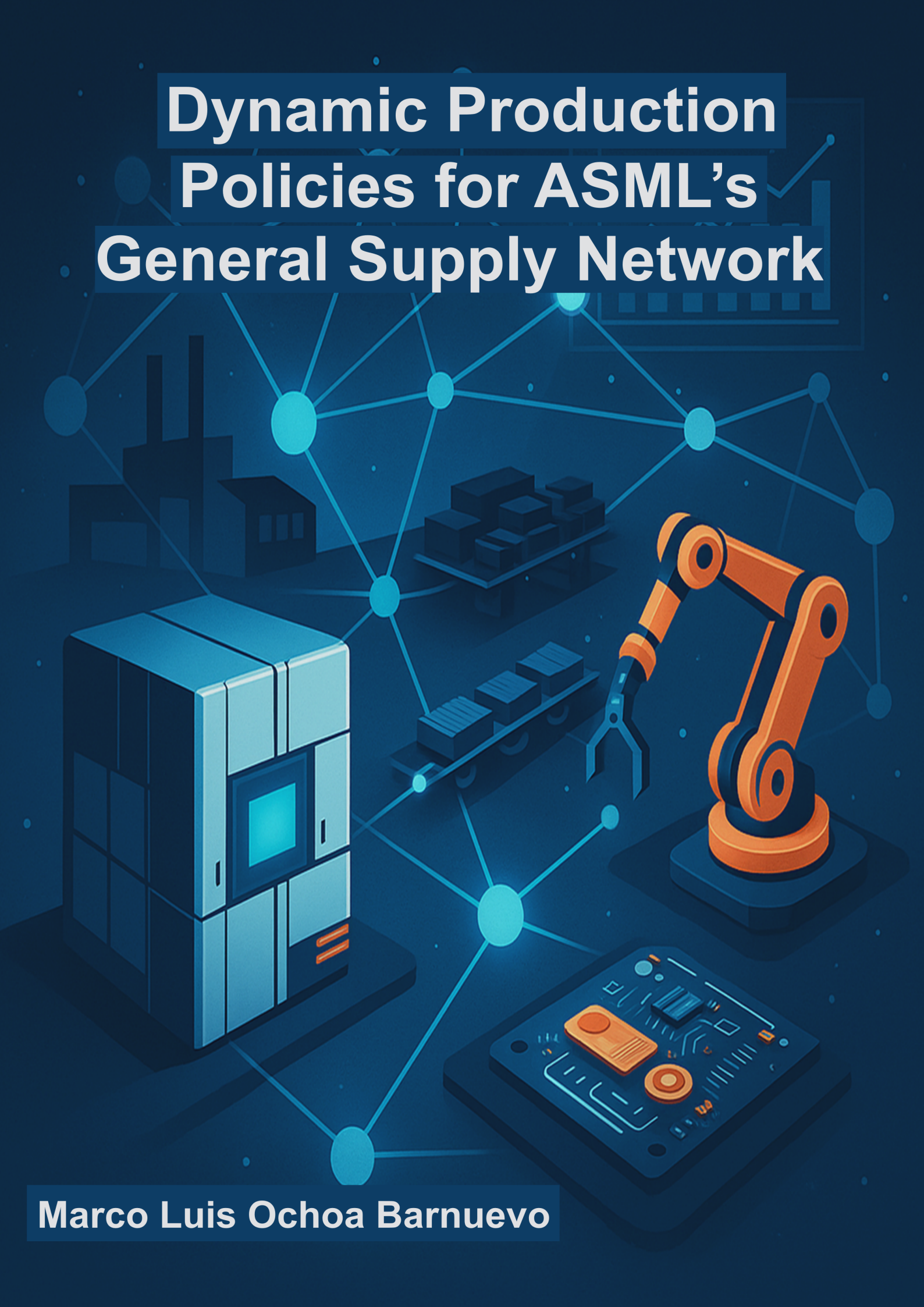
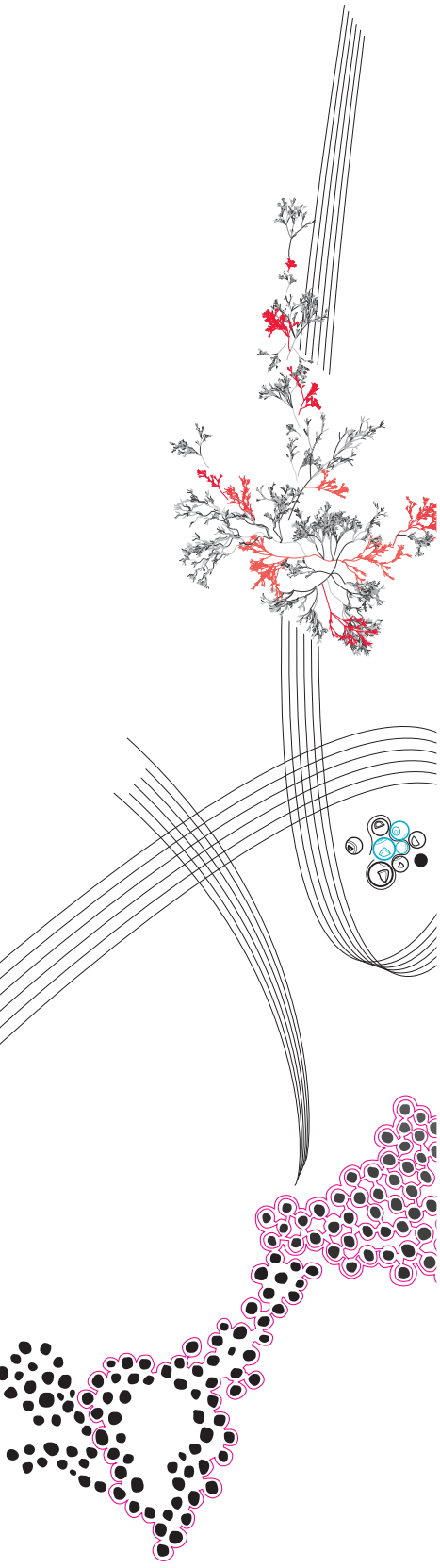
Dynamic Production Policies for ASML's General Supply Network



Marco Luis Ochoa Barnuevo



MSc Industrial Engineering and Management
Final Project

Dynamic Production Policies for ASML's General Supply Network

Marco Luis Ochoa Barnuevo

University of Twente (UT) Supervisors:
prof.dr.ir. MRK. (Martijn) Mes
dr. DRJ. (Dennis) Prak

External Supervisors:
ASML:
MSc. T. (Tjum) van Dijck
dr. T. (Tijn) Fleuren
Eindhoven University of Technology (TU/e):
dr. W. (Willem) van Jaarsveld

June, 2025

Department of Industrial Engineering and Business Information Systems (IEBIS)
Faculty of Behavioural Management and Social Sciences (BMS)
Industrial Engineering and Management (IEM)
University of Twente (UT)

UNIVERSITY OF TWENTE.

Acknowledgements

This thesis marks a significant chapter in my life, enriched by incredible people, memorable experiences, and valuable learning opportunities. I extend my heartfelt gratitude to the University of Twente for awarding me the UT Scholarship, which made this master's journey possible. Special thanks to Marco Schutten, former Director of the IEM program, for kindly providing a recommendation letter, and Ellen van Zeijts and Annemieke van der Grijspaarde for their invaluable support and assistance in navigating the scholarship process.

I am deeply grateful to (and will miss) all the professors at UT, whose courses sparked my passion for Operations Research and significantly developed my skills. Their openness to discussions and willingness to clarify concepts made my learning experience enjoyable and productive. Also, a special thanks to all the PhD candidates who made me feel welcome in their workspace and were always up for a good conversation.

My sincere appreciation goes to my outstanding team of supervisors. I especially thank Tjum van Dijck, my supervisor at ASML, whose constant support, valuable brainstorming sessions, and critical insights greatly shaped this thesis. Your guidance was essential in transforming my ideas into reality. I will genuinely miss you and our insightful discussions. Special thanks to Fabian Akkerman for his instrumental support, from helping me connect with ASML through Willem van Jaarsveld, facilitating my experiments on the Snellius supercomputer, and inviting me to present this work at the TKI Dinalog conference.

My sincere thanks also go to Martijn Mes, whose mentorship and support profoundly impacted my learning journey at UT and this thesis. Your passion and expertise are truly inspiring. Dennis Prak, thank you for courageously guiding me through this complex project; your insightful critiques helped sharpen my methods. To Tijn Fleuren and Willem van Jaarsveld, my second ASML supervisor and external supervisor from TU/e respectively, thank you for your impactful feedback that sparked crucial improvements in my thesis. Finally, I would like to acknowledge the wonderful Strategy and Improvement Team at ASML for making me feel truly welcomed as part of the team. In particular, I am grateful to Maarten Hendriks, whose guidance and example left a lasting impression on me.

To my dearest friends, thank you for your support, encouragement, and understanding throughout this journey. Your presence has been a source of motivation and encouragement, and it has made this journey more enjoyable and memorable.

Lastly, but most importantly, I want to express my deepest gratitude to my family, especially to my mother and my girlfriend. Their unconditional love and belief in my abilities have been the driving force behind my success. Their sacrifices, patience, and encouragement have provided me with the strength and determination to overcome obstacles and pursue my goals and dreams.

I hope this acknowledgment adequately expresses my gratitude to everyone who contributed to this journey. This achievement belongs as much to you as it does to me. Thank you sincerely.

Management Summary

This thesis addresses the complex challenge of inventory management in multi-echelon supply networks characterized by low-volume, high-mix demand, divergent and convergent (or general) material flows, and shared capacity constraints, features emblematic of ASML's high-tech assembly environment. Traditional analytical models and heuristics fall short in this setting: serial surrogate methods oversimplify topologies, Guaranteed-Service assumptions are unrealistic in volatile demand environments, and existing allocation rules (e.g., FCFS or fixed reservations) cannot adapt to asymmetric lead times or capacity constraints. Moreover, prior DRL studies typically assume continuous actions or ignore shared capacities, limiting their applicability to real-world, discrete-action systems.

To bridge these gaps, we propose an interpretable and efficient capacity-aware echelon base-stock policy and a DRL-based approach:

1. **Network-Compatible Base-Stock Sizing (G-MATCH)**: Extends convergent-only methods to convergent-divergent networks and reduces cost by roughly 90% compared to earlier base-stock extensions in uncapacitated experiments.

2. **Capacity-Aware Guided Base-Stock Optimization (GBS)**: Introduces a Capacity Constraint Ratio to iteratively raise static base-stock levels at the most constrained nodes until no further cost improvement is possible. In capacitated systems, GBS yields an additional 5–8% cost reduction.

3. **Enhanced Allocation Rules.** We propose a family of "water-filling"-style allocation policies tailored to convergent–divergent network nodes. While standard Water-Filling (WF) greedily allocates all available units to the highest shortfall, a "reservations" variant (WFR) withholds a fixed buffer to address future risk. Our experiments show that both WF and WFR achieve over 6% cost reduction compared to commonly used FCFS and that WF's straightforward, shortage-driven logic often outperforms more reservation schemes.

4. **Deep Controlled Learning (DCL).** We embed a lightweight, discrete-action DRL layer on top of GBS+WF, which sequentially refines ordering/reservation decisions per node each period using real-time pipeline and capacity data. DCL learns state-dependent buffers (e.g., prioritizing high-penalty end items or rerouting and balancing around bottlenecks) while respecting discrete orders and shared-capacity constraints. Experiments show DCL consistently adds further savings (up to 2–5% under tighter capacity in smaller networks; 1–2% under moderate or uncapacitated settings), but gains shrink below 1% with larger/deeper networks. Training is computationally intensive (12 minutes on HPC or 8+ hours on standard hardware for a 7-node case) and offers limited interpretability; however, behavioral insights can help enhance heuristic methods.

Extensive experiments allow to see the following managerial implications.

- *Heuristic "Quick Win":* A single run of capacity-aware base-stock optimization (GBS) plus Water-Filling allocation delivers 90–95 % of achievable cost savings with minimal effort and full traceability. For instance, a 7-node, 1,000-period simulation completes in under 30 s, making periodic offline tuning trivial.

- *Selective DRL Deployment:* When capacity groups retain slack (e.g., moderate overcapacity) or lead-time asymmetries and transient bottlenecks arise, the DRL-based

layer (DCL) can extract an additional 1–5 % savings by dynamically synchronizing paired buffers and prioritizing high-penalty end items.

- *Computational Trade-Offs:* Training DCL at scale (e.g., our 7-node case) is resource-intensive (12 min on an HPC cluster or +8 h on standard hardware), offers no convergence guarantee, and yields a less interpretable policy. Use it only where incremental gains justify the cost.
- *DCL Limitations:* When capacity groups are fully saturated, especially in deeper, multi-echelon networks, DCL's incremental gains fall below 1%. This can be driven both by the lack of headroom and by the DRL method's own limitations in such tight settings. In these regimes, organizations should either invest in physical capacity expansion or prioritize further enhancements to the DRL approach (e.g., richer action spaces, advanced exploration strategies) before expecting meaningful policy improvements.
- *Heuristic Enhancement via DCL Insights:* Even if full DRL deployment is impractical, patterns learned by DCL, such as dynamically shifting inventory toward faster paths or higher-penalty products, can be extracted into simpler, transparent rules or decision-support tools, bridging performance and interpretability.

Contributions and Outlook.

By combining capacity-aware base-stock optimization, shortage-driven allocation, and a discrete DRL roll-out, this thesis delivers the first scalable, data-driven framework for general (convergent-divergent) multi-echelon networks with shared capacities and volatile demand. Our approach preserves the interpretability and simplicity of classical heuristics while layering on lightweight, state-dependent learning to capture residual savings. Future research may explore policy architectures specialized for divergent versus single-successor nodes within a coordinated scheme to share value across partitions; additionally, dynamic reservation rules with brief lookahead could better balance risk and responsiveness; and finally, adapting our methods to non-stationary, correlated demand could be of great value for more realistic high-tech instances. In practice, these methods offer supply-chain managers a clear, implementation-friendly path: begin with GBS + WF to secure most savings, then deploy DCL selectively where capacity slack remains and lead-time asymmetries are most pronounced.

Overall, this work demonstrates that pragmatic extension of established heuristics with discrete, smart-allocation and capacity-aware capabilities, plus DCL, can bridge the gap between elegant theory and the messy constraints of real-world, high-tech supply networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis investigates how to better support inventory control in modern high-tech supply chains, such as ASML's, by developing more flexible and responsive decision-making approaches. These supply chains face increasing pressure to manage multi-echelon inventories under uncertainty, shared components, and capacity constraints. Such complexities challenge traditional planning methods, motivating the exploration of both heuristic and emerging data-driven techniques.

1.1 Context

ASML, established in 1984 as a Philips-ASML joint venture, has become the global leader in semiconductor lithography systems. The company employs 39,000 people from 143 nationalities across more than 60 global locations and reported €7.5B in Q3 2024 sales, with projections of around €28 billion for the full year. ASML holds a unique position as the sole manufacturer of extreme ultraviolet (EUV) lithography systems, enabling 3nm chip production, while also producing deep ultraviolet (DUV) systems. ASML's commitment to innovation is evident in its substantial R&D investments, reaching $4.592 billion for the twelve months ending September 30, 2024, an 11.58% increase year-over-year (ASML, 2024). This focus on continuous improvement in accuracy and speed (measured in wafers processed per hour) is crucial for maintaining the industry's pace with Moore's Law, which predicts the doubling of transistors on a chip every 18 to 24 months. Within ASML, the Planning and Delivery (P&D) department is responsible for integrating planning activities across the company. The Strategy team within P&D actively develops innovative methodologies to improve supply chain decision-making and find ways that help production plans remain resilient in an environment where disruptions and variability are inevitable. Recent research by Van Dijck et al. (2024) explored the potential of Deep Reinforcement Learning (DRL) to optimize inventory management within ASML's supply chain and their findings demonstrated that DRL-based policies could outperform traditional benchmark approaches. However, their model assumes a fully convergent supply structure and a single end-product, which oversimplifies ASML's real-world operations. This research builds upon their work and aims to better capture the nature of ASML's operations.

1.2 Problem Description

The semiconductor industry operates at the forefront of technological advancement, where rapid innovation is accompanied by significant supply chain complexities. High capital investments, long lead times, and relentless pressure for miniaturization and performance

improvements drive manufacturers to adopt increasingly sophisticated planning methods (Geng and Jiang, 2009). The COVID-19 pandemic and ongoing geopolitical tensions have further exposed vulnerabilities within global semiconductor supply chains, disrupting production and exacerbating supply-demand imbalances across industries ranging from consumer electronics to data centers. However, these disruptions are only part of a broader landscape of structural challenges that require more adaptive and resilient planning approaches.

The industry's high dependence on intricate, multi-tiered supply networks makes production planning especially difficult. As market demand fluctuates, variations are amplified throughout the supply chain, creating a "bullwhip effect" that leads to inefficiencies in capacity utilization and inventory management (Sucky, 2009). Traditional planning methods struggle to mitigate these fluctuations, particularly as firms must balance technological complexity, uncertain demand, and stringent quality requirements, all while ensuring cost efficiency and meeting tight production schedules (Madanchian and Taherdoost, 2024; Smirnov et al., 2021). Consequently, companies that can dynamically adjust inventory levels, anticipate disruptions, and optimize resource allocation gain a competitive advantage in an industry where technological leadership and billions in capital investment shape market positioning (Chong et al., 2017).

ASML, as the sole manufacturer of EUV lithography machines, plays a crucial role in the semiconductor supply chain. Its upstream position exposes it to extreme demand fluctuations, driven by the investment cycles of chip manufacturers and technological advancements. Managing production and inventory in this context is highly complex, as ASML must navigate long supplier lead times, the high value of its products, capacity constraints, and the challenges of low-volume, high-precision manufacturing. Unlike high-volume industries, where inventory adjustments can be made in small increments, ASML's capital-intensive equipment and component dependencies require precise planning to avoid excessive stockpiling or critical shortages. Given the scale and cost of its systems, even minor miscalculations, like rounding up/down, in inventory levels can lead to critical supply shortages and production delays.

Furthermore, ASML operates within a general supply network, which features both convergent (assembly) parts, wherein multiple components are combined into intermediate subassemblies before culminating in the final product, and divergent (distribution) parts, wherein common components diverge to serve multiple end-products, thereby creating interdependencies across various product lines. This structure leads to multi-commonality, where multiple end-products depend on shared upstream components, making inventory and replenishment planning significantly more complex than unidirectional systems. Additionally, ASML must account for capacity groups, where replenishment decisions are restricted by shared capacity constraints, which require careful resource allocation across competing products. Further complexities arise from non-stationary and stochastic demand, where fluctuations occur over different time horizons, making forecasting possibly unreliable due to the difficulty of capturing long-term trends and short-term variations in a single predictive model. Moreover, non-deterministic lead times and uncertain supply availability (e.g., supplier delays and transportation disruptions) make planning difficult, forcing ASML to either hold excess safety stock or risk stockouts (or shortages) that disrupt production and delay downstream deliveries.

To manage these challenges, ASML employs three primary mathematical models for tactical production-inventory planning: RampFlex, CFO, and a Material Requirements Planning (MRP) approach. RampFlex is a multi-stage stochastic programming model within a rolling horizon framework, primarily used for safety stock placement—determining where

to buffer stock to hedge against uncertainty. However, it relies on demand forecasts, which can introduce errors in ordering decisions. CFO enables rapid production planning but lacks the flexibility to effectively manage capacity constraints. Finally, at an operational level, ASML's ordering decisions are governed by an MRP-style approach, where replenishment is based on forecasted demand and predefined rules. Together, these approaches provide a layered structure for planning, yet they do not directly address key operational questions such as when and how much to order or produce in real-time, nor how to allocate materials during shortages. There is currently no integrated mechanism to support dynamic, supply network-wide coordination of decisions under uncertainty. As such, ASML's planning toolkit, while strategically sound, lacks the operational agility required to respond effectively to real-time disruptions, capacity bottlenecks, and shifting demand patterns.

This gap has motivated interest in more adaptive and data-driven methods within ASML, particularly Deep Reinforcement Learning (DRL) has emerged as a promising approach to learn decision policies directly from system interaction. Van Dijck et al. (2024) applied DRL in a simplified, fully convergent setting and demonstrated performance improvements over heuristic policies, even under non-stationary demand. While their work provides a strong proof of concept, it does not fully capture the structural and operational realities of ASML's network. Specifically, their model assumes a single end product, omitting multi-commonality and capacity constraints, and a fully convergent structure, which overlooks ASML's broader general network configuration.

As a result, current models, whether heuristic or learning-based, struggle to handle the full scope of ASML's supply chain complexities, including its multi-echelon structure, shared components, group capacity constraints, non-stationary demand, and uncertain lead times. While RampFlex, CFO, and MRP offer structure, they fall short in providing flexible, real-time decision support. Likewise, while DRL has demonstrated potential in simplified networks, its applicability to more realistic, general settings remains limited and underexplored. To advance decision-making in this context, there is a need for an integrated approach that not only captures ASML's operational complexity, but also enables dynamic coordination and policy learning. This thesis addresses that gap by developing and benchmarking analytical and DRL-based methods for inventory and production optimization in general supply networks, aiming to enhance decision support in high-tech manufacturing environments.

1.3 Research Design

Given the supply network complexities discussed previously, such as multi-echelon dependencies, multi-commonality, capacity constraints, and demand uncertainties, developing effective inventory policies for ASML requires a structured yet practical approach. This section outlines the research goal, scope, and key research questions guiding this study.

Research Goal

This research aims to develop and systematically benchmark a well-designed heuristic method to optimize tactical and operational inventory decisions in complex multi-echelon supply networks like ASML's. Additionally, it explores how advanced data-driven methods, particularly DRL, could further improve these heuristic policies or reveal superior alternatives.

This thesis contributes the following:

- A stylized yet representative simulation model capturing ASML's key complexities

- Fast and well-performing analytical heuristic methods tailored to the given inventory system.
- A scalable DRL method for adaptive decision-making.
- A benchmark and insight-rich comparison of heuristic and DRL behavior

Research Questions

To guide this study, the following central research question is formulated:

How can an effective heuristic method be developed to optimize inventory management in ASML's complex supply network, and how can a DRL method contribute to improving this method or uncovering superior inventory policies?

Addressing this question involves a stepwise approach that incrementally builds a comprehensive understanding and solution framework. The process begins with analyzing the specific complexities of ASML's supply network. Based on these insights, a suitable inventory modeling framework is constructed, followed by formal problem formulation and the development of a simulation environment for evaluation. Next, appropriate heuristic benchmarking methods are identified to serve as a performance baseline. A scalable DRL-based approach is then designed, implemented, and tested within the same environment. Finally, both methods are validated, systematically compared, and analyzed to extract actionable insights that can inform decision-support tools for ASML's supply chain.

To address the central question systematically, it is broken down into the following subquestions:

1. What are the defining complexities of ASML's multi-echelon inventory system?

   Before designing an optimization framework, it is essential to understand ASML's supply chain complexities. Unlike traditional high-volume inventory systems, ASML operates in a low-volume, high-value production environment with multi-echelon dependencies, shared components, and finite ordering capacities. This raises several fundamental questions:

   (a) How do ASML's general supply network structures, shared components, and capacity constraints differentiate it from conventional inventory models?

   (b) How do ASML's discrete production setting, and demand uncertainties impact inventory planning?

   (c) What service level definitions are most appropriate given ASML's operational constraints?

2. What inventory modeling approaches have been explored for complex supply networks, such as ASML's, and how are such problems typically structured?

   Once the key complexities of multi-echelon supply networks are identified, the next step is to explore existing modeling approaches that have been proposed in the literature. A wide range of inventory control models and solution methods have been developed for different network structures, each tailored to specific system characteristics and complexity levels.

   (a) What inventory control models have been used in the context of multi-echelon and general networks?

   (b) How have researchers addressed modeling challenges in networks with both convergent and divergent flows, especially when structural interdependencies and component sharing are present?
   (c) What types of solution methods are typically applied in different settings, and under what conditions are they effective?
   (d) How can insights from studies on simpler network structures inform further modeling choices in more complex general settings?

3. How can the problem be formally modeled, and how should the simulation environment be designed?

   Once an appropriate inventory modeling approach is selected, the next step is to formulate the problem in a way that captures uncertainties evolving over time. A suitable framework for this is a Markov Decision Process (MDP), which supports building a simulation environment for testing and benchmarking.

   (a) What is the appropriate formulation of the MDP components, including state space, action space, transition dynamics, reward function, time horizon, and discount factor, to realistically capture the system's behavior?
   (b) How should the simulation model be constructed to replicate key structural features of ASML's supply network, while remaining abstract enough to allow controlled experimentation?
   (c) How should the evaluation setup be structured, starting with simplified test instances and gradually scaling up to more complex representations that reflect ASML's operational characteristics?

4. How can a heuristic method be designed to provide a good analytical benchmark for inventory decisions?

   After the simulation environment and problem formulation are in place, the next step is to design a practical and interpretable heuristic policy. A well-designed analytical method, like a heuristic, should serve not only as a baseline for performance and a source of operational insights, but also as an easily implementable solution. This benchmark should be tailored to ASML's specific constraints and system behavior.

   (a) How can components for this heuristic method be correctly designed and implemented.
   (b) What material allocation rules should be used to manage shortages effectively?

5. How can DRL be designed to be scalable, efficient, and suitable for ASML's decision environment?

   While heuristics provide interpretable and computationally efficient solutions, they may struggle with dynamic, high-dimensional decision spaces. To address this, we explore the potential of Deep Reinforcement Learning (DRL) to learn adaptive policies directly from system interaction. Designing such a DRL approach requires a scalable and system-specific design. Additional factors include careful consideration of ASML's discrete decisions (e.g., production quantities), which are often subject to group capacity limits and component interdependencies. These characteristics require tailored action space formulations and scalable learning architectures to ensure that DRL policies remain computationally feasible and practically relevant for benchmarking and insight generation.

(a) What DRL architectures and action space formulations are best suited for ASML's discrete decision-making environment?

(b) How can action space structuring, state representation, and reward shaping be designed to support efficient learning?

(c) How can DRL be tuned (e.g., hyperparameters) to ensure scalability, convergence, and generalization across network scenarios?

(d) How can DRL-generated insights be translated into enhancements for heuristic interpretable methods and actionable recommendations for supply chain managers?

6. How can heuristic and DRL policies be effectively evaluated, and what performance can be expected under different conditions?

   After both the heuristic and DRL-based policies are developed, it is essential to compare their performance systematically. This requires a well-structured experimental design that captures key KPIs, tests generalizability, and supports meaningful analysis of strengths, weaknesses, and practical applicability.

   (a) What evaluation setup (KPIs and test scenarios) best captures meaningful differences in cost, service level, and adaptability between methods?

   (b) Under which conditions do heuristic and DRL methods perform best, or break down?

   (c) How can insights from DRL inform improvements to the heuristic methods and support actionable guidance for supply chain decision-makers?

Research scope

Given the complexity of ASML's supply network, directly modeling its full operational reality is impractical due to the vast number of interacting components, constraints, and uncertainties. Instead, this study adopts a stylized yet representative problem formulation that remains computationally tractable while capturing essential system characteristics. These include multi-echelon dependencies, shared components (multi-commonality), group-based capacity constraints, demand uncertainty, and discrete decision-making. To maintain focus and feasibility, the model does not incorporate non-stationary demand patterns or non-deterministic lead times, though these are recognized as relevant factors in ASML's real-world operations.

The scope of this research is limited to tactical and operational inventory decision-making—specifically, determining when and how much to order or produce, and how to allocate materials in the presence of shortages. It explicitly excludes broader elements such as detailed production scheduling, supplier negotiation strategies, or large-scale disruptions caused by geopolitical or systemic risks. Although simplified, the model is designed to generate insights that can support the development of new, adaptive replenishment strategies aligned with ASML's operational needs or be combined with existing tools such as RampFlex, particularly in the context of safety stock placement. The practical implementation of such strategies, however, falls outside the scope of this work.

1.4 Overview of Research Methodology

To guide the reader through the remainder of this thesis, Figure 1.1 presents an overview of the overall research structure. This outlines the sequential phases of this thesis, from

analyzing ASML's supply network and reviewing existing literature, to developing a representative model, designing both heuristic and DRL-based methods, and conducting a comparative evaluation. Each research question maps to a specific phase in this process. The upcoming chapters follow this structure: Chapter 2 reviews relevant literature, Chapter 3 introduces the model and simulation setup, Chapters 4 and 5 present the heuristic and DRL methods, respectively, Chapter 6 covers the computational experiments and their evaluation, and Chapter 7 concludes with conclusions and recommendations.

