Inventory Optimization Under Uncertainty: Adaptive Decision-Making with Reinforcement Learning in the Quality-Dependent Newsvendor Problem

A Thesis

submitted to the designated

by the Assembly

of the Department of Computer Science and Engineering

Examination Committee

by

Nefeli Eleftheria Sextou

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER SYSTEMS ENGINEERING

WITH SPECIALIZATION

IN DATA SCIENCE AND ENGINEERING

University of Ioannina

School of Engineering

Ioannina 2025

Examining Committee:

- **Kostantinos Parsopoulos**, Professor, Department of Computer Science and Engineering, University of Ioannina (Advisor)
- **Konstantina Skouri**, Professor, Department of Mathematics, University of Ioannina
- **Ioannis Konstantaras**, Associate Professor, Department of Business Administration, University of Macedonia

Acknowledgements

I would like to begin by expressing my deepest gratitude to my advisor, Professor Konstantinos Parsopoulos, whose guidance, encouragement, and insight have been invaluable throughout this thesis. Beyond supporting this work, he first introduced me to the field of optimization and operations research during my undergraduate studies and thesis, providing guidance that shaped my approach to research and laid the foundation for my continued academic interests.

I am also grateful to Professor Konstantina Skouri for her valuable insight, particularly during the early stages while I was learning about newsvendor and inventory problems, and to her PhD candidate, Eirini Tziora, for extending that support.

Moreover, I am especially thankful to PhD candidate Dimitra Triantali, who has been both a mentor and a friend. Her advice, feedback, and generous willingness to share knowledge have greatly enriched my work and perspective.

As a source of enduring inspiration, I wish to express my appreciation to Professor Isaac Lagaris, whose experience, mentorship, thoughtful perspectives, and expansive curiosity have continually encouraged me to broaden my thinking and interests.

My acknowledgements extend to the other members of our lab, Professor Gerasimos Meletiou, PhD candidate Adam Kypriadis, and MSc candidate Spyros Motsenigos, for contributing to an engaging and dynamic research environment.

Finally, I am profoundly grateful to my family and friends, whose encouragement, patience, and unwavering support have been a constant source of strength throughout this journey.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Nefeli Eleftheria Sextou, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2025.
Inventory Optimization Under Uncertainty: Adaptive Decision-Making with Reinforcement Learning in the Quality-Dependent Newsvendor Problem.
Advisor: Kostantinos Parsopoulos, Professor.

The present thesis investigates the application of reinforcement learning (RL) to stochastic inventory management, focusing on the Quality-Dependent Newsvendor Problem (QDNP). Moreover, a data-driven variant incorporating kernel density estimation (KDE) is considered. The QDNP models inventory decisions under uncertain demand and product quality deterioration, integrating donation as a strategic decision. Donations reduce waste by redirecting surplus products to food banks and other charitable organizations, supporting social and environmental responsibility. Also, they improve profits by leveraging corporate social responsibility (CSR) incentives, which are appealing to socially and ecologically conscious consumers, and avoid disposal costs.

To this end, a state-of-the-art deep reinforcement learning (DRL) approach, namely the Soft Actor-Critic (SAC) algorithm, is employed to maximize expected profit by simultaneously optimizing decisions on order quantities, pricing, and donations across multiple real-world store instances. In this context, SAC is introduced as an alternative to classical stochastic optimization methods, with the additional potential for knowledge transfer. The results obtained indicate that SAC can closely approximate the analytical solution, while maintaining practical accuracy. A KDE-based extension is also studied, which empirically approximates the demand component distribution instead of assuming full prior knowledge. The corresponding results showed slightly increased approximation errors but remained within acceptable thresholds. Further-

more, zero-shot policy transfer showed that pretrained SAC policies can adapt to moderate changes in demand and cost parameters, although retraining is required for larger deviations.

Using SAC within a controlled inventory problem, this study demonstrates how sequential decision-making, and stochastic uncertainty can be addressed through RL. The findings provide a concrete example of RL applied to inventory management and contribute to the growing intersection of reinforcement learning and operations management, assessing its potential for integrating data-driven components and dynamic decision-making methods.

Keywords: Quality-Dependent Newsvendor Problem, Deep Reinforcement Learning, Soft Actor-Critic, Stochastic Optimization, Policy Transfer, Data-Driven Decision Making, Inventory Management, Kernel Density Estimation

Εκτεταμενη Περιληψη

Νεφέλη Ελευθερία Σέξτου, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2025.
Βελτιστοποίηση Αποθεμάτων υπό Αβεβαιότητα: Προσαρμοστική Λήψη Αποφάσεων με Ενισχυτική Μάθηση στο Πρόβλημα του Εφημεριδοπώλη με Εξάρτηση από την Ποιότητα.
Επιβλέπων: Κωνσταντίνος Παρσόπουλος, Καθηγητής.

Η παρούσα διπλωματική εργασία διερευνά την εφαρμογή της ενισχυτικής μάθησης (reinforcement learning, RL) στη διαχείριση αποθεμάτων, εστιάζοντας στο quality-dependent newsvendor problem (QDNP) και σε μια παραλλαγή βασισμένη σε δεδομένα που ενσωματώνει εκτίμηση πυκνότητας πυρήνα (kernel density estimation, KDE). Το QDNP μοντελοποιεί τις αποφάσεις αποθεμάτων υπό αβέβαιη ζήτηση και υποβάθμιση ποιότητας προϊόντων, ενσωματώνοντας τις δωρεές ως στρατηγική επιλογή. Οι δωρεές μειώνουν τη σπατάλη με την ανακατεύθυνση των πλεονάζοντων προϊόντων σε τράπεζες τροφίμων και άλλους φιλανθρωπικούς οργανισμούς, υποστηρίζοντας την κοινωνική και περιβαλλοντική υπευθυνότητα, ενώ παράλληλα βελτιώνουν τα κέρδη αξιοποιώντας τα κίνητρα εταιρικής κοινωνικής ευθύνης (corporate social responsibility, CSR), προσελκύοντας καταναλωτές με κοινωνική και οικολογική συνείδηση και αποφεύγοντας τα κόστη απόρριψης.
Μια προηγμένη προσέγγιση βαθιάς ενισχυτικής μάθησης (deep reinfocement learning, DRL), ο αλγόριθμος Soft Actor-Critic (SAC), χρησιμοποιήθηκε για τη μεγιστοποίηση των αναμενόμενων κερδών, καθοδηγώντας ταυτόχρονα αποφάσεις σχετικά με ποσότητες παραγγελίας, τιμολόγηση και δωρεές σε πολλαπλά πραγματικά καταστήματα. Η μελέτη εισάγει τον SAC ως εναλλακτική στα κλασικά μοντέλα στοχαστικής βελτιστοποίησης, με την επιπλέον δυνατότητα μεταφοράς γνώσης. Τα αποτελέσματα δείχνουν ότι ο SAC προσεγγίζει στενά την αναλυτική λύση, διατηρώντας

πρακτική ακρίβεια. Η επέκταση του QDNP βασισμένη στο KDE, η οποία προσεγγίζει εμπειρικά την κατανομή της τυχαίας συνιστώσας της ζήτησης αντί να υποθέτει προϋπάρχουσα γνώση, οδήγησε σε ελαφρώς αυξημένα σφάλματα προσέγγισης αλλά παρέμεινε εντός αποδεκτών ορίων. Επιπλέον, αξιολογήθηκε η ικανότητα γενίκευσης και μεταφοράς των πολιτικών SAC μέσω zero-shot policy transfer, εφαρμόζοντας εκπαιδευμένες πολιτικές σε παραλλαγμένες εκδοχές των προβλημάτων χωρίς επαναεκπαίδευση, και συγκρίνοντας την απόδοσή τους με τη βέλτιστη εκ νέου εκπαίδευση.

Η μεθοδολογία της μελέτης βασίστηκε σε μια συστηματική πειραματική διαδικασία σε τέσσερα προβλήματα καταστημάτων, χρησιμοποιώντας δεδομένα από σχετικές μελέτες και προσομοιώσεις για περιπτώσεις όπου η κατανομή της τυχαίας συνιστώσας της ζήτησης δεν είναι γνωστή. Οι πολιτικές SAC εκπαιδεύτηκαν και αξιολογήθηκαν με πολλαπλές επαναλήψεις για κάθε περίπτωση, παρέχοντας στατιστικά σταθερές εκτιμήσεις των αναμενόμενων αποδόσεων. Η υλοποίηση έγινε σε Python, αξιοποιώντας Stable-Baselines3, PyTorch και Gymnasium, διασφαλίζοντας αναπαραγωγιμότητα και συνέπεια αποτελεσμάτων.

Εφαρμόζοντας τον SAC σε ένα ελεγχόμενο πρόβλημα αποθεμάτων, η μελέτη καταδεικνύει πώς η διαδοχική λήψη αποφάσεων και η στοχαστική αβεβαιότητα μπορούν να αντιμετωπιστούν μέσω της ενισχυτικής μάθησης. Τα ευρήματα παρέχουν ένα συγκεκριμένο παράδειγμα εφαρμογής της ενισχυτικής μάθησης στη διαχείριση αποθεμάτων και υπογραμμίζουν το γόνιμο έδαφος για περαιτέρω έρευνα στην ενσωμάτωση μοντελοποιητικών στοιχείων βασισμένων σε δεδομένα και δυναμικών μεθόδων λήψης αποφάσεων.

Λέξεις-Κλειδιά: Newsvendor Problem, Quality-Dependent Newsvendor Problem, Reinforcement Learning, Deep Reinforcement Learning, Soft Actor-Critic, Stochastic Optimization, Policy Transfer, Data-Driven Decision Making, Inventory Management, Kernel Density Estimation

Chapter 1

Introduction

1.1 Scope and Goals

Inventory theory is a central subfield of Operations Research (OR) and Operations Management (OM), concerned with understanding and optimizing the tradeoff between supply and demand under uncertainty. The field emerged during the post-World War II economic expansion, a period when science and industry collaborated intensely to improve the efficiency of production and distribution systems. Seminal publications from this era include *Studies in the Mathematical Theory of Inventory and Production* by Arrow, Karlin and Scarf (1958) [1], and *Analysis of Inventory Systems* by Whitin and Hadley (1963) [2]. These works laid the mathematical and conceptual foundations for the study of inventory systems, establishing the discipline as a cornerstone of applied OR. Since then, inventory theory has evolved into a rich body of models addressing increasingly complex and realistic supply chain phenomena.

Mathematical inventory models formalize the tradeoffs that firms face when managing stock under various operational and environmental constraints. Depending on the application, the relevant models incorporate diverse assumptions about key factors such as the nature of demand, cost structures, time efficiency, quality, production rates, product scope, supply chain structure, and more. Each case and their combinations lead to specialized models that reflect the operational particularities of different

industries. A classic inventory model example is the Economic Order Quantity (EOQ) model [3, 4], which yields closed-form expressions for optimal order size under deterministic demand and instantaneous replenishment. Another model is the base-stock model [5], which addresses stochastic demand in periodic review systems, outputting inventory levels that minimize expected shortage and holding costs. These two models can be generalized into the classic inventory model called the (Q, R), policy which specifies a fixed order quantity Q placed whenever the inventory level falls below a reorder point R [2, 5].

Arguably, the most recognizable stochastic model within this family is the newsvendor problem, which captures inventory decision-making under demand uncertainty during a single selling season with the goal of balancing inventory underage and overage. The newsvendor model itself has been extended in numerous directions in efforts to model the complexities of the real world. These variations may integrate and refine price decisions, selling season breakdown into individual time periods, operational costs and profits, quality modeling, demand modeling, and more

One such variation is the Quality Dependent Newsvendor Problem (QDNP) introduced by Özbilge *et al.* [6], which builds upon Petruzzi and Dada's [7] review and model extensions on price effects within a newsvendor framework. The QDNP is a two-period newsvendor problem, under demand uncertainty across the selling horizon, which incorporates product quality deterioration within demand modeling. It is framed as an expected profit maximization problem dependent on decisions regarding the initial order quantity, the pricing of the product within each time period and the percentage of the product donated.

The problem, primarily addresses the operational objectives of maximizing the expected profit and reducing disposal costs. Also, it has a broader social and environmental responsibility dimension through the inclusion of donations. By redirecting surplus to food banks and charities, a company contributes to tackling food waste, which is a global challenge with significant economic and ecological consequences, as well as food insecurity. Özbilge *et al.* [6] showed that, under certain conditions, retailers may even improve profits by donating. This is achieved as government-backed corporate social responsibility (CSR) incentives such as tax deductions, along with the appeal to socially conscious consumers can offset revenue loss due to underage or overage, while simultaneously reducing disposal costs.

The QDNP provides elegant analytical insights, showing that a retailer's donation

behavior for a product is shaped by the remaining shelf life, on-hand inventory, and the per-unit reward from CSR incentives or consumer preferences. The numerical results in the original study further demonstrate that donation can consistently reduce the expected waste compared to models without it, while the framework as a whole outperforms both single-period and two-period benchmarks. Finally, its relative advantage is especially pronounced under high uncertainty or when the second period is short [6] .

While the QDNP is typically solved analytically, its structure as a stochastic profit maximization problem makes it equally compelling to examine through numerical methods. Classical approaches in stochastic optimization can be applied for this purpose, yet reinforcement learning (RL) offers a particularly attractive alternative. Building on W. Powell's *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Stochastic Optimization* (2022) [8], RL can be situated within the broader stochastic optimization paradigm as a methodology for sequential decision-making under uncertainty. In RL, an agent learns to make sequential decisions under uncertainty, improving its policy through iterative interaction with the environment rather than relying on explicit derivation. This aligns closely with the QDNP setting, in which order quantities, pricing, and donation decisions evolve across time periods under uncertain demand and quality dynamics within the selling horizon.

Exploring RL in this context not only provides a computational complement to the analytical solution but also connects the problem to a rapidly advancing field at the intersection of machine learning and operations management, where RL is only beginning to gain traction in inventory and supply chain applications. Although deep reinforcement learning (DRL) has achieved high-profile breakthroughs in other domains, such as robotics, strategic games, and traffic optimization, its use in inventory management has been more sporadic. Boute *et al.* [9] note in their review that practical adoption is hampered by the complexity of algorithm design, the heavy computational effort required, and sensitivity in training. Nonetheless, these same limitations have sparked a wave of interest, with researchers increasingly treating RL as a promising but still maturing approach to sequential decision-making under uncertainty.

Among RL algorithms, Soft Actor-Critic (SAC) represents a strong candidate for solving the QDNP. According to its developers, Haarnoja *et al.* [10], SAC consistently outperforms several strong baseline RL methods across both simple and complex continuous tasks, and its performance is relatively insensitive to hyperparameter set-

tings, making it practical for challenging real-world problems. Its properties, which include high sample efficiency, stable learning, and robust policy generation are particularly valuable in the QDNP, where a stochastic demand component creates a highly uncertain decision environment. Furthermore, its maximum entropy framework encourages exploration while avoiding unpromising paths, maximizing expected returns and reducing the risk of convergence to suboptimal strategies.

The present work employs the Soft Actor-Critic (SAC) algorithm as a gradient-free, stochastic optimization method, positioning it as an alternative to traditional metaheuristic approaches for tackling the QDNP. The primary objective is to assess whether SAC can effectively maximize the expected profit function when order, pricing, and donation decisions are made under demand uncertainty. Beyond its role as an optimizer, SAC can offer athe advantage of knowledge transfer, which is not applicable in classic stochastic optimization methods. Specifically, a learned policy can be applied to modified instances of a problem, where customer-related parameters, costs, or rewards are perturbed to reflect possible real-world fluctuations. The present study examines the extent to which, such policy transfer is successful in the context of a real-world problem instance of the QDNP.

A second contribution is the formulation of a data-driven variant of the QDNP that relaxes the distributional assumptions of the original model. While the baseline formulation presumes full knowledge distribution characteristics of the random demand component, the proposed extension approximates the distribution using kernel density estimation (KDE). SAC is individually tuned for both the original and the data-driven model variants and applied on real-world problem instances using SAC, enabling a direct comparison of performance across settings and assessing whether the data-driven framework is a viable alternative for inventory management under demand uncertainty.

1.2 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 provides background information, introducing the Newsvendor Problem and fundamental concepts of reinforcement learning, including a taxonomy of RL algorithms. Chapter 3 presents a detailed description of the Quality-Dependent Newsvendor Problem (QDNP), its

mathematical formulation, and the proposed non-parametric extension. Chapter 4 focuses on the Soft Actor-Critic (SAC) algorithm and the proposed RL environment setup for applying SAC to the QDNP. Chapter 5 escribes the experimental setup, covering the real-world problem instances that were used as benchmarks, the tuning of algorithm parameters, and the overall methodology and implementation approach. Chapter 6 reports the experimental results, while Chapter 7 concludes the study. Comprehensive results are provided in the Appendices at the end of the thesis.

Chapter 2

Background Information

2.1 The Newsvendor Problem

The origins of the Newsvendor Problem trace back to the work of economist F.Y Edgeworth in 1888, who applied an early version to optimize cash flow in banking operations [11]. The term "Newsvendor" gained prominence nearly a century later, in 1951, when Morse and Kimball introduced it in their seminal work *Methods of Operations Research* [12]. They presented an example of a newspaper vendor, referred to as a "newsboy," facing uncertain daily demand, where leftover newspapers (inventory) would become worthless the next day. Their main goal was to illustrate the use of the Poisson distribution in modeling stochastic demand.

The modern formulation of the Newsvendor Problem was developed by T.M Whitin in 1955 [13]. He introduced key economic considerations like cost minimization and profit maximization and integrated price effects into the model. In his framework, selling price and stocking quantity are set simultaneously. He also expanded the model to express demand as a probability distribution dependent on price, thereby making price a decision variable. Mills [14, 15], as discussed in [7], further refined this formulation by explicitly modeling the mean demand as a function of the selling price .

To provide a concrete illustration of the classical formulation, the following example demonstrates the classical Newsvendor model using a basic single-period inventory setting. It includes several common analytical perspectives, such as maximizing expected profit, minimizing expected cost, and determining the optimal order quantity using the critical ratio and the inverse of the demand distribution function.

Example 2.1. A newspaper vendor must decide how many newspapers Q to order each morning. Each newspaper is purchased at a wholesale cost of $C_w = \$0.30$ and sells each at a retail price of $p = \$1.00$. The unsold newspapers have no salvage value, i.e., $v = \$0.00$. Demand D is uncertain and considered to follow a normal distribution $N(100, 15^2)$. It is necessary to also define and compute the *marginal profit* $m = p - C_w = \$0.70$, the *marginal loss* $l = C_w - v = \$0.30$, the *underage cost* $C_u = m = \$0.70$ and the *overage cost* $C_o = l = \$0.30$.

The problem can be modelled in different ways, depending on what information the newspaper vendor needs to make managerial decisions.

1. **Expected Profit Maximization Model**

$$\mathbb{E}[\Pi(Q)] = \sum_d \left[\min(d, Q)\, p - Q\, C_w\right] \mathbb{P}(D = d) \tag{2.1}$$

   This formulation computes the average profit over all possible demand values d. The term $\min(d, Q)$ reflects that no more than Q units can be sold, even if demand exceeds supply. The expected value is found by weighting each profit outcome by the probability of the corresponding demand level. The optimal order quantity Q^* maximizes this expected profit.

2. **Expected Cost Minimization Model**

$$\mathbb{E}[C(Q)] = \mathbb{E}\left[C_o(Q - D)^+ + C_u(D - Q)^+\right] \tag{2.2}$$

   Here, the cost is modeled into two parts: overage and underage. The term $(Q - D)^+$ captures unsold inventory (when supply exceeds demand), while $(D - Q)^+$ captures lost sales (when demand exceeds supply). Each is multiplied by its respective unit cost. The optimal order quantity Q^* minimizes the expected total cost across all possible demand outcomes.

3. **Optimal Solution Derived from the Critical Ratio**

$$F(Q^*) = \frac{C_u}{C_u + C_o} = \frac{0.70}{0.70 + 0.30} = 0.70 \tag{2.3}$$

The *critical ratio* represents the optimal service level, which is the probability that demand is less than or equal to the chosen order quantity Q^*. This probability is given by the cumulative distribution function (cdf) F. The ratio balances the underage and overage costs. In this example, the vendor should order enough to satisfy demand for 70% of the time.

4. **Optimal Solution Derived from the Quantile Function**

$$Q^* = F^{-1}(0.70) = \mu + z_{0.70} * \sigma = 100 + 0.524 * 15 \approx 107.86 \tag{2.4}$$

Since demand $D \sim N(100, 15^2)$, the optimal order quantity can be computed directly using the *quantile function*, the inverse cdf. The target service level 0.70 is determined by the critical ratio. The $z_{0.70}$ value is the standard normal score corresponding to the 70-th percentile. Multiplying it by the standard deviation σ, and adding it to the mean μ, gives us the optimal order quantity Q^* that balances underage and overage costs.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is one of the three fundamental areas of machine learning, alongside supervised and unsupervised learning. Unlike supervised learning, which relies on labeled datasets, or unsupervised learning, which discovers patterns in unlabeled data, RL employs an *agent* that learns by interacting with a dynamic *environment*. The agent receives feedback in the form of rewards or penalties, allowing it to learn a *policy*, i.e., a strategy that maximizes its long-term cumulative *reward* [16, 17, 18].

2.2.1 Fundamental Concepts of RL

RL problems are usually modeled as Markov Decision Processes (MDPs), a mathematical framework for decision-making in environments that evolve over time. MDPs provide a structured way to represent the interaction between an agent and its environment by defining a set of *states*, *actions*, *transition probabilities*, and *rewards*. They

also have a remarkable mermoryless feature called the Markov Property, which states that the future state depends only on the current state and action. MDPs are well suited for RL because they can capture the sequential nature of decision-making as well as the uncertainty in how actions influence future outcomes.

Formally, an MDP is defined as a 4-tuple (S, A, T, R), where S is the *state space*, a set of possible states the environment can be in; A is the *action space*, a set of actions the agent can take; $T : S \times A \times S \to [0, 1]$ is the *transition probability function*, where $T(s, a, s')$ denotes the probability of moving to state s' after taking action a in state s.; and $R : S \times A \times S \to \mathbb{R}$ is the reward function, where $R(s, a, s')$ denotes the reward earned when moving to state s' after taking action a in state s. This framework supports both *episodic* and *continuing* tasks. In *episodic* tasks, the *agent* interacts with the environment in *episodes*, which are subsequences of one or more time steps which end once a terminal state is reached. On the other hand, in *continuing* tasks, the agent interacts with the environment indefinitely [16, 17, 18, 19].

A policy defines how the agent behaves within an MDP. A *deterministic* policy $\pi : S \to A$ maps each state to a specific action. A *stochastic* policy $\pi : S \times A \to [0, 1]$ specifies the probability of selecting action a in state s, such that the probabilities over all actions adhere to $\sum_{a \in A} \pi(s, a) = 1$. When a policy is applied to an MDP, it generates a sequence of state transitions and rewards. This sequence, called a *trajectory*, is commonly denoted as $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$. In practice, a policy must also balance *exploration*, selecting actions that uncover new information about the environment, with *exploitation*, choosing actions that are already known to produce high rewards. This tradeoff is often managed by introducing randomness into action selection and gradually reducing it as the agent gains more knowledge [16, 17, 18, 19].

The expected cumulative reward over this trajectory is known as the *return*. In *finite horizon* tasks, the interaction between the agent and the environment lasts for a fixed number of time steps T and the return is:

$$R(\tau) = \sum_{t=0}^{T} r_t \ . \tag{2.5}$$

These tasks are usually episodic and have a clear endpoint. In *infinite-horizon* tasks, the agent-environment interaction may continue indefinitely. The return in this scenario is *discounted* and given by:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \ , \tag{2.6}$$

where $\gamma \in [0, 1)$ is a discount factor. This allows for the rewards to be geometrically distributed with immediate rewards contributing more to the return and being more important than future rewards. This helps in bounding the return and ensures convergence. Finally, a different formulation, utilized in continuing tasks without natural episode terminal point, which is the *average reward* criterion. It is defined as:

$$R(\tau) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} r_t \ . \tag{2.7}$$

The agent here seeks to maximize the long-run average reward per time step and treats all the rewards equally [16, 17, 18, 19].

The goal in RL is to find a policy that maximizes the expected return over all possible trajectories the agent may experience. Since both the environment and the policy can be stochastic, the trajectory is treated as a random variable. Therefore, the probability of a trajectory under a stochastic policy π is given by:

$$P(\tau \mid \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t)\, P(s_{t+1} \mid s_t, a_t) \ , \tag{2.8}$$

where $\rho_0(s_0)$ is the distribution over initial states, $\pi(a_t|s_t)$ is the probability of selecting action a_t in state s_t, and $P(s_{t+1}|s_t, a_t)$ is the probability of transitioning to state s_{t+1}. The *expected return* over all trajectories generated by following policy π is expressed as:

$$J(\pi) = \mathbb{E}_{\pi}[R(\tau)] = \int_{\tau} P(\tau \mid \pi) R(\tau) \ , \tag{2.9}$$

and it captures both the randomness in the environment and the agent's behavior. If the policy is *deterministic*, π assigns probability 1 to a single action, while the trajectory distribution still accounts for stochasticity in the environment. In either scenario, the *optimal* policy is expressed as $\pi^* = \arg\max_{\pi} J(\pi)$.

A policy defines the agent's behavior. RL evaluates the effectiveness of that behavior using *value functions* or *action-value functions*. Most MDP algorithms find optimal policies by learning such functions. A *value function* is used to estimate the expected return starting from a given state s and applying a policy π. It is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\tau \sim \pi(\cdot|s)} \left[R(\tau) \,\middle|\, s_0 = s \right] \ , \tag{2.10}$$

where $\tau \sim \pi(\cdot \mid s)$ denotes the trajectories being sampled given the policy π. An *action-value function* is used to calculate the expected return starting in a given state

s, taking action a and following policy π. This is given by :

$$Q^{\pi}(s,a) = \mathbb{E}_{\tau\sim\pi(\cdot|s)}\left[R(\tau)\,\middle|\, s_0 = s, a_0 = a\right] \ . \tag{2.11}$$

If the policy followed was the optimal policy π^*, then the *optimal value function* and the *optimal action-value function* are expressed as:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \ , \tag{2.12}$$

and:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \ , \tag{2.13}$$

respectively.

These functions are recursive and are most commonly used in their *Bellman* form, which decomposes the value of a state or state-action pair into immediate reward plus the discounted value of successor states. The *on-policy* Bellman expectation equations can be written as:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[\mathop{\mathbb{E}}_{s'\sim T(\cdot|s,a)}\left[\, R(s,a,s') + \gamma V^{\pi}(s')\,\right]\right] \ , \tag{2.14}$$

and:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{s'\sim T(\cdot|s,a)}\left[R(s,a,s') + \gamma \mathop{\mathbb{E}}_{a'\sim\pi(\cdot|s')}\left[Q^{\pi}(s',a')\right]\right] \ . \tag{2.15}$$

These are called *on-policy* because the inner action expectations are taken with respect to the same policy π that generated the data. If $\pi = \pi^*$, then that results in the optimal *on-policy* action-value equation. On the other hand, the optimal *off-policy* Bellman equation for the action-value function is given by:

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s'\sim T(\cdot|s,a)}\left[\, R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\,\right] \ , \tag{2.16}$$

where the inner expectation is replaced by maximization. The policy acts greedily here. Focus is usually placed on action-value functions, also referred to as *Q-functions* because they directly evaluate specific state-action pairs, providing an immediate criterion for action selection, without requiring access to the transition model. In practice, value and Q-functions are computed by several different approaches that include dynamic programming, Monte Carlo estimation, temporal-difference methods, and function approximation [16, 17, 18, 19].

2.2.2 Taxonomy of RL Algorithms

Algorithms operating within an MDP framework can be broadly divided into *model-based* and *model-free* methods. *Model-based* approaches are either given or learn the model, the transition dynamics and reward function of the environment, and utilize it to anticipate future states and rewards, enabling planning. However, the model may be unavailable or prohibitively complex. In contrast, *model-free* approaches do not require knowledge of the model, and they learn through experience acquisition. This makes them easier to implement but potentially costly in real-world settings [17, 18, 19].

Model-free methods are further categorized as either *value-based* and *policy-based*. *Value-based* optimize the Q-function directly, learning the policy $\pi^* \approx \arg max_\pi Q^\pi(s, a)$. These methods are very sample-efficient, have small variance with respect to function estimation and rarely get trapped in local minima. Their disadvantages lie in their difficulty in capturing continuous action spaces and the ϵ-greedy strategy employed in conjunction with the max operator often leading to overestimation. The ϵ-greedy strategy is a method that balances exploration and exploitation by choosing a random action with a probability ϵ (exploration) and a known seemingly good action with a probability $1 - \epsilon$ (exploitation). Contrarily, *policy-based* approaches optimize the policy directly by iteratively updating the policy until the expected cumulative return is maximized. They offer simpler policy parameterization, handle continuous and high dimensional action spaces well and tend to converge more easily [17, 18, 19].

Methods that combine characteristics of *value-based* and *policy-based* algorithms, are called *actor-critic* algorithms. They use the value-based methods to learn a Q-function (or value function) to achieve higher sample efficiency and leverage the policy-based methods to learn the policy function regardless of action space characteristics. The algorithms inherit the corresponding advantages and disadvantages. The model-free family includes more specialized categorizations, which are not mentioned here to maintain focus on the primary broad categories [16, 17, 18].

Finally, it must be noted that while most RL algorithms are MDP-based, there exists a simpler class of methods called *Bandit* algorithms. They do not consider state transitions and utilize repeated action selection and their immediate rewards. They can be considered a special case of an MDP with a single state [16, 17, 18].

Chapter 3

Problem Description

3.1 A Quality-Dependent Newsvendor Problem

A recent extension of the classical Newsvendor Problem was introduced by Özbilge *et al.* [6], namely the Quality Dependent Newsvendor Problem (QDNP). This problem incorporates product perishability and consumer sensitivity to both price and quality, offering a more realistic representation of food retail dynamics. It also integrates the option of donation as a profitable socially and environmentally responsible decision-making opportunity within the model's temporal horizon.

The newsvendor component of the QDNP builds on the framework of Petruzzi and Dada [7], who reviewed and extended the literature on price effects in the newsvendor setting and used it to explore the interplay between operational and marketing considerations in firm-level decision-making. Their analysis includes both single-period and multi-period models, with demand represented using either additive or multiplicative forms.

The real-world problem that is modeled through the aforementioned QDNP is a food retailer's pricing and inventory management problem where the product deteriorates continuously. It is modeled as a two-period single-product newsvendor problem

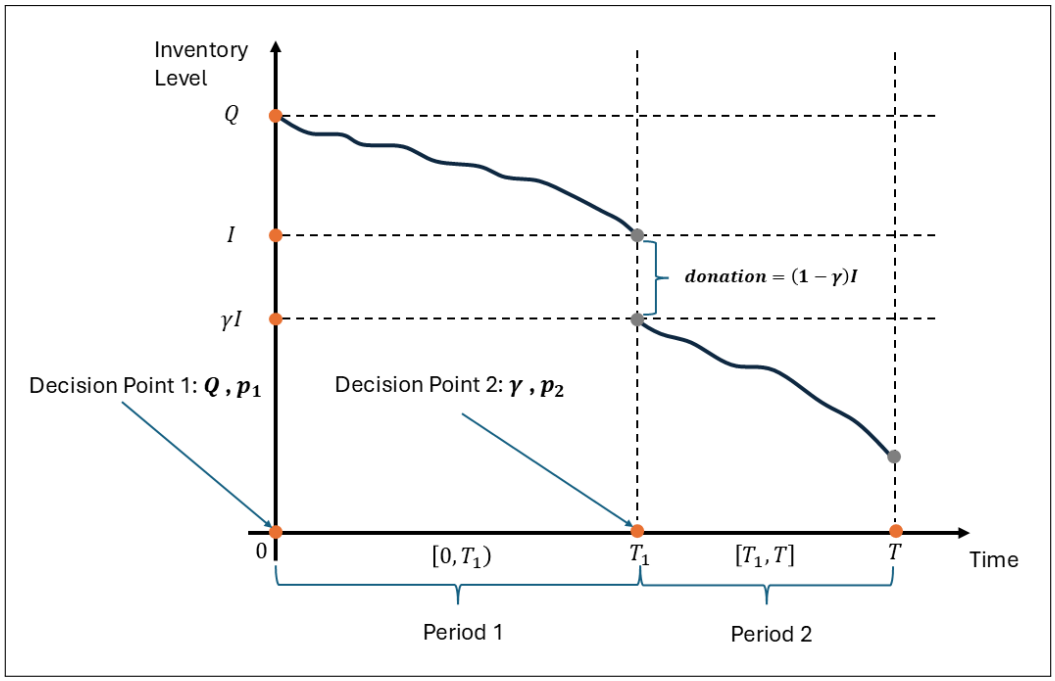


Figure 3.1: QDNP sequence of events

where demand is uncertain and constitutes a function of both price and quality. It must be noted that the inclusion of quality as an integral component of demand is why this variation of the newsvendor problem is deemed "quality dependent".

The selling season is defined as the product's shelf life T. It is broken down into two periods, the beginning of each, marking a distinct decision-making opportunity. The first decision point is time 0, i.e., the beginning of the first period. At that point, the retailer must decide on the stocking quantity Q, as well as the initial selling price p_1. The second decision point is at time point T_1, which can be any time point within the selling horizon where the quality of the product remains high enough to meet the donation recipient's food safety constraints. This point is exogenous and marks the beginning of the second period. Here, the retailer must decide whether and what percentage of the inventory I will be donated, denoted as $(1-\gamma)$, where γ expresses the percentage of inventory retained for sale. Also, the new selling price p_2 for each product unit shall be determined. This price update typically involves a discount, which is a common policy for perishable products in supermarkets aiming at increasing profit. Also it is dependent on the on-hand inventory and the remaining shelf life. Price adjustments upwards of the original price are rare and may occur if inventory is exhausted very early in the selling season [6]. Figure 3.1 depicts the sequence of events for the QDNP through the relationship between inventory level I

and time.

In both periods, demand is considered to be uncertain and it is henceforth expressed through a random component of demand ε, which follows the same distribution in each period. The distribution has a non-decreasing hazard rate, a finite mean μ, and a standard deviation $\sigma > 0$. Its cdf $F(.)$ is twice differentiable, invertible, and defined on an interval $[A, B]$.

The newsvendor model is considered suitable due to fresh food seasonality and quality variation among harvests making it reasonable to consider each batch of product units unique. It captures the challenge of ordering perishable items under uncertain demand and limited selling horizons. However, it does not account for substitution effects or repurchases within the selling horizon. Furthermore, quality and deterioration rates are assumed to be known at all times. This is deemed realistically feasible through the use of reliable IoT technology, particularly time-temperature indicators (TTIs). TTI data can be used to estimate the product shelf life and be input to food science models for deriving quality deterioration rates [6].

