NS-Gym: Open-Source Simulation Environments and Benchmarks for Non-Stationary Markov Decision Processes

Anonymous submission

Abstract

In many real-world applications, agents must make sequential decisions in environments where conditions are subject to change due to various exogenous factors. These non-stationary environments pose significant challenges to traditional decision-making models, which typically assume stationary dynamics. Non-stationary Markov decision processes (NS-MDPs) offer a framework to model and solve decision problems under such changing conditions. However, the lack of standardized benchmarks and simulation tools has hindered systematic evaluation and advance in this field. We present NS-Gym, the first simulation toolkit designed explicitly for NS-MDPs, integrated within the popular Gymnasium framework. In NS-Gym, we segregate the evolution of the environmental parameters that characterize non-stationarity from the agent's decision-making module, allowing for modular and flexible adaptations to dynamic environments. We review prior work in this domain and present a toolkit encapsulating key problem characteristics and types in NS-MDPs. This toolkit is the first effort to develop a set of standardized interfaces and benchmark problems to enable consistent and reproducible evaluation of algorithms under non-stationary conditions. We also benchmark six algorithmic approaches from prior work on NS-MDPs using NS-Gym. Our vision is that NS-Gym will enable researchers to assess the adaptability and robustness of their decision-making algorithms to non-stationary conditions.

Introduction

Many real-world problems involve agents making sequential decisions over time under exogenous sources of uncertainty. Such problems exist in autonomous driving (Kiran et al. 2021), medical diagnosis and treatment (Yu et al. 2021), emergency response (Mukhopadhyay et al. 2022), vehicle routing (Li, Yan, and Wu 2021), and financial portfolio optimization (Pendharkar and Cusatis 2018). We define an *agent* as an entity capable of computation that *acts* based on *observations* from the environment (Kochenderfer, Wheeler, and Wray 2022). Decision-making for such agents is widely modeled by Markov decision processes (MDPs), a general mathematical model for stochastic control processes.

A canonical challenge in such problems, motivated by practical scenarios, is non-stationarity, where the distribution of environmental conditions can change over time. While non-stationarity has been well-explored from both control and decision-theoretic perspectives, several conceptual paradigms of non-stationarity exist, which lead to different mathematical formalisms for how the environmental parameters change and how the agent (and the control process) interacts with the changes. Ackerson and Fu (1970) provide one of the earliest conceptualizations of a system operating in "switching" environments, where the mean and covariance of the underlying process can change over time. Campo, Mookerjee, and Bar-Shalom (1991) formalize the switching process, where some environment parameters can change after a random sojourn time, as a sojourn-time-dependent Markov chain, which is semi-Markovian.

Recent investigations of non-stationary stochastic control processes involve two major threads: the first problem deals with an agent trying to adapt to a single change in the environment (which can either be observed (Pettet et al. 2024) or unobserved (Luo et al. 2024)); and the second problem models situations where environmental parameters can change continuously over time (Lecarpentier and Rachelson 2019). In an orthogonal line of work, Chandak et al. (2020b) present a problem formulation where the agent's goal is to maximize a forecast of future performance (of the control policy) instead of directly modeling the non-stationarity. Notably, these problem classes provide fundamentally different formalisms (or treatments) for non-stationarity.

Indeed, not only are the formalisms different, but we point out another interesting observation from prior work on non-stationary stochastic control processes: while recent prior work on *stationary* Markov decision processes (MDP) use standard benchmark problems, e.g., by using the popular Gymnasium toolkit (Towers et al. 2023), there are no standard problems or benchmarks for *non-stationary* MDPs. For example, Lecarpentier and Rachelson (2019) evaluate non-stationarity using a custom non-stationary bridge environment (an abstract problem where an agent must navigate on a grid-based slippery maze where the properties of the surface change over time), Chandak et al. (2020b) use problems motivated by real-world applications such as recommendation systems and diabetes treatment, and Pettet et al. (2024) use well-known benchmark problems used for stationary MDPs (e.g., the cartpole problem from Gymnasium (Towers et al. 2023)) and introduce non-stationarity manually.

