



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Handling Partial Observability in Non-Stationary Reinforcement Learning via Hidden Markov Models

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE & ENGINEERING -
INGEGNERIA INFORMATICA

Author: **Caterina Motti**

Student ID: 252240

Advisor: Prof. Marcello Restelli

Co-advisors: Gusmeroli Enea, Lavelli Alessandro, Tenedini Davide

Academic Year: 2025-26

Abstract

The remarkable success of Reinforcement Learning in solving complex decision-making problems is built upon the foundational assumption of environmental stationarity. Although this simplification has enabled the development of algorithms with strong convergence guarantees, capable of learning stable, optimal policies through interaction with fixed environments, it rarely holds in real-world applications where system dynamics evolve over time. This challenge is further compounded by partial observability, as the underlying processes driving these environmental shifts are often unobservable, making them subtle and difficult for the agent to detect.

This work specifically addresses non-stationary reinforcement learning under partial observability, modelling the environment as a Hidden Markov Model where a latent context governs the transition dynamics. We propose a decoupled framework that separates latent mode estimation from policy optimisation. The latent estimator is learned offline from a static dataset of transitions, using an extended version of the classical Baum-Welch algorithm to account for conditioning on the state-action tuples. The learned model is then deployed online to maintain a belief over the currently active latent mode, using Bayesian filtering to augment the agent’s observation space, thus reducing the non-stationary, partially observable problem to a stationary belief Markov Decision Process, solvable with standard reinforcement learning algorithms.

We evaluate this framework on discrete and continuous non-stationary variants of the Cliff Walking environment, under both deterministic and stochastic latent dynamics. Our results demonstrate that the proposed latent estimator consistently bridges the performance gap with the oracle baseline, achieving convergence in scenarios where the naive agent systematically fails.

Keywords: reinforcement learning, non-stationary environments, partial observability, latent space, hidden Markov models, Bayesian filtering

Abstract in lingua italiana

Il notevole successo dell'apprendimento per rinforzo (Reinforcement Learning) nella risoluzione di problemi decisionali complessi si basa su un'assunzione fondamentale: la stazionarietà dell'ambiente. Sebbene tale semplificazione abbia consentito lo sviluppo di algoritmi dotati di forti garanzie di convergenza, capaci di apprendere politiche stabili e ottimali in contesti fissi, essa raramente trova riscontro nelle applicazioni del mondo reale, dove la dinamica del sistema evolve nel tempo. Questa sfida è ulteriormente aggravata dalla parziale osservabilità, poiché i processi sottostanti che guidano tali cambiamenti ambientali sono spesso non osservabili, subdoli e difficili da rilevare per l'agente.

Questo lavoro si occupa esattamente di apprendimento per rinforzo in contesti non stazionari di osservabilità parziale, modellando l'ambiente come un modello Markoviano nascosto in cui un contesto latente ne governa le dinamiche di transizione. Proponiamo un'architettura disaccoppiata che separa la stima dello stato latente dall'ottimizzazione della politica. Lo stimatore latente è appreso offline da un dataset statico di transizioni tramite una versione estesa dell'algoritmo classico di Baum-Welch che condiziona le emissioni sulle tuple stato-azione. Questo modello viene poi impiegato online, tramite un filtraggio Bayesiano, per mantenere un'opinione (belief) sul contesto latente attualmente attivo. In questo modo, il problema non stazionario e parzialmente osservabile viene ridotto a un classico processo decisionale Markoviano, risolvibile tramite algoritmi standard di Reinforcement Learning.

Abbiamo validato questo sistema su varianti non stazionarie, sia discrete che continue, dell'ambiente Cliff Walking, sia sotto dinamiche latenti deterministiche che stocastiche. I nostri risultati dimostrano che lo stimatore latente proposto colma il divario di prestazioni rispetto all'oracolo, raggiungendo la convergenza in scenari in cui l'agente ingenuo fallisce sistematicamente.

Parole chiave: apprendimento per rinforzo, ambienti non-stazionari, osservabilità parziale, spazi latenti, modelli Markoviani nascosti, filtro Bayesiano

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
1.1 Motivation	1
1.2 Contributions	2
1.3 Document Structure	2
2 Preliminaries	5
2.1 Concept Drift	5
2.1.1 Drift Taxonomy	5
2.1.2 Drift Detection and Adaptation	6
2.2 Reinforcement Learning	7
2.2.1 Markov Decision Process	8
2.2.2 Policies and Value Functions	9
2.2.3 Standard RL Algorithms	11
2.2.4 Stationarity Assumptions	13
2.3 Concept Drift in Reinforcement Learning	13
2.3.1 RL-Specific Challenges	14
2.3.2 Modelling Non-Stationarity in RL	15
2.3.3 Hidden Markov Models	17
3 Related Work	21
3.1 Active Approaches	21
3.1.1 Model Quality	21
3.1.2 Change-Point Detection	22
3.1.3 Latent Space Detection	23

3.1.4	Performance-Driven Adaptation	24
3.2	Passive Approaches	25
3.2.1	Latent Context Inference	26
3.2.2	Optimistic exploration	27
3.2.3	Robust Planning	27
3.2.4	Tracking	28
4	Problem Formulation	31
4.1	Environment	31
4.2	Augmented State Space	32
4.3	Objective Function	33
4.4	Assumptions	33
5	Methodology	35
5.1	Architectural Overview	35
5.2	Latent Mode Estimator	36
5.2.1	Tabular Input-Conditioned Baum-Welch	37
5.2.2	Neural Input-Conditioned Baum-Welch	38
5.2.3	Training Considerations	39
5.3	Context-Conditioned Policy Optimization	41
5.3.1	Bayesian Filtering	41
5.4	Numerical Stability	42
6	Experimental Settings	45
6.1	Environments	45
6.1.1	Cliff Walking Wind	45
6.1.2	Continuous Cliff Walking Wind	46
6.2	Latent Dynamics	48
6.3	Results	49
6.3.1	Discrete Domain	49
6.3.2	Continuous Domain	53
7	Conclusions	59
7.1	Summary of Contributions	59
7.2	Limitations and Future Work	60
7.2.1	Direct Improvements to the Current Framework	60
7.2.2	Extensions Beyond the Current Assumptions	61

Bibliography	63
A Appendix A	69
A.1 Baum-Welch Algorithm	69
A.2 Input-Conditioned Baum-Welch Algorithms	70
A.3 Hyperparameters	71
 List of Figures	 73
 List of Tables	 75
 List of Acronyms	 77

1 | Introduction

In this chapter, we introduce the motivation behind this work, outline its main contributions, and describe the organisation of the remainder of the document.

1.1. Motivation

The remarkable success of Reinforcement Learning in solving complex decision-making problems has been built on a single foundational assumption: the stationarity of the environment. Standard Reinforcement Learning (RL) algorithms, framed as stochastic approximation methods [9], rely on the strict requirement that the transition dynamics and reward structure remain fixed throughout the agent’s lifetime, providing the theoretical guarantees for convergence to stable, optimal policies [57, 60]. Yet, this assumption is almost always violated in practice, and when it occurs, the learned policies become unreliable.

Real-world scenarios are inherently dynamic, subject to changes in their underlying processes that may not be directly observable by the agent. For example, consider a robot learning to navigate an environment where the surface properties change over time due to friction or terrain texture in ways that are not directly measurable by the robot’s sensors. Or consider a clinical decision support system that must adapt treatment recommendations to a patient whose physiological response to therapy changes over time, in ways that no single measurement can fully characterise. In both cases, the challenge is not merely environmental change, but the fact that these changes are driven by a hidden variable that shifts the dynamics without leaving a direct trace in the agent’s observations. To the agent, the world simply becomes less predictable since identical actions lead to different outcomes, and the policies it has learned begin to fail.

Despite the importance of this challenge, the vast majority of RL research has focused exclusively on stationary settings. Non-stationarity and partial observability are widely recognised as among the most significant open challenges for the deployment of RL systems beyond controlled simulations, yet they remain comparatively under-explored [20]. We

aim to bridge this gap by focusing specifically on settings in which non-stationarity arises from a latent, unobservable process.

1.2. Contributions

This work makes the following contributions:

- **A decoupled framework for non-stationary RL:** We propose an architecture that separates latent mode estimation from policy optimisation into two independent components. The latent estimator is trained offline on a static dataset of historical trajectories, and its output is used online to augment the environment’s state space, reducing the non-stationary Partially Observable Markov Decision Process (POMDP) to a stationary belief Markov Decision Process (MDP) without requiring structural modifications to the RL algorithm.
- **Extension of Baum-Welch algorithms:** We extend the classical Baum-Welch algorithm to handle input-conditioned emissions, where the emission distribution is conditioned on the full state-action pair rather than the latent state alone. We derive both a tabular variant for discrete environments and a neural variant for continuous state and action spaces.
- **Empirical evaluation on non-stationary environments:** We evaluate the proposed framework on two non-stationary environments based on the Cliff Walking task, under both deterministic and stochastic latent dynamics, benchmarking against a standard RL baseline and an oracle agent with direct access to the true latent mode.

1.3. Document Structure

This document is structured as follows:

- **Chapter 2: Preliminaries**

We introduce the foundational mathematical concepts to provide the theoretical background needed to understand our contribution. Starting from concept drift in Supervised Learning, we introduce Reinforcement Learning and its inherent complexities, presenting Partially Observable Markov Decision Processes, Hidden Markov Models and the Baum-Welch algorithm as the key building blocks of the proposed approach.

- **Chapter 3: Related Work**

We provide a comprehensive literature overview, surveying state-of-the-art work on non-stationary RL. We organise existing work into two groups: whether the agent actively adapts to non-stationarity or passively detects it.

- **Chapter 4: Problem Formulation**

We formally define the non-stationary RL problem addressed in this work by modelling the environment as an Hidden Mode MDP and effectively framing it as a POMDP governed by latent modes. To resolve the resulting partial observability, we employ the belief state as a sufficient statistic for the agent’s history, restoring the Markov property of the environment. Furthermore, we state the theoretical assumptions that bound the scope of this work.

- **Chapter 5: Methodology**

We detail the proposed decoupled architecture by breaking it down into two phases: the offline phase, where the latent mode estimator is learned via the input-conditioned Baum-Welch algorithm, and the online phase, in which the frozen estimator augments the state space, enabling standard RL agents to learn optimal policies without needing any structural modifications.

- **Chapter 6: Experiments**

To validate our framework, we evaluate its performance on two non-stationary extensions of the Cliff Walking task. We first introduce the experimental setup, detailing the specific environmental dynamics and the evolution of the underlying latent variable. Then, we discuss the empirical results: we evaluate the accuracy of the latent estimator and analyse the agent’s online control capabilities under both deterministic and stochastic latent dynamics, comparing its performance to a naive baseline and a fully observable oracle.

- **Chapter 7: Conclusions and Future Developments**

Finally, we summarise the key findings and contributions of this work, discuss its limitations, and outline promising directions for future research.

2 | Preliminaries

This chapter presents the theoretical foundations needed to understand the core contribution of this work. First, we introduce the concept drift, along with drift detection and adaptation in the most general case. Second, we formalise RL through the MDP framework, introducing the concepts of policies and value functions, and the standard algorithms that serve as baselines in our experiments. Finally, we delve into non-stationary RL, presenting the POMDP framework to model environments governed by hidden contexts, and introduce Hidden Markov Model (HMM) as the mathematical tool to infer and track these hidden states.

2.1. Concept Drift

Classical Machine Learning (ML) approaches assume that the underlying data distribution is generated at random according to a fixed, but unknown, stationary probability distribution. However, in real-world scenarios, this assumption rarely holds as the data flows continuously and the data generation process may change over time. When this assumption is violated, the performance of the learned models degrades silently, with no explicit signal indicating that the knowledge they have acquired is becoming obsolete.

This is exactly the idea of concept drift: the target variable predicted by the model changes over time, leading to poor prediction accuracy and a degradation of the learned mapping. This has motivated the need for novel learning approaches that aim to accurately detect and effectively adapt to concept drift.

2.1.1. Drift Taxonomy

Let $\mathcal{P}$ be the data-generating process producing a sequence of samples (x_t, y_t) drawn from an unknown probability distribution $p_t(x, y)$. The conditional distribution $p_t(y | x)$ and the marginal distribution $p_t(x)$ represent the posterior and prior distributions, respectively. According to the surveys [18], [24], and [46], concept drift can be categorised according to three independent dimensions:

- Based on which component of the distribution changes over time:
 - **Real**: the posterior probability $p_t(y|x)$ changes over time.
 - **Virtual**¹: the prior probability $p_t(x)$ changes over time.
- Based on the rate at which changes occur:
 - **Abrupt**: a sudden and instantaneous change.
 - **Gradual**: a smooth, gradual transition over time.
 - **Incremental**: a continuous, incremental change over time.
- Based on the duration of the effect:
 - **Permanent**: not limited in time.
 - **Transient**: after a certain amount of time, it vanishes.
 - **Recurring**: follows a cyclical pattern. In this case, the ability to reuse previously acquired knowledge becomes particularly relevant.

2.1.2. Drift Detection and Adaptation

Drift detection refers to the techniques used to identify change points or change time intervals. According to [46], concept drift detection algorithms can be classified according to the test statistics they apply into:

- **Error-rate based**: These methods monitor the online error rate of the algorithm. Statistical theory guarantees that as long as the distribution is stationary, the error will decrease or stabilise. On the contrary, when the distribution changes, the error rate increases. If a statistically significant change is detected, a drift is signalled. The most referenced works in the literature are Drift Detection Method (DDM) [23] and ADaptive WINdowing (ADWIN) [10].
- **Data-distribution based**: These algorithms monitor the dissimilarity between the current and historical distributions. If this dissimilarity is proven to be statistically significant, a drift is triggered. Notable examples in literature include the Hellinger Distance-based Drift Detection Method (HDDDM) [17] and Kolmogorov-Smirnov Windowing (KSWIN) [52]

¹It is called *virtual* because the model is not technically obsolete as the mapping $p(y | x)$ remains unchanged. However, performance may still degrade if the new $p(x)$ covers regions of the input space that were not clearly represented before.

- **Multiple hypothesis tests** Drift is detected through the use of multiple hypothesis tests in parallel or a hierarchical way. A prominent example of the hierarchical framework is employed in Just-In-Time (JIT) classifiers [4].

Once drift is detected, an adaptation mechanism may be triggered. Drift adaptation refers to all methods that aim to minimise the drop in accuracy and achieve the fastest recovery rate [24]. Depending on the type of drift, a different adaptation technique can be used:

- **Training a new model:** The most straightforward approach, in which the model is retrained on the most recent data once a drift is detected. It is particularly effective for global drift and abrupt drift, where past knowledge becomes entirely obsolete. A classic implementation is the FLOating Rough Approximation 2 (FLORA2) algorithm [62], which relies on sliding windows.
- **Model ensemble:** A set of models is maintained, and their output is combined according to a voting or weighting rule. In case of recurring drift, previously trained models can be reused, avoiding the cost of retraining from scratch. Prominent work in this category includes the Dynamic Weighted Majority (DWM) algorithm [39], which dynamically adjusts the ensemble by creating or removing experts based on their recent performance.
- **Incremental learning:** The model is updated incrementally as new data arrives, without requiring an explicit drift signal. A foundational example in the literature is the Hoeffding Tree (also known as Very Fast Decision Tree) [19].

Another distinction can be made based on whether adaptation is triggered explicitly or occurs continuously [18]:

- **Active:** The model is updated after a signal from the drift detection mechanism. This approach is more stable, but strongly depends on the accuracy of the detector.
- **Passive:** The model parameters are continuously updated as data arrives. This allows for faster recovery to drift, but it may lead to noise overfitting.

2.2. Reinforcement Learning

RL [57] is an ML paradigm for solving sequential decision making problems. It involves an agent that interacts with the environment by taking actions and receives a reward that provides feedback on the quality of its behaviour. Unlike Supervised Learning (SL), which relies on a dataset that instructs the model on the exact correct output, an RL agent learns purely from its own experience.

2.2.1. Markov Decision Process

Most RL problems, or sequential decision-making problems at large, can be formalised as a Markov Decision Process (MDP) [51].

An MDP is a stochastic process defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$ where:

- $\mathcal{S}$ is the set of states that can be finite or infinite.
- $\mathcal{A}$ is the set of actions that can be finite or infinite.
- P is the state transition probability distribution, $P(s' \mid s, a)$.
- R is the reward function, $R(s, a) = \mathbb{E}[r \mid s, a]$, where r is the immediate scalar reward signal generated by the environment.
- $\gamma \in [0, 1]$ is the discount factor, which represents the present value of future rewards.
- μ is the initial state distribution that specifies how the initial state s_0 is generated.

The state transition function P must satisfy a fundamental assumption about the environment dynamics, known as the **Markov Assumption** (or Markov property). Informally, this property states that the future is independent of the past given the present: this means that the current state contains all the necessary information needed to determine the future. Formally, a stochastic process is said to be Markovian if the probability distribution of the future state, conditioned on all past states and actions, depends only on the current state and action:

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = P(s_{t+1} \mid s_t, a_t).$$

A typical RL interaction is illustrated in Figure 2.1. Starting from an initial state $s_0 \sim \mu$, the process unfolds over discrete time steps t . At each time step t , the agent observes the state s_t and reward r_t , then chooses an action $a_t \in \mathcal{A}$ according to a decision-making strategy. The environment responds by emitting the immediate reward r_{t+1} and transitioning to a successor state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$.

The agent's goal is to maximise the total amount of reward it receives. This objective introduces complex challenges such as the *credit assignment problem* (determining which past action led to a delayed reward) and the *exploration-exploitation dilemma* (balancing the exploitation of known high-rewarding strategies with exploring potentially better actions).

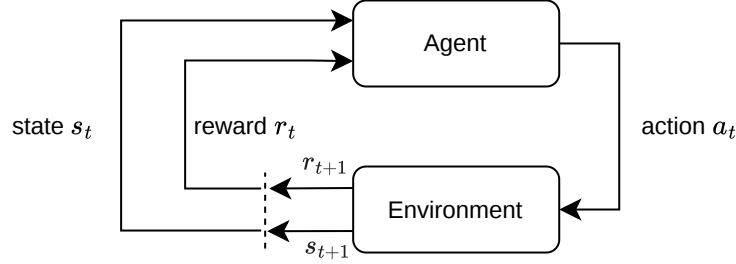


Figure 2.1: Agent-Environment interaction loop. The standard agent-environment interaction loop in RL showing the continuous interaction through states, actions, and rewards. Redrawn from [57].

