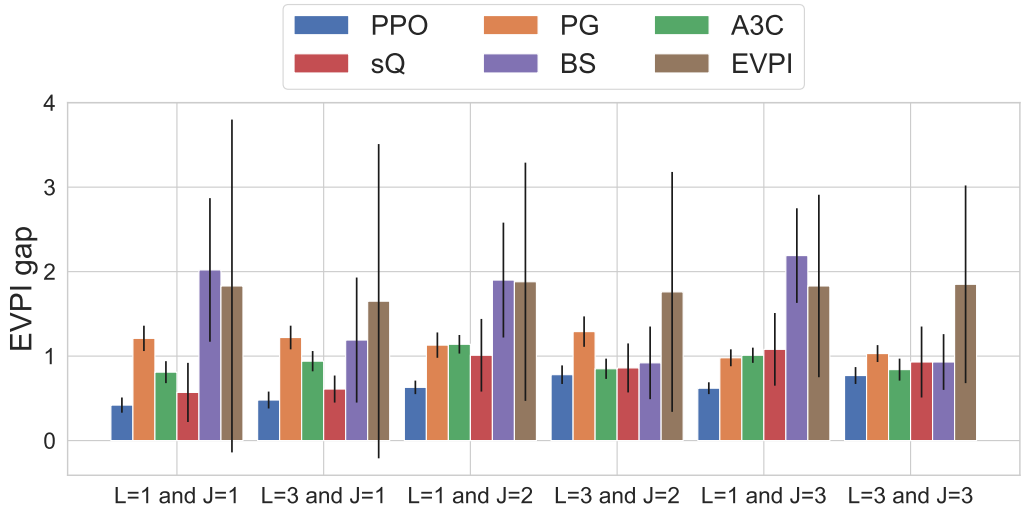


Figure 6. EVPI gap, grouped by experiment, for the experimental setup with two product types. For each number of local warehouses and length of lead times, the figure compares the performance of DRL algorithms, static inventory policies, and EVP. The lower the gap, the better the agent.

Figure 6 Alt Text. Bar chart comparing the expected value of perfect information gap for different agents, grouped by experiment, within a setup involving two product types. The chart shows bars for each combination of the number of local warehouses and length of lead times. Each bar represents the gap of deep reinforcement learning algorithms, static inventory policies, and the expected value problem. A lower bar indicates a smaller gap and, thus, better performance.

the BS policy encounters difficulty. When $L = 3$, static inventory policies achieve results comparable to DRL algorithms, with PPO still outperforming others.

5. Discussion and Insights

In summary, the effectiveness of various DRL algorithms and static inventory policies in SCIM depends on factors such as the number of product types and local warehouses and the length of lead times. PPO consistently emerges as the *best-performing algorithm* across all conducted experiments, offering a robust approach for addressing the SCIM problem within the designed experimental setup. Although the sQ policy does not achieve the same level of performance as PPO, it still performs commendably and appears to be the *second-best choice* among the evaluated algorithms.

Specifically, for a single product type and lead times set to one, the A3C algorithm is preferable to PG, particularly in experiments with one and two local warehouses. Meanwhile, the sQ policy shows superior results compared to the BS policy, which consistently underperforms. Conversely, by extending lead times to three, performance appears to balance as the number of local warehouses increases.

In experiments involving two product types, the performance gap between various

agents remains consistent. Typically, when reducing the number of local warehouses and decreasing lead times, A3C – and especially PG – tend to perform worse than the sQ policy, with the BS policy showing markedly poor performance with lead times set to one. Contrarily, as the number of local warehouses and lead times increase, static inventory policies gradually align with the results of DRL algorithms.

Moreover, DRL algorithms consistently and significantly outperform the EVP model, suggesting that the complexities of the problem cannot be adequately captured by merely considering the average demand value. More robust strategies are thus essential in these scenarios. These findings underscore the articulate interplay between the number of product types and local warehouses, the length of lead times, and the relative performance of DRL algorithms versus static inventory policies. Decision-makers should carefully consider these factors when choosing an *appropriate agent* that balances performance and computational efficiency.

It is crucial to emphasize that all agents reach *convergence* in each experiment conducted, as documented in Figure A1 of the supplementary material. Specifically, PG tends to converge faster than other DRL agents, leading to reduced training times. In contrast, the A3C algorithm usually takes longer to converge, resulting in extended training durations, while PPO holds an intermediate position in terms of convergence and training times.