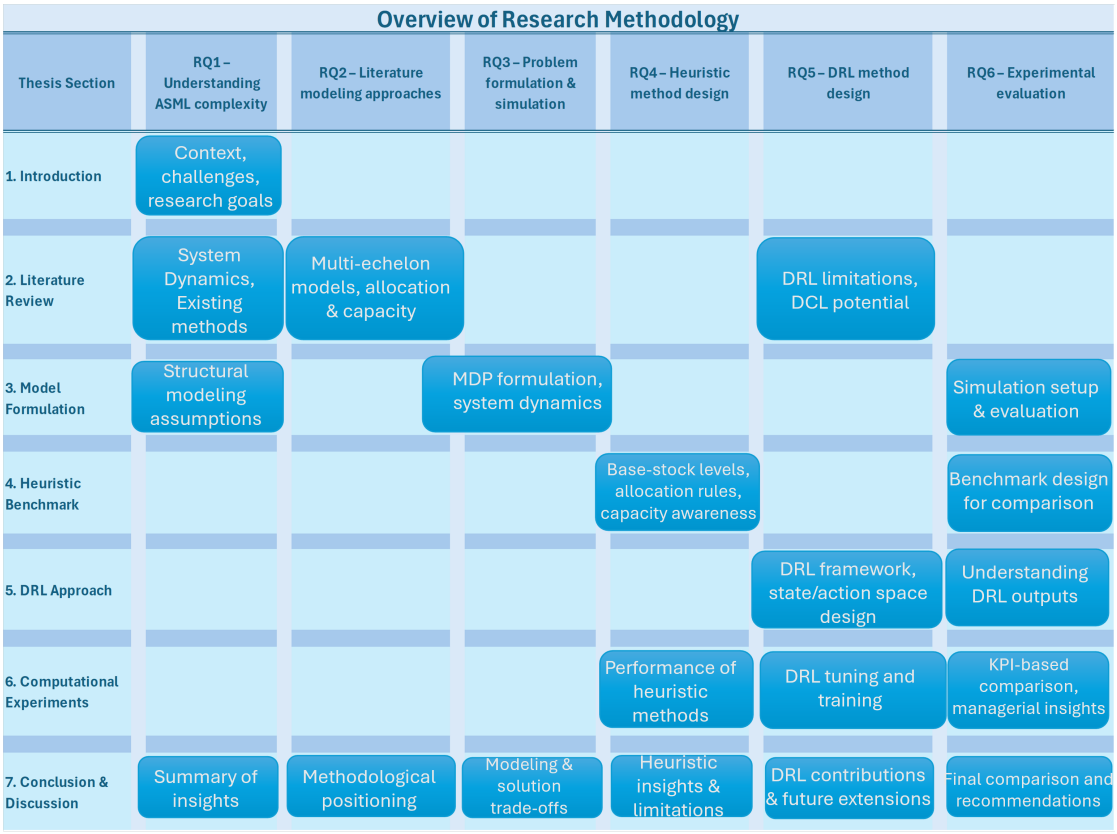


FIGURE 1.1: Overview of the research methodology.

Chapter 2

Literature Review

The challenge of inventory optimization in complex multi-echelon supply chains characterized by shared components (multi commonality), stochastic demand, finite capacities, and intricate network structures has been widely explored in the literature through various analytical models, heuristic methods, and increasingly, data-driven approaches. We structure this review around (i) foundational frameworks for inventory optimization, (ii) control policies in general networks, (iii) capacity-constrained systems, and (iv) emerging DRL-based approaches.

2.1 Multi-Echelon Inventory Optimization and Challenges

Multi-echelon inventory management optimizes ordering and production across supply chain stages to balance cost minimization, service-level maintenance, and uncertainty mitigation. Challenges stem from interconnected stages, shared components, finite capacities, and demand/lead time variability (De Kok et al., 2018). By coordinating inventory holistically rather than in isolation, multi-echelon systems can reduce total inventory by up to 30% while improving item availability by 5% (Clark and Scarf, 1960).

2.1.1 Primary Frameworks for Multi-Echelon Inventory Optimization

Two foundational frameworks have emerged to model and solve multi-echelon systems. The Guaranteed Service Model (GSM), introduced by Simpson (1958) and generalized by Graves and Willems (2003), simplifies multi-echelon inventory optimization by assuming deterministic service from upstream stages. In essence, each stage guarantees delivery within a fixed lead time, allowing downstream nodes to plan accordingly without modeling upstream uncertainty.

The deterministic service time decouples stages in the supply chain and enables the use of linear programming to determine optimal safety stock levels (Eruguz et al., 2016). Because of its tractability and ease of implementation, GSM has seen widespread adoption in settings where lead times are relatively stable. However, GSM's rigidity often leads to excessive safety stock or service failures in more volatile environments. This is particularly problematic in high-tech industries like semiconductor manufacturing, where disruptions propagate quickly and unpredictably through the network (Simchi-Levi and Zhao, 2012). Even recent extensions that incorporate demand-bound formulations still struggle to dynamically adjust to real-time variability (Eruguz et al., 2016).

In contrast, the Stochastic Service Model (SSM), originating from Clark and Scarf (1960), explicitly accounts for randomness in demand and lead times. Unlike GSM, SSM

models how disruptions at one stage propagate downstream. If a stage fails to meet its replenishment targets due to demand or supply variability, it triggers delays that accumulate at subsequent stages, increasing backorders across the supply chain. This requires solving high-dimensional stochastic dynamic programs. Clark and Scarf (1960) established the optimality of echelon base-stock policies for un-capacitated serial systems under constant lead times, a result later extended to convergent systems by Rosling (1989), who demonstrated their equivalence to serial structures. For divergent systems, Diks and De Kok (1998) derived optimal policies under restrictive assumptions (e.g., allowing negative shipments), though practical implementation remains challenging. Moreover, general multi-echelon networks lack universally optimal solutions due to combinatorial complexity (De Kok et al., 2018), and they have been largely overlooked in existing literature, see De Kok and Visschers (1999) for example. Optimal outcomes have generally been limited to specific cases like fixed batch sizes or independent Poisson demand within two-echelon systems (Nadar et al., 2014). Rong et al. (2017) discuss heuristic strategies for un-capacitated general systems, but since their study is centered on distribution systems, they do not extend their results to general systems.

2.1.2 Extensions and Alternatives to Base-Stock Policies

Despite the fact that base-stock policies may not be optimal in general systems, they remain widely applied due to their simplicity and tractability. However, alternative policies such as (s, S), (R, Q), and MRP-based planning are also used, and each offers trade-offs in complexity and responsiveness. The (s, S) policy triggers replenishment when inventory falls below a reorder point, while (R, Q) policies order fixed quantities. Though effective for stochastic demand, these methods face computational intractability in multi-echelon systems due to the exponential growth of decision variables (Axsäter, 2015). Echelon-stock (R,Q) policies extend traditional (R,Q) logic by accounting for downstream inventory positions when making replenishment decisions, thereby centralizing control across multiple stages to reduce overall stock levels. This differs from installation-stock policies, where each stage manages its own inventory independently, which tends to amplify demand variability throughout the network (Axsäter and Rosling, 1993). Moreover, a method that is widely adopted in structured manufacturing environments is the Material Requirements Planning (MRP) due to its simplicity and integration with ERP systems. MRP combines bills of materials with production schedules to forecast replenishment needs, but its reliance on deterministic forecasts limits effectiveness under uncertainty, often requiring manual adjustments to stay responsive (Snyder et al., 2016).

2.1.3 Inventory Allocation in Divergent and General Supply Networks

Divergent networks, where upstream stages distribute inventory to multiple downstream nodes, face critical allocation challenges during stockouts. Poor allocation of scarce inventory in such systems can exacerbate service imbalances and propagate shortages downstream, leading to significant inefficiencies, increased backorder costs, and a systemic decline of service levels. This is why allocation mechanisms are critical for maintaining service levels and minimizing overall costs in these settings. Several allocation methods exist, each with trade-offs in complexity, accuracy, and scalability. Table 2.1 summarizes their key features.

A commonly used yet theoretically limited rule is first-come-first-served (FCFS), which allocates inventory strictly in the order demands arrive, irrespective of origin (Zipkin, 2000). To address this, Eppen (1981) introduced the balance assumption, where unmet demand is fully backordered. Under this approach, available inventory is allocated to the

earliest backorders first, simplifying analysis in models like the one-warehouse-multi-retailer (OWMR) system. However, its efficacy is contingent on system parameters as noted by Doğru et al. (2009). Persistent imbalances in retailer imbalances may be better addressed by alternative allocation rules such as: Proportional allocation, which distributes inventory based on demand share, reducing disparities but ignoring critical needs, and shortfall minimization, which allocates inventory to minimize the maximum shortfall between base-stock levels and actual inventory (Kaynov et al., 2024).

To achieve optimality, Axsäter (1990) proposed the Projection Method, which formulates allocation as a non-convex optimization problem to provide exact solutions. Although highly accurate, its exhaustive search process is computationally infeasible for large networks. To overcome these scalability challenges, heuristic approximations such as the METRIC model (Sherbrooke, 1968) and Graves' two-moment approach (Graves, 1985) trade off precision for traceability. These methods simplify calculations by approximating key performance metrics—such as backorders and fill rates—using stochastic lead time or demand assumptions, rather than solving the exact optimization problem. While less precise, they are widely adopted in practice for scenarios with stochastic lead times.

Recent advances in adaptive balancing for machine learning include Randomized Sequential Allocation (Kaynov et al., 2024), which assigns all inventory in a random sequence to reduce allocation bias of FCFS, while it is fairer, it can lead to unfulfilled demand if stock runs out early. Randomized Balanced Allocation (Stranieri et al., 2024) improves upon this by dynamically assigning units to retailers based on real-time demand, better balancing inventory and reducing performance gaps by up to 5% as shown in simulations.

2.1.4 Capacitated Systems

Finite production capacities complicate multi-echelon optimization, invalidating traditional base-stock assumptions. For the GSM, Graves and Schoenmeyr (2016) propose a heuristic where replenishment orders are capped at capacity limits while maintaining near-optimal base-stock levels. Similarly, Huh et al. (2016) analyze shortfall processes to derive near-optimal policies for capacitated serial systems (SSM), later extended by Van Dijck et al. (2024) to capacitated assembly systems.

Recent advances combine production and buffer planning through a rolling horizon framework to balance feasibility and cost-effectiveness, though scalability remains a challenge (Fleuren et al., 2022). For dynamic adaptation, Woerner et al. (2016) combines perturbation analysis with iterative tuning of base-stock levels to adapt policies to capacity-induced bottlenecks, albeit at computational costs. Despite these advancements, optimizing capacitated multi-echelon systems with convergent and divergent flows remains highly complex. Existing heuristics offer solutions for specific cases, but a scalable, universally effective method is still missing, highlighting the need for stronger approaches that balance capacity constraints, demand variability, and service levels in large networks.

2.2 Deep Reinforcement Learning for Inventory Management

Recent advances have positioned DRL as a promising alternative to traditional heuristics for inventory management in complex, uncertain, and capacitated supply networks. DRL leverages neural network architectures to dynamically learn policies directly from system interactions, addressing many shortcomings of static inventory approaches (Sutton and Barto, 2018; Mnih et al., 2015).

The subsequent discussion examines the development of DRL methodologies within the

Method	Description (with Pros and Cons)	References
FCFS	Allocates based on the demand arrival order. **Pros**: Simple, easy to use. **Cons**: Ignores priority, imbalances.	Zipkin (2000)
Balance Assumption	Assumes all unmet demand is backordered, prioritized by arrival. **Pros**: Simple, tractable in OWMR. **Cons**: Limited flexibility, sensitive to system parameters.	Eppen (1981); Doğru et al. (2009)
Proportional Allocation	Distributes proportionally across demand. **Pros**: Reduces disparity among retailers. **Cons**: Ignores critical needs, may neglect high-priority demands.	Kaynov et al. (2024)
Shortfall Minimization	Minimizes maximum shortfall across nodes. **Pros**: Efficient for addressing critical shortages. **Cons**: Needs accurate shortfall estimation.	Kaynov et al. (2024)
Projection Method	Solves allocation as a non-convex optimization problem for exact solutions. **Pros**: Highly accurate. **Cons**: Computationally expensive, not scalable for large networks.	Axsäter (1990)
METRIC Model	Uses stochastic lead-time approximations for backorder estimates. **Pros**: Scalable, computationally efficient. **Cons**: Less precise than optimization-based methods.	Sherbrooke (1968)
Graves' Two-Moment	Approximates performance using mean and variance. **Pros**: Simple calculations, efficient for analysis. **Cons**: Assumes normality, may be inaccurate for non-normal demand.	Graves (1985)
Randomized Sequential Allocation	Allocates inventory in random order to reduce bias. **Pros**: Fairer than FCFS, reduces order bias. **Cons**: Risk of unfulfilled demand if inventory depletes early.	Kaynov et al. (2024)
Randomized Balanced Allocation	Iteratively assigns units to retailers in random order to balance stock. **Pros**: Fair, low bias, prevents imbalances. **Cons**: May not guarantee optimal allocation.	Stranieri et al. (2024)

TABLE 2.1: Overview of Key Inventory Allocation Methods in Divergent and General Supply Networks

context of Inventory Management, with a focus on addressing the complexities associated with multi-echelon systems.

2.2.1 Evolution of DRL Methods in Inventory Management

DRL methods have evolved to tackle inventory challenges like partial observability and dynamic demand. This section traces their progression from RL to tailored DRL algorithms specifically for inventory management, highlighting advancements in scalability and adaptability.

From Tabular Q-Learning to Deep Q-Networks (DQNs)

Early RL applications in inventory management relied on tabular Q-learning for small-scale systems (Giannoccaro and Pontrandolfo, 2002). However, these methods failed to scale to multi-echelon networks due to exponential growth in state-action spaces (Geevers et al., 2024). The advent of Deep Q-Networks (DQNs) marked a breakthrough by using neural networks to approximate Q-values, enabling partial scalability (Mnih et al., 2015). While DQNs excelled in deterministic environments like Atari games, they struggled in inventory settings due to **partial observability** (e.g., unobserved in-transit stock) and **large discrete action spaces** (e.g., order quantities), often producing unstable or unrealistic policies (Oroojlooyjadid et al., 2021).

Policy Gradient Methods: Direct Policy Optimization

Policy gradient methods, such as Asynchronous Advantage Actor-Critic (A3C), circumvented DQN limitations by optimizing stochastic policies directly through gradient ascent (Mnih et al., 2015). A3C's parallelized training improved exploration in stochastic environments, but its **hyperparameter sensitivity** and **unstable convergence** hindered reliable deployment in complex supply chains (Gijsbrechts et al., 2022).

Proximal Policy Optimization (PPO): Stabilizing Training

Proximal Policy Optimization (PPO) addressed instability by constraining policy updates, enabling robust performance in joint replenishment and capacitated lot-sizing problems (Schulman et al., 2017; Van Hezewijk et al., 2023). PPO's success in two-echelon divergent networks (Stranieri et al., 2024) highlighted its adaptability to multi-stage systems, though its reliance on continuous action spaces limited granularity in discrete order decisions.

Deep Controlled Learning (DCL): A Paradigm Shift

Deep Controlled Learning (DCL) reframed DRL as a classification task, iteratively refining policies through supervised learning on "elite" trajectories (Temizöz et al., 2023). This approach achieved faster convergence in high-stochasticity settings (e.g., perishable inventory, random lead times) and demonstrated adaptability to non-stationary demand in semiconductor supply chains (Van Dijck et al., 2024). Unlike traditional DRL, DCL avoids value function approximation pitfalls, making it uniquely suited for trend-sensitive environments like ASML's high-tech manufacturing.

2.2.2 Addressing Multi-Echelon Complexities

DRL shows promise for multi-echelon inventory optimization, but applying it to low-demand, high-tech settings poses unique challenges. Additionally, unlike structured networks, general supply chains have complex interdependencies with convergent and divergent flows, needing adaptive replenishment and efficient allocation to manage shortages.

Existing methods, such as those proposed by Geevers et al. (2024), employ a continuous action space with PPO, which is unsuitable in sparse-demand environments, where every order must be rounded to an integer, producing systematic over- or under-stocking that is costly in capacity-constrained, high-value settings. Techniques like action masking limit infeasible moves (Peng et al., 2019), yet converting continuous outputs to integers (e.g., by rounding Soft Actor-Critic signals) still injects error and erodes service levels in low-volume regimes (Vanvuchelen et al., 2025). For such contexts, multi-discrete action spaces that let the agent choose exact unit quantities are a closer fit to operational reality.

Integer-valued action spaces lead to combinatorial growth when allocating stock across many nodes, with the number of feasible actions exploding as recipients increase (Kaynov et al., 2024). To address this, multi-discrete action factorization decomposes joint decisions into per-node outputs, reducing dimensionality from exponential to linear (Van Dijck et al., 2024). However, this decomposition introduces new issues in divergent networks, where local allocations may exceed available inventory, necessitating corrective mechanisms to maintain feasibility, see Kaynov et al. (2024).

Moreover, the complexity of effective allocation strategies, as discussed in Section 2.1.3, also extends to DRL implementations. Allocation rules can restrict the action space, but when applied sequentially, they may bias inventory toward upstream or early-served nodes, which can make it hard for downstream nodes to recover from shortages. To mitigate this,

Kaynov et al. (2024) use randomized sequential allocation to reduce bias by distributing full inventory in random order, while Stranieri et al. (2024) improves this by randomly allocating unit-by-unit to improve fairness.

Transfer learning offers another path to scalability. Oroojlooyjadid et al. (2021) showed for a serial system that a Beer-Game agent's policy can be transplanted to new parameter settings with little retraining. However, in the case of general systems, the agent's performance and transferred knowledge may degrade under complex interdependencies and information asymmetries.

Recent advances in general systems include Pirhooshyaran and Snyder (2021), who developed a DNN-based order-up-to policy with proportional allocation; Wang and Hong (2023) propose a new method for large-scale inventory optimization using Recurrent Neural Networks (RNNs), encoding the supply network's structure within an RNN for rapid end-to-end policy evaluation and gradient estimation, achieving large speedups compared to classical simulation-based approaches; and Harsha et al. (2025), whose programmable actor RL (PARL) framework combines deep policy iteration with mathematical programming to handle combinatorial action spaces and state-dependent constraints. PARL outperforms state-of-the-art RL algorithms and heuristics by up to 14.7% in complex settings, leveraging sample average approximation for per-step optimality, additionally, the authors provide an open-source Python library to democratize RL adoption in inventory management. Complementing these approaches, DCL has emerged as a promising model-based alternative for high-variability systems by treating policy learning as a classification task over simulation-based state-action pairs. This approach improves sample efficiency and minimizes hyperparameter tuning. Recent studies by Van Dijck et al. (2024) and Temizöz et al. (2023) demonstrate that DCL achieves sub-0.2% optimality gaps with substantially lower training times.

Allocation rules from the previous section also apply to DRL since they can constrain the action space (see sub-section 2.1.3). However, these strategies can lead to systematic biases, particularly when sequential allocation rules are applied, as downstream warehouses may receive insufficient inventory, making it harder to recover from shortages. Kaynov et al. (2024) addressed this issue by introducing a randomized sequential allocation rule, where orders are executed in full, one by one, until the central warehouse inventory is depleted. While this mitigates bias in order fulfillment, it does not guarantee that all locations receive stock, leading to potential supply imbalances. To improve fairness, Stranieri et al. (2024) proposed a balanced allocation rule for continuous action spaces, ensuring that stock is allocated equitably while preventing infeasible shipments. Their approach, applicable to multi-discrete action spaces, prevents disproportionate inventory imbalances while maintaining feasibility.

2.3 Research Gap and Contribution

The review shows a clear disconnect between current theory and the needs of ASML-type supply chains. First, general multi-echelon networks with both convergent and divergent flows remain largely unsolved. Most studies adopt the Guaranteed-Service Model, yet volatile, low-volume demand indicates that a Stochastic-Service perspective is more appropriate; beyond the distribution-focused heuristic of Rong et al. (2017), no practical SSM-based method exists for truly general topologies. Second, allocation in divergent stages is still rudimentary: simple FCFS rules or recent random-sequence variants (Kaynov et al., 2024; Stranieri et al., 2024) either bias stock or fail to scale to mixed convergent–divergent settings. Third, group-based capacity constraints (common at ASML) are virtually absent

from the literature; serial shortfall approaches (e.g., Huh et al. (2016)) cannot cope with shared-capacity groups, leaving both planning and allocation under capacity constraints highly ad-hoc. Finally, recent DRL work either assumes continuous actions or ignores capacity sharing; existing discrete methods are limited to small networks, and no study combines SSM dynamics with discrete, capacity-aware DRL in general systems.

This thesis fills these gaps by developing a unified, capacity-aware ordering framework for general multi-echelon systems with volatile demand. We (i) extend SSM reasoning to general networks with shared-capacity groups, (ii) design new allocation rules that balance fairness and feasibility in convergent–divergent structures, and (iii) integrate these components into a discrete, multi-action DRL scheme. Together, these contributions provide the first scalable, data-driven solution tailored to ASML's high-tech, low-demand supply chain and advance the state of general multi-echelon inventory control.

Chapter 3

Model Formulation

This chapter formalizes the tactical production-inventory planning problem for ASML's multi-echelon supply chain as an infinite-horizon Markov Decision Process (MDP). The formulation captures ASML's role as central decision-maker coordinating production across nodes in a network spanning from raw material suppliers to the final assembly of lithography systems. The objective is to determine optimal stationary production policies that minimize total costs, comprising inventory holding penalties for components and backlog costs for unmet end-product demand, under capacity constraints and stochastic demand patterns.

To structure the formulation, the chapter proceeds as follows. Section 3.1 introduces the case context and key structural features of ASML's supply network. Section 3.2 defines the components of the Markov Decision Process (MDP) model, including the sets and parameters, the state space, the action space, the reward function, and the transition dynamics.

3.1 Case Context

ASML's supply chain exhibits two distinctive structural features that shape the modeling approach. First, **convergent flows**, where ultra-precision modules—such as EUV light sources and wafer stages—integrate thousands of components into atomically synchronized assemblies. Second, **divergent flows**, where shared subsystems like silicon carbide mirrors or laser modules branch across product lines (EUV, DUV, metrology). In addition, components are grouped into **capacity-constrained processing stages**, such as optical polishing or sub-nanometer testing, where shared resources—like ion-beam figuring tools or vibration-isolated cleanroom cells—create contention and bottlenecks.

Figure 3.1 illustrates the structural setup of our stylized supply network, organized into three logical layers: upstream **suppliers** (C_0, C_1), intermediate **pre-assembly processing stages** (C_2, C_3, C_4), and downstream **end products** (C_5, C_6). Each component C_i is **permanently assigned to a single facility or group** G_k (denoted $C_i \in \mathcal{C}_g$), with arrows indicating material flows, annotated by production lead times l_i, and each component incurring a holding cost h_i. For example, C_0 is processed at Group G_0 and flows to both C_2 and C_3, each with a lead time of 2. Moreover, final components C_5 and C_6 represent end-products ($e \in \mathcal{C}_{end}$) with stochastic period-specific customer demand $d_{e,t}$. Unmet demand is backlogged at a cost of b_e per unit. For instance, group G_4 processes both C_5 and C_6, subject to a capacity of Q_4, with respective backlog penalties $b_5 = 35$ and $b_6 = 40$.

In this representation, *groups* (G_k) denote physical production or assembly resources, subject to shared capacities (Q_k) and operational constraints, while *components* (C_i) are the individual flow units being stocked, moved, and consumed.

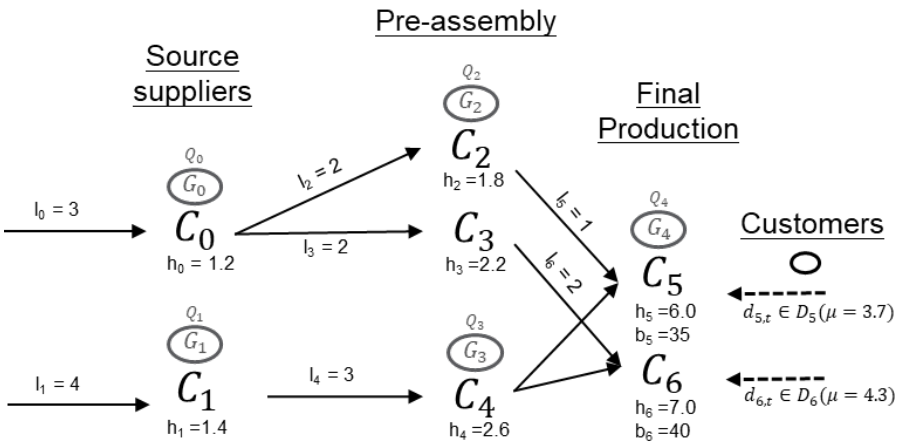


FIGURE 3.1: Stylized general system network. Each component C_i is processed at group G_k, with arcs indicating material flows and annotated with production lead times l_i, holding costs h_i, and backlog and demands $(b_e;d_{e,t})$ for each end item $(e \in \mathcal{C}_{end})$.

Stylized Network Features. The example network in Figure 3.1 embodies a set of structural assumptions aligned with ASML's real-world supply system:

- **Convergence and divergence:** Components may merge into assemblies (e.g., C_2, C_4 into C_5) or split across multiple end-products (e.g., C_4 to C_5, C_6), reflecting complex module integration and shared subassemblies.
- **Single-use components:** Each component is consumed in exactly one downstream assembly (e.g., C_0 flows to C_2 and C_3, but is not reused in subsequent stages). This prevents component recycling, ensuring traceability and avoiding requalification costs in systems with irreversible assembly steps.
- **Universal Group Contribution:** Every facility or group G_k contributes indirectly to **all** end products (C_5, C_6) through various assembly paths, which ensures that all groups are vital and fully utilized in the supply chain.
- **Acyclic structure:** The network follows a forward-only flow with no loops, consistent with irreversible manufacturing steps and certification sequences.
- **Deterministic timing and cost structure:** Each arc has a fixed production lead time, and each component incurs a deterministic holding cost—assumptions that reflect ASML's engineered-to-order operations and need for cost visibility.

These structural assumptions ensure logical consistency with ASML's real-world supply dynamics and provide a tractable basis for decomposition and analysis. Additionally, three operational constraints further define the modeling scope. First, production facilities are subject to **capacity limits**, particularly at shared or bottleneck stages like optics polishing or cleanroom testing. Second, **lead times** vary across components, driven by their complexity and sourcing origin. Finally, **end-product demand** $d_{e,t}$ follows discrete, low-volume distributions—typical of ASML's high-mix, low-volume environment.

3.2 MDP Components

The MDP formulation begins by defining the sets and parameters, followed by the state space, action space, reward function, and transition dynamics. This formulation builds in part on the pure assembly model proposed by Van Dijck et al. (2024).

3.2.1 Sets and Parameters

The model employs several sets and parameters (Table 3.1) to formally describe the supply network structure, component relationships, and operational constraints. Many of these elements have already been illustrated in Figure 3.1.

The bill-of-materials (BOM) is captured by a binary matrix $A \in \{0,1\}^{|\mathcal{C}|\times|\mathcal{C}|}$, where $a_{ij} = 1$ indicates that component i is required to produce component j. This representation supports both convergent and divergent flows, as components may have multiple upstream suppliers $\mathcal{U}(i)$ and multiple downstream consumers $\mathcal{D}(i)$. Finally, the system evolves over discrete time periods $t \in \mathcal{T}$, during which production decisions are made, inventories are updated, and demands for end-products are realized.