The sequence of interactions collected up to time t is called a trajectory or history:

$$\tau_t = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_t, a_t, r_t).$$

2.2.2. Policies and Value Functions

The agent's decision-making strategy is specified by a policy, which is a mapping from a trajectory to an action.

In the most general case, a policy is a mapping $\pi : \mathcal{H} \rightarrow \Delta(\mathcal{A})$ where $\mathcal{H}$ is the set of all possible trajectories of all lengths and $\Delta(\mathcal{A})$ is the space of probability distributions over actions.

A policy can be stochastic if it selects actions according to a probability distribution, or deterministic if it maps a state directly to an action. In a stationary and Markovian setting, the agent chooses actions based only on the current state, i.e., $a_t \sim \pi(\cdot | s_t)$. In this case, the policy is said to be stationary.

For a fixed policy π and a starting state $s_0 = s$, we define:

- **State-Value function** $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$ as the expected discounted sum of future rewards (also called *return*) starting from state s and following policy π :

$$V^\pi(s) = \mathbb{E}_{\substack{a_t \sim \pi(\cdot | s_t) \\ s_{t+1} \sim P(\cdot | s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s \right].$$

- **Action-Value function** (or Q-value) $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ as the expected return

starting from state s , taking action a , and subsequently following policy π :

$$Q^\pi(s, a) = \mathbb{E}_{\substack{a_t \sim \pi(\cdot | s_t) \\ s_{t+1} \sim P(\cdot | s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a \right].$$

Value functions can be decomposed into immediate reward plus the discounted reward of the expected subsequent states. These recursive formulations are called the **Bellman equations**:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\substack{a_t \sim \pi(\cdot | s_t) \\ s_{t+1} \sim P(\cdot | s_t, a_t)}} [R(s_t, a_t) + \gamma V^\pi(s_{t+1}) \mid s_t = s]; \\ Q^\pi(s, a) &= \mathbb{E}_{\substack{s_{t+1} \sim P(\cdot | s_t, a_t) \\ a_{t+1} \sim \pi(\cdot | s_{t+1})}} [R(s_t, a_t) + \gamma Q^\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a]. \end{aligned}$$

A policy π is considered better than (or equal to) another policy π' if its expected return is better than (or equal to) that of π' for all states. Formally:

$$\pi \geq \pi' \iff V^\pi(s) \geq V^{\pi'}(s), \quad \forall s \in \mathcal{S}.$$

This partial ordering guarantees the existence of at least one policy that is better than or equal to all others: we call this the optimal policy, denoted by π^* . Solving an RL problem means exactly finding the optimal policy π^* , which is the one that achieves the highest expected reward over the long run. Although there may exist multiple optimal policies, they all share the same optimal value functions:

- **Optimal state-value function** $V^*(s) = \max_{\pi} V^\pi(s), \forall s \in \mathcal{S}$.
- **Optimal action-value function** $Q^*(s, a) = \max_{\pi} Q^\pi(s, a), \forall s \in \mathcal{S} \text{ and } \forall a \in \mathcal{A}$.

Following the same logic we used before for the state-value and action-value functions, we can derive the **Bellman optimality equations**:

$$V^*(s) = \max_{a_t \in \mathcal{A}} \mathbb{E}_{s_{t+1} \sim P(\cdot | s_t, a_t)} [R(s_t, a_t) + \gamma V^*(s_{t+1}) \mid s_t = s]; \quad (2.1)$$

$$Q^*(s, a) = \mathbb{E}_{s_{t+1} \sim P(\cdot | s_t, a_t)} \left[R(s_t, a_t) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^*(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a \right]. \quad (2.2)$$

Ultimately, for any MDP we can always find a deterministic optimal policy by maximising over the optimal state-action value function:

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg \max_{a \in \mathcal{A}} Q^*(s, a); \\ 0 & \text{otherwise.} \end{cases}$$

2.2.3. Standard RL Algorithms

Countless algorithms have been proposed to solve RL problems. Before introducing the specific methods we use as baselines for our experiments, we provide a taxonomy based on the algorithm’s learning strategy.

Model-based vs Model-free Model-based algorithms estimate the transition model P and the reward model R from experience in order to learn optimal policies and optimal value functions. Model-free algorithms directly either search for optimal value functions or find a value function of a policy and improve it using function approximation.

Value-based vs Policy-based vs Actor-Critic Value-based methods estimate the optimal value function and implicitly derive the policy. Policy-based methods directly parametrise and optimise the policy itself. A hybrid approach, known as Actor-Critic, combines both by learning a value function (the critic) to update the policy (the actor).

On-policy vs Off-policy On-policy algorithms evaluate and improve a policy that is used both to make decisions and to explore the environment, so they suffer significantly from the exploration-exploitation dilemma. Off-policy algorithms use a behaviour policy to generate the trajectory data and simultaneously learn an optimal target policy.

Tabular vs Function Approximation Tabular methods maintain a look-up table to store exact values for each state or state-action pair, making them suitable for environments with small discrete state and action spaces. For high-dimensional or continuous spaces, function approximation techniques are employed. In modern applications, Deep Neural Networks are the standard choice for function approximation, leading to the field of Deep Reinforcement Learning (DRL).

Online vs Offline In online RL, the agent continuously interacts with the environment, collecting trajectories and updating its policy in real-time. In offline RL, also referred to as batch RL, the agent learns from a pre-collected dataset of past trajectories.

Based on these distinctions, we introduce the two algorithms used as baselines: Tabular Q-learning and Soft Actor-Critic (SAC).

Tabular Q-learning

Tabular Q-Learning [60] is a model-free, off-policy, and value-based algorithm. It is best suited for discrete small state and action spaces, as exact value functions are stored and

updated iteratively using the Bellman optimality equation (2.2). It maintains a look-up table of value functions, which is updated as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s, a) \right],$$

where $\alpha \in (0, 1]$ is the learning rate and $r = R(s, a)$ is the immediate reward.

Since the Q-learning update relies on the transitions collected by the agent, if the agent acts purely greedily by always selecting the action yielding the highest estimated value, the Q-table will only become accurate for the initially visited state-action pairs. To discover the optimal policy, the agent must actively explore the environment. This exploration-exploitation trade-off is typically addressed using an ϵ -greedy policy: the agent exploits by choosing the action that maximises the Q-value with probability $1 - \epsilon$, and explores with probability ϵ by sampling an action uniformly at random from the action space.

Soft Actor-Critic

While Tabular Q-Learning is highly effective in discrete domains, it becomes completely intractable in continuous state and action spaces due to the curse of dimensionality. To operate in continuous environments, modern RL relies on function approximators, such as neural networks. A state-of-the-art method leveraging this approach is SAC [26], a model-free, off-policy, actor-critic algorithm.

Rather than relying on external heuristic mechanisms like the ϵ -greedy policy, SAC incorporates exploration directly into its optimisation objective through entropy maximisation, using an entropy term $H(\pi(\cdot | s))$, which measures the randomness of the policy. The differential Shannon entropy encourages π to be stochastic:

$$H(\pi(\cdot | s)) = - \int_a \pi(a | s) \log \pi(a | s) da.$$

The objective is:

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi}} \left[\gamma^t (R(s_t, a_t) + \alpha_{ent} H(\pi(\cdot | s_t))) \right],$$

where ρ_{π} is the state-action trajectory distribution induced by the policy (or state-action occupancy), and $\alpha_{ent} > 0$ is the temperature parameter that determines the relative importance of the entropy term against the reward. The state-action occupancy is formally defined as:

$$\rho_{\pi}(s_t, a_t) = P(s_t | \pi) \pi(a_t | s_t).$$

In this formulation, the entropy term is analogous to the ϵ parameter, as both explicitly prevent the policy from collapsing into premature determinism. However, while ϵ -greedy enforces exploration through blind, uniformly random actions, SAC’s entropy maximisation provides a state-dependent exploration that scales effectively to continuous domains.

2.2.4. Stationarity Assumptions

The theoretical guarantees of standard RL algorithms, including the convergence to optimal value functions, rely on fundamental **stationarity assumptions** on the environment dynamics. Specifically, both the transition probability and the reward function are assumed to be independent of the timestep t :

$$\begin{aligned} P(s_{t+1} = s' \mid s_t = s, a_t = a) &= P(s' \mid s, a) \quad \forall t \geq 0; \\ \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] &= R(s, a) \quad \forall t \geq 0. \end{aligned}$$

However, the stationarity assumptions presented above are rarely satisfied in real-world scenarios. When they are violated, the theoretical convergence guarantees of standard RL algorithms no longer hold, and the learned policy may become suboptimal. In the following section, this problem is formalised, and the tools needed to address it are introduced.

2.3. Concept Drift in Reinforcement Learning

While concept drift in SL affects the input–output distribution, in RL, it impacts the agent–environment interaction itself, making detection and adaptation substantially more challenging. Moreover, drift can affect different components of the underlying MDP, since the transition function, the reward function, and even the state-action space may evolve over time.

Let us consider a real-world scenario to better understand how drift can impact an agent. Consider a robot trained to navigate in a warehouse. Initially, the environment is empty, and the robot learns the optimal policy to navigate. Over time, obstacles are introduced, or the goal is modified to reflect new priorities. The optimal policy learned in the original environment is no longer valid, yet the agent does not have an explicit signal indicating that a change has occurred.

Before delving deeper, we must establish a clear distinction between two terms that are often used interchangeably in literature: task and context.

- A **task** defines the goal the agent is trying to achieve, characterised by the reward function R .

- A **context** identifies a specific environment in which the task is performed, formalised by both the transition dynamics P and the reward function R .

In our warehouse scenario, the context changes when an obstacle is placed (altering the transition dynamics) while the task shifts when a certain zone is prioritised (altering the reward function alone). It follows that a change of context may include a change of task, or the two may coincide.

To properly frame the problem, it is useful to distinguish between related but distinct areas in the literature:

- **Multi-task RL** focuses on a single agent solving multiple tasks simultaneously, but assuming that the set of tasks is known and fixed during training.
- **Meta-RL** aims to learn a procedure that enables rapid adaptation to new contexts, assuming access to the context distribution during training.
- **Non-stationary RL** refers to settings where the environment dynamics change over time.
- **Continual RL** emphasises the agent’s ability to learn continuously without forgetting previously acquired knowledge.
- **Lifelong RL** focuses on the long-term perspective, where an agent accumulates and transfers knowledge across a potentially infinite sequence of tasks.

2.3.1. RL-Specific Challenges

Solving non-stationary RL problems adds additional challenges: the agent must simultaneously realise that its current policy is no longer optimal and adapt to the new environment dynamics. Moreover, drift detection and adaptation are considerably harder here than in SL, since the agent is both the source of data and the subject of evaluation.

Operating in this setting introduces several distinct challenges that must be taken into account:

- **Sample Efficiency:** Since learning depends on the state-reward samples obtained, if these samples are not informative about the change of context, the algorithm will take longer to learn the new policy.
- **Delayed Rewards:** In a changing environment, it is challenging to determine whether a low reward arises from poor agent performance or is the result of an environmental drift.

- **Exploration-Exploitation:** In stationary settings, exploration decays as the agent converges to the optimal policy. In non-stationary environments, the agent must continuously explore to detect context changes. Thus, a permanent component of exploration must be maintained, which will inevitably affect short-term performance.
- **Stability-Plasticity Dilemma:** This dilemma is about balancing the importance of recent versus past data. Failing to resolve this conflict leads to catastrophic forgetting [47], where the RL agent adapts well to recent experiences while deteriorating its capabilities on past ones. This is particularly relevant in the DRL setting, where learning a new context results in the backpropagation process overwriting the weights that were successful in the previous context.
- **Computation complexity:** Single-agent RL algorithms already suffer from the curse of dimensionality with large state-action spaces, so deep approaches, which make use of GPUs, are often employed. Continuous drift detection and adaptation put an additional computational burden on the system, which must be taken into account.

Addressing these challenges is not straightforward, as it requires balancing conflicting goals to maintain a robust and adaptable agent. These challenges motivate the need for a formal framework to model and track context changes over time.

2.3.2. Modelling Non-Stationarity in RL

Non-stationary RL problems can be modelled from two complementary views, depending on whether the non-stationarity is treated as a property of the environment or as a consequence of the agent’s limited observability [38].

Multiple Contexts View² Since every component of the environment may be time-dependent, we have a sequence of MDPs $\{M_k\}_{k \in \mathbb{N}^+}$, where $M_k = \langle \mathcal{S}_k, \mathcal{A}_k, P_k, R_k \rangle$. This implies that there is a discrete set of (potentially unrelated) contexts. Changes in the context of the environment are known as changepoints. The set of changepoints $\{T_i\}_{i \geq 1}$, which is an increasing sequence of random integers, can be finite or infinite. In the most extreme case, no context is visited for more than a single timestep.

Partially Observable View As depicted in Figure 2.2, the environment itself is stationary in its dynamics, and the perceived non-stationarity is caused by the agent’s

²This perspective is referred to as *non-stationary function view* in [38] as each component of the MDP is a function that may depend on time. The term *multiple contexts view* is adopted here to better highlight the contrast with the partially observable view.

limited perspective, so it is a consequence of unobserved phenomena. We can formalise these non-stationary RL problems as POMDP [35], in which the environment dynamics and/or the reward function are governed by hidden variables that are not directly observable by the agent.

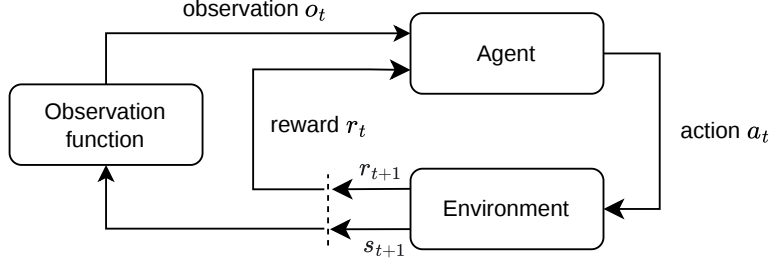


Figure 2.2: POMDP interaction loop. Agent-environment interaction loop under partial observability, highlighting the separation between the unobservable state s_{t+1} and the agent’s observation o_t .

A POMDP is a stochastic process defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu, \Omega, O \rangle$ where:

- $\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu$ describe the underlying MDP.
- Ω is a finite set of observations the agent can experience of the environment.
- O is the observation function defining the probability distribution $O(o|s_{t+1}, a)$ of observing $o \in \Omega$ after taking action a and reaching s_{t+1} .

Another crucial distinction presented in the survey [38] regards the source of non-stationarity, specifically whether it is influenced by the agent itself:

- **Passive non-stationary environments:** We assume that the non-stationarity does not depend on the agent’s behaviour. This allows us to model the stochastic function $p(z'|z)$ without taking into account the effects of our own changing policy on this distribution.
- **Active non-stationarity environments:** The agent behaviour influences the non-stationarity of the environment, which can be modelled in a Markovian way with the function $p(z'|s, a, z)$.
- **Hybrid environments:** The non-stationarity is influenced by both the agent behaviour and external events. A prominent example is Multi-Agent Reinforcement Learning (MARL), where an agent may be able to influence the environment perceived

by another agent.

In the POMDP setting, the agent does not know the current state to act optimally; therefore, it must maintain a **belief state** b that summarises its previous experiences. This belief state is often modelled as a probability distribution over the possible hidden contexts. We rely on the HMM framework to describe how the environment transitions between hidden contexts.

2.3.3. Hidden Markov Models

An HMM is used to model a Markov process with unobservable (hidden) states. Given:

- $\mathcal{Z} = \{z_1, \dots, z_N\}$ is the discrete set of hidden states (or contexts).
- $\mathcal{O} = \{o_1, \dots, o_M\}$ is the set of all possible observations.

an HMM is formally defined by a tuple $\langle A, B, b_1 \rangle$, where:

- A is the hidden state transition model, where $A(z_j | z_i) = P(z_{t+1} = z_j | z_t = z_i)$ represents the probability of transitioning from context z_i to context z_j .
- B is the emission model, where $B(o | z) = P(o_t = o | z_t = z)$ is the probability of perceiving observation $o \in \mathcal{O}$ while the environment is in the hidden context z .
- b_1 is the initial state distribution ³, such that $b_1(z) = P(z_1 = z)$ determines the probability of starting in hidden state z at $t = 1$.

Following [53], any HMM-based approach must address three fundamental problems:

- Evaluation (**Forward algorithm**): Given the sequence of observations $O_{1:T} = \{o_1, \dots, o_T\}$ computing the probability that this observed sequence is produced by a given model $\lambda = \langle A, B, b_1 \rangle$, i.e., $P(O_{1:T} | \lambda)$.
- Decoding (**Viterbi algorithm**): Finding the optimal sequence of hidden states that best explains a given sequence of observations $O_{1:T} = \{o_1, \dots, o_T\}$ given a model $\lambda = \langle A, B, b_1 \rangle$.
- Learning (**Baum-Welch algorithm**): Maximizing $P(O_{1:T} | A, B, b_1)$ by adjusting the model parameters.

³In the literature of HMM, the hidden state distribution is conventionally denoted as π . However, to avoid conflict with the RL policy π , we denote the state distribution as b .

The Forward Algorithm

The forward algorithm computes the probability of seeing an observation sequence given a model: the **forward variable** $\alpha_t(z) = P(O_{1:t}, z_t = z \mid \lambda)$. It is calculated recursively:

$$\begin{aligned}\alpha_1(z) &= b_1(z) \cdot B(o_1 \mid z), \\ \alpha_t(z) &= B(o_t \mid z) \sum_{z' \in \mathcal{Z}} \alpha_{t-1}(z') A(z \mid z'), \quad \text{for } t > 1.\end{aligned}\tag{2.3}$$

The Viterbi Algorithm

The Viterbi algorithm finds the most probable sequence of hidden contexts that generated a given observation. It calculates the maximum probability of a single path ending in state z at time t , denoted by $\delta_t(z)$:

$$\begin{aligned}\delta_1(z) &= b_1(z) B(o_1 \mid z), \\ \delta_t(z) &= B(o_t \mid z) \max_{z' \in \mathcal{Z}} [\delta_{t-1}(z') A(z \mid z')], \quad \text{for } t > 1.\end{aligned}\tag{2.4}$$

Let us define $\psi_t(z)$ as the array that stores the argument maximising the equation above at each step:

$$\psi_t(z) = \arg \max_{z' \in \mathcal{Z}} [\delta_{t-1}(z') A(z \mid z')].\tag{2.5}$$

Then, the optimal path is recovered by backtracking through the pointers in ψ starting from the state $z_T^* = \arg \max_z \delta_T(z)$.