In the context of static inventory policies, the sQ policy requires more training time to converge than the BS policy. This difference can be attributed to the increased number of parameters to optimize inherent to the sQ policy. Despite the extended training duration, the sQ policy generally performs better, highlighting the trade-offs that decision-makers need to consider when selecting an appropriate agent based on factors such as *convergence, training time, and performance*.

Ultimately, the training times between DRL algorithms and static inventory policies are reasonably similar. For completeness, Table A2 of the supplementary material provides the *average training times* for each agent.

5.1. *Conclusion and Future Research*

In this study, we presented an MDP formulation for a two-echelon inventory control system, demonstrating that the PPO algorithm consistently arises as the best-performing algorithm across various scenarios. While there is a trade-off between implementation complexity and performance, its results remain consistent across all the experiments we analyzed. Additionally, we highlighted the sQ policy as a robust alternative to DRL algorithms such as A3C and PG, offering a balance between performance and computational efficiency.

Building on these findings, several potential avenues for *future research* can be explored. For example, our system could benefit from incorporating alternative allocation rules for managing the on-hand stocks of the central warehouse. While we have intro-

duced a balanced rule, we have not compared it to others, such as a priority rule that gives priority to local warehouses with the highest backorder cost in an asymmetrical environment. We reserve this topic for further investigation in future studies.

Another two areas for exploration in the context of multiple product types are the substitution effect and perishable products. Currently, there is limited literature on multi-echelon inventory control systems that incorporate these specific assumptions, and expanding research in this direction could significantly enhance our understanding of SCIM strategies in more challenging and practical environments.

Lastly, investigating machine learning techniques for demand forecasting presents a promising avenue. The idea is to identify the best technique through a benchmark and then use it to evaluate dynamic inventory policies against static ones. Moreover, such a forecast could be integrated into the state vector of the DRL agents. While preliminary studies have investigated this approach (Dehaybe et al. 2023), a comparative analysis among different DRL algorithms has not yet been addressed and represents a promising focus for future research.

Data Availability Statement

The datasets generated and analyzed during the current study can be generated using the open-source software library developed for this research, which can be accessed at `https://github.com/frenkowski/SCIMAI-Gym`.

Disclosure Statement

The authors report there are no competing interests to declare.

Funding

This research received no external funding.

Notes on contributors

Francesco Stranieri is a Ph.D. student enrolled in the Italian National Ph.D. program in Artificial Intelligence (Industry 4.0 area) at the Polytechnic of Turin and the University of Milano-Bicocca. He obtained his Bachelor's and Master's degrees in Computer Science from the University of Milano-Bicocca in 2019 and 2021, respectively. He won the 2nd prize in the "Green Technologies and Applications" line in the Best Student Research "Video & Poster Competition" at the IEEE EUROCON 2023 Conference. In 2023, he completed a six-month internship at Bristol Myers Squibb in

Switzerland, focusing on deep reinforcement learning and digital twin for supply chain operations and risk assessment. He is an active member of several associations, including the Italian Association of Operations Research (AIRO), the Italian Association for Artificial Intelligence (AIxIA), and IEEE Eta Kappa Nu (IEEE-HKN).

Fabio Stella, Ph.D. in Computational Mathematics and Operations Research, is an Associate Professor of Operations Research at the Department of Informatics, Systems, and Communication of the University of Milano-Bicocca. His main research interests are artificial intelligence (AI) and machine learning (ML). Fabio has published more than 100 papers and served as Program Chair/Reviewer for top international conferences on AI and ML, including AISTATS, ECAI, ICLR, ICML, IJCAI, NeurIPS, PAKDD, PGM, RecSys, SIGIR, and UAI. He has been awarded as a top 10% reviewer at NeurIPS in 2020, 2022, and 2023, ICML and AISTATS in 2022, and IJCAI in 2023. He has been Associate Editor of IEEE Intelligent Systems since 2021 and is currently leading two international research projects funded through a competitive process by the European Commission.

Chaaben Kouki is a Professor of Operations Management at the ESSCA School of Management in Angers, France. He received his Ph.D. and Master's degrees in Industrial Engineering from Ecole Centrale Paris in 2010 and 2007, respectively, and his Bachelor's degree in Industrial Engineering from Ecole Nationale d'Ingénieurs de Tunis – ENIT in 2005. Before joining ESSCA School of Management, he was a faculty member at Rennes School of Business from 2012 to 2015 and an inventory manager at Michelin Group from 2010 to 2012. His primary research focus is on inventory management systems.