Table 3.1: Key sets and parameters

Symbol	Description
Sets	
$\mathcal{C}$	All components/products
$\mathcal{G}$	Production facilities
$\mathcal{T}$	Time periods
$\mathcal{C}_g$	Components at group g
$\mathcal{C}_{end}$	End-products
$\mathcal{U}(i)$	Upstream or predecessor components required to produce i (per BOM A)
$\mathcal{D}(i)$	Downstream or successor components that consume i (divergent flows)
Parameters	
l_i	Production lead time of component i
A	BOM adjacency matrix
$d_{e,t}$	Stochastic demand of end-product e at time period t
h_i	Holding cost per unit of component i per time period
b_e	Backlog penalty per unit of end-product e per time period
Q_g	Cpacity of group g

3.2.2 State Space

The system state s_t at period t fully describes inventory positions and production pipeline status across all nodes. For each component $i \in \mathcal{C}$, the state vector combines on-hand inventory $I_{i,t}$ with pipeline inventory $P_{i,t,k}$ representing units scheduled for completion in k periods. Formally:

$$s_t = \{[I_{i,t}, P_{i,t,1}, \ldots, P_{i,t,l_i-1}]\}_{i\in\mathcal{C}} \tag{3.1}$$

where $I_{i,t} \in \begin{cases} \mathbb{Z} & \text{if } i \in \mathcal{C}_{end} \\ \mathbb{Z}^+ & \text{otherwise} \end{cases}$, since it allows negative values for end-products (backlogs).

The exclusion of demand $d_{i,t}$ from the state vector is justified by its role as an exogenous stochastic process, assumed to be independent and identically distributed across periods.

3.2.3 Action Space

At each decision stage t, the agent selects production quantities $X_{i,t}$ for all components, subject to three operational constraints, which define the allowable action space $\mathcal{A}(s_t)$ from state s_t. First, group capacity limits (Constraint 3.2a) ensure the total production at each group n does not exceed Q_g. Second, material availability constraints (3.2b) prevent production starts that exceed current component inventories, critical for high-value items like EUV mirror modules. Third, integrality constraints (3.2c) enforce discrete unit production, matching ASML's low-volume context. Formally:

$$\mathcal{A}(s) = \Bigg\{ \sum_{i \in \mathcal{C}_g} X_{i,t} \leq Q_g \quad \forall g \in \mathcal{G}; \tag{3.2a}$$

$$\sum_{j \in \mathcal{D}(i)} X_{j,t} \leq I_{i,t} \quad \forall i \in \mathcal{C}; \tag{3.2b}$$

$$X_{i,t} \in \mathbb{N}_0 \quad \forall i \in \mathcal{C} \Bigg\} \tag{3.2c}$$

3.2.4 Reward Function

The immediate reward r_t represents negative total costs, combining inventory holding costs for upstream components and service-level costs for end-products. For upstream items ($i \notin \mathcal{C}_{end}$), costs accrue linearly with on-hand inventory. End-products incur holding costs only on surplus stock ($I_{j,t} - d_{j,t}$) and backlog penalties on unmet demand ($d_{j,t} - I_{j,t}$), with $b_j > h_j$ reflecting service-level agreements. The cost function $C(s_t, a_t)$ is:

$$C(s_t, a_t) = \sum_{i \notin \mathcal{C}_{end}} h_i I_{i,t} + \sum_{j \in \mathcal{C}_{end}} \left[h_j (I_{j,t} - d_{j,t})^+ + b_j (d_{j,t} - I_{j,t})^+ \right] \tag{3.3}$$

3.2.5 Transition Dynamics

State transitions follow a four-step sequence mirroring ASML's operational cycle. First, pipeline inventories age: $P_{i,t+1,k} = P_{i,t,k+1}$ for $k < l_i - 1$, moving items closer to completion. Completed production ($k = 1$) joins on-hand inventory: $I_{i,t+1} \leftarrow I_{i,t} + P_{i,t,1}$. Second, end-product demand $d_{i,t}$ is realized and fulfilled, reducing $I_{i,t}$ for $i \in \mathcal{C}_{end}$ and creating backlogs if inventory is insufficient. Third, new production starts $X_{i,t}$ consume upstream inventories per the BOM matrix A: each unit of $j \in \mathcal{D}_i$ reduces $I_{i,t}$ by one. Finally, all nodes update their production pipelines with new starts: $P_{i,t+1,l_i-1} = X_{i,t}$. These transitions are captured formally by:

$$s_{t+1} = f(s_t, X_{i,t}) \tag{3.4}$$

$$P_{i,t+1,j} \leftarrow P_{i,t,j+1} \quad \forall 1 \leq \ell < l_i - 1, \tag{3.4a}$$

$$P_{i,t+1,l_i-1} \leftarrow X_{i,t}, \tag{3.4b}$$

$$I_{i,t+1} \leftarrow I_{i,t} + P_{i,t,1} - \sum_{j \in \mathcal{D}(i)} X_{j,t} \quad (i \notin \mathcal{C}_{end}), \tag{3.4c}$$

$$I_{i,t+1} \leftarrow I_{i,t} + P_{i,t,1} - d_{i,t} \quad (i \in \mathcal{C}_{end}) \tag{3.4d}$$

3.3 Policy Definition and Evaluation

We consider a *stationary* policy $\pi : \mathcal{S} \to \mathcal{A}$ that assign to each state $s \in \mathcal{S}$ a valid action $\pi(s) \in \mathcal{A}(s)$. Under such a policy, the system evolves according to

$$A_t = \pi(S_t), \qquad S_{t+1} \sim f(S_t, A_t),$$

where f is the transition function introduced in Subsection 3.2.5.

We seek a policy π that **minimises the long-run average cost per period**:

$$J(\pi) = \lim_{T \in \mathbb{N}, T \to \infty} \frac{1}{T} \mathbb{E}_\pi \Big[\sum_{t=0}^{T-1} C(S_t, A_t) \Big],$$

where $C(S_t, A_t)$ is the one-period cost defined in (1.1). All solution-specific choices (e.g., finite-horizon simulation, warm-up length, episode count) are discussed in later chapters.

Chapter 4

Heuristic for a General Multi-Echelon System

To establish a heuristic for our general multi-echelon network, we reviewed two main approaches in the literature: GSM and SSM. GSM, although well-studied for general networks, imposes a strict service-level target at each echelon, which is unsuited for ASML's low-volume environment because any local shortage rapidly propagates further upstream or downstream. Instead, we adopt a SSM framework using an echelon base-stock policy, which is widely used and known to perform effectively in this setting.

Our method builds a full decision-making framework , which divides this chanter by (i) computing appropriate base-stock levels, (ii) defining how inventory is allocated across competing demands, and (iii) refining base-stock levels to account for capacity constraints using simulation-optimization. These components are organized as follows:

Together, these steps form our complete heuristic policy framework tailored to ASML's supply network. The performance of the resulting base-stock levels and allocation rules, and capacity-aware base-stock levels is assessed in later chapters.

4.1 Base-Stock Level Computation

While base-stock policies are conceptually straightforward, determining base-stock levels in networks with both convergent and divergent flows remains challenging. A notable reference is Rong et al. (2017), which proposes a decomposition approach that breaks the network into serial chains—one per end item—and reaggregates them by matching expected backorders, rather than summing base-stock levels. While their focus is on distribution systems, they briefly outline an extension to general networks. However, the treatment of convergent structures is limited. To guide the transformation of such assembly flows into serial representations, we refer to Rosling (1989) and Van Dijck et al. (2024), which offer more concrete structural insights for handling convergent parts of the network.

To operationalize this approach, we structure the base-stock computation in five steps:

1. Decomposition into serial systems;
2. Custom discrete distributions for low-volume demand;
3. Shang & Song heuristic to compute echelon base-stock levels for each serial system;
4. Conversion from echelon to local base-stock levels;
5. Backorder matching to unify base-stocks at shared nodes.
6. Calculate final echelon base-stock levels.

Each step is described in detail below.

4.1.1 Decomposing a General System

We transform the original multi-echelon network into multiple **serial systems** $w \in W$, one for each end item. Each of these serial systems represents a linearized view of the supply path leading to a specific end product. To construct them, we trace each end item's path back to the external supply, duplicating any convergent components across serial chains. This ensures that each resulting subsystem forms a single uninterrupted chain from source to end item, as described by Rosling (1989) and Van Dijck et al. (2024).

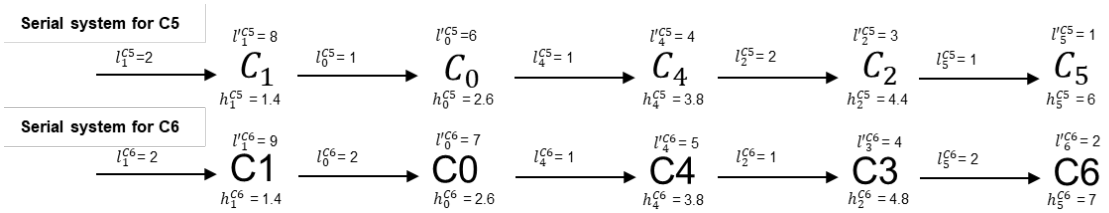


FIGURE 4.1: Decomposition of the general system into two serial chains—one for each end item (C_5, C_6)—with updated echelon lead times and adjusted holding costs.

In Figure 4.1, the general system from Figure 3.1 is decomposed into two purely serial chains, each terminating in one of the end products ($C5$ or $C6$). Each component i has an *echelon lead time* l'^w_i, which is the total time from the given component node to the end item in serial system w. To mirror the sequential progression of materials in a serial system, we sort the component nodes in descending order of l'^w_i (e.g., in the $C5$ subnetwork, $C1$ might have the largest l'^w_i, followed by $C0$, $C4$, and finally $C2$). If multiple nodes share the same l'^w_i, we *combine* them into a single node in the serial system, summing their holding costs and demands.

The serialized production lead time l^w_i for a component node is obtained by subtracting the echelon lead time of its immediate successor from its own. For example, if $l'^{C5}_1 = 8$ and $l'^{C5}_0 = 6$, then $l^{C5}_1 = 8 - 6 = 2$. Holding costs in the serialized chain are updated sequentially from upstream to downstream. At each node, the adjusted holding cost h^w_i is computed by adding its original holding cost, subtracting the original costs of any predecessors, and then subtracting the adjusted cost of the previous node. For instance, $h^{C5}_0 = 1.4 + 1.2 = 2.6$, and $h^{C5}_4 = 2.6 - 1.4 + 2.6 = 3.8$.

This serialized transformation enables the application of well-established single-product inventory heuristics, such as those by Shang and Song, while respecting the topology of the original network. It is the first key step in our overall base-stock policy computation framework.

4.1.2 Low-Volume Demand Distributions

Each end item inherits a custom discrete demand distribution, which can be a discrete Erlang or other empirically fitted approximation suitable for low-volume, high-tech manufacturing. This aligns with Van Dijck et al. (2024). Note that Rong et al. (2017) use Poisson demand however, the method of Shang & Song (2003) does not require Poisson per se; they only need to evaluate or invert $\Pr(D \leq s)$ and $\mathbb{E}[(D-s)^+]$. Thus, our custom discrete distributions easily fit into this framework.

4.1.3 Echelon Base-Stock Computation via Shang & Song (2003)

Within each new **serial system** w, we determine echelon base-stock levels for each component node c using the *Shang & Song* heuristic, which approximates optimal policies in uncapacitated serial supply chains. Let $\widetilde{D}_{i_w}$ be the echelon leadtime demand for node i in subsystem w. Define b_{ℓ_w} as the backorder penalty for end item ℓ_w, and let H_i be the echelon holding cost at node i, where $H_i = h_i - h_{\mathcal{U}(i)}$ and $H_0 = h_0$. Then:

- Case 1: i is the end-item node in subsystem w.

$$S_{i_w}^{\text{ECH}} \;=\; F_{\widetilde{D}_{i_w}}^{-1}\Big(\frac{b_{\ell_w} + \sum_{j\in A(0,\mathcal{U}(i))} H_j}{b_{\ell_w} + \sum_{j\in A(0,i)} H_j}\Big).$$

- Case 2: i is an internal node (non-end) in subsystem w.

$$S_{i_w}^{\text{ECH}} \;=\; \frac{1}{2}\Big[G_{\widetilde{D}_{i_w}}^{-1}(\alpha_1) \;+\; G_{\widetilde{D}_{i_w}}^{-1}(\alpha_2)\Big],$$

  where

$$\alpha_1 \;=\; \frac{b_{\ell_w} + \sum_{j\in A(0,\mathcal{U}(i))} H_j}{b_{\ell_w} + \sum_{j\in A(0,i)} H_j} \quad \text{and} \quad \alpha_2 \;=\; \frac{b_{\ell_w} + \sum_{j\in A(0,\mathcal{U}(i))} H_j}{b_{\ell_w} + \sum_{j\in A(0,\ell_w)} H_j}.$$

Here, $A(0,\mathcal{U}(i))$ is the set of nodes from the external supply up to immeditate upstream nodes for i $U(i)$, and $A(0,\ell_w)$ is the set of nodes from the external supply up to the end item ℓ_w. The function $G_{\widetilde{D}_{i_w}}^{-1}$ is a continuous approximation to the discrete inverse CDF for $\widetilde{D}_{i_w}$, ensuring we can handle non-integer or stepwise demands.

4.1.4 From Echelon to Local Base-Stock

Once we obtain echelon base-stock levels $S_{i_w}^{\text{ECH}}$ for each node i in serial system w, we compute the local base-stock levels $s_{i_w}^{\text{S-LOC}}$ using the transformation proposed by Rong et al. (2017). We use the prefix S-LOC to indicate that these levels are **serial-system-based** local base-stock levels:

$$s_{i_w}^{\text{S-LOC}} = S_{i_w}^{\text{ECH}} - S_{j_w}^{\text{ECH}}, \quad \text{where } j_w \text{ is the successor of } i_w \text{ in serial system } w.$$

This transformation is well-suited to divergent distribution systems—like those emphasized in their work—where each node has a unique downstream path. However, in general supply networks with convergent flows, as considered in our setting, the serialized representation may assign successors that are not directly connected in the original network. For example, in the serial system for end item $C5$ (see Figure 4.1), node $C4$ appears upstream of both $C2$ and $C5$, even though it only supplies $C5$ in the actual network.

To preserve consistency with the original network topology, we define $s_{i_w}^{\text{G-LOC}}$, a **general-system-based** local base-stock level, by referencing each node's actual immediate successor in the general system, as long as that successor lies on the path to end item ℓ_w:

$$s_{i_w}^{\text{G-LOC}} = S_{i_w}^{\text{ECH}} - S_{j_w}^{\text{ECH}}, \quad \text{where } j_w \in \mathcal{D}(i_w) \cap A(i_w, \ell_w),$$

where $\mathcal{D}(i_w)$ is the set of immediate successors of i_w in the general network, and $A(i_w, \ell_w)$ is the set of nodes on the directed path from i_w to end item ℓ_w.

This ensures that local base-stock levels reflect the true flow of materials toward the end item under consideration, especially in regions of the network with convergence. Importantly, this adjustment complements the serial approach: in purely divergent systems, where each node has a single downstream path, our method reduces to the original rule.

Finally, for end-item nodes, we set $s_{i_w}^{\text{S-LOC}} = s_{i_w}^{\text{G-LOC}} = S_{i_w}^{\text{ECH}}$.

4.1.5 Backorder Matching for Aggregation

After computing local base-stock levels $s_{i_w}^{\text{LOC}}$ in each subsystem w, some nodes i appear in multiple subsystems and thus have multiple values. But in the actual network, each node can only hold one physical base-stock level, s_i. Simply summing or taking the maximum of these values ignores **risk pooling**—the fact that high demand for one product may be offset by low demand for another. To account for this, Rong et al. (2017) propose **backorder matching**: instead of matching stock levels, we match their effect on expected backorders.

Thus, the key idea behind **backorder matching** is to align the expected backorders in the full network with the sum of expected backorders across all serial subsystems in which a node i appears.

In each subsystem w, we compute a local base-stock level $s_{i_w}^{\text{LOC}}$ for node i, and evaluate its expected backorders as:

$$\mathbb{E}[(D_{i_w} - s_{i_w}^{\text{LOC}})^+],$$

where D_{i_w} is the lead time demand for node i associated with end item w. Importantly, these demands are always calculated using the **general network's local lead times**, even when working within serialized subsystems.

To determine a unified physical base-stock level s_i for node i, we match the total expected backorders in the general network to the sum of subsystem-level backorders:

$$s_i = \min \left\{ s \in \mathbb{Z}_{\geq 0} \,\middle|\, \mathbb{E}[(D_i - s)^+] \leq \sum_{w: i \in w} \mathbb{E}[(D_{i_w} - s_{i_w}^{\text{LOC}})^+] \right\}.$$

Here, both D_i and D_{i_w} use the same lead time parameters but differ in scope. D_i reflects the full network's demand aggregated across all end items, while D_{i_w} captures demand routed only to end item ℓ_w. The aggregation preserves consistency with the original network while using the serial simplifications.

4.1.6 Final Echelon Base-Stock Levels

Up to this point, we've described how to compute unified base-stock levels s_i by performing backorder matching on local base-stock levels $s_{i_w}^{\text{LOC}}$, where the demand D_{i_w} is evaluated using general-network lead times. These local levels can be derived from either general-system-based inputs $s_{i_w}^{\text{G-LOC}}$, or serial-system-based ones $s_{i_w}^{\text{S-LOC}}$, resulting in two distinct sets of unified local base-stock levels:

- $s_i^{\text{G-MATCH}}$: based on $s_{i_w}^{\text{G-LOC}}$,
- $s_i^{\text{S-MATCH}}$: based on $s_{i_w}^{\text{S-LOC}}$.

An alternative, more serialization-aligned modeling choice would be to compute D_{i_w} using **serial-system lead times**, rather than those from the general network. In that case, both the local base-stock levels and their corresponding expected backorders would be fully aligned with the structure and timing of each serial subsystem. This would produce yet another variant of local base-stock levels: $s_i^{\text{FullS-MATCH}}$.

- $s_i^{\text{FullS-MATCH}}$: based on $s_{i_w}^{\text{S-LOC}}$ and serial lead times.

Since we ultimately operate under an **echelon base-stock policy**, each variant of local base-stock levels is converted into its corresponding echelon form:

$$S_i = s_i + \sum_{j \in \mathcal{D}_i} S_j, \quad \forall i \in \mathcal{C}$$

where $\mathcal{D}_i$ is the set of immediate successors of node i in the general network. The recursion begins at the end-items and proceeds upstream, ensuring each echelon level captures all downstream inventory needs.

Finally, due to the discrete and low-volume nature of demand, we cannot simply round final base-stock levels at the end of the pipeline. Instead, we track both the rounded-up and rounded-down versions throughout the Shang and Song initial computation, backorder matching and echelon transformation steps. As a result, we obtain in total six variants of final echelon base-stock levels. These variants are summarized in Table 4.1, and later evaluated in the experimental section.

TABLE 4.1: Variants of unified *echelon* base-stock levels under different modeling assumptions

Type	**Notation**	**Lead Times**	**Characteristics**
General-system-based	$S_i^{\text{G-MATCH}\uparrow}$ $S_i^{\text{G-MATCH}\downarrow}$	General	Aligned with full network structure and timing
Hybrid Serial-based (Rong et al., 2017)	$S_i^{\text{S-MATCH}\uparrow}$ $S_i^{\text{S-MATCH}\downarrow}$	General	Serial structure with general-network timing
Fully serial-based	$S_i^{\text{FullS-MATCH}\uparrow}$ $S_i^{\text{FullS-MATCH}\downarrow}$	Serial	Based entirely on serialized structure and timing

Clarifying the Modeling Variants. The different base-stock variants introduced above stem from how we compute local base-stock levels within each serial subsystem, and how we later unify them. The central distinction lies in whether the decomposition respects the structural and temporal features of the original network. The **general-system-based** variant ($S_i^{\text{G-MATCH}}$) uses the actual immediate successors from the original network to define local base-stock levels, and applies general-network lead times when evaluating backorders. This approach ensures consistency with the true material flow, particularly in convergent areas where a component may feed multiple downstream nodes in different ways.

By contrast, the **serial-system-based** variant ($S_i^{\text{S-MATCH}}$), inspired by Rong et al. (2017), assumes that local successors follow the structure of the serial chain. While this is valid for divergent distribution networks, it can misrepresent flow dependencies in general systems where serialized successors are not always real successors. The **fully serial-based** variant ($S_i^{\text{FullS-MATCH}}$) goes a step further by also replacing lead times with those derived from the serial chain, thereby fully aligning structure and timing—but at the cost of deviating further from the true network logic.

Among these, the general-system-based approach remains the most aligned with the physical network and is therefore expected to yield more realistic stock targets. Nonetheless, all six variants (up/down rounded) are carried forward to the experiments in further chapters.

4.2 Allocation Methods for Operating the General System

While computing base-stock levels provides inventory targets for each node (how much to order and hold), **they do not specify how that inventory is used once it arrives. In fact, operating the network under demand uncertainty and capacity constraints requires effective allocation rules**. This is particularly important in *divergent supply networks*, where upstream resources must be distributed among competing downstream demands. As highlighted in the literature, poor allocation decisions can amplify shortages, degrade service levels, and raise total costs.

As reviewed in Chapter 2, allocation in divergent networks has been extensively studied in the context of one-warehouse-multi-retailer (OWMR) systems, but generalizations remain limited. Early approaches, such as FCFS and balance assumptions (Eppen, 1981), offer simplicity but lack robustness under demand asymmetries. More recent work—including projection-based heuristics (Axsäter, 1990), two-moment approximations (Graves, 1985), and adaptive rules (Stranieri et al., 2024)—attempts to improve scalability and fairness, yet none fully address the dynamic interdependencies of convergent-divergent structures like ASML's supply network.

4.2.1 Illustrative Example: Allocation in Divergent vs. General Networks

Figure 4.2 shows the difference in allocation complexity between a simple divergent system and a more realistic general network.

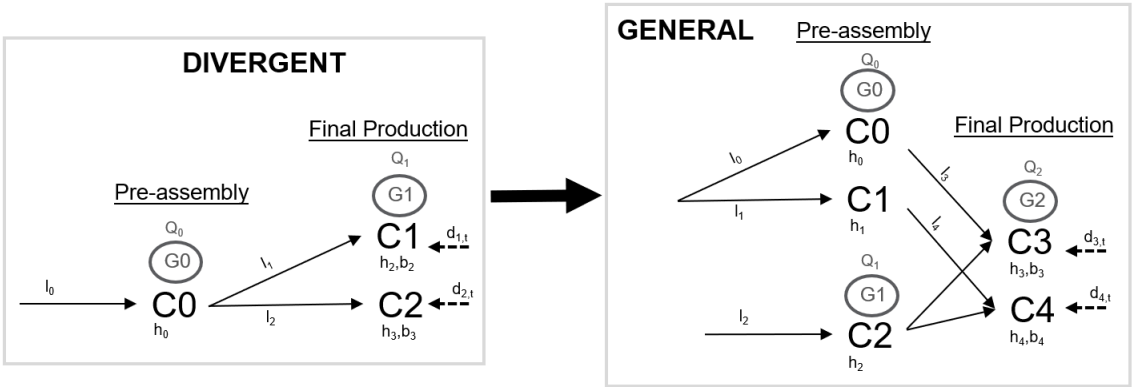


FIGURE 4.2: Allocation challenge in a divergent (left) and general (right) system.

In the divergent setting (left), warehouse C_0 supplies both end-products C_1 and C_2. When stock is limited, we must decide how to split it between them. This is a classic one-to-many allocation problem.

In the general system (right), end-products C_3 and C_4 depend on multiple upstream nodes (C_0, C_1, and C_2). Unlike the fully divergent system, their production is constrained by the **least available inventory** among their suppliers. For example, C_3 requires inputs from both C_1 and C_2, and its output is limited by the most constrained node. Similarly, allocation from C_2 must consider not only the demand of its direct successors but also whether C_0 and C_1 have enough supply to support C_3 and C_4. This type of indirect constraint propagation—where the availability at one node affects how useful allocations are downstream—is what makes allocation in general networks particularly challenging and underexplored.

It is important to mention that in general systems, instead of treating allocation locally at each predecessor, we adopt a global view: in each period, we consider the pool of all successors that share at least one common predecessor, and coordinate allocations across this group. This allows the system to make better use of shared resources and ensures fairness across tightly interconnected product paths.

Overview of Proposed Allocation Rules

To address the resource distribution challenges highlighted (see Section 4.2), we design and evaluate several allocation policies. These range from simple sequential heuristics like First-Come-First-Served (FCFS) and Randomized Sequential Allocation, to more sophisticated, fairness-aware dynamic strategies based on the water-filling (WF) principle. The water-filling methods further explore variants including prioritization by shortage, and incorporate mechanisms such as halting allocations, reserving inventory for constrained nodes, and realizing these reservations over time. Each method is detailed in the subsequent sections, accompanied by illustrative examples to clarify its operational mechanics and distinctions.

4.2.2 Baseline Allocation Methods

First-Come-First-Served (FCFS)

The **First-Come-First-Served (FCFS)** rule allocates inventory to successor nodes based on a predetermined, fixed sequence. For each successor, if all its requisite inputs are available and can process its demand, this is supplied in full. Otherwise, the algorithm skips to the next node. This continues until inventory is depleted or all successors are considered.

Example: Suppose node C_0,C_1 and C_2 have 3 units of inventory and must serve C_3 and C_4, in that order. If both require 2 units and both can be served, FCFS will allocate 2 units first to C_3, then 1 unit to C_4. If only 1 unit is available, C_3 gets it, and C_4 gets nothing.

While easy to implement, FCFS may be unfair, since the first nodes are consistently prioritized over others.

Randomized Sequential Allocation (RandomWF)

Inspired by Stranieri et al. (2024), this rule adds fairness by randomly permuting the order of successors in each period. One unit is allocated at a time to the first feasible node in the randomized sequence. The randomness is seeded to ensure reproducibility.

Example: With C_0, C_1 and C_2 serving C_3 and C_4, we randomly choose between successors. Suppose the random order is $[C_4, C_3]$ and inventory is 2 units. If both are feasible, 1 unit goes to C_4, then 1 to C_3. This prevents systematic bias over time.

Even though this method is relatively simple to implement, it is still myopic and does not consider base-stock targets or current inventories.

4.2.3 Water-Filling Allocation Strategies

Water-filling (WF) allocation strategies constitute a class of methods designed to distribute resources more equitably by prioritizing nodes with the greatest need. The guiding principle is analogous to water filling a container's lowest points. These methods typically rely on a quantified measure of "shortage" at each successor node to direct allocation decisions, aiming for a more balanced inventory distribution across the system.

Standard Water-Filling (Minimizing the Maximum Shortage) (WF)

The previous **Water-Filling** algorithm, in its basic form, prioritizes allocations to the feasible node experiencing the highest unmet demand, quantified by its shortage. A node is feasible if all its direct predecessors have non-zero available inventory. This approach draws inspiration from simpler divergent systems, where a common objective is to minimize the maximum shortage observed among all direct successors (Kaynov et al., 2024). However, in general systems, accurate distribution is hard to pre-calculate when successors depend on multiple, potentially constrained inputs. Consequently, our adaptation for general systems employs an iterative, unit-by-unit allocation process. At each iteration, one unit is given to the feasible node with the highest shortage.

The absolute shortage for node i is:

$$\text{shortage}_i = S_i - I_i^{\text{ECH}} - \text{allocated}_i$$

where S_i is base-stock, I_i^{ECH} is echelon inventory, and allocated_i (units assigned this period, initially zero) tracks current allocations. However, the use of absolute shortages can be potentially misleading when target base-stock levels (S_i) differ substantially among successor nodes. For instance, a shortage of 10 units might be relatively minor for a node with a target base-stock $S_i = 1000$ but critically urgent for a node with $S_i = 20$. The **Relative Water-Filling** normalizes shortage by S_i, reflecting proportional urgency:

$$\text{Relativeshortage}_i = \frac{S_i - I_i^{\text{ECH}} - \text{allocated}_i}{S_i} \quad (S_i > 0)$$

Example: C_0 supplies C_3, C_4.

- C_3: $S_3 = 5, I_3^{\text{ECH}} = 2 \implies$ Abs. $\text{shortage}_3 = 3$, Rel. $\text{shortage}_3 = 0.6$.
- C_4: $S_4 = 10, I_4^{\text{ECH}} = 6 \implies$ Abs. $\text{shortage}_4 = 4$, Rel. $\text{shortage}_4 = 0.4$.

This indicates that C_3 is proportionally further from achieving its target inventory level ($0.6 > 0.4$) and should be prioritized, despite C_4's larger absolute shortage. Therefore, 1 unit is allocated to C_3, and shortages are then updated for the next iteration.