The Baum-Welch Algorithm

The Baum-Welch algorithm is an iterative Expectation-Maximization (EM) procedure that maximises the likelihood of the observed sequence by adjusting the model parameters.

In the expectation step (E-step), the forward algorithm is executed, producing $\alpha_t(z)$. Similarly, we define the backward variable $\beta_t(z) = P(O_{t+1:T} \mid z_t = z, \lambda)$, which calculates the probability of ending the sequence of observations in $O_{t+1:T}$:

$$\begin{aligned}\beta_T(z) &= 1, \\ \beta_t(z) &= \sum_{z' \in \mathcal{Z}} A(z' \mid z) B(o_{t+1} \mid z') \beta_{t+1}(z'), \quad \text{for } t < T.\end{aligned}\tag{2.6}$$

Once we have $\alpha_t(z)$ and $\beta_t(z)$, we can calculate the marginal probability of being in context

z at time t . This corresponds to the agent’s belief state in a POMDP:

$$\gamma_t(z) = P(z_t = z \mid O_{1:T}, \lambda) = \frac{\alpha_t(z)\beta_t(z)}{\sum_{k \in \mathcal{Z}} \alpha_t(k)\beta_t(k)}. \quad (2.7)$$

Then, we compute the joint probability of being in context z at time t and context z' at time $t + 1$, defined as $\xi_t(z, z')$:

$$\begin{aligned} \xi_t(z, z') &= P(z_t = z, z_{t+1} = z' \mid O_{1:T}, \lambda) \\ &= \frac{\alpha_t(z)A(z' \mid z)B(o_{t+1} \mid z')\beta_{t+1}(z')}{\sum_{k \in \mathcal{Z}} \sum_{w \in \mathcal{Z}} \alpha_t(k)A(w \mid k)B(o_{t+1} \mid w)\beta_{t+1}(w)}. \end{aligned} \quad (2.8)$$

In the maximisation step (M-step), the transition and emission models are updated using the expected values derived from the previous step. The (z, z') -th entry of the transition matrix is re-estimated as:

$$\hat{a}_{zz'} = \frac{\sum_{t=1}^{T-1} \xi_t(z, z')}{\sum_{t=1}^{T-1} \gamma_t(z)}. \quad (2.9)$$

On the other hand, the update of the emission model B depends on the chosen parametrisation. In the standard tabular formulation, where the emission matrix stores a categorical distribution for each hidden state, the (z, o) -th entry of the emission matrix is re-estimated as:

$$\hat{b}_{zo} = \frac{\sum_{t=1}^T \gamma_t(z) \mathbb{1}_{o_t=o}}{\sum_{t=1}^T \gamma_t(z)}. \quad (2.10)$$

However, when dealing with continuous observation spaces, this tabular approach becomes intractable. In such cases, a parametric model $B_\theta(o \mid z)$ that relies on function approximation is needed. Depending on the complexity of the environment’s dynamics, common alternatives include:

- **Gaussian Emissions** [54]: The emission model is assumed to follow a Gaussian distribution for each latent state, defined by parameters $\theta = \{(\mu_z, \Sigma_z)\}_{z \in \mathcal{Z}}$, which are estimated by updating the mean and covariance based on the posterior probabilities $\gamma_t(z)$ derived in the E-step.
- **Auto-Regressive Hidden Markov Models (AR-HMMs)** [34]: The emission is modelled as a function of the previous observation, typically via a linear regression $o_t = W_z o_{t-1} + \epsilon_z$. In this case, the weights W_z are estimated via weighted least squares, where each observation pair (o_{t-1}, o_t) is weighted by the posterior probability $\gamma_t(z)$. This provides a closed-form solution that effectively captures temporal dependencies within the observation sequence itself.

Regardless of the chosen emission model, this two-step procedure is iterated until the change in log-likelihood $\log P(O_{1:T} \mid \lambda)$ between successive iterations falls below a threshold. It can be shown that each iteration is guaranteed to increase or maintain the likelihood of the observed sequence, ensuring convergence to a local maximum [53]. Therefore, the choice of initialisation may affect the quality of the solution found. The full pseudocode of the Baum-Welch algorithm is reported in Appendix A.1 for completeness.

3 | Related Work

This chapter surveys the current state of the art regarding RL in non-stationary environments. We organise the literature into active and passive approaches, based on whether drift detection is explicit or not.

3.1. Active Approaches

Active approaches, as in SL, embed an explicit change-detection mechanism in the learning loop to trigger an adaptation upon an alarm. In the most general setting, a monitoring metric is chosen, and a change is declared when this metric crosses a threshold. Upon detection, an adaptation method is triggered; therefore, the quality of adaptation is directly coupled to the quality of detection. This coupling is both the strength and the weakness of active methods. On one hand, by explicitly triggering adaptation upon detection, these approaches can react quickly to abrupt shifts. On the other hand, they may struggle with gradual or continuous drift. Furthermore, the choice of monitoring metric is itself problem-dependent, and a poorly chosen metric may fail to detect changes that do not directly impact the quantity being observed.

Based on the nature of their detection signal, we categorise active methods into four distinct families: model quality monitoring, statistical change-point detection, latent space detection, and performance-driven adaptation. The reviewed literature is summarised in Table 3.1.

3.1.1. Model Quality

One of the most well-known methods in the literature is Reinforcement Learning with Context Detection (RL-CD) [8, 16]. It assumes that the environment is composed of several contexts, which are stationary and cannot be directly observed by the agent, but can only be inferred from the sequence of transitions and rewards. Moreover, it assumes that context changes occur at a low frequency and are independent of the agent’s actions. RL-CD is model-based since it uses a set of partial models M that estimate transition probabilities P_m and rewards R_m for a given context m . The algorithm initialises a single

model and iteratively adds others when the predictive quality of the active model falls below a minimum threshold. At each iteration, each partial model is evaluated on the current environment, and the best model is used for prediction, so it becomes active. Therefore, a context change is inferred whenever the current active model is replaced. A major drawback of this approach is that it is not theoretically grounded, as it employs heuristics and is based on several problem-dependent parameters.

3.1.2. Change-Point Detection

To overcome the lack of theoretical guarantees in heuristic approaches, most active methods rely on sequential analysis and statistical hypothesis testing to detect changes in the transition and/or reward functions.

An improvement of the aforementioned RL-CD is Sequential Change-Point Detection RL-CD (SCPD-RLCD) [27], which replaces the arbitrary model quality measure with a change-point detection method taken from sequential analysis. SCPD-RLCD monitors a metric that represents the combined effects of the change in transition and reward functions, computed using a CUMulative SUM (CUSUM) sequential statistical test ¹ [7]. This resulting method comes with the guarantees of sequential hypothesis testing, namely, bounded false-alarm rates and controlled detection delays.

However, [6] proves that this approach leads to a loss in performance due to delayed detection. For this reason, they formalise the problem within the Quickest Change Detection framework to minimise the time spent executing a sub-optimal policy. They propose a two-threshold switching method, which computes the Kullback-Leibler (KL) divergence between the transition distributions of two contexts under the same policy. By tracking this metric, the method provides a stopping rule for declaring a context change that strictly optimises the reward-detection trade-off.

Targeting the same issue, a more recent refinement of RL-CD is Model Based RL Context Detection (MBCD) [3], which focuses explicitly on minimising the detection delay, enabling fast recovery and therefore better performance. MBCD maintains an unbounded collection of probabilistic environment models, each specialising in a distinct latent context. A collection of models and policies for different contexts is maintained so that they can be deployed whenever they occur. This approach is particularly relevant for high-dimensional continuous problems, as the detection delay of the multivariate variant of CUSUM [29] employed is proved to be independent of the dimensionality of the data.

¹The CUSUM accumulates deviations from an expected value over time to detect shifts. By aggregating this data, rather than checking single observations, it effectively filters out noise.

While model-based methods explicitly track environment dynamics to detect shifts, they are highly memory-intensive, as they store several models with their respective policies. Model-free approaches bypass the need to estimate transition and reward functions, making them less memory-intensive.

The algorithm Non-Stationarity Detector for RL (NSD-RL) [11] uses a change-detection mechanism for non-stationarities in both the transition and/or reward functions, as long as they impact the agent’s performance. It implements a statistical hypothesis test coupled with an importance sampling strategy, which enables off-policy evaluation for comparing the expected reward at two different iterations. The outcome of the statistical test is then analysed using a CUSUM. Since it is a policy-gradient approach, once drift is detected, the adaptation is triggered by resetting the first and second moments of the optimiser, discarding the information accumulated for old dynamics, hence enabling learning of a new policy.

A complementary model-free approach is proposed in [31], which targets value-based and actor-critic RL algorithms instead of policy-gradient ones. It monitors the distributions of the changes in value functions, i.e., ΔV or ΔQ , using a two-sample t-test for independent samples. The intuition is that in a stationary environment, these incremental updates should be identically distributed, so a significant shift in their distribution is a reliable signal of environmental change.

Similarly, CUSUM-RL [42] is based on the stationarity of the optimal Q function, but it works offline. It uses pre-collected historical data to detect changes in Q^* via a CUSUM-based statistical test. When a change point is detected, all the historical data is discarded, and a new policy is trained on the most recent data. This offline design makes it suitable for settings where data collection and policy optimisation are decoupled (e.g., healthcare).

It is worth mentioning Context Q-Learning [49], an adaptation of the well-known Q-learning algorithm to handle environments with multiple contexts. It makes use of state-reward samples, also called experience tuples, to detect context changes using an online parametric Dirichlet change-point [50] algorithm. However, it assumes knowledge of the pattern of context changes, as well as a finite number of contexts. So, when a change is detected, based on the pattern of changes, the Q values are updated for the correct context.

3.1.3. Latent Space Detection

A recent line of work tackles change-point detection within a learned latent representation of the environment, leveraging the idea that a compact latent space can be more responsive to semantic changes than the original raw observations.

A notable approach involves continual state representation learning [12] where the agent monitors the reconstruction loss of a Variational AutoEncoder (VAE). Specifically, it uses Welch’s t-test [61] to identify statistically significant deviations in the loss. Once a change is detected, the method employs a generative replay method to maintain past knowledge, enabling adaptation to the new context while preventing catastrophic forgetting.

Learning Latent and Changing Dynamics (LLCD) [43] uses a Recurrent State-Space Model (RSSM) [28] to embed high-dimensional observations into a more compact latent space. While the standard RSSM assumes stationary dynamics, LLCD adapts it by performing non-stationarity detection directly in this latent space, using a modified version of the online Bayesian change detection [2]. Upon detecting a change, the model adapts the transition dynamics by adding a regularisation term to the RSSM loss function, proportional to the magnitude of the detected drift.

A similar method is Behaviour-Aware Detection and Adaptation (BADA) [44], in which change detection is made in the latent space. Trajectories are collected and embedded into a compact behavioural representation via a neural encoder. The Wasserstein distance is computed between successive windows; when this metric surpasses a data-driven threshold, a change is signalled, and adaptation is triggered. The latter is achieved by adjusting the policy parameters via regularisation proportional to the magnitude of the detected drift.

3.1.4. Performance-Driven Adaptation

Within the model-free paradigm, some approaches do not rely on explicit change-point detectors applied to the environment’s dynamics. Instead, the core idea is to detect or anticipate drift through signals directly tied to the agent’s performance.

An example is GPAction [21], which reframes environmental drift in RL into an error-rate-based drift detection in SL. The method employs Gaussian Process (GP) to predict the current action based on the states observed, i.e., $P(a_t | s_t, \Delta s_t)$, then the Mean Squared Error (MSE) between the predicted and true actions is computed, which is the target error monitored. The increase in MSE suggests a change in the environment dynamics, i.e., $P(\Delta s_t | s_t, a_t)$. It should be noted that GPAction does not propose an adaptation mechanism.

Rather than reacting to a change after it has occurred, Prognosticator [13] focuses on proactive adaptation. It is a model-free policy-gradient algorithm that optimises the policy by targeting its predicted performance in future episodes. By modelling the past policy performance as a time series, it uses a regressor to forecast the future, updating the policy based on this forecast. Unlike traditional algorithms, which optimise policies based on

past data, making them sub-optimal when deployed in non-stationary environments, this approach can effectively adapt to continuous drift. Moreover, no performance degradation occurs if the system is stationary, as Prognosticator reduces to a standard policy-gradient estimator. However, this method relies on the assumption that the performance of any policy changes smoothly over time, without abrupt jumps; therefore, it may fail to adapt in scenarios of abrupt environment shifts.

Algorithm	Architecture	Detection Principle	Adaptation Strategy
RL-CD [8, 16]	Model-based	Model Quality	Context Switching
SCPD-RLCD [27]	Model-based	Statistical Test	Context Switching
Quickest Change Detection [6]	Model-based	Optimal Stopping	Optimal Control adjustment
MBCD [3]	Model-based	Statistical Test	Context Switching
NSD-RL [11]	Model-free	Statistical Test	Optimizer Reset
Context Q-Learning [49]	Model-free	Statistical Test	Context Switching
Value Function Monitoring [31]	Model-free	Statistical Test	Policy Retraining
CUSUM-RL [42]	Model-free	Statistical Test	Policy Retraining
LLCD [43]	Model-based	Latent Shift Detection	Regularization
BADA [44]	Model-free	Latent Shift Detection	Regularization
GPAction [21]	Model-free	Supervised Error	N/A
Prognosticator [13]	Model-free	Time-Series Forecast	Proactive Update

Table 3.1: Taxonomy of active approaches. Summary of state-of-the-art methods that employ explicit drift detection mechanisms to identify and characterise environmental shifts.

3.2. Passive Approaches

Passive approaches continuously adapt the model without explicitly detecting when the drift occurred. This design choice comes with an inherent trade-off: by avoiding the cost of explicit monitoring, passive methods are generally more sample-efficient and also applicable to gradual and continuous changes that may not produce a detectable signal. However, since adaptation is not explicitly triggered, they may suffer from slow adaptation. We organise passive methods into four families based on their core mechanism: latent context inference, optimistic exploration, robust planning, and tracking. The reviewed methods are summarised in Table 3.2.

3.2.1. Latent Context Inference

A fundamental work is represented by Hidden Mode MDP (HM-MDP) [15], a stochastic model for MDPs for non-stationary environments. Formally, an HM-MDP is defined as a finite set of MDPs that share the same state and action space but have possibly different transition and/or reward functions. Each MDP corresponds to a mode, which is not directly observable, and the transition between modes is governed by a Markov chain. It should be noted that HM-MDP are a subclass of POMDPs, so it can always be reformulated as one, though doing so is inefficient. To solve HM-MDPs, a variant of the Baum-Welch algorithm 2.3.3 is proposed, along with a variant of the Bellman equations to calculate the optimal policy.

A modern extension of this concept for continuous control is Lifelong Latent Actor-Critic (LILAC) [63], an off-policy algorithm that operates in non-stationary environments. They formalise the problem with Dynamic Parameter-MDPs (DP-MDPs), a sequence of stationary MDPs related by a set of latent parameters, which, as opposed to POMDPs, account for inter-episode context switching. LILAC extends the SAC algorithm by introducing a variational inference network that outputs a distribution over latent variables for the current episode. Both the actor and the critic are conditioned on this inferred latent variable, allowing the policy to implicitly adapt to the changing dynamics. However, a major drawback of this approach is its monolithic latent representation: there is no difference between the invariant dynamics of the environments and the factors driving non-stationarity. Since these elements are entangled into a single representation, even an isolated environmental shift can perturb the entire representation.

To overcome this limitation, Factored Adaptation for Non-Stationary-RL (FANS-RL) [22] builds upon the ADaptive RL (AdaRL) [30] framework, extending it to non-stationary environments. Rather than encoding all change in a single shared embedding, the latent space is decomposed into individual change factors via a causal graphical structure, so that each component of the non-stationarity is represented separately and the policy is conditioned only on the factors relevant to drift. However, it must be noted that these causal approaches to non-stationary RL rely on strong structural assumptions. As highlighted by [45], the problem of learning uniquely identifiable disentangled factors is generally ill-posed. FANS-RL addresses this by using a causal graph as its inductive bias, but doing so requires that the assumed graph structure correctly reflects the true data-generating process, an assumption that is difficult to verify in practice. In real-world environments where sources of drift may be highly entangled or where the causal structure may itself evolve over time, these guarantees may not hold.

3.2.2. Optimistic exploration

A prominent family of algorithms relies on the “optimism in the face of uncertainty” (OFU) principle. In sequential decision-making problems, this principle states that an agent should act as if the environment is as promising as plausibly possible. This naturally encourages the exploration of under-visited regions, allowing for a degree of passive adaptation.

The core methods are Upper Confidence RL (UCRL) [5] and its episodic improvement UCRL2 [32], which build confidence sets of plausible MDPs using empirical estimates of transitions and rewards. From these sets, the most optimistic model is chosen to drive exploration. Although both algorithms are designed to work within stationary environments, their confidence-set mechanism naturally handles uncertainty.

Variational UCRL2 [48] extends UCRL2 to fully account for non-stationarity by incorporating a variation budget directly into the construction of confidence sets. The algorithm assumes that the cumulative sum of all changes in the transition probabilities and reward functions over a given time horizon is strictly upper-bounded. By widening its confidence bounds in proportion to this budget, the algorithm compensates for old data. However, a major limitation of this approach is the assumption that these variation budgets are bounded and known in advance, which is rarely the case in real-world scenarios.

To address this limitation, Sliding Window UCRL2 with Confidence Widening (SW-UCRL2-CW) [14] maintains a sliding window of the most recent observation upon which the confidence sets are constructed. It incorporates a novel confidence-widening mechanism that introduces extra optimism, allowing it to discard old, obsolete data. Furthermore, the authors propose a Bandit-Over-RL (BORL) wrapper to adaptively tune the algorithm, thus removing the need to know the variation budget in advance and making the method effectively parameter-free with respect to the degree of non-stationarity.