Finally, donation is assumed to bring the retailer a reward $R > 0$ per donated product unit. This reflects any manner in which donation is profitable for the retailer: government incentives for waste reduction, including tax deductions for charitable actions, image improvement, and consumer preference due to social and environmental responsibility. It also counteracts the disposal cost $C_d > 0$, which arises because most municipalities in both Europe and North America enforce some form of a pay-as-you-throw (PAYT) policy [6].

These assumptions above provide the basis for the modeling formulation presented in the following section.

3.2 Mathematical Formulation of QDNP

The mathematical formulation of the QDNP, as proposed by Özbilge *et al.* [6], incorporates models for quality loss, demand, and expected profit. This section outlines these elements, with a focus on the modeling components adopted for the purposes of the thesis.

3.2.1 Quality Deterioration

Özbilge *et al.* [6] extended the quality-loss models of Bowman *et al.* [20] and Osvald and Stirn [21]. Fresh food products have peak quality at the moment of harvest or production. After that point, they start to lose nutritional value and their quality decays with time. For example, a fruit ripens and eventually it spoils. These changes are evident to the consumer, who usually accepts ripeness below a certain threshold. This threshold is dependent on the chemical characteristics of each food product. The deterioration rate and hence, the product lifetime, is also affected by changing environmental conditions throughout the supply chain. For example, batches of fruit are kept at optimal conditions at the supplier's storage facility, but their quality decay rates increase when they are loaded onto trucks and transported to a supermarket, where they will also be on display. This is due to exposure to different temperatures as well as inevitable human errors. Unforeseen events may also happen, despite the fact that precautions are taken into consideration when designing the respective supply chain.

It is assumed that at the supplier's facility, the retailer observes an initial product quality level q. The product's effective quality e_t can be estimated from TTI data, since it is not always externally observable. The quality of the product is assumed to drop d units during transportation, depending on factors such as travel time and technologies for maintaining optimal transport conditions, including temperature and humidity, among others. This results in two categories of products: *fresh* products and *aged* products. Fresh products have a quality level at or above the discernibility threshold v, while aged products have a quality level below v, at time t_0 when the product is on the shelf and the selling season begins.

The retailer monitors the quality drop and can decide how to move forward. The maximum quality is considered to be at 100%. If the product units are below a minimum acceptable quality level η, they are considered unsellable because they are too spoiled for consumption or unappealing to customers. Assume that t_0 is the intitial time, and that customers start noticing visible changes in quality at time t_1. Until time $t_1 > t_0$, quality is stable for a time interval $k = t_1 - t_0$. Quality remains acceptable to customers but decays in the time interval from t_1 to t_2. Product quality is unacceptable at any moment $t \geq t_2$. The product shelf life can be expressed as $T = t_2 - t_0$ and depends on the quality level $q - d$ upon arrival at the retailer, on

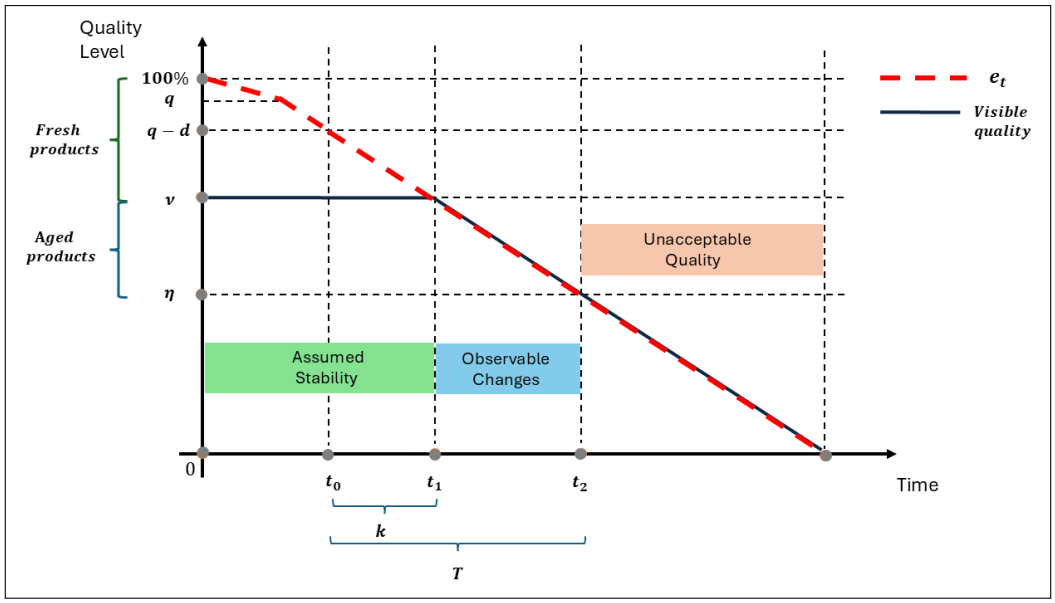


Figure 3.2: Linear quality deterioration in the selling season

the minimum acceptable quality level η and, on the deterioration rate $\lambda > 0$ at the retailer's site.

In the case of *linear* quality deterioration, let $\delta(t) = min\{v, e_t\}$, where:

$$e_t = \begin{cases} q - d - \lambda t \,, & \text{if } t \leqslant T \\ 0 \,, & \text{otherwise.} \end{cases} \tag{3.1}$$

An important assumption here is that customers are not accurate judges of quality, and they perceive it as v for fresh products when $q - d \geq v$, while they start perceiving quality changes when $e_t < v$. Therefore, the modeling is dependent on product type. The perceived linear quality deterioration function for fresh products at a single time point can be expressed as follows:

$$\delta(t) = \begin{cases} \nu \,, & \text{if } t \leqslant \frac{q-d-\nu}{\lambda} \,, \\ q - d - \lambda t \,, & \text{if } t > \frac{q-d-\nu}{\lambda} \,. \end{cases} \tag{3.2}$$

Since the QDNP is considered into two different selling periods, it is necessary to take the average case, which for the first period can be expressed as:

$$\begin{aligned} \bar{\delta}_1(T_1) &= \int_0^{T_1} \delta(t)\, dt \\ &= \frac{1}{T_1} \left[\int_0^{k} \nu \, dt + \int_k^{T_1} (q - d - \lambda t)\, dt \right] \\ &= \frac{1}{T_1} \left\{ k\nu + \left[q - d - \frac{\lambda(T_1 + k)}{2} \right] (T_1 - k) \right\} \,, \end{aligned} \tag{3.3}$$

where $k \equiv \frac{q-d-v}{\lambda}$. It is assumed, without loss of generality, that quality decay becomes apparent during this period because the retailer would not consider repricing unless the quality is discernibly low. For the second period, only when $t > \frac{q-d-\nu}{\lambda}$, the average quality loss is given by the form below:

$$\bar{\delta}_2(T_2) = \frac{1}{T_2}\int_{T_1}^{T}(q-d-\lambda t)\,dt = q-d-\frac{\lambda(2T-T_2)}{2}\ .$$

Figure 3.2 depicts all stages of the linear quality deterioration model within one selling season.

3.2.2 Demand Modeling

Demand is sensitive to both price and quality. Assuming that quality is modeled as a function of time, demand is expressed as a function of price, time, and a random component that represents the uncertainty. Thus, its form for either selling period becomes:

$$D_i(p_i,t,\varepsilon) = y_i(p_i,t)+\varepsilon, \quad i=1,2\ . \tag{3.4}$$

The term $y_i(p_i,t)$ represents the deterministic part of the demand, and it is a linear function decreasing in price p_i and increasing in perceived quality $\delta(t) \geq 0$. It is defined as

$$y_i(p_i,t) = \alpha+\phi\delta(t)-bp_i, \quad i=1,2\ , \tag{3.5}$$

where α, ϕ, and b are constants. The parameter α describes market size, while $\phi > 0$ and $b > 0$ model the customers' sensitivity to quality and price, respectively.

The QDNP model considers two selling periods, namely *Period 1* of length T_1 and *Period 2* of length T_2 while the selling season is $T = T_1 + T_2$. Random demand over the selling season is formulated as:

$$\int_0^{T_1} D_1(p_1,t,\varepsilon)\,dt + \int_{T_1}^{T} D_2(p_2,t,\varepsilon)\,dt = \big[\bar{y}_1(p_1,T_1)+\varepsilon\big]T_1 + \big[\bar{y}_2(p_2,T_2)+\varepsilon\big]T_2\ , \tag{3.6}$$

where:

$$\bar{y}_1(p_1,T_1) = \frac{1}{T_1}\int_0^{T_1} y_1(p_1,t)\,dt = a-bp_1+\phi\,\bar{\delta}_1(T_1)\ , \tag{3.7}$$

and

$$\bar{y}_2(p_2,T_2) = \frac{1}{T_2}\int_{T_1}^{T} y_2(p_2,t)\,dt = a-bp_2+\phi\,\bar{\delta}_2(T_2) \tag{3.8}$$

represent the average deterministic demand over Period 1 and Period 2 respectively. This formulation also enables the definition of an upper bound $\bar{p}_i$ for the price, when solving $\bar{y}_i(p_i,T_i)+A=0$ to find p_i for $i=1,2$.

3.2.3 Objective Function

The main objective of the QDNP, as defined in the original formulation [6], is to maximize the retailer's expected profit over the selling season. In our case, the closed-form, constrained maximization approach is abstracted away in favor of a form that is more suitable for reinforcement learning application. Nvertheless, the objective function is preserved in its original form and continues to guide the optimization process, while its structural properties remain consistent with the original model, retaining all its mathematical properties.

Formally, the objective function can be expressed as follows,

$$V = \mathbb{E}\left[\Pi_1(z_1, p_1)\right] + \alpha\, \mathbb{E}\left[\Pi_2(z_2, p_2)\right] \ , \tag{3.9}$$

where the terms $\mathbb{E}\left[\Pi_1(z_1, p_1)\right]$ and $\mathbb{E}\left[\Pi_2(z_2, p_2)\right]$ describe the first and second period expected profit respectively. The discount rate α is set to 1 but may also take different values. The stocking factors z_1 and z_2 measure the deviation between the supply rate and expected average demand in each period. They play a central role in profit determination by shaping surplus and shortage costs within the QDNP framework, and they are defined as:

$$z_1 = \frac{Q}{T_1} - \bar{y}_1(p_1, T_1) \ , \tag{3.10}$$

and:

$$z_2 = \frac{\gamma I}{T_2} - \bar{y}_2(p_2, T_2) \ , \tag{3.11}$$

where Q is the initial inventory at the beginning of the first period and γI is the portion of inventory retained for sale at the beginning of the second period. Positive values of z_i reflect overstocking as a buffer against uncertainty, while negative values indicate an anticipated shortage. The inventory level for a given random component of demand value is $I = max(0, (z_1 - \varepsilon)T_1)$.

The expected profit in the first period is a newsvendor problem without disposal cost C_d. This is defined as:

$$\mathbb{E}\left[\Pi_1(z_1, p_1)\right] = (p_1 - C_0)\left[\bar{y}_1(p_1, T_1) + \mu\right] T_1 - C_0\, \Lambda(z_1) T_1 - (p_1 + C_s - C_0)\, \Theta(z_1) T_1 \ , \tag{3.12}$$

where C_s and C_0 denote the salvage and purchasing cost per unit respectively. The second period expected profit is a newsvendor problem that considers both disposal cost and donation reward per unit R, and it is formulated as:

$$\mathbb{E}[\Pi_2(z_2, p_2)] = \left[(p_2 - R)\left(\bar{y}_2(p_2, T_2) + \mu\right) - (R + C_d)\, \Lambda(z_2) - (p_2 + C_s - R)\, \Theta(z_2)\right] T_2 + RI. \tag{3.13}$$

The terms $\Lambda(z_i)$ and $\Theta(z_i)$ represent the expected surplus and shortage respectively, playing a principal role in capturing the impact of demand uncertainty within the profit functions. They are defined as:

$$\Lambda(z) = \int_A^z (z-u)\, dF(u) = \int_A^z (z-u) f(u)\, du \;, \tag{3.14}$$

$$\Theta(z) = \int_z^B (u-z)\, dF(u) = \int_z^B (u-z) f(u)\, du \;, \tag{3.15}$$

where $f(.)$ is the corresponding probability density function (pdf) of the random demand component's cdf $F(.)$.

The expected profit functions $\mathbb{E}[\Pi_i(z_i, p_i)]$ are concave with respect to each decision variable, z_i and p_i, as proven in Petruzzi and Dada [7] and Ozblige *et al.* [6] works. The total objective function V inherits this concavity. This means that each profit function can be expressed with regard to only one of the variables by using the first-order conditions to derive the other variable as a function of the chosen one. Thus, the prices p_1 and p_2 are determined in terms of their respective stocking factors z_1 and z_2 as follows:

$$p_1(z_1) = \frac{a + \phi\bar{\delta}_1(T_1) + bC_0 + \mu}{2b} - \frac{\Theta(z_1)}{2b} \;, \tag{3.16}$$

and:

$$p_2(z_2) = \frac{a + \phi\bar{\delta}_2(T_2) + bR + \mu}{2b} - \frac{\Theta(z_2)}{2b} \;. \tag{3.17}$$

These functions can replace p_1 and p_2 in Eqs. (3.7)-(3.11), (3.16) and (3.17). Thus the maximization problem can be expressed as follows:

$$\max_{z_1, z_2} \quad V(z_1, z_2) \;, \tag{3.18}$$

$$\text{s.t.} \quad \max\{A,\ -\overline{y}_2(p_2, T_2)\} \leqslant z_2 \leqslant \frac{I}{T_2} - \overline{y}_2(p_2, T_2) \;, \tag{3.19}$$

where constraints on z_2 are maintained as in the original formulation since the problem is solved with respect to (z_1, z_2), while no constraint is applied on z_1.

3.3 Proposed Non-Parametric Extension of the QDNP Model

The original QDNP framework assumes that the random shock of the demand follows a known, well-behaved probability distribution. This is a distribution with finite mean and variance, a non-decreasing hazard rate, and a twice differentiable and invertible

cdf. However, in many real-world applications, this assumption may not hold. In practice, the true distribution of demand shock is often unknown, and may be difficult to validate.

Instead, firms can collect historical demand shock observations over time, resulting in an empirical dataset. While this dataset can be used to fit a parametric distribution, doing so imposes a specific functional form that may not accurately reflect the underlying behavior of the random variable. The employment of a non parametric approach like Kernel Density Estimation (KDE) can address this limitation while also preserving the smoothness and differentiability properties required by the QDNP model (given a sufficiently smooth kernel), allowing the integration of empirical uncertainty without altering the mathematical structure of the original formulation[22, 23].

KDE is a non-parametric method for estimating the probability density function of a random variable, based on observed data. Instead of assuming a specific distribution form, KDE builds a smooth curve that reflects the shape of the data. It is closely related to a histogram but with additional properties, such as continuity and smoothness, depending on the kernel function used [22, 24, 25].

Let $(x_1, x_2, \ldots, x_n)$ be a sample of independent and identically distributed observations drawn from an unknown continuous distribution with density $f(x)$. The KDE $\hat{f}_h(x)$ of the probability density function is defined as:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) , \tag{3.20}$$

where $K(.)$ is the *kernel function* and $h > 0$ is the *bandwidth* or *smoothing parameter*. The kernel function determines the shape of the local contributions around each data point. The bandwidth regulates how wide each kernel is spread, and controls the smoothness of the estimated density. Furthermore, n is the number of data points and x_i represents a single data point [22, 24, 25].

The selected kernel function employed here is the standard Gaussian kernel, with a zero mean and a standard deviation $\sigma = 1$. Given that it is smooth, symmetrical, and infinitely differentiable, its properties are aligned with the analytical requirements of the original QDNP, which are inherited by the KDE. The kernel is expressed as:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2} . \tag{3.21}$$

It is worth noting that other kernel functions exist (e.g.,Epanechnikov, uniform, triangular, biweight etc) and they all result in consistent estimators under appropriate

circumstances. The final choice of the kernel function depends on theoretical or computational requirements [26, 27].

The bandwidth is crucial in balancing the bias-variance tradeoff in the estimator. An h value that is too small may lead to overfitting (high variability in the estimate), while an h value that is too large may lead to underfitting (low variability in the estimate). In practice, its value is determined using well known heuristics such as Scott's Rule or Silverman's Rule. For the definitions to be compliant with the *Python SciPy* library utilized for implementation in the present thesis, the multivariate version of the rules are considered. Scott's rule is expressed as $h = n^{-1/d+4}$, while Silverman's Rule is defined as $h = \left(\frac{4}{n(d+2)}\right)^{\frac{1}{d+4}}$. The number of data points available is represented by n, while d is the number of dimensions. All data points are considered to have equal contribution to the estimate [28, 29, 30, 31].

The parts of the QDNP model that require the use of the KDE for approximating the pdf of the unknown demand shock are primarily Eqs. (3.14) and (3.15). The approximated expected surplus and shortage are then modeled as follows:

$$\hat{\Lambda}_h(z) = \int_A^z (z-u)\, d\hat{F}_h(u) = \int_A^z (z-u)\hat{f}_h(u)\, du \ , \tag{3.22}$$

$$\hat{\Theta}_h(z) = \int_z^B (u-z)\, d\hat{F}_h(u) = \int_z^B (u-z)\hat{f}_h(u)\, du \ , \tag{3.23}$$

where $\hat{f}_h(u)$ represents the pdf approximation using KDE as expressed in Eq. (3.20), and $\hat{F}_h(u)$ represents the approximation of the cdf using the KDE. The cdf is derived by integrating over the pdf as follows:

$$\hat{F}(x) = \int_{-\infty}^{x} \hat{f}(t)\, dt \tag{3.24}$$

Taken together, the KDE-based definitions and associated specifications presented in this subsection provide a comprehensive non-parametric formulation of the QDNP model, which incorporates empirical demand shock uncertainty while preserving its original analytical structure.

Chapter 4

Reinforcement Learning Framework

4.1 Soft Actor-Critic

Soft Actor-Critic (SAC) is an off-policy, model-free deep reinforcement learning (DRL) algorithm for continuous action spaces. It combines high sample efficiency with stable learning. Its foundational innovation is the augmentation of the standard objective of expected return maximization with an entropy term. This encourages the policy being learned to act as randomly as possible while still achieving high rewards. *Maximum entropy* RL improves exploration, avoids premature convergence to suboptimal strategies, and yields policies that are more robust to model and estimation errors. SAC builds on the general principle of Q-learning, estimating the expected return for each state-action pair and selecting actions that maximize this value, but integrates a stochastic policy and entropy maximization into the actor-critic structure. It is considered to be DRL due to the implementation being based on neural networks [10].

In the original 2018 publication by Haarnoja *et al.* [10] the authors reported that SAC outperforms several strong baseline RL approaches, such as Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO) and Twin Deep Delayed Deterministic Policy Gradient (TD3), on a range of continuous action and state space

benchmark problems, ranging from simple locomotion tasks to high-dimensional humanoid control. In addition, it is very robust when it comes to hyperparameter choices, making it easier to tune in comparison to other RL methods that require meticulous hyperparameter tuning.

The characteristics above point to SAC being a promising and practical RL approach that can be applied to the QDNP, an inherently stochastic problem due to the random component of demand.

4.1.1 Theoretical Foundations

Haarnoja et al [10], establish the modeling foundation of the method as an infinite-horizon MDP (S, A, T, R), where S is the continous state space, A is the continuous action space, $T : S \times S \times A \to [0, \infty)$ is the unkown transition probability density function and $R : S \times A \to [R_{min}, R_{max}]$ is the bounded reward function. $s_t \in S$ represents the current state, $a_t \in A$ represents the current action and $T_\pi(s_t)$ and $T_\pi(s_t, a_t)$ are the state and state-action marginals of the trajectory distribution induced by policy $\pi(a_t|s_t)$ respectively.

The authors further enhance their approach by adopting an extension of the standard RL objective that encourages stochastic policies, referred to as *maximum entropy RL*. Instead of maximizing only the expected cumulative reward, the agent also maximizes the expected entropy of its policy. For an infinite-horizon maximum entropy objective under a stochastic policy, this is given by

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(s_t,a_t)\sim T_\pi} \left[R(s_t, a_t) + \alpha \, \mathcal{H}\big(\pi(\cdot \mid s_t)\big) \right] \tag{4.1}$$

where $\mathcal{H}(\pi(\cdot \mid s_t)) = -\mathbb{E}_{a_t\sim\pi} [\log \pi(a_t \mid s_t)]$ is the Shannon entropy and $\alpha > 0$ is the *temperature parameter* controlling the tradeoff between reward maximization and entropy. For $a \to 0$ it is feasible to obtain the standard RL objective. This formulation encourages wider exploration, avoids premature convergence to suboptimal policies, allows the policy to represent multiple equally good ways of action choice and has also been observed to be faster than others when applied to difficult tasks [10, 32].

As presented in the original publication [10], the SAC algorithm is derived starting from a maximum entropy variation of *policy iteration*. Policy iteration is a method used to find an optimal policy that alternates between two steps, *policy evaluation* and policy *improvement*. The value of the current policy is estimated during the *policy evaluation*

step while the policy is updated greedily, by choosing actions that yield the highest expected reward based on current value estimates, during the *policy improvement* step. The algorithm refines the policy by repeating these steps until convergence to an optimal policy. This is based solely on a tabular setting for theoretical analysis and proof of convergence [10, 32].

Here, this is achieved through the steps of soft policy evaluation and soft policy improvement. In the evaluation step, the Q-value for a fixed policy is estimated via repeated application of the soft Bellman backup operator. In the improvement step, the policy is updated by minimizing the Kullback-Leibler (KL) divergence between the policy distribution and a softmax distribution over Q-values, yielding a policy that is closest to the optimal softmax distribution within the constraints of the chosen policy class (e.g., Gaussian policies). [10, 32, 33].

4.1.2 The Soft Actor-Critic Algorithm

Haarnoja *et al.* [10] expanded their formulation to one based on function approximators, with the purpose of handling large continuous state and action domains. They considered a value function $V_\psi(s_t)$, a soft Q-function $Q_\theta(s_t, a_t)$ and a policy $\pi_\phi(a_t|s_t)$, where ψ, θ and ϕ are neural network parameters. The policy must be tractable, for example a Gaussian with its mean and covariance produced by a neural network.

There are three error functions that are used within the algorithm. The first one is related to the value function and given by the squared residual error:

$$J_V(\psi) = \mathbb{E}_{s_t \sim D}\left[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}\left[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t \mid s_t)\right]\right)^2\right] , \tag{4.2}$$

which is used to train the value network. This network predicts how good a state is under the current policy by minimizing the regression loss between what the value network predicts and what is estimated through Q-values and the policy. The states s_t are sampled from the replay memory D, which contains states the agent has encountered and stored for future exploitation. The corresponding gradient is given as:

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)\left(V_\psi(s_t) - \left(Q_\theta(s_t, a_t) - \log \pi_\phi(a_t \mid s_t)\right)\right) , \tag{4.3}$$

and provides a step-by-step update rule for improving the value network parameters.

The second error function is the Q-function loss, given by

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim D}\left[\frac{1}{2}\left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t)\right)^2\right] , \tag{4.4}$$

Algorithm 4.1 Soft Actor-Critic (SAC)

Initialize parameter vectors ϕ (policy), θ_1, θ_2 (Q-functions), and ψ (value function)
Initialize target parameters $\bar{\psi} \leftarrow \psi$
Initialize replay buffer $\mathcal{D}$
for each iteration **do**
    for each environment step **do**
        Sample action $a_t \sim \pi_\phi(\cdot \mid s_t)$
        Observe $s_{t+1} \sim p(\cdot \mid s_t, a_t)$ and reward $R(s_t, a_t)$
        Store new experience $(s_t, a_t, R(s_t, a_t), s_{t+1})$ in $\mathcal{D}$
    end for
    for each gradient step **do**
        Choose $min(Q(\theta_1), Q(\theta_2))$ for $\hat{\nabla}_\psi J_V(\psi)$
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$ ▷ Value network update
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i) \quad \text{for } i = 1, 2$ ▷ Q-network updates
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Policy network update
        $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$ ▷ Target value update
    end for
end for

which trains the Q-network to predict a "target" value derived by taking the immediate reward from the current state and adding the discounted predicted value of the next state as follows

$$\hat{Q}(s_t, a_t) = R(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim T} \left[V_{\bar{\psi}}(s_{t+1}) \right] \ . \tag{4.5}$$

This target value is produced by leveraging a separate target value network $V_{\bar{\psi}}$, with parameters $\bar{\psi}$. This network is slowly updated, aiming to achieve more stable training. The Q-loss gradient is given as follows:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(s_t, a_t) \left(Q_\theta(s_t, a_t) - \left(R(s_t, a_t) + \gamma \, V_{\bar{\psi}}(s_{t+1}) \right) \right) \ . \tag{4.6}$$

Finally, The third and final error is related to the KL divergence, and it is instrumental in policy improvement. Its initial form is:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D} \left[D_{\mathrm{KL}} \left(\pi_\phi(\cdot \mid s_t) \, \middle\| \, \frac{\exp(Q_\theta(s_t, \cdot))}{Z_\theta(s_t)} \right) \right] \ . \tag{4.7}$$

To minimize $J_\pi(\phi)$, the policy is reparameterized using a neural network to produce the action a_t as $a_t = f_\phi(\epsilon_t; s_t)$ where ϵ_t is a noise vector sampled from a stationary

distribution, such as an isotropic Gaussian. This enables the policy objective to be rewritten and expressed by the error:

$$J_\pi(\phi) \;=\; \mathbb{E}_{s_t\sim D,\,\epsilon_t\sim\mathcal{N}(0,I)}\Big[\log\pi_\phi\big(f_\phi(\epsilon_t;s_t)\mid s_t\big) \;-\; Q_\theta\big(s_t,f_\phi(\epsilon_t;s_t)\big)\Big]\,, \tag{4.8}$$

which expresses that the policy prefers actions that have both high Q-values and a low log-probability penalty, thus maintaining sufficient randomness in the process. The derivation of Eq. (4.8) results in the gradient

$$\hat{\nabla}_\phi J_\pi(\phi) \;=\; \nabla_\phi\log\pi_\phi\big(a_t\mid s_t\big) + \Big(\nabla_{a_t}\log\pi_\phi\big(a_t\mid s_t\big) - \nabla_{a_t}Q_\theta(s_t,a_t)\Big)\,\nabla_\phi f_{a_t} \tag{4.9}$$

where $a_t = f_\phi(\epsilon_t; s_t)$. The gradient comprises two parts. The first part, namely $\nabla_\phi \log \pi_\phi\big(a_t \mid s_t\big)$, corresponds to direct changes in the log-probability of actions. The second part describes changes in the policy's output affected by both the log-probability and the Q-values of the actions.

It must be highlighted that, in the derivation, the temperature parameter α is omitted from the formulas by absorbing it into the reward. Specifically, the reward term in the theoretical expressions, denoted here as $R(s_t, a_t)$, corresponds to the scaled reward $\alpha^{-1}r(s_t, a_t)$, where $r(s_t, a_t)$ is the actual reward produced by the environment.

The pseudocode of the SAC algorithm is presented in Algorithm 4.1. Aiming to alleviate positive bias and the consequent degradation of performance in value based methods, as well as speed up the training process, the authors in [10] chose to employ two Q-function networks with parameters θ_i and error functions $J_Q(\theta_i)$, for $i = 1, 2$. These two networks are trained independently. The minimum of the two Q-functions is used in the value network update in step 11 of the pseudocode, as described by Eq. (4.3). After initializing all parameter vectors and the replay memory (lines 1-3 of the pseudocode), the agent iteratively collects experiences from the environment using the current policy in a single step (lines 5-9), and then, for one or multiple gradient steps, it updates all neural networks via stochastic gradient descent using batches sampled from the replay memory (lines 10-16).

4.2 Proposed SAC Environment for QDNP

To apply SAC to the QDNP, it is necessary to formalize the problem as an RL environment. This process involves defining the state space, which captures the relevant

information available to the agent at each decision point, and is normalized to improve learning stability; the action space, which specifies the set of possible decisions; and the reward function, which provides feedback on the quality of those decisions. It also requires establishing the episode structure, which determines how the decision-making process unfolds over time and when an episode begins and ends.

The state space must contain the decision variables and other values that may be useful to the agent. Here, the decision variables are z_1 and z_2. The state vector s_i, however, also includes the current step index i, which indicates in which time step within the episode the vector was produced, the random component of demand ε, and the expected inventory level I at the end of the first period. The latter two are crucial for allowing the agent to observe information related to the problem's inherent uncertainty at each time step. Furthermore, z_2 must satisfy the bounds defined in the maximization problem of Eq. (3.19) in Chapter 3. These bounds ensure feasible solutions. In total, the state vector is five-dimensional and can be expressed as

$$s_i = \begin{bmatrix} i, & I, & \varepsilon, & z_1, & z_2 \end{bmatrix}^\mathsf{T} \in \mathbb{R}^5 \, . \tag{4.10}$$

When employing neural network-based approaches, it is standard practice to rescale input vectors in order to stabilize training. Accordingly, the state vector s_i is rescaled to lie within $[-1, 1]$ at the end of each time step. All variables except for the discrete time index i are standardized using the transformation: $\hat{x} = \frac{x-\mu}{s}$. The mean μ is either derived from the known distributional properties of the variable (or from its feasible bounds), or estimated empirically from data. The scale s is chosen as half of the known feasible range of each state variable, ensuring that the transformed values lie within $[-1, 1]$. This procedure ensures that the continuous features are centered and scaled, yielding a symmetric and robust representation in which no single variable dominates during policy or value network updates. The time index i is included directly without rescaling, since it is discrete and confined to a small finite range [34, 35].

The action space defines the set of possible decisions available to the agent at each time step. In this application, actions correspond directly to assigning values to the decision variables z_1 and z_2, which determine the stocking factors in the first and second period, respectively. At each step, the policy outputs an action vector a_i containing the proposed values for these decision variables:

$$a_i = \begin{bmatrix} z_1, & z_2 \end{bmatrix}^\mathsf{T} \in \mathbb{R}^2 \, . \tag{4.11}$$

Algorithm 4.2 Episode Structure for the QDNP Environment

Initialize step index $i \leftarrow 0$
Initialize state vector $s_i \in \mathbb{R}^5$ ▷ Eq. (4.10)
Initialize action bounds for a_i ▷ Eq. (4.11)
Initialize episode reward $V \leftarrow 0$
for $i = 0$ to 1 **do**
    if $i = 0$ **then** ▷ Step 0: First period
        Apply action a_0 to z_1: $z_1 \leftarrow a_0$
        Sample random demand component ε
        Estimate inventory level $I = max(0, (z_1 - \varepsilon)\, T_1)$
        Set reward for first step $R_0 \leftarrow \mathbb{E}[\Pi_1(z_1)]$
        Update state vector s_0 with current step information
        Rescale state vector s_0 to $[-1, 1]$
        done = False ▷ Non-terminal step
        $i \leftarrow 1$ ▷ Prepare for next step
        return (s_0, R_0, done)
    else ▷ Step 1: Second period
        Apply action a_1 to z_2: $z_2 \leftarrow a_1$
        Clip z_2 to feasible bounds ▷ Eq. (3.19)
        Set reward for second step $R_1 \leftarrow \mathbb{E}[\Pi_2(z_2)]$
        Update state vector s_1 with current step information
        Rescale state vector s_1 to $[-1, 1]$
        Compute total episode reward $V \leftarrow R_0 + R_1$
        done = True ▷ Terminal step
        return (s_1, V, done)
    end if
end for

For implementation purposes, these actions are restricted within the bounds $[A, B]$ to ensure feasibility as well as maintain stable learning by preventing extreme exploration attempts. To complete the environment setup before describing the episode structure, the reward function is the expected profit function given in Eq. (3.9) in Chapter 3.

Finally, the episode structure is based on the concept of backwards induction,

which is used in the analytical solution of the QDNP, as in [6]. In the analytical solution, the second period problem is solved first under uncertainty, before moving forward to the first period problem. Here, the opposite is utilized, resulting in an episodic structure consisting of two time steps.

The episodic procedure is described in Algorithm 4.2. Lines 1-4 of the pseudocode initialize the episode step index i, the state vector s_i, the action bounds for a_i, and the episode reward R, which is set to zero at the start. The step index i indicates the current episode step and serves as a flag to ensure the second step follows the first. The first episode step (lines 6-15) applies action a_0 to z_1, samples the random demand component ε, and uses these values to estimate the inventory level I. The expected profit for the first period, $\mathbb{E}[\Pi_1(z_1)]$, is computed and assigned as the step reward R_0. The state vector is then updated with the information from this step and rescaled to $[-1, 1]$. The step is marked as non-terminal by setting $done = False$, the step index is updated to $i = 1$, and the state vector, reward, and termination flag are returned. This gives the agent intermediate information on the key variables associated with uncertainty, ε and I, which, when combined with the reward signal, enable policy refinement within the episode rather than than only at its conclusion.

The second episode step is described in lines 16-24. Action a_1 is applied to z_2 and clipped to its feasible bounds according to the constraints in Eq. (3.19). The expected profit for the second period, $\mathbb{E}[\Pi_2(z_2)]$, is computed and assigned as the step reward R_1. The state vector is updated and rescaled, and the total episode reward is calculated as $V = R_0 + R_1$. The termination flag is then set to $done = True$, marking the step as terminal and signaling to SAC that the episode has ended. The updated state vector, total reward, and termination flag are returned.

This two-step procedure is repeated from initialization to termination for each episode. During training, the observed rewards are used to update the neural network weights and improve SAC's learned policy. During evaluation, the policy acts according to the mean of the action distribution, producing episodes that reflect the distribution of possible rewards under the uncertainty of ε.

Chapter 5

Experimental Methodology

5.1 Test Data

The experimental evaluation employed the data used in the original study of Ozbilge *et al*. [6], which in turn is based on the case study of Wang and Li [36]. The dataset consists of parameters for four supermarket branches, with product quality modeled as linearly degrading over time. The parameters and their values are reported in Table 5.1 and correspond to those presented in Chapter 3, with Stores 1-4 denoting unique problem instances.

The stochastic component of demand ε follows a uniform distribution with zero mean, and it is bounded in $[A, B] = [-2, 2]$. The experiments were restricted to the linear quality deterioration case across the four problem instances. In terms of product type, the linear case corresponds to products such as vegetables.

As mentioned in Chapter 4, it must also be noted that, an additional variation of the model was considered for scenarios in which, the distribution of the random demand component is unknown. In this setting, the model estimates the distribution empirically from data samples by computing descriptive statistics such as the mean

Store	a	b	ϕ	q	d	v	h	λ/hr	T_1 (hrs)	T (hrs)	C_0	C_d	C_s	R
1	7.92	4.86	4.86	0.90	0	0.8	0.2	0.009	48	78	1	0.05	0.05	0.8
2	6.73	4.13	4.13	0.92	0	0.8	0.2	0.009	48	80	1	0.05	0.05	0.8
3	10.30	6.32	6.32	0.83	0	0.8	0.2	0.009	48	70	1	0.05	0.05	0.8
4	6.34	3.89	3.89	0.85	0	0.8	0.2	0.009	48	72	1	0.05	0.05	0.8

Table 5.1: Parameters per problem instance

Hyperparameter	**Value**
Discount factor (γ)	0.99
Replay memory size	10^6
Number of hidden layers	2
Number of nodes per layer	256
Batch size	256
Time steps	46000
Target update interval	1

Table 5.2: Fixed hyperparameter values

Hyperparameter	**Values**		
Learning rate	3×10^{-4}	3×10^{-6}	-
Gradient steps	1	4	-
τ	5×10^{-3}	5×10^{-3}	1

Table 5.3: Tested hyperparameter values

and standard deviation, and approximating the probability density and cumulative distribution functions using KDE.