Initial experiments indicate that the relative version consistently outperforms the absolute version. For this reason, we adopt the relative shortage as our default shortage measure and use it in all subsequent Water-Filling methods.

Full-Stop Water-Filling (WFFS)

Standard water-filling algorithm continues to allocate inventory to the feasible node with the highest shortage. However, if high-priority nodes are temporarily unfulfillable (due to missing inputs from any predecessor), the algorithm skips them and serves the next highest-shortage feasible nodes. This behavior can lead to perpetuating unfairness if certain high-priority nodes are persistently blocked. To address this, we propose the **Full-Stop Water-Filling** method, which adds a fairness safeguard and suspends allocations if the node with the greatest shortage cannot be serviced because of unavailable inputs. No other feasible nodes receive allocations, conserving inventory for when the high-priority node is again serviceable. This conservative strategy aims to prevent lower-priority nodes from consuming resources when a more urgent node, though temporarily unserviceable, exists.

Example: C_3 has the highest relative shortage for inventory from C_2 but also needs a component from C_0. If C_0 is out of stock, C_3 is infeasible. C_0 halts allocation, not serving C_4 even if feasible.

Water-Filling with Reserve (WFR)

Rather than stopping the allocation process entirely, the water-filling with reserve method introduces a more flexible response. When the node with the highest relative shortfall cannot be served due to missing input from some (but not all) of its predecessors, the algorithm does not skip the node or halt. Instead:

a) It **reserves** one unit of its inventory for the constrained node.

b) Allocation continues, with relative shortages recalculated considering allocated_i and now reserved_i:

$$\text{Relativeshortage}_i = \frac{S_i - I_i^{\text{ECH}} - \text{allocated}_i - \text{reserved}_i}{S_i}$$

This policy aims to ensure that reservations count toward meeting a node's target and influence subsequent allocation decisions. This method could potentially maintain fairness without halting the system and keeping high-priority nodes in focus even if they are only partially feasible. Reservations are considered only within the current allocation period.

Water-Filling with Reserve Realization (WFRR)

This method adds persistence to reservations. units reserved for a constrained node in a given period t are carried forward into the subsequent period $t+1$. So, at the start of period $t+1$, **Water-Filling with Reserve Realization** first attempts to "realize" these outstanding reservations: if all inputs for a reserved node are now available, the allocation is executed. This precedes any Water-Filling with Reserve logic in the current period for the remaining available inventory and any new or remaining demands.

Example: In period 1, we reserved a unit for C_3 (blocked by C_0). In period 2, if C_0 and C_2 are now available, we allocate to C_3 from the reservation before other period 2 allocations. This could help prevent the loss of intent behind reservations in Water-Filling with Reserve. However, it may limit responsiveness when previously low-priority nodes face sudden demand spikes, potentially delaying reallocation to more urgent needs in the current period.

Summary of Allocation Methods

The different allocation methods with their differentiating characteristics are summarized in Table 4.2.

Table 4.2: Features of Allocation Methods

Method	**Shortage-Based**	**Uses Reserves**	**Tracks Over Time**
FCFS	No	No	No
RandomWF	No	No	No
Water-Filling Variants:			
WF	Yes (Relative)	No	No
WFFS	Yes (Relative)	Halts System Instead	No
WFR	Yes (Relative)	Yes (Intra-period)	No
WFRR	Yes (Relative)	Yes	Yes (Inter-period)

4.3 Capacity-Aware Base Stock Optimization

Group-based capacity constraints, common at ASML, are mostly overlooked in inventory literature. Traditional methods like serial shortfall heuristics (Huh et al. (2016)) fail with overlapping capacity groups, and most DRL approaches ignore shared capacities. No method currently combines capacity-aware planning and discrete control in multi-echelon networks. This study employs a simulation-optimization heuristic to refine base stock levels in multi-echelon supply chain networks subject to capacity constraints. The method aims to minimize total system costs by iteratively adjusting base stock levels, guided by the degree to which nodes are capacity-constrained.

The core of the methodology is an iterative, greedy search, which is outlined in Algorithm 1. It begins with an initial set of base stock levels (lines 4–6). The performance of these levels, primarily in terms of average total system cost, is assessed through discrete-event simulation over multiple trajectories. Concurrently, a "Capacity Constraint Ratio" (CCR) is computed for each node. This CCR quantifies the proportion of time periods in which a node's desired replenishment order (based on its current accepted base stock level and echelon inventory) could not be fully met due to insufficient capacity in its associated capacity group. A period is classified as capacity-constrained only when the group's residual capacity is not only insufficient to fulfill the node's order but also less than or equal to the minimum inventory available from the node's immediate predecessors.

The optimization proceeds iteratively. In each iteration (line 10), the node with the highest CCR not yet unsuccessfully adjusted is selected (line 11). This is the most bottlenecked node. The base stock level of this bottleneck node is then tentatively increased by a small, fixed amount (lines 15–16). The system's performance with this modified base stock level is re-evaluated via simulation (lines 17). When evaluating this tentative change, the policy under test uses the new, modified base stock levels, but the CCR calculation continues to determine desired orders based on the previously accepted base stock levels to ensure a consistent measure of constraint. If this change improves system cost (line 18), it is accepted (lines 19–22), and the process may restart by considering all nodes again. Otherwise, the node is marked as unsuccessfully tried (line 23). If all nodes have been tried without success (line 24), the algorithm terminates.

The algorithm stops when no further improvements are found, constraints are minor, or a maximum number of iterations is reached (line 25). The best-performing base stock levels are returned (line 28).

Algorithm 1 Guided Base Stock Optimization

Input: Network_Configuration, Initial_Base_Stock_Levels (BSL), Simulation_Parameters, Max_Iterations
Output: Optimized_BSL, Best_Mean_System_Cost

current_BSL ← Initial_Base_Stock_Levels
best_BSL ← current_BSL
(best_mean_cost, current_node_CCR) ← EvaluateConfiguration(current_BSL, Sim_Params, Network_Config)
iteration ← 0
nodes_unsuccessfully_tried ← [ ]
while iteration < Max_Iterations **do**
    iteration++
    Select most_constrained_node_idx (with highest CCR value) from nodes **not in** nodes_unsuccessfully_tried
    if no such node exists **then**
        break ▷ Convergence or no significant constraints
    end if
    tentative_BSL ← current_BSL
    tentative_BSL[most_constrained_node_idx]++
    (evaluated_cost, evaluated_CCR) ← EvaluateConfiguration(tentative_BSL, Sim_Params, Network_Config)
    if evaluated_cost < best_mean_cost **then**
        best_mean_cost ← evaluated_cost
        best_BSL ← tentative_BSL
        current_BSL ← tentative_BSL
        current_node_CCR ← evaluated_CCR
        nodes_unsuccessfully_tried ← [ ]
    else
        Add most_constrained_node_idx to nodes_unsuccessfully_tried
        if AllNodesTried(nodes_unsuccessfully_tried, num_nodes) **then**
            break ▷ No improvement possible this round
        end if
    end if
end while
return best_BSL, best_mean_cost

Chapter 5

Deep Reinforcement Learning Approach

Chapter 2 discussed how Deep Reinforcement Learning (DRL) addresses limitations of heuristic inventory control in complex environments. DRL learns directly from interactions, adapting to stochastic dynamics and interdependencies not easily captured analytically. However, common DRL, like Deep Q-Networks (DQN), struggles with stability and scalability in low-volume inventory settings (Oroojlooyjadid et al., 2021). Actor-critic methods such as A3C and PPO (Gijsbrechts et al., 2022; Van Hezewijk et al., 2023) also underperform compared to simple policies, due to inefficiencies and sensitivity issues. To overcome these, new DRL approaches integrate structural information. Programmable Actor Reinforcement Learning (PARL) uses mixed-integer optimization for action selection (Harsha et al., 2025), but its high computational cost and inconsistency in outperforming heuristics remain drawbacks. Conversely, Deep Controlled Learning (DCL) emerges as a model-based, sample-efficient alternative for high-variability inventories, avoiding value function approximation and treating policy learning as a classification task over simulation-based state-action pairs. It shows sub-0.2% optimality gaps with reduced training time and tuning effort (Van Dijck et al., 2024; Temizöz et al., 2023).

This chapter describes the implementation of the DCL method, Section 5.2 explains the DCL methodology and core principles, then Section 5.3 covers node-wise decomposition for simpler decision-making, and finally, Section 5.4 details the neural network architecture, training/testing, and feature extraction.

5.1 DRL Methodology: Deep Controlled Learning (DCL)

Deep Controlled Learning (DCL) acts as a model-centric framework for approximate policy iteration, converting reinforcement learning tasks into a supervised learning paradigm. Rather than calculating expected value functions for an entire spectrum of states and actions, DCL determines the optimal action per state by running controlled trials on a chosen group of potential actions under uniform stochastic conditions. These trials directly compare action results, enabling DCL to label each state optimally without the noise, bias, and instability of temporal difference methods.

As described in Temizöz et al. (2023), the DCL algorithm operates by starting from a pool of N sampled states, where DCL revisits each state under multiple exogenous scenarios M (e.g., demand realizations) and simulates trajectories of fixed length H. For each candidate action $a \in \mathcal{R}_g(s)$ in a state ($\mathcal{R}_g(s) = \{a^{(1)}, \ldots, a^{(m)}\} \subseteq \mathcal{A}(s)$), the algorithm runs M independent roll-outs of length H under common random numbers (CRN) (to

control variance between cost estimates) and computes the empirical cost-to-go

$$\widehat{C}_g(s,a) = \frac{1}{M}\sum_{m=1}^{M}\sum_{t=0}^{H-1} C_t^{(m)}(s,a),$$

where $C_t^{(n)}(s,a)$ is the one-step cost for policy-iteration g at time t in the m^{th} simulation. Then labels the state with the action incurring the lowest cost $a^\star(s) = \arg\min_{a\in\mathcal{R}_K(s)} \widehat{C}_K(s,a)$. In addition to CRN, DCL utilizes a lightweight bandit approach known as Sequential Halving to focus on the most promising actions ($\lceil \alpha\,|\mathcal{R}_g(s)|\rceil, \alpha \in (0,1)$) and identify the action with the least cost ($a^\star(s) = \arg\max_{a\in\mathcal{R}_K(s)} \widehat{Q}_K(s,a)$). This method allows for efficient use of simulation budgets by avoiding exhaustive simulations of every action across all scenarios and iterations.

The collected $(s, a^\star)$ pairs form a dataset $\mathcal{D}_g$ that trains a neural network policy f_θ through cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{(s,a^\star)\in\mathcal{D}_g} \mathbf{1}_{a^\star(s)}^\top \log f_\theta(s),$$

optimised by stochastic gradient descent (learning rate η, mini-batch size, early-stop patience)

Throughout, DCL adopts a finite–horizon perspective: each training example is generated by first running the system for a "warm-up" of L periods—so that the state s reflects realistic, steady-state behavior—and then simulating an additional H steps to estimate the cumulative cost-to-go from a given state s. In the very first policy iteration or generation (for DCL), these roll-outs are driven by a fixed *rollout policy*, which here is the base-stock heuristic from Chapter 4. Once N such $(s, a^\star)$ labels have been collected, the neural policy f_θ is retrained to minimize classification error. In each new generation, the policy trained in the previous one is used to guide the rollouts, replacing the initial heuristic. This self-labeling helps the model refine its own decisions over time. However, if early policies are poor, this feedback loop can reinforce suboptimal behavior. This procedure—sample N states, label them via H-step cost roll-outs after an L-period warm-up, update f_θ—is repeated for G generations, yielding progressively improved decision rules. Table 5.1 provides the suggested DCL parameter settings by Temizöz et al. (2023).

This approach presents multiple advantages in inventory management contexts. Firstly, by circumventing the need to assess long-term value functions in noisy settings, DCL bypasses the instability common in actor-critic models and the overestimation bias found in Q-learning. Secondly, it promotes structured exploration by explicitly comparing plausible actions under consistent external conditions, enhancing resilience to randomness. Thirdly, it simplifies policy design by framing the learning challenge as a straightforward classification problem. However, extending DCL to general supply networks poses notable challenges, especially related to the combinatorial escalation of the action space as both nodes and their capacity for orders grow.

To keep computational demands manageable and sustain learning efficacy, we integrate two core modifications: (1) a sequential decision-making framework to break down global actions into node-level decisions (Section 5.3) as done in Van Dijck et al. (2024), and (2) a structured output space that encapsulates feasible actions in a succinct and comprehensible manner (Section 5.4). These enhancements enable DCL to scale effectively in real-world supply chain scenarios while preserving its core strengths in sample efficiency and policy refinement.

TABLE 5.1: Hyperparameter settings used in DCL (based on Temizöz et al. (2023))

Parameter	**Value**
Roll-out horizon (H)	40
Warm-up length (L)	100
Exogenous scenarios per action (M)	1000
Sampled states per generation (N)	5000
Policy iterations (G)	3
Candidate set size (m)	20
Halving ratio (α)	0.5
Neural network layers	(256, 128, 128, 128)
Mini-batch size	64
Optimizer	Adam
Learning rate (η)	0.001
Early stopping patience	10 epochs

5.2 Network Decomposition and Sequential Decision Process

One of the main challenges in applying DCL to a general supply network lies in managing the explosive growth of the action space. An approach, where ordering decisions for all components are made simultaneously, quickly becomes computationally infeasible as the number of components increases. To mitigate this, Van Dijck et al. (2024) proposes to restructure the decision-making process into a **sequential, component-by-component framework**, which significantly reduces the action space at each decision step while preserving the integrity of the system dynamics.

In this sequential approach, decisions are made individually for each component node $i \in \mathcal{C}$ following a predetermined topological order $\mathcal{K}$, typically progressing from upstream to downstream. This structure substantially reduces the dimensionality of the action space and is key to making DCL tractable in large networks. While $\mathcal{K}$ is primarily a modeling tool to structure the decision flow, it implicitly introduces a degree of prioritization in group-based capacity usage: nodes earlier in the sequence may consume more of the shared capacity, which can bias learning in tightly constrained capacity groups. Although this is a known limitation of sequential decision-making, simultaneously optimizing decisions across all nodes would be computationally infeasible due to the combinatorial explosion of joint actions.

Moreover, we extend the approach of Van Dijck et al. (2024) to accommodate flows, where multiple successors compete for inventory, and propose a different treatment. In these cases, we propose to perform allocation simultaneously for all competing successors using the chosen rule (see Section 4.2). This allocation is executed during the decision step of the *first* node in the sequence that has multiple successors, and we ensure that the allocated quantities are consistent across the group and respect the shared capacity and supply available inventory. When the sequence proceeds to the remaining competing nodes, their decisions are based on the allocation result already determined. It is important to mention that during rollouts, if DCL begins with a competing successor node, the full allocation cannot be applied upfront. We resolve this by treating the current node as the starting point of a reduced competing set and reapplying the allocation rule to the remaining successors based on residual supply and capacity. This preserves the intent of the simultaneous allocation logic.

Even though this reduces the bias of sequential ordering and DCL is trained to recognize and adapt to this structure, the sequential structure still induces a mild priority effect in group capacity usage, and performance in tightly constrained settings may still reflect the implicit ordering advantage of earlier nodes.

In the subsequent sub-sections, the state is expanded, and we present both the sub-decision dynamics as well as the revised event transitions.

5.2.1 Augmented State Definition

To support sequential decision-making, we extend the period-level system state s_t by introducing two additional elements:

- A **decision pointer** $K \in \{0, 1, \ldots, |\mathcal{C}|\}$, indicating the current component node for which a decision is being made.
- A vector of **remaining capacities** $\{R_{g,t}\}_{g\in\mathcal{G}}$, capturing the available production capacity for each capacity group at the current decision point.

Formally, the full augmented state at decision step K is:

$$\boxed{s_t = [\{I_{i,t}, P_{i,t,1}, \ldots, P_{i,t,l_i-1}\}_{i\in\mathcal{C}},\ \{R_{g,t}\}_{g\in\mathcal{G}},\ K]\,.}$$

At the start of each period, the pointer is initialized to $K = 0$ and all group capacities are fully available, i.e., $R_{g,t} = Q_g$ for each $g \in \mathcal{G}$.

5.2.2 Sub-decision Dynamics

At each sub-decision step K, the agent selects an order quantity $X_{K,t} \in \mathbb{N}_0$ for component node K, subject to two feasibility conditions:

1. **Material Availability**:

   $$X_{K,t} \le I_{j,t} \quad \forall j \in \mathcal{U}(K),$$

   where $\mathcal{U}(K)$ denotes the set of immediate upstream components required to produce node K.

2. **Capacity Availability**:

   $$X_{K,t} \le R_{g(K),t},$$

   where $g(K)$ is the capacity group associated with component K.

Upon selecting $X_{K,t}$, the state is immediately updated as follows:

$$\begin{aligned} s_{t+1} &= f(s_t, X_{K,t}) \\ P_{K,t,\,l_k-1} &\leftarrow X_{K,t}, \\ I_{j,t} &\leftarrow I_{j,t} - X_{K,t}, \quad \forall\, j \in U(K), \\ R_{g(K),t} &\leftarrow R_{g(K),t} - X_{K,t}, \\ K &\leftarrow K + 1. \end{aligned} \tag{5.3}$$

Here, we add production start to pipeline tail for component K, consume upstream inventories, decrease the remaining capacity of group $g(K)$, and advance the pointer to the next component. This process is repeated until all components $K = 1, \ldots, |\mathcal{C}|$ have received a production start decision.

5.2.3 Period-End Event Transition

After completing all component decisions for the current period, the system undergoes a single **event transition** $s_t \to s_{t+1}$. In this case, we adapt the transition dynamics function of Chapter 3 not only to be aware of the dynamics that have happened already in the sub-decision dynamics, but also to advance the production pipelines, update on-hand inventories, and realize external demand where applicable.

The event transition dynamics are defined as:

$$
\begin{aligned}
s_{t+1} &= f(s_t, [d_{i,t}]_{i \in \mathcal{C}_{end}}]) \\
P_{i,t+1,\ell} &= P_{i,t,\ell+1} && \forall i \in \mathcal{C}, \quad 1 \le \ell < l_i - 1, \\
P_{i,t+1,l_i-1} &= 0 && \forall i \in \mathcal{C}, \\
I_{i,t+1} &= \begin{cases} I_{i,t} + P_{i,t,1} - d_{i,t}, & \text{if } i \in \mathcal{C}_{\text{end}}, \\ I_{i,t} + P_{i,t,1}, & \text{otherwise}, \end{cases} && \\
R_{g,t+1} &= Q_g && \forall g \in \mathcal{G}, \\
K &\leftarrow 0.
\end{aligned}
\tag{5.4}
$$

In the last two equations, we reset all $R_{g,t}$ to full capacity Q_g for the new period, and the pointer K resets to 0.

5.3 Neural Network Architecture

The core of our Deep Controlled Learning approach is a neural network trained to approximate the optimal policy learned via supervised classification over state-action pairs. This section details the design of the network architecture, including the input representation, output construction, and training pipeline. The architectural choices are driven by the need for scalability, expressiveness, and compatibility with the sequential decision structure introduced in Section 5.3. In this section, we present the input and output representations, feature engineering, the training pipeline, and some implementation details.

5.3.1 Input Representation

Each input to the neural network corresponds to the current sub-decision state s_t when a decision must be made for a specific component node. The input captures the complete status of the supply network, the remaining production capacities, and the pointer to the current decision node. Formally, the input vector includes:

- On-hand inventory $I_{i,t}$ for all components $i \in C$
- Pipeline inventory $P_{i,t,k}$ for all $i \in C$ and $1 \le k \le l_i$
- Group capacity availability $Q_{g,t}$ for all groups $g \in G$
- Pointer value for the current decision node
- (Optionally) additional engineered features can be included, see in Subsection 5.3.4.

This design ensures that the network is fully aware of the current system configuration while focusing attention on the specific component node being considered.

5.3.2 Output Construction

We implement a shared neural network whose output layer spans the concatenated action spaces of all nodes: if node n has $|A_n|$ feasible order quantities, the output dimension is $\sum_n |A_n|$. We assign a contiguous slice of this output vector to each node via cumulative offsets: node 0 occupies indices $[0, |A_0|-1]$, node 1 occupies $[|A_0|, |A_0|+|A_1|-1]$, and so on. Since order decisions are discrete and bounded by the node's capacity, the output layer consists of a categorical softmax distribution over all feasible order quantities for that node. During inference time, given the current decision pointer K, we apply a dynamic mask that zeroes out probabilities outside the slice for node K, and the action with the highest probability is chosen.

One key challenge is that the total number of output classes can grow rapidly with the number of nodes and capacity sizes. For example, with 10 nodes each having 15 feasible actions, the output layer already spans 150 nodes. This could pose challenges in training efficiency and over- (for smaller systems) and under- (for larger systems) fitting risk, which we try to mitigate through decomposition by limiting next nodes options based on previously made ones, by using simulatanesuly making divergent nodes' actions with the heuristic, and by having a strong rollout heuristic that supplies quality (state, best-action) labels to steer the learning toward high-impact actions and avoiding low-value or infeasible ones.

5.3.3 Neural Network Training

During training, we treat the network as a classifier over discrete order quantities for the active node. Each training example consists of:

- An input vector $\phi(s)$ encoding the system state (including any global and node-specific features) together with a node identifier K, indicating which node's decision is to be made.
- A labeled "best" action a^* for node K, obtained from heuristic or previous-generation neural network rollouts.

By training on many $(\phi(s), K, a^*)$ samples, the network learns to predict, for each node in each state, which discrete order quantity minimizes expected cost. At evaluation time, given a new state and node K, we repeat the masking and softmax steps and select the action with highest predicted probability among the feasible set for K.

5.3.4 Feature Extraction

To support decision-making within the MDP framework, a structured set of features is extracted from the system state at each decision epoch. These features capture both global system information and local node-specific dynamics. Formally, given the system state s_t, the feature vector $\phi(s_t)$ is constructed as follows:

The features are organized into five main groups, each serving a specific purpose in capturing the dynamics of the supply chain:

- **Basic State:** Essential state information for decision-making:
  - K_t – Current Node
  - $\{I_{i,t}\}_{i \in \mathcal{C}}$ – Inventory Levels
  - $\{P_{i,t,k}\}_{i \in \mathcal{C}, l=1,\ldots,l_i}$ – Production Levels per Capacity Group

- **New State:** Aggregate measures for state representation:
  - $\sum_{l=1}^{l_i} P_{i,t,l}$ – Total Production in Node
  - $\{P_{i,t,1}\}_{i\in\mathcal{C}}$ – Inventory Arriving in Next Period
  - $\{I_{i,t}^{\mathrm{ECH}}\}_{i\in\mathcal{C}}$ – Echelon Inventory Levels
- **Capacity Awareness:** Current utilization of capacity:
  - $\mathrm{CapUtil}_{g(K_t)} = 1 - \frac{q_{g(K_t),t}}{Q_{g(K_t)}}$ – Group Capacity Utilization
  - $|\mathcal{C}_{g(K_t)}|$ – Components Sharing Group
- **Structural Awareness:** Network configuration insights:
  - $|\mathcal{U}(K_t)|$ – Direct Predecessors
  - $|\mathcal{D}(K_t)|$ – Direct Successors
  - $\mathbf{1}\{|\mathcal{D}(K_t)|> 1\}$ – Divergence Indicator
- **Bottleneck Awareness:** Shortage dynamics capture:
  - $\mathrm{RelShort}_{K_t} = \max\left(0, \frac{S_{K_t} - I_{K_t,t}^{\mathrm{ECH}}}{S_{K_t}}\right)$ – Current Node Shortage
  - $\mathrm{MaxReceivable}_{K_t} = \min\left(q_{g(K_t),t}, \min_{j\in\mathcal{U}(K_t)} I_{j,t}\right)$ – Receivable Inventory Limit
  - $\mathrm{AggRelShort}_{K_t} = \sum_{j\in\mathcal{D}(K_t)} \mathrm{RelShort}_j$ – Aggregated Shortages
  - $\mathrm{BottleneckRatio}_{K_t} = \frac{\mathrm{MaxReceivable}_{K_t}}{\mathrm{AggRelShort}_{K_t}}$ – Capacity-Demand Ratio
  - $\{\mathrm{Backlog}_e\}_{e\in\mathcal{C}_{end}}$ – End-Product Backlogs
  - $\{\mathrm{RelShort}_j\}_{j\in\mathcal{D}(K_t)}$ – Per-Node Shortages

Zero-padding handles variable network topology for $\mathrm{RelShort}_j$ and $\mathrm{Backlog}_e$, ensuring fixed-dimensional inputs for learning stability. The best subset of feature is defined in Section 6.2.1

Chapter 5 Conclusion. Chapter 5 introduced DCL as a supervised-learning–style DRL method tailored to multi-echelon inventory systems. Instead of estimating value functions, DCL labels states by comparing candidate actions through controlled rollouts under common random numbers, then trains a classifier to predict the best action. This avoids instability of actor-critic or Q-learning approaches in low-volume, discrete-action settings. To manage combinatorial explosion, we employ a sequential decision structure (similar to Van Dijck et al. (2024)): at each period, nodes follow a fixed topological order but also adapt it for divergent parts, where we apply simultaneous allocation at the first-occurring node. The neural network uses a shared architecture: inputs combine global system state and node-specific features (which we also extracted), and the output layer concatenates the action spaces of all nodes. During training, each example is a (state + node ID, best action) pair, and at evaluation time we mask out irrelevant slices so the softmax focuses only on feasible actions for the current node. We considered scalability issues, where the input layer size increases as the number of nodes and their lead times grow, and the output increases as the number of nodes and capacity constraints grow.

Chapter 6

Computational Experiments

This chapter describes the experimental design structured to systematically evaluate and compare the heuristic and DRL-based methods developed in previous chapters.

This chapter consists of three stages defined as follows:

1. **Heuristic Benchmarking:** Identify top base-stock configurations and allocation rules on uncapacitated 7-node network.
2. **DCL Tuning:** Determine the most effective DCL setup by varying state features, planning horizon, and neural network architecture under controlled conditions.
3. **Comparative Evaluation (Heuristics vs DCL):** Contrast the tuned DCL policy against the best heuristics in (i) small uncapacitated, (ii) small capacitated, and (iii) full seven-node capacitated networks, revealing where data-driven decisions improve upon static rules.

To approximate the infinite-horizon average cost of a policy π, we run N independent simulations, each of length $W + H$. We discard the first W periods as a warm-up, then compute the per-episode average cost over the next H steps:

$$\mathcal{C}_{W,H}(\tau;\pi) \;=\; \frac{1}{H}\sum_{t=W}^{W+H-1} C(S_t,\pi(S_t)).$$

The policy's empirical average cost is

$$\widehat{J}_{W,H}(\pi) \;=\; \frac{1}{N}\sum_{n=1}^{N}\mathcal{C}_{W,H}(\tau^{(n)};\pi),$$

which (as $N \to \infty$ and with large W, H) approximates the true long-run average cost. Lower $\widehat{J}_{W,H}(\pi)$ means a better (cost-minimizing) policy. In our experiments we use $N = 1000$ trajectories of 1000 periods each, with a 60-period warm-up. Warmup definition can be found in Appendix A.1

Moreover, the following elements are common across all stages and defined once here for brevity.

Experimental Networks. We consider three progressively complex network topologies. The 5-node system (Figure 6.1, left) includes three upstream buffers (B_0, B_1, B_2) feeding two downstream modules (M_3, M_4), and serves as the base for both uncapacitated and capacitated tests. The 6-node network (Figure 6.1, right) extends this setup by adding

a third module (M_5), enabling analysis of competitive allocation under shared upstream resources. The full 7-node network, introduced in Chapter 3 (Figure 3.1), reflects our leading example for ASML's general supply configuration and is used in the final experiments under moderate and tight capacity constraints. All networks are initially evaluated without capacity limits; later stages introduce explicit overcapacity settings relative to mean demand.