A very recent advancement is Action wise Observation-Aware Spectral-UCRL (AOAS-UCRL) [56], which extends UCRL framework to POMDPs. Assuming the observation model is known, the algorithm introduces an action-wise estimation to infer the underlying transition dynamics. Confidence sets are then constructed directly over these inferred probabilities to drive exploration.

3.2.3. Robust Planning

While optimistic approaches handle environmental uncertainty by encouraging exploration, robust methods take the opposite view: preparing for the worst-case scenario. A notable model-based method in this category is Risk Averse Tree Search (RATS) [40], which relies

on lookahead search trees for optimal planning. It is based on the strong assumptions that the transition and reward functions evolve continuously and slowly over time, a property formalised as Lipschitz continuity. At each decision step, RATS builds a tree of reachable states starting from the current state, using an action based on the best response to the worst-case scenario. Since RATS must build a search tree at every decision step, it is not suitable for large state space problems.

3.2.4. Tracking

A fundamental concept grounding model-free passive adaptation is tracking [58]. In stationary environments, RL algorithms typically employ a decaying learning rate to guarantee convergence to a fixed optimal policy. However, we have already seen that in a non-stationary setting, the optimal policy must be non-stationary too. A static, fixed policy leads to catastrophic forgetting and a loss of plasticity. Therefore, the agent must “track” the shifting optimal policy by balancing the speed of adaptation against the variance of continuous updates.

The Kalman Temporal Difference (KTD) framework [25] formalised value function tracking in non-stationary environments as a filtering problem. It employs non-linear Kalman filtering [37] to continuously track the optimal value function. The intuition lies in explicitly addressing two sources of non-stationarity: the internal non-stationarity due to Generalised Policy Iteration (as the evaluated policy continuously shifts) and the external drift of the environment.

A prominent line of work in this direction adapts the update rule of a well-known RL algorithm so that the agent continuously tracks the shifting optimal policy without the need to maintain past policies.

For instance, Q-Follow the Perturbed Leader (Q-FPL) [64], formulated for MDP where reward may vary over time, operates in discrete learning phases. During each phase, Q -values are computed from the reward samples. These values are then perturbed to choose actions for the subsequent interval using the Follow the Perturbed Leader (FPL) strategy [36] to drive exploration. Finally, to prevent old information from interfering with future decisions, Q -values are zeroed before the next phase begins.

Similarly, Repeated Update QL (RUQL) [1] modifies Q-learning by decoupling update frequency from action selection probability, ensuring that Q -values are updated uniformly across the action space regardless of how often each action is chosen. In doing so, it ensures that Q -values are updated consistently across the entire action space, thereby preventing action starvation and implicitly adjusting for environment drifts.

However, both RUQL and Q-FPL suffer from the same poor learning efficiency typical of standard Q-Learning (QL) in non-stationary environments. Since they only maintain one policy at a time and lack any policy retention mechanism, they are prone to catastrophic forgetting when the environment revisits a previously seen context.

A parallel line of research tackles tracking through policy update scheduling. Rather than modifying the underlying update formulas, these approaches dynamically time when the agent should learn, pause, or completely reset its policy to maintain adaptability and mitigate forgetting.

Following this principle, a recent method [41] demonstrates that suspending policy updates at certain intervals leads to improved performance, as it allows the agent to better handle the aleatoric uncertainty inherent in the environment. They also derive the theoretically optimal ratio between update and hold duration.

Similarly, Non-Stationary Natural Actor-Critic (NS-NAC) [33] combines Natural Actor-Critic updates with a restart-based exploration mechanism that periodically resets the policy. It is the first policy-gradient algorithm for non-stationary MDPs under the average reward setting. The authors also provide a parameter-free variant using the BORL wrapper.

Algorithm	Architecture	Core Principle	Adaptation Strategy
HM-MDP [15]	Model-based	Latent Inference	Belief State Update
LILAC [63]	Model-free*	Latent Inference	Belief State Update
FANS-RL [22]	Model-based	Latent Inference	Continuous Causal Update
Variational UCRL2 [48]	Model-based	Optimistic Exploration	Confidence Bound Update
SW-UCRL2-CW [14]	Model-based	Optimistic Exploration	Confidence Bound Update
AOAS-UCRL [56]	Model-based	Optimistic Exploration	Confidence Bound Update
RATS [40]	Model-based	Robust Planning	Minimax Tree Search
KTD [25]	Model-free	Optimal Tracking	Dynamic Filtering
Q-FPL [64]	Model-free	Optimal Tracking	Periodic Reset
RUQL [1]	Model-free	Optimal Tracking	Update Scheduling
Pausing Policy [41]	Model-free	Optimal Tracking	Update Scheduling
NS-NAC [33]	Model-free	Optimal Tracking	Periodic Reset

* LILAC infers a latent context representation via variational inference, but performs control through SAC, a model-free algorithm.

Table 3.2: Taxonomy of passive approaches. Summary of state-of-the-art methods that handle non-stationarity through intrinsic algorithmic robustness or continuous adaptation, without relying on explicit drift detection.

4 | Problem Formulation

In this chapter, we establish the mathematical framework employed in the rest of this work. First, the environment is formalised as an HM-MDP, highlighting its dependence on unobservable latent contexts, thus framing the non-stationary problem as a POMDP. To resolve this lack of observability, we then present our state augmentation solution, which leverages an HMM to track the environment’s history and infer a sufficient statistic known as the belief. Finally, we define the objective function that guides the agent’s policy optimisation and explicitly state the theoretical assumptions that bound the scope of this work.

4.1. Environment

We formalise the non-stationary environment as modified version of an HM-MDP, defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, A, P, R, \gamma \rangle$ where:

- $\mathcal{S}$ and $\mathcal{A}$ are respectively the state space and the action space, which can be either finite or infinite depending on the specific problem.
- $\mathcal{Z}$ is the set of latent modes ¹, which is assumed to be finite. The cardinality of $\mathcal{Z}$ is assumed to be bounded by a known upper limit N .
- A is the latent mode transition function, i.e. $A(z_{t+1} \mid z_t)$.
- P is the state transition function, i.e. $P(s_{t+1} \mid s_t, a_t, z_t)$.
- R is the reward function, i.e. $R(s_t, a_t) = \mathbb{E}[r_t \mid s_t, a_t]$, where r_t is the immediate scalar reward signal generated by the environment at time t .
- $\gamma \in [0, 1]$ is the discount factor.

At any given time step t , a latent mode z_t alters the environment’s underlying dynamics, making the state transition dynamics mode-dependent. Therefore, the probability of

¹In this work, the terms *mode*, *latent mode*, *hidden mode*, *latent context* and *hidden context* will be used interchangeably to refer to the elements $z \in \mathcal{Z}$.

transitioning to the next state s_{t+1} is conditioned on the current state s_t , the chosen action a_t , and the current latent mode z_t .

In this setup, the agent has full observability of the state and action spaces, but not of the latent modes. Since the latent mode is an unobserved variable, the environment violates the Markov property from the agent's point of view. Thus, the problem can be framed as an POMDP where the unobservable component is the latent mode. As a result, if the agent tries to optimise a policy based solely on the observable state, it would suffer sub-optimality due to perceptual aliasing, as identical state-action pairs may lead to different next states depending on the hidden mode.

4.2. Augmented State Space

To restore the Markov property and allow the use of standard RL algorithms, we approach the problem through *state augmentation*. Specifically, we augment the agent's observation space with a sufficient statistic that summarises the historical trajectory and infers the active latent mode: the belief. As previously introduced for POMDPs in Chapter 2, the belief is the posterior probability distribution over the possible latent modes given the history of observed states and actions:

$$b_t(z) = P(z_t = z \mid s_{1:t}, a_{1:t-1}).$$

To compute this posterior, we model the environment dynamics as an input-conditioned emission HMM, formally defined by the tuple $\langle b_1, \mathbf{A}, B \rangle$, where:

- $b_1 \in \Delta(\mathcal{Z})$ is the initial belief distribution over the latent modes at time $t = 1$.
- $\mathbf{A}$ is the latent transition matrix, representing the probability of transitioning from mode i to mode j , i.e., $\mathbf{A}_{ij} = P(z_{t+1} = j \mid z_t = i)$.
- B is the input-conditioned emission model. Unlike standard HMMs where emissions depend exclusively on the latent state, here the emissions are conditioned on the current state-action pair, i.e., $B(s_{t+1} \mid s_t, a_t, z_t) = P(s_{t+1} \mid s_t, a_t, z_t)$. This design choice is motivated by the need to resolve the ambiguity introduced by mode-dependent dynamics: without action-conditioning, identical state-action pairs could generate different successor states under different latent modes, making the emission distribution uninformative as a discriminative signal for latent mode inference.

Given this model, the belief can be recursively updated at each timestep through online Bayesian filtering. When a new state s_{t+1} is observed after taking action a_t from state s_t ,

the updated posterior belief b_{t+1} is computed from b_t via standard HMM filtering equations. This inferred belief b_t acts as a sufficient statistic for the history of the environment, thus reducing the POMDP to a fully observable belief MDP. In practice, at each timestep the agent receives the augmented observation $x_t = (s_t, b_t)$, and acts according to a policy $\pi(a_t | s_t, b_t)$ conditioned on this augmented state. It is worth noting that dealing with the full continuous belief often introduces significant computational overhead and exploration challenges in standard RL algorithms. For this reason, in the actual implementation, the control policy is conditioned on a compact representation of the belief.

4.3. Objective Function

As we have already seen in Chapter 2, the ultimate goal of the agent is to learn a policy π that maximises the expected cumulative discounted reward, also referred to as the expected return. Given the MDP formalised above, we define the objective function as:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \gamma^t r_t \right]$$

where τ is the trajectory of states and actions experienced by the agent during an episode, r_t is the immediate reward, $\gamma \in [0, 1]$ is the discount factor, and T is the time horizon.

4.4. Assumptions

To bound the problem addressed in this work, we introduce a set of assumptions regarding the environments and the agent's capabilities.

- **Finite latent modes:** We assume that the non-stationarity is driven by a discrete and finite set of latent modes.
- **Latent Markovian dynamics:** The transition sequence of the hidden modes z_t is assumed to follow a first-order Markov chain. This implies that the future latent state z_{t+1} depends entirely on the current latent state z_t , making the transition dynamics between modes history-independent. Furthermore, the latent Markov chain is assumed to be ergodic (i.e., irreducible and aperiodic). This guarantees that no latent mode acts as an absorbing state, reflecting the continuous, dynamic nature of the environment's non-stationarity.
- **Fully observable physical state:** The partial observability only affects the latent modes, while the state variables of the environment are fully observable and noise-free.

- **Passive non-stationarity:** We assume that the non-stationarity is an exogenous problem, i.e., the transitions between latent modes are independent of the agent's actions. We do not cover settings in which the agent drives non-stationarity, nor adversarial settings where the environment intentionally switches modes specifically to minimise the agent's expected return.
- **Mode-independent reward:** The reward function $R(s_t, a_t)$ depends only on the state-action pair and not on the latent mode. Therefore, non-stationarity manifests solely through the transition dynamics $P(s_{t+1} \mid s_t, a_t, z_t)$.

In other words, these assumptions imply that the perceived non-stationarity is just a byproduct of partial observability. An agent that only observes the state s_t perceives the environment as non-Markovian and unpredictable. Augmenting the state space with the inferred belief b_t resolves this ambiguity by explicitly accounting for the hidden mode.

5 | Methodology

In this chapter, we present the overall architectural design, which separates latent mode estimation from policy optimisation into two distinct components. The latent mode estimator is described first, focusing on the input-conditioned Baum-Welch algorithm used to learn its parameters offline. Next, we detail the online state augmentation procedure, which leverages the pre-trained latent context estimator via Bayesian filtering to enable the use of standard, off-the-shelf RL algorithms for control.

5.1. Architectural Overview

As presented in Chapter 4, the non-stationary environment is modelled as a POMDP in which the hidden context $z_t \in \mathcal{Z}$ governs the transition dynamics. Since these latent modes shift over time, an RL agent should simultaneously learn the control optimisation task while identifying the underlying environmental context. However, standard RL agents generally struggle to isolate these problems, leading to conflicting objectives that manifest as policy instability and, ultimately, suboptimal performance.

We therefore propose a framework that decouples latent context tracking from policy optimisation into two components, as illustrated in Figure 5.1:

1. **Offline latent model learning:** We employ a modification of the Baum-Welch algorithm to account for conditioning the emission on the current state-action pair. Using a static historical dataset $\mathcal{D}$ of trajectories, the parameters λ of the latent estimator are learned.
2. **Online context-conditioned control:** The pre-trained latent mode estimator is embedded into the non-stationary environment as a real-time context tracking module. Every transition (s_t, a_t, s_{t+1}) is mapped to a latent state prediction $\hat{z}_{t+1}$, which is used to construct an augmented observation $x_t = [s_{t+1}, \hat{z}_{t+1}]$ passed to the RL agent.

This two-phase structure, offline latent model learning followed by online policy optimisation, allows us to decouple the environment’s complexity from the agent’s decision-making

process. By augmenting the original non-stationary environment with the estimated hidden mode, we effectively reframe the POMDP into an MDP, allowing for the use of standard RL algorithms for policy optimisation. In this architecture, the quality of the latent estimate becomes the primary driver of the agent’s performance, making it the foundational prerequisite for achieving reliable and stable control.

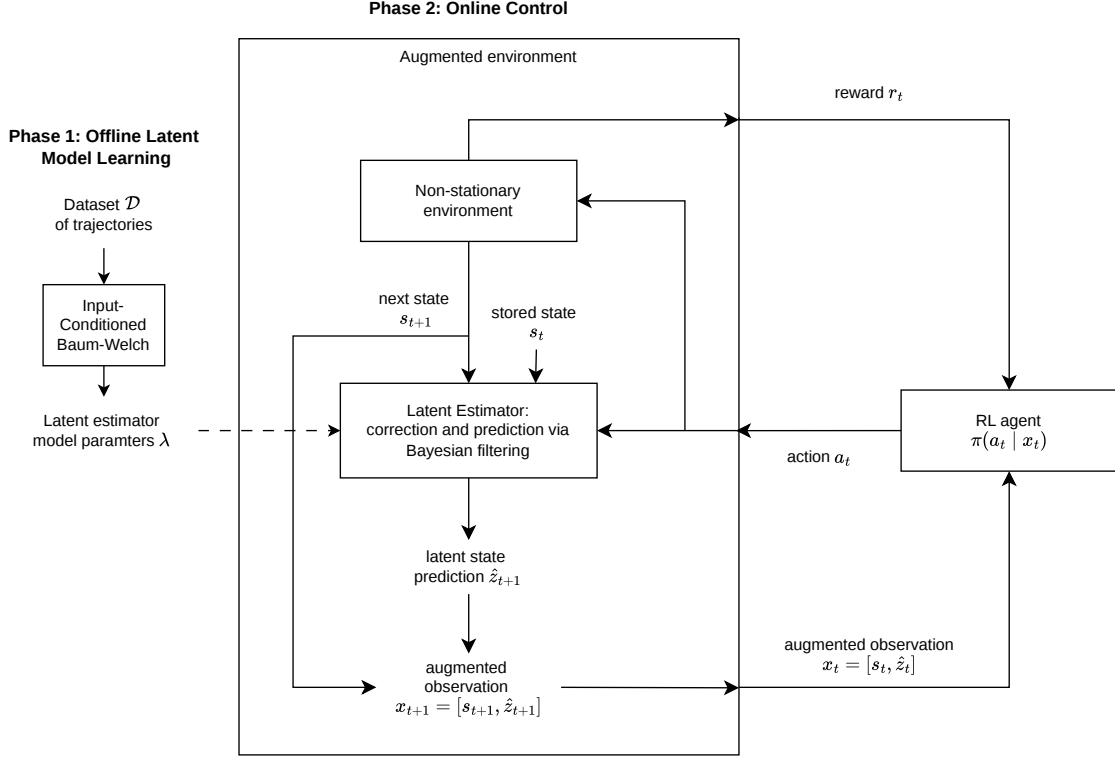


Figure 5.1: Architectural overview of the proposed latent-conditioned framework.

The framework is composed of two phases: an offline phase, where the input-emission conditioned Baum-Welch algorithm processes historical trajectories $\mathcal{D}$ to identify the latent model parameters λ , and an online control phase. During deployment, the frozen model is embedded within an augmented environment wrapper to perform real-time Bayesian filtering using the incoming transition (s_t, a_t, s_{t+1}) , providing an augmented observation x_{t+1} that enables the RL agent to adapt its policy to non-stationary environmental shifts.

5.2. Latent Mode Estimator

The latent mode estimator proposed builds upon the input-conditioned emission HMM formalised in Chapter 4. Since the set of latent contexts $\mathcal{Z}$ is assumed to be discrete and finite, the latent transition model $\mathbf{A}(z_{t+1} | z_t)$ is represented by a matrix, whose entry

$\mathbf{A}(z' | z)$ denotes the probability of transitioning from mode z to mode z' .

On the other hand, since the emission model outputs the next state of the environment, the parametrisation of $B(s_{t+1} | s_t, a_t, z_t)$ is dictated by the nature of the environment's state space $\mathcal{S}$. When $\mathcal{S}$ is finite and discrete, the emission model is defined as a categorical probability distribution over a finite set of next states. In this case, every individual (s_t, a_t, s_{t+1}) tuple can be mapped to a distinct probability mass, without loss of information. Conversely, when the specific problem involves a continuous state space, the emission model is represented as a continuous probability density function as the cardinality of $\mathcal{S}$ becomes infinite. In this case, the algorithm relies on function approximation, leveraging deep neural networks to approximate the continuous dynamics by outputting the statistical parameters that define the next-state density distribution.

In both cases, the parameter optimisation is made using a static dataset $\mathcal{D}$ of historical interaction sequences $\tau = (s_1, a_1, s_2, a_2, \dots, s_T)$ collected from the non-stationary environment using a random policy, via an adaptation of the classical Baum-Welch algorithm extended to handle input-conditioned emissions.