For testing purposes, the empirical dataset is generated from a uniform distribution with the same bounds $[A, B]$ as above, but the implementation treats it as if it were drawn from an unknown distribution with 100000 samples. In this case, the clipping bounds are set to the data-driven range $[A, B] = [\min(Data), \max(Data)]$, where $Data$ refers to the set of samples of the simulated unknown distribution.

5.2 Tuning and Evaluating SAC

5.2.1 Hyperparameter Settings

Two applications of SAC were examined. The first one refers to an environment that assumes known characteristics of the random demand component distribution, while the second one refers to an environment where these characteristics are estimated from empirically collected data. A model name is omitted in the first case, as it refers to the standard QDNP. In contrast, the second case employs kernel density estimation and is referred to as KDE-QDNP to differentiate it from the baseline. These cases can effectively asses SAC performance on the two QDNP modeling approaches. The same

procedure is followed for hyperparameter tuning in both cases.

Hyperparameter values were initially set according to those reported in [10], for tasks of varying difficulty. To further refine the implementation for the considered problem instances, preliminary exploratory runs were conducted during the implementation phase. Observations from these runs were used to calibrate the hyperparameters toward ranges that produced responsive learning behavior. The *Stable-Baselines3 Python* library implementation was used for SAC, while the *Gymnasium Python* library was used for setting up the problem environment.

The hyperparameters that assumed fixed values for all experiments are listed in Table 5.2. The *discount factor* γ controls the importance of the target Q-value in Eq. (4.5). The *replay memory size* determines how much memory is available to store the agent's experiences. The *number of hidden layers* and *number of nodes per layer* define the neural network architecture. The *time steps* correspond to the number of individual iterations the agent is trained for. When translated into episodes of two time steps each, the agent is trained for $\frac{46000}{2} = 23000$ episodes. Finally, the *target update interval* specifies how frequently the target networks are updated after each gradient step.

The hyperparameters that were actively tuned are listed in Table 5.3. The *learning rate* determines the step size used by the optimizer to adjust network weights. The *gradient steps* indicate the number of gradient updates performed per time step. Lastly, the *target smoothing coefficient* τ controls the proportion of the current network parameters mixed with the previous target network parameters during updates.

The optimizer and activation function were chosen as in [10], namely Adam and ReLU, respectively. While SAC is generally robust to its hyperparameters, it is sensitive to the reward scale, i.e., the entropy coefficient α. In the present setting, instead of a fixed reward scale, the automatic reward scale update feature available in the *Stable-Baselines3* implementation was used. This feature optimizes an entropy loss function to balance exploration and exploitation, with lower values favoring exploitation and higher values favoring exploration [37]. It should also be noted that all other hyperparameters not explicitly listed here retained their default values in *Stable-Baselines3*, as they do not significantly influence the algorithm's behavior.

5.2.2 Evaluating SAC

The QDNP's objective is to find decision variable values that lead to the maximization

Algorithm 5.1 Policy Evaluation Procedure

Load trained SAC policy
Set policy to deterministic mode
for $i = 1$ to 100 **do** ▷ evaluation rounds
    for $j = 1$ to 1000 **do** ▷ samples (rollouts)
        for $t = 1$ to 2 **do** ▷ steps in episode
            Generate action $a_{j,t} = \mu_\phi(s_{j,t})$
            Observe reward $r_{j,t}$ and current state $s_{j,t}$
        end for
        Collect total episode reward $R_j = r_{j,1} + r_{j,2}$
    end for
    Compute mean total reward of round i: $R_i = \frac{1}{1000} \sum_{j=1}^{1000} R_j$
end for
Compute overall mean of the 100 round means: $R_{total} = \frac{1}{100} \sum_{i=1}^{100} R_i$

of the expected profit. Following Haarnoja *et al.* [10], the trained SAC policy was set to deterministic mode for evaluation, where the mean of the Gaussian action distribution is chosen. To obtain a robust estimate of performance, the policy was then evaluated over 100 evaluation rounds, each consisting of 1000 samples. The mean return was computed for each round, and the overall evaluation result was reported as the average across rounds. This approach provides a statistically stable estimate of the policy's expected return and was also employed for the variables z_1, z_2, p_1, p_2, Q, and γ. The procedure is presented in the pseudocode in Algorithm 5.1.

Given that each data point corresponds to the mean return of 1000 rollouts, it was expected to approximately follow a Gaussian distribution by virtue of the Central Limit Theorem. This assumption was further confirmed empirically using the Shapiro-Wilk normality test. Pairwise comparisons between hyperparameterizations were then conducted using the Student's *t*-test to identify statistically significant differences. Additional insights were obtained through descriptive statistics and boxplots.

It should be noted that the variance within each rollout reflects the stochasticity of the policy distribution, and it does not carry comparative meaning across hyperparameterizations. Instead, statistical comparisons rely on the variability of the mean returns across repeated rollouts, which captures the stability and effectiveness of each configuration. Likewise, differences in total training and evaluation runtime

Algorithm 5.2 Monte Carlo BFGS Maximization Procedure

Fix random seed for reproducibility
for $i = 1$ to 10000 **do** ▷ Monte Carlo iterations
    Sample random demand component $\varepsilon \sim U(A, B)$
    Initialize candidate solution $x_0 = (1.0, 1.0)$
    Apply BFGS to maximize objective function $V^*(x_0; \varepsilon)$
    while BFGS does not converge **do**
        Reinitialize x_0 with random values in $[A, B]$
        Retry BFGS
    end while
    Record optimized objective value V^*
    Compute and store decision variables z_1, z_2, $p_1(z_1)$, $p_2(z_2)$, Q, γ
end for

are considered at the level of hyperparameterizations rather than individual rollouts.

All of the insights above are examined in conjunction with each hyperparameterization's stability and output smoothness during training. These properties are assessed via plots of the 100-point moving average of the total return and the evolution of the entropy coefficient throughout the duration of training.

The aforementioned procedures are applied individually for each problem instance, comparing the different hyperparameterizations and concluding on which ones perform best.

5.2.3 Comparison Against Analytical and Monte Carlo Benchmarks

After identifying the best-performing hyperparameterization(s) of SAC, the results, comprising the overall mean total reward, the resulting means of the z_1 and z_2 decision variables, the corresponding derivative prices p_1 and p_2, the initial inventory Q, and the inventory retention percentage γ are compared against those reported in [6], as well as against results obtained through the application of a Monte Carlo Broyden-Fletcher-Goldfarb-Shanno (MC-BFGS) method to the analytical model described therein.

In this the MC-BFGS case, the random demand component ε is sampled 10000 times, with BFGS applied separately for each realization. This procedure yields a Monte Carlo approximation of the expected profit over the distribution of ε, providing aggregate results that can be directly compared with those of SAC. The imple-

mentation is summarized in Algorithm 5.2. It should be noted that while the point $(1, 1)$ was used as a consistent starting point for optimization, having been found to perform reliably across all iterations, a fallback strategy for random restarts was also included to handle potential convergence failures. However, further refinement of this procedure was deemed outside the scope of the present thesis, as the specific solution methodology was intended solely to serve as an alternative computational benchmark. The *SciPy Python* library implementation was used for the BFGS algorithm, which is based on the typical algorithm due to Nocedal and Wright [38].

5.2.4 Transfer Capabilities of SAC on QDNP

To assess the generalization and transferability capabilities of a trained SAC agent on the standard QDNP model, a dedicated evaluation procedure was designed. Specifically, the agent was trained on the Store 1 problem instance and then directly applied to perturbed versions of the same instance without any further retraining. The perturbations were generated by modifying a single problem parameter by $\pm 10\%$, $\pm 20\%$, and $\pm 30\%$ of its original value, while keeping all the other parameters fixed.

The performance of the transferred policy was compared against a ground-truth baseline, achieved by retraining SAC from scratch under the perturbed parameter setting. The parameters under investigation belong to two groups. The first group is related to demand modeling and includes a, b, and ϕ. The second group refers to the costs and donation reward and includes C_0, C_d, C_s, and R. Therefore, a systematic examination is made across the two most crucial parameter groups of the model, with regard to how robustly a learned policy can generalize beyond its training environment.

From a reinforcement learning formality perspective, following Lazaric's framework on knowledge transfer in reinforcement learning [39], this setup can be formally described as a transfer problem between MDPs with a shared state and action space $S \times A$. The Store 1 problem instance defines the source task $M_{source} = (S, A, T_{source}, R_{source})$, and each perturbed instance defines a corresponding target task $M_{target} = (S, A, T_{target}, R_{target})$. Since the perturbations affect only the transition and/or reward parameters while preserving the state and action spaces, this constitutes a case of transfer with fixed $S \times A$ [39]. The knowledge transferred corresponds to the policy π_{source} learned by SAC on the source task, which is then applied to the target tasks

System	CPU Model	Threads/Core	Cores/Socket	RAM (GB)	Problem Instance
PC1	Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz	2	4	8	1
PC2	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz	2	4	8	2
PC3	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz	2	4	8	3
PC4	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz	2	4	8	4

Table 5.4: System characteristics and corresponding problem instances

without additional training. This form of transfer is commonly referred to as *zero-shot policy transfer* [40]. The evaluation compares the expected return $V_{M_{target}}^{\pi_{source}}$ of the transferred policy against the optimal expected return $V_{M_{target}}^{\pi^*_{target}}$ obtained by retraining SAC on each target MDP, thereby quantifying any performance degradation under structured environment perturbations.

5.3 Implementation Details

All experiments were implemented in Python 3.10.12, primarily relying on the *Stable-Baselines3* library for reinforcement learning and *PyTorch* as the underlying deep learning framework. Environment modeling was supported through *Gymnasium*. Supporting libraries included *NumPy* and *SciPy* for numerical computations, statistics and optimization routines, *Pandas* for data handling, and *Matplotlib* for visualization.

Each problem instance was executed on a designated machine to ensure consistency and to avoid confounding effects from varying system performance. The system specifications per problem instance are reported in Table 5.4. This setup was maintained for both applications of SAC examined. The MC-BFGS and all result data processing were implemented via Jupyter Notebook in Python 3.10.12, on an 8 GB system with an *Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz* CPU model, with 4 Cores and 8 threads. All random generators are initialized with a seed value of 190.

Chapter 6

Experimental Results

6.1 SAC Tuning Results

The tuning of SAC required a careful evaluation framework as its performance depends not only on the final returns but also on the stability of learning. The twelve tested hyperparameterizations, labeled H1-H12, are shown in Table 6.1. The evaluation of each candidate considered several aspects, including the learning dynamics observed during training and the descriptive statistics that were computed from the evaluation of the final learned policy. In addition, boxplots and statistical hypothesis tests were employed to identify significant pairwise differences between hyperparameterizations.

To determine the most suitable hyperparameterization for each problem instance, learning curves were inspected through the 100-point moving average of the total reward. Preference was given to runs in which the average return reached a stable plateau within the computational budget. The behavior of the entropy coefficient was also monitored, since it provides insight into the balance between exploration and exploitation. Higher values, close to 1, indicate exploratory action choices, while values below 0.5 correspond to increasingly exploitative policy behavior. Ideally, the

Label	Learning Rate	τ	Gradient Steps
H1	0.0003	0.005	1
H2	0.0003	0.005	4
H3	0.0003	0.05	1
H4	0.0003	0.05	4
H5	0.0003	1.0	1
H6	0.0003	1.0	4
H7	3e-06	0.005	1
H8	3e-06	0.005	4
H9	3e-06	0.05	1
H10	3e-06	0.05	4
H11	3e-06	1.0	1
H12	3e-06	1.0	4

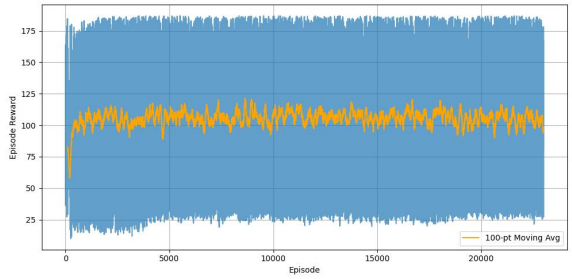
Table 6.1: Legend for hyperparameterization labels

	SAC				SAC (KDE-QDNP)			
Problem Instance	**Store 1**	**Store 2**	**Store 3**	**Store 4**	**Store 1**	**Store 2**	**Store 3**	**Store 4**
Smooth (H)	1,2,3,5,6	1,3,5,6	1,3,5,6	1,3,4,5	1,2,3,5,6	1,2,3,5,6	1,3,5,6	1,3,4,5,6
Top 5 (based on Mean) (H)	7,11,8,5,6	3,10,1,9,2	9,3,4,7,10	9,3,7,5,10	5,8,11,4,6	10,2,9,7,3	3,4,10,7,8	7,3,8,12,2
Smooth & Top 5 (H)	5,6	3,1	3	3,5	5,6	2,3	3,5	3
Best (H)	5	3	3	3	5	3	3	3

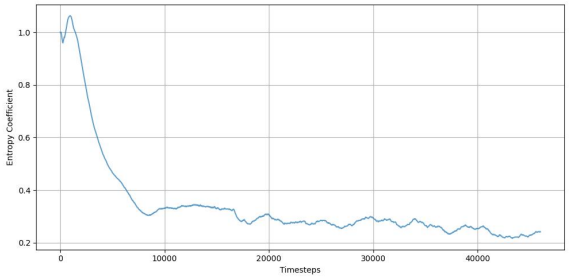
Table 6.2: Best hyperparameterizations per problem instance, for SAC and SAC (KDE-QDNP)

entropy coefficient should decrease smoothly into the exploitation regime without large oscillations. If oscillations occur, they should remain within stable bands such as between 0.2 and 0.4, without abrupt jumps or drops. This pattern indicates that the agent is transitioning consistently from exploration to exploitation and has therefore learned a sufficiently stable policy.

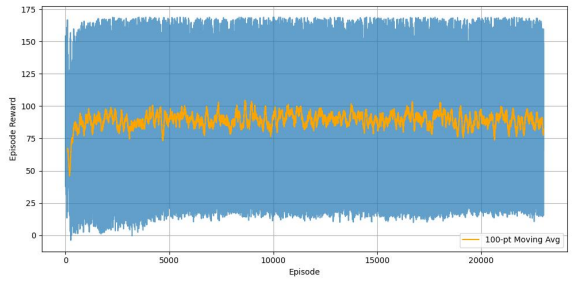
The results of the comparison are summarized in Table 6.2. For each problem instance, candidates with the smoothest entropy coefficient trajectories were identified from Figures A.1-A.16 in Appendix A for the standard SAC implementation and Figures B.1-B.16 in Appendix B for the SAC application on the KDE enhanced QDNP. These figures also display the 100-point moving average of the total reward per episode, where the hyperparameterizations with smooth entropy dynamics typically coincide with runs that reach and maintain a stable plateau over most of the training



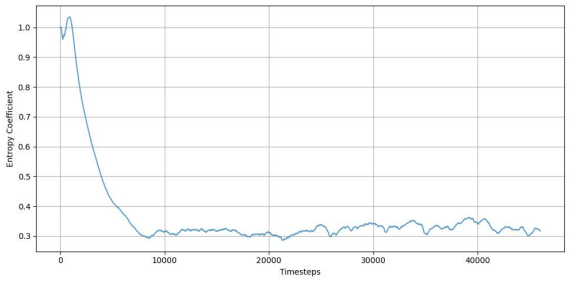
(a) Reward progress per episode: Store 1 - H5



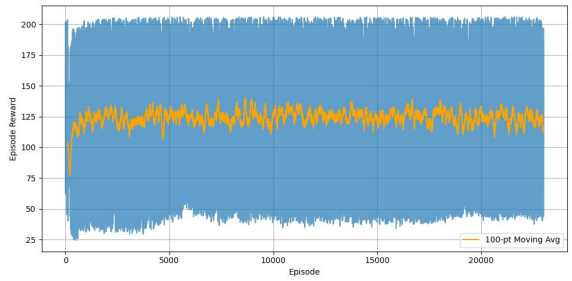
(b) Entropy coefficient per timestep: Store 1 - H5



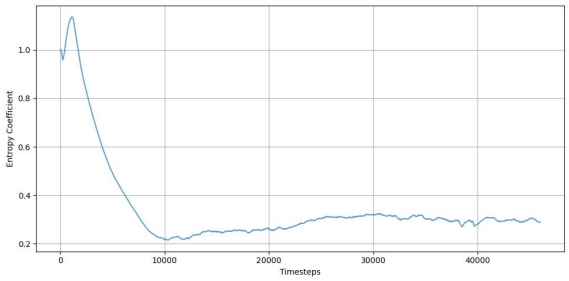
(c) Reward progress per episode: Store 2 - H3



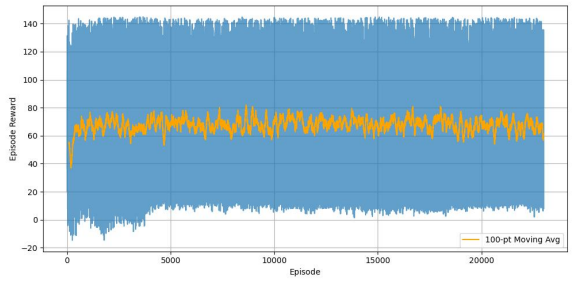
(d) Entropy coefficient per timestep: Store 2 - H3



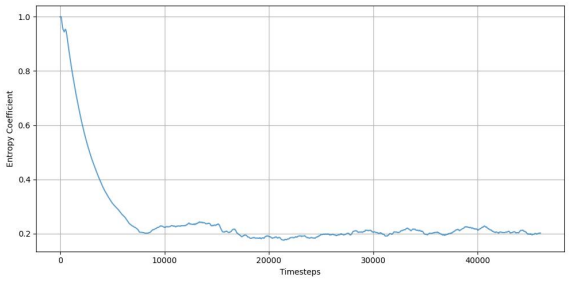
(e) Reward progress per episode: Store 3 - H3



(f) Entropy coefficient per timestep: Store 3 - H3

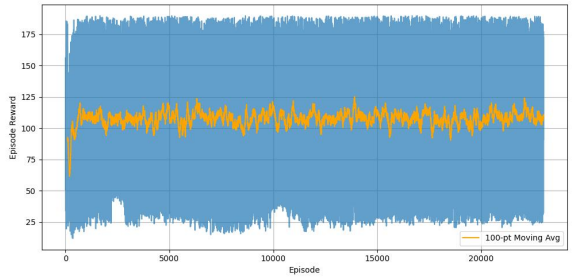


(g) Reward progress per episode: Store 4 - H3

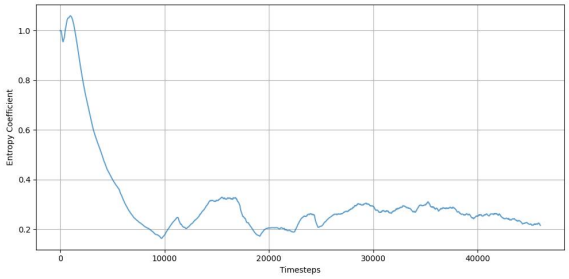


(h) Entropy coefficient per timestep: Store 4 - H3

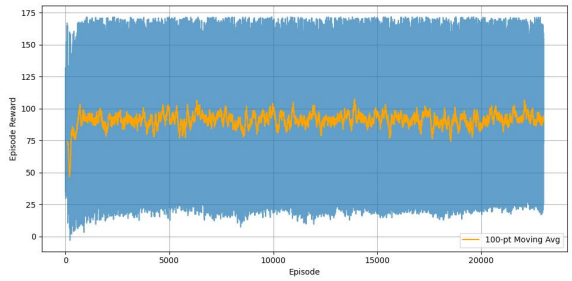
Figure 6.1: Tuning progress of best hyperparameterizations, per problem instance (SAC)



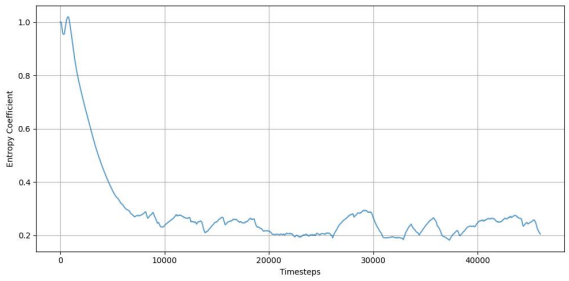
(a) Reward progress per episode: Store 1 - H5



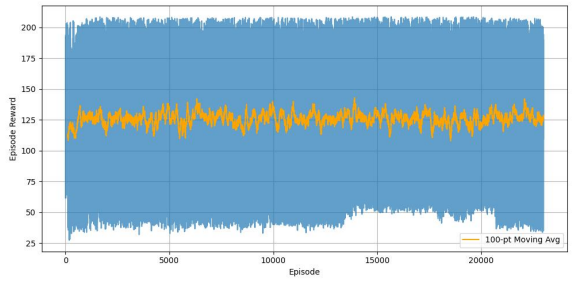
(b) Entropy coefficient per timestep: Store 1 - H5



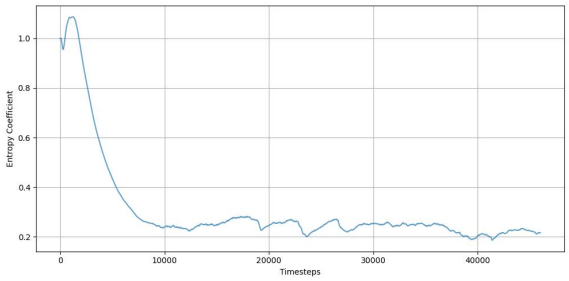
(c) Reward progress per episode: Store 2 - H3



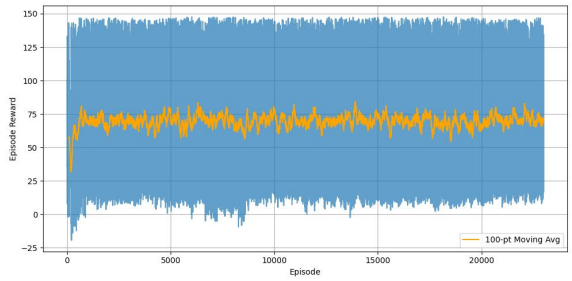
(d) Entropy coefficient per timestep: Store 2 - H3



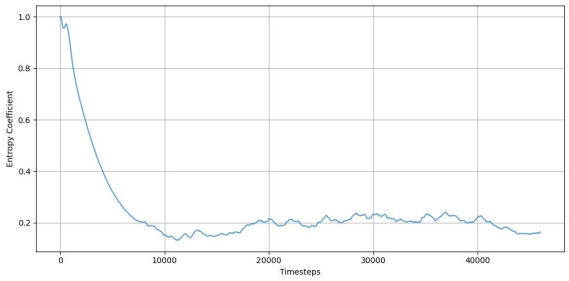
(e) Reward progress per episode: Store 3 - H3



(f) Entropy coefficient per timestep: Store 3 - H3



(g) Reward progress per episode: Store 4 - H3



(h) Entropy coefficient per timestep: Store 4 - H3

Figure 6.2: Tuning progress of best hyperparameterizations, per problem instance (SAC (KDE-QDNP))

Problem Instance	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Store 1	9	8	9	8	7	7	7	7	9	8	7	8
Store 2	6	6	6	10	10	9	6	9	6	6	9	11
Store 3	9	9	7	7	7	7	7	7	7	3	7	9
Store 4	10	5	5	10	5	11	5	10	5	5	11	4

Table 6.3: Significance sums per hyperparameterization (SAC)

Problem Instance	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Store 1	9	8	9	8	8	8	11	8	9	8	8	8
Store 2	6	6	6	11	11	9	6	9	6	6	9	11
Store 3	9	9	8	4	11	5	4	5	5	4	5	9
Store 4	11	5	5	10	11	10	6	5	5	6	11	5

Table 6.4: Significance sums per hyperparameterization (SAC on KDE-QDNP)

horizon. In parallel, the top five candidates were selected according to the mean of the performance distribution obtained from the evaluation of the final policy. This information is reported in the descriptive statistics in Tables C.1-C.4 for standard SAC and Tables-C.5-C.8 for the SAC on the KDE-QDNP application. These tables are found in Appendix C and include unique rankings based on the mean total reward achieved by each candidate, with higher values ranked first. Since the QDNP is a maximization problem, the best hyperparameterizations are listed in descending order in Table 6.2. The final selection corresponds to those that satisfy both criteria, namely smooth entropy behavior and inclusion among the top five performers.

A Shapiro-Wilk normality test was applied to the distributions of mean total profit obtained during the evaluation phase of both applications of SAC, where the final learned policy is executed. All cases satisfied the test, indicating that the distributions can be considered approximately normal.

Provided all distributions satisfied the normality assumption, pairwise t-tests were applied to compare them and identify statistically significant differences. The detailed results of these tests are reported in Appendix D for both SAC applications. For each pairwise comparison, a value of 1 is assigned if the null hypothesis is rejected (indicating a statistically significant difference) and 0 if it is not rejected. The corresponding significance sums represent the total number of null hypothesis rejections for each candidate. They are summarized in Table 6.3 for the SAC application on the stan-

Problem Instance	Store 1	Store 2	Store 3	Store 4
Hyperparameterization	H5	H3	H3	H3
SAC	1038.21	798.66	6910.44	2502.00
SAC (KDE-QDNP)	1080.00	873.60	6949.44	2623.09

Table 6.5: Runtimes of best hyperparameterizations per SAC application and problem instance (seconds)

dard QDNP and in Table 6.4 for the SAC application on the KDE-QDNP variant. The higher the sum, the more frequently a given hyperparameterization differs significantly from the others, whereas lower sums indicate candidates whose performance is more statistically similar to the rest.

The best hyperparameterizations identified for each problem instance and each SAC application were not unique in most cases. To resolve these ties, the results from the boxplots and the t-tests were examined in order to determine whether the competing candidates perform equivalently or if one can be considered clearly superior.

For the standard SAC method, no statistical differences were observed between any of the identified pairs of hyperparameterizations. The corresponding values in Figure D.1 are all equal to 0, indicating that the resulting p-values were higher than the considered significance level of 0.05. This conclusion is further supported by the boxplots in Figures E.1, E.2, and E.4, in Appendix E, which show nearly identical distributions for the compared candidates. Based on these results, hyperparameterization $H5$ was selected for Store 1, while hyperparameterization $H3$ was selected for Stores 2-4, as summarized in Table 6.2.

For the KDE-QDNP SAC application, the pairwise t-test comparison for Store 1 showed a value of 1 between hyperparameterizations 5 and 6 in Table D.2, indicating a p-value below 0.05 and thus a statistically significant difference. Hyperparameterization 5 emerged as the better choice, achieving a higher mean, which is also visually supported by the boxplots in Figure E.5, where its distribution lies consistently higher. A similar outcome was obtained for Store 3, where both the values in Table D.2 and the boxplots in Figure E.7 point to hyperparameterization 3 as the superior candidate. For Store 2, no significant difference was detected between hyperparameterizations 2 and 3, although Figure E.6 shows a slight displacement between their distributions. To maintain consistency across SAC applications and among problem instances, hy-

perparameterization 3 was selected for Store 2 as well. Finally, hyperparameterization 3 was the sole candidate for Store 4, making it the natural choice.

The tuning progress plots of the selected hyperparameterizations, showing the 100-point moving average of total reward and the entropy coefficient per timestep, are provided in Figure 6.1 for the standard SAC application and Figure 6.2 for the KDE-QDNP SAC application. These plots illustrate the learning dynamics of the chosen finalists, highlighting the stable reward trajectories and smooth entropy behavior observed during training.

Examining the boxplots and significance sums reveals that several hyperparameterizations perform at comparable levels in each scenario. In many cases, the medians of several candidates are very close to, or even coincident with, those of the top performers. These were nevertheless rejected due to insufficient stability or smoothness during training. Notably, although instability during learning is observed in such cases, the final policies they produce can still achieve results that are on par with the more stable setups. When looking at the overall picture, only a few hyperparameterizations in each problem instance appear to become trapped in what can be interpreted as local maxima, yielding consistently lower outcomes than the best-performing runs. This observation aligns with SAC's resilience to variations in hyperparameter configurations. All experiments were conducted with random generators seeded at 190. Each evaluation consisted of 100 runs of 1000 samples, and all cases passed a normality test. Given the large number of samples per run, the distribution of mean outcomes is approximately normal according to the Central Limit Theorem. Therefore, while different seeds could introduce minor variations, these are not expected to affect the overall conclusions, making extensive retesting across multiple seeds unnecessary.

Finally, as an additional observation, the chosen hyperparameterizations for each problem instance, besides Store 3, correspond to faster runtime implementations for both SAC applications. The best runtimes are summarized in Table 6.5, while detailed runtime information for all hyperparameterizations can be found in Tables F.1 and F.2 in Appendix F.

6.2 Comparative Analysis

The results of the SAC applications are compared with that of the analytical solution

Method	**Expected Profit**	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
Analytical	103.82	-	-	1.65	-	236.79	-	-
SAC	105.88	1.60	-1.08	1.65	1.25	229.17	0.85	0.15
SAC (KDE-QDNP)	108.62	1.67	-1.08	1.65	1.25	232.30	0.84	0.16
MC-BFGS	104.38	0.99	1.49	1.63	1.35	205.29	0.86	0.14

(a) Store 1

Method	**Expected Profit**	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
Analytical	87.17	-	-	1.66	-	210.06	-	-
SAC	89.38	1.56	-0.99	1.66	1.24	206.07	0.82	0.18
SAC (KDE-QDNP)	92.17	1.51	-0.99	1.66	1.25	203.44	0.84	0.16
MC-BFGS	91.61	1.04	3.01	1.63	1.34	185.72	0.83	0.17

(b) Store 2

Method	**Expected Profit**	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
Analytical	123.09	-	-	1.62	-	273.88	-	-
SAC	124.40	1.64	-1.27	1.62	1.25	267.40	0.82	0.18
SAC (KDE-QDNP)	126.41	1.95	-1.05	1.62	1.27	282.16	0.77	0.23
MC-BFGS	121.26	0.99	1.22	1.60	1.34	241.94	0.83	0.17

(c) Store 3

Method	**Expected Profit**	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
Analytical	69.29	-	-	1.62	-	186.75	-	-
SAC	67.99	1.30	-0.98	1.61	1.21	179.98	0.73	0.27
SAC (KDE-QDNP)	70.42	1.27	-1.06	1.61	1.21	178.53	0.73	0.23
MC-BFGS	75.37	1.01	0.66	1.60	1.31	171.66	0.70	0.3

(d) Store 4

Table 6.6: Aggregate (mean) results for all problem instances.

rreported in [6], as well as with the results of MC-BFGS for each problem instance. The comparison includes the total expected profit, the decision variables z_1 and z_2, the corresponding prices p_1 and p_2, the initial stocking quantity Q, and the inventory retention and donation percentages γ and $1-\gamma$, respectively. All the information is provided in Table 6.6. The analytical solution is considered as the ground truth. Therefore, only the variables included in [6] can be directly compared with the results obtained here. The differences for the expected profit, the price p_1, and for Q are reported in Table 6.7. In the difference tables (Tables 6.7a, 6.7c, and 6.7d), a plus sign

Instance	SAC	SAC (KDE)	MC-BFGS
Store 1	+2.06	+4.80	+0.56
Store 2	+2.21	+5.00	+4.44
Store 3	+1.31	+3.32	−1.83
Store 4	−1.30	+1.13	+6.08

(a) Expected profit differences

Instance	SAC	SAC (KDE)	MC-BFGS
Store 1	1.98%	4.62%	0.54%
Store 2	2.54%	5.74%	5.09%
Store 3	1.06%	2.70%	1.49%
Store 4	1.88%	1.63%	8.77%

(b) Relative error w.r.t the expected profit of the analytical solution

Instance	SAC	SAC (KDE)	MC-BFGS
Store 1	0.00	0.00	−0.03
Store 2	0.00	0.00	−0.03
Store 3	0.00	0.00	−0.02
Store 4	−0.01	−0.01	−0.02

(c) p_1 differences

Instance	SAC	SAC (KDE)	MC-BFGS
Store 1	−7.62	−4.49	−31.50
Store 2	−3.99	−6.62	−24.34
Store 3	−6.48	+8.28	−31.94
Store 4	−6.77	−8.22	−15.09

(d) Q differences

Table 6.7: Comparisons with the analytical solution (Ground Truth).

indicates overestimation, a minus sign indicates underestimation, and zero denotes exact agreement with the analytical solution. For the relative error table (Table 6.7b), only absolute percentage values are reported.

Although the model is not time-series based, it estimates the expected profit over a multi-hour selling period. The analytically derived value is treated as the reference or "true" value, and the relative errors express the difference between this benchmark and the approximated final expected profit produced by each implementation. This is analogous to the concept of mean absolute percentage error (MAPE) applied to a single observation, as discussed by Lewis [41], who considers forecasts with MAPE below 10% to be highly accurate approximations.

The expected profit is the primary metric of interest, and the aforementioned relative error is used to quantify deviations across implementations. According to Table 6.7b, SAC achieved consistent results, with relative errors no greater than 2.54% of the expected profit (Store 2), when compared to the analytical solution. Such errors are small and unlikely to meaningfully affect decision-making in practice. For SAC application on the KDE enhanced QDNP, relative errors increasd in all instances except Store 4, with the largest difference being 5.75% for Store 2. Although higher, the achieved error remains modest and unlikely to pose operational issues. Over the 78-hour selling horizon (see Table 5.1 in Chapter 5), this corresponds to an average

misestimate of approximately 0.06 per hour. At the given disposal and salvage values ($C_d = C_s = 0.05$), this is roughly equivalent to the value of one product unit per hour, essentially the scale of a single salvage or disposal event. In contrast, MC-BFGS yields inconsistent performance across problem instances. While errors are acceptable for Stores 1-3, the deviation for Store 4 reaches 8.77%, which approaches a magnitude that could be considered practically significant.

Tables 6.7c and 6.7d report the differences in p_1 and Q, respectively, compared with the analytical solution. For p_1, the differences for both SAC applications are identical and negligible, while MC-BFGS shows slightly larger but still manageable deviations of 0.02 to 0.03. This suggests that SAC provides more reliable approximations. For the stocking quantity Q, the deviations are notably higher for MC-BFGS, while SAC applications remain closer to the analytical solution, regardless of QDNP modeling.