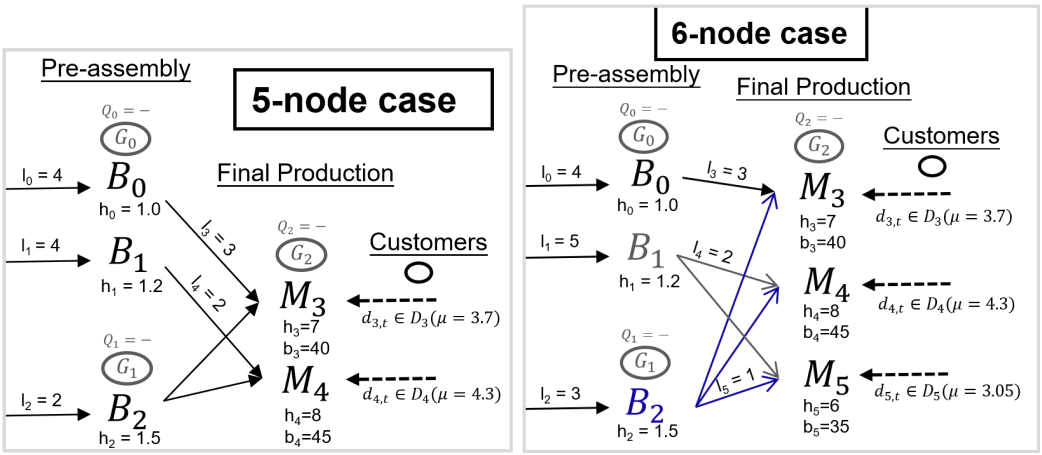


FIGURE 6.1: Smaller cases. Each buffer B or module M is processed at group G_k, with arcs indicating material flows and annotated with production lead times l_i, holding costs h_i, and backlog and demands $(b_e;d_{e,t})$ for each end item or module $(e \in \mathcal{C}_{end})$.

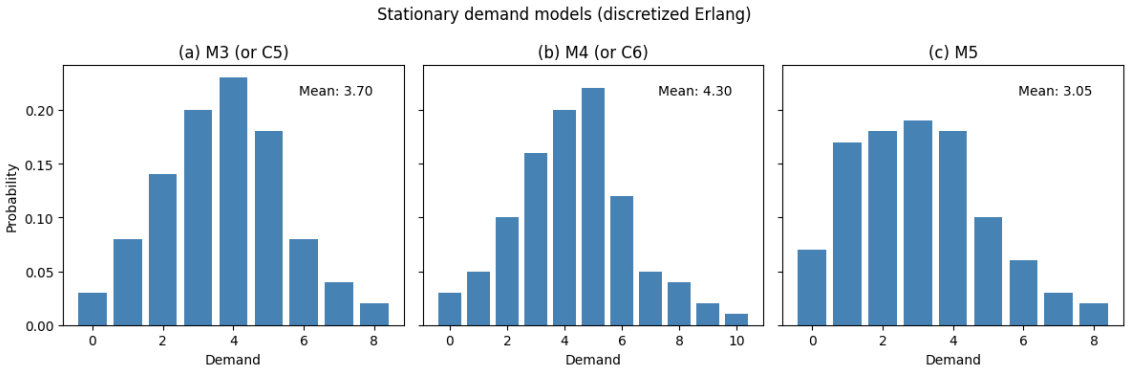


FIGURE 6.2: Demand Distributions

Demand Distributions. All demand processes are stationary and follow discrete distributions that approximate Erlang shapes. These are based on company insights and are consistent with the profiles used by Van Dijck et al. (2024). The 5-node and 7-node networks use the middle and left-hand side demand profiles in Figure 6.2, which represent different end-items. The 6-node system includes all three demand types to simulate multiple end-products.

Key Performance Indicators Averages over trajectories are reported for:

- **(Per-period) Total Cost:** Holding + backlog costs.

- **Holding / Backlog Cost:** Separate breakdowns.
- **RLIP (Requested Line-Item Performance):** Fraction of requested demand that is immediately satisfied. (Per end-product)
- **% Backlog:** Percentage of periods with unmet demand.
- **Average Inventory per Node (Inv):** Per-node average inventory levels.
- **DCL Training Metrics (tuning stage):**
  - **Train $\mathcal{L}$:** Training error at convergence, indicating model fit.
  - **Val. $\mathcal{L}$:** Generalization error on held-out scenarios.
  - **Epochs, s/epoch, train time:** Number of training epochs, runtime per epoch, total training time.

Computational Environment All experiments were executed on the Dutch national supercomputing infrastructure, supported by the SURF Cooperative (grant no. EINF-5192) to ensure reproducibility and consistent performance measurement. The supply chain simulations and DCL implementation were developed in `C++20` within the `Dynaplex Software Library` (Akkerman et al., 2023), neural network training and inference were performed using `PyTorch` (Temizöz et al., 2023). Simulations ran on a thin node of the Snellius supercomputer, operated by SURF, featuring two AMD Rome processors at 2.6 GHz, 128 cores in total, and 256 GiB of RAM.

6.1 Heuristic Benchmarking Experiment

In this first experiment, we test all 36 combinations of three base-stock sizing strategies—G-MATCH, S-MATCH (Rong et al., 2017), and FullS-MATCH, and six allocation rules (FCFS, RandomWF, WF, WFFS, WFR, WFRR) on the 7-node uncapacitated system in Figure 3.1. Method descriptions are provided in Chapter 4, and full results appear in Appendix A.2. As these methods are designed for uncapacitated inventory systems, production constraints are not considered.

We report per-period average cost and its standard deviation (see Table A.2 for these and the corresponding up- and down-rounded base-stock levels). Figure 6.3 summarizes the findings. The left panel averages over allocation rules for each base-stock strategy to identify the best base-stock sizing method. Using the best one (G-MATCH), the right panel then compares the performance of all allocation rules.

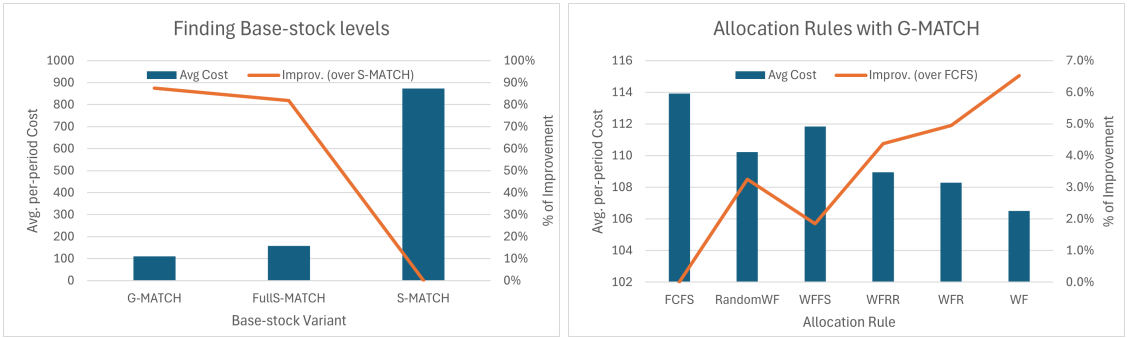


FIGURE 6.3: Main Results of 36 Heuristic Configurations on 7-Node System

From the left graph in Figure 6.3, we see that $S^{\text{G-MATCH}}$ clearly outperforms the alternatives, showing a nearly 90% cost reduction over $S^{\text{S-MATCH}}$ proposed by Rong et al. (2017). This confirms that modeling the network's actual structure and lead times (as $S^{\text{G-MATCH}}$ does) is required. $S^{\text{S-MATCH}}$, despite using general-network lead times, wrongly assumes a serial structure and performs poorly. The original method by Rong et al. (2017) remains valuable for distribution systems with its structured backorder matching heuristic. However, the proposed extension to general systems lacks formal evaluation and overlooks the dynamics of convergent parallel flows, causing its performance to drop. Finally, the fully serial-based ($S^{\text{FullS-MATCH}}$) variant, which applies the serial assumption for both structure and timing, performs much better than $S^{\text{S-MATCH}}$ but remains inferior to G-MATCH.

These results underscore that if a general system is approximated as serial systems, consistency between network structure and lead times is critical to achieve good performance. While directly modeling the true general system (as with $S^{\text{G-MATCH}}$) is always preferable, the Fully Serial variant was tested specifically to emphasize this point and show that even with fully serial matching, performance is much better than $S^{\text{S-MATCH}}$, which mixes general lead times with serial assumptions. Although the Fully Serial variant itself is unlikely to be practically useful (given that directly modeling the general system is computationally manageable), this can serve as a quick sanity check for general systems that are currently serially approximated.

Moreover, something less relevant but still important is that upward rounding of base-stock levels often provided a slight cost advantage (see in Table A.2). Moving onto the allocation rules, the left graph in Figure 6.3, across all base-stock models, Waterfilling rules (WF) and its derivatives (WFR, WFFS) consistently emerged as the most effective allocation strategies, outperforming FCFS for more than 5%. Even though we can see that the standard water-filling WF (push all inventory) outperforms the rule with reservations (WFR) with G-MATCH base-stock levels, WF and WFR produce similar results in other base-stock level variants (see the appendix table) such as FullS-MATCH, so more research is needed to determine whether WF always outperforms WFR.

Experiment Conclusion G-MATCH clearly outperforms the other base-stock methods, confirming the importance of matching both network structure and lead times in general systems. S-MATCH performs poorly due to its inconsistent assumptions, while FullS-MATCH performs moderately well but remains inferior. Among allocation rules, Water-Filling and its variants (WF, WFR, WFFS) outperform FCFS and randomWF, with WF performing best under G-MATCH.

6.2 DCL Tuning Experiments

Having identified the best-performing heuristic (G-MATCH base-stock levels with WF allocation) in the previous experiment, we now use it as the rollout policy for tuning the DCL algorithm. The goal is to fine-tune DCL parameters and architectural components to enhance learning performance and generalization, while keeping computational costs manageable.

We first fix several easy-to-tune hyperparameters based on preliminary testing and literature:

- **Warm-up length (L)**: Set to 60 (validated for 5–7-node networks in Appendix A.1) to ensure realistic state sampling.

- **Number of exogenous scenarios (M)**: $M = 1000$ achieves >80% action-label significance (Dynaplex metric) for all tested networks, which is sufficient to train the classifier reliably. Reducing M may lead to poor label quality and suboptimal learning, while increasing it offers limited performance gains at a higher computational cost.
- **Mini-batch size and Early Stopping**: 64 batch-size performs well across cases; increasing to 128 roughly doubled training time without consistent improvements, and in some cases even led to overfitting. For early stopping, we observe from trial and error in our cases that 10-15 epochs is the sweet spot, where decreasing or increasing it under/over fits.
- **Learning rate and optimizer**: Fixed to 0.001 with Adam, following Temizöz et al. (2023). We did not explore tuning these further.

The tuning process is organized into three building steps. Table 6.1 summarizes the steps, their configuration, and objectives. To ease learning and match the rollout policy, DCL is trained under a maximum order size of 11 (about 37.5% above the mean demand).

Step	Network Used	Samples (N) *(in thousands)*	Other Fixed Settings	Purpose
1. Feature Subset Selection	5-node (uncap.) Max order size of 11	15-25	Neural Network (Temizöz), H=35 (5x MCLT)	Find best input representation that generalizes well
2. Rollout Horizon Tuning	5-node (uncap.) Max order size of 11	25	Best feature subset, Neural Network (Temizöz), Horizon Mult. ∈ [1–5]	Tune rollout length relative to network lead times
3. NN Architecture Search	5- & 7-node (uncap.) Max order size of 11	25, 35	Best feature + horizon per network	Find NN that adapts to complexity and avoids
4. Number of Generations	5- & 7-node (uncap.) Max order size of 11	100, 200	Best # Generations, Feature Subset 6 (H)3x MCLT, NN Arch=[256, 128, 128, 64]	Find the number of generations

TABLE 6.1: DCL Tuning Experiments Overview

First, we identify the best subset of features for representing system states using the 5-node uncapacitated system. We fix the neural network to the standard architecture from Temizöz et al. (2023), with a training set of $N = 15{,}000$ samples (extended to 25,000 for top subsets), and use a rollout horizon $H = 35$ ($5\times$ the network's maximum cumulative lead time), which is sufficient for accounting of actions' long-term effects. Feature subsets are evaluated in this small setup because their relative performance tends to scale consistently with network size and training samples, given shared structural and decision dynamics.
Next, we tune the rollout horizon using a new parameter: the horizon multiplier, which expresses the horizon as a multiple of the network's maximum cumulative lead time (MCLT). This allows the planning horizon to scale naturally with network lead times; this multiplier can be assumed to generalize to different networks. We test values from $1\times$ to $5\times$ MCLT, again using the 5-node setup and best-performing feature subset.
Now, we find a neural network architecture, which is inherently network-specific due to varying input/output sizes. This step is done after fixing the best feature subset and rollout length. We compare several popular architectures (e.g., Pyramid, TwoLayer, DeepTapered) for their learning capacity and scalability. This staged approach avoids exhaustive tuning across all combinations of features, horizons, and architectures, as this would be computationally infeasible.

Finally, we employ the currently optimal hyperparameter settings and adjust the number of generations, using a calibrated number of samples determined through trial and error. Our aim is to determine if executing multiple generations provides any advantage.

6.2.1 Feature Subset Selection

We begin by identifying the most effective subset of features for representing system states. This is done using the 5-node uncapacitated system and previously mention parameters. The goal is to understand which input features enable the DCL model to learn high-quality policies efficiently. Since decision logic is structurally similar across systems, we assume the best-performing subset generalizes well to larger networks. We start from the basic configuration and incrementally add capacity, structural, and bottleneck features, which were described in Section 5.3.4. To evaluate, we use as KPIs: The train and validation losses (explained in the start of the chapter); and the relative per-period cost improvement based on the baseline features (Subset 1).

The evaluated subsets are the following:

- **Subset 1:** Basic State (Baseline: node, inventory, production)
- **Subset 2:** New State (Aggregate production & echelon inventory)
- **Subset 3:** Basic + Capacity Awareness
- **Subset 4:** Basic + Capacity + Structural Awareness
- **Subset 5:** Subset 4 + Local Shortages (MR, RS, Backlog)
- **Subset 6:** Subset 4 + Aggregated Shortages
- **Subset 7:** Subset 4 + Per-Node Shortages
- **Subset 8:** Subset 4 + Bottleneck Ratio
- **Subset 9:** Subset 6 + Ratio

Table 6.2: Performance Comparison of Feature Subsets (Lower Values Indicate Improvement)

Subset	Inputs	Train $\mathcal{L}$	Val. $\mathcal{L}$	Cost Imp. (%)
1 (Baseline)	15	0.666	0.852	0.00
2	21	0.712	0.837	+21.67
3	17	0.622	0.856	-2.45
4	20	0.606	0.790	-4.24
5	24	0.630	0.752	-4.35
6	23	0.572	0.746	**-5.01**
7	24	0.570	0.725	-5.01
8	21	0.645	0.786	-3.12
6 (25k)	23	0.541	0.631	-5.28
7 (25k)	24	0.548	0.636	-5.39
9 (25k)	24	0.539	0.634	-5.24

Table 6.2 reveals insights about feature subset evaluation:

- **Subset 2 Failure:** The New State's aggregated features increased costs by 21.67% despite additional inputs, which shows that high-level summaries cannot replace granular data
- **Capacity+Structural Awareness:** 2.45%, 4.24% cost reduction shows the importance of capacity utilization tracking and network context.

- **Bottleneck Features:** Including backlog information for end products showed no improvement, likely due to delayed backlog signals hindering timely decisions. The standalone bottleneck ratio (Subset 8) improved costs **-3.12%** over the baseline, possibly because it oversimplifies bottleneck pressure, missing supply constraint complexities. Notably, Subset 6 (Aggregated Shortages) matched Subset 7 (Per-Node Shortages) in performance with fewer inputs. This indicates that aggregated shortages better capture essential dynamics without node-specific details and reduce complexity and local variation noise.

- **Extended Evaluation of Top Performing Subsets:** Encouraged by Subsets 6 and 7, we tested 25,000 samples for scalability and stability on larger datasets. Subset 6 improved **-5.28%** over the baseline with smooth convergence and a validation loss of **0.631**, indicating better generalization. Although Subset 7 achieved slightly better improvement **-5.39%**, it required more inputs (24 vs. 23 for Subset 6) and showed signs of overfitting in later stages with fluctuating validation losses (0.041 vs 0.029). The per-node detail in Subset 7 does not justify the extra complexity and overfitting risk. Subset 6's concise design with stable convergence suggests a more efficient way to capture bottleneck effects.

- **Introducing Feature Subset 9** This subset extends Subset 6 by adding the Bottleneck Ratio indicator, slightly enhancing cost reduction to **-5.24%**. However, it introduced additional multicollinearity, leading to a less stable convergence path in training. Thus, while it proved competitive, its increased complexity did not justify the trade-off.

Experiment Conclusion In light of these results, **Feature Subset 6** – *Basic State + Capacity Awareness + Structural Awareness + Aggregated Shortages* – remains the best choice due to its balance of cost efficiency, input dimensionality, and stability across different sample sizes (15-25 thousand samples) as seen table 6.2.

6.2.2 Rollout Horizon Tuning

The rollout horizon length (H) in DCL represents the planning horizon over which the impact of current actions is evaluated. This look-ahead depth is crucial for ensuring that the agent accounts for both immediate and propagated effects of production decisions and must be at least one full production period (including all components lead times), we call this the maximum cumulative lead time or MCLT. We tune the rollout horizon length, expressed as a multiplier of the network's MCLT to ensure that the planning horizon scales with network complexity. Using the same 5-node system and the best feature subset, we identify the shortest horizon length that achieves stable learning without unnecessary computation. The tested configurations include the following rollout horizon lengths $H \in [7, 14, 21, 28, 25]$, which are relative to the MCLTs from 1x to 5x.

Experiment Conclusion The results indicate that performance improves substantially up to a rollout length of 2x MCLT, where it already shows signs of stability. From 3x MCLT onward, the performance remains consistent, with minimal added benefit when extending beyond 5x MCLT. Given this observation, the stability observed at 2x MCLT suggests that in larger networks, 2x or 3x MCLT may be sufficient for keeping proper decision-making and reducing computational demands.

Table 6.3: Average Total Cost across Different Rollout Lengths

Policy Configuration	Cost	Train time (s)
DCL-WF-FS6-H7	300.76	75.3
DCL-WF-FS6-H14	110.45	101.1
DCL-WF-FS6-H21	**110.05**	**155.4**
DCL-WF-FS6-H28	110.32	196.6
DCL-WF-FS6-H35	110.50	243.9

6.2.3 Neural Network Architecture Search

The choice of neural network architecture in DCL is important for capturing complex action-state dependencies, maintaining scalability, and avoiding overfitting as sample and network size increase. This step is performed per network (5- and 7-node cases), using the best feature subset and tuned rollout horizon (3xMCLT or 21 for the 5-node case and 27 for the 7-node case). To assess how different structures impact learning and generalization in our systems, we systematically tested several widely used architectures (Pyramid, TwoLayer, ThreeEqualLayers, WideMidLayer, and DeepTapered), ranging from shallow and narrow to deeper and more hierarchical topologies.

Table 6.4: Training diagnostics for the 5-node network ($N = 25\,000$ samples, $M = 1\,000$ scenarios, horizon = 21)

Architecture	Layers	Train $\mathcal{L}$	Val. $\mathcal{L}$	Epochs	s/epoch
Pyramid	[64, 128, 64]	0.59	0.63	36	1.86
TwoLayer	[256, 256]	0.50	0.67	41	2.93
ThreeEqualLayers	[128, 128, 128]	0.56	0.63	36	2.14
WideMidLayer	[128, 256, 128]	0.53	0.62	36	2.87
DeepTapered	[256, 128, 128, 64]	0.54	0.60	36	2.74

Table 6.5: Training diagnostics for the 7-node network ($N = 35\,000$ samples, $M = 1\,000$ scenarios, horizon = 27)

Architecture	Layers	Train $\mathcal{L}$	Val. $\mathcal{L}$	Epochs	s/epoch
Pyramid	[64, 128, 128]	0.636	0.715	46	2.03
TwoLayerThin	[128, 128]	0.662	0.733	36	2.11
TwoLayerWide	[256, 256]	0.599	0.740	36	2.27
ThreeEqualLayers	[128, 128, 128]	0.581	0.720	56	1.95
WideMidLayer	[128, 256, 128]	0.678	0.778	26	2.62
DeepTapered	[256, 128, 128, 64]	0.628	0.712	46	2.02
DeepTaperedNew	[256, 128, 128, 128]	0.729	0.774	21	2.98

The results of the architecture search are shown in Tables 6.4 and 6.5. In the 5-node system, all models converged, but DeepTapered ([256, 128, 128, 64]) achieved the best generalization with the lowest validation loss and balanced training time, while WideMidLayer ([128, 256, 128]) also performed well and the pyramid architecture ([64, 128, 64]) performed efficiently but neither of both outperform DeepTapered. TwoLayer ([256, 256]) obtained the lowest training loss but showed a higher validation gap, indicating overfitting. In the 7-node system, DeepTapered again offered a favorable trade-off between generalization and efficiency, followed by the Pyramid ([64, 128, 128]), which remained competitive but

training time per epoch does not justify its simplicity, while ThreeEqualLayers ([128, 128, 128]) also yielded good validation loss but required more epochs. Wider or deeper variants offered no added benefit and sometimes worsened overfitting or training speed.

Experiment Conclusion DeepTapered architectures show better scalability for DCL (as suggested by Temizöz et al. (2023) and support effective training and strong generalization as the network size increases. For future experiments and larger systems, DeepTapered ([256, 128, 128, 64]) is recommended for both 5- and 7-node settings, though ThreeEqualLayers is also suitable for larger problems where longer training time is acceptable. Moreover, the Pyramid architecture can be used for applications that require fast and reliable learning with minimal risk of overfitting.

6.2.4 Number of Generations

DCL refines an initial heuristic policy by training successive generations of neural-network classifiers, starting with the previous generation's policy. The number of generations is a key hyperparameter affecting model bootstrapping and the balance between performance gains and training costs.

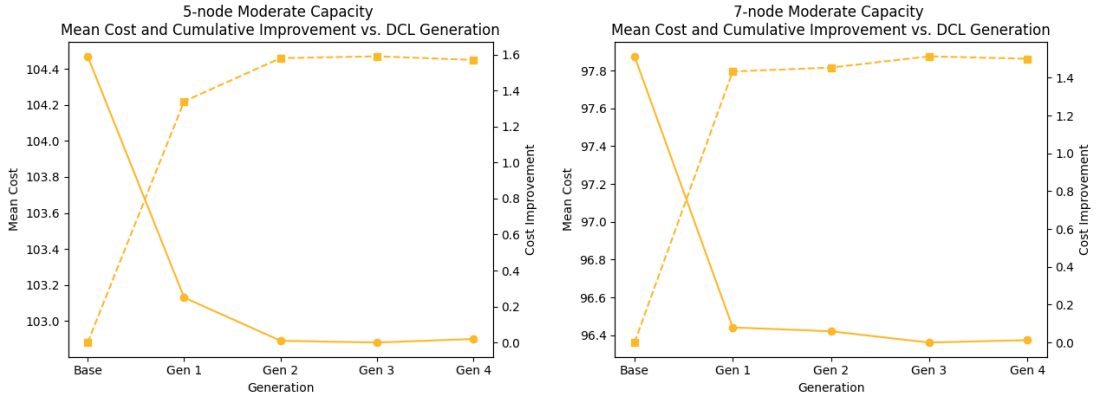


FIGURE 6.4: Mean Cost and Cumulative Improvement per Generation

Experiments on 5- and 7-node networks in Figure show that the first generation achieves the most significant cost reduction (1.2–1.6 units on average), while subsequent generations offer marginal improvements (0–0.2 units). By generations 3 and 4, improvements have stagnated. This performance comes with high computational costs, with training time increasing from 5 (Gen 0 to 1) to 40 minutes (next generations) for 5-node systems and from 17 minutes to 1.5 hours for 7-node systems. Similar trends are observed elsewhere. Therefore, we conclude that the majority of the benefits can be obtained in practice with just one generation of DCL beyond heuristics (for our experiments), and subsequent generations are not cost-effective when weighing training time against cost savings.

DCL Best Practices for our Systems

In this section, we started by selecting reasonable values for easy-to-tune parameters, and then moved into defining best engineered subset of features, in which Subset 6 (Basic State + Capacity & Structural Awareness + Aggregated Shortages) achieved over a 5% cost reduction compared to the baseline state representation with least number of features and best generalization. Additionally, we found via experiments that setting a rollout horizon

to 2-3 times the network's maximum cumulative lead time results in an ideal look-ahead length for training. Furthermore, implementing a four-layer "DeepTapered" architecture enhances DCL's scalability more effectively than other designs. Lastly, employing a single policy generation with the proper identified state sampling yields the most substantial improvements, with future generations offering no notable advancements. Find the final tuning practices which we will use in the next experiments in Table 6.6

Our tuning experiments have shown that a DCL setup that has been well-adjusted offers promising performance. In our subsequent experiments, we intend to determine whether this method outperforms our best heuristic methods.

Table 6.6: Tuned DCL Parameters

Hyperparameter	**Tuned Value**
Rollout Horizon (H)	2–3× MCLT
Sampled States (N)	100–150 (small), 200 000 (large)
Scenarios per Action (M)	1 000 (ensures $>$ 80% label significance)
Warm-Up Steps (L)	60 (5/6/7-node systems)
Policy Iterations (G)	1
Architecture	[256,128,128,128] (default) Alternatives: [128,256,128], [64,128,64]
Batch Size	64
Early Stop Patience	10–15 epochs (optimal to avoid under/over-fitting)

6.3 Comparative Evaluation Experiments (Heuristics vs DCL)

This section presents three comparative experiments designed to evaluate the performance of heuristics versus the DCL policy under increasingly realistic conditions. Table 6.7 summarizes the key experimental configurations.

First, we start by studying how lead-time asymmetries influence inventory allocation in the 5- and 6-node networks when no capacity constraints exist. All policies use the general-focused base-stock levels (G-MATCH) identified in Experiment 1. We conduct lead-time stress tests on methods and compare the two best-performing heuristic allocation rules, Water-Filling (WF) and its reservation variant (WFR), against the DCL policy trained on top of G-MATCH base stocks and WF allocation logic (DCL-WF). The goal is to understand how DCL dynamically negotiates trade-offs differently from rule-based methods. DCL is trained using $N = 100{,}000$ samples and $M = 1{,}000$ demand scenarios, with a planning horizon set to $H = 2 \times$ MCLT to provide adequate look-ahead.
Next, we impose shared-capacity limits on the same networks to evaluate policy adaptation under constrained production. We test three lead-time scenarios and five capacity configurations (C1–C5), using WF as the fixed allocation rule. We compare: (i) the original G-MATCH base-stocks, (ii) Guided Base-Stock (GBS) levels re-optimized for each capacity case, and (iii) DCL trained on top of GBS+WF. DCL is trained with $N = 100{,}000$ samples and $M = 1{,}000$ scenarios, except for lead-time case S1 in the 5-node network ($N = 125{,}000$), and the 6-node case ($N = 150{,}000$), to handle larger state-spaces from extended lead times and number of nodes. The horizon was set to $H = 3 \times$MCLT (one more than uncapacitated) because DCL needs a longer look-ahead for capacity constraints. The MLP architecture remained $\{256, 128, 128, 64\}$ with a minibatch size of 64 and early stopping patience of 15. Finally, we scale to the full seven-node ASML network under both moderate and tight

capacity configurations, where we evaluate the same three policies and DCL training setup as the previous experiment, but increasing the training sample size to $N = 200{,}000$ to support the complexity of the full network. Neural network training convergence and run times can be found in Appendixes A.4, A.5, A.6.

TABLE 6.7: Overview of Comparative Evaluation Experiments

Exp.	Setting	Network(s)	Capacity	DCL Training
(1)	Lead-time asymmetries	5-, 6-node	None	$N = 100{,}000$, $H = 2 \times$ MCLT
(2)	Capacity-constrained cases	5-, 6-node	C1–C5	[l]$N = 100{,}000$–$150{,}000^*$, $H = 3 \times$ MCLT
(3)	Full-network under capacity	7-node	C3,C4	$N = 200{,}000$, $H = 3 \times$ MCLT

**S1 in 5-node uses* $N = 125{,}000$*; 6-node case uses* $N = 150{,}000$ *to handle extended state-spaces. All DCL models use MLP* $\{256, 128, 128, 64\}$*, minibatch 64, and early stopping (patience 15).*

6.3.1 Small Uncapacitated Cases under Lead-Time Stress Tests

This experiment investigates how lead-time asymmetries influence inventory allocation in smaller, uncapacitated networks. We benchmark the two strongest heuristic rules (WF and WFR) against the DCL policy trained on top of WF logic and G-MATCH base-stocks. By focusing on fewer nodes, it allows for clearer observation of how and where DCL's learned behavior departs from heuristics. Full results, including comparisons with all other allocation rules, are provided in Appendix A.3. Results focus on RLIP at each retailer, average inventory at selected nodes, and % backlog. We evaluate five distinct lead-time scenarios:

Lead-time scenarios. Each vector lists different lead times$(l_0, l_1, l_2, l_3, l_4, (l_5))$.