References

J. C. Alves and G. R. Mateus. Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. In *Lecture Notes in Computer Science*, pages 584–599. Springer International Publishing, 2020. . URL https://doi.org/10.1007/978-3-030-59747-4_38.

E. Bakshy, L. Dworkin, B. Karrer, K. Kashin, B. Letham, A. Murthy, and S. Singh. Ae: A domain-agnostic platform for adaptive experimentation. In *Conference on Neural Information Processing Systems*, pages 1–8, 2018.

R. N. Boute, J. Gijsbrechts, W. van Jaarsveld, and N. Vanvuchelen. Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*, 298 (2):401–412, Apr. 2022. . URL https://doi.org/10.1016/j.ejor.2021.07.016.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

S. K. Chaharsooghi, J. Heydari, and S. H. Zegordi. A reinforcement learning model for supply chain ordering management: An application to the beer game. *Decision Support Systems*, 45(4):949–959, Nov. 2008. . URL https://doi.org/10.1016/j.dss.2008.03.007.

X. Chao, X. Gong, C. Shi, and H. Zhang. Approximation algorithms for perishable inventory systems. *Operations Research*, 63(3):585–601, June 2015. . URL https://doi.org/10.1287/opre.2015.1386.

T. de Kok, C. Grob, M. Laumanns, S. Minner, J. Rambau, and K. Schade. A typology and literature review on stochastic multi-echelon inventory models. *European Journal of Operational Research*, 269(3):955–983, Sept. 2018. . URL https://doi.org/10.1016/j.ejor.2018.02.047.

H. Dehaybe, D. Catanzaro, and P. Chevalier. Deep reinforcement learning for inventory optimization with non-stationary uncertain demand. *European Journal of Operational Research*, Oct. 2023. . URL https://doi.org/10.1016/j.ejor.2023.10.007.

M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer International Publishing, 2019. . URL https://doi.org/10.1007/978-3-030-05318-5_1.

V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4): 219–354, 2018. . URL https://doi.org/10.1561/2200000071.

K. Geevers, L. van Hezewijk, and M. R. K. Mes. Multi-echelon inventory optimization using deep reinforcement learning. *Central European Journal of Operations Research*, July 2023. ISSN 1613-9178. . URL http://dx.doi.org/10.1007/s10100-023-00872-2.

J. Gijsbrechts, R. N. Boute, J. A. V. Mieghem, and D. J. Zhang. Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. *Manufacturing & Service Operations Management*, 24(3):1349–1368, May 2022. . URL https://doi.org/10.1287/msom.2021.1064.

G. J. Gordon and T. M. Mitchell. Approximate solutions to markov decision processes. 1999.

C. D. Hubbs, H. D. Perez, O. Sarwar, N. V. Sahinidis, I. E. Grossmann, and J. M. Wassick. Or-gym: A reinforcement learning library for operations research problems, 2020. URL https://arxiv.org/abs/2008.06319.

W. T. Huh, G. Janakiraman, J. A. Muckstadt, and P. Rusmevichientong. Asymptotic opti-

mality of order-up-to policies in lost sales inventory systems. *Management Science*, 55(3): 404–420, Mar. 2009. . URL https://doi.org/10.1287/mnsc.1080.0945.

T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, Nov. 1994. . URL https://doi.org/10.1162/neco.1994.6.6.1185.

I. Jackson, M. Jesus Saenz, and D. Ivanov. From natural language to simulations: applying ai to automate simulation modelling of logistics systems. *International Journal of Production Research*, page 1–24, Nov. 2023. ISSN 1366-588X. . URL http://dx.doi.org/10.1080/00207543.2023.2276811.

I. Kaynov, M. van Knippenberg, V. Menkovski, A. van Breemen, and W. van Jaarsveld. Deep reinforcement learning for one-warehouse multi-retailer inventory management. *International Journal of Production Economics*, 267:109088, Jan. 2024. ISSN 0925-5273. . URL http://dx.doi.org/10.1016/j.ijpe.2023.109088.

L. Kemmer, H. von Kleist, D. de Rochebouët, N. Tziortziotis, and J. Read. Reinforcement learning for supply chain optimization. In *European Workshop on Reinforcement Learning*, volume 14, 2018.

X. Liu and Y. L. Tu. Capacitated production planning with outsourcing in an OKP company. *International Journal of Production Research*, 46(20):5781–5795, Jan. 2008. . URL https://doi.org/10.1080/00207540701348779.