Looking at the broader results in Table 6.6, which also include the decision variables z_1 and z_2, as well as p_2, γ, and $1 - \gamma$, both SAC applications produce consistent results across Stores 1, 2, and 3, even when the pairs of z_1 and z_2 differ. Store 4 shows larger discrepancies across all variables, but the resulting expected profit remains close to the analytical benchmark. It should be emphasized that all reported values were rounded to two decimal places for comparability with [6] as well as to reflect the level of precision used in managerial decision-making (e.g., supermarket prices are typically expressed up to two decimals). As such, some values that appear identical on the tables may in fact differ in finer detail, given the actual algorithm output is of higher precision. Therefore, larger observed deviations in expected profit are not necessarily inconsistent with seemingly identical intermediate variables.

Overall, the results demonstrate that SAC yields sufficiently accurate approximations across all four problem instances. Under SAC, the KDE enhanced version of the QDNP model produces results that are systematically higher than the analytical solution, as expected due to distributional approximation, but still within an acceptable margin. In contrast MC-BFGS shows less consistent performance. It is noteworthy that during experimentatino higher sensitivity to the choice of starting point was observed, with some runs leading to convergence failures, was observed. This difficulty is avoided with reinforcement learning, which does not require gradient information or careful initialization.

% of Original Value	a	b	ϕ	C_0	C_d	C_s	R
-10 %	0.16%	0.25%	0.02%	0.13%	0.47%	0.02%	0.12%
+10 %	0.30%	0.54%	0.17%	1.09%	0.01%	0.01%	0.30%
-20 %	4.86%	0.89%	0.08%	1.12%	0.01%	0.04%	1.01%
+20 %	0.74%	1.32%	0.02%	3.68%	0.05%	0.11%	1.63%
-30 %	16.09%	1.60%	0.12%	2.38%	0.03%	0.00%	1.76%
+30 %	1.49%	5.25%	0.21%	15.13%	0.04%	0.07%	4.30%

Table 6.8: Expected profit relative error across parameter perturbations

6.3 Zero-Shot Policy Transfer Results

The focus of this analysis is the expected profit, which is the central performance metric since its maximization is the direct objective of the QDNP. To evaluate generalization performance, the relative error is measured, defined as the proportional deviation between the expected profit obtained from a pretrained SAC policy on the Store 1 problem instance (the source task) when directly applied to a perturbed version of the same instance (a target task), and the corresponding ground truth. The ground truth is defined as the expected profit achieved when SAC is retrained from scratch under the perturbed parameter setting. Perturbations were introduced by adjusting each parameter individually across small, moderate, and large deviations from its original value, while leaving all other parameters unchanged.

The results presented in Table 6.8 show that SAC exhibits strong transfer capabilities in this task. For perturbations of 10%, relative errors remain negligible across all parameters and consistently fall below 1%, indicating that the zero-shot transferred policy generalizes reliably to small environment variations. At perturbations of 20%, performance remains robust with relative errors generally below 5%, although isolated increases such as a 20% reduction in a or a 20% increase in C_0 approach this threshold. At the largest perturbation level of 30%, most parameters still yield moderate errors, but notable outliers emerge for a 30% reduction in a with an error of 16.09% and a 30% increase in C_0 with an error of 15.13%. These cases exceed the 10% tolerance often cited for accurate approximations [41], suggesting that zero-shot transfer becomes unreliable under substantial changes in the demand parameter dependent on market size or in the per-unit purchasing cost.

The complete results, presented in AppendixG from Table G.1 to Table G.7, pro-

vide further insight into the behavior of the decision variables z_1 and z_2 as well as derived variables under each perturbation. While deviations are observable compared to the retrained ground truth cases, the overall expected profits remain closely aligned. This suggests that the transferred policy is able to exploit alternative decision pathways that yield near-equivalent returns, highlighting a degree of robustness in policy behavior even under partial model mismatch. As noted in Section 6.2, all reported values are rounded to two decimal places for comparability with [6] and to reflect managerial decision-making precision. Consequently, some values that appear identical may differ in finer detail, and observed deviations in expected profit can arise even when intermediate decision variables are close.

Overall, the findings indicate that SAC can generalize effectively across a broad range of individual parameter variations in the QDNP under zero-shot policy transfer. For small to moderate changes in demand or cost parameters, the transferred policy maintains near-optimal expected profit, reducing the need for frequent retraining and saving computational resources. However, once perturbations become sufficiently large in select parameters, retraining is required to recover near-optimal performance. These results suggest that SAC's zero-shot transferability can be leveraged to make operational decisions without retraining the model, unless substantial deviations from the source problem occur, highlighting SAC's suitability for online systems capable of accurate, real-time decision-making.

Chapter 7

Conclusion

The focal point of the present thesis was the examination of the application of SAC, an RL algorithm, to the QDNP and a data-driven KDE-based variant. The central objective was to assess whether SAC can serve as an effective alternative to analytical and classical optimization approaches, particularly under demand uncertainty, while also exploring the potential of policy transfer to handle environmental changes.

The results show that SAC reliably approximates the analytical solution with small errors across multiple real-world problem instances, remaining well within practical tolerances for decision-making. The KDE-enhanced model introduced slight increases in approximation error but remained within the same tolerance levels that deem the framework successful in estimation. In comparison, the gradient-based MC-BFGS method used as a baseline approach was less stable and more sensitive to initialization, underscoring the robustness and flexibility of RL in this context.

Moreover, zero-shot policy transfer experiments showed that a pretrained SAC policy can generalize effectively across changes in demand and cost parameters. For small and medium perturbations, deviations from retrained ground truth solutions were negligible, suggesting that SAC offers practical flexibility in dynamic environments. Significant performance degradation was only observed for larger pertrurbations. In such scenarios, retraining is required to restore near-optimality.

The results add to the discussion on how RL can be applied in operations management. Evidence from this study suggests that SAC can provide a stable and effective way of solving the QDNP and its data-driven variant. The considered problem setting

was shown to be a useful testbed as it offers an analytical benchmark for comparison, it contains sequential decision-making embedded in its formulation, and the stochastic component is isolated in a controlled and testable way. These features make it possible to examine SAC's behavior under clear conditions, while also pointing to its potential in more dynamic, data-driven environments where adaptability and generalization matter most.

Future work could extend the current experiments by incorporating a larger and more diverse set of store instances and by exploring alternative quality deterioration patterns, such as exponential decay relevant for products like meat. Additional opportunities include considering a wider range of demand distributions, evaluating policy transfer under simultaneous changes in multiple parameters, and integrating more data-driven components. Expanding along these directions would allow for a more comprehensive understanding of SAC's performance and generalization capabilities in complex, real-world environments.

In conclusion, by applying SAC to the QDNP and its data-driven extension, this thesis provides a focused example of RL within a well-structured inventory model and lays the groundwork for extending such methods to more complex and data-driven decision problems. The demonstrated accuracy, robustness, and zero-shot transferability of SAC suggest that the framework can support operational decision-making in practice, enabling near-optimal stocking and pricing decisions even under changing demand and cost conditions. This highlights the potential of RL as a flexible tool for firms seeking adaptive, data-driven inventory management strategies that reduce reliance on frequent retraining or rigid analytical models.

Bibliography

[1] K. J. Arrow, *Studies in the mathematical theory of inventory and production / by Kenneth J. Arrow, Samuel Karlin [and] Herbert Scarf. With contributions by Martin J. Beckmann, John Gessford [and] Richard F. Muth.*, ser. Stanford mathematical studies in the social sciences. Stanford, Calif: Stanford University Press, 1958.

[2] G. Hadley and T. Whitin, *Analysis of Inventory Systems*, ser. International series in management). Prentice-Hall, 1963. [Online]. Available: https://books.google.gr/books?id=TrQ-AAAAIAAJ

[3] D. Erlenkotter, "Ford whitman harris and the economic order quantity model," *Operations Research*, vol. 38, no. 6, pp. 937–946, December 1990. [Online]. Available: https://ideas.repec.org/a/inm/oropre/v38y1990i6p937-946.html

[4] F. W. Harris, "How many parts to make at once," *Operations Research*, vol. 38, no. 6, pp. 947–950, December 1990. [Online]. Available: https://ideas.repec.org/a/inm/oropre/v38y1990i6p947-950.html

[5] M. Spearman, "Of physics and factory physics," *Production and Operations Management*, vol. In press, 03 2014.

[6] A. Özbilge, E. Hassini, and M. Parlar, "Optimal pricing and donation policy for fresh goods," *European Journal of Operational Research*, vol. 312, no. 1, pp. 198–210, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221723004708

[7] N. C. Petruzzi and M. Dada, "Pricing and the newsvendor problem: A review with extensions," *Operations Research*, vol. 47, no. 2, pp. 183–194, 1999.

[8] W. B. Powell, *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. Hoboken, NJ: John Wiley & Sons, Inc., 2022. [Online]. Available: https://doi.org/10.1002/9781119815068

[9] R. N. Boute, J. Gijsbrechts, W. van Jaarsveld, and N. Vanvuchelen, "Deep reinforcement learning for inventory control: A roadmap," *European Journal of Operational Research*, vol. 298, no. 2, pp. 401–412, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221721006111

[10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290

[11] F. Y. Edgeworth, "The mathematical theory of banking," *Journal of the Royal Statistical Society*, vol. 51, no. 1, pp. 113–127, 1888. [Online]. Available: http://www.jstor.org/stable/2979084

[12] P. Morse, G. Kimball, and S. Gass, *Methods of Operations Research*, ser. Dover Books on Computer Science. Dover Publications, 2012. [Online]. Available: https://books.google.gr/books?id=NdlQAQAAQBAJ

[13] T. M. Whitin, "Inventory control and price theory," *Management Science*, vol. 2, no. 1, pp. 61–68, 1955. [Online]. Available: http://www.jstor.org/stable/2627238

[14] E. S. Mills, "Uncertainty and price theory," *The Quarterly Journal of Economics*, vol. 73, no. 1, pp. 116–130, None 1959. [Online]. Available: https://ideas.repec.org/a/oup/qjecon/v73y1959i1p116-130..html

[15] E. Mills, *Price, Output, and Inventory Policy: A Study in the Economics of the Firm and Industry*, ser. Operations Research Society of America. Publications in operations research. Wiley, 1962. [Online]. Available: https://books.google.gr/books?id=QTYYAAAAIAAJ

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[17] H. Dong, Z. Ding, and S. Zhang, *Deep reinforcement learning : fundamentals, research and applications*. Singapore: Springer, 2020.

[18] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*. Springer Publishing Company, Incorporated, 2014.

[19] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[20] P. Bowman, J. Ng, M. Harrison, T. Sánchez López, and A. Ilic, "Sensor-based condition monitoring," *BRIDGE Deliverables*, no. D-3.6, 2009.

[21] A. Osvald and L. Z. Stirn, "A vehicle routing algorithm for the distribution of fresh vegetables and similar perishable food," *Journal of Food Engineering*, vol. 85, no. 2, pp. 285–295, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0260877407004141

[22] E. Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065 – 1076, 1962. [Online]. Available: https://doi.org/10.1214/aoms/1177704472

[23] E. Silver, D. Pyke, and D. Thomas, *Inventory and Production Management in Supply Chains*. Taylor & Francis, 2016. [Online]. Available: https://books.google.gr/books?id=uY2ODwAAQBAJ

[24] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832 – 837, 1956. [Online]. Available: https://doi.org/10.1214/aoms/1177728190

[25] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, ser. Springer series in statistics. Springer, 2009. [Online]. Available: https://books.google.gr/books?id=eBSgoAEACAAJ

[26] V. A. Epanechnikov, "Non-parametric estimation of a multivariate probability density," *Theory of Probability & Its Applications*, vol. 14, no. 1, pp. 153–158, 1969. [Online]. Available: https://doi.org/10.1137/1114019

[27] M. Wand and M. Jones, *Kernel Smoothing*, ser. Chapman & Hall/CRC Monographs on Statistics and Applied Probability. CRC Press, 1994. [Online]. Available: https://books.google.gr/books?id=lUFZDwAAQBAJ

[28] D. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, ser. A Wiley-interscience publication. Wiley, 1992. [Online]. Available: https://books.google.gr/books?id=7crCUS_F2ocC

[29] D. W. Scott, "Scott's rule," *WIREs Computational Statistics*, vol. 2, no. 4, pp. 497–502, 2010. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.103

[30] H. Läuter, "Silverman, b. w.: Density estimation for statistics and data analysis. chapman & hall, london – new york 1986, 175 pp., £12.—," *Biometrical Journal*, vol. 30, no. 7, pp. 876–877, 1988. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.4710300745

[31] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[32] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[33] F. Nielsen, "What is...an information projection?" *Notices of the American Mathematical Society*, vol. 65, p. 1, 03 2018.

[34] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, 2007. [Online]. Available: http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738

[35] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, ser. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2011. [Online]. Available: https://books.google.gr/books?id=pQws07tdpjoC

[36] X. Wang and D. Li, "A dynamic product quality evaluation based pricing model for perishable food supply chains," *Omega*, vol. 40, no. 6, pp. 906–917, 2012, special Issue on Forecasting in Management Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048312000412

[37] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of*

Machine Learning Research, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[38] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. [Online]. Available: https://books.google.gr/books?id=VbHYoSyelFcC

[39] A. Lazaric, *Transfer in Reinforcement Learning: A Framework and a Survey*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 143–173. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_5

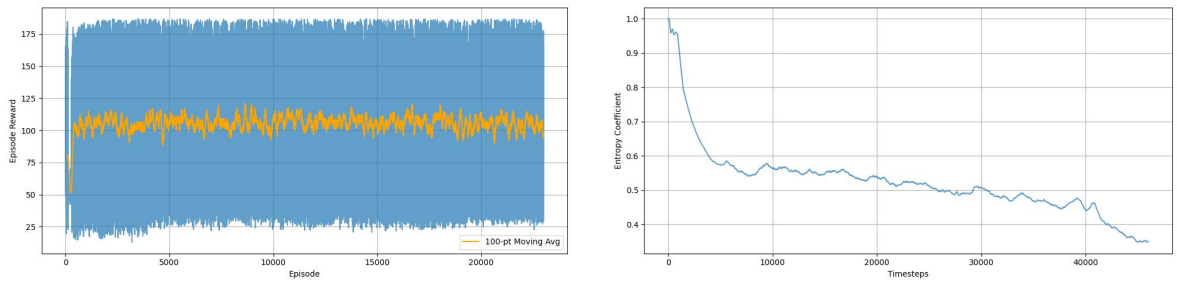
[40] Z. Wu, Y. Xie, W. Lian, C. Wang, Y. Guo, J. Chen, S. Schaal, and M. Tomizuka, "Zero-shot policy transfer with disentangled task representation of meta-reinforcement learning," 2022. [Online]. Available: https://arxiv.org/abs/2210.00350

[41] C. D. Lewis, *Industrial and business forecasting methods : a practical guide to exponential smoothing and curve fitting*. London ;: Butterworth Scientific, 1982.

Appendix A

SAC Training Tracking

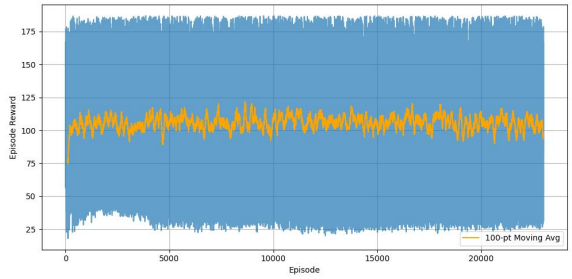
A.1 Store 1



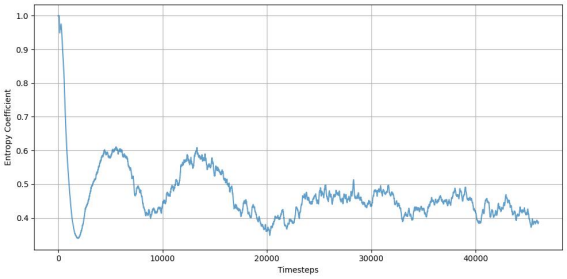
(a) Reward progress per episode (H1)

(b) Entropy coefficient per timestep (H1)

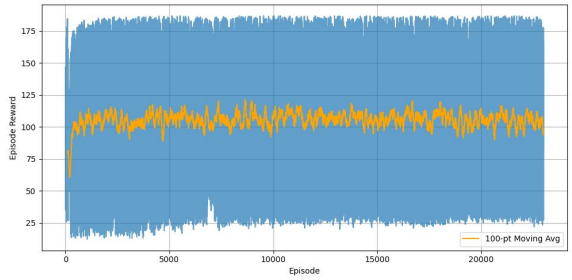
Figure A.1: Store 1 - Hyperparameterizations (H): 1-1



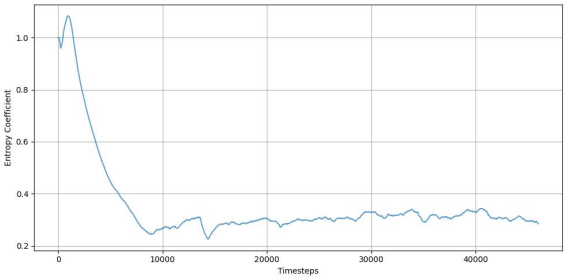
(a) Reward progress per episode (H2)



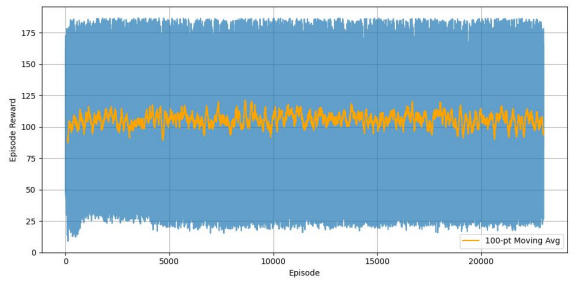
(b) Entropy coefficient per timestep (H2)



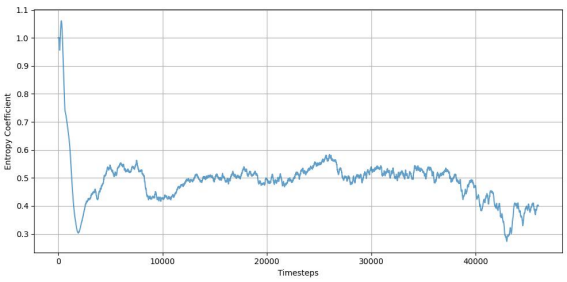
(c) Reward progress per episode (H3)



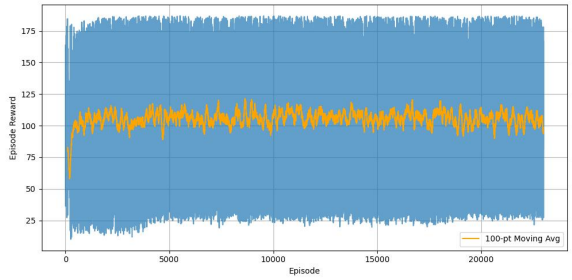
(d) Entropy coefficient per timestep (H3)



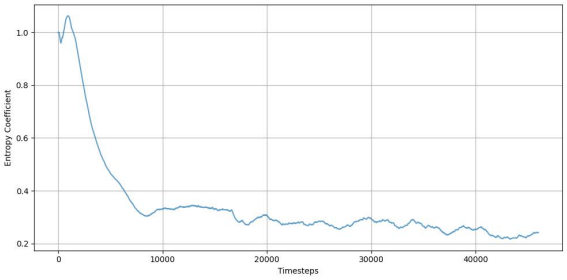
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

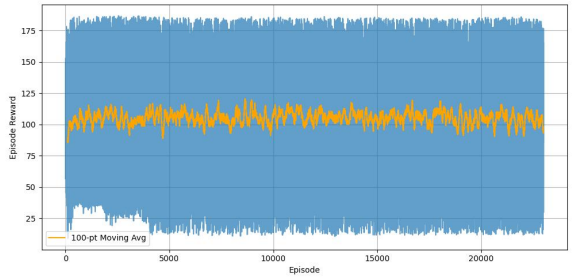


(g) Reward progress per episode (H5)

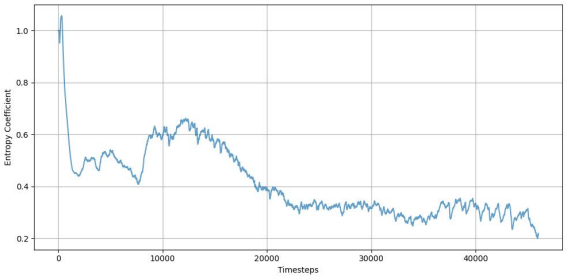


(h) Entropy coefficient per timestep (H5)

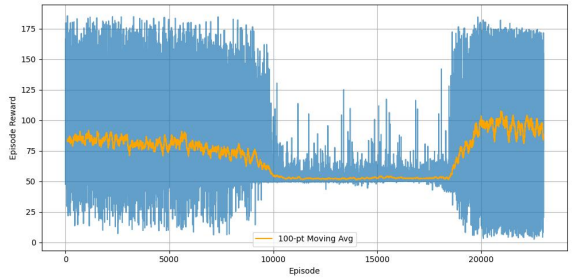
Figure A.2: Store 1 - Hyperparameterizations (H): 2-5



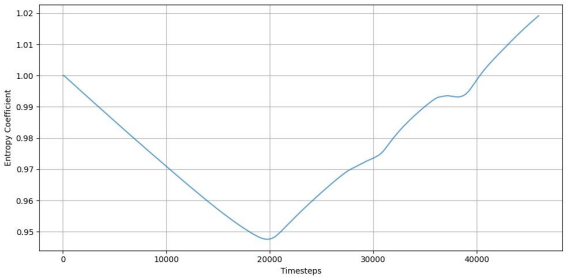
(a) Reward progress per episode (H6)



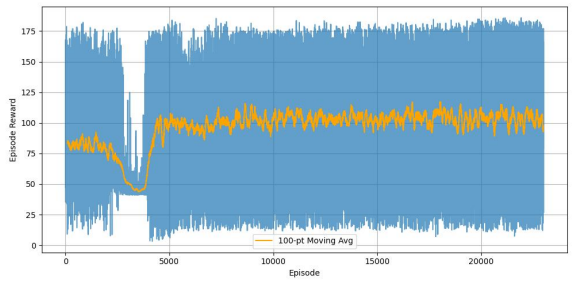
(b) Entropy coefficient per timestep (H6)



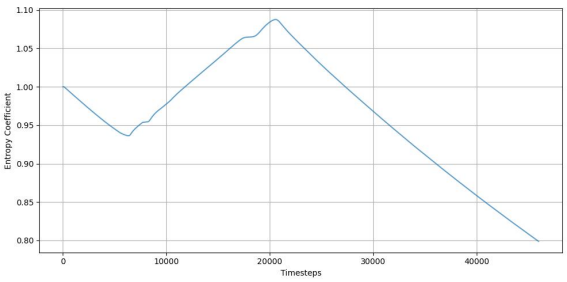
(c) Reward progress per episode (H7)



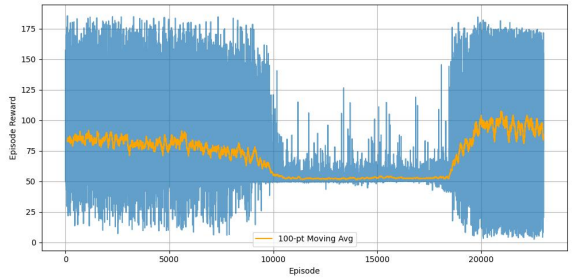
(d) Entropy coefficient per timestep (H7)



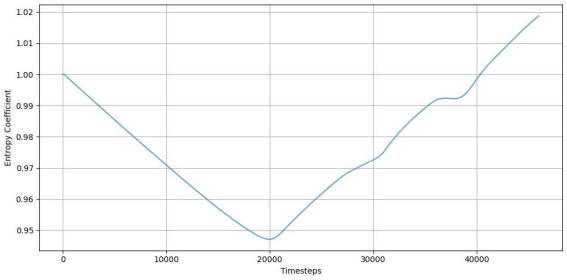
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

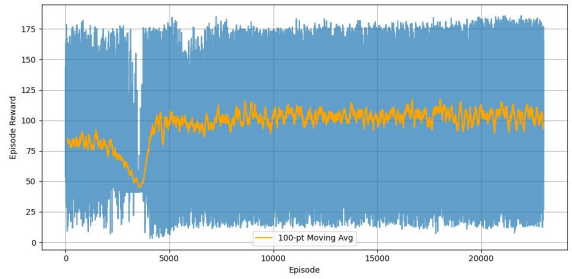


(g) Reward progress per episode (H9)

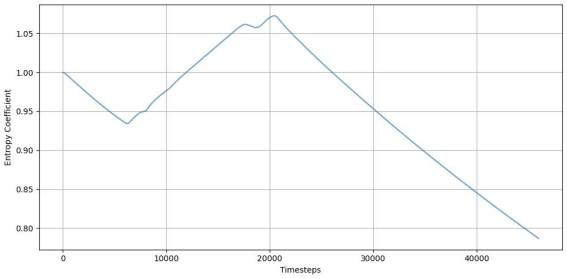


(h) Entropy coefficient per timestep (H9)

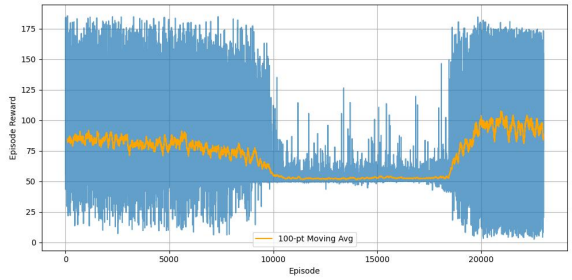
Figure A.3: Store 1 - Hyperparameterizations (H): 6-9



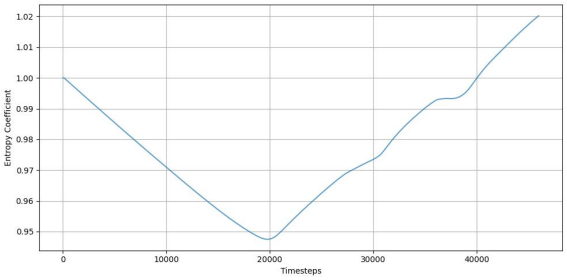
(a) Reward progress per episode (H10)



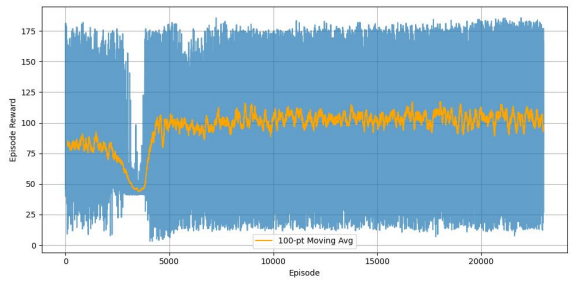
(b) Entropy coefficient per timestep (H10)



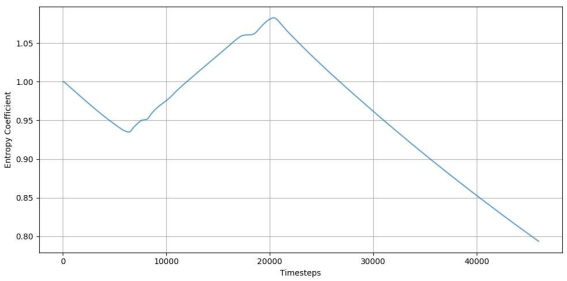
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



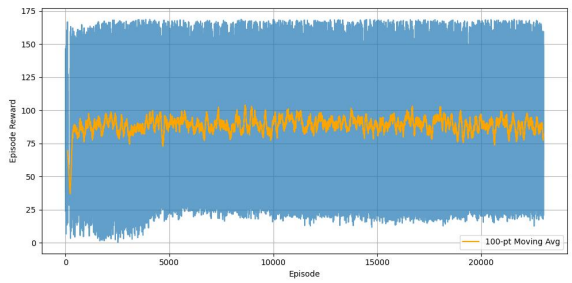
(e) Reward progress per episode (H12)



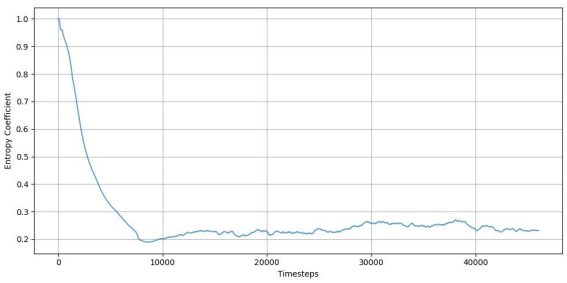
(f) Entropy coefficient per timestep (H12)

Figure A.4: Store 1 - Hyperparameterizations (H): 10-12

A.2 Store 2

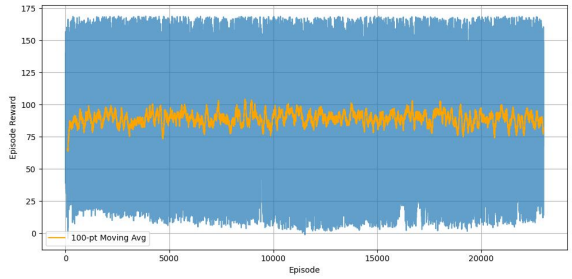


(a) Reward progress per episode (H1)

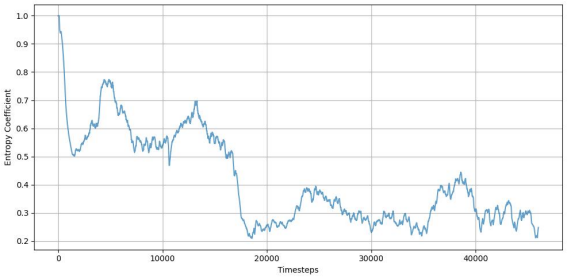


(b) Entropy coefficient per timestep (H1)

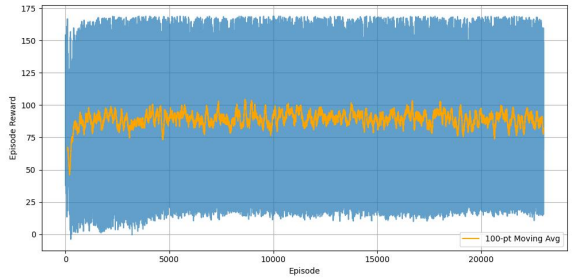
Figure A.5: Store 2 - Hyperparameterizations (H): 1-1



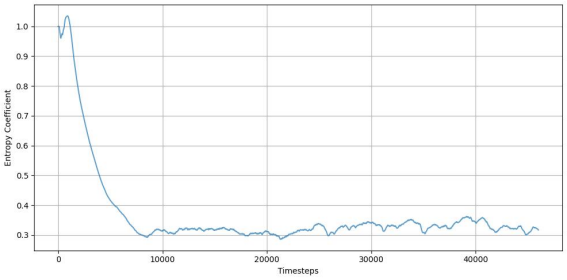
(a) Reward progress per episode (H2)



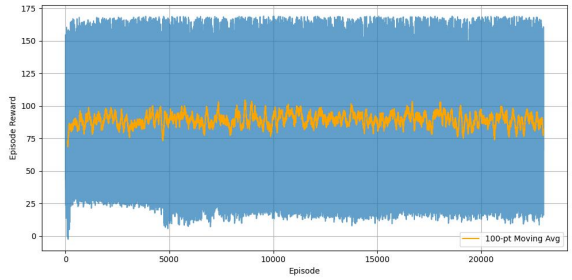
(b) Entropy coefficient per timestep (H2)



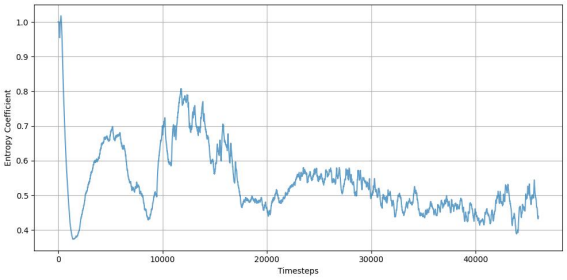
(c) Reward progress per episode (H3)



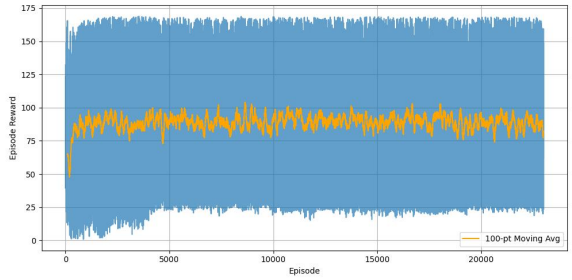
(d) Entropy coefficient per timestep (H3)



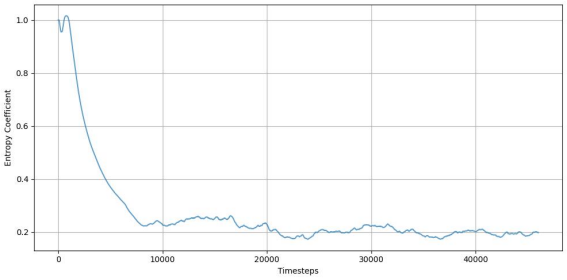
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

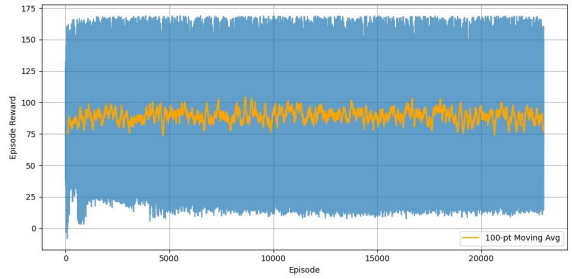


(g) Reward progress per episode (H5)

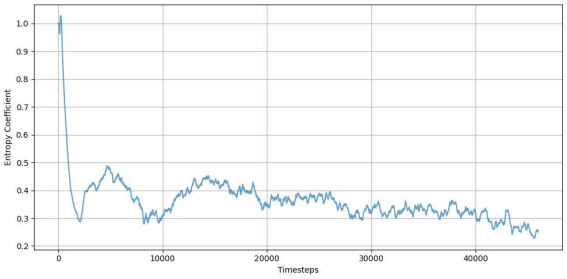


(h) Entropy coefficient per timestep (H5)

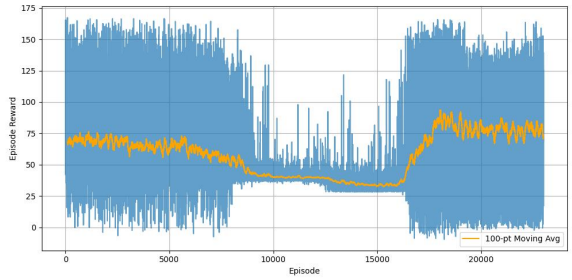
Figure A.6: Store 2 - Hyperparameterizations (H): 2-5