- **S1** *Baseline* $(4, 4, 2, 3, 2)$:
  Shared node B_2 replenishes fastest; downstream modules differ slightly.
- **S2** *Slow expensive* $(4, 4, 2, 2, 3)$:
  Same as baseline but with a slower expensive module $M4$.
- **S3** *Inflated Pipeline* $(4, 4, 2, 6, 2)$:
  Slow M_3 response, exaggerating its apparent shortage. Tests over-prioritization of apparently "needy" nodes versus M_4 (LT=2).
- **S4** *Urgent Integration* $(5, 5, 2, 4, 1)$:
  M_4 becomes time-critical (one period) while upstream is sluggish. Tests priority management for immediate-disruption nodes.
- **S5** B_2 *Bottleneck* $(2, 2, 6, 3, 1)$: Shared node becomes the slowest link, forcing re-allocation under a bottleneck. Tests dynamic material rerouting strategies.
- **S6** *Shared-Node Competition* $(4, 5, 3, 3, 2, 1)$:
  Adds retailer M_5 $(l_5 = 1)$ to test two- and three-module competition via B_1 and B_2. Tests allocation balancing under competition.

Analysis: WF, WFR, and DCL Dynamics Table 6.8 and Figure 6.5 shows the results, where **WFR never outperforms WF in any scenario**; its mean cost is consistently 0.1–0.3% *higher* (blue bars slightly higher, orange bars unchanged). This indicates that

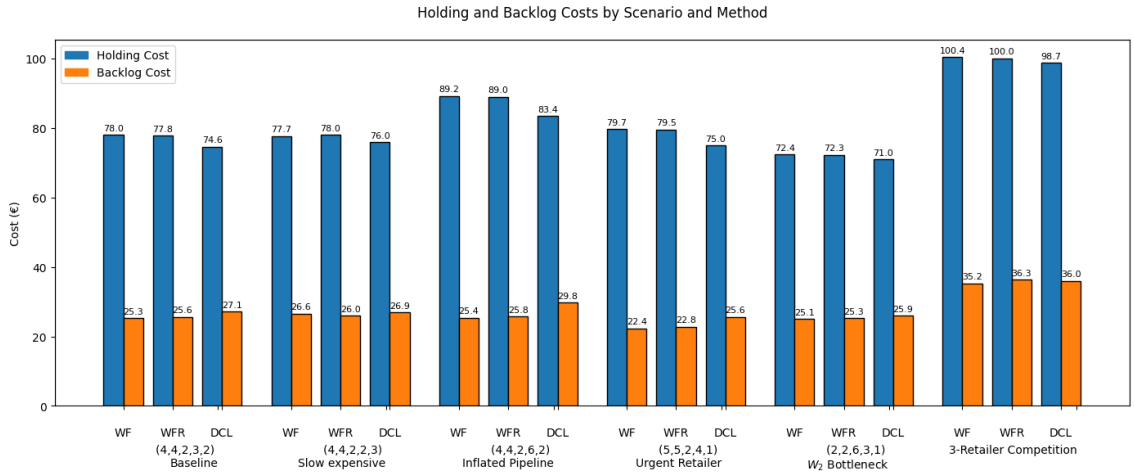


FIGURE 6.5: Cost breakdown by Scenario and Method for the Smaller Uncapacitated Cases

a *static* "hold-reserves" rule is rarely the right amount of inventory to withhold: in many periods the reservation is unnecessary, while in genuinely tight states it is insufficient. Nevertheless, we believe that the idea of reserving stock is not fully wrong; what is missing is *state-dependence* which DCL is able to learn. DCL lowers mean cost by 0.6–1.6% compared to WF (see MeanCost column) by selectively and strategically managing inventory.

The following four consistent DCL behaviors are visible across the 5 scenarios:

1. **Aggressive cutting inventories of B_2:** Instead of over-stocking the fast shared buffer, DCL holds just enough there to smooth immediate flows, cutting B_2 inventory by 35–45 % compared to WF.
2. **Smart "reservations" or inventory control upstream:** When a downstream module risks stock-out, DCL reallocates saved units to its dedicated buffer (B_0 or B_1), creating a targeted cushion exactly where it's needed.
3. **Lead-time–aware rerouting:** In the scenarios where long lead times exaggerate the perceived urgency of some modules, DCL diverts supply from slow paths toward faster or higher-priority nodes, correcting misallocations and reducing backlog spikes.
4. **Penalty-sensitive bias:** If one module carries a higher backlog cost, DCL shifts inventory through its paths to protect it first, even at the expense of backlog in less critical nodes.

These tactics allow DCL to hold or "reserve" inventory upstream more strategically, creating flexible buffers that can be released only under real-time pressure, something neither WF nor WFR can achieve. The result is a balanced, adaptive policy that consistently trims holding costs without raising backlog penalties. Detailed scenario statistics are in Appendix A.3.

The following summarize how DCL adapts its tactics to each scenario:

S1 DCL reduces costs by 1.6% by strategically cutting B_2 inventory from 4.7 to 2.7 units and accepting targeted backlogs.

S2 DCL prioritizes the expensive, long-lead-time module by increasing upstream inventory while still cutting B_2, which avoids overstock and reducing cost by 1.5%.

S3 DCL corrects the inflated urgency of M_3 by shifting inventory toward the faster M_4, lowering backlog misallocation and improving cost by 1.2%.

S4 DCL anticipates the stockout risk at the most time-sensitive node, M_4, and reroutes

TABLE 6.8: Comparative performance under lead-time stresses (per-period averages over 1,000 trajectories). RLIP = Service level at M_3,M_4,(M_5).

Scenario	Policy	MeanCost	% Backlog	$\mathbf{Inv_{B0}/Inv_{B2}}$ / RLIP
(4,4,2,3,2) – Baseline				
	WF	103.30	23.91%	3.62/4.72 / (91.9|93.1)%
	WFR	103.50	24.21%	3.65/4.76 / (91.8|93.1)%
	DCL	**101.73**	25.45%	**4.71/2.66** / (91.6|92.4)%
(4,4,2,2,3) – Slow expensive				
	WF	103.98	23.93%	3.65/4.02 / (93.4|91.3)%
	WFR	104.28	23.49%	3.62/3.95 / (93.5|91.5)%
	DCL	**102.90**	24.38%	**4.69/2.55** / (93.5|91.1)%
(4,4,2,6,2) – Inflated Pipeline				
	WF	114.57	22.15%	3.71/3.83 / (92.0|93.0)%
	WFR	114.79	22.47%	3.76/3.90 / (91.8|93.0)%
	DCL	**113.16**	25.22%	**3.82/2.02** / (90.2|92.2)%
(5,5,2,4,1) – Urgent Module				
	WF	102.08	21.42%	3.91 / (93.1|93.7)%
	WFR	102.29	21.77%	3.98 / (92.9|93.7)%
	DCL	**100.65**	24.06%	**2.23** / (91.9|93.0)%
(2,2,6,3,1) – W_2 Bottleneck				$\mathbf{Inv_{W0}/Inv_{W1}/Inv_{W2}}$ / RLIP
	WF	97.43	24.83%	3.11/3.75/6.85 / (91.1|93.9)%
	WFR	97.59	25.02%	3.14/3.75/6.88 / (91.0|93.9)%
	DCL	**96.89**	25.44%	2.52/3.26/6.91 / (91.3|93.3)%
3-Module Competition				$\mathbf{Inv_{W1}/Inv_{W2}}$ / RLIP
	WF	135.66	34.65%	6.15/5.26 / (91.4|91.2|94.9)%
	WFR	136.30	35.47%	6.19/5.38 / (90.9|91.1|94.9)%
	DCL	**133.63**	35.89%	7.07/3.87 / **(91.0|92.9|92.8)%**

inventory from the shared buffer B_2 to reach M_4 earlier, even if it means slightly more backlog at less urgent modules, which reduces total cost by 1.4%.

S5 With all paths slow, DCL increases upstream inventories (at B_0 and B_1) by small amounts to smooth delivery and somewhat reduce variability, which leads to a modest 0.6% gain.

S6 DCL focuses on the fastest and highest-penalty module M_4, allocating more aggressively toward it during tight periods (see increase of RLIP) while holding less in the shared buffer, which reduces both unnecessary holding and backlog costs, and achieves the largest improvement (1.6%)

Experiment Conclusion WF is too eager, WFR too rigid, and the optimum lies between them, accessible only through state awareness. DCL approaches that optimum by tailoring its reservation size to real-time pipeline risk, releasing or withholding inventory exactly when the cost trade-off justifies it. This adaptive logic explains its consistent 0.6–1.6 % advantage across all asymmetric lead-time scenarios. Full policy statistics appear in Appendix A.3.

6.3.2 Small Capacitated Cases under Lead-Time Variantions

This experiment evaluates how the Guided Base-Stock Optimization (GBS) improves upon the General-Focused basestock (designed for uncapacitated systems) when capacity limits

are imposed. We also measure how DCL, which in previous experiment demonstrated state-dependent reservation benefits, further reduces cost under capacity constraints. We reuse the 5-node and 6-node networks from prior sections, applying five capacity scenarios (C1–C5) defined by percentage overcapacity relative to mean demand (see Tables 6.9–6.10). For 5-node configurations, three lead-time cases (S1: $(4, 4, 2, 3, 2)$; S2: $(4, 4, 2, 6, 2)$; S4: $(4, 4, 6, 3, 2)$) are tested. For the 6-node network, we evaluate a single lead-time case (S5: $(4, 5, 3, 1, 3, 2)$). Analysis concentrates on total cost and node-level inventories as capacity slack shrinks.

Capacity Grouping and Scenarios. In the 5-node system, capacity groups are $\{B_0, B_1\}$, $\{B_2\}$, and $\{M_3, M_4\}$. Since M_3 and M_4 have mean demands of 3.7 and 4.3 (sum = 8), a capacity of 9 corresponds to 12.5% overcapacity. Table 6.9 defines five cases:

Table 6.9: 5-node capacity scenarios (C1–C5) relative to mean demand (=8).

Case	**G1** $(B_0 + B_1)$	**G2** (B_2)	**G3** $(B_3 + B_4)$
C1 – Tight G3	10	10	**9**
C2 – Tight G2	10	**9**	10
C3 – +12.5%	9	9	9
C4 – +25%	10	10	10
C5 – +37.5%	11	11	11

In the 6-node network, capacity groups are $\{W_0, W_1\}, \{W_2\}, \{M_3 + M_4 + M_5\}$. With mean demand $3.05 + 3.7 + 4.3 \approx 11$, a capacity of 13 corresponds to 18.2% overcapacity. Table 6.10 defines five cases:

Table 6.10: 6-node capacity scenarios (C1–C5) relative to mean demand (=11).

Case	**G1** $(B_0 + B_1)$	**G2** (B_2)	**G3** $(M_3 + M_4 + M_5)$
C1 – Tight G3	14	14	**13**
C2 – Tight G2	14	**13**	14
C3 – +18.2%	13	13	13
C4 – +27.3%	14	14	14
C5 – +36.4%	15	15	15

Results and Visualization. Figure 6.6 plots the percentage reduction in mean cost of GBS and DCL versus the General-Focused baseline over capacity scenarios C1–C5 for the three 5-node lead-time cases (S1–S3) and the 6-node case (S5). In all 5-node setups, GBS delivers its largest savings (≈8–10%) when downstream modules or shared buffers are tight (C1, C3) and negligible benefit when more capacity slack appears (C4–C5). DCL builds on GBS: in C1 it adds roughly 4–5 percentage points more savings, in C3 around 2–3 points, and in C2 only about 1 point. Even under mild slack (C4–C5), DCL finds ≈1% extra improvement. the same pattern holds but with slightly smaller magnitudes for the 6-node case (S5).

To further explain these cost patterns, Graphs in Figure 6.7 display average on-hand inventories at buffers B_0, B_1, B_2 and modules $M_3, M_4, (M_5)$ under three policies (WF baseline, WF+GBS, and DCL) for the tight scenarios C1–C3 in each lead-time variant S1,S2,S4,S5. In each figure, three subplots correspond to C1 (downstream modules tight), C2 (shared buffer B_2 tight), and C3 (uniform tightness). Viewing these figures together reveals consistent patterns that are similar to the uncapacitated experiment: DCL consistently reduces inventory at the shared bottleneck buffer (B_2) and reallocates it into the faster

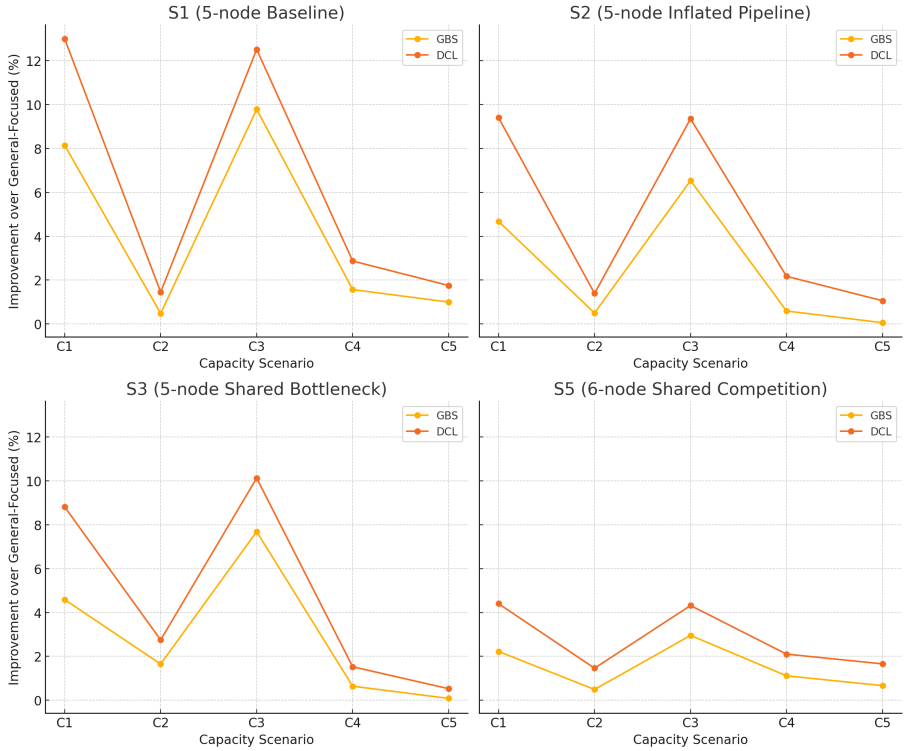


FIGURE 6.6: Improvement (%) of GBS and DCL over the General-Focused basestock across capacity scenarios C1–C5 for each network and lead-time case: S1, S2, S3, and 6-node S5.

upstream group (B_0/B_1), and module stocks are biased toward the higher-penalty nodes. Additionally, DCL's key advantage over GBS lies in **synchronizing buffer inventories** between the shared buffer B_2 and its paired upstream buffers (B_0 for M_3, B_1 for M_4). Whereas GBS sets static basestocks independently, sometimes leaving B_2 stocked when the matching upstream buffer is low (or vice versa). DCL, on the hand, maintains average levels so that when B_2 replenishes, its partner buffer is also likely to have stock, maximizing "both-buffers-available" events. In practice this means DCL holds B_2 more leanly and balances B_0 and B_1 according to module penalties and expected B_2 availability, avoiding idle inventory in one buffer and wasted capacity in the other.

Below we discuss how the above logic appears in each lead-time variant, referring to average-inventory levels for C1–C3.

S1 In the baseline case, GBS raises basestocks for both modules but may leave B_2 average higher than ideal relative to B_0/B_1, causing occasional mismatches. DCL trims B_2 and raises the upstream buffer for the higher-penalty module (e.g., B_0 or B_1 as appropriate), so that when B_2 is available, its partner buffer is ready. Under uniform tightness, DCL then splits inventory evenly across B_0 and B_1 rather than concentrating in one, ensuring both module inputs align with B_2 timing. These adjustments appear in first graph in Figure 6.7 as lower B_2 and balanced upstream levels, which yields the extra cost reduction over GBS.

S2 With M_3's lead time inflated, GBS static CCR can overstock B_2 for M_3 even when B_0/B_2 alignment is poor and M_4 suffers. DCL responds by moderating B_2's average and stocking B_1 (for M_4) or B_0 only to the extent that realistically matches B_2's capacity pattern in order to align inventories to correct lead-time distortions. Under uniform tightness, DCL balances B_0 and B_1 in proportion to module penalties and expected B_2 timing, so

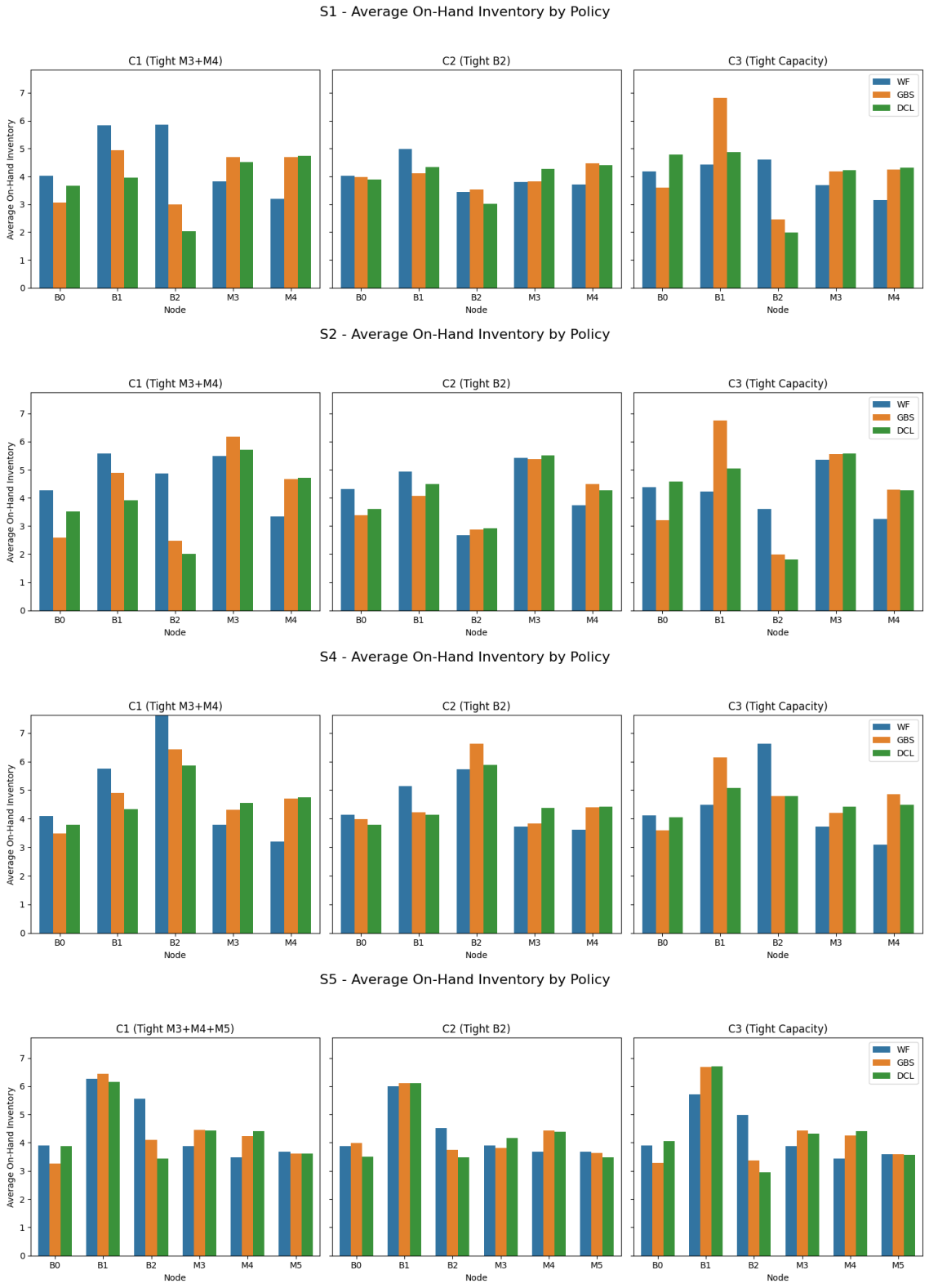


FIGURE 6.7: Average On-Hand Inventory by Policy for all scenarios. Subfigures (left to right) represent C1 (Tight M3+M4+M5), C2 (Tight B2), and C3 (Tight Capacity).

both upstream buffers refill in step with B_2. The second graph of Figure 6.7 shows these synchronized levels and explains DCL's improved performance over GBS.

S4 Here the shared buffer B_2 has a long lead time. GBS often stocks B_2 heavily but may not align upstream buffer levels, causing low joint availability. DCL holds B_2 at a moderate average and adjusts B_0/B_1 so that when B_2 replenishes, the appropriate upstream buffer is ready—prioritizing the higher-penalty module's pair. Under uniform tightness, DCL splits upstream inventory according to module costs and B_2's realistic availability pattern, rather than loading one buffer. The last plot in Figure 6.7 reflects these aligned average inventories and underpins the observed extra savings versus GBS in the shared-bottleneck scenario.

S5 In the 6-node case, DCL surpasses GBS by better coordinating shared buffer availability among three end-items. GBS inflates B_2 and evenly distributes stock, causing mismatches under tight capacity due to unaligned inventories. In contrast, DCL optimizes $B_2$and allocates B_0 and B_1 based on specific needs and backlog penalties. In C1, GBS reduces B_0 to boost B_1 for M_4/M_5, leaving M_3 understocked when B_2 arrives. DCL balances B_0 with B_2's cadence, increasing M_3 availiability, while maintaining adequate B_1 for high-penalty modules, leading to increased B_0 and M_3 stock levels alongside M_4. In C2, where B_2 bottlenecks, GBS inflates module stocks, stalling production if buffers misalign. DCL sets B_2 to realistic capacity and maintains B_0/B_1 at necessary levels, ensuring partner buffers align when B_2 opens, resulting in moderate averages and less idle inventory. In C3, with uniform constraints, GBS evenly distributes stock while ignoring dependencies. DCL adjusts based on module penalties and lead-times, raising B_1 for M_4/M_5 and B_0 for M_3 while keeping B_2 lean. This reduces module shortages and provides cost advantages over GBS.

Experiment Conclusion

Figure 6.7 consolidates average on-hand inventories for buffers B_0, B_1, B_2 and modules M_3, M_4 (and M_5 in S5) under WF, WF+GBS, and DCL across the tight scenarios C1–C3 in each lead-time variant (S1, S2, shared-bottleneck, three-module competition). A consistent pattern emerges: DCL maintains a somewhat leaner average at the shared buffer B_2 and reallocates that inventory into the upstream buffers B_0 and B_1 in a way that aligns with each module's penalty and the actual timing of B_2 availability. Crucially, DCL's gain arises from **synchronizing** paired buffers—ensuring that whenever B_2 is replenished, its matching upstream buffer (B_0 for M_3, B_1 for M_4 or M_5) also has stock—rather than from large shifts in total inventory.

- Under downstream-tightness (C1), DCL slightly trims B_2 relative to GBS and boosts the upstream buffer feeding the highest-penalty module so that "both-buffers-available" events increase.
- When the shared buffer is tight (C2), static GBS often inflates all module targets but still faces mismatches; DCL holds B_2 at realistic levels and keeps upstream reserves poised to pair with any sporadic B_2 replenishment.
- In uniform tightness (C3), instead of concentrating stock in one upstream buffer based on static signals, DCL balances B_0 and B_1 according to module costs and expected B_2 timing, avoiding idle stock and enabling more resilient fulfillment.
- These same principles correct distorted signals in the inflated-pipeline variant (S2) and avoid overstocking the slow shared buffer in the shared-bottleneck case: DCL's

state-aware adjustments consistently align paired buffers under each lead-time pattern. In S5's three-module competition, the synchronization logic extends to coordinating three end-items: DCL times B_0 and B_1 stocks to match B_2 availability in proportion to each module's urgency, again with modest changes to B_2 but smarter upstream placement.

Summary and KPI-driven Take-Aways

- **Role of capacity:** In C1–C3, capacity binds, forcing GBS to raise basestocks at bottleneck nodes. Inventory KPI shifts under GBS indicate which nodes are identified as bottlenecks. In contrast, under C4–C5, GBS and DCL converge to similar inventory patterns since no node experiences sustained blocking.
- **DCL's advantage:** KPI patterns confirm that DCL lowers on-hand inventory at the most constrained buffer, raises average availability at the paired upstream buffer feeding the critical module, and improves module inventory availability.
- **Managerial insight:** These results highlight that while a well-tuned static heuristic handles gross capacity constraints, incremental dynamic coordination of shared and upstream buffers, timing paired availability to module-specific needs, yields consistent, explainable benefits, especially when lead-time asymmetries or competition distort naive stocking signals.

6.3.3 The Large Seven-node Capacitated Network

The objective of this experimnet is to determine whether DCL's benefit persists in the large ASML-inspired network.

Experiment-specific settings.

- **Capacity regimes:** moderate Q_g=10 (≈25 % over mean) and tight Q_g=9 (≈12.5 %).
- **Policies:** WF (uncap.), WF-GBS, DCL.
- **DCL training:** N=200,000 samples; horizon $H = 3\,\text{MCLT}$.

Below we discuss cost-composition, inventory allocation, service metrics, and the reasons DCL's extra gain shrinks under tight capacity.

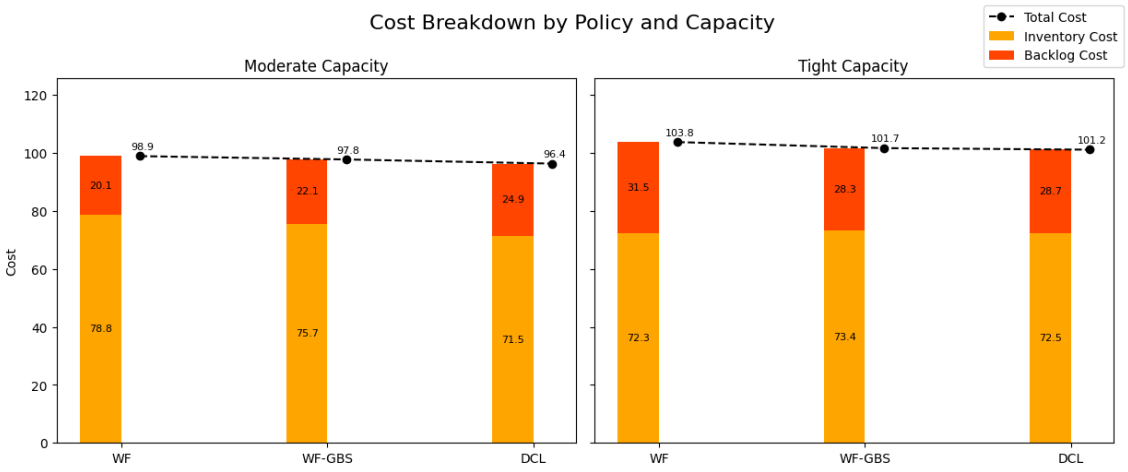


FIGURE 6.8: Inventory Cost (gold), Backlog Cost (orange) and Total Cost (dashed) per period.

Cost Composition

The cost breakdown (Figure 6.8) shows that WF+GBS captures the bulk of achievable savings by statically adjusting base-stock targets to average capacity constraints. Under moderate capacity, WF+GBS lowers inventory cost relative to WF, but incurs a modest rise in backlog cost, nonetheless it yields a net total-cost reduction of 1%. DCL further reduces inventory cost more sharply but accepts a somewhat larger backlog increase, resulting in an additional total-cost drop of 1.5%. Under tight capacity, WF+GBS's static tuning delivers a 2% large savings gain by raising some buffers in exchange for lower backlog costs. Here DCL's further improvement is small (0.5%): it trims inventory slightly more where possible and rebalances to get marginal savings. Further exploration is required to understand its limited performance on this tight scenario.