5.2.1. Tabular Input-Conditioned Baum-Welch

Let us consider an environment characterised by finite, discrete state and action spaces. In this case, the emission dynamics can be exactly parametrised using tabular data structures. In practice, the emission model $\mathbf{B}$ is stored as an $N \times |\mathcal{S}| \times C$ tensor, where $N = |\mathcal{Z}|$ is the number of hidden states and $C = |\mathcal{S}| \cdot |\mathcal{A}|$ is the dimension of the state-action pair.

During the expectation step (E-step), the forward variable $\alpha_t(z)$ and the backward variable $\beta_t(z)$ are calculated recursively over the dataset, with a modification to account for the conditioning over the state-action pair.

The forward variable represents the joint probability of experiencing the partial trajectory up to step t and landing in latent state z :

$$\begin{aligned} \alpha_1(z) &= b_1(z) \mathbf{B}(s_2 | s_1, a_1, z), \\ \alpha_t(z) &= \mathbf{B}(s_{t+1} | s_t, a_t, z) \sum_{z' \in \mathcal{Z}} \alpha_{t-1}(z') \mathbf{A}(z | z'), \quad \text{for } 1 < t \leq T-1. \end{aligned} \quad (5.1)$$

The backward variable computes the probability of the future transition sequence given

the latent mode at step t :

$$\begin{aligned}\beta_{T-1}(z) &= 1, \\ \beta_t(z) &= \sum_{z' \in \mathcal{Z}} \mathbf{A}(z' | z) \mathbf{B}(s_{t+2} | s_{t+1}, a_{t+1}, z') \beta_{t+1}(z'), \quad \text{for } 1 \leq t < T-1.\end{aligned}\quad (5.2)$$

From these variables, the marginal posterior probability $\gamma_t(z)$ of being in mode z at step t , and the joint posterior probability $\xi_t(z, z')$ of transitioning from mode z to z' between steps t and $t+1$, are computed as:

$$\gamma_t(z) = \frac{\alpha_t(z) \beta_t(z)}{\sum_{k \in \mathcal{Z}} \alpha_t(k) \beta_t(k)}, \quad (5.3)$$

$$\xi_t(z, z') = \frac{\alpha_t(z) \mathbf{A}(z' | z) \mathbf{B}(s_{t+2} | s_{t+1}, a_{t+1}, z') \beta_{t+1}(z')}{\sum_{k \in \mathcal{Z}} \sum_{w \in \mathcal{Z}} \alpha_t(k) \mathbf{A}(w | k) \mathbf{B}(s_{t+2} | s_{t+1}, a_{t+1}, w) \beta_{t+1}(w)}. \quad (5.4)$$

In the maximisation step (M-step), the model parameters are updated by averaging these posteriors across the dataset. The latent transition matrix $\mathbf{A}$ is re-estimated via state-frequency counting, so the (z, z') -th entry becomes:

$$\hat{a}_{z,z'} = \frac{\sum_{t=1}^{T-2} \xi_t(z, z')}{\sum_{t=1}^{T-2} \gamma_t(z)}. \quad (5.5)$$

Note that $\xi_t(z, z')$ is defined for $t = 1, \dots, T-2$, since it requires β_{t+1} . The transition matrix update, therefore, sums over the same range.

Unlike the standard Baum-Welch algorithm, where the emission depends only on z and the observation, here the input-conditioned emission tensor $\mathbf{B}$ is indexed by the full state-action-successor tuple. The (s, a, z, s') -th entry is re-estimated using only those timesteps whose observed transition matches (s, a, s') :

$$\hat{b}_{s,a,z,s'} = \frac{\sum_{t: s_t=s, a_t=a, s_{t+1}=s'} \gamma_t(z)}{\sum_{t: s_t=s, a_t=a} \gamma_t(z)}. \quad (5.6)$$

For completeness, the pseudocode for this algorithm is provided in Appendix A.2.

5.2.2. Neural Input-Conditioned Baum-Welch

Let us now consider an environment characterised by continuous state and action spaces. In this setting, the tabular parametrisation of the emission model is no longer feasible,

since the cardinality of $\mathcal{S}$ is infinite. We therefore make use of a neural network B_θ to approximate $B(s_{t+1} \mid s_t, a_t, z_t)$, while keeping the discrete, tabular formulation for the latent transition model $\mathbf{A}$ and the initial distribution b_1 . We denote the resulting parameter set as $\lambda_\theta = \langle \mathbf{A}, B_\theta, b_1 \rangle$.

The emission network B_θ takes as input a vector (s_t, a_t) and outputs, for each latent state $z \in \mathcal{Z}$, the parameters of a Gaussian distribution over the next observation. Assuming independence across observation dimensions, the emission probability can be factorised as:

$$B_\theta(s_{t+1} \mid s_t, a_t, z) = \prod_{i=1}^d \mathcal{N}(s_{t+1}^{(i)}; \mu_\theta^{(i)}(s_t, a_t, z), \sigma_\theta^{(i)}(s_t, a_t, z)). \quad (5.7)$$

where $\mu_\theta, \sigma_\theta$ are the outputs of a shared multi-layer perceptron with separate linear heads.

Parameter optimisation proceeds via the EM algorithm. The E-step is identical to the tabular case: the forward-backward recursion is executed using B_θ in place of the tabular entries, yielding the posterior marginals $\gamma_t(z)$ and $\xi_t(z, z')$. On the other hand, in the M-step optimisation, the emission parameters θ are estimated by gradient ascent maximising the log-likelihood over the latent state assignment sampled according to the probabilities inferred during the E-step. Specifically, at each iteration, a hidden mode $\hat{z}_t$ is sampled from the current posterior γ_t , and the emission network is updated by maximising:

$$\mathcal{L}_{\text{emit}}(\theta) = \frac{1}{T} \sum_{t=1}^{T-1} \log B_\theta(s_{t+1} \mid s_t, a_t, \hat{z}_t), \quad \hat{z}_t \sim \gamma_t(\cdot), \quad (5.8)$$

via Adam for a fixed number of gradient steps. The use of hard sampling of a latent mode is introduced as exact marginalisation would require evaluating the network and accumulating backpropagation gradients for all possible latent modes at every timestep. In this way, the computational and memory requirements of the backward pass are significantly reduced.

For completeness, the pseudocode for this algorithm is provided in Appendix A.3.

5.2.3. Training Considerations

Since the Baum-Welch algorithm is an EM algorithm, it is guaranteed to converge only to a local maximum of the likelihood. As a result, the quality of the solution depends on the initialisation of the model parameters, which must be carefully chosen to avoid degenerate solutions.

Initialisation The initial distribution b_1 is concentrated on a single mode, i.e.,

$$b_1(z) = \begin{cases} 1 & \text{if } z = 0 \\ 0 & \text{otherwise,} \end{cases}$$

which accounts for the fact that all episodes begin from a known, fixed latent mode.

The latent transition matrix $\mathbf{A}$ is initialised as a uniform distribution perturbed by small noise to break symmetry, followed by row-wise normalisation. In contrast, a fully uniform initialisation would impose equal transition probabilities across all successor states, which is an unrealistic assumption for most latent dynamics and would prevent the algorithm from identifying meaningful structure during training.

The initialisation of the emission model depends on the nature of the problem. In the tabular case, the emission tensor $\mathbf{B}$ is initialised at random, drawing each entry independently from a uniform distribution over $[0, 1]$, followed by normalisation along the observation axis to ensure that each input-conditioned emission sums to one. In the neural case, the emission network B_θ is initialised using the default PyTorch weight initialisation scheme for linear layers, which provides an adequate degree of symmetry breaking without any additional intervention.

Convergence Convergence is monitored via the log-likelihood of the dataset $\mathcal{D}$ under the current parameters λ , computed by marginalising the forward variable across all M independent trajectories contained in the dataset:

$$\log P(\mathcal{D} \mid \lambda) = \sum_{m=1}^M \log \sum_{z \in \mathcal{Z}} \alpha_{T-1}^{(m)}(z).$$

This exact formulation applies identically to the neural setting to evaluate the dataset log-likelihood under the parameters λ_θ .

In the tabular case, the EM algorithm guarantees a monotonic increase of the log-likelihood at each iteration. Consequently, training is stopped early whenever the improvement between two consecutive iterations falls below a tolerance of 10^{-4} , with a maximum limit of 2000 iterations.

In the neural case, this monotonicity guarantee no longer holds, since the emission parameters θ are updated via gradient ascent on stochastically sampled latent assignments $\hat{z}_t$, rather than in closed form. The same stopping criterion is applied, but the parameters yielding the highest log-likelihood observed during training are retained, rather than those

from the final iteration.

5.3. Context-Conditioned Policy Optimization

Once the parameters of the input-conditioned HMM (either the tabular λ or the neural λ_θ) are learned offline, they are employed to perform real-time context tracking: the framework transitions from offline learning to online control. The frozen model is embedded within an environment wrapper to act as a real-time context tracker. By continuously processing the incoming transitions, it provides the agent with an augmented state representation, allowing the deployment of standard, off-the-shelf RL algorithms without requiring any modifications.

The optimal augmented state representation should be $x_t = (s_t, b_t)$, where the belief b_t is an N -dimensional probability vector. However, in practice, this would introduce significant challenges due to the curse of dimensionality during exploration. To maintain a compact state representation, we perform a dimensionality reduction of the belief by extracting a scalar latent sample $\hat{z}_t$, which represents the expected latent state.

During the online control phase, at each timestep t , the agent observes the augmented state x_t , containing the forecast $\hat{z}_t$, and selects an action based on its policy $a_t \sim \pi(a_t | x_t)$. The wrapper intercepts a_t , forwards it to the non-stationary environment, and receives the next state s_{t+1} and the reward r_t . This transition (s_t, a_t, s_{t+1}) is immediately fed to the latent estimator, which executes Bayesian filtering using the frozen model λ or λ_θ to compute the prediction $\hat{z}_{t+1}$ for the subsequent step.

5.3.1. Bayesian Filtering

The latent estimator operates sequentially. At the start of the episode ($t = 1$), it initialises the prior belief using the learned initial distribution $\bar{b}_1(z) = b_1$. At each timestep, the estimator receives the tuple (s_t, a_t, s_{t+1}) and executes a two-step update:

- **Correction:** The observed transition (s_t, a_t, s_{t+1}) is used to refine the prior belief $\bar{b}_t$ into a posterior b_t , via the Bayes' rule:

$$b_t(z) = \frac{B(s_{t+1} | s_t, a_t, z) \bar{b}_t(z)}{\sum_{k \in \mathcal{Z}} B(s_{t+1} | s_t, a_t, k) \bar{b}_t(k)},$$

which guarantees that $\sum_z b_t(z) = 1$.

- **Prediction:** The posterior belief b_t is propagated through the transition matrix to

obtain the prior predictive belief for the next timestep:

$$\bar{b}_{t+1}(z') = \sum_{z \in \mathcal{Z}} \mathbf{A}(z' | z) b_t(z).$$

The resulting $\bar{b}_{t+1}$ serves simultaneously as the prior for the correction step at time $t + 1$ and as the source from which the scalar latent estimate $\hat{z}_{t+1}$ is extracted.

By using the predictive distribution $\bar{b}_{t+1}$, the estimator shifts from reactive tracking to proactive forecasting, providing a crucial temporal advantage for the control policy.

This two-step procedure is necessary to maintain meaningful tracking over time. If the estimator were to rely solely on the prediction step, performing open-loop forecasting without environmental feedback, the repeated multiplication by the transition matrix $\mathbf{A}$ would cause the belief to rapidly converge to the stationary distribution of the latent Markov chain, since the chain is ergodic. By incorporating the correction step, the predictive belief is grounded in the actual observed transition, preventing the distribution from collapsing into an uninformative prior and allowing the estimator to dynamically track environmental shifts as they unfold.

5.4. Numerical Stability

The recursive nature of both the forward-backward algorithm and the online Bayesian filter requires specific countermeasures to ensure numerical stability, as they involve iterative products of probability values that can be arbitrarily small. Moreover, for long sequences such as complete episodes, floating-point arithmetic may lead to numerical underflow, causing the intermediate variables α_t , β_t , and b_t to vanish before convergence is reached.

To address this problem, all computations are performed in log-probability space. Products of probabilities become sums of log-probabilities, and marginalisation over a discrete set is computed via the numerically stable *log-sum-exp* identity:

$$\log \sum_k \exp(x_k) = \max_k x_k + \log \sum_k \exp(x_k - \max_k x_k),$$

which avoids both overflow and underflow by centring the exponentials on the largest term before summation.

An additional numerical safeguard is applied during online Bayesian filtering. At deployment time, the agent may visit state-action pairs that are under-represented or absent from the offline training dataset $\mathcal{D}$. In such cases, the emission model may assign near-zero

probabilities to the observed transition, producing log-likelihoods that approach $-\infty$ and causing the belief to become numerically degenerate. To prevent this, the per-state log-likelihood $\ell_t(z) = \log B(s_{t+1} \mid s_t, a_t, z)$ is clamped to a lower bound of -20 before the correction step. Since $e^{-20} \approx 2 \times 10^{-9}$, this threshold is negligible under normal operating conditions but sufficient to ensure that no latent state is ever assigned strictly zero posterior mass due to distribution shift between training and deployment.

6 | Experimental Settings

In this chapter, we present the experimental environments developed to assess the capabilities of our proposed framework. Firstly, we describe the simulation environments, along with the introduced non-stationarity. We then report the experimental results, comparing our latent estimator against two baselines: a standard RL agent, representing the lower bound of performance, and an oracle agent, which serves as the theoretical upper bound.

6.1. Environments

We extend the classic *Cliff Walking* environment into two non-stationary variants, operating in discrete and continuous domains, respectively. In both versions, the transition dynamics are influenced by a latent factor, represented as wind, that influences the effect of the agent’s action.

The base environment is the standard implementation provided by OpenAI Gym [59]. It consists of a grid in which the agent must navigate to reach a goal while avoiding the cliff. The agent receives a reward of -1 for each step, -100 if it falls into the cliff (resulting in a reset to the initial state), and $+10$ upon reaching the goal. The original environment terminates when the agent reaches the goal.

6.1.1. Cliff Walking Wind

To introduce non-stationarity, we define a set of latent states $\mathcal{Z}$ with cardinality $|\mathcal{Z}| = 5$, corresponding to the four cardinal wind directions plus a windless mode. At each timestep, a different latent mode may be active, and its temporal evolution follows a cyclic pattern. The active wind mode acts as an additive perturbation to the agent’s actions. Therefore, the agent’s position is updated through a purely kinematic equation: the new position is computed by summing the current position, the chosen action, and the active wind, followed by a clipping operation to enforce the map’s boundary constraints.

In this environment, the agent has sufficient space to counteract the wind, which results in an optimal policy that simply pushes against the walls. To mitigate this exploit, we

increase the complexity of the task by designing a narrow safe path that the agent must traverse to minimise negative rewards. Deviating from this path into the rest of the map incurs the same severe penalty as falling off the cliff. The final grid is represented in Figure 6.1.

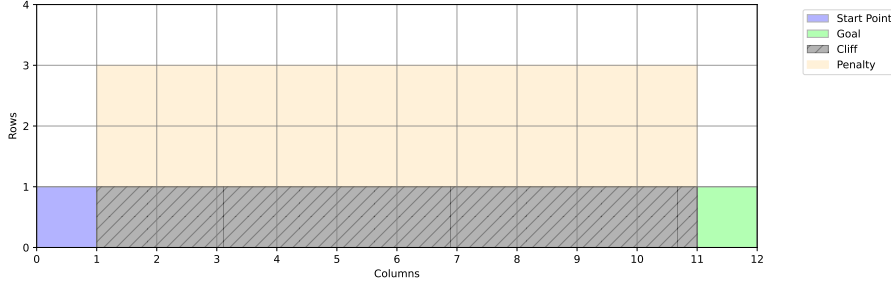


Figure 6.1: Visual representation of the Discrete Cliff Walking Wind environment. The agent starts at the bottom-left cell (blue) and must navigate to the goal (green) at the opposite end. The cliff (grey) along the bottom edge and the penalty zones (yellow) restrict the navigable area. This layout explicitly forces the agent to traverse the safe corridor.

For computational simplicity, we limit each episode in our simulations to a finite horizon of 500 timesteps. Consequently, the optimal policy requires the agent to reach the goal state without stepping into the penalty area and, upon arrival, to actively counteract the wind to remain within the goal for the remainder of the episode.

Since this problem has discrete state and action spaces, we employ a tabular version of the standard Q -learning algorithm. To effectively manage the exploration-exploitation trade-off, an ϵ -greedy strategy is employed. The exploration parameter is initially set to 1.0, its maximum, ensuring high exploration at the beginning of the learning process. This value then linearly decreases until it reaches a final, constant lower bound of 0.05, allowing the agent to exploit its learned policy while maintaining a minimal level of random exploration. The full list of hyperparameters used for this algorithm is reported in Appendix A.3.

6.1.2. Continuous Cliff Walking Wind

We adapt the aforementioned problem into continuous state and action spaces. The agent’s position is defined by coordinates $(x, y) \in [0, 12] \times [0, 4]$, while the action is a two-dimensional vector (dx, dy) , in which each component takes values in $[-1, 1]$. As in the discrete environment, the latent variable is a finite set of wind modes and acts as an additive perturbation to the agent’s actions. However, the magnitude of the wind

perturbation is bounded at a maximum value of 0.5, making the agent’s action sufficiently powerful to overcome the environmental perturbation.

To address the same boundary-exploitation issue observed in the discrete domain, we implement a similar safe path constraint, forcing the agent to actively learn a wind-compensation strategy rather than passively exploiting the walls. The final environment is represented in Figure 6.2. Given the increased complexity compared to the discrete task, we employ a dense reward signal to facilitate exploration. In addition to the standard step (-1) and terminal rewards (+10), the agent receives a bonus proportional to the progress made toward the goal, measured as the reduction in Euclidean distance at each step. This shaping encourages the agent to navigate the safe corridor and discover the goal state.