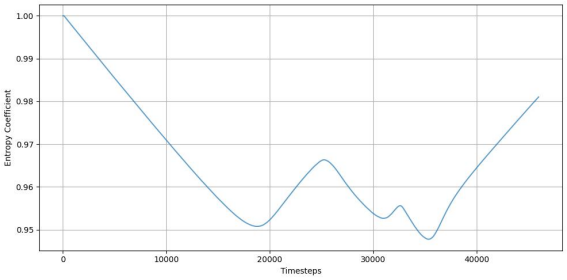
(a) Reward progress per episode (H6)



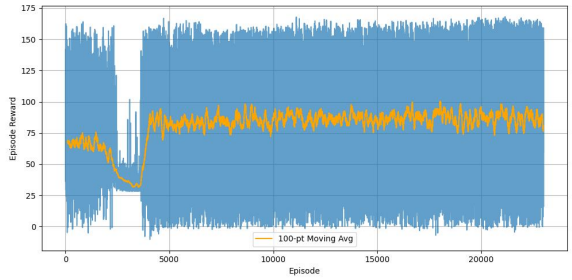
(b) Entropy coefficient per timestep (H6)



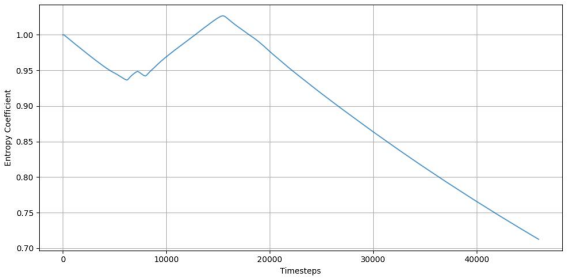
(c) Reward progress per episode (H7)



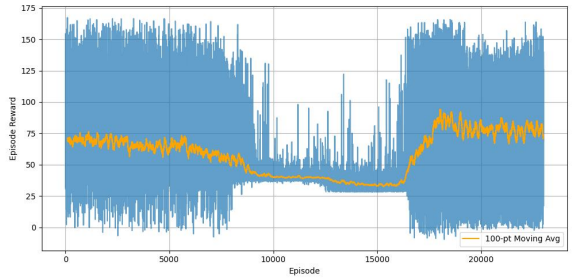
(d) Entropy coefficient per timestep (H7)



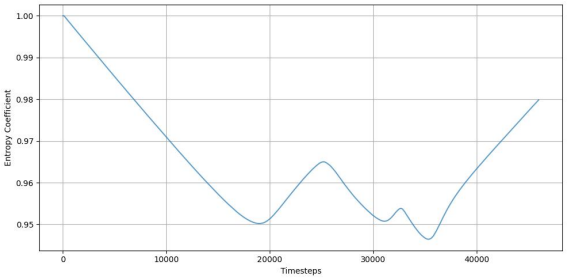
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

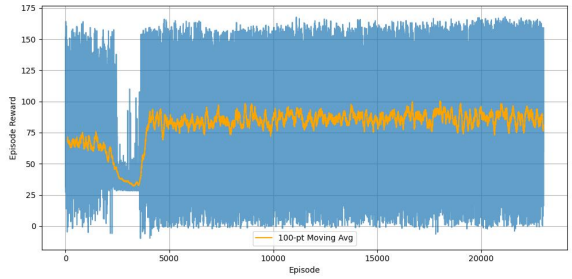


(g) Reward progress per episode (H9)

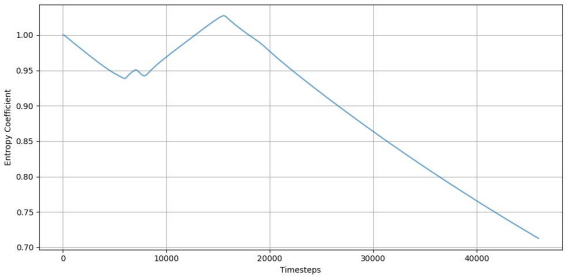


(h) Entropy coefficient per timestep (H9)

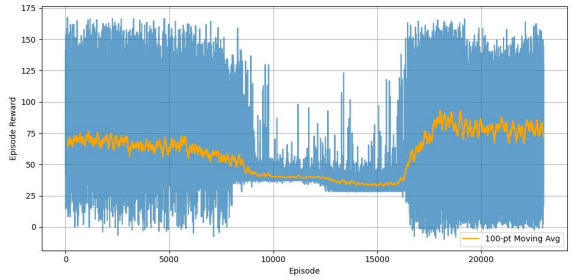
Figure A.7: Store 2 - Hyperparameterizations (H): 6-9



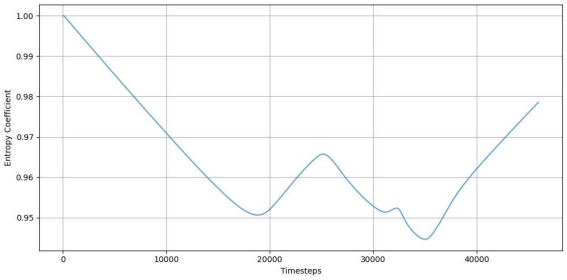
(a) Reward progress per episode (H10)



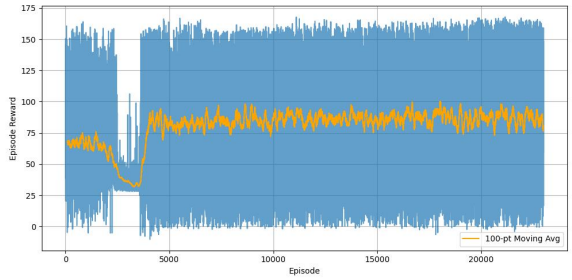
(b) Entropy coefficient per timestep (H10)



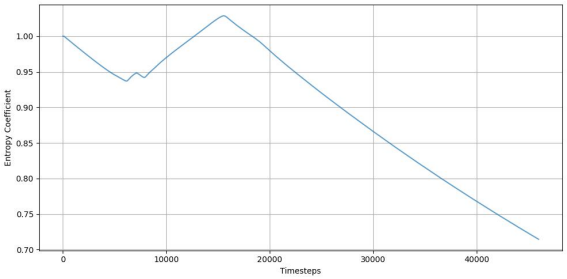
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



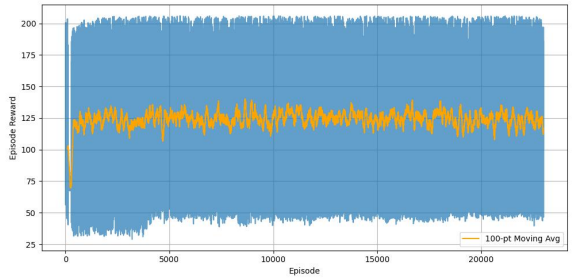
(e) Reward progress per episode (H12)



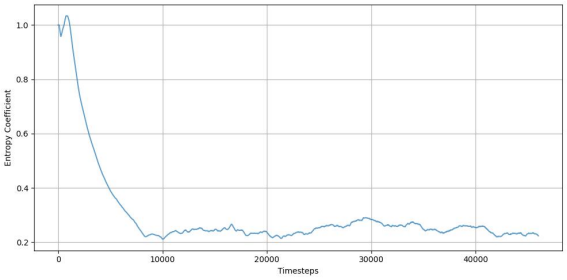
(f) Entropy coefficient per timestep (H12)

Figure A.8: Store 2 - Hyperparameterizations (H): 10-12

A.3 Store 3

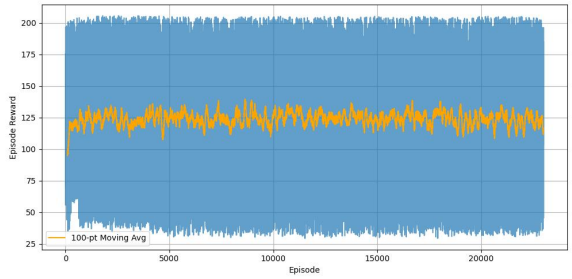


(a) Reward progress per episode (H1)

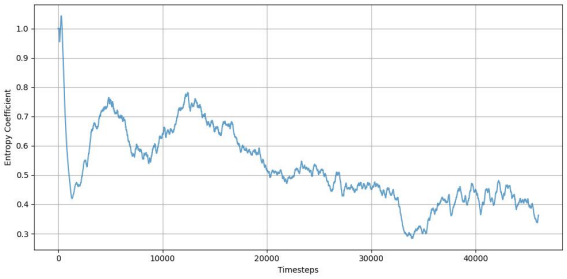


(b) Entropy coefficient per timestep (H1)

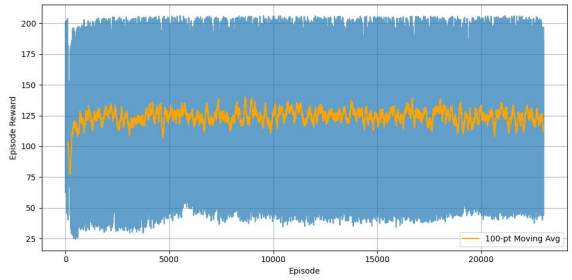
Figure A.9: Store 3 - Hyperparameterizations (H): 1-1



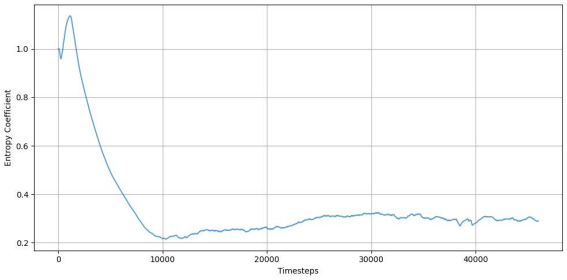
(a) Reward progress per episode (H2)



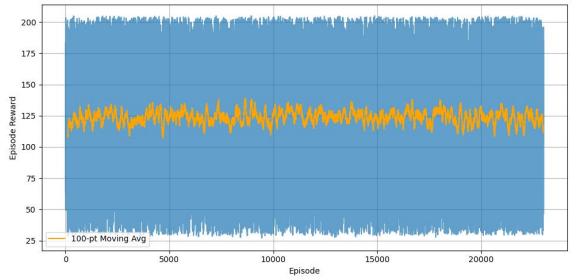
(b) Entropy coefficient per timestep (H2)



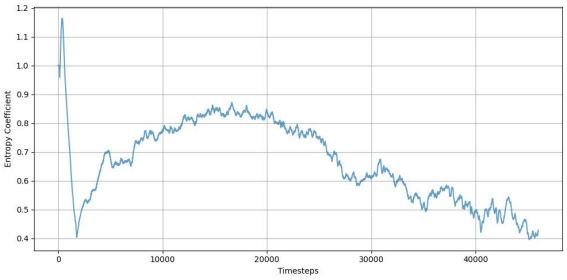
(c) Reward progress per episode (H3)



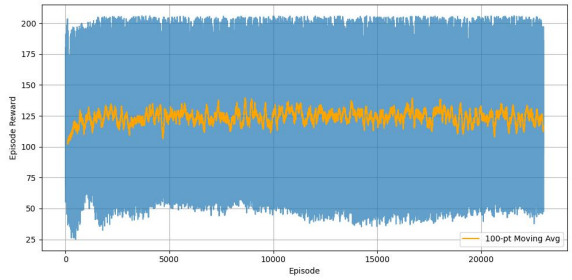
(d) Entropy coefficient per timestep (H3)



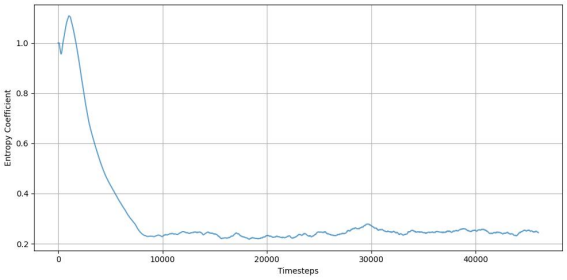
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

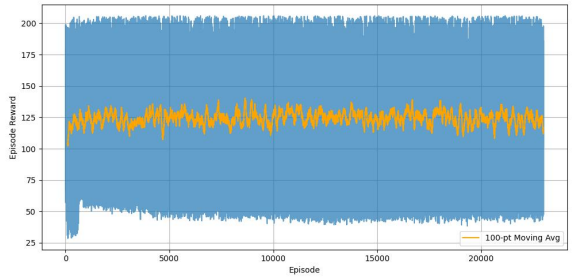


(g) Reward progress per episode (H5)

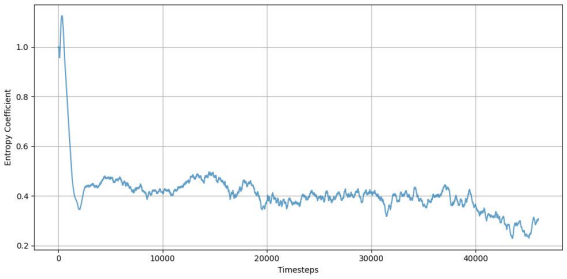


(h) Entropy coefficient per timestep (H5)

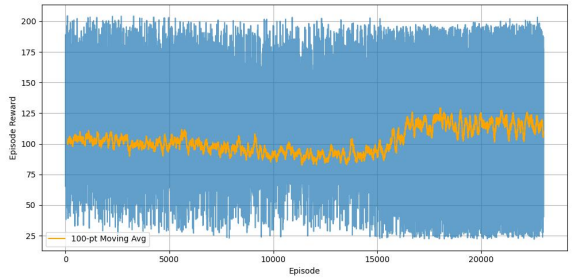
Figure A.10: Store 3 - Hyperparameterizations (H): 2-5



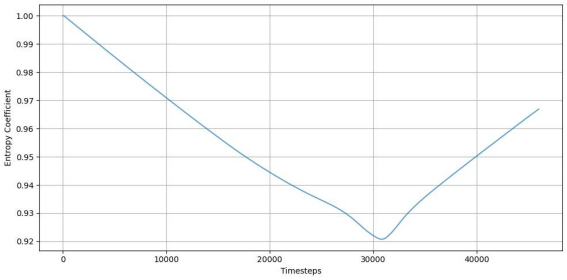
(a) Reward progress per episode (H6)



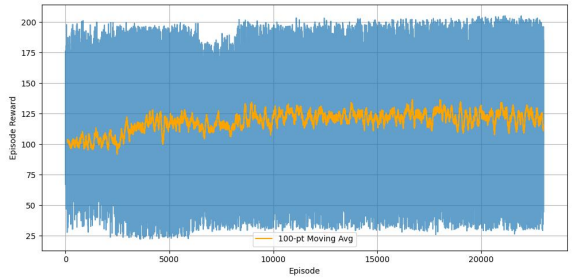
(b) Entropy coefficient per timestep (H6)



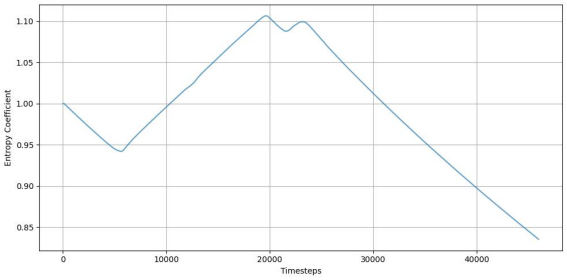
(c) Reward progress per episode (H7)



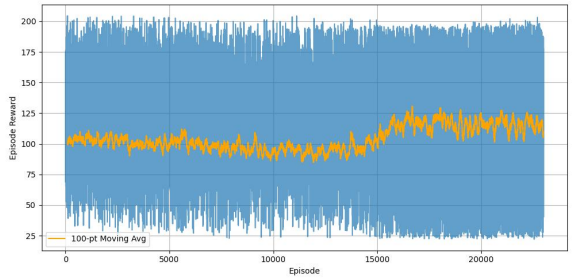
(d) Entropy coefficient per timestep (H7)



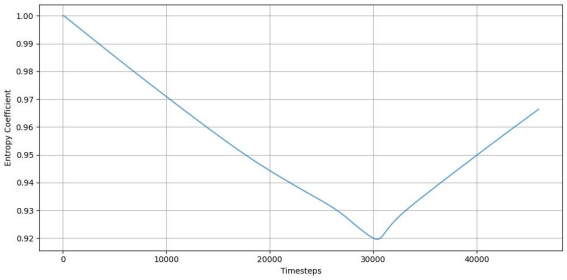
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

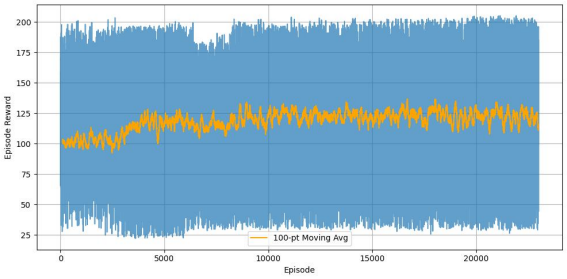


(g) Reward progress per episode (H9)

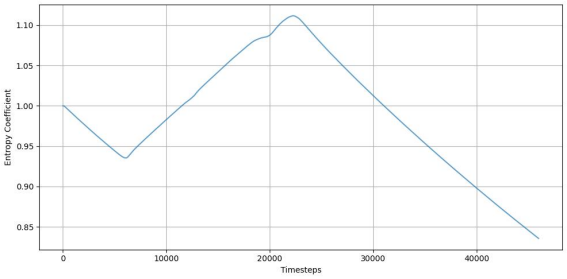


(h) Entropy coefficient per timestep (H9)

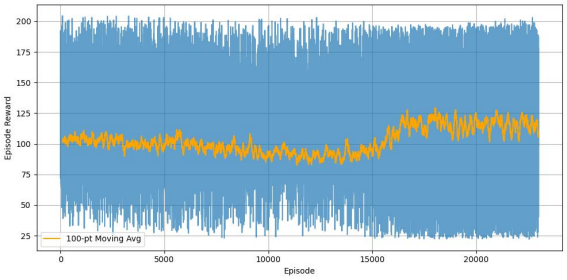
Figure A.11: Store 3 - Hyperparameterizations (H): 6-9



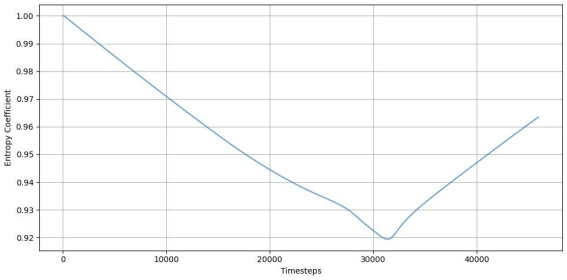
(a) Reward progress per episode (H10)



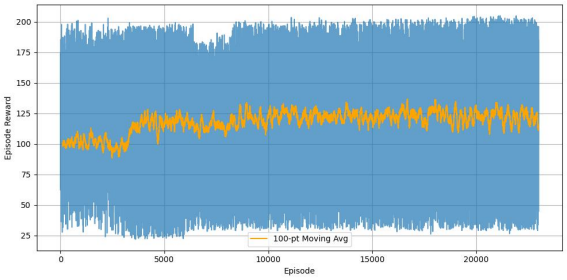
(b) Entropy coefficient per timestep (H10)



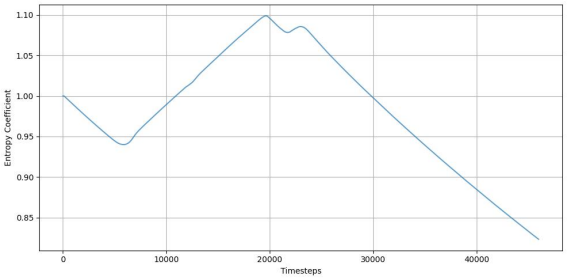
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



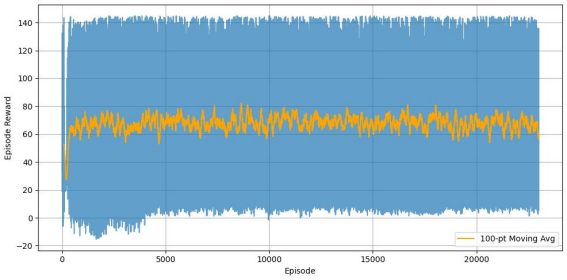
(e) Reward progress per episode (H12)



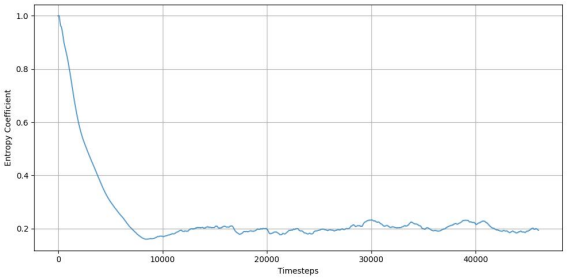
(f) Entropy coefficient per timestep (H21)

Figure A.12: Store 3 - Hyperparameterizations (H): 10-12

A.4 Store 4

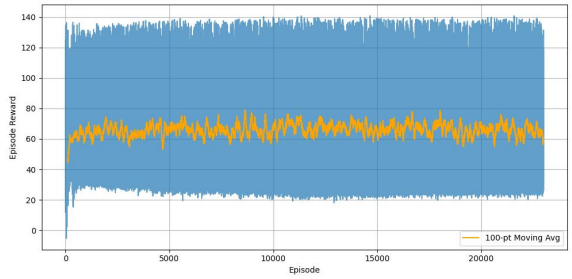


(a) Reward progress per episode (H1)

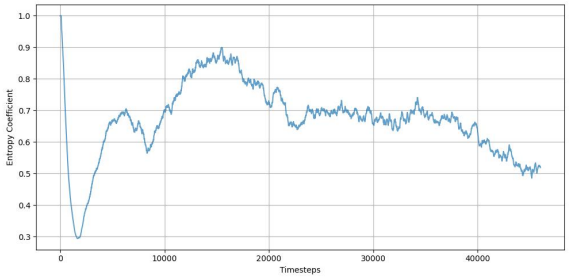


(b) Entropy coefficient per timestep (H1)

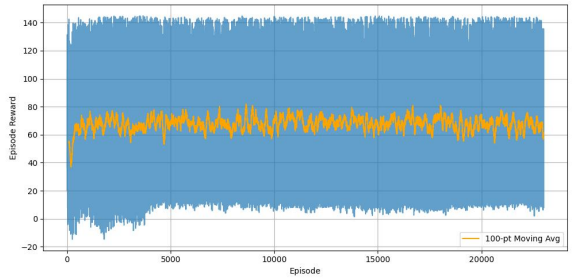
Figure A.13: Store 4 - Hyperparameterizations (H): 1-1



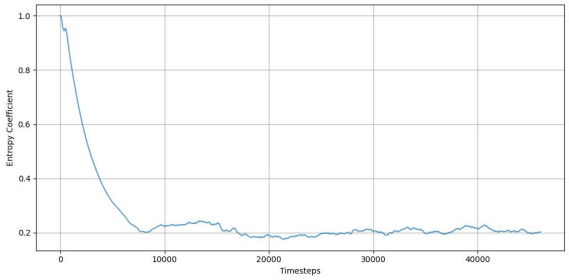
(a) Reward progress per episode (H2)



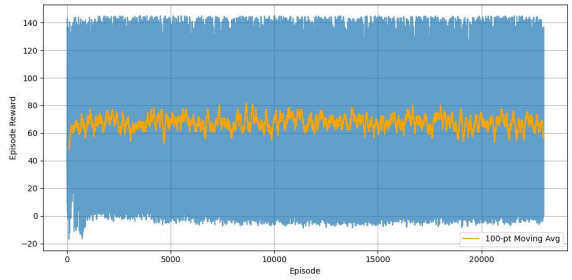
(b) Entropy coefficient per timestep (H2)



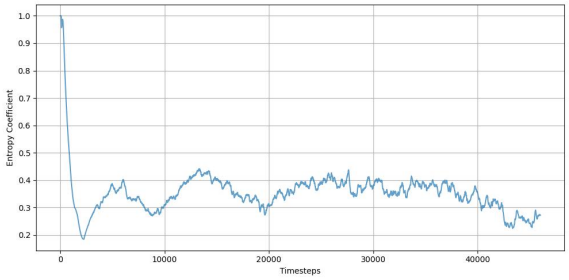
(c) Reward progress per episode (H3)



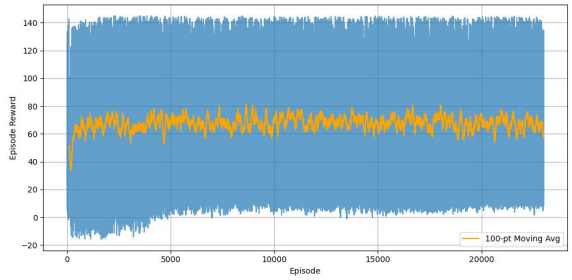
(d) Entropy coefficient per timestep (H3)



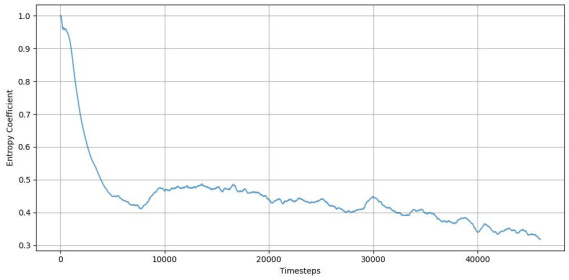
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

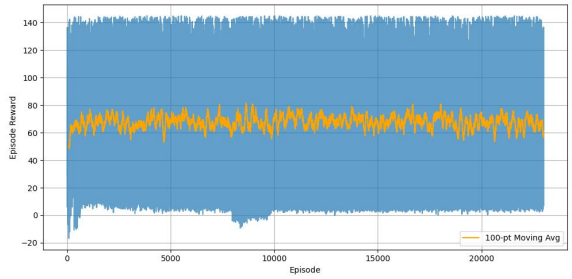


(g) Reward progress per episode (H5)

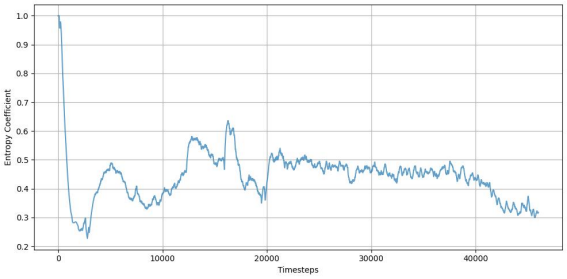


(h) Entropy coefficient per timestep (H5)

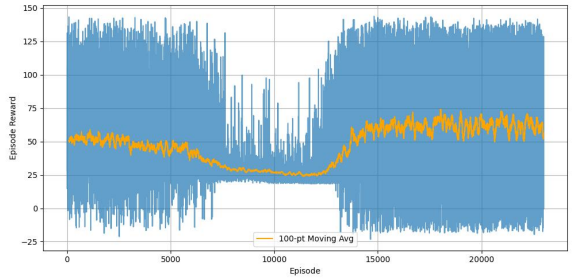
Figure A.14: Store 4 - Hyperparameterizations (H): 2-5



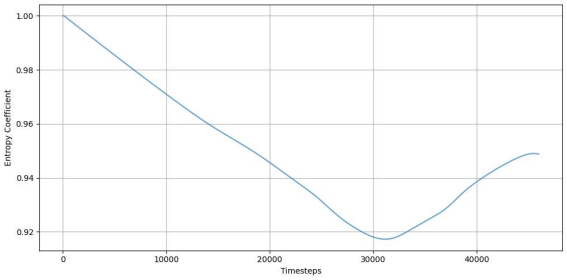
(a) Reward progress per episode (H6)



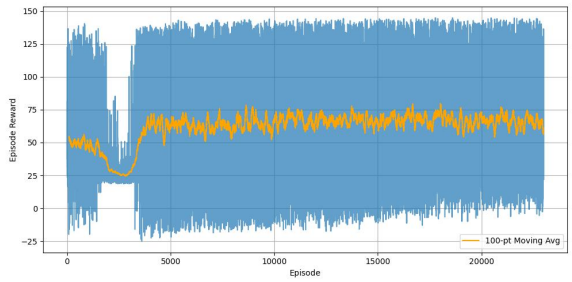
(b) Entropy coefficient per timestep (H6)



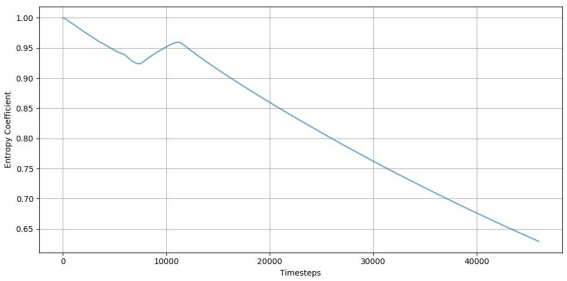
(c) Reward progress per episode (H7)



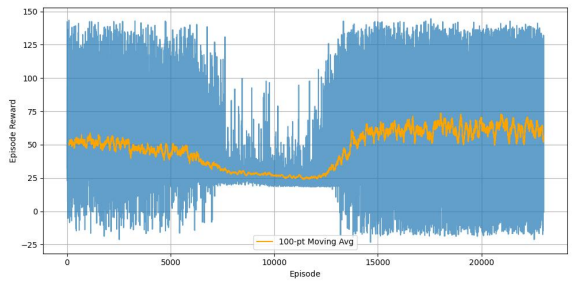
(d) Entropy coefficient per timestep (H7)



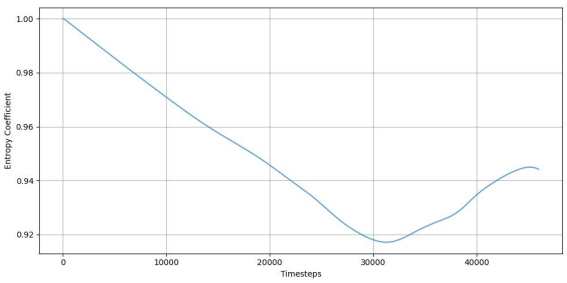
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

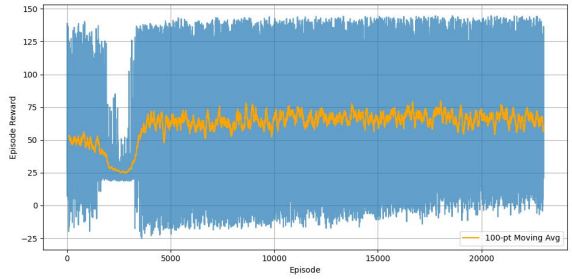


(g) Reward progress per episode (H9)

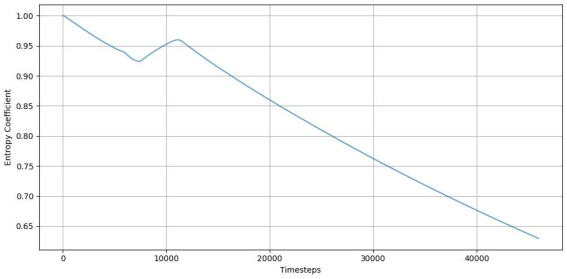


(h) Entropy coefficient per timestep (H9)

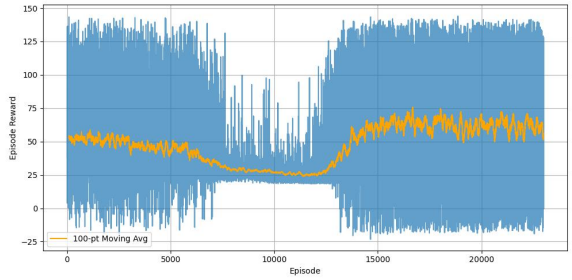
Figure A.15: Store 4 - Hyperparameterizations (H): 6-9



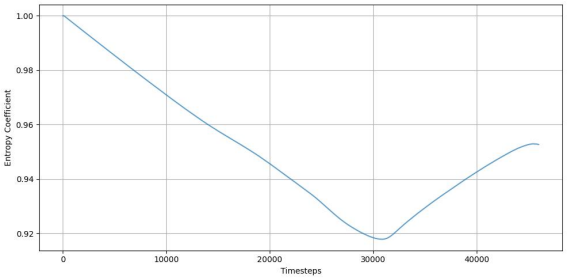
(a) Reward progress per episode (H10)



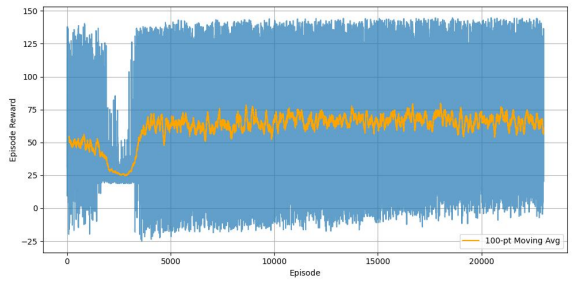
(b) Entropy coefficient per timestep (H10)



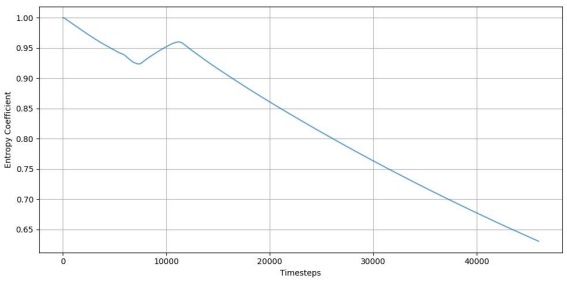
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



(e) Reward progress per episode (H12)



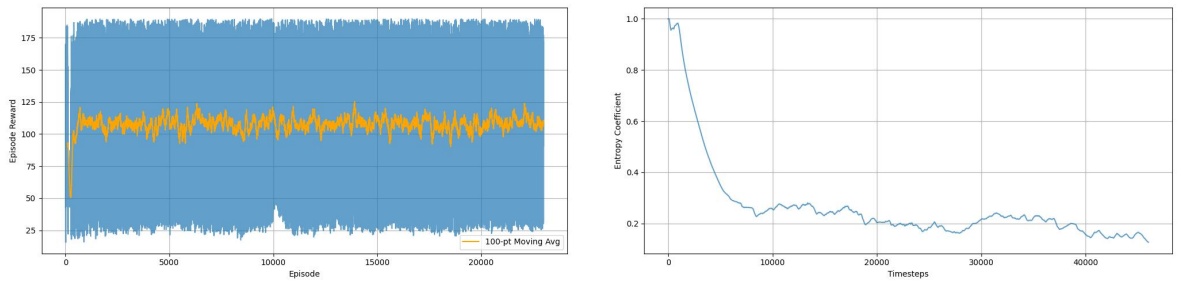
(f) Entropy coefficient per timestep (H21)

Figure A.16: Store 4 - Hyperparameterizations (H): 10-12

Appendix B

SAC (KDE-QDNP) Training Tracking

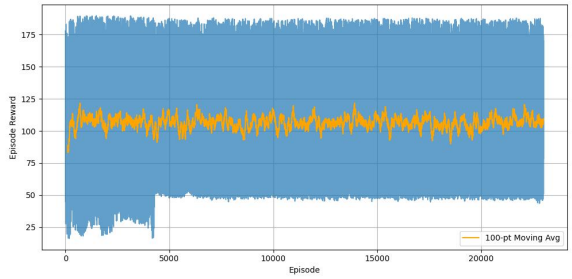
B.1 Store 1



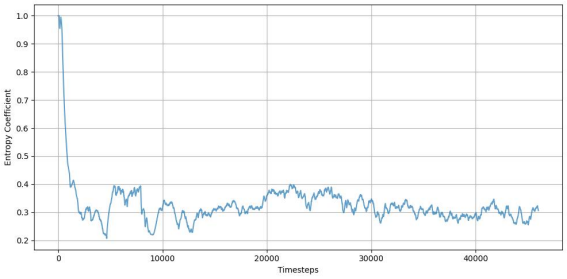
(a) Reward progress per episode (H1)