Node-Level Inventory Allocations

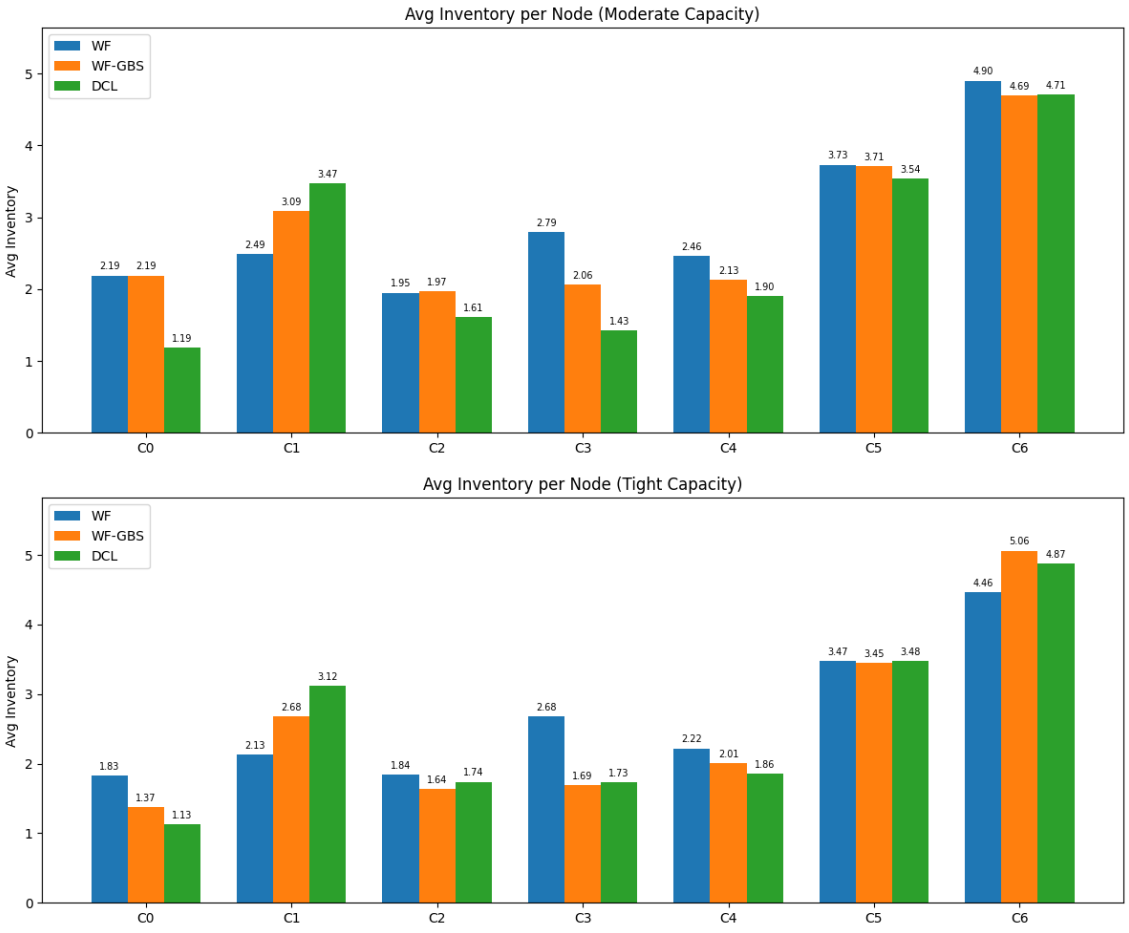
The average-on-hand inventory plots (Figure 6.9) reveal consistent allocation patterns under DCL versus WF and WF+GBS:

Upstream supplier differences: Here we see that C0 is slower or more constrained, DCL holds noticeably less than WF+GBS in the slow supplier (C0) (e.g., moderate: WF+GBS ≈2.19 at C0 vs. DCL ≈1.19), which explains having too much inventory in longer-to-replenish paths (C0) is not effective if other parallel partners (C1) are not simultaneously stocked. Moreover, C1 is relatively faster and critical for certain mid-tier paths, which explains why DCL holds more than WF+GBS (e.g., moderate: GBS ≈3.09 vs. DCL ≈3.47). Since C1 supplies C4 (and via C4 eventually feeds end-items C5/C6), holding more at C1 ensures that when downstream buffers (say C2/C3) become available, C1 can quickly replenish its C4.

Mid-tier buffers (C2, C3, C4): Since C2 and C3 depend on C0, DCL carries their inventories more leanly than WF+GBS. Although WF+GBS may set a higher static target at C2/C3 to guard against average shortages, if C0 inventory is uneven or slow, that static cushion may not translate into actual simultaneous availability at both inputs. DCL's lower average at C2/C3 matches its leaner C0 holding, avoiding idle stock that cannot be used because its partner input is missing. Moreover, C4 depends on C1, and DCL's higher C1 stock supports a moderate average at C4. Rather than inflating C4 statically, DCL trusts the faster upstream path (C1) to replenish C4 when needed, so it carries only enough average inventory at C4 to meet likely immediate demand when both C1 and its second input (C2/C3 as applicable) align.

End-items (C5, C6): This nodes are composed from mid-tier components and require specific mid-tier inputs: e.g., C5 might need (C2 and C4), C6 needs (C3 and C4). Because C4 draws from faster supplier C1 (where DCL holds more), and C2/C3 draw from leaner C0, DCL ensures that when C4 replenishes, its partner mid-tier (C2 or C3) has a matching—but leaner—average level, so joint availability events occur often enough. Moreover, the average inventory for the higher-backlog-cost end-item (C6) is slightly higher under DCL than WF+GBS, even if overall end-levels are slightly lower or similar. By aligning upstream/mid-tier availability toward the critical path feeding C6, DCL sustains or marginally improves its RLIP. The average-inventory bars show that C6's level under DCL remains competitive, reflecting this prioritization.

FIGURE 6.9: Average on-hand inventory at each node (C_0–C_6). Up = moderate capacity,down = tight capacity.



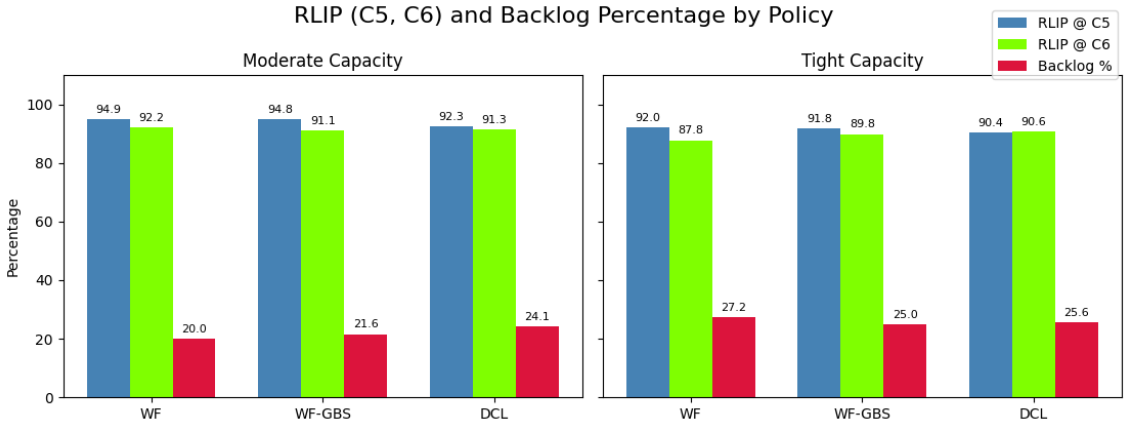


FIGURE 6.10: RLIP at C_6 (blue) and backlog frequency (red) under each policy. Solid = RLIP_{C_6}, dashed = % periods with any backlog. Left = moderate capacity; right = tight.

In sum, DCL's learned policy respects the production setup: it avoids overstocking a mid-tier buffer if its upstream partner is lean, instead reallocating to faster or higher-priority paths. WF+GBS, by contrast, raises targets for many buffers independently based on average constraint ratios, which can leave one input stocked without its partner; this static approach cannot guarantee the simultaneous availability that actual production requires. The average-inventory figure 6.9 visually corroborates these alignment patterns.

Service Level & Backlog Trade-offs

The RLIP/backlog frequency plots in Figure 6.10 illustrate how these allocation patterns translate into service outcomes:

- **Under moderate capacity (C4)**: WF+GBS increases fill at the critical end-item (C6) compared to WF, but backlog frequency rises slightly overall. DCL further concentrates service protection: by aligning buffer averages toward the highest-penalty path, DCL preserves or slightly boosts fill for C6 while allowing marginally more backlog at the less-critical end-item (C5). The plot shows DCL's backlog bar higher than WF+GBS, but RLIP@C6 is maintained or improved. This trade-off reduces total cost, where a slight sacrifice at C5 frees inventory to guard C6 more effectively.
- **Under Tight capacity (C3)**: WF+GBS already lifts C6 fill by raising static targets. DCL's alignment yields only minor inventory shifts, with its service/backlog bars having more balanced RLIPs for end items and with a slight additional protection for C6. Because capacity is scarce, any reallocation must be cautious; the plots confirm that DCL's policy avoids harming C6 fill, and in turn reflects the small difference in total cost.

Why DCL's Gain Shrinks under Tight Capacity

In smaller networks (5- and 6-node), DCL often sees its largest gains under tight capacity because the reduced action space simplifies learning on a narrower set of impactful reallocations. In the full 7-node system under tight capacity, however, DCL's incremental improvement falls below that seen under moderate constraints. An easy explanation for this would be that as network depth and action dimensionality grow, our current neural policy faces greater scalability challenges in representing and learning effective decisions.

However, we hypothesize that GBS already does a very good job and sets each node's base-stock near its shared-capacity limits. Now, DCL tries to do better by moving inventory around (a strategy that worked well in smaller 2-echelon systems), for example, by reducing inventory in the middle nodes (like C2, C3, C4) and sending it to more urgent downstream nodes (like the final products C5 and C6). This fails because middle nodes can't be easily refilled as they depend on already constrained upstream components (C0 and C1). Additionally, middle nodes serve multiple end-items, reducing inventory impacts final products, incurring high backlog penalties. Thus, any reallocation by DCL may do more harm than good, explaining its minimal improvement.

Why DCL's Gain Shrinks in the 7-Node Tight Case

In smaller networks (5- and 6-node), DCL often sees its largest gains under tight capacity because the reduced action space focuses learning on a narrower set of impactful reallocations. In the full 7-node system under tight capacity, however, DCL's incremental improvement falls below that seen under moderate constraints. While increased network depth and action dimensionality may strain the neural policy's scalability, our primary hypothesis is

structural: GBS already sets each node's base-stock near its shared-capacity limits, leaving little uncommitted inventory. Any attempt by DCL to trim mid-tier buffers (C2–C4) and shift stock toward urgent end-items (C5, C6) fails because (a) upstream components (C0, C1) are themselves capacity-bound and cannot quickly refill a reduced mid-tier buffer, and (b) each mid-tier node feeds multiple end-items, so reducing its inventory simultaneously harms several high-penalty products. Thus, under these simultaneous bottlenecks and tight upstream availability, virtually no reallocation remains beneficial, explaining why DCL's extra gain drops below 1% in the tight 7-node scenario.

We hypothesize that in the 7-node system under tight capacity, GBS optimizes node inventory targets to reach close to shared-group capacity. DCL attempts to improve by reallocating inventory, a strategy effective in smaller 2-echelon systems, by shifting stock from middle nodes (C2-C4) to urgent downstream nodes (C5 and C6). This fails because middle nodes can't be easily refilled as they depend on already constrained upstream components (C0 and C1). Additionally, middle nodes serve multiple end-items; reducing inventory impacts several final products, incurring high backlog penalties. Thus, any reallocation by DCL may do more harm than good, explaining its minimal improvement.

Summary & Managerial Insights

We elaborate on the summary insights as follows:

- **Align to dependencies**: In a general network, inventory policies must ensure that paired inputs (e.g., C0→C2/C3 and C1→C4) replenish in sync so end-items can be produced when needed. DCL's learned averages achieve this alignment by holding leaner stocks at slower upstream buffers and reinforcing faster ones feeding critical mid-tier nodes.
- **Static tuning vs. dynamic alignment**: WF+GBS's capacity-aware static base-stock adjustments deliver the principal cost benefit by addressing average bottlenecks. DCL refines further by aligning average inventories to the production and penalty structures, yielding additional savings when capacity slack exists. Under extreme tightness, its gains shrink.
- **Service-prioritization via alignment**: By focusing limited inventory where upstream and mid-tier paths align for the highest-penalty end-item, DCL sustains its RLIP even when overall stock is leaner. Managers can replicate this idea: rather than uniformly raising safety stocks under capacity, target dynamic alignment of paired buffers to critical products.
- **Interpret figures directly**: The average-inventory charts illustrate where DCL holds less at slower nodes (C0), more at faster ones (C1), and sets mid-tier/end stocks to match production dependencies. The service/backlog plots show how these averages translate into maintained or improved fill at the most critical end-item, at the cost of marginal extra backlog elsewhere. The cost breakdown charts confirm the net benefit.

Conclusion of experiment Under *moderate capacity* ($Q_g = 10$), WF-GBS removes most capacity refusals, and DCL's dynamic reallocation of ≈4–5 units cuts total cost by an additional 1.4 %. Under *tight capacity* ($Q_g = 9$), WF-GBS's gives largest savings (≈2%); DCL's mid-period tweaks can bring only a additional ≈0.5 % improvement. This confirms that, while state-dependent trained DCL policy always adds benefit, its marginal return shrinks as the network depth grows together with tight capacity constraints.

Chapter 7

Conclusion, Discussion & Future Research

This thesis tackles inventory management in multi-echelon supply networks with low-volume, high-mix demand, divergent/convergent flows, and shared-capacity constraints, as exemplified by ASML's operations. We extended on the fully-convergent approach of Van Dijck et al. (2024) and adapted their Markov Decision Process (MDP) and action-space decomposition suited to discrete orders to accommodate divergent allocations, multiple end items, and shared capacities. Based on this model, we developed and evaluated:

- A network-compatible base-stock sizing approach (G-MATCH) that respects the general network graph and lead times, extending Rong et al. (2017) beyond distribution-only settings and demonstrating roughly 90% cost reduction over their proposed base-stocks general extension in uncapacitated tests.
- A family of allocation rules, including standard Water-Filling and reservation variants; the standard version emerged as most powerful when combined with appropriate base-stocks, while reservation-based variants show promise if extended with lookahead or priority weighting.
- A Guided Base-Stock (GBS) optimization that iteratively adjusts static targets for shared-capacity groups via a capacity-constrained ratio metric, yielding 5–8% cost improvement when capacity is binding.
- A Deep Controlled Learning (DCL) method for discrete, divergent-aware DRL: we decomposed outputs per node similar to Van Dijck et al. (2024) but handle simultaneous allocations among successors sharing predecessors; we engineered input features for the neural network combining global and node-specific context (improving costs by over 5% relative to a baseline feature set). We used a shared neural network with an output layer covering the entire joint action space (a concatenation of all node-specific action spaces $\sum_n(actionDim_n)$). During inference, outputs are dynamically masked to select only feasible actions for the current node. We also conducted systematic tuning of DCL parameters (feature subsets, horizon multiplier, network architectures).

Extensive experiments on 5-, 6-, and 7-node networks under varied lead-time and capacity scenarios yielded these insights:

Heuristic performance: G-MATCH base-stocks combined with Water-Filling allocation outperformed serial or surrogate methods by large margins (e.g., 60–90% cost reduction over

state-of-the-art distribution heuristics misaligned with the true network). Under capacity constraints, GBS further improved costs by roughly 5–8% over unmodified base-stocks, by raising targets at nodes most frequently constrained.

DRL refinements and observed limits: DCL, initialized with the best heuristic rollout, achieved higher gains (up to 2–5%) under tight capacity in the 5- and 6-node systems compared to looser capacity settings (1–2%). This is expected, as tighter capacity constraints reduce the set of feasible actions, making the learning process more focused and increasing the likelihood of identifying effective allocation decisions. Yet in the full 7-node network, the additional benefit under tight capacity unexpectedly shrank below the moderate-capacity gains. This reversal may reflect two possible factors: first, scalability challenges in our neural policy as network depth and action dimensionality grow; and second, a low room for improvement, once GBS targets have already filled nearly all capacity groups, resulting in a system with simultaneous bottlenecks and scarce upstream availability that leave little headroom to reallocate stock without incurring expensive shortages. Therefore, when training, we observe that the complexity of a supply network, exemplified by longer lead times, an increasing number of nodes and echelons, as well as greater asymmetry in lead times or backlog penalties, inherently elevates the effort. This effort also scales with capacity size, driven by end-item demand. Training becomes especially challenging not only as the capacity increases, leading to a broader action space, but also with tighter capacity restrictions in networks with more than two echelons (e.g., fully tight 7-node).

Modeling and methodological contributions: We extended the fully-convergent MDP modeling of Van Dijck et al. (2024) to handle general divergent/convergent topologies with shared capacities and multiple end items. We extended the approach of Rong et al. (2017) and validated that base-stock computation requires compatibility between network structure and lead times, as shown by G-MATCH's superiority over the hybrid-serial-structure and general-timing of their method. For DRL, we adapted output decomposition from Van Dijck et al. (2024) to simultaneous allocations among successors sharing predecessors and enriched input features with global and node-specific context; these design choices improved learning stability and policy quality in non-trivial networks.

Limitations and future directions:

- **Scalability in larger/deeper networks:** As depth and lead times increase, training DCL may become slow. Partitioned learning (e.g., separate neural networks for divergent and single-successor nodes) could help find targeted features, improve learning and reduce state-action complexity, but coordinating across partitions and the credit-sharing requires careful design and is an open problem.
- **Adaptive allocation sequencing:** Our rollout applies decisions in fixed topological order, allowing simultaneous multi-node decisions for nodes with at least one common predecessor; however, shared capacity usage remains biased by this order. A further extension of the heuristic would be to allow simultaneous decisions in nodes with a shared capacity group, but here there is no need for hard water-filling approaches but a proportional shortage-based allocation would suffice as capacity is drawn from a single source. This could better capture real-world simultaneity, though representing it within an MDP while maintaining feasibility is complex.
- **Reservation enhancements:** Reservation-based allocation rule (namely WFR) could incorporate brief lookahead or priority-weighted simulations to decide when

reserving inventory truly reduces downstream risk, rather than using fixed or purely myopic reserves.

- **Extension to non-stationary demand:** Adapting our DRL approach to non-stationary demand remains a challenging open problem. Van Dijck et al. (2024) proposed a time-varying demand model for a pure-assembly system with a single end-item; extending this to our setting is substantially more complex due to the presence of multiple end items with (possibly correlated) evolving demand distributions. This significantly increases the state and training space complexity. A potential path forward for our heuristic could involve periodically adjusting base-stock levels based on distributions drawn from forecasted demand scenarios, but doing so reliably and efficiently would require careful design to avoid excessive computational burden.

In ASML-like networks, a capacity-aware base-stock adjustment (GBS) combined with Water-Filling allocation captures the majority of cost savings with high efficiency and interpretability. For example, GBS optimization on a 7-node system over 1000 periods completes in under 30 seconds. DRL-based control provides additional—but smaller—gains by dynamically aligning buffer stocks and prioritizing high-penalty items in response to transient imbalances, though these benefits only materialize when capacity slack exists. Moreover, training DRL policies for larger, deeper networks is computationally intensive (e.g., 12 minutes on a supercomputer for the 7-node case, estimated 8+ hours on standard hardware), with no guarantee of convergence and limited interpretability of the resulting policy.

Managerial implications: In ASML-like networks, a capacity-aware base-stock adjustment (GBS) combined with Water-Filling allocation captures the majority of cost savings with high efficiency and interpretability. For example, running the heuristic on a 7-node system over 1000 periods completes in under 30 seconds. DRL-based dynamic control yields further, smaller improvements by synchronizing paired buffers and prioritizing high-penalty items when temporary imbalances occur, though these benefits only materialize when capacity slack exists. Moreover, training DRL policies for larger, deeper networks is computationally intensive (e.g., 12 minutes on a supercomputer for the 7-node case, estimated 8+ hours on standard hardware), with no guarantee of convergence and limited interpretability of the resulting policy. Furthermore, when capacity groups are saturated in networks with more than two echelons, additional gains from our DRL-based method are limited. In such cases, it may be more effective to invest in capacity expansion or explore enhancements to the learning approach itself. Nonetheless, insights from DCL's learned policy, such as how to balance upstream inventories given lead-time asymmetries and penalties, can inform enhancements to simpler, transparent heuristics or decision-support tools.

In summary, this work shows how a focused heuristic, grounded in correct network structure and extended for capacity constraints, forms a strong baseline, and how a targeted DRL method can refine it further where flexibility exists and its capabilities allow. The combination preserves interpretability and computational tractability while capturing transient, state-dependent opportunities that static rules overlook, offering a practical blueprint for inventory control in modern, capacity-constrained multi-echelon supply networks.

Bibliography

Akkerman, F., Begnardi, L., Lo Bianco, R., Temizoz, T., Mes, M., and van Jaarsveld, W. (2023). DynaPlex.

ASML (2024). Asml investor day 2024. Accessed: March 24, 2025.

Axsäter, S. (2015). *Inventory control*, volume 225. Springer.

Axsäter, S. (1990). Simple solution procedures for a class of two-echelon inventory problems. *Operations Research*, 38(1):64–69.

Axsäter, S. and Rosling, K. (1993). Installation vs. echelon stock policies for multilevel inventory control. *Management Science*, 39(10):1274–1280.

Chong, A., Lo, C., and Weng, X. (2017). The business value of it investments on supply chain: A contingency perspective. *Journal of Business Research*, 80:37–46.

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

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

De Kok, T. G. and Visschers, J. W. (1999). Analysis of assembly systems with service level constraints. *International Journal of Production Economics*, 59(1–3):313–326.

Diks, E. and De Kok, A. (1998). Optimal control of a divergent multi-echelon inventory system. *European Journal of Operational Research*, 111(1):75–97.

Doğru, M. K., de Kok, A. G., and van Houtum, G. J. (2009). A numerical study on the effect of the balance assumption in one-warehouse multi-retailer inventory systems. *Flexible Services and Manufacturing Journal*, 21(3-4):114–147.

Eppen, G. (1981). Centralized ordering policies in a multi-warehouse system with lead times and random demand. *Multi-level production/inventory control systems*, pages 51–67.

Eruguz, A. S., Sahin, E., Jemai, Z., and Dallery, Y. (2016). A comprehensive survey of guaranteed-service models for multi-echelon inventory optimization. *International Journal of Production Economics*, 172:110–125.

Fleuren, T., Yasemin, M., Hendriks, M., and Renata, S. (2022). Tactical production planning and strategic buffer placement under demand and supply uncertainty in the high-tech manufacturing industry. Working paper.

Geevers, K., Van Hezewijk, L., and Mes, M. R. K. (2024). Multi-echelon inventory optimization using deep reinforcement learning. *Central European Journal of Operations Research*, 32(3):653–683.

Geng, N. and Jiang, Z. (2009). A review on strategic capacity planning for the semiconductor manufacturing industry. *International journal of production research*, 47(13):3639–3655.

Giannoccaro, I. and Pontrandolfo, P. (2002). Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2):153–161.

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., and Zhang, D. J. (2022). 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.

Graves, S. C. (1985). A multi-echelon inventory model for a repairable item with one-for-one replenishment. *Management Science*, 31(10):1247–1256.

Graves, S. C. and Schoenmeyr, T. (2016). Strategic safety-stock placement in supply chains with capacity constraints. *Manufacturing amp; Service Operations Management*, 18(3):445–460.

Graves, S. C. and Willems, S. P. (2003). Optimizing strategic safety stock placement in supply chains. *Manufacturing Service Operations Management*, 2(1):68–83.

Harsha, P., Jagmohan, A., Kalagnanam, J., Quanz, B., and Singhvi, D. (2025). Deep policy iteration with integer programming for inventory management. *Manufacturing Service Operations Management*, page msom.2022.0617.

Huh, W. T., Janakiraman, G., and Nagarajan, M. (2016). Capacitated multiechelon inventory systems: Policies and bounds. *Manufacturing Service Operations Management*, 18(4):570–584.

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

Madanchian, M. and Taherdoost, H. (2024). Ai-powered innovations in high-tech research and development: From theory to practice. *Computers, Materials & Continua*, 81(2).

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

Nadar, E., Akan, M., and Scheller-Wolf, A. (2014). Technical note—optimal structural results for assemble-to-order generalized m -systems. *Operations Research*, 62(3):571–579.

Oroojlooyjadid, A., Nazari, M., Snyder, L., and Takáč, M. (2021). A deep q-network for the beer game: A deep reinforcement learning algorithm to solve inventory optimization problems.

Peng, Z., Zhang, Y., Feng, Y., Zhang, T., Wu, Z., and Su, H. (2019). Deep reinforcement learning approach for capacitated supply chain optimization under demand uncertainty. In *2019 Chinese Automation Congress (CAC)*, page 3512–3517, Hangzhou, China. IEEE.

Pirhooshyaran, M. and Snyder, L. V. (2021). Simultaneous decision making for stochastic multi-echelon inventory optimization with deep neural networks as decision makers. (arXiv:2006.05608). arXiv:2006.05608 [cs].

Rong, Y., Atan, Z., and Snyder, L. V. (2017). Heuristics for base-stock levels in multi-echelon distribution networks. *Production and Operations Management*, 26(9):1760–1777.

Rosling, K. (1989). Optimal inventory policies for assembly systems under random demands. *Operations Research*, 37(4):565–579.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Sherbrooke, C. C. (1968). Metric: A multi-echelon technique for recoverable item control. *Operations Research*, 16(1):122–141.

Simchi-Levi, D. and Zhao, Y. (2012). Performance evaluation of stochastic multi-echelon inventory systems: A survey. *Advances in Operations Research*, 2012:1–34.

Simpson, K. F. (1958). In-process inventories. *Operations Research*, 6(6):863–873.

Smirnov, D., Van Jaarsveld, W., Atan, Z., and De Kok, T. (2021). Long-term resource planning in the high-tech industry: Capacity or inventory? *European Journal of Operational Research*, 293(3):926–940.

Snyder, L. V., Atan, Z., Peng, P., Rong, Y., Schmitt, A. J., and and, B. S. (2016). Or/ms models for supply chain disruptions: a review. *IIE Transactions*, 48(2):89–109.

Stranieri, F., Stella, F., and Kouki, C. (2024). Performance of deep reinforcement learning algorithms in two-echelon inventory control systems. *International Journal of Production Research*, 62(17):6211–6226.

Sucky, E. (2009). The bullwhip effect in supply chains—an overestimated problem? *International Journal of Production Economics*, 118(1):311–322. Special Section on Problems and models of inventories selected papers of the fourteenth International symposium on inventories.

Sutton, R. S. and Barto (2018). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Temizöz, T., Imdahl, C., Dijkman, R., Lamghari-Idrissi, D., and Jaarsveld, W. v. (2023). Deep controlled learning for inventory control. (arXiv:2011.15122). arXiv:2011.15122 [cs].

Van Dijck, T., Fleuren, T., Temizoz, T., Merzifonluoglu, Y., Hendriks, M., and Van Jaarsveld, W. (2024). Inventory planning in capacitated high-tech assembly systems under non-stationary demand. *SSRN Electronic Journal*.

Van Hezewijk, L., Dellaert, N., Van Woensel, T., and 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.

Vanvuchelen, N., De Moor, B. J., and Boute, R. N. (2025). The use of continuous action representations to scale deep reinforcement learning for inventory control. *IMA Journal of Management Mathematics*.

Wang, T. and Hong, L. J. (2023). Large-scale inventory optimization: A recurrent neural networks–inspired simulation approach. *INFORMS Journal on Computing*, 35(1):196–215.

Woerner, S., Laumanns, M., and Wagner, S. M. (2016). Simulation-based optimization of capacitated assembly systems under beta-service level constraints. *Decision Sciences*, 49(1):180–217.

Zipkin, P. H. (2000). Foundations of inventory management. *McGraw-Hill/Irwin*.