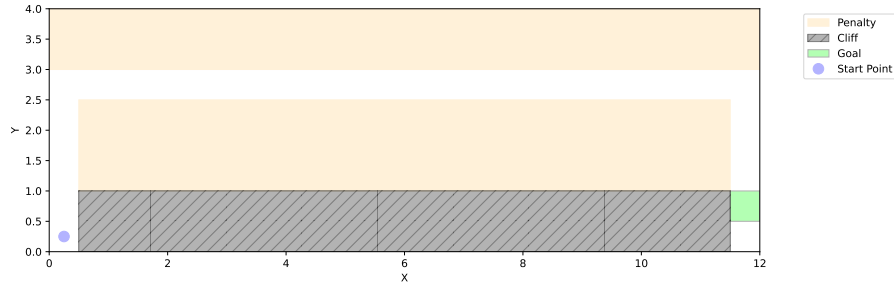


Figure 6.2: Visual representation of the Continuous Cliff Walking Wind environment. The agent starts at the bottom-left position (blue) and must navigate to the goal area (green) at the opposite end. The cliff (grey) along the bottom edge and the penalty zones (yellow) restrict the navigable area. This layout explicitly forces the agent to traverse the central safe corridor, preventing it from exploiting the map boundaries to counteract the wind.

As in the discrete setting, we limit each episode in our simulations to a finite horizon of 500 timesteps. Consequently, the optimal policy remains the same since the agent is subject to the same win conditions.

To solve this continuous problem, we employ the SAC algorithm utilising the implementation offered by the Stable-Baselines3 library [55]. Given the complexity of navigating a continuous space subject to shifting latent winds, robust exploration is critical. To this end, we leverage SAC’s intrinsic maximum-entropy framework, configuring the entropy coefficient to be automatically tuned, thereby ensuring a dynamic balance between exploration and exploitation during training. Furthermore, an initial warm-up phase consisting of 40,000 purely random steps is performed to populate the initial replay buffer with a diverse set of transitions before any gradient updates. To facilitate smooth, temporally correlated exploration, we additionally inject Ornstein-Uhlenbeck action noise. The full

configuration used for this algorithm is reported in Appendix A.3.

6.2. Latent Dynamics

Before delving into the results, we detail the temporal evolution of the latent wind. To evaluate the framework’s inference capabilities under varying degrees of environmental uncertainty, we model the latent transition dynamics according to a stochastic process. As represented in Figure 6.3, at each timestep there is a probability p that the current mode changes and a corresponding probability $1 - p$ that the current mode persists (represented by self-loops). In our experiments, we evaluate the framework against a deterministic setting, where $p = 1$, and a stochastic one, where $p = 0.9$, which represents a more complex scenario.

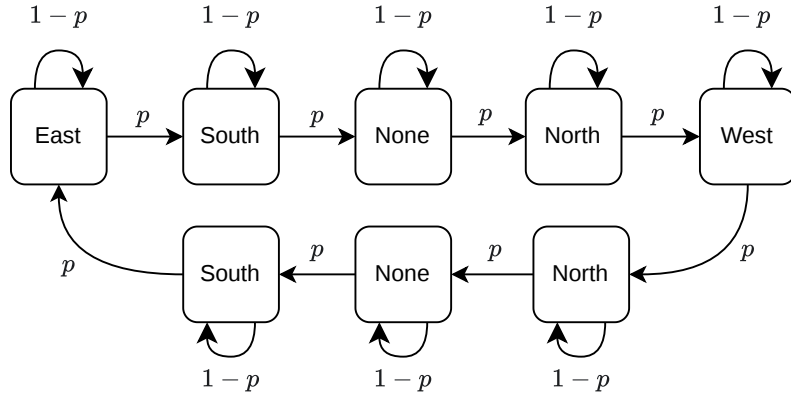


Figure 6.3: Visual representation of the latent chain. The environment cycles through a fixed sequence of wind modes, in order: East, South, None, North, West, North, None, and South.

It is crucial to note that while the environment exhibits only five distinct physical wind modes (East, South, West, North, and None), the underlying transition model must be structured using eight latent states. This state space expansion is necessary to preserve the Markovian property of the transition chain. If we were to use only five states, a mode such as None would have ambiguous outward transitions: it could transition to either North or South depending on the previous step. Such ambiguity would violate the Markov assumption, which strictly requires that the next state depend solely on the current one. By expanding the chain to eight distinct states, each state uniquely determines its successor, allowing the sequence to be perfectly deterministic and memoryless.

6.3. Results

To accurately evaluate the performance of our proposed framework, we establish an upper bound through an Oracle baseline that we construct to mimic our agent’s exact architecture. Specifically, the Oracle employs the same latent estimator mechanism to compute the belief state over the latent modes. However, instead of learning the HMM parameters from collected data, the Oracle’s estimator is explicitly parametrised with the ground-truth transition and observation models of the environment. Conversely, a lower bound is established using a standard agent trained on the observable state space without any latent inference mechanism, thus making it vulnerable to the environment’s non-stationarity. Our proposed architecture, which learns a latent estimator model to infer the belief state, aims to bridge the performance gap between the naive standard agent and the optimal Oracle.

To train the latent estimator model, we use a static dataset of transitions gathered using a purely random behaviour policy. This ensures an unbiased coverage of the state-action space and prevents overfitting to the specific trajectory distribution of a policy. The dataset is generated by running the random agent for 200 independent episodes.

In the subsequent section, the results are presented in terms of cumulative rewards. The curves reflect the agent’s performance during independent evaluation rollouts, rather than during the active training phase. Specifically, at each evaluation step, the performance is computed over 5 independent episodes, and the final curves are averaged across 5 different random seeds. This ensures that the cumulative rewards capture the policy’s exploitation, free from the noise and suboptimal choices introduced by exploratory actions. The solid lines represent the mean cumulative reward across these runs, while the shaded regions denote the 95% confidence intervals.

6.3.1. Discrete Domain

We begin our empirical evaluation within the discrete control domain, which represents a fundamentally simpler task compared to its continuous counterpart. This reduced complexity provides an ideal, controlled simulation to isolate and validate the core inference capabilities of the proposed framework.

Deterministic Latent Dynamics Figure 6.4 illustrates the performance of the aforementioned agents in the discrete environment under deterministic dynamics.

The standard agent fails to converge to any meaningful policy throughout training,



Figure 6.4: Cumulative Reward in the Discrete Deterministic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator.

plateauing at a cumulative reward of -500 , consistent with a policy that never reaches the goal. This result confirms that, without access to the latent mode, the agent cannot resolve the perceptual aliasing induced by the wind and is therefore unable to learn a goal-reaching policy. In contrast, both the Oracle and the proposed Latent Estimator converge to high-performing policies. The Oracle achieves a mean cumulative reward of 4756 at convergence, establishing the performance upper bound. The Latent Estimator reaches 4413.40, closing the gap with the Oracle and substantially outperforming the standard agent. The slight performance difference naturally reflects the intrinsic uncertainty of the inferred belief state, as opposed to the Oracle’s perfect, deterministic knowledge of the mode. A summary of the performance metrics, averaged over the last 100 evaluation episodes, is provided in Table 6.1.

Agent Architecture	Mean Reward	Standard Deviation
Oracle	4756.00	± 9.75
Latent Estimator (Proposed)	4413.40	± 169.77
Standard Agent	-500.00	± 0.00

Table 6.1: Convergence Performance in the Discrete Deterministic Domain.

We evaluate the quality of the latent estimator inference by making a direct comparison between the HMM’s predictions and the underlying Oracle modes. To ensure our analysis strictly reflects the estimator’s steady-state performance at convergence, we aggregate data exclusively from the final 100 evaluation episodes across all 5 independent random seeds. Furthermore, because the HMM outputs a continuous probability distribution (the belief state) over the latent states, we apply a confidence threshold of 0.85 to filter the inferred sequence, thus isolating the model’s high-certainty predictions.

Wind Mode	Total Samples	Accuracy (%)	Assigned HMM States
None	62,048	95.42	0, 1
North	61,836	80.29	3, 5
South	61,678	79.89	2, 4
West	31,050	78.87	6
East	31,004	77.44	7
Overall	247,616	83.45	-

Table 6.2: Estimator Accuracy Report in the Discrete Deterministic Domain. Evaluated on high-confidence transitions.

As detailed in Table 6.2, the overall estimation accuracy is 83.45%, compared to the perfect inference (100% accuracy) of the oracle baseline. The per-mode breakdown reveals that the None wind condition achieves the highest accuracy (over 95%). The lower accuracy on East and West modes (around 77 – 78%) reflects their reduced frequency in the dataset, which is directly tied to the structure of the latent chain: East and West appear only once per full cycle, resulting in half the available training samples compared to the North, South, and None modes, which appear twice.

Furthermore, the state-to-wind mapping reveals that the estimator correctly captured the cyclical nature of the deterministic chain. Physical modes that appear twice in the cycle (such as None, North, and South) are assigned to two distinct latent states (e.g., states 0 and 1 for None), demonstrating the model’s ability to disambiguate identical observations.

Stochastic Latent Dynamics We extend our analysis by introducing stochastic latent transitions ($p = 0.9$) to evaluate the framework’s robustness under environmental uncertainty. Figure 6.5 illustrates the mean cumulative reward curves for the discrete domain under these conditions.

As in the deterministic case, the standard agent fails to converge to any meaningful policy, plateauing at a cumulative reward of -495.62 . Both the Oracle and the Latent Estimator converge to positive cumulative rewards, confirming that latent mode inference remains

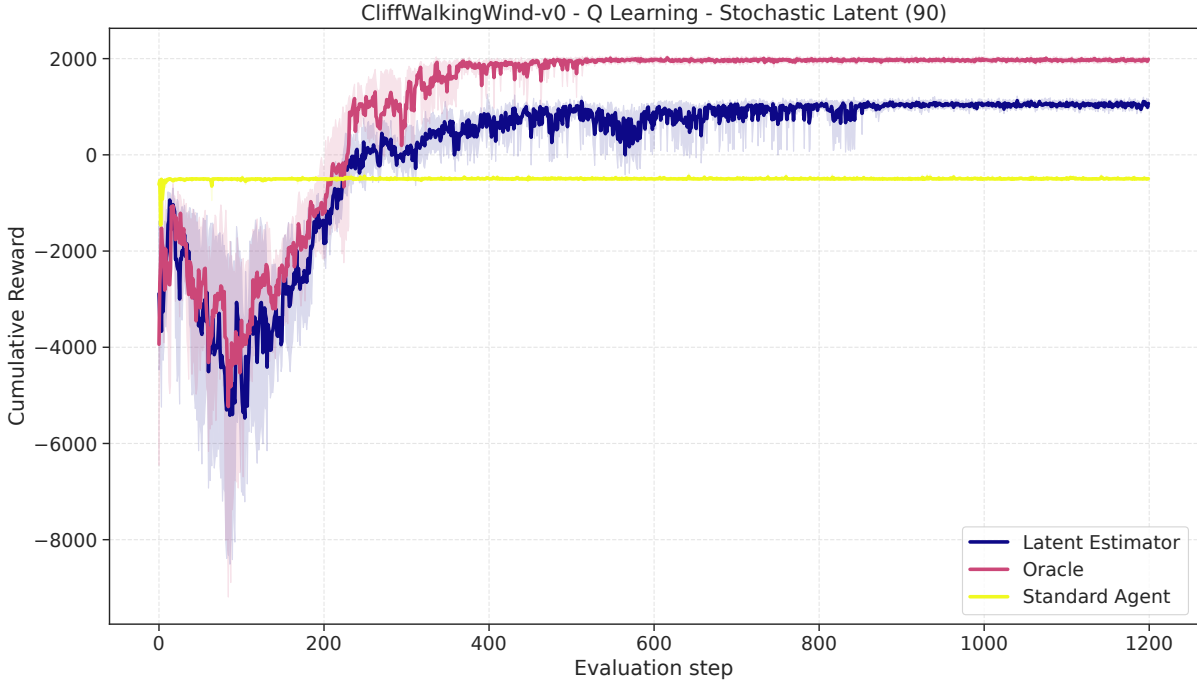


Figure 6.5: Cumulative Reward in the Discrete Stochastic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator.

effective even under stochastic dynamics. However, the performance gap between the two widens considerably: the Oracle achieves 1969.33, while the Latent Estimator reaches 1039.08. A detailed summary of the convergence metrics is provided in Table 6.3.

To investigate the source of this performance gap, we conduct two ablation studies in which one component of the learned model is replaced by its ground-truth counterpart. When the transition matrix $\mathbf{A}$ is learned and the emission matrix $\mathbf{B}$ is set to its true value, the agent achieves 1641.08, recovering a significant portion of the Oracle’s performance. Conversely, when $\mathbf{A}$ is set to its true value and $\mathbf{B}$ is learned, performance drops to -356.78 , close to the standard agent baseline. This asymmetry identifies the emission matrix $\mathbf{B}$ as the primary bottleneck of the learned estimator. We speculate that the factors inducing this degradation are twofold. First, the tabular nature of $\mathbf{B}$ requires each tuple (s, a, z, s') to be observed a sufficient number of times in the offline dataset to produce an accurate estimate, which becomes increasingly difficult to satisfy as the state-action-latent space grows. Second, in the deterministic case, the near-deterministic structure of $\mathbf{A}$ provided a strong corrective signal that partially compensated for imprecision in $\mathbf{B}$, which is not the case in stochastic settings, as $\mathbf{A}$ itself introduces uncertainty.

This finding is confirmed by the estimator’s inference accuracy analysis. As detailed in

Agent Architecture	Mean Reward	Standard Deviation
Oracle	1969.33	± 62.35
Learned A, True B	1641.08	± 89.09
Latent Estimator (Proposed)	1039.08	± 99.14
True A, Learned B	-356.78	± 172.84
Standard Agent	-495.62	± 19.19

Table 6.3: Convergence Performance in the Discrete Stochastic Domain.

Table 6.4, the Latent Estimator achieves an overall accuracy of only 46.62%, a sharp drop from the 83.45% observed in the deterministic case. The per-mode breakdown reveals that the West and North modes are identified with the highest reliability (78.36% and 62.66%, respectively), while the None mode exhibits the lowest accuracy (16.85%). However, this degradation is not attributable solely to the learned model: even the Oracle, operating with ground-truth parameters, achieves only 87.56% accuracy at the same confidence threshold of 0.85, with 26.93% of transitions excluded as low-confidence. This establishes an empirical upper bound on estimation quality in this setting and confirms that the stochastic latent dynamics introduce an irreducible degree of ambiguity that no estimator can fully resolve.

Wind Mode	Total Samples	Accuracy (%)	Assigned HMM States
West	23,992	78.36	4, 7
North	38,612	62.66	5, 6
East	20,655	54.50	2
South	45,023	36.82	0, 3
None	37,048	16.85	1
Overall	165,330	46.62	-

Table 6.4: Estimator Accuracy Report in the Discrete Stochastic Domain. Evaluated on high-confidence transitions.

6.3.2. Continuous Domain

Having established the effectiveness of the latent estimator in the discrete setting under both deterministic and stochastic dynamics, we now evaluate its generalisation to continuous state and action spaces.

Deterministic Latent Dynamics Figure 6.6 illustrates the performance of the agents under deterministic latent dynamics ($p = 1$).

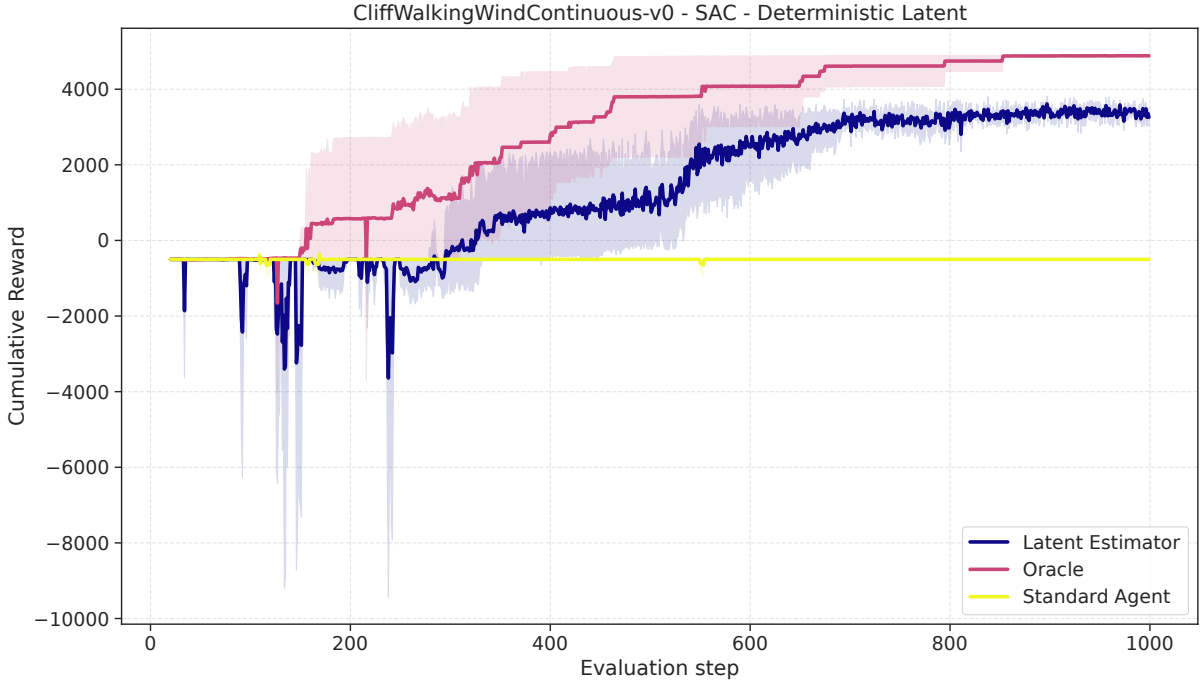


Figure 6.6: Cumulative Reward in the Continuous Deterministic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator. The initial episodes of random exploratory steps are excluded.

The transition to a continuous domain inherently worsens the complexity of the task, increasing the standard agent’s vulnerability to unobservable environmental shifts. Indeed, the baseline completely fails to learn a wind-compensation strategy, plateauing at -501.03 . Conversely, our latent estimator provides the policy with the necessary context to navigate the safe corridor, achieving 3380.90 , thus effectively converging toward the oracle’s upper bound of 4882.59 . This result demonstrates that the decoupled architecture successfully scales to continuous control tasks. Table 6.5 summarises the final convergence performance for the three agents.

Agent Architecture	Mean Reward	Standard Deviation
Oracle	4882.59	± 9.20
Latent Estimator (Proposed)	3380.90	± 254.93
Standard Agent	-501.03	± 0.51

Table 6.5: Convergence Performance in the Continuous Deterministic Domain.