In this paper, we identify key characteristics of non-stationary MDPs that affect decision-making, review prior work in this area, and present the first simulation toolkit specifically tailored for non-stationary MDPs. We argue that four key considerations affect decision-making in non-stationary MDPs, where environmental factors can change over time: *what* changes? *how* does it change? can the agent *detect* the change? can the agent *know* the updated parameter that has changed? These questions summarize the nature of the change and the key properties of modeling approaches from prior work. Based on these questions, we present NS-Gym (Non-Stationary Gym), the first collection of simulation environments for non-stationary MDPs. Inspired by the seminal work of Campo, Mookerjee, and Bar-Shalom (1991), we segregate the evolution of the environmental parameters that characterize non-stationarity and the agent's decision-making module. This modularization enables us to configure various components (and types) of non-stationary MDPs seamlessly. The NS-Gym toolkit is based on Python and is completely compatible with the widely popular Gymnasium framework. Instead of developing a new simulation environment from scratch, we build upon the existing Gymnasium toolkit due to its popularity and ensure that the large user base already familiar with Gymnasium can easily use NS-Gym (we keep all standard Gymnasium functionalities and interfaces intact). Specifically, we make the following contributions. We make the following contributions:

1. We present the first simulation toolkit for NS-MDPs that provides a tailored, standardized, and principled set of interfaces for non-stationary environments.
2. We identify canonical problem instances for decision-making in non-stationary environments, e.g., decision-making where the agent *knows* about the change but is not *aware* of exactly what the change is, or decision-making where the agent is aware of the change.
3. We present an overview of prior work on non-stationary decision-theoretic models and a programming interface that unifies prior work.
4. Our simulation framework extends the widely popular Gymnasium toolkit, thereby requiring minimal added efforts from researchers in online planning, reinforcement learning, and decision-making in using our toolkit.
5. We present the first set of benchmark results (and open-source implementations using NS-Gym) that compares six algorithmic approaches for solving NS-MDPs.
6. Our benchmark results are presented across a series of problem types in non-stationary environments.

The rest of the paper is organized as follows. We begin by describing characteristics of NS-MDPs and prior work. Then, we identify canonical problem instances, describe our framework, and present a tutorial of how to use it. Finally, we present benchmark results using NS-Gym.

Characteristics of NS-MDPs and Prior Work

We begin by describing a comprehensive framework for decision-making in non-stationary environments. Admittedly, we point out that the conceptual boundaries of what constitutes an *agent* are unclear in this context. Instead, we leave this question open and point out the key components relevant to decision-making; whether these components are

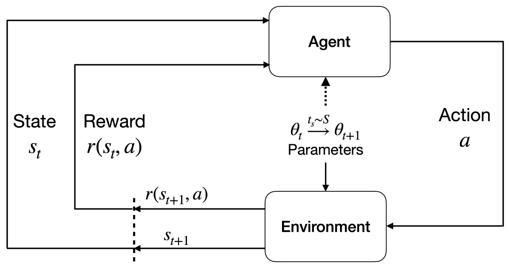

Figure 1: An overall framework for non-stationary Markov decision processes. At time t, the agent observes the state $s_t \in S$ and takes an action $a \in A$. The environment emits a reward signal $r(s_t, a)$ and transitions to the next state s_{t+1}. The transition and the reward are governed by parameters θ, which do not necessarily have a stationary distribution. In general, the evolution of θ occurs through a semi-Markov chain whose textitsojourn time is distributed as S, which might be non-memoryless. Depending on the problem, the agent can detect and/or observe the evolution of θ.

part of the agent or those supporting the agent is orthogonal to our discussion.