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

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1928–1937. JMLR.org, 2016.

P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, Carlsbad, CA, Oct. 2018. USENIX Association. ISBN 978-1-939133-08-3. URL https://www.usenix.org/conference/osdi18/presentation/moritz.

A. Oroojlooyjadid, M. Nazari, L. V. Snyder, and M. Takáč. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 24(1):285–304, Jan. 2022. . URL https://doi.org/10.1287/msom.2020.0939.

J. Parker-Holder, V. Nguyen, and S. J. Roberts. Provably efficient online hyperparameter optimization with population-based bandits. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17200–17211. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c7af0926b294e47e52e46cfebe173f20-Paper.pdf.

Z. Peng, Y. Zhang, Y. Feng, T. Zhang, Z. Wu, and H. Su. Deep reinforcement learning approach for capacitated supply chain optimization under demand uncertainty. In *2019*

Chinese Automation Congress (CAC). IEEE, Nov. 2019. . URL https://doi.org/10.1109/cac48633.2019.8997498.

S. Punia, K. Nikolopoulos, S. P. Singh, J. K. Madaan, and K. Litsiou. Deep learning with long short-term memory networks and random forests for demand forecasting in multi-channel retail. *International Journal of Production Research*, 58(16):4964–4979, Mar. 2020. ISSN 1366-588X. . URL http://dx.doi.org/10.1080/00207543.2020.1735666.

B. Rolf, I. Jackson, M. Müller, S. Lang, T. Reggelin, and D. Ivanov. A review on reinforcement learning algorithms and applications in supply chain management. *International Journal of Production Research*, 61(20):7151–7179, Nov. 2022. ISSN 1366-588X. . URL http://dx.doi.org/10.1080/00207543.2022.2140221.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schulman15.html.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

M. Shajalal, P. Hajek, and M. Z. Abedin. Product backorder prediction using deep neural network on imbalanced data. *International Journal of Production Research*, 61(1):302–319, Mar. 2021. ISSN 1366-588X. . URL http://dx.doi.org/10.1080/00207543.2021.1901153.

R. Sharma, A. Shishodia, A. Gunasekaran, H. Min, and Z. H. Munim. The role of artificial intelligence in supply chain management: mapping the territory. *International Journal of Production Research*, 60(24):7527–7550, Feb. 2022. ISSN 1366-588X. . URL http://dx.doi.org/10.1080/00207543.2022.2029611.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct. 2017. . URL https://doi.org/10.1038/nature24270.

F. Stranieri and F. Stella. A deep reinforcement learning approach to supply chain inventory management, 2022. URL https://arxiv.org/abs/2204.09603.

F. Stranieri, E. Fadda, and F. Stella. Combining deep reinforcement learning and multi-stage stochastic programming to address the supply chain inventory management problem. *International Journal of Production Economics*, 268:109099, Feb. 2024. ISSN 0925-5273. . URL http://dx.doi.org/10.1016/j.ijpe.2023.109099.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

L. van Hezewijk, N. Dellaert, T. Van Woensel, and N. Gademann. Using the proximal policy optimisation algorithm for solving the stochastic capacitated lot sizing problem. *International Journal of Production Research*, 61(6):1955–1978, Apr. 2022. ISSN 1366-588X. . URL http://dx.doi.org/10.1080/00207543.2022.2056540.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff,

Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Oct. 2019. . URL https://doi.org/10.1038/s41586-019-1724-z.

H. Wang, J. Tao, T. Peng, A. Brintrup, E. E. Kosasih, Y. Lu, R. Tang, and L. Hu. Dynamic inventory replenishment strategy for aerospace manufacturing supply chain: combining reinforcement learning and multi-agent simulation. *International Journal of Production Research*, 60(13):4117–4136, Jan. 2022. ISSN 1366-588X. . URL http://dx.doi.org/10.1080/00207543.2021.2020927.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, May 1992. . URL https://doi.org/10.1007/bf00992696.

C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. M. Bayen, S. Kakade, I. Mordatch, and P. Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines, 2018. URL https://arxiv.org/abs/1803.07246.

Y. Yan, A. H. Chow, C. P. Ho, Y.-H. Kuo, Q. Wu, and C. Ying. Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review*, 162: 102712, June 2022. . URL https://doi.org/10.1016/j.tre.2022.102712.

L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, Nov. 2020. . URL https://doi.org/10.1016/j.neucom.2020.07.061.