(b) Entropy coefficient per timestep (H1)

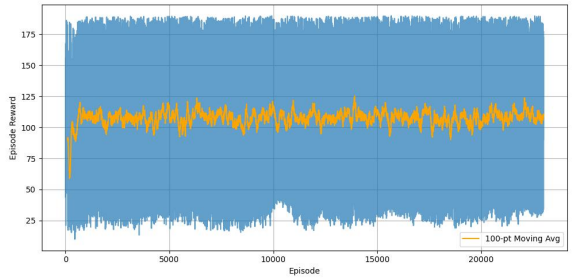
Figure B.1: Store 1 - Hyperparameterizations (H): 1-1



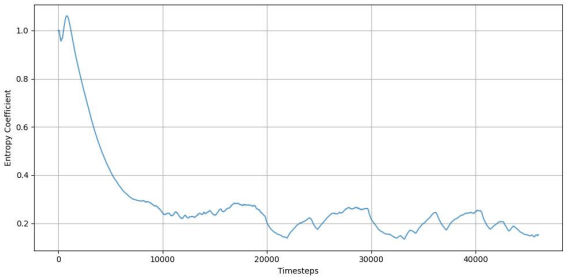
(a) Reward progress per episode (H2)



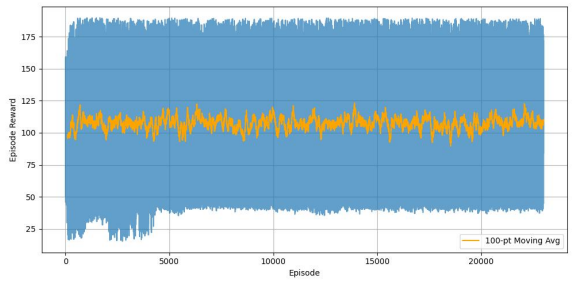
(b) Entropy coefficient per timestep (H2)



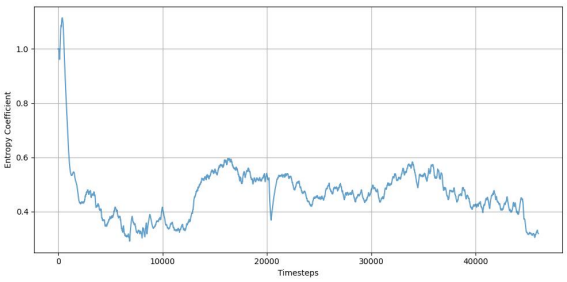
(c) Reward progress per episode (H3)



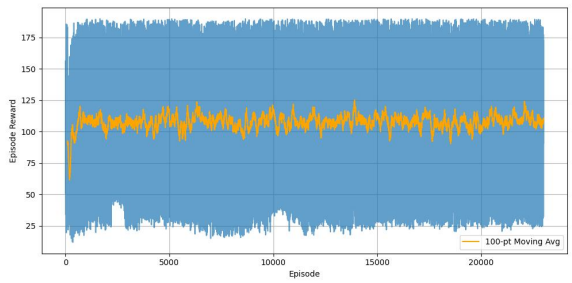
(d) Entropy coefficient per timestep (H3)



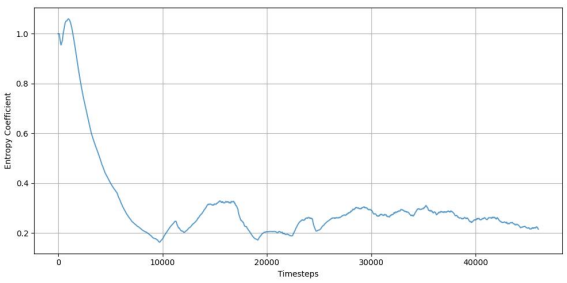
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

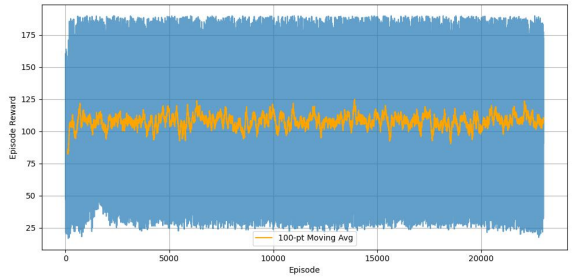


(g) Reward progress per episode (H5)

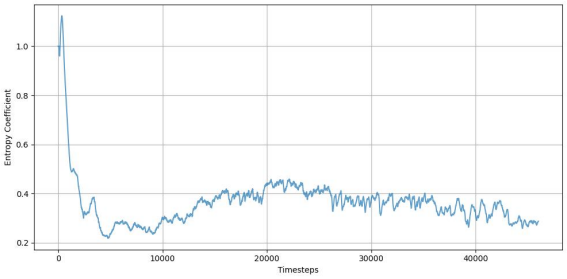


(h) Entropy coefficient per timestep (H5)

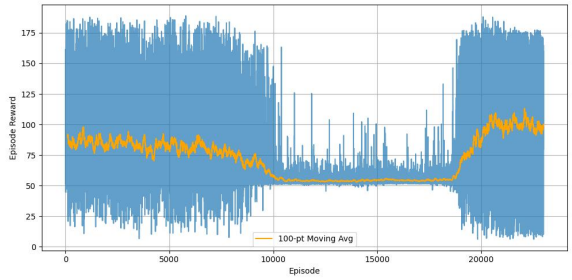
Figure B.2: Store 1 - Hyperparameterizations (H): 2-5



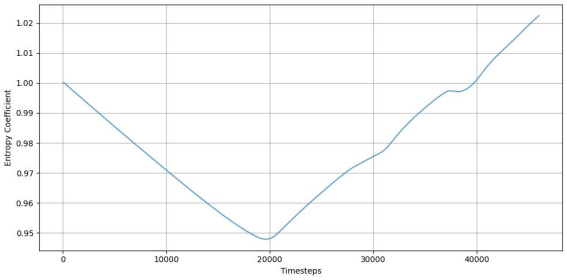
(a) Reward progress per episode (H6)



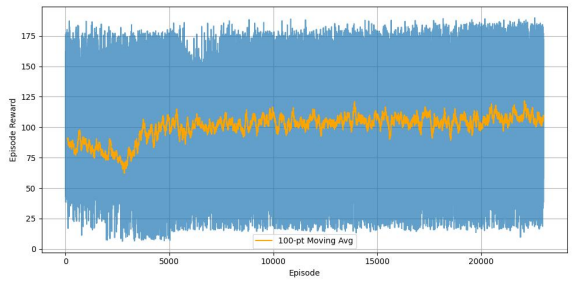
(b) Entropy coefficient per timestep (H6)



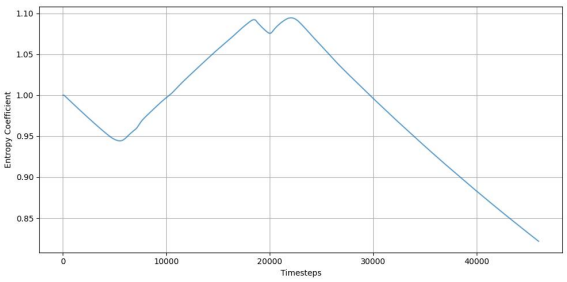
(c) Reward progress per episode (H7)



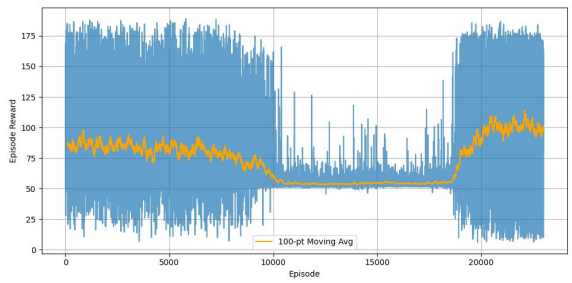
(d) Entropy coefficient per timestep (H7)



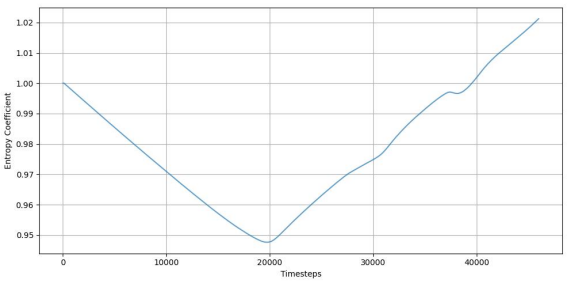
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

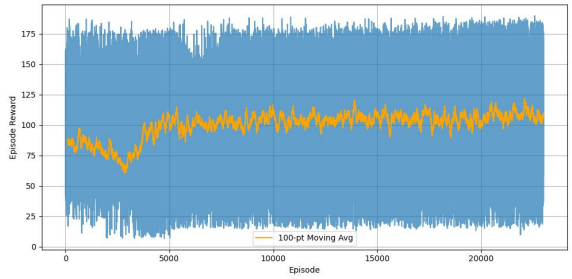


(g) Reward progress per episode (H9)

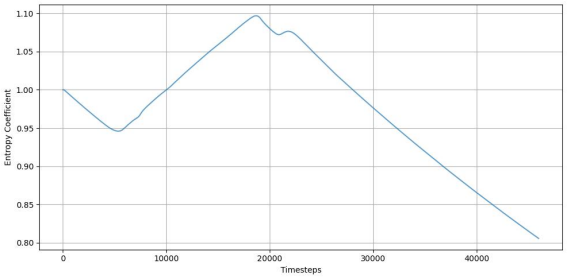


(h) Entropy coefficient per timestep (H9)

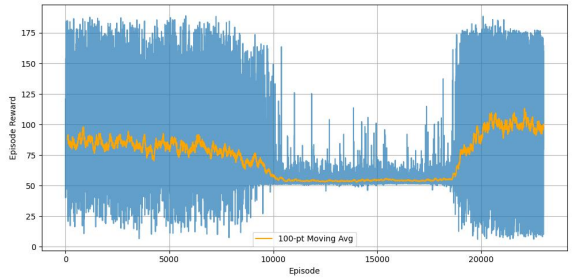
Figure B.3: Store 1 - Hyperparameterizations (H): 6-9



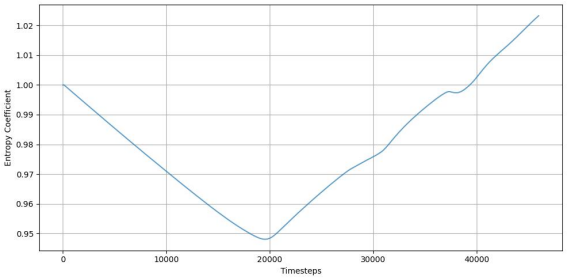
(a) Reward progress per episode (H10)



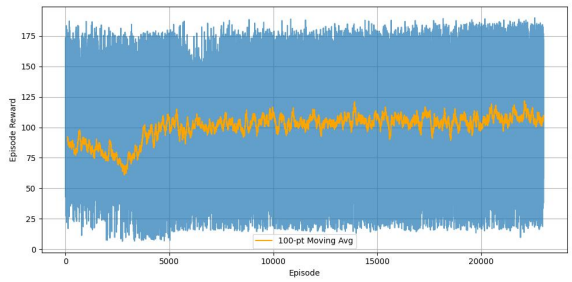
(b) Entropy coefficient per timestep (H10)



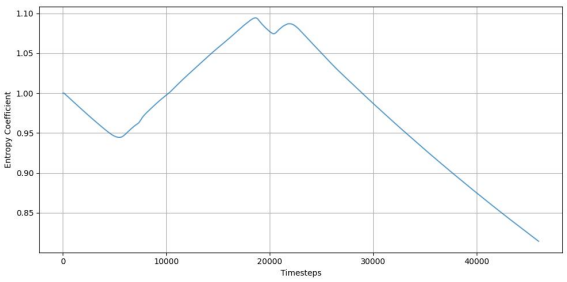
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



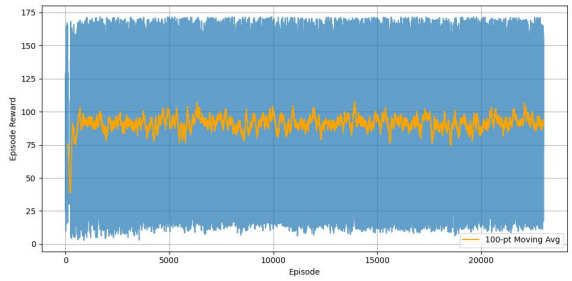
(e) Reward progress per episode (H12)



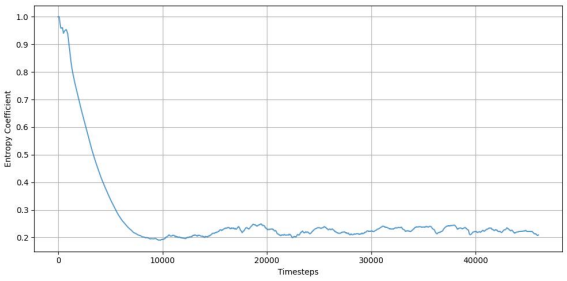
(f) Entropy coefficient per timestep (H12)

Figure B.4: Store 1 - Hyperparameterizations (H): 10-12

B.2 Store 2

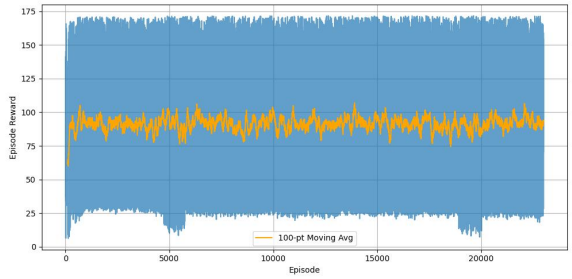


(a) Reward progress per episode (H1)

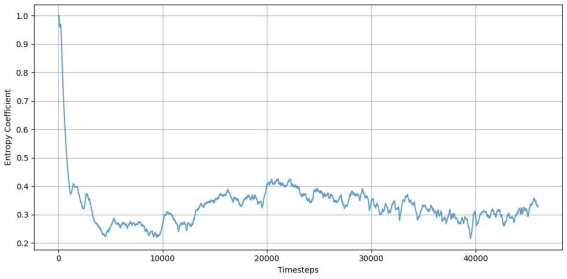


(b) Entropy coefficient per timestep (H1)

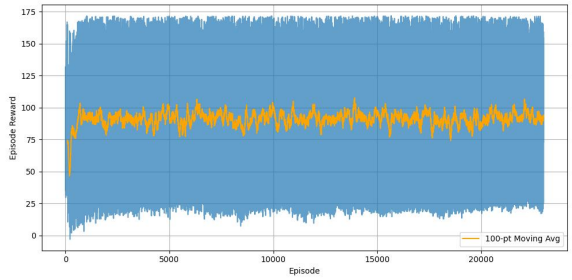
Figure B.5: Store 2 - Hyperparameterizations (H): 1-1



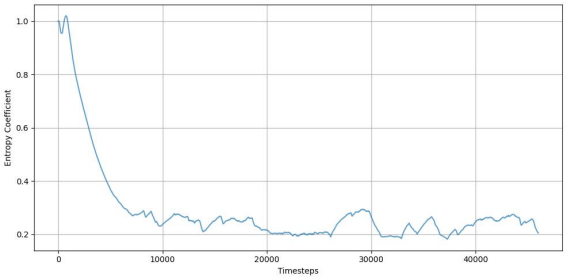
(a) Reward progress per episode (H2)



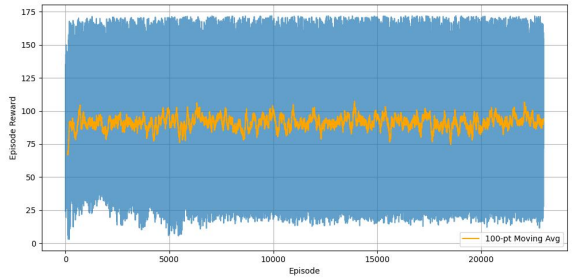
(b) Entropy coefficient per timestep (H2)



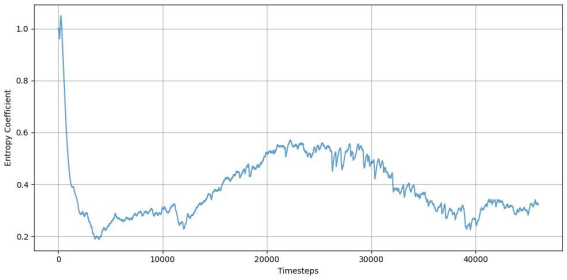
(c) Reward progress per episode (H3)



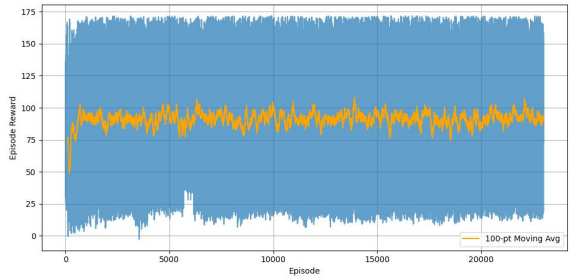
(d) Entropy coefficient per timestep (H3)



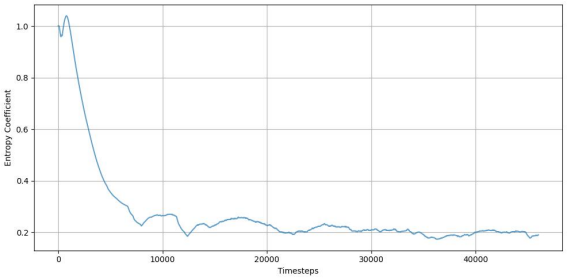
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

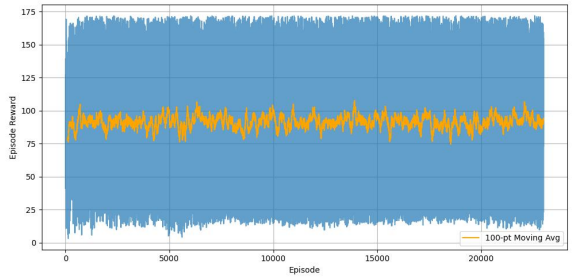


(g) Reward progress per episode (H5)

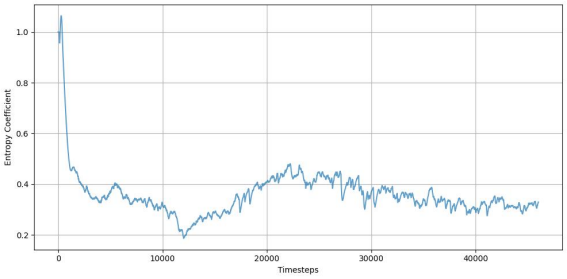


(h) Entropy coefficient per timestep (H5)

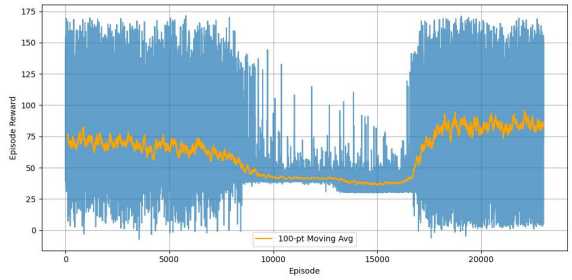
Figure B.6: Store 2 - Hyperparameterizations (H): 2-5



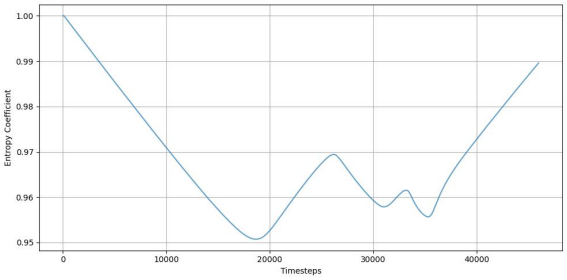
(a) Reward progress per episode (H6)



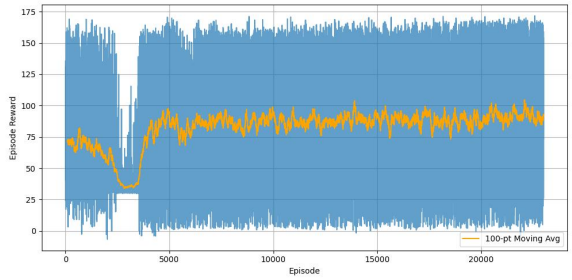
(b) Entropy coefficient per timestep (H6)



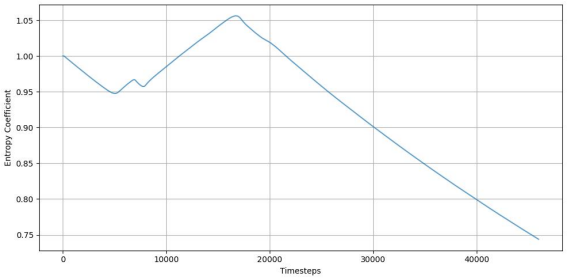
(c) Reward progress per episode (H7)



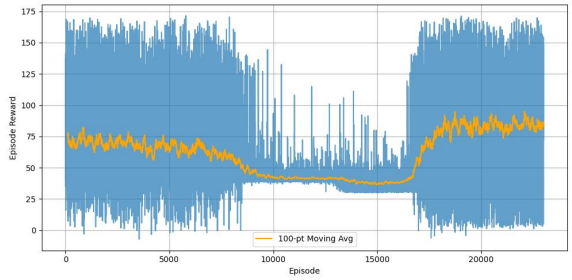
(d) Entropy coefficient per timestep (H7)



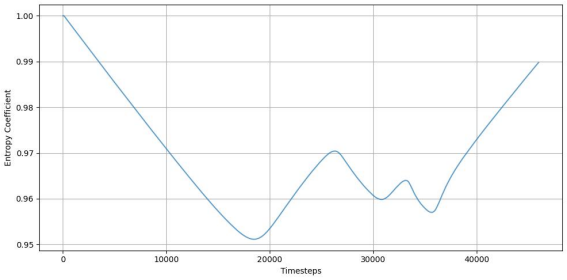
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

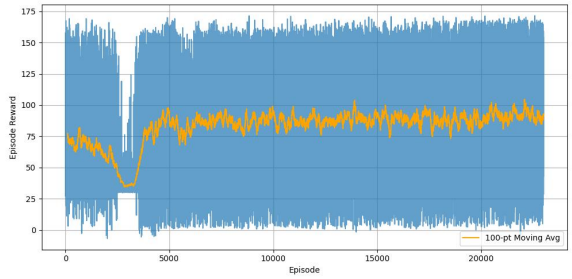


(g) Reward progress per episode (H9)

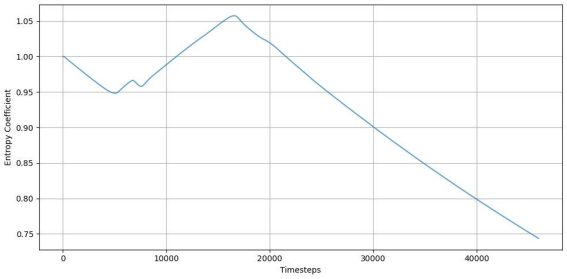


(h) Entropy coefficient per timestep (H9)

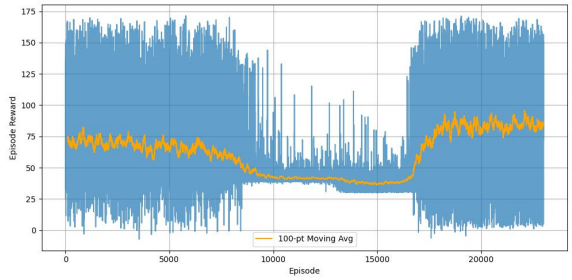
Figure B.7: Store 2 - Hyperparameterizations (H): 6-9



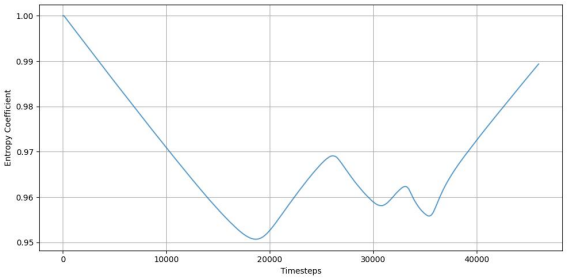
(a) Reward progress per episode (H10)



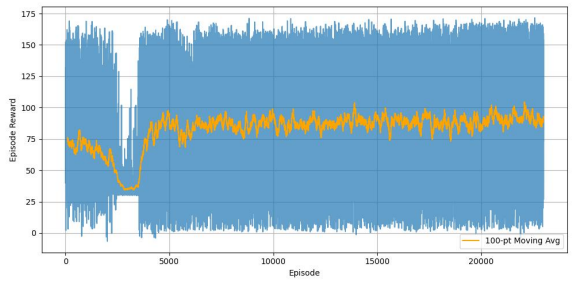
(b) Entropy coefficient per timestep (H10)



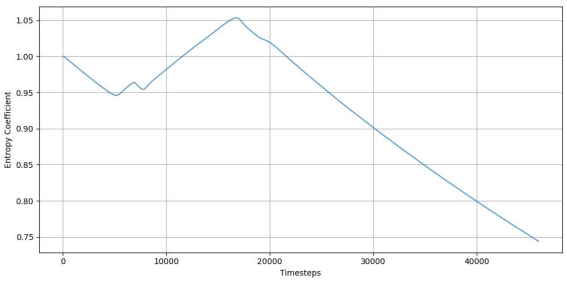
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



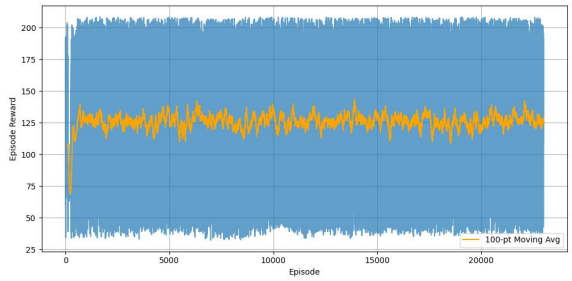
(e) Reward progress per episode (H12)



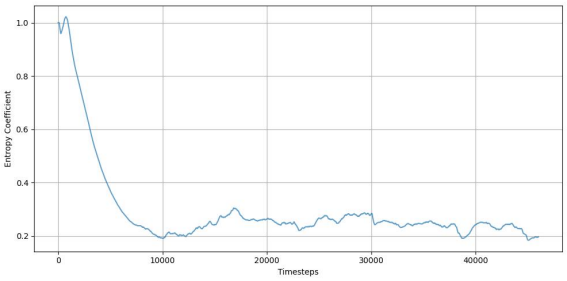
(f) Entropy coefficient per timestep (H12)

Figure B.8: Store 2 - Hyperparameterizations (H): 10-12

B.3 Store 3

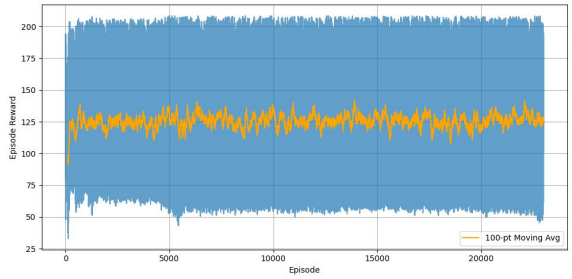


(a) Reward progress per episode (H1)

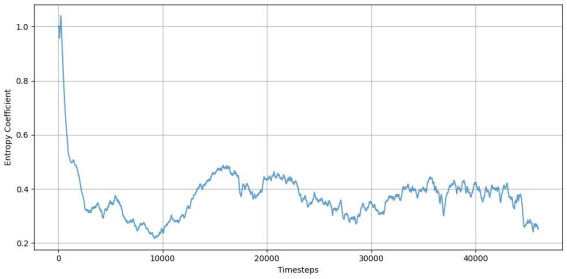


(b) Entropy coefficient per timestep (H1)

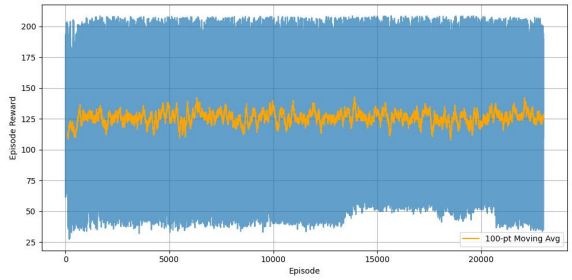
Figure B.9: Store 3 - Hyperparameterizations (H): 1-1



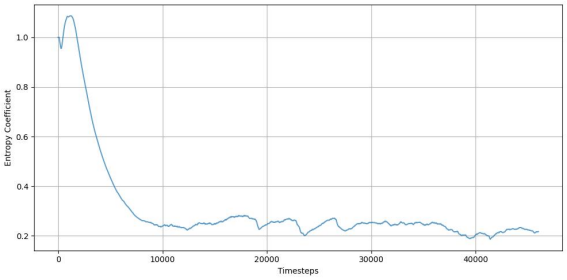
(a) Reward progress per episode (H2)



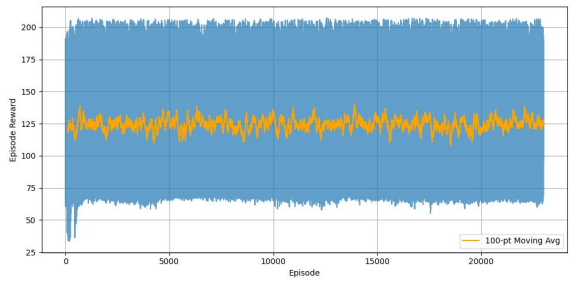
(b) Entropy coefficient per timestep (H2)



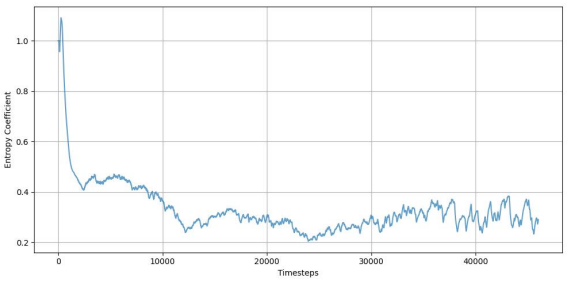
(c) Reward progress per episode (H3)



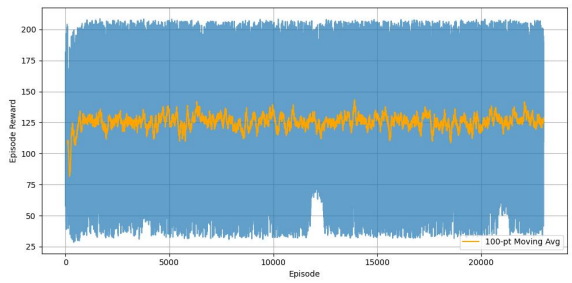
(d) Entropy coefficient per timestep (H3)



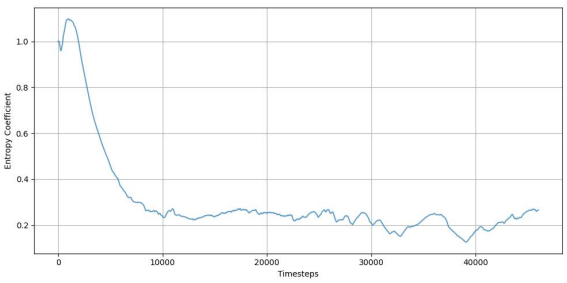
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

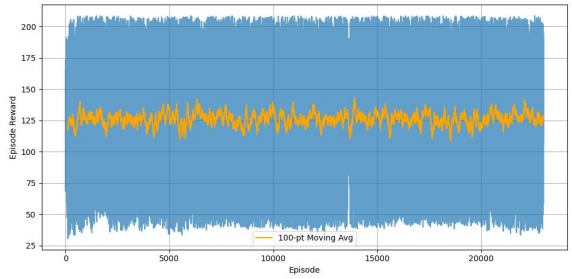


(g) Reward progress per episode (H5)

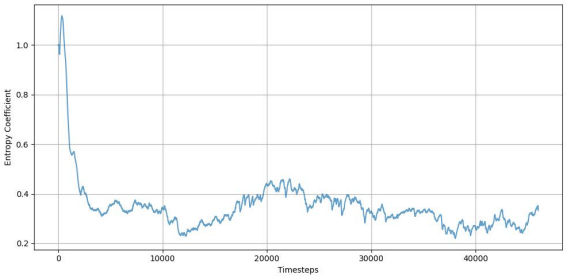


(h) Entropy coefficient per timestep (H5)

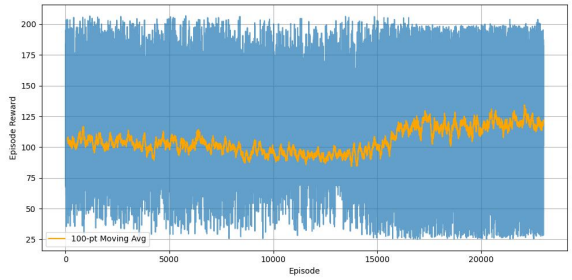
Figure B.10: Store 3 - Hyperparameterizations (H): 2-5



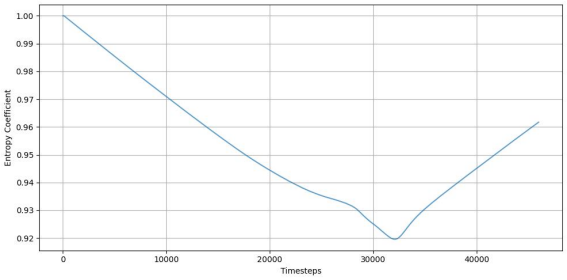
(a) Reward progress per episode (H6)



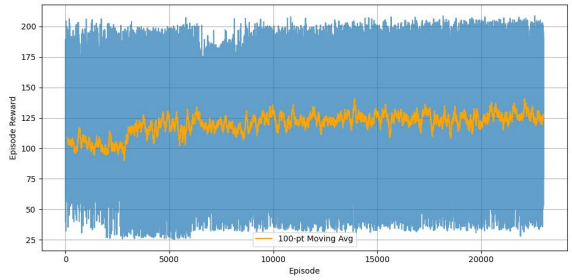
(b) Entropy coefficient per timestep (H6)