We refer to an agent as an entity that receives observations from an environment and can act or make decisions that interact with said environment. For simplicity, we assume a discrete-time process, although this discussion also extends to continuous-time stochastic control processes. Our fundamental model is that of a Markov decision process (Puterman 2014). We refer to the current state of the environment by $s \in \mathcal{S}$ and an action by $a \in \mathcal{A}$, where $\mathcal{S}$ and $\mathcal{A}$ denote the set of all states and actions, respectively. After taking an action: 1) the agent receives a scalar signal $r(s, a)$ from the environment, which can be perceived as a reward or a loss and is a measure of the agent's utility, and 2) the agent transitions to a new state, governed by a transition function $P(s' \mid s, a, \theta)$, where $\theta \in \Theta$ denotes a set of observable environmental parameters. We argue that explicitly specifying θ is critical to modeling non-stationary decision-making problems, as highlighted below.

We show a schematic of the major decision-theoretic components in Figure 1. In a non-stationary stochastic control process, the environmental parameters θ or the agent's utility function $r(s, a)$ can change over time. The manner in which the change evolves over time can be modeled by a Markov chain or, more generally, by a semi-Markov chain as proposed by Campo, Mookerjee, and Bar-Shalom (1991). While this formalism has often not been used in recent work (which has focused less on the statistical properties of the changes), we argue that a formal representation of how the environmental parameters evolve is particularly important from the perspective of studying NS-MDPs. We use the same high-level formalism as Campo, Mookerjee, and Bar-Shalom (1991), i.e., the parameters θ evolve in time through a sojourn time distribution, which can be non-memoryless, thereby making the resulting stochastic process semi-Markovian (Hu and Yue 2007). If the sojourn-time distribution is memoryless, then the resulting process is a

Model	Reference	Is the change notified?	Is the change known?	What changes?	Nature of the change	Is the change bounded?
Piecewise Stationary MAB	Garivier and Moulines (2011)	No	No	Reward	The reward distribution is fixed over certain time periods, and then changes at unknown time steps.	No
Non-stationary MAB	Besbes, Gur, and Zeevi (2014)	No	No	Reward	The reward can change at arbitrary time points.	Yes
Piecewise Stationary MDP	Auer, Jaksch, and Ortner (2008)	No	No	Transition, Reward	Bounded change analyzed as part of the UCRL2 algorithm	Yes
Non-Stationary MDP	Cheung, Simchi-Levi, and Zhu (2020)	N/A	No	Transition, Reward	The reward and transition can change at every time step	Yes
Non-Stationary MDP	Chandak et al. (2020b)	Yes	No	Transition, Reward	Transition and reward can change after each episode, but remain fixed within an episode	No
Non-Stationary MDP	Chandak et al. (2020a)	Yes	No	Transition, Reward	Transition and reward can change after each episode, but remain fixed within an episode	Yes
Non-Stationary MDP	Lecarpentier and Rachelson (2019)	Yes	Yes	Transition	The agent knows the current parameters, but not the future evolution.	Yes
Non-Stationary MDP	Pettet et al. (2024)	Yes	Yes	Transition	A single discrete change	Yes
Non-Stationary MDP	Luo et al. (2024)	Yes	No	Transition	A single discrete change	No
Non-Stationary Bandits with Periodic Variation	Chakraborty and Shettiwar (2024)	No	No	Reward	Periodic Variation	Yes

Table 1: Prior work on non-stationary Markov decision processes, categorized by important characteristics that affect decision making.

continuous-time Markov chain (Hu and Yue 2007).

Motivated by how decision-making components are implemented in practice, we introduce two additional components: first, we introduce a runtime monitor that tracks the parameters θ and *detects* changes; in practice, the monitor can be implemented as an anomaly detector (Chandola, Banerjee, and Kumar 2009). Note that while a monitor can track and detect changes in θ; it might not by itself be equipped to update the transition model P. For example, consider an autonomous driving agent trained on video feeds without rain. During decision-making, if it rains, an anomaly detector can trivially identify feeds that are out-of-distribution of the training data. However, the detector is usually not equipped to update the agent's internal model of how rainfall might affect the car's movement. From the agent's perspective, we refer to the ability to detect these environmental changes as *receiving a notification* about the change; note that we use this terminology to emphasize the segregation between the agent and the anomaly detector.