Following the same evaluation setting already described, we now analyse the estimator’s inference accuracy for this setting. To ensure a meaningful evaluation, an adjustment is required: when the agent collides with a map boundary, its kinematic update is heavily

clipped, causing the physical transition from state s_t to s_{t+1} to appear identical regardless of the active wind mode. In these cases, the wall acts as a physical filter that effectively destroys the observability of the latent perturbation, rendering the HMM incapable of disambiguating the underlying context. An analysis of the trajectories, summarised in Table 6.6, confirms the severity of this issue as over 91% of the evaluated transitions consist of ambiguous interactions with the boundaries, either sliding along walls or sustaining hard collisions. Because these aliased states heavily dominate the continuous dataset, they are explicitly excluded from the accuracy analysis.

Interaction	Total Samples	Percentage
Wall Hugging (Sliding)	2,018,707	80.91%
Hard Collisions (Stuck)	260,715	10.45%
Total Boundary Interactions	2,279,422	91.36%

Table 6.6: Wall Interactions in the Continuous Deterministic Domain. Analysis of agent-boundary interactions leading to observation aliasing.

As reported in Table 6.7, the overall latent estimator accuracy is 64.26%, whereas the oracle baseline retains a perfect 100% accuracy. The per-mode breakdown reveals a significant disparity in inference capabilities: while it successfully identifies the East, West, and South modes with very high precision (ranging from approximately 86% to 95%), it exhibits a severe performance degradation on the North mode (25.93%) and entirely fails to recognise the None context.

Wind Mode	Total Samples	Accuracy (%)	Assigned HMM States
East	1,119	94.91	5
West	5,664	94.24	3, 4, 6, 7
South	2,417	86.02	1, 2
North	2,703	25.93	0
None	2383	-	-
Overall	14,286	64.26	-

Table 6.7: Estimator Accuracy Report in the Continuous Deterministic Domain. Evaluated on high-confidence transitions, excluding boundary collisions.

Crucially, despite this low accuracy, the Latent Estimator still managed to converge to a near-optimal policy, as demonstrated in the previous reward analysis. This provides a significant insight into the decoupled architecture: the learned representations do not need to perfectly align with the human-readable environmental variables. As long as

the HMM aggregates the transition dynamics into a belief state that provides sufficient context to navigate the safe corridor, the RL agent can successfully solve the underlying non-stationary MDP.

Stochastic Latent Dynamics Finally, we evaluate the framework in its most challenging configuration: the continuous domain governed by stochastic latent dynamics ($p = 0.9$). Figure 6.7 illustrates the mean cumulative reward curves for the continuous domain under these conditions. Due to the extreme complexity of navigating a narrow corridor while subjected to unpredictable perturbations, the evaluation in this setting was expanded to 10 independent random seeds to better capture the performance at convergence.

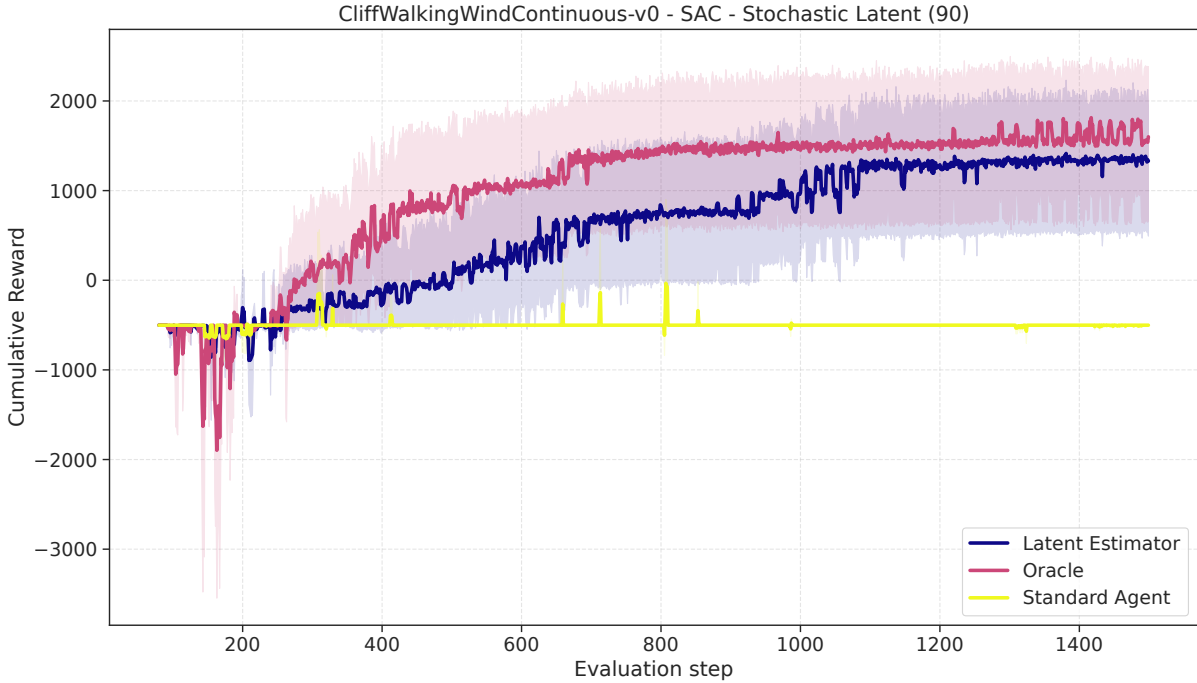


Figure 6.7: Cumulative Reward in the Continuous Stochastic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator. The initial episodes of random exploratory steps are excluded.

Once again, the standard agent fails to discover any goal-reaching policy and collapses to a negative mean return of -503.06 . In contrast, both the oracle and the latent estimator successfully converge to high-performing policies. Table 6.8 summarises the final metrics. A distinctive feature of these results is the unusually high standard deviation (≈ 1200) observed in both the oracle and the proposed framework. This variance indicates that the stochastic latent dynamics severely complicates the initial exploration phase: in a few “unlucky” seeds, the agents simply fail to discover the goal state before converging to a sub-optimal local minimum. Crucially, the latent estimator’s performance closely tracks

the oracle’s distribution, demonstrating that this instability is an intrinsic property of the MDP’s exploration landscape, rather than a failure of the inference module.

Agent Architecture	Mean Reward	Standard Deviation
Oracle	1624.01	± 1288.28
Latent Estimator (Proposed)	1342.48	± 1231.84
Standard Agent	-503.06	± 9.24

Table 6.8: Convergence Performance in the Continuous Stochastic Domain.

To further validate these behavioural results, we analyse the estimator’s inference accuracy. Consistent with the methodology established for the deterministic case, we explicitly exclude boundary interactions to prevent the physical walls from masking the wind observability, which, as reported in Table 6.9, still dominates 91.58% of the interactions.

Interaction	Total Samples	Percentage
Wall Hugging (Sliding)	3,000,857	80.18%
Hard Collisions (Stuck)	426,711	11.40%
Total Boundary Interactions	3,427,568	91.58%

Table 6.9: Wall Interactions in the Continuous Stochastic Domain. Analysis of agent-boundary interactions leading to observation aliasing.

As reported in Table 6.10, the latent estimator achieves an overall accuracy of 79.21%. It is crucial to note that the oracle achieves an accuracy of only 78.11% under the same conditions. These results prove that the combination of continuous kinematics and stochastic transitions injects an irreducible degree of perceptual ambiguity into the environment. A detailed examination of the per-mode breakdown reveals that the model successfully captures most of the latent dynamics. The North, South, None, and East modes are identified with high precision, all exceeding 83% accuracy. Notably, the HMM allocates two distinct latent states to the North, South, and None conditions, effectively disambiguating their occurrences despite the highly stochastic transitions. Conversely, the estimator struggles significantly with the West perturbation, where accuracy drops sharply to 27.77%.

Wind Mode	Total Samples	Accuracy (%)	Assigned HMM States
North	8,486	93.45	0, 6
South	1,508	87.47	2, 3
None	2,661	87.41	1, 4
East	2,237	83.15	5
West	3,187	27.77	7
Overall	18,079	79.21	-

Table 6.10: Estimator Accuracy Report in the Continuous Stochastic Domain.
Evaluated on high-confidence transitions, excluding boundary collisions.

7 | Conclusions

In this final chapter, we conclude our work by summarising our contributions in light of the experimental results. We provide a critical discussion of the results, examining the performance and the bottlenecks of our approach across different latent dynamics. Acknowledging the boundaries of our work, we then detail the limitations encountered, particularly concerning the emission model formulation and belief representation. Finally, we outline several promising directions for future research, distinguishing between immediate architectural refinements and broader extensions that relax our assumptions.

7.1. Summary of Contributions

In this work, we addressed the challenge of RL in non-stationary environments, focusing on a scenario where an unobservable latent variable governs the transition dynamics, hence in a partially observable environment. We provide three main contributions, validated by our empirical results:

1. **Decoupled framework for non-stationary RL:** We successfully implemented and evaluated a framework that separates latent state inference from policy optimisation. As demonstrated by our results, equipping a standard RL agent (Q-Learning or SAC) with a belief state allows it to converge to high-performing policies in environments where standard baseline agents systematically fail.
2. **Input-conditioned Baum-Welch algorithms:** We successfully extended the classical Baum-Welch algorithm to handle input-conditioned emissions for both discrete (tabular) and continuous (neural) environments. Through our empirical analysis and ablation studies, we demonstrated that learning the transition dynamics of the latent process is an easier task than estimating the emission model, which is the true bottleneck of latent inference.
3. **Empirical evaluation on non-stationary environments:** We evaluated the proposed framework on discrete and continuous non-stationary variants of the Cliff Walking environment under both deterministic and stochastic latent dynamics. The

empirical evaluation confirms the effectiveness of the proposed latent estimator in managing the environment’s non-stationarity. Throughout our experiments, we observed that even when the framework does not achieve perfect inference accuracy, it successfully compresses the environmental uncertainty into a belief state that is sufficient for high-level decision-making.

7.2. Limitations and Future Work

Based on the empirical results, we outline several directions for future research. These are organised into two groups. The first concerns direct improvements to the current framework, preserving its core assumptions and structure, directly targeting its limitations. The second explores more substantial extensions that would require relaxing the assumptions introduced in Chapter 4.

7.2.1. Direct Improvements to the Current Framework

The following directions build directly on the proposed architecture, targeting its limitations without altering the underlying problem formulation.

- Online emission model refinement:** The primary bottleneck identified in this work is the quality of the learned emission model $\mathbf{B}$ in the discrete setting. The tabular parametrisation requires each tuple (s, a, z, s') to be observed a sufficient number of times in the dataset to produce a reliable estimate. Even in the relatively compact discrete environment used in this work with $|\mathcal{S}| = 48$, $|\mathcal{A}| = 4$, and $|\mathcal{Z}| = 8$, the number of entries to be estimated is substantial, causing many tuples to remain under-represented, leading to a sparse emission model. This limitation becomes particularly critical in the stochastic setting, where the compensatory effect of a near-deterministic transition matrix $\mathbf{A}$ is no longer available, and the estimator must rely more heavily on the emission model to maintain an accurate belief. A natural direction is to continue updating the emission model $\mathbf{B}$ incrementally as the agent collects new transitions, using an online variant of the Baum-Welch algorithm. This would allow the emission model to be refined as the agent explores the environment, compensating for under-represented tuples in the dataset, while preserving the modularity of the decoupled framework and keeping the latent estimation independent from the control objective.
- Passing the full belief to the policy:** In the current implementation, the belief vector b_t is compressed into a scalar latent estimate $\hat{z}_t$ before being passed to the

RL agent. While this reduces the dimensionality of the augmented state, we lose valuable information regarding the model’s uncertainty, potentially forcing the agent to commit to actions based on low-confidence estimations. A natural extension is to pass the N -dimensional belief vector directly, allowing the policy to learn behaviours that explicitly account for the estimator’s uncertainty. Whether this approach would improve the performance remains an open question, as the increased dimensionality of the observation space complicates exploration and requires more training data.

- **Handling an unknown number of latent modes:** The current framework assumes that the number of latent modes $|\mathcal{Z}| = N$ is known a priori. In practice, this information may not be available, and misjudging the latent space complexity poses a fundamental problem: an overly small state space leads to underfitting, failing to capture distinct physical regimes, while an excessive capacity often results in degenerate solutions. One possible mitigation strategy is to deliberately overshoot the number of latent states, relying on the HMM to merge redundant states by assigning them similar emission profiles. However, this approach may lead to degenerate solutions where the extra states capture noise rather than meaningful dynamics.
- **Evaluation on diverse environments:** The framework was evaluated exclusively on variants of the Cliff Walking task. Validating it on a broader set of non-stationary environments is necessary to assess its generality and robustness.

7.2.2. Extensions Beyond the Current Assumptions

The following directions require revisiting the assumptions stated in Chapter 4 and represent longer-term research questions that could broaden the framework’s applicability.

- **Action-dependent latent transitions:** The current framework assumes passive non-stationarity, according to which the latent mode transitions are independent of the agent’s actions. Relaxing this assumption to allow action-dependent transitions, as we did for the emission model, would broaden the applicability to settings where the agent’s behaviour influences the evolution of the latent context.
- **Extension to continuous latent spaces:** The current framework assumes a discrete and finite set of latent modes. Extending it to settings where the latent variable is continuous would require replacing the discrete transition matrix $\mathbf{A}$ with a continuous latent dynamics model parametrised by a neural network, and adopting particle filtering in place of the exact Bayesian filter to maintain a tractable posterior over the continuous latent space.

Bibliography

- [1] S. Abdallah. Addressing environment non-stationarity by repeating Q-learning updates. *Journal of Machine Learning Research*, 17(1):1624–1654, 2016.
- [2] R. P. Adams and D. J. C. MacKay. Bayesian online changepoint detection. Technical Report arXiv:0710.3742, University of Cambridge, Cambridge, UK, 2007.
- [3] L. N. Alegre, A. L. C. Bazzan, and B. C. da Silva. Minimum-delay adaptation in non-stationary reinforcement learning via online high-confidence change-point detection. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '21*, pages 97–105. International Foundation for Autonomous Agents and Multiagent Systems, 2021.
- [4] C. Alippi, G. Boracchi, and M. Roveri. Just-in-time classifiers for recurrent concepts. *IEEE Transactions on Neural Networks and Learning Systems*, 24(4):620–634, 2013.
- [5] P. Auer and R. Ortner. Logarithmic Online Regret Bounds for Undiscounted Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006.
- [6] T. Banerjee, M. Liu, and J. P. How. Quickest Change Detection Approach to Optimal Control in Markov Decision Processes with Model Changes. In *2017 American Control Conference (ACC)*, pages 3851–3856, 2017.
- [7] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*, volume 104. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [8] E. W. Basso and P. M. Engel. Reinforcement Learning in Non-Stationary Continuous Time and Space Scenarios. In *Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, pages 61–70. SBC, 2009.
- [9] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [10] A. Bifet and R. Gavaldà. Learning from Time-Changing Data with Adaptive Win-

- dowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. Society for Industrial and Applied Mathematics, 2007.
- [11] Canonaco Giuseppe, Restelli Marcello, and Roveri Manuel. Model-Free Non-Stationarity Detection and Adaptation in Reinforcement Learning. In *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2020.
 - [12] H. Caselles-Dupré, M. Garcia-Ortiz, and D. Filliat. Continual State Representation Learning for Reinforcement Learning using Generative Replay. In *NeurIPS 2018 Workshop on Continual Learning*, 2018.
 - [13] Y. Chandak, G. Theodoropoulos, S. Shankar, M. White, S. Mahadevan, and P. Thomas. Optimizing for the future in non-stationary MDPs. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1414–1425. PMLR, 13–18 Jul 2020.
 - [14] W. C. Cheung, D. Simchi-Levi, and R. Zhu. Reinforcement Learning for Non-Stationary Markov Decision Processes: The Blessing of (More) Optimism. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1843–1854. PMLR, 2020.
 - [15] S. P. M. Choi, D.-Y. Yeung, and N. L. Zhang. Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making. In R. Sun and C. L. Giles, editors, *Sequence Learning: Paradigms, Algorithms, and Applications*, pages 264–287. Springer, 2001.
 - [16] B. C. da Silva, E. W. Basso, A. L. C. Bazzan, and P. M. Engel. Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd International Conference on Machine Learning, ICML '06*, pages 217–224. Association for Computing Machinery, 2006.
 - [17] G. Ditzler and R. Polikar. Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*, pages 41–48, 2011.
 - [18] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.
 - [19] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

- [20] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.
- [21] Z. Fang and U. Zdun. Detecting Environment Drift in Reinforcement Learning Using a Gaussian Process. In *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 992–999. IEEE, 2024.
- [22] F. Feng, B. Huang, K. Zhang, and S. Magliacane. Factored adaptation for non-stationary reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 30635–30648, 2022.
- [23] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with Drift Detection. In A. L. C. Bazzan and S. Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, volume 3171, pages 286–295. Springer Berlin Heidelberg, 2004.
- [24] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, 2014.
- [25] M. Geist, O. Pietquin, and G. Fricout. Tracking in Reinforcement Learning. In C. S. Leung, M. Lee, and J. H. Chan, editors, *Neural Information Processing*, pages 502–511. Springer, 2009.
- [26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.
- [27] E. Hadoux, A. Beynier, and P. Weng. Sequential Decision-Making under Non-stationary Environments via Sequential Change-point Detection. In *Learning over Multiple Contexts (LMCE)*, 2014.
- [28] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*, pages 2555–2565. PMLR, 2019.
- [29] J. D. Healy. A note on multivariate cusum procedures. *Technometrics*, 29(4):409–412, 1987.
- [30] B. Huang, F. Feng, C. Lu, S. Magliacane, and K. Zhang. Adarl: What, where, and how to adapt in transfer reinforcement learning. *arXiv*, 2021.
- [31] M. Hussein, M. Keshk, and A. Hussein. Non-stationarity Detection in Model-Free Reinforcement Learning via Value Function Monitoring. In T. Liu, G. Webb, L. Yue,

- and D. Wang, editors, *AI 2023: Advances in Artificial Intelligence*, pages 350–362. Springer Nature, 2024.
- [32] T. Jaksch, P. Auer, and R. Ortner. Near-optimal Regret Bounds for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008.
 - [33] N. Jali, E. Pathak, P. Sharma, G. Qu, and G. Joshi. Natural Policy Gradient for average reward non-stationary RL, 2025.
 - [34] B.-H. Juang and L. Rabiner. Mixture autoregressive hidden markov models for speech signals. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(6):1404–1413, 1985.
 - [35] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.
 - [36] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005.
 - [37] R. E. Kalman. A new approach to linear filtering and prediction problems. *J. Basic Eng.*, 82(1):35–45, 1960.
 - [38] K. Khetarpal, M. Riemer, I. Rish, and D. Precup. Towards Continual Reinforcement Learning: A Review and Perspectives. *The Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.
 - [39] J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8(12):2755–2790, 2007.
 - [40] E. Lecarpentier and E. Rachelson. Non-Stationary Markov Decision Processes, a Worst-Case Approach using Model-Based Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
 - [41] H. Lee, M. Jin, J. Lavaei, and S. Sojoudi. Pausing policy learning in non-stationary reinforcement learning. *CoRR*, abs/2405.16053, 2024.
 - [42] M. Li, C. Shi, Z. Wu, and P. Fryzlewicz. Testing stationarity and change point detection in reinforcement learning. *The Annals of Statistics*, 53(3), 2025.
 - [43] Z. Liu, J. Lu, J. Xuan, and G. Zhang. Learning Latent and Changing Dynamics in Real Non-Stationary Environments. *IEEE Transactions on Knowledge and Data Engineering*, 37(4):1930–1942, 2025.