Appendix A

A.1 Warm-Up Period Estimation Using Moving Averages

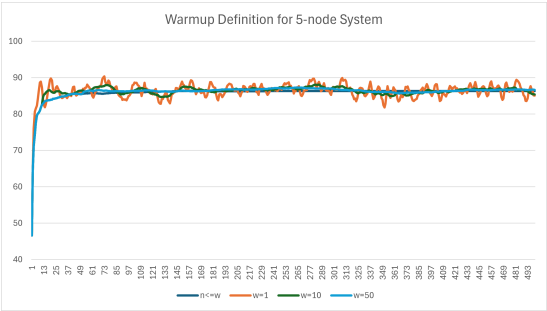
To ensure accurate steady-state performance evaluation, we need to remove the initial transient behavior of the system. We estimate the appropriate warm-up period using a visual moving average method applied to the time series of period costs obtained from simulation runs.

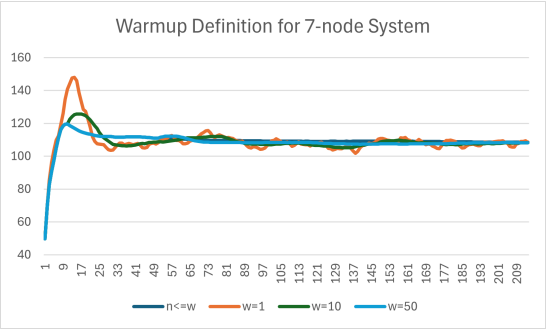
Let Y_t denote the cost observed in simulation period t, for $t = 1, 2, \ldots, T$. To smooth short-term fluctuations and reveal longer-term trends, we compute moving averages of Y_t over windows of length w. The moving average at time t, denoted $\bar{Y}_t^{(w)}$, is defined as:

$$\bar{Y}_t^{(w)} = \frac{1}{w} \sum_{i=0}^{w-1} Y_{t-i} \quad \text{for} \quad t \geq w$$

We evaluate the moving averages for window sizes $w = 1$, $w = 10$, and $w = 50$, across two different network configurations: 5-node from Figure 6.1 and 7-node from Figure 3.1. For each case, the time series of period costs was plotted along with its smoothed counterparts. Figure A.1 shows that cost behavior stabilizes after about 60 periods in both network instances. Beyond this, plots remain flat, indicating the end of the transient phase. Thus, we conservatively choose 60 periods as the warm-up duration for simulations.

FIGURE A.1: Warmup Estimation with Moving Averages for the 5-node (left) and 7-node (right) cases





A.2 Benchmarking 36 Heuristic Configurations on the 7-Node System

TABLE A.1: Experiment 1: 7-node, uncapacitated, all 36 configurations (per-period mean cost and standard deviation)

Policy	Bound	Allocation	Mean Cost	Std. Dev.
G-MATCH				
	UP	WF	106.311	0.1298
	DOWN	WF	106.686	0.1276
	UP	WFR	108.220	0.1397
	DOWN	WFR	108.350	0.1359
	UP	WFRR	108.859	0.1378
	DOWN	WFRR	109.027	0.1343
	UP	WFFS	111.621	0.1540
	DOWN	WFFS	112.034	0.1540
	UP	RandomWF	109.704	0.1459
	DOWN	RandomWF	110.739	0.1483
	UP	FCFS	112.470	0.1669
	DOWN	FCFS	115.383	0.1771
FullS-MATCH				
	UP	WF	157.526	0.1112
	DOWN	WF	153.178	0.1078
	UP	WFR	157.526	0.1112
	DOWN	WFR	153.178	0.1078
	UP	WFFS	157.526	0.1112
	DOWN	WFFS	153.178	0.1078
	UP	WFRR	159.310	0.1186
	DOWN	WFRR	154.683	0.1121
	UP	RandomWF	163.475	0.1332
	DOWN	RandomWF	158.272	0.1237
	UP	FCFS	169.691	0.1653
	DOWN	FCFS	163.286	0.1489
S-MATCH				
	UP	WF	765.404	1.0795
	DOWN	WF	806.721	1.0911
	UP	WFR	765.404	1.0795
	DOWN	WFR	806.721	1.0911
	UP	WFFS	805.060	1.1830
	DOWN	WFFS	846.016	1.1955
	UP	WFRR	770.734	1.0733
	DOWN	WFRR	810.777	1.0855
	UP	RandomWF	911.091	1.1986
	DOWN	RandomWF	981.223	1.2516
	UP	FCFS	1064.580	0.9804
	DOWN	FCFS	1151.010	0.9772

A.3 5-node Uncapacitated Case Under Different Scenarios

Lead Times	Policy	MeanC	HoldC	Backl.C	% Backlog	$\mathbf{RLIP}_{M3}$	$\mathbf{RLIP}_{M_4}$	$\mathbf{Inv}_{B2}$	$\mathbf{Inv}_{M3}/\mathbf{Inv}_{M4}$
(4,4,2,3,2)									
	FCFS-U	103.95	78.03	25.82	24.07%	92.34%	92.55%	4.72	4.13 / 4.04
	WFRandom-U	103.62	78.03	25.59	24.02%	92.06%	92.88%	4.72	4.09 / 4.07
	WF-U	103.30	78.00	25.30	23.91%	91.92%	93.13%	4.72	4.06 / 4.10
	WFFS-U	104.51	77.67	26.84	24.93%	91.30%	92.81%	4.84	4.01 / 4.06
	WFR-U	103.50	77.84	25.65	24.21%	91.77%	93.07%	4.76	4.04 / 4.08
	S1–DCL	101.73	74.60	27.13	25.45%	91.61%	92.43%	2.66	3.97 / 3.98
(4,4,2,6,2)									
	FCFS-U	116.03	89.33	26.70	22.82%	92.75%	91.79%	3.83	6.01 / 3.95
	WFRandom-U	115.30	89.28	26.02	22.49%	92.35%	92.45%	3.83	5.93 / 4.02
	WF-U	114.57	89.23	25.34	22.15%	92.03%	93.04%	3.83	5.87 / 4.07
	WFFS-U	116.37	88.73	27.64	23.38%	90.89%	92.74%	4.03	5.77 / 4.03
	WFR-U	114.78	89.00	25.79	22.47%	91.82%	92.97%	3.90	5.83 / 4.06
	S2–DCL	113.16	83.38	29.78	25.22%	90.21%	92.15%	2.02	5.55 / 3.91
(5,5,2,4,1)									
	FCFS-U	104.07	79.88	24.19	22.45%	94.01%	92.11%	3.91	5.31 / 3.22
	WFRandom-U	103.08	79.79	23.29	21.99%	93.61%	92.88%	3.91	5.23 / 3.28
	WF-U	102.08	79.72	22.36	21.42%	93.13%	93.73%	3.91	5.13 / 3.36
	WFFS-U	104.47	79.27	25.20	22.90%	91.64%	93.42%	4.12	5.03 / 3.33
	WFR-U	102.29	79.50	22.79	21.77%	92.88%	93.70%	3.98	5.09 / 3.35
	S3–DCL	100.65	75.01	25.64	24.06%	91.86%	93.01%	2.23	4.90 / 3.23
(2,2,6,3,1)									
	FCFS-U	102.33	72.90	29.43	25.68%	93.60%	89.73%	6.85	4.24 / 3.16
	WFRandom-U	99.83	72.64	27.18	25.34%	92.12%	92.01%	6.85	4.11 / 3.25
	WF-U	97.43	72.37	25.06	24.83%	91.11%	93.88%	6.85	4.00 / 3.31
	WFFS-U	97.96	72.23	25.73	25.31%	90.69%	93.85%	6.91	3.97 / 3.30
	WFR-U	97.59	72.29	25.30	25.02%	90.95%	93.88%	6.88	3.99 / 3.30
	S4–DCL	96.89	70.95	25.94	25.44%	91.28%	93.29%	6.91	4.00 / 3.27

3-Retailer case								$\mathbf{RLIP}_{M_5}$	$\mathbf{Inv}_{B2}$	$\mathbf{Inv}_{M3}/\mathbf{Inv}_{M4}/\mathbf{Inv}_{M5}$
	FCFS-U	143.52	101.00	42.52	35.71%	96.44%	92.26%	86.00%	5.26	4.11 / 3.66 / 4.00
	WFRandom-U	138.65	100.86	37.79	35.51%	93.91%	91.62%	90.31%	5.26	4.04/ 3.89 / 3.75
	WF-U	135.65	100.41	35.25	34.65%	94.92%	91.44%	91.18%	5.26	3.99 / 3.88 / 3.75
	WFFS-U	137.98	99.57	38.41	36.97%	94.70%	89.92%	90.84%	5.57	3.83 / 3.84 / 3.71
	WFR-U	136.29	100.03	36.26	35.47%	94.87%	90.91%	91.09%	5.38	3.92 / 3.86 / 3.74
	DCL	133.63	98.67	34.96	35.89%	92.78%	90.97%	92.87%	3.87	3.92 / 4.00 / 3.52

TABLE A.2: Experiment 2 results: 5-node uncapacitated case under four lead-time scenarios.

A.4 Training of Classifiers for Uncapacitated Experiments

In this appendix we report on the convergence behavior, final performance and computational cost of the six neural-network policies trained under the Small Uncapacitated Lead-Time Stress Test scenarios. Figure A.2 shows, for each case, the evolution of training and validation loss over epochs, and Table A.3 summarizes the key metrics.

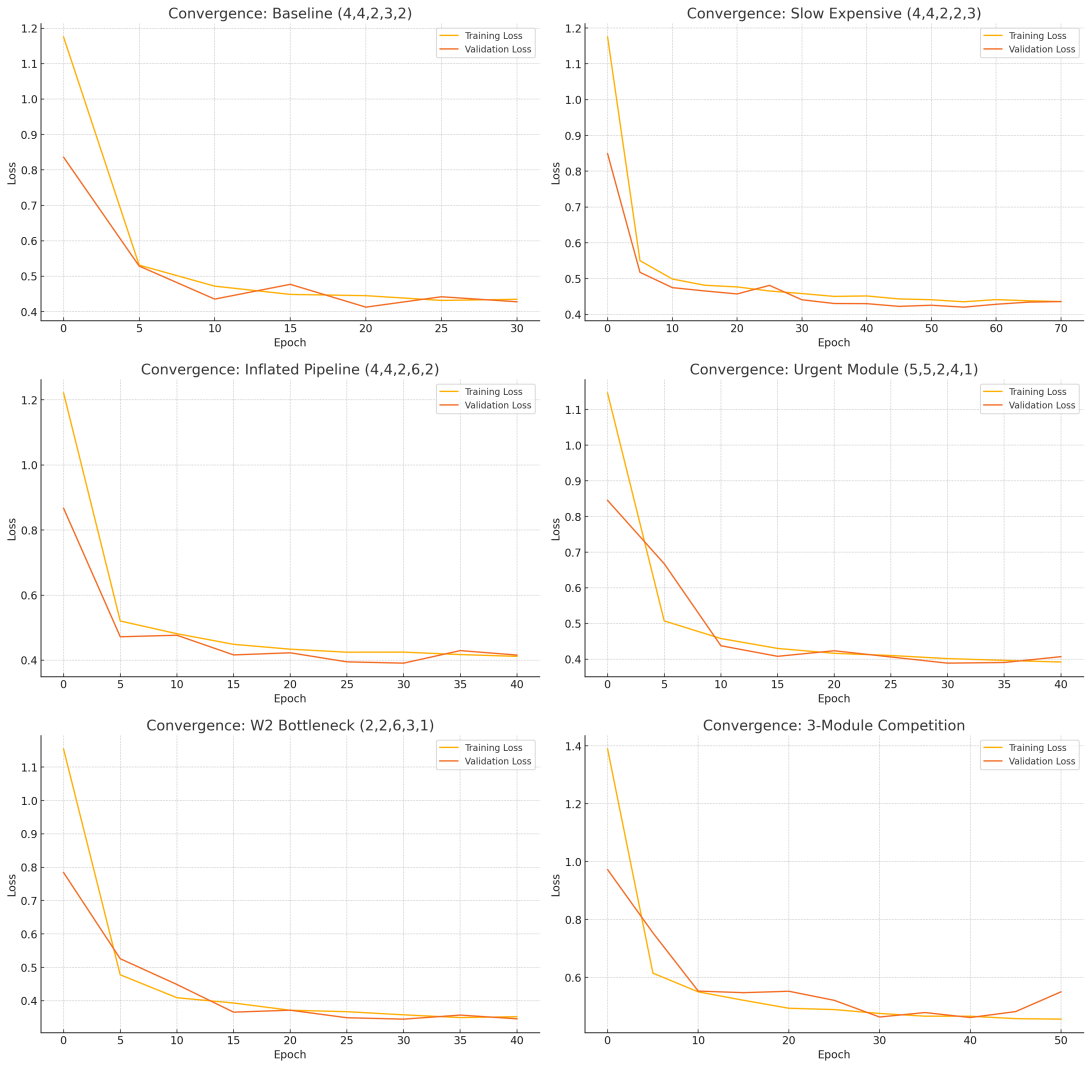


FIGURE A.2: Training (gold) and validation (orange) loss versus epoch for each network configuration.

Classifier Convergence and Performance Across all six Small-Uncapacitated cases, each 5- or 6-node MLP converged in 31–71 epochs (182–413s) to stable validation losses between 0.34 and 0.46, yielding cost improvements of –0.13 to –0.36 versus the G-MATCH+WF. In particular, the Baseline (4,4,2,3,2) network reached a val-loss of 0.41 at epoch 20 (–0.36 cost, 31ep/182s), the Slow-Expensive (4,4,2,2,3) case converged in 71ep (0.42 val-loss, –0.24 cost, 339s) with minor overfitting after epoch 25, and the Inflated-Pipeline (4,4,2,6,2) ran 41ep (0.39 val-loss, –0.33 cost, 258s). The Urgent-Module (5,5,2,4,1) also took 41ep (0.39 val-loss, –0.31 cost, 245s), while the W2-Bottleneck (2,2,6,3,1) achieved its best at epoch 30 (0.34 val-loss, –0.12 cost, 41ep/230s) with minimal train–val gap. Finally, the 3-Module

Case	Epochs	Best Val $\mathcal{L}$	Cost Imp.	Time (s)
Baseline (4,4,2,3,2)	31	0.41	-0.36	182.2
Slow Expensive (4,4,2,2,3)	71	0.42	-0.25	339.2
Inflated Pipeline (4,4,2,6,2)	41	0.39	-0.33	258.1
Urgent Module (5,5,2,4,1)	41	0.39	-0.31	244.8
W2 Bottleneck (2,2,6,3,1)	41	0.34	-0.13	230.4
3-Module Competition	51	0.46	-0.29	413.2

TABLE A.3: Summary of Training for Small Uncapacitated Classifiers.

Competition needed 51ep (0.46 val-loss, –0.29 cost, 413s) to stabilize in this more complex six-node topology. All classifiers show smooth training–validation improvements and deliver consistent cost reductions under lead-time stress.

A.5 Training of Classifiers for Small Capacitated Experiments

In this appendix we report on the convergence behavior, final performance and computational cost of the neural-network policies trained under the Small Capacitated Lead-Time Variations. Figures A.3 and A.4 shows, for each case, the evolution of training and validation loss over epochs, and Table A.4 summarizes the key metrics.

Results for S1: (4, 4, 2, 3, 2)

Capacity case	Best Val.$\mathcal{L}$ at/ Epoch	Cost Imp.	Epochs	Train time(s)
C1 Tight G3	0.40/ 15	–1.25	31	296
C2 Tight G2	0.44/ 35	–0.6	51	352
C3 +12.5%	0.33/ 45	–0.84	61	420
C4 +25%	0.48/ 35	–0.32	46	342
C5 +37.5%	0.44/ 15	–0.20	31	257

Results for S4: (4, 4, 6, 3, 2)

Capacity case	Best Val.$\mathcal{L}$ at/ Epoch	Cost Imp.	Epochs	Train time(s)
C1 Tight G3	0.49/ 20	–1.14	36	260
C2 Tight G2	0.46/ 35	–0.28	51	284
C3 +12.5%	0.48/ 70	–0.84	76	408
C4 +25%	0.50/ 20	–0.26	36	229
C5 +37.5%	0.51/ 45	–0.11	71	385

Results for S3: (4, 4, 2, 6, 2)

Capacity case	Best Val.$\mathcal{L}$ at/ Epoch	Cost Imp.	Epochs	Train time(s)
C1 Tight G3	0.47/ 35	–1.32	41	357
C2 Tight G2	0.43/ 45	–0.27	61	357
C3 +12.5%	0.46/ 55	–0.82	66	390
C4 +25%	0.49/ 46	–0.49	46	318
C5 +37.5%	0.44/ 31	–0.65	46	295

Results for S5: (4, 5, 3, 1, 3, 2))

Capacity case	Best Val.$\mathcal{L}$ at/ Epoch	Cost Imp.	Epochs	Train time(s)
C1 Tight G3	0.46/ 15	–0.60	31	582
C2 Tight G2	0.47/ 75	–0.22	91	1087
C3 +12.5%	0.42/ 40	–0.36	56	854
C4 +25%	0.52/ 35	–0.25	36	622
C5 +37.5%	0.55/ 30	–0.15	31	628

TABLE A.4: Summary of Training for Small Capacitated Classifiers.

In the four waiting time scenarios (inflated channeling, W2 bottleneck, and 3-module competition), our reduced capacity cases consistently converged in less than 100 epochs with modest training times (<20 min per case), demonstrating robust learning across diverse network structures and capacity constraints. The Tight G3 variant (C1) repeatedly achieved the largest cost improvements (from –1.13 to –1.32) with relatively low validation losses (0.40–0.47) and early stopping points (15–35 epochs), indicating strong policy gains under strict constraints in the early stages. In contrast, the +12.5% capacity relaxation (C3) often achieved the next best cost improvements (from –0.84 to –0.36), albeit with a higher number of epochs (40–75), suggesting diminishing returns from additional capacity relaxations. The intermediate variants C2 (strict G2) and C4 (+25%) yielded moderate cost benefits (from –0.22 to –0.27 and from –0.25 to –0.49, respectively) with validation

losses clustered around 0.43–0.52, while the most flexible network (+37.5%, C5) converged more quickly but offered the smallest improvements (from –0.11 to –0.65). In general, strict capacity adjustments generate the greatest cost savings, while excessive capacity slack accelerates convergence at the expense of policy performance.

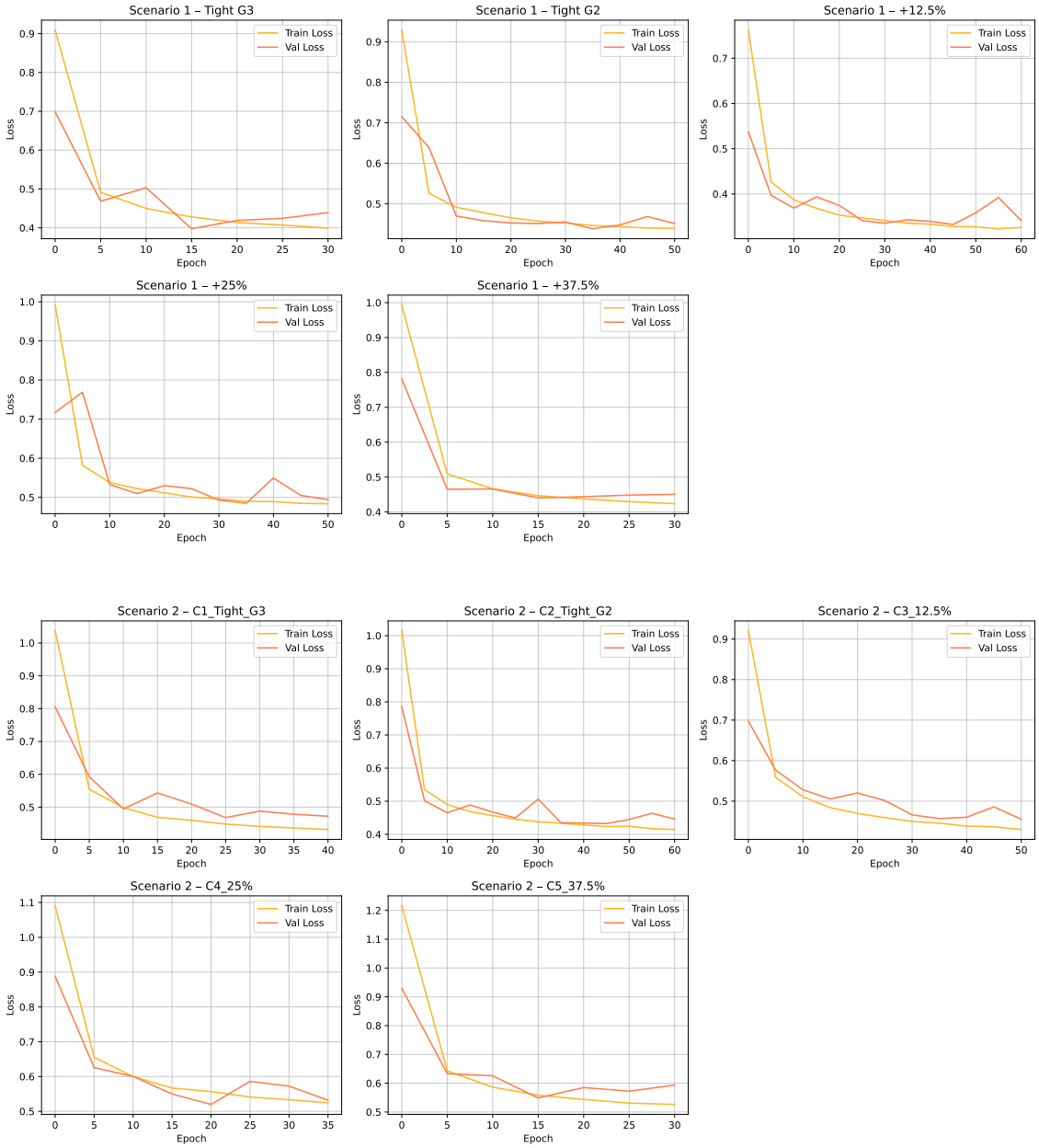


FIGURE A.3: Training (gold) and validation (orange) loss versus epoch for each network configuration (Capacitated Scenarios 1–2).

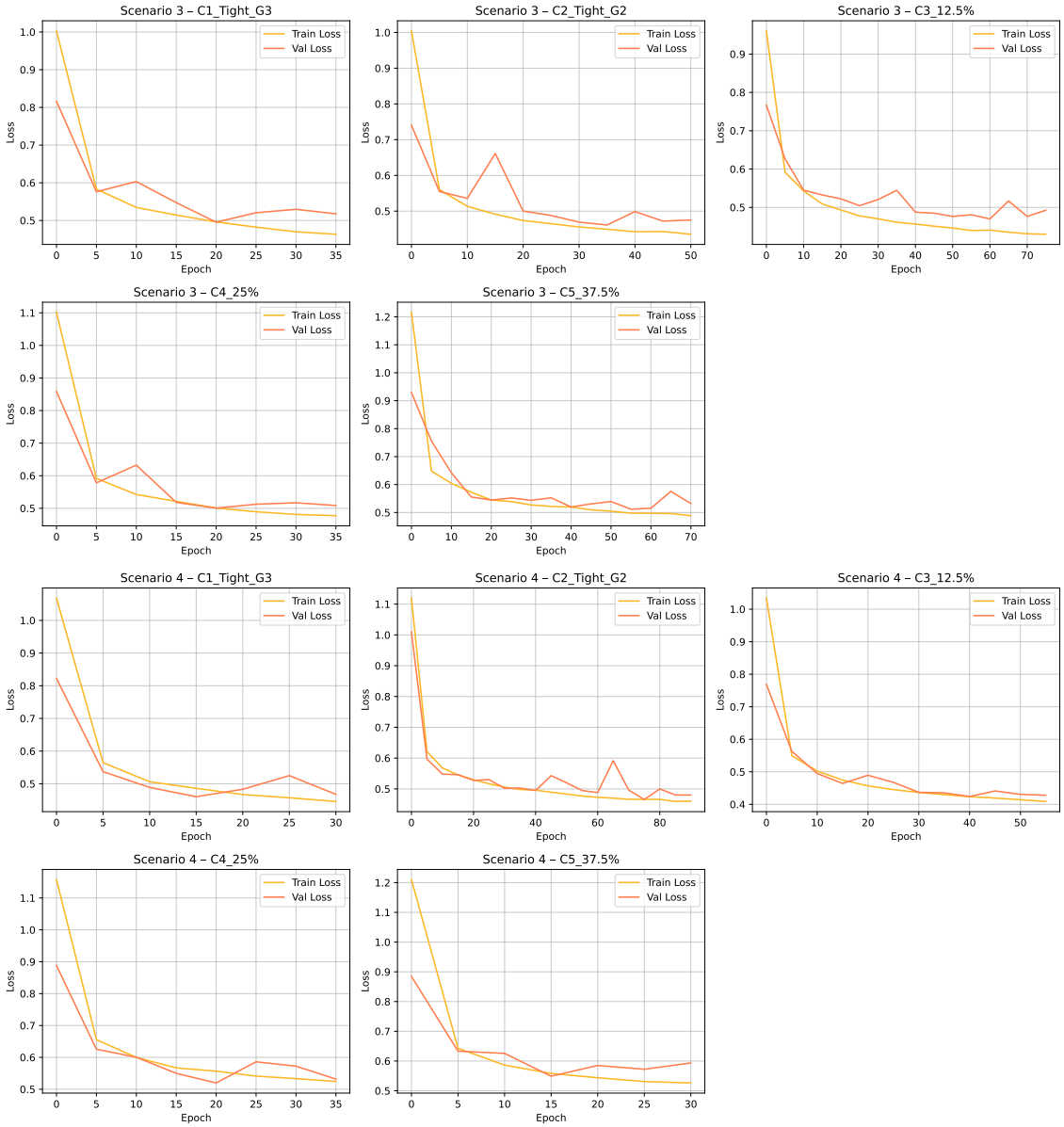


FIGURE A.4: Training (gold) and validation (orange) loss versus epoch for each network configuration (Capacitated Scenarios 3–4).

A.6 Training of Classifiers for Large 7-node Experiments

Capacity case	Best Val.$\mathcal{L}$ at/ Epoch	Cost Imp.	Epochs	Train time(s)
Tight Cap	0.28/ 40	–0.09	56	885.7
Moderate Cap	0.31/ 40	–0.20	56	1067.0

TABLE A.5: Summary of Training for Large Capacitated Classifiers.

For the 7-node tightly capacitated system, the validation curve shows pronounced oscillations, where after an initial sharp decline to around epoch 15, it rises again by 20% to recover at epoch 40, while training loss decreases steadily. This "irregular" behavior, together with a marginal improvement in cost around zero in later epochs, suggests that the model has difficulty dealing with the tight capacities and learning better policies. The moderate capacity network shows a better behavior, with a slightly slower initial drop in validation loss, but ultimately reaching a comparable minimum of about 0.31 in epoch 40 with a small overfitting thereafter. Both cases seems to struggle reaching to a convergent point.

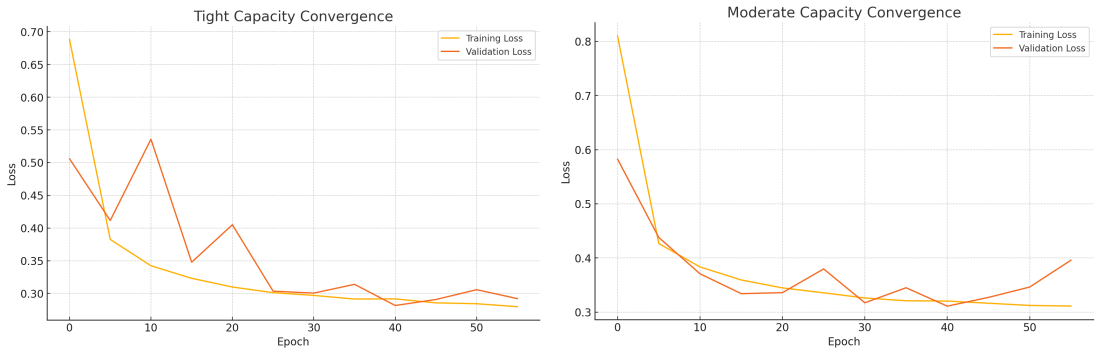


FIGURE A.5: Training (gold) and validation (orange) loss versus epoch for Tight (left) and Moderate (right) Capacitated 7-node system.