Appendix A. Supplementary Material

A.1. *Hyperparameters Selection*

The process of selecting appropriate values for hyperparameters in ANNs is widely acknowledged to be complex and time-consuming, as extensively discussed in the relevant literature (Feurer and Hutter 2019; Yang and Shami 2020). Moreover, hyperparameters play a crucial role in the context of DRL algorithms since they can significantly influence training, convergence, and, consequently, their relative performance (Boute et al. 2022).

In our experimental setup, each DRL agent was trained for a specific number of episodes, which varied depending on the number of local warehouses.

For one product type:

- 25000 episodes with 1 and 2 local warehouses.
- 50000 episodes with 3 local warehouses.

For two product types:

- 50000 episodes with 1 and 2 local warehouses.
- 75000 episodes with 3 local warehouses.

Using the Population-Based Bandit (PB2) algorithm (Parker-Holder et al. 2020), the DRL agents determined the best hyperparameter values from those we selected, which are presented in Table A1. PB2 adapts hyperparameters during training, leading to improved performance and convergence of the agents. In our implementation, DRL agents accessed demand values from the preceding two time steps (i.e., $\tau = 2$). However, preliminary results suggest that comparable performances can be achieved when agents access demand values from the last or the last three time steps.

Hyperparameter	Values		
	PPO	PG	A3C
Learning Rate	{7e-4, 1e-5, 3e-5, 5e-5, 7e-5}	{1e-5, 3e-5, 7e-5, 1e-6}	{1e-5, 3e-5, 5e-5, 7e-5}
Neurons per Hidden Layer	{(32, 32), (64, 64), (128, 128)}	{(32, 32), (64, 64), (128, 128)}	{(32, 32), (64, 64), (128, 128)}
Gamma	0.99	0.99	0.99
Train Batch Size	{2000, 4000, 8000}	200, 600, 800}	{200, 800, 1400}
Gradient Clipping	-	-	{20, 40, 60}
Stochastic Gradient Descent Iterations	{5, 15, 25}	-	-
Stochastic Gradient Descent Mini-Batch Size	{128, 256, 512}	-	-

Table A1. The selected hyperparameters for the DRL algorithms. For hyperparameters not included in the table, we used the default values provided by Ray.

To identify the optimal static inventory policy parameters – S_j^i for the BS policy and s_j^i and Q_j^i for the sQ policy – we employed a data-driven method using Bayesian optimization via Ax (Bakshy et al. 2018), an open-source Python platform designed for optimizing simulation experiments. As delineated in Section 3.2 of the original manuscript, we opted to make reordering decisions independently, and as for the DRL algorithms, we designated the same lower and upper bound for each parameter.

In our experimental setup, each static inventory policy was optimized for a specific number of training iterations based on the number of local warehouses.

For one product type:

- 50 training iterations with 1 local warehouse.
- 75 training iterations with 2 local warehouses.
- 100 training iterations with 3 local warehouses.

For two product types:

- 75 training iterations with 1 local warehouse.
- 100 training iterations with 2 local warehouses.
- 125 training iterations with 3 local warehouses.

By adjusting the number of episodes and training iterations according to the size and complexity of the experiment, we ensure that each agent strikes a balance between computational resources and time. This approach promotes accurate and reliable results while guaranteeing consistent convergence for each agent, as shown in Figure A1.

A.2. *Training Times*

Table A2 presents the training times for the various agents implemented, calculated as averages across the two scenarios considered (i.e., with $L = 1$ and $L = 3$) for each combination between the number of product types and local warehouses. In particular, PG typically converges more rapidly than A3C and PPO, leading to shorter training durations. When examining static inventory policies, the (s, Q) policy usually takes a longer time to converge compared to the BS policy, primarily due to the increased number of parameters that need optimization. Ultimately, the training durations for DRL algorithms and static inventory policies are reasonably comparable.