- [44] Z. Liu, J. Lu, G. Zhang, and J. Xuan. A behavior-aware approach for deep reinforcement learning in non-stationary environments without known change points. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, pages 4634–4642. International Joint Conferences on Artificial Intelligence Organization, 8 2024.
- [45] F. Locatello, S. Bauer, M. Lucic, G. Raetsch, S. Gelly, B. Schölkopf, and O. Bachem. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4114–4124. PMLR, 2019.
- [46] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2018.
- [47] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.
- [48] R. Ortner, P. Gajane, and P. Auer. Variational regret bounds for reinforcement learning. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 115 of *Proceedings of Machine Learning Research*, pages 81–90. PMLR, 2020.
- [49] S. Padakandla, P. K. J., and S. Bhatnagar. Reinforcement learning algorithm for non-stationary environments. *Applied Intelligence*, 50(11):3590–3606, 2020.
- [50] K. Prabuchandran, N. Singh, P. Dayama, A. Agarwal, and V. Pandit. Change point detection for compositional multivariate data. *Appl. Intell.*, 52(2):1930–1955, 2022.
- [51] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.
- [52] C. Raab, M. Heusinger, and F.-M. Schleif. Reactive soft prototype computing for concept drift streams. *Neurocomputing*, 416:340–351, 2020.
- [53] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [54] L. R. Rabiner, B.-H. Juang, S. E. Levinson, and M. M. Sondhi. Some properties of continuous hidden markov model representations. *AT&T Technical Journal*, 64:1251–1270, 1985.

- [55] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [56] A. Russo, A. M. Metelli, and M. Restelli. Achieving $\tilde{O}(\sqrt{T})$ regret in average-reward POMDPs with known observation models. In *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pages 4168–4176. PMLR, 2025.
- [57] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2 edition, 2018.
- [58] R. S. Sutton, A. Koop, and D. Silver. On the role of tracking in stationary environments. In *Proceedings of the 24th International Conference on Machine Learning*, pages 871–878. ACM, 2007.
- [59] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, T. De Lazcano, A. Khalid, N. Mercer, T. Basaklar, S. Grijalva, O. Garcia, R. Dossa, F. Pardo, A. Petang, H. Tan, Z. Huang, N. Kallias, A. Black, C. Huang, S. Zhang, M. Wang, A. Negli, M. Singla, N. Lucas, J. Reuss, D. Oreshkin, S. Sezener, J. Roussel, G. Sasic, D. Geng, and F. Foundation. Gymnasium: A standard interface for reinforcement learning environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [60] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.
- [61] B. L. Welch. The generalization of ‘Student’s’ problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.
- [62] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.
- [63] A. Xie, J. Harrison, and C. Finn. Deep reinforcement learning amidst continual structured non-stationarity. In *Proc. Int. Conf. Mach. Learn. (ICML)*, pages 11393–11403, 2021.
- [64] J. Y. Yu, S. Mannor, and N. Shimkin. Markov Decision Processes with Arbitrary Reward Processes. In S. Girgin, M. Loth, R. Munos, P. Preux, and D. Ryabko, editors, *Recent Advances in Reinforcement Learning*, pages 268–281. Springer, 2008.

A | Appendix A

A.1. Baum-Welch Algorithm

The classical Baum-Welch algorithm, introduced in Chapter 2, is an instance of the EM algorithm used to learn the parameters $\lambda = \langle b_1, \mathbf{A}, \mathbf{B} \rangle$ of a standard HMM from a sequence of observations. We report its pseudocode here as a baseline for comparison with the tabular and neural input-conditioned variants presented below.

Algorithm A.1 Classical Baum-Welch Algorithm

Require: Observation sequence $o_{1:T}$, number of latent states N , max EM iterations I , tolerance ϵ

Ensure: Optimized model parameters $\lambda = \langle b_1, \mathbf{A}, \mathbf{B} \rangle$

- 1: Initialise $b_1, \mathbf{A}, \mathbf{B}$ randomly or with prior knowledge
- 2: $\ell_{\text{prev}} \leftarrow -\infty$
- 3: **for** $i = 1, \dots, I$ **do**
- 4: **E-step:**
- 5: Compute forward variables $\alpha_t(z)$ using Eq. (2.3)
- 6: Compute backward variables $\beta_t(z)$ using Eq. (2.6)
- 7: Compute $\gamma_t(z)$ using Eq. (2.7) and $\xi_t(z, z')$ using Eq. (2.8)
- 8: Compute log-likelihood $\ell \leftarrow \log \sum_z \alpha_T(z)$
- 9: **if** $|\ell - \ell_{\text{prev}}| < \epsilon$ **then**
- 10: **break**
- 11: **end if**
- 12: $\ell_{\text{prev}} \leftarrow \ell$
- 13: **M-step:**
- 14: Update transition matrix $\hat{\mathbf{A}}$ using Eq. (2.9)
- 15: Update emission matrix $\hat{\mathbf{B}}$ using Eq. (2.10)
- 16: Update initial distribution $\hat{b}_1 = \gamma_1(z)$.
- 17: **end for**
- 18: **return** $b_1, \mathbf{A}, \mathbf{B}$

A.2. Input-Conditioned Baum-Welch Algorithms

The tabular and neural variants of the input-conditioned Baum-Welch algorithm share the same overall E-step/M-step structure, but differ in three key aspects: how the emission model is updated (closed-form vs. gradient-based), how the emission probabilities are computed (table lookup vs. neural network forward pass), and how the final parameters are selected (last iteration vs. best checkpoint).

Algorithm A.2 Tabular Input-Conditioned Baum-Welch

Require: dataset $\mathcal{D} = \{(s_t, a_t, s_{t+1})\}_{t=1}^{T-1}$, number of latent states N , max EM iterations I , tolerance ϵ

Ensure: Optimized parameters $\lambda = \langle b_1, \mathbf{A}, \mathbf{B} \rangle$

- 1: Initialise $b_1(0) = 1$, $b_1(z) = 0$ for $z \neq 0$
- 2: Initialise $\mathbf{A}$ uniform with small random noise, row-normalised
- 3: Initialise $\mathbf{B}$ with entries drawn uniformly from $[0, 1]$, normalised along the observation axis
- 4: $\ell_{\text{prev}} \leftarrow -\infty$
- 5: **for** $i = 1, \dots, I$ **do**
- 6: **E-step:**
- 7: Compute $\alpha_t(z)$ for each episode using Eq. (5.1)
- 8: Compute $\beta_t(z)$ for each episode using Eq. (5.2)
- 9: Compute $\gamma_t(z)$ using Eq. (5.3)
- 10: Compute $\xi_t(z, z')$ using Eq. (5.4)
- 11: Compute log-likelihood $\ell \leftarrow \sum_{\text{ep}} \log \sum_z \alpha_T(z)$
- 12: **if** $|\ell - \ell_{\text{prev}}| < \epsilon$ **then**
- 13: **break**
- 14: **end if**
- 15: $\ell_{\text{prev}} \leftarrow \ell$
- 16: **M-step:**
- 17: Update $\hat{\mathbf{A}}$ using Eq. (5.5)
- 18: Update $\hat{\mathbf{B}}$ using Eq. (5.6)
- 19: Update $\hat{b}_1(z) \leftarrow \gamma_1(z)$
- 20: **end for**
- 21: **return** $b_1, \mathbf{A}, \mathbf{B}$

Algorithm A.3 Neural Contextual Baum-Welch

Require: dataset $\mathcal{D} = \{(c_t, s_{t+1})\}_{t=1}^{T-1}$, number of latent states N , max EM iterations I , max emission iterations I_θ , tolerance ϵ

Ensure: learned parameters $\langle b_1, \mathbf{A}, B_\theta \rangle$

```

1: Initialise  $b_1(0) = 1$ ,  $b_1(z) = 0$  for  $z \neq 0$ 
2: Initialise  $\mathbf{A}$  uniform with small random noise, row-normalised
3: Initialise emission network  $B_\theta$  with default weight initialisation
4:  $\ell_{\text{prev}} \leftarrow -\infty$ ,  $\ell_{\text{best}} \leftarrow -\infty$ ,  $p \leftarrow 0$ 
5: for  $i = 1, \dots, I$  do
6:   E-step:
7:     Compute  $B_\theta(s_{t+1} \mid s_t, a_t, z)$  for all  $t, z$  using Eq. (5.7)
8:     Compute  $\alpha_t(z)$  for each episode using Eq. (5.1)
9:     Compute  $\beta_t(z)$  for each episode using Eq. (5.2)
10:    Compute  $\gamma_t(z)$  using Eq. (5.3)
11:    Compute  $\xi_t(z, z')$  using Eq. (5.4)
12:    Compute log-likelihood  $\ell \leftarrow \sum_{\text{ep}} \log \sum_z \alpha_T(z)$ 
13:    if  $\ell > \ell_{\text{best}}$  then
14:       $\ell_{\text{best}} \leftarrow \ell$ , save  $\langle b_1^*, \mathbf{A}^*, B_\theta^* \rangle$ 
15:    end if
16:    if  $|\ell - \ell_{\text{prev}}| < \epsilon$  then
17:      break
18:    end if
19:     $\ell_{\text{prev}} \leftarrow \ell$ 
20:    M-step:
21:      Update  $\hat{\mathbf{A}}$  using Eq. (5.5)
22:      Update  $\hat{b}_1(z) \leftarrow \gamma_1(z)$ 
23:      for  $j = 1, \dots, I_\theta$  do
24:        Sample  $\hat{z}_t \sim \gamma_t(\cdot)$  for all  $t$ 
25:        Update  $B_\theta$  via Adam on  $\mathcal{L}_{\text{emit}}$  using Eq. (5.8)
26:      end for
27:    end for
28: Restore best parameters  $\langle b_1^*, \mathbf{A}^*, B_\theta^* \rangle$ 
29: return  $b_1, \mathbf{A}, B_\theta$ 

```

A.3. Hyperparameters

In this appendix, we detail the specific hyperparameter configurations used for the algorithms employed in our experiments.

Table A.1 reports the configuration for the Tabular Q-Learning used in the discrete environment experiments. Both the learning rate and the exploration rate (ϵ) follow a linear decay schedule to balance early exploration and late-stage exploitation. Specifically, the exploration rate is decayed over the first half of the training phase (*Exploration Fraction*

$= 0.5$), after which it is kept constant at its minimum value to maintain a baseline level of stochasticity.

Hyperparameter	Value
Initial Learning Rate (α_{start})	0.01
Final Learning Rate (α_{end})	0.001
Discount Factor (γ)	0.99
Initial Exploration Rate (ϵ_{start})	1.0
Final Exploration Rate (ϵ_{end})	0.05
Exploration Fraction	0.5
Learning Rate Schedule	Linear Decay
Exploration Schedule	Linear Decay

Table A.1: Tabular Q-Learning Hyperparameters. Configuration parameters used for training the discrete agent.

Table A.2 reports the configuration for the SAC algorithm used in the continuous environment experiments.

Hyperparameter	Value
Learning rate	3×10^{-4}
Buffer size	1,000,000
Batch size	256
Learning starts	20,000
Train frequency	16
Gradient steps	16
Gamma (γ)	0.99
Tau (τ)	0.005
Entropy coefficient	auto
Target entropy	auto
Network architecture	[256, 256]
Action noise	Ornstein-Uhlenbeck ($\sigma = 0.1$)

Table A.2: Soft Actor-Critic Hyperparameters. Configuration parameters used for training the continuous agent.

List of Figures

2.1	Agent-Environment interaction loop. The standard agent-environment interaction loop in RL showing the continuous interaction through states, actions, and rewards. Redrawn from [57].	9
2.2	POMDP interaction loop. Agent-environment interaction loop under partial observability, highlighting the separation between the unobservable state s_{t+1} and the agent's observation o_t	16
5.1	Architectural overview of the proposed latent-conditioned framework. The framework is composed of two phases: an offline phase, where the input-emission conditioned Baum-Welch algorithm processes historical trajectories $\mathcal{D}$ to identify the latent model parameters λ , and an online control phase. During deployment, the frozen model is embedded within an augmented environment wrapper to perform real-time Bayesian filtering using the incoming transition (s_t, a_t, s_{t+1}) , providing an augmented observation x_{t+1} that enables the RL agent to adapt its policy to non-stationary environmental shifts.	36
6.1	Visual representation of the Discrete Cliff Walking Wind environment. The agent starts at the bottom-left cell (blue) and must navigate to the goal (green) at the opposite end. The cliff (grey) along the bottom edge and the penalty zones (yellow) restrict the navigable area. This layout explicitly forces the agent to traverse the safe corridor.	46
6.2	Visual representation of the Continuous Cliff Walking Wind environment. The agent starts at the bottom-left position (blue) and must navigate to the goal area (green) at the opposite end. The cliff (grey) along the bottom edge and the penalty zones (yellow) restrict the navigable area. This layout explicitly forces the agent to traverse the central safe corridor, preventing it from exploiting the map boundaries to counteract the wind. .	47

6.3	Visual representation of the latent chain. The environment cycles through a fixed sequence of wind modes, in order: East, South, None, North, West, North, None, and South.	48
6.4	Cumulative Reward in the Discrete Deterministic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator.	50
6.5	Cumulative Reward in the Discrete Stochastic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator.	52
6.6	Cumulative Reward in the Continuous Deterministic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator. The initial episodes of random exploratory steps are excluded.	54
6.7	Cumulative Reward in the Continuous Stochastic Domain. Episodic return of the standard agent (lower bound), the Oracle (upper bound), and the proposed Latent Estimator. The initial episodes of random exploratory steps are excluded.	56

List of Tables

3.1	Taxonomy of active approaches. Summary of state-of-the-art methods that employ explicit drift detection mechanisms to identify and characterise environmental shifts.	25
3.2	Taxonomy of passive approaches. Summary of state-of-the-art methods that handle non-stationarity through intrinsic algorithmic robustness or continuous adaptation, without relying on explicit drift detection.	29
6.1	Convergence Performance in the Discrete Deterministic Domain.	50
6.2	Estimator Accuracy Report in the Discrete Deterministic Domain. Evaluated on high-confidence transitions.	51
6.3	Convergence Performance in the Discrete Stochastic Domain. . .	53
6.4	Estimator Accuracy Report in the Discrete Stochastic Domain. Evaluated on high-confidence transitions.	53
6.5	Convergence Performance in the Continuous Deterministic Domain.	54
6.6	Wall Interactions in the Continuous Deterministic Domain. Analysis of agent-boundary interactions leading to observation aliasing.	55
6.7	Estimator Accuracy Report in the Continuous Deterministic Domain. Evaluated on high-confidence transitions, excluding boundary collisions.	55
6.8	Convergence Performance in the Continuous Stochastic Domain.	57
6.9	Wall Interactions in the Continuous Stochastic Domain. Analysis of agent-boundary interactions leading to observation aliasing.	57
6.10	Estimator Accuracy Report in the Continuous Stochastic Domain. Evaluated on high-confidence transitions, excluding boundary collisions. . .	58
A.1	Tabular Q-Learning Hyperparameters. Configuration parameters used for training the discrete agent.	72
A.2	Soft Actor-Critic Hyperparameters. Configuration parameters used for training the continuous agent.	72

List of Acronyms

ML	Machine Learning
SL	Supervised Learning
RL	Reinforcement Learning
MDP	Markov Decision Process
MARL	Multi-Agent Reinforcement Learning
DRL	Deep Reinforcement Learning
POMDP	Partially Observable Markov Decision Process
HMM	Hidden Markov Model
AR-HMM	Auto-Regressive Hidden Markov Model
EM	Expectation-Maximization
SAC	Soft Actor-Critic
QL	Q-Learning
GP	Gaussian Process
MSE	Mean Squared Error
CUSUM	CUmulative SUM
RL-CD	Reinforcement Learning with Context Detection
SCPD-RLCD	Sequential Change-Point Detection RL-CD
KL	Kullback-Leibler
MBCD	Model Based RL Context Detection
NSD-RL	Non-Stationarity Detector for RL
VAE	Variational AutoEncoder

LLCD	Learning Lantent and Changing Dynamics
RSSM	Recurrent State-Space Model
BADA	Behaviour-Aware Detection and Adaptation
HM-MDP	Hidden Mode MDP
LILAC	Lifelong Latent Actor-Critic
AdaRL	ADaptive RL
FANS-RL	Factored Adaptation for Non-Stationary-RL
DP-MDP	Dynamic Parameter-MDP
UCRL	Upper Confidence RL
OFU	Optimism in Face of Uncertainty
SW-UCRL2-CW	Sliding Window UCRL2 with Confidence Widening
BORL	Bandit-Over-RL
AOAS-UCRL	Action wise Observation-Aware Spectral-UCRL
RATS	Risk Averse Tree Search
KTD	Kalman Temporal Difference
RUQL	Repeated Update QL
Q-FPL	Q-Follow the Perturbed Leader
FPL	Follow the Perturbed Leader
NS-NAC	Non-Stationary Natural Actor-Critic