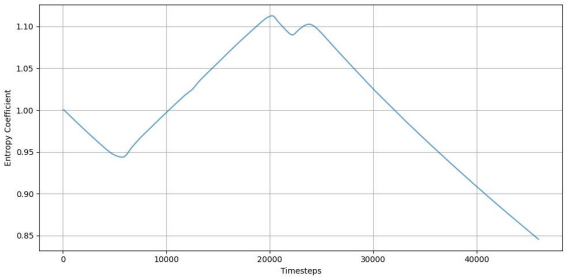
(c) Reward progress per episode (H7)



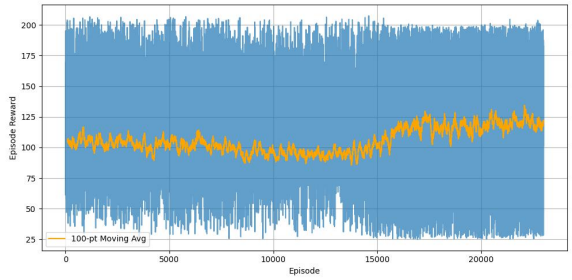
(d) Entropy coefficient per timestep (H7)



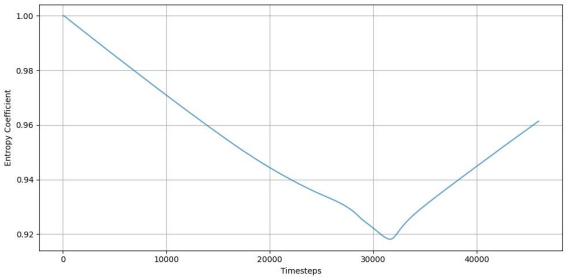
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

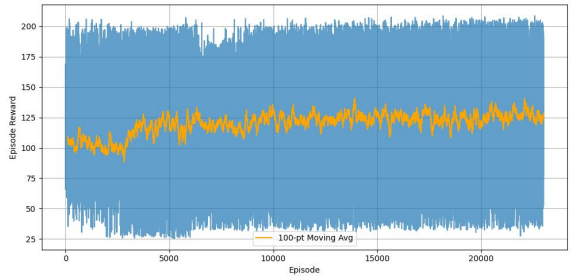


(g) Reward progress per episode (H9)

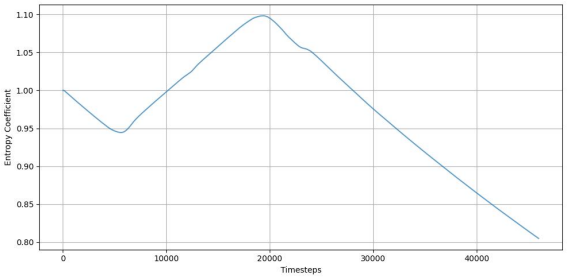


(h) Entropy coefficient per timestep (H9)

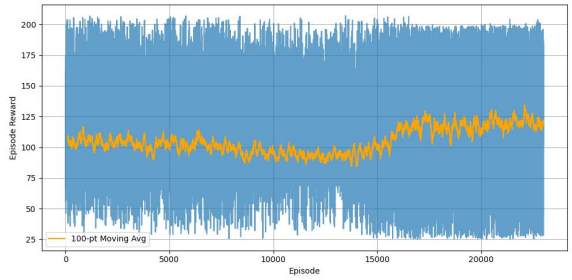
Figure B.11: Store 3 - Hyperparameterizations (H): 6-9



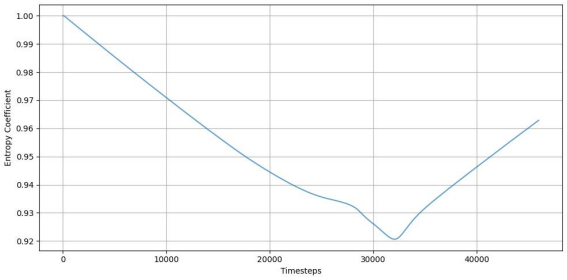
(a) Reward progress per episode (H10)



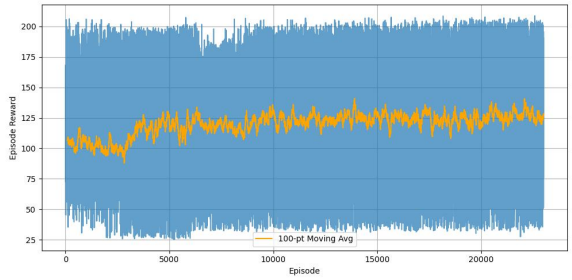
(b) Entropy coefficient per timestep (H10)



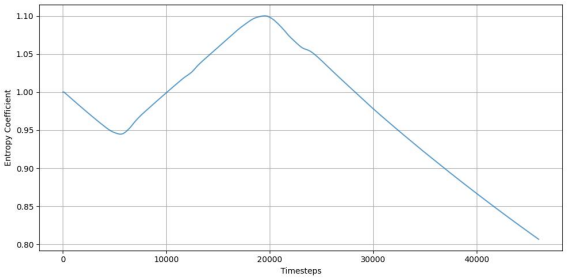
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



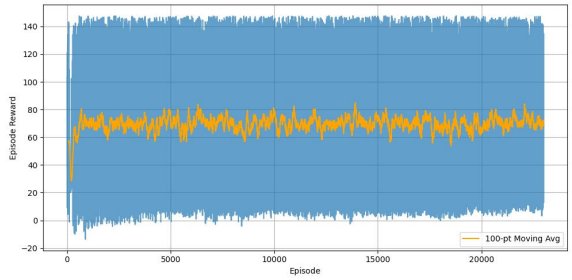
(e) Reward progress per episode (H12)



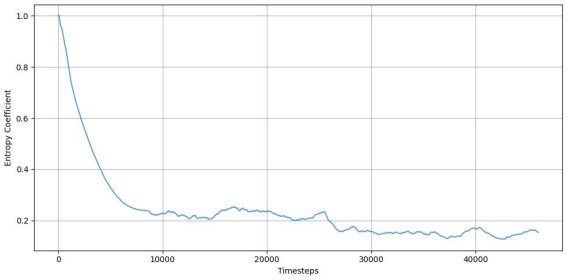
(f) Entropy coefficient per timestep (H12)

Figure B.12: Store 3 - Hyperparameterizations (H): 10-12

B.4 Store 4

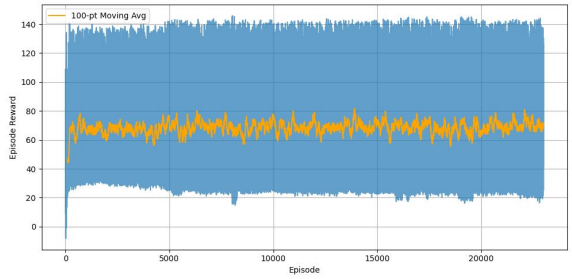


(a) Reward progress per episode (H1)

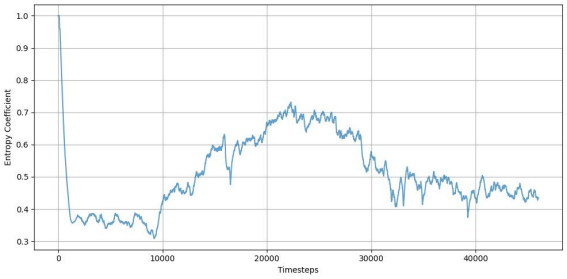


(b) Entropy coefficient per timestep (H1)

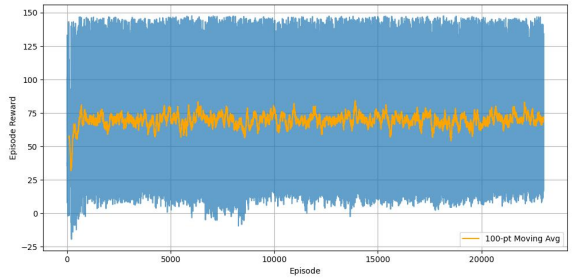
Figure B.13: Store 4 - Hyperparameterizations (H): 1-1



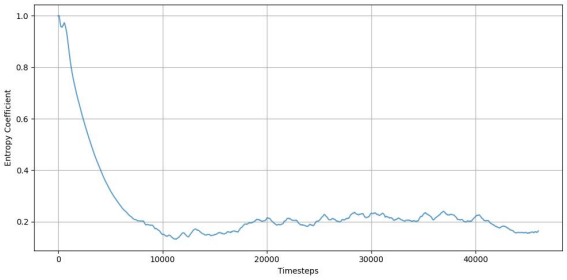
(a) Reward progress per episode (H2)



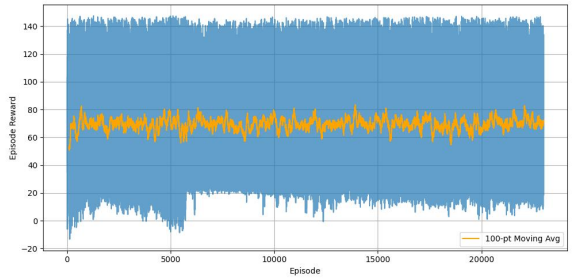
(b) Entropy coefficient per timestep (H2)



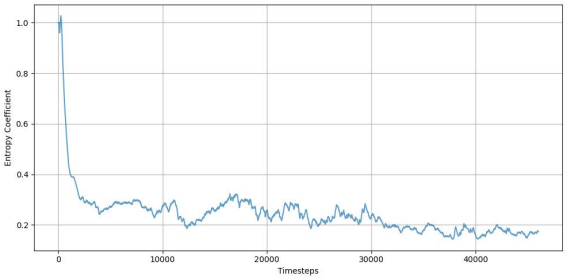
(c) Reward progress per episode (H3)



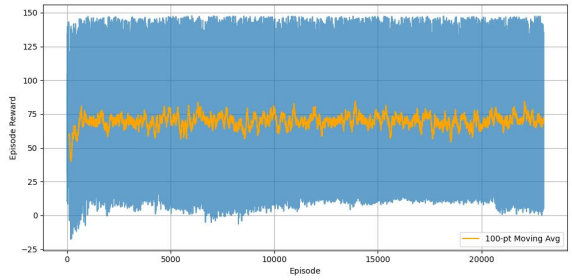
(d) Entropy coefficient per timestep (H3)



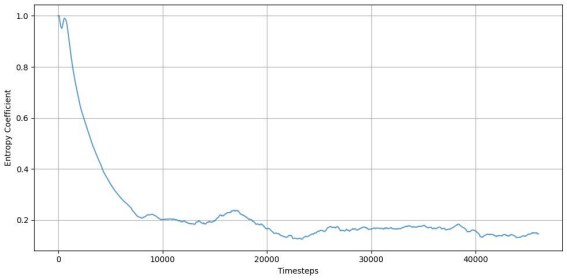
(e) Reward progress per episode (H4)



(f) Entropy coefficient per timestep (H4)

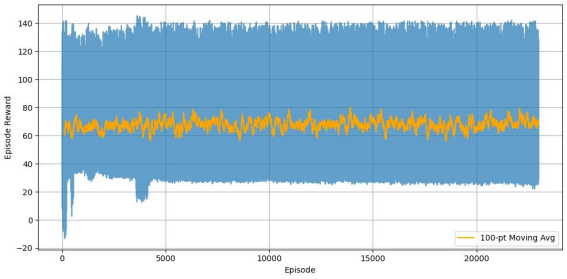


(g) Reward progress per episode (H5)

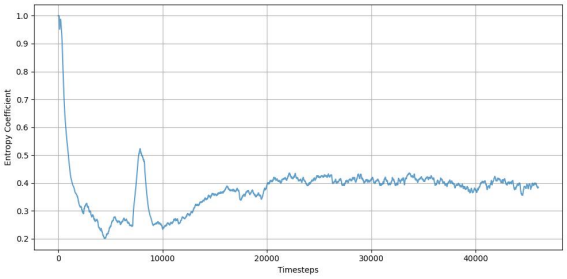


(h) Entropy coefficient per timestep (H5)

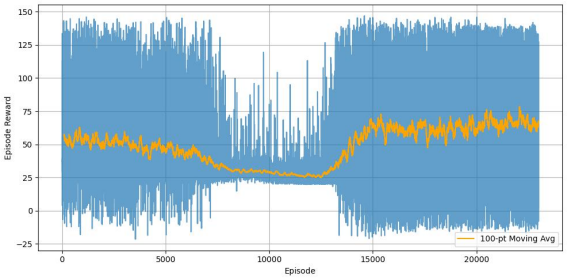
Figure B.14: Store 4 - Hyperparameterizations (H): 2-5



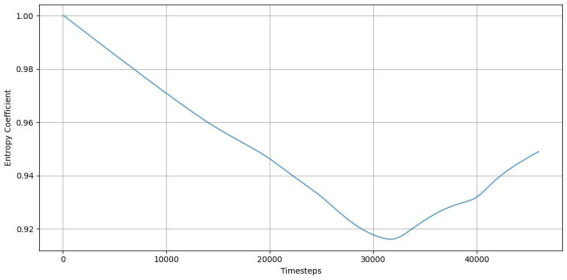
(a) Reward progress per episode (H6)



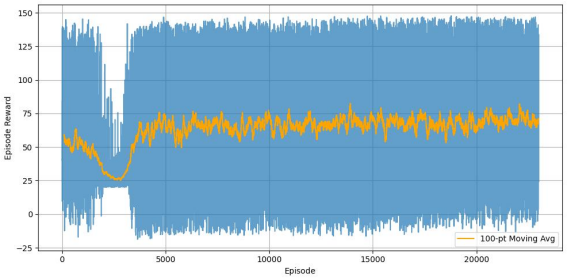
(b) Entropy coefficient per timestep (H6)



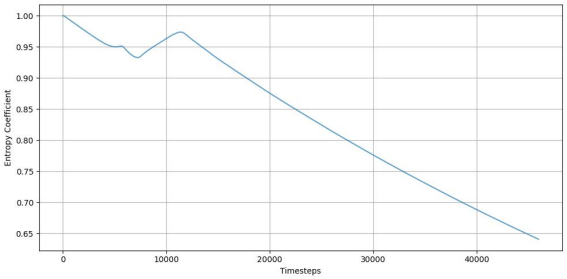
(c) Reward progress per episode (H7)



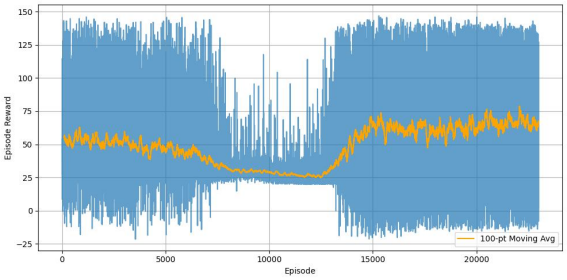
(d) Entropy coefficient per timestep (H7)



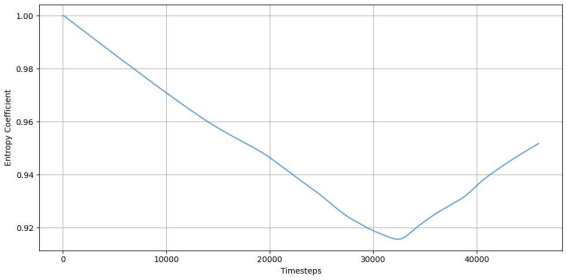
(e) Reward progress per episode (H8)



(f) Entropy coefficient per timestep (H8)

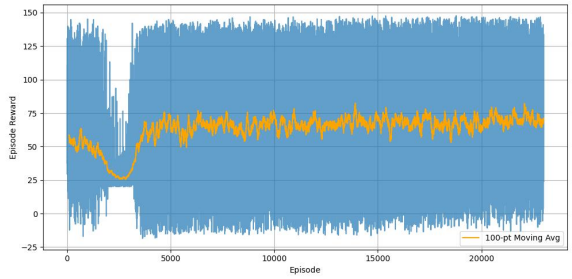


(g) Reward progress per episode (H9)

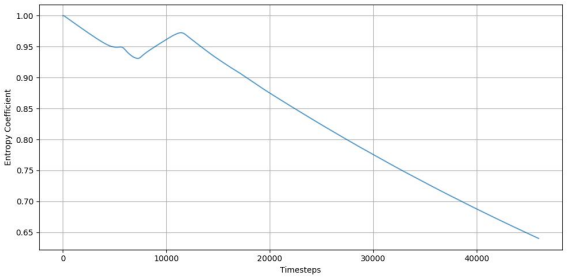


(h) Entropy coefficient per timestep (H9)

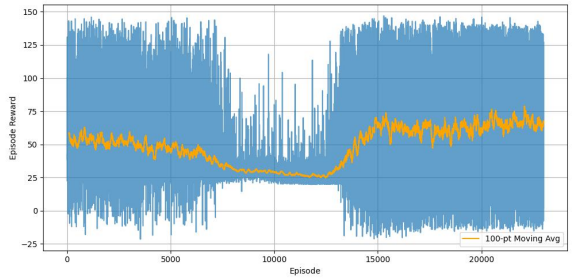
Figure B.15: Store 4 - Hyperparameterizations (H): 6-9



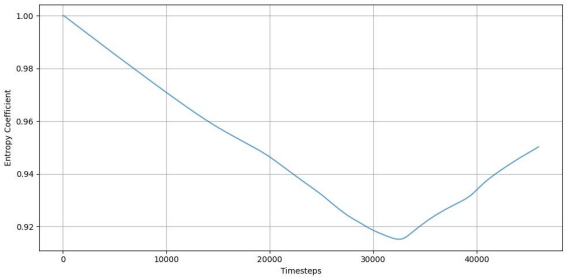
(a) Reward progress per episode (H10)



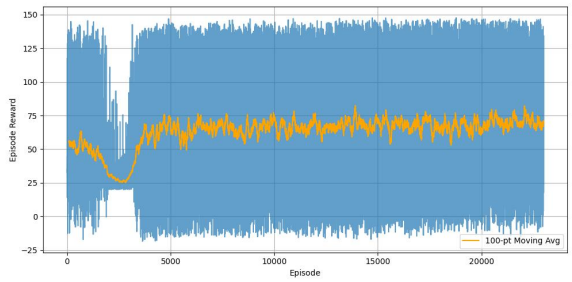
(b) Entropy coefficient per timestep (H10)



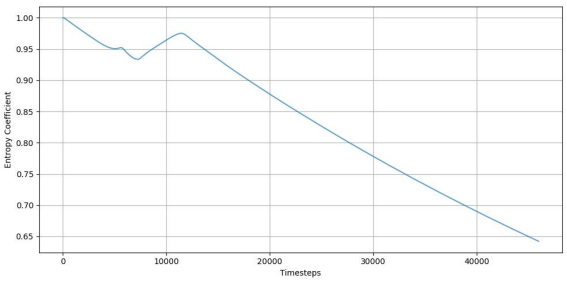
(c) Reward progress per episode (H11)



(d) Entropy coefficient per timestep (H11)



(e) Reward progress per episode (H12)



(f) Entropy coefficient per timestep (H12)

Figure B.16: Store 4 - Hyperparameterizations (H): 10-12

Appendix C

Descriptive Statistics

C.1 SAC

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	94.914040	105.160780	94.931460	105.156053	105.871230	105.787896	105.900294	105.876871	94.904494	105.183404	105.896734	105.176055
Median	94.713588	104.955486	94.730840	104.948077	105.723738	105.622710	105.780687	105.751017	94.704251	104.977680	105.797274	104.970709
Std. Dev	1.580311	1.594083	1.580403	1.590601	1.595078	1.600612	1.587050	1.588341	1.580343	1.596268	1.582424	1.595093
Max	99.082860	109.372167	99.100522	109.358431	109.978907	109.943640	109.954304	109.946552	99.073303	109.399716	109.923573	109.390932
Min	91.191960	101.410906	91.208971	101.416251	102.099517	101.998907	102.191416	102.149159	91.182531	101.428708	102.247177	101.423698
Mean+Std. Dev	96.494351	106.754863	96.511863	106.746655	107.466308	107.388507	107.487344	107.465212	96.484838	106.779672	107.479159	106.771148
Mean-Std. Dev	93.333729	103.566696	93.351056	103.565452	104.276152	104.187284	104.313244	104.288530	93.324151	103.587136	104.314310	103.580962
Median+Std. Dev	96.293899	106.549569	96.311243	106.538678	107.318816	107.223321	107.367737	107.339357	96.284594	106.573949	107.379699	106.565802
Median-Std. Dev	93.133277	103.361402	93.150437	103.357475	104.128660	104.022098	104.193637	104.162676	93.123908	103.381412	104.214850	103.375616
Rank by Mean	11	8	10	9	4	5	1	3	12	6	2	7
Rank by Median	11	8	10	9	4	5	2	3	12	6	1	7

Table C.1: Descriptive statistics per hyperparameterization: Store 1 (SAC)

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	89.363494	89.332251	89.410225	78.927438	78.836182	88.515556	89.223566	88.501943	89.334633	89.375508	88.519611	80.078872
Median	89.282998	89.206268	89.340100	78.826824	78.737488	88.334249	89.078234	88.320656	89.215858	89.263159	88.338569	79.951400
Std. Dev	1.501731	1.537974	1.508857	1.265769	1.262093	1.548331	1.546735	1.550059	1.535932	1.532776	1.550371	1.314090
Max	93.194261	93.284184	93.260922	82.111304	82.008266	92.604474	93.227208	92.600395	93.272845	93.296912	92.618132	83.398594
Min	86.093415	85.735547	86.074406	75.680272	75.596683	84.901075	85.582332	84.885125	85.759696	85.824104	84.900619	76.729759
Mean+Std. Dev	90.865225	90.870225	90.919082	80.193207	80.098274	90.063886	90.770302	90.052003	90.870565	90.908285	90.069981	81.392962
Mean-Std. Dev	87.861764	87.794278	87.901367	77.661670	77.574089	86.967225	87.676831	86.951884	87.798700	87.842732	86.969240	78.764781
Median+Std. Dev	90.784729	90.744242	90.848957	80.092593	79.999581	89.882579	90.624969	89.870715	90.751791	90.795935	89.888940	81.265490
Median-Std. Dev	87.781268	87.668295	87.831242	77.561055	77.475395	86.785918	87.531498	86.770596	87.679926	87.730383	86.788198	78.637310
Rank by Mean	3	5	1	11	12	8	6	9	4	2	7	10
Rank by Median	2	5	1	11	12	8	6	9	4	3	7	10

Table C.2: Descriptive statistics per hyperparameterization: Store 2 (SAC)

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	116.497372	116.436541	124.389348	124.381438	123.845801	123.763142	124.369607	123.828140	124.395936	124.083580	123.729592	116.620698
Median	116.287709	116.226804	124.262831	124.268349	123.629104	123.555800	124.283855	123.620195	124.253408	123.873136	123.523107	116.411209
Std. Dev	1.615521	1.613744	1.607706	1.604231	1.611321	1.611970	1.599203	1.613576	1.611727	1.617925	1.611069	1.618719
Max	120.765668	120.699360	128.499800	128.467064	128.090208	128.017833	128.433773	128.080805	128.533892	128.331676	127.981990	120.897764
Min	112.690909	112.633762	120.617212	120.652775	120.029826	119.950279	120.706588	120.006857	120.586723	120.237018	119.920847	112.808143
Mean+Std. Dev	118.112893	118.050285	125.997054	125.985668	125.457122	125.375112	125.968810	125.441716	126.007663	125.701505	125.340661	118.239418
Mean-Std. Dev	114.881851	114.822798	122.781642	122.777207	122.234480	122.151172	122.770404	122.214564	122.784210	122.465655	122.118523	115.001979
Median+Std. Dev	117.903230	117.840548	125.870537	125.872580	125.240425	125.167770	125.883058	125.233771	125.865135	125.491061	125.134176	118.029928
Median-Std. Dev	114.672188	114.613061	122.655124	122.664118	122.017784	121.943830	122.684652	122.006619	122.641681	122.255211	121.912038	114.792490
Rank by Mean	11	12	2	3	6	8	4	7	1	5	9	10
Rank by Median	11	12	3	2	6	8	1	7	4	5	9	10

Table C.3: Descriptive statistics per hyperparameterization: Store 3 (SAC)

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	63.308824	67.902250	67.991940	63.071168	67.920361	63.735060	67.934852	67.500318	67.993033	67.912819	66.526275	67.853893
Median	63.161098	67.808534	67.907582	62.922482	67.803092	63.652081	67.819016	67.396872	67.910018	67.823024	66.540752	67.740167
Std. Dev	1.419390	1.385760	1.409574	1.416688	1.427971	1.454656	1.426953	1.462501	1.408112	1.388080	1.269699	1.375760
Max	67.052594	71.460906	71.605671	66.834534	71.580662	67.591527	71.592306	71.265132	71.603224	71.477583	69.690617	71.391747
Min	60.039279	65.086482	65.004244	59.819639	64.808639	60.471965	64.829364	64.143952	65.012880	65.079703	64.148482	65.116417
Mean+Std. Dev	64.728214	69.288010	69.401514	64.487855	69.348332	65.189716	69.361805	68.962819	69.401145	69.300899	67.795974	69.229653
Mean-Std. Dev	61.889434	66.516491	66.582367	61.654480	66.492389	62.280404	66.507899	66.037818	66.584921	66.524739	65.256576	66.478133
Median+Std. Dev	64.580488	69.194294	69.317156	64.339169	69.231063	65.106737	69.245969	68.859372	69.318130	69.211105	67.810450	69.115927
Median-Std. Dev	61.741708	66.422775	66.498008	61.505794	66.375121	62.197424	66.392063	65.934371	66.501906	66.434944	65.271053	66.364407
Rank by Mean	11	6	2	12	4	10	3	8	1	5	9	7
Rank by Median	11	5	2	12	6	10	4	8	1	3	9	7

Table C.4: Descriptive statistics per hyperparameterization: Store 4 (SAC)

C.2 SAC (KDE-QDNP)

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	97.560555	107.958187	97.579398	108.498560	108.646806	108.070201	107.074849	108.621817	97.595917	107.939310	108.598517	107.951473
Median	97.499423	107.912940	97.518363	108.445383	108.628776	108.075399	107.112320	108.574797	97.535430	107.893365	108.547034	107.905871
Std. Dev	1.400246	1.414297	1.400409	1.427434	1.426777	1.416955	1.393539	1.427764	1.399480	1.413647	1.427498	1.413685
Max	100.807574	111.206885	100.826148	111.678105	111.708137	111.005875	109.911941	111.739862	100.841999	111.194156	111.737746	111.193580
Min	93.887898	104.242834	93.905773	104.769163	104.952948	104.435091	103.501768	104.901861	93.925198	104.227019	104.873230	104.236766
Mean+Std. Dev	98.960802	109.372484	98.979807	109.925993	110.073583	109.487156	108.468388	110.049581	98.995398	109.352957	110.026015	109.365157
Mean-Std. Dev	96.160309	106.543890	96.178989	107.071126	107.220030	106.653246	105.681311	107.194053	96.196437	106.525662	107.171018	106.537788
Median+Std. Dev	98.899669	109.327237	98.918772	109.872817	110.055553	109.492354	108.505859	110.002560	98.934910	109.307012	109.974532	109.319555
Median-Std. Dev	96.099176	106.498644	96.117954	107.017949	107.201999	106.658444	105.718781	107.147033	96.135949	106.479717	107.119535	106.492186
Rank by Mean	12	6	11	4	1	5	9	2	10	8	3	7
Rank by Median	12	6	11	4	1	5	9	2	10	8	3	7

Table C.5: Descriptive statistics per hyperparameterization: Store 1 (SAC (KDE-QDNP))

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	91.975033	92.123588	92.017680	86.529785	87.027662	91.161589	92.059398	91.262583	92.118421	92.169705	91.194830	87.407536
Median	91.925957	92.102042	91.967215	86.503886	87.010497	91.115600	92.108315	91.218008	92.168358	92.185192	91.146590	87.396448
Std. Dev	1.381943	1.379716	1.383427	1.301025	1.316347	1.376483	1.371801	1.377879	1.374346	1.376576	1.378007	1.326009
Max	95.023423	95.111325	95.074076	89.638581	90.172535	94.326073	94.917156	94.419470	94.990255	95.094186	94.366509	90.579475
Min	88.389859	88.576109	88.428043	83.164886	83.620216	87.572962	88.562620	87.673305	88.613970	88.641656	87.602399	83.967262
Mean+Std. Dev	93.356975	93.503304	93.401107	87.830810	88.344009	92.538072	93.431199	92.640462	93.492767	93.546281	92.572837	88.733545
Mean-Std. Dev	90.593090	90.743872	90.634254	85.228760	85.711314	89.785107	90.687597	89.884704	90.744075	90.793129	89.816824	86.081526
Median+Std. Dev	93.307900	93.481758	93.350642	87.804911	88.326844	92.492083	93.480116	92.595887	93.542704	93.561768	92.524596	88.722458
Median-Std. Dev	90.544015	90.722326	90.583789	85.202861	85.694150	89.739118	90.736513	89.840130	90.794012	90.808615	89.768583	86.070439
Rank by Mean	6	2	5	12	11	9	4	7	3	1	8	10
Rank by Median	6	4	5	12	11	9	3	7	2	1	8	10

Table C.6: Descriptive statistics per hyperparameterization: Store 2 (SAC (KDE-QDNP))

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	118.952844	118.994749	126.927676	126.790462	125.301610	126.403000	126.602872	126.430161	126.414917	126.625263	126.424856	119.000418
Median	118.895376	118.936690	126.905819	126.754432	125.330507	126.361421	126.561424	126.389994	126.368119	126.659560	126.384153	118.942570
Std. Dev	1.426470	1.427466	1.443317	1.438468	1.414173	1.428944	1.430552	1.428906	1.420955	1.442561	1.428323	1.427500
Max	122.264514	122.308992	130.011175	130.039281	128.183329	129.689955	129.892143	129.717937	129.723341	129.658412	129.717250	122.314380
Min	115.210970	115.250555	123.182958	123.020958	121.672739	122.663621	122.847381	122.690428	122.681140	122.915698	122.686430	115.255896
Mean+Std. Dev	120.379314	120.422215	128.370993	128.228930	126.715782	127.831944	128.033424	127.859067	127.835871	128.067824	127.853179	120.427918
Mean-Std. Dev	117.526374	117.567283	125.484359	125.351994	123.887437	124.974057	125.172320	125.001255	124.993962	125.182702	124.996534	117.572918
Median+Std. Dev	120.321847	120.364156	128.349136	128.192900	126.744679	127.790365	127.991976	127.818901	127.789074	128.102121	127.812475	120.370070
Median-Std. Dev	117.468906	117.509225	125.462502	125.315965	123.916334	124.932477	125.130872	124.961088	124.947165	125.216999	124.955830	117.515070
Rank by Mean	12	11	1	2	9	8	4	5	7	3	6	10
Rank by Median	12	11	1	2	9	8	4	5	7	3	6	10

Table C.7: Descriptive statistics per hyperparameterization: Store 3 (SAC (KDE-QDNP))

Hyperparameterization	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12
Mean	65.173086	70.188102	70.415985	65.675607	68.947855	65.664629	70.450820	70.395359	70.173253	70.063359	69.609051	70.209196
Median	65.152903	70.202278	70.452941	65.676691	69.020764	65.635785	70.504441	70.407746	70.174917	70.056503	69.647746	70.226581
Std. Dev	1.251242	1.307308	1.282185	1.258459	1.173657	1.256127	1.294996	1.268614	1.307999	1.311513	1.208535	1.306753
Max	68.116344	73.013481	73.046158	68.602083	71.336411	68.575800	73.183923	72.915970	73.007739	72.925833	72.081011	73.029867
Min	61.916669	66.868347	67.181262	62.357266	66.045447	62.358331	67.190121	67.189701	66.844140	66.719353	66.571009	66.892866
Mean+Std. Dev	66.424328	71.495410	71.698170	66.934066	70.121512	66.920756	71.745816	71.663973	71.481252	71.374872	70.817586	71.515949
Mean-Std. Dev	63.921844	68.880794	69.133799	64.417149	67.774197	64.408503	69.155824	69.126744	68.865254	68.751846	68.400516	68.902444
Median+Std. Dev	66.404145	71.509586	71.735127	66.935150	70.194421	66.891912	71.799438	71.676361	71.482916	71.368016	70.856281	71.533334
Median-Std. Dev	63.901661	68.894970	69.170756	64.418233	67.847106	64.379659	69.209445	69.139132	68.866918	68.744990	68.439211	68.919828
Rank by Mean	12	5	2	10	9	11	1	3	6	7	8	4
Rank by Median	12	5	2	10	9	11	1	3	6	7	8	4

Table C.8: Descriptive statistics per hyperparameterization: Store 4 (SAC (KDE-QDNP))

Appendix D

Statistical Testing Results: t-test

D.1 SAC

Store 1	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	1	0	1	1	1	1	1	0	1	1	1
H2	1	-	1	0	1	1	1	1	1	0	1	0
H3	0	1	-	1	1	1	1	1	0	1	1	1
H4	1	0	1	-	1	1	1	1	1	0	1	0
H5	1	1	1	1	-	0	0	0	1	1	0	1
H6	1	1	1	1	0	-	0	0	1	1	0	1
H7	1	1	1	1	0	0	-	0	1	1	0	1
H8	1	1	1	1	0	0	0	-	1	1	0	1
H9	0	1	0	1	1	1	1	1	-	1	1	1
H10	1	0	1	0	1	1	1	1	1	-	1	0
H11	1	1	1	1	0	0	0	0	1	1	-	1
H12	1	0	1	0	1	1	1	1	1	0	1	-

Store 2	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	0	0	1	1	1	0	1	0	0	1	1
H2	0	-	0	1	1	1	0	1	0	0	1	1
H3	0	0	-	1	1	1	0	1	0	0	1	1
H4	1	1	1	-	0	1	1	1	1	1	1	1
H5	1	1	1	0	-	1	1	1	1	1	1	1
H6	1	1	1	1	1	-	1	0	1	1	0	1
H7	0	0	0	1	1	1	-	1	0	0	1	1
H8	1	1	1	1	1	0	1	-	1	1	0	1
H9	0	0	0	1	1	1	0	1	-	0	1	1
H10	0	0	0	1	1	1	0	1	0	-	1	1
H11	1	1	1	1	1	0	1	0	1	1	-	1
H12	1	1	1	1	1	1	1	1	1	1	1	-

Store 3	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	0	1	1	1	1	1	1	1	1	1	0
H2	0	-	1	1	1	1	1	1	1	1	1	0
H3	1	1	-	0	1	1	0	1	0	0	1	1
H4	1	1	0	-	1	1	0	1	0	0	1	1
H5	1	1	1	1	-	0	1	0	1	0	0	1
H6	1	1	1	1	0	-	1	0	1	0	0	1
H7	1	1	0	0	1	1	-	1	0	0	1	1
H8	1	1	1	1	0	0	1	-	1	0	0	1
H9	1	1	0	0	1	1	0	1	-	0	1	1
H10	1	1	0	0	0	0	0	0	0	-	0	1
H11	1	1	1	1	0	0	1	0	1	0	-	1
H12	0	0	1	1	1	1	1	1	1	1	1	-