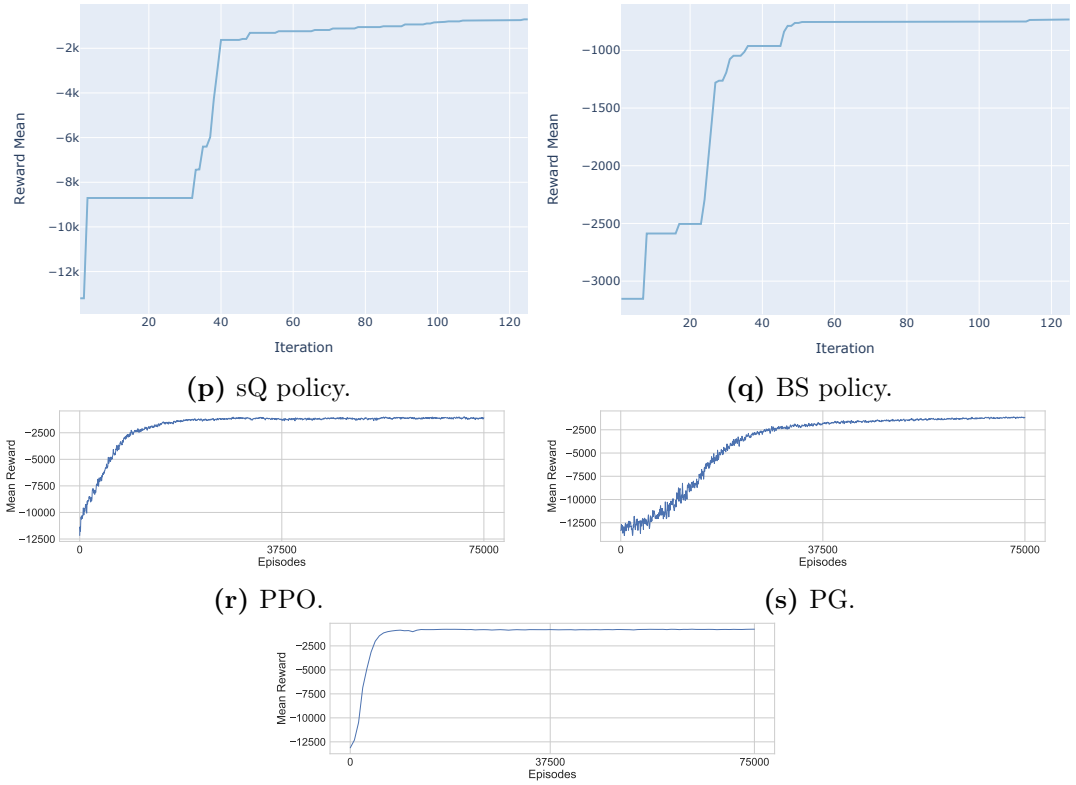
	PPO	PG	A3C	sQ	BS
$J = 1$	1.27 ± 0.18	1.18 ± 0.18	1.57 ± 0.46	1.06 ± 0.65	0.49 ± 0.10
$J = 2$	1.29 ± 0.17	1.16 ± 0.16	1.40 ± 0.17	1.78 ± 0.38	1.29 ± 0.17
$J = 3$	3.01 ± 0.55	2.70 ± 0.49	3.73 ± 0.59	3.30 ± 0.74	1.88 ± 0.50

(a) $I = 1$.

	PPO	PG	A3C	sQ	BS
$J = 1$	2.87 ± 0.59	2.52 ± 0.50	3.27 ± 0.79	1.79 ± 0.36	1.41 ± 0.10
$J = 2$	3.50 ± 0.50	2.83 ± 0.48	3.96 ± 0.63	2.68 ± 0.36	2.01 ± 0.43
$J = 3$	5.89 ± 1.47	5.31 ± 0.87	7.49 ± 2.03	3.71 ± 0.47	3.02 ± 0.48

(b) $I = 2$.

Table A2. The average training times (in minutes) for DRL algorithms and static inventory policies across the two values of lead times considered for one product type (Table A2a) and two product types (Table A2b) and for each number of local warehouses.



(p) sQ policy.

(q) BS policy.

(r) PPO.

(s) PG.

(t) A3C.

Figure A1. Convergence trajectories during training of sQ policy (Figure A1p), BS policy (Figure A1q), PPO (Figure A1r), PG (Figure A1s), and A3C (Figure A1t) in the most challenging experiment, which involves two product types, three local warehouses, and lead times of three. Plots related to other experiments conducted can be found in the GitHub repository associated with our open-source library (available at `https://github.com/frenkowski/SCIMAI-Gym`).

Figure A1 Alt Text. Line graphs showing the convergence trajectories for different agents during the most challenging experimental setup with two product types, three local warehouses, and lead times of three. The plots include the sQ policy, the BS policy, the PPO algorithm, the PG algorithm, and the A3C algorithm. Each line represents the learning progression over time, with convergence indicating the optimization of the agent's performance. Additional plots for other experiments can be found in the associated GitHub repository for our open-source library.