Store 4	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	1	1	0	1	1	1	1	1	1	1	1
H2	1	-	0	1	0	1	0	1	0	0	1	0
H3	1	0	-	1	0	1	0	1	0	0	1	0
H4	0	1	1	-	1	1	1	1	1	1	1	1
H5	1	0	0	1	-	1	0	1	0	0	1	0
H6	1	1	1	1	1	-	1	1	1	1	1	1
H7	1	0	0	1	0	1	-	1	0	0	1	0
H8	1	1	1	1	1	1	1	-	1	1	1	0
H9	1	0	0	1	0	1	0	1	-	0	1	0
H10	1	0	0	1	0	1	0	1	0	-	1	0
H11	1	1	1	1	1	1	1	1	1	1	-	1
H12	1	0	0	1	0	1	0	0	0	0	1	-

Figure D.1: Pairwise t-test results (1 = Significant difference: p-value<0.05): Stores 1-4 (SAC)

D.2 SAC (KDE-QDNP)

Store 1	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	1	0	1	1	1	1	1	0	1	1	1
H2	1	-	1	1	1	0	1	1	1	0	1	0
H3	0	1	-	1	1	1	1	1	0	1	1	1
H4	1	1	1	-	0	1	1	0	1	1	0	1
H5	1	1	1	0	-	1	1	0	1	1	0	1
H6	1	0	1	1	1	-	1	1	1	0	1	0
H7	1	1	1	1	1	1	-	1	1	1	1	1
H8	1	1	1	0	0	1	1	-	1	1	0	1
H9	0	1	0	1	1	1	1	1	-	1	1	1
H10	1	0	1	1	1	0	1	1	1	-	1	0
H11	1	1	1	0	0	1	1	0	1	1	-	1
H12	1	0	1	1	1	0	1	1	1	0	1	-

Store 2	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	0	0	1	1	1	0	1	0	0	1	1
H2	0	-	0	1	1	1	0	1	0	0	1	1
H3	0	0	-	1	1	1	0	1	0	0	1	1
H4	1	1	1	-	1	1	1	1	1	1	1	1
H5	1	1	1	1	-	1	1	1	1	1	1	1
H6	1	1	1	1	1	-	1	0	1	1	0	1
H7	0	0	0	1	1	1	-	1	0	0	1	1
H8	1	1	1	1	1	0	1	-	1	1	0	1
H9	0	0	0	1	1	1	0	1	-	0	1	1
H10	0	0	0	1	1	1	0	1	0	-	1	1
H11	1	1	1	1	1	0	1	0	1	1	-	1
H12	1	1	1	1	1	1	1	1	1	1	1	-

Store 3	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	0	1	1	1	1	1	1	1	1	1	0
H2	0	-	1	1	1	1	1	1	1	1	1	0
H3	1	1	-	0	1	1	0	1	1	0	1	1
H4	1	1	0	-	1	0	0	0	0	0	0	1
H5	1	1	1	1	-	1	1	1	1	1	1	1
H6	1	1	1	0	1	-	0	0	0	0	0	1
H7	1	1	0	0	1	0	-	0	0	0	0	1
H8	1	1	1	0	1	0	0	-	0	0	0	1
H9	1	1	1	0	1	0	0	0	-	0	0	1
H10	1	1	0	0	1	0	0	0	0	-	0	1
H11	1	1	1	0	1	0	0	0	0	0	-	1
H12	0	0	1	1	1	1	1	1	1	1	1	-

Store 4	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	1H2
H1	-	1	1	1	1	1	1	1	1	1	1	1
H2	1	-	0	1	1	1	0	0	0	0	1	0
H3	1	0	-	1	1	1	0	0	0	0	1	0
H4	1	1	1	-	1	0	1	1	1	1	1	1
H5	1	1	1	1	-	1	1	1	1	1	1	1
H6	1	1	1	0	1	-	1	1	1	1	1	1
H7	1	0	0	1	1	1	-	0	0	1	1	0
H8	1	0	0	1	1	1	0	-	0	0	1	0
H9	1	0	0	1	1	1	0	0	-	0	1	0
H10	1	0	0	1	1	1	1	0	0	-	1	0
H11	1	1	1	1	1	1	1	1	1	1	-	1
H12	1	0	0	1	1	1	0	0	0	0	1	-

Figure D.2: Pairwise t-test results (1 = Significant difference: p-value<0.05): Stores 1-4 (SAC (KDE-QDNP))

Appendix E

Boxplots

E.1 SAC

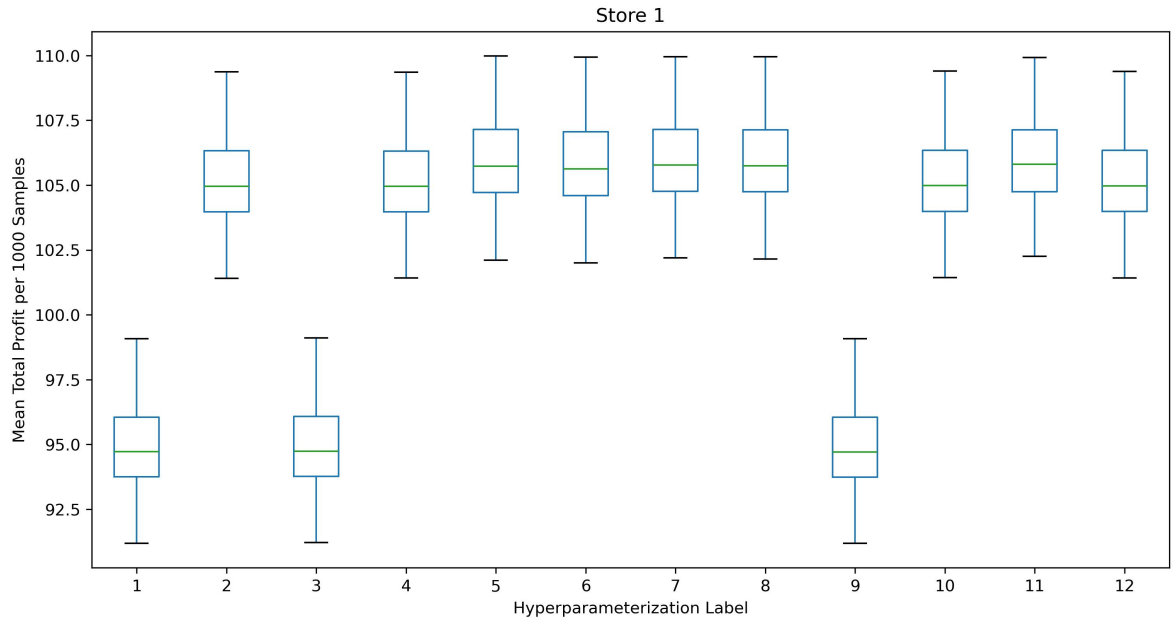


Figure E.1: Mean Return per hyperparameterization (Store 1)

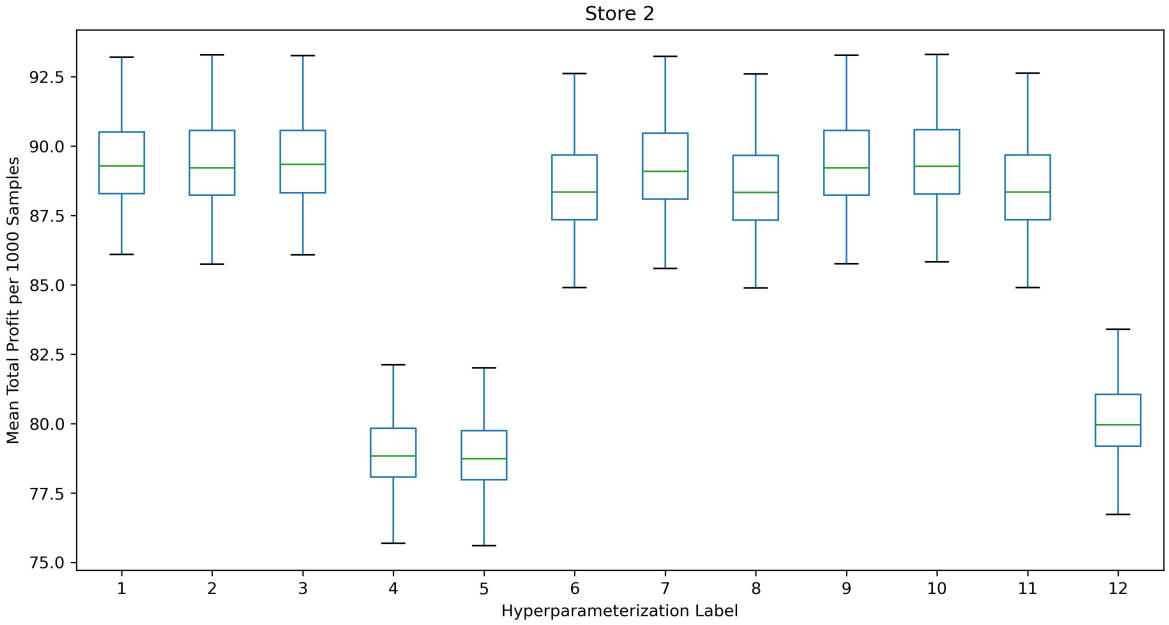


Figure E.2: Mean Return per hyperparameterization (Store 2)

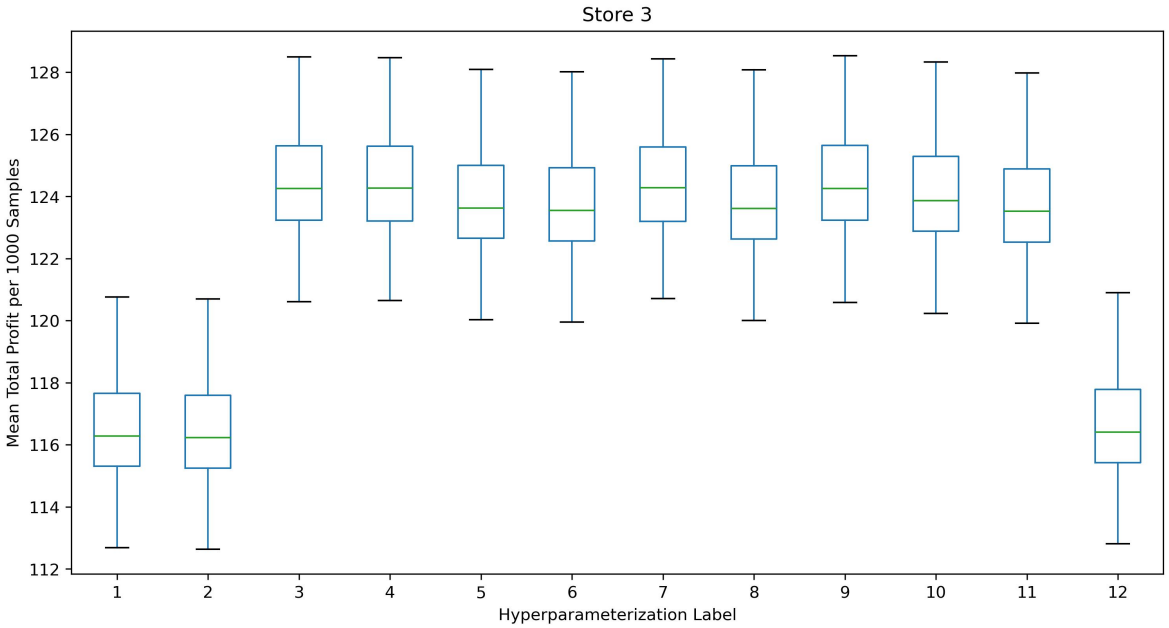


Figure E.3: Mean Return per hyperparameterization (Store 3)

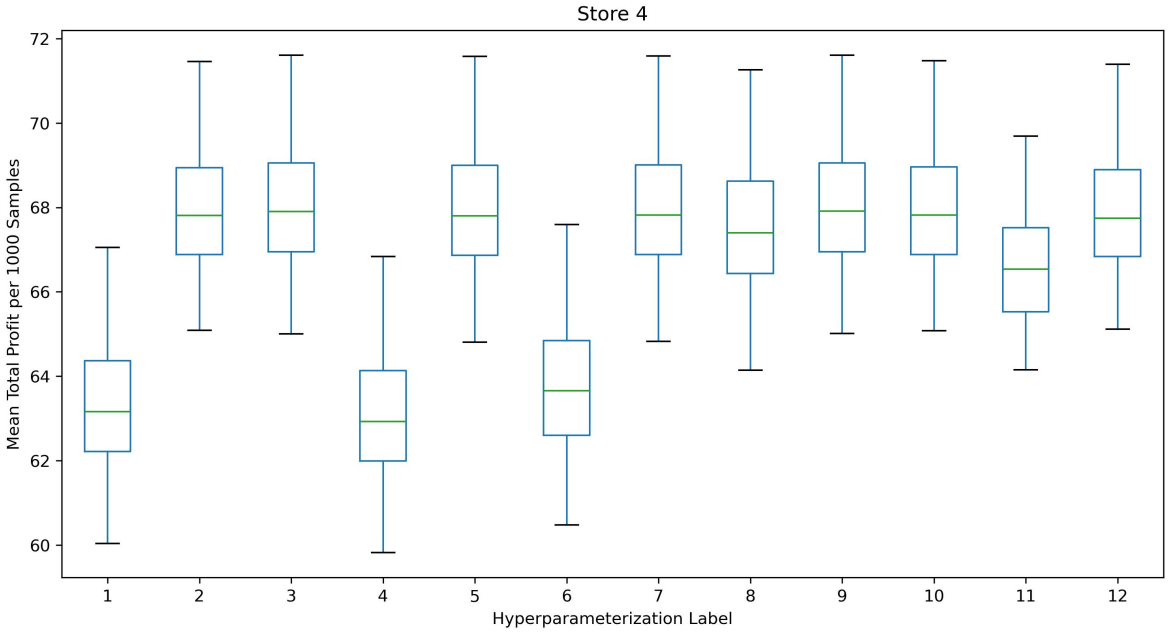


Figure E.4: Mean Return per hyperparameterization (Store 4)

E.2 SAC (KDE-QDNP)

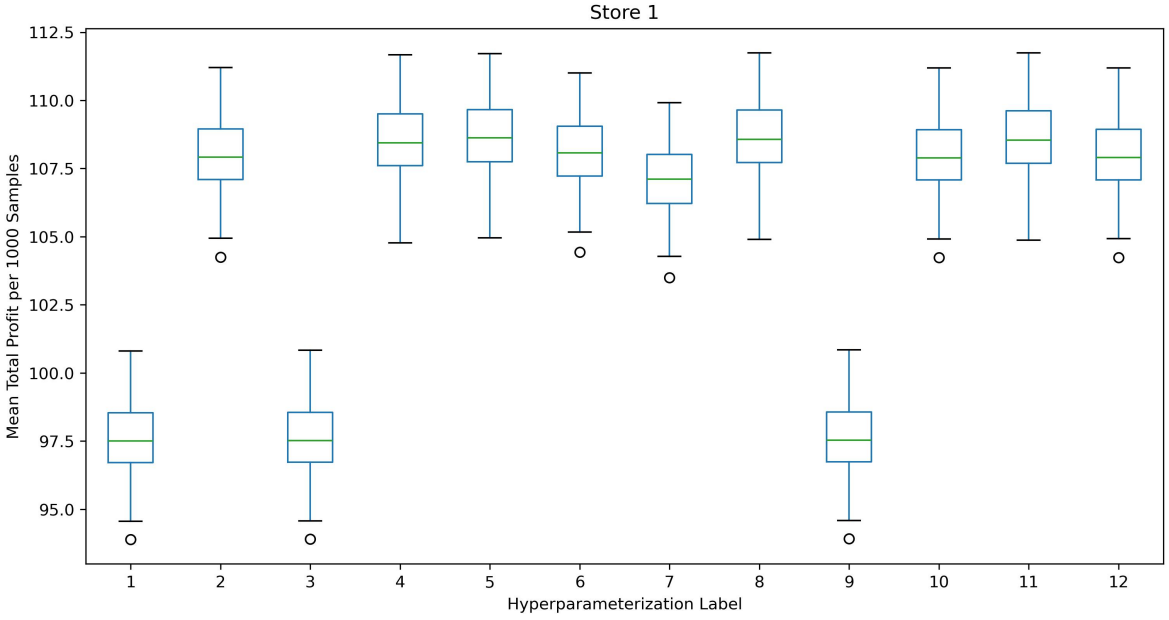


Figure E.5: Mean Return per hyperparameterization (Store 1)

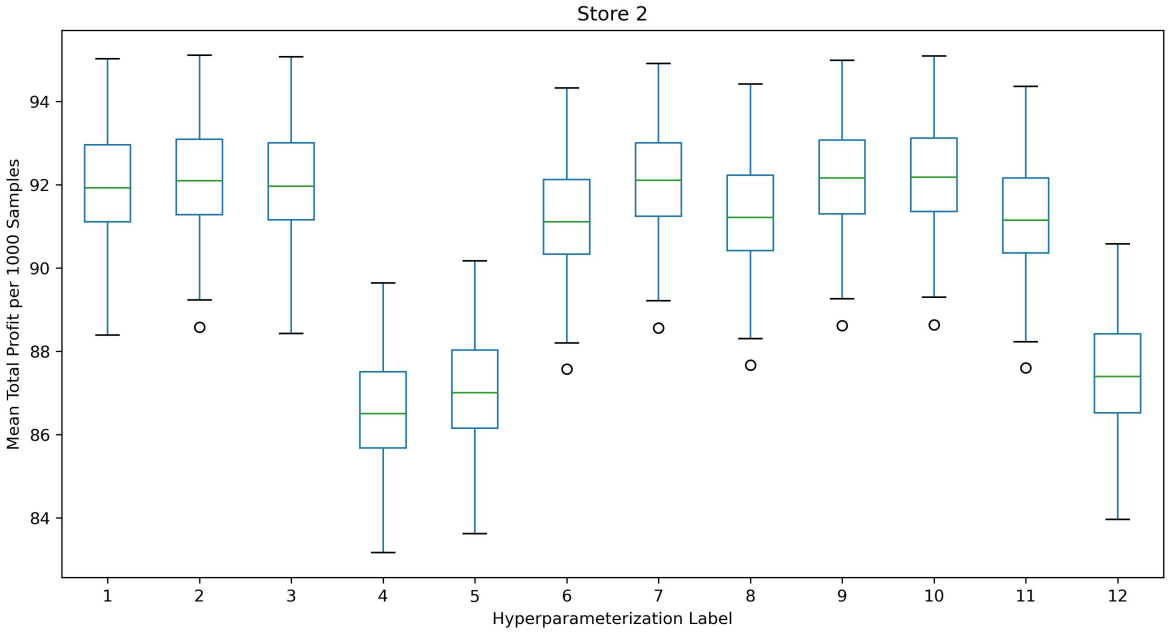


Figure E.6: Mean Return per hyperparameterization (Store 2)

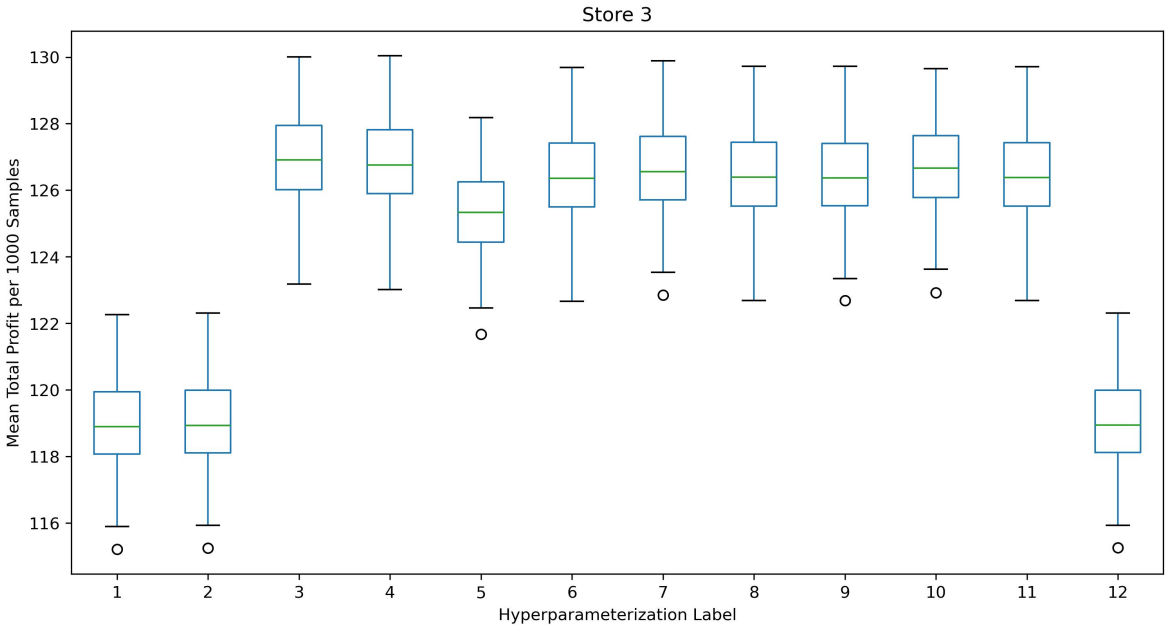


Figure E.7: Mean Return per hyperparameterization (Store 3)

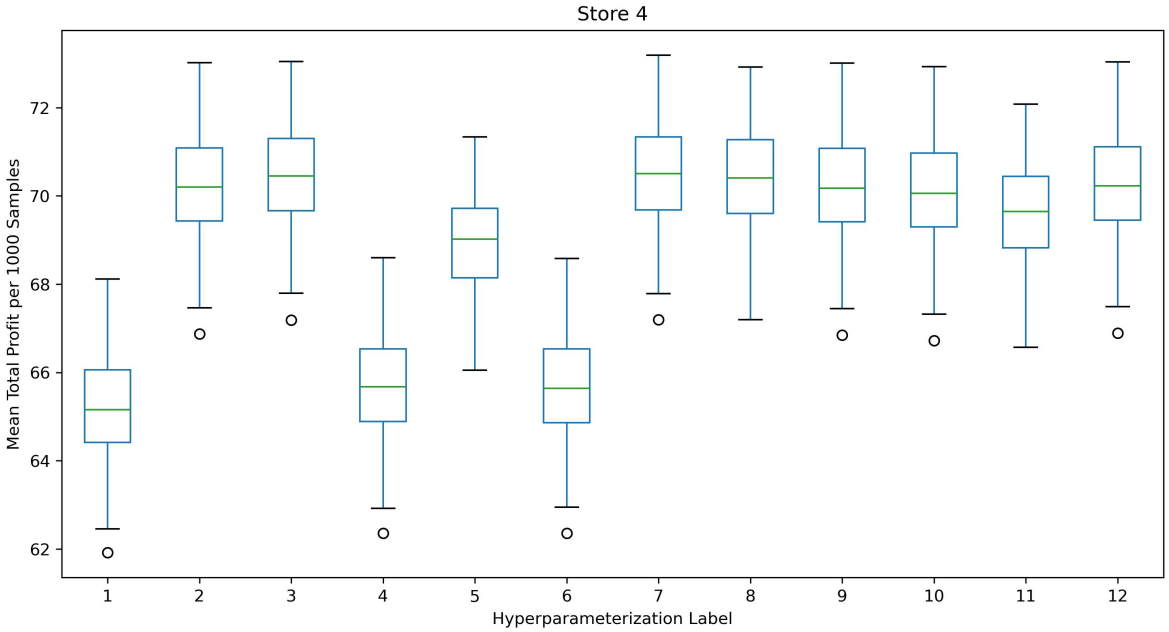


Figure E.8: Mean Return per hyperparameterization (Store 4)

Appendix F

Runtime Tables

Hyperparameterization	Store 1 (PC1)	Store 2 (PC2)	Store 3 (PC3)	Store 4 (PC4)
H1	1026.68	791.20	1890.31	2468.13
H2	3395.56	2712.91	1883.10	8779.58
H3	1044.78	798.66	6910.44	2502.00
H4	3605.81	787.09	1918.75	2455.69
H5	1038.21	836.22	6910.14	9222.93
H6	3547.17	2620.56	6567.51	2492.48
H7	3650.55	2697.44	1908.35	2531.45
H8	1029.08	2642.24	6579.75	9233.74
H9	1017.06	2713.69	1923.09	2511.62
H10	3361.58	807.55	6938.04	8787.53
H11	1020.69	2619.64	6540.50	9220.67
H12	3355.51	816.31	1884.16	8807.38

Table F.1: SAC runtime (seconds), per problem instance

Hyperparameterization	Store 1 (PC1)	Store 2 (PC2)	Store 3 (PC3)	Store 4 (PC4)
H1	1088.53	860.69	1965.73	2591.83
H2	3419.20	2845.71	1963.73	8857.98
H3	1092.43	873.60	6949.44	2623.09
H4	3610.73	944.05	1969.11	2590.28
H5	1080.00	825.18	7013.16	9296.46
H6	3680.19	2659.64	6658.05	2595.47
H7	3644.58	2822.56	1997.88	2640.70
H8	1078.65	2612.62	6595.33	9317.48
H9	1084.40	2857.82	1979.68	2620.36
H10	3403.30	877.34	6996.39	8948.94
H11	1117.32	2672.80	6615.26	9385.38
H12	3354.88	849.54	1968.34	9012.11

Table F.2: SAC (KDE-QDNP) runtime (seconds), per problem instance

Appendix G

Zero-Shot Policy Transfer: Complete Results Tables

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	74.51	1.47	-0.98	1.57	1.18	204.28	0.82	0.18
+10 %	141.38	1.57	-1.36	1.73	1.31	246.65	0.89	0.11
-20 %	46.16	1.64	-0.67	1.49	1.12	192.90	0.75	0.25
+20 %	180.76	1.62	-1.59	1.81	1.37	267.87	0.92	0.08
-30	22.16	1.65	-0.50	1.41	1.05	174.63	0.70	0.30
+30 %	223.22	1.59	-1.89	1.89	1.42	285.54	0.95	0.05

(a) SAC Pre-trained on original Store 1 parameters

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	74.63	1.15	-1.27	1.56	1.16	190.41	0.84	0.16
+10 %	141.81	1.82	-1.12	1.73	1.33	258.40	0.87	0.12
-20 %	48.52	1.04	-1.26	1.48	1.08	166.67	0.78	0.22
+20 %	182.10	1.83	-1.24	1.81	1.39	277.75	0.92	0.08
-30 %	26.41	0.29	-1.34	1.37	0.99	117.60	0.82	0.18
+30 %	226.59	1.90	-1.36	1.90	1.46	300.04	0.95	0.05

(b) SAC Trained with the perturbed parameter

Table G.1: Comparison of SAC performance with original and perturbed parameter training: a

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	142.56	1.60	-1.24	1.78	1.33	240.92	0.87	0.13
+10 %	77.04	1.62	-1.02	1.54	1.18	218.28	0.82	0.18
-20 %	189.97	1.58	-1.35	1.94	1.43	251.71	0.89	0.11
+20 %	54.64	1.46	-0.99	1.46	1.11	199.35	0.83	0.17
-30 %	252.78	1.60	-1.48	2.14	1.56	264.14	0.90	0.10
+30 %	36.12	1.54	-0.87	1.38	1.07	191.24	0.79	0.21

(a) SAC Pre-trained on original Store 1 parameters

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	142.92	1.80	-1.02	1.78	1.35	249.91	0.86	0.14
+10 %	77.46	1.34	-1.25	1.54	1.16	206.05	0.85	0.15
-20 %	191.67	1.91	-0.90	1.94	1.47	266.86	0.88	0.12
+20 %	55.37	1.04	-1.42	1.45	1.09	181.31	0.84	0.16
-30 %	256.89	1.98	-0.84	2.14	1.63	282.03	0.91	0.09
+30 %	38.12	0.67	-1.50	1.37	1.03	154.38	0.86	0.14

(b) SAC Trained with the perturbed parameter

Table G.2: Comparison of SAC performance with original and perturbed parameter training: b

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	94.18	1.60	-1.11	1.62	1.23	221.24	0.84	0.16
+10 %	117.97	1.44	-1.30	1.68	1.25	229.70	0.87	0.13
-20 %	83.02	1.58	-1.03	1.58	1.22	212.66	0.84	0.16
+20 %	131.01	1.57	-1.24	1.72	1.27	243.55	0.87	0.13
-30 %	72.58	1.43	-1.07	1.55	1.20	197.99	0.85	0.15
+30 %	144.29	1.54	-1.31	1.75	1.28	249.99	0.88	0.12

(a) SAC Pre-trained on original Store 1 parameters

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	94.20	1.48	-1.18	1.61	1.23	215.75	0.85	0.15
+10 %	118.17	1.70	-1.08	1.68	1.27	241.40	0.85	0.15
-20 %	83.09	1.34	-1.22	1.58	1.21	201.69	0.86	0.14
+20 %	131.03	1.57	-1.18	1.72	1.27	243.62	0.88	0.12
-30 %	72.49	1.18	-1.29	1.54	1.18	186.88	0.87	0.13
+30 %	144.59	1.81	-1.04	1.75	1.30	262.31	0.86	0.14

(b) SAC Trained with the perturbed parameter

Table G.3: Comparison of SAC performance with original and perturbed parameter training: ϕ

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	129.67	1.67	-1.07	1.60	1.25	243.81	0.84	0.16
+10 %	83.57	1.60	-1.12	1.70	1.25	217.60	0.85	0.15
-20 %	154.96	1.71	-1.08	1.55	1.25	257.43	0.83	0.17
+20 %	63.37	1.49	-1.21	1.75	1.24	200.77	0.86	0.14
-30 %	179.95	1.60	-1.13	1.50	1.25	264.32	0.85	0.15
+30 %	42.57	1.59	-1.16	1.80	1.24	193.57	0.85	0.15

(a) SAC Pre-trained on original Store 1 parameters

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	129.84	1.72	-1.06	1.60	1.25	246.39	0.83	0.17
+10 %	84.49	1.02	-1.47	1.69	1.22	192.03	0.92	0.08
-20 %	156.72	1.98	-0.90	1.55	1.27	270.18	0.80	0.20
+20 %	65.79	1.04	-1.49	1.74	1.21	181.37	0.91	0.09
-30 %	184.33	1.98	-0.91	1.50	1.27	281.93	0.80	0.20
+30 %	50.16	0.26	-1.92	1.76	1.18	138.60	0.98	0.02

(b) SAC Trained with the perturbed parameter

Table G.4: Comparison of SAC performance with original and perturbed parameter training: C_0

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	105.93	1.55	-1.17	1.65	1.24	226.84	0.85	0.15
+10 %	105.89	1.60	-1.13	1.65	1.25	229.31	0.85	0.15
-20 %	105.90	1.48	-1.21	1.65	1.24	223.66	0.86	0.14
+20 %	105.85	1.57	-1.13	1.65	1.25	227.93	0.85	0.15
-30 %	105.98	1.59	-1.07	1.65	1.25	228.90	0.86	0.14
+30 %	105.84	1.58	-1.14	1.65	1.25	228.16	0.85	0.15

(a) SAC Pre-trained on original Store 1 parameters

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	105.43	1.88	-0.94	1.65	1.26	242.22	0.81	0.19
+10 %	105.88	1.62	-1.11	1.65	1.25	230.26	0.84	0.16
-20 %	105.91	1.52	-1.17	1.65	1.24	225.36	0.86	0.14
+20 %	105.80	1.50	-1.20	1.65	1.24	224.47	0.86	0.14
-30 %	105.95	1.50	-1.20	1.65	1.24	224.76	0.86	0.14
+30 %	105.80	1.51	-1.20	1.65	1.24	225.08	0.86	0.14

(b) SAC Trained with the perturbed parameter

Table G.5: Comparison of SAC performance with original and perturbed parameter training: C_d

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	106.09	1.61	-1.11	1.65	1.25	229.63	0.85	0.15
+10 %	105.69	1.63	-1.11	1.65	1.25	230.73	0.84	0.16
-20 %	106.25	1.68	-1.06	1.65	1.25	232.64	0.84	0.16
+20 %	105.37	1.44	-1.23	1.65	1.24	221.80	0.87	0.13
-30 %	106.48	1.46	-1.21	1.65	1.24	222.99	0.87	0.13
+30 %	105.16	1.75	-1.07	1.65	1.25	235.89	0.83	0.17

(a) SAC Pre-trained on original Store 1 parameters

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	106.11	1.52	-1.17	1.65	1.24	225.50	0.86	0.14
+10 %	105.68	1.61	-1.12	1.65	1.25	229.40	0.85	0.15
-20 %	106.21	1.70	-1.08	1.65	1.25	233.83	0.83	0.17
+20 %	105.49	1.64	-1.10	1.65	1.25	230.92	0.84	0.16
-30 %	106.48	1.49	-1.22	1.65	1.24	223.96	0.86	0.14
+30 %	105.09	1.80	-1.00	1.65	1.26	238.51	0.82	0.18

(b) SAC Trained with the perturbed parameter

Table G.6: Comparison of SAC performance with original and perturbed parameter training: C_s

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	104.74	1.54	-1.26	1.65	1.19	226.44	0.88	0.12
+10 %	107.30	1.57	-1.05	1.65	1.29	227.65	0.83	0.17
-20 %	103.38	1.58	-1.38	1.65	1.14	228.07	0.89	0.11
+20 %	108.53	1.51	-0.98	1.65	1.34	225.08	0.82	0.18
-30 %	102.47	1.56	-1.49	1.65	1.09	227.37	0.91	0.09
+30 %	108.79	1.37	-0.91	1.65	1.38	218.84	0.82	0.18

(a) SAC Pre-trained on original Store 1 parameters

% of Original Value	Expected Profit	z_1	z_2	p_1	p_2	Q	γ	$1-\gamma$
-10 %	104.87	1.49	-1.21	1.65	1.20	224.26	0.89	0.11
+10 %	107.62	1.75	-1.07	1.65	1.29	236.23	0.79	0.21
-20 %	104.44	1.19	-1.48	1.64	1.13	210.93	0.95	0.05
+20 %	110.33	1.97	-1.02	1.65	1.34	246.23	0.71	0.29
-30 %	104.31	1.03	-1.68	1.64	1.07	204.30	0.98	0.02
+30 %	113.68	1.98	-1.09	1.65	1.38	246.70	0.65	0.35

(b) SAC Trained with the perturbed parameter

Table G.7: Comparison of SAC performance with original and perturbed parameter training: R

Short Biography

Nefeli Eleftheria Sextou completed the integrated Master's program, receiving a diploma in Computer Science and Engineering from the Department of Computer Science and Engineering at the University of Ioannina in 2023. Her diploma thesis, titled "Optimized Multi-Criteria Decision Analysis Through Median Ranking and Analytical Hierarchy Process", addresses the consensus (median) ranking problem, aiming to find a representative ranking that aggregates individual preferences. The work explores metaheuristic approaches and integrates criteria-based rankings through the Analytic Hierarchy Process, providing a framework for decision-making applications. Building on a broad and multidisciplinary education in computer science, engineering, and data science, her present research interests lie in optimization, machine learning, and operations research, and she is equipped to contribute across a wide range of roles in data science, machine learning, and analytics.