We introduce a second component, a *model updater*, which is a computational entity that can update the transition model by observing the changed parameters θ. We do not argue that every agent designed for decision-making in non-stationary environments must have these components; indeed, we point out algorithmic prior work where one or both of these components are absent. Instead, we argue that these components sufficiently describe the infrastructure required for decision-making in non-stationary environments, whether a specific agent designs these components or simply assumes their existence is orthogonal to our discussion. Given these components, we categorize prior work in non-stationary stochastic control processes by answering four key questions, as highlighted in Table 1.

Framework Description

In this section, we outline the general structure of NS-Gym, elaborate upon our design decisions, and describe the general experimental pipeline using NS-Gym. The project's source code can be found in the supplementary material.

Background

The *environment* object in Gymnasium encapsulates an MDP, providing a set of states and possible actions and defining how actions influence state transitions and rewards. The *observation* object represents the current state information available to the agent. Additionally, *Info* object is a dictionary containing auxiliary diagnostic information beneficial for debugging or gaining additional insights about the environment, though not used for learning. The standard workflow in Gymnasium involves initializing the environment to set up the initial state and obtain the first observation. The agent then interacts with the environment in a loop: it receives an observation, decides on an action, executes the action, and receives the next observation, a reward, and a done status indicating if the episode has ended, after which the environment is reset for a new episode.

Overview

We develop NS-Gym to allow researchers access to the breadth of NS-MDP specifications in the literature while maintaining the familiar interface popularized by the Gymnasium library (Towers et al. 2023). In its current version, NS-Gym provides a set of wrappers to augment the classic control suite of Gymnasium environments and three grid world environments. We refer to these Gymnasium environments (i.e., the stationary counterparts of the non-stationary environments we develop) as *base environments*. At a high level, each wrapper introduces non-stationarity by modifying some parameters that the base environment exposes. The modification potentially occurs at each decision epoch or through specific functions over decision epochs configured by the user. For example, in a deterministic environment such as the "CartPole" (we provide a detailed description of the environment in the technical appendix), an example change is varying the value of the gravity, thereby altering the dynamics of the cart. In stochastic environments, the probability distribution over possible next states, given the current state action pair, changes. For example, in the classic Frozen Lake environment (see a detailed description of the environment in the technical appendix), this change might increase (or decrease) the coefficient of friction, making the movement of the agent more (or less) uncertain. Figure 2 illustrates the high-level structure of the wrapper.

Problem Types and Notifications

A key feature of the NS-Gym library is its ability to manage the interaction between the environment and the decision-making agent. These interactions encapsulate the following *problem types*, which we explain using the Frozen Lake environment. Consider the problem setting in the Frozen Lake environment where the agent's probability of going in its indented direction is θ_1 in the base environment. Now, the lake becomes more slippery, and this probability changes to θ_2. We model the following settings.

1. Problems where the agent receives a message that the extent to which the lake is slippery has changed (corresponding to a successful anomaly detection), but it is unaware of the exact change (i.e., it does not know θ_2). This setting is motivated by prior work by Luo et al. (2024)).
2. Problems where the user is aware of the exact environmental change, i.e., it knows θ_2. However, in non-stationary settings, the agent might not have time to train a new policy from scratch. This setting is motivated by prior work by Pettet et al. (2024) and Lecarpentier and Rachelson (2019).
3. Problems where the agent is not notified about the change, i.e., it is unaware that the probability is no longer θ_1. This setting is motivated by prior work by Garivier and Moulines (2011).
4. In an orthogonal thread, we identify the frequency of the change, i.e., problems with a single change in an environment variable (Luo et al. 2024; Pettet et al. 2024) (e.g., the change is from θ_1 to θ_2) or multiple changes within an episode (Cheung, Simchi-Levi, and Zhu 2020) (e.g., the change is $\theta_1 \rightarrow \theta_2 \rightarrow \theta_3 \rightarrow \ldots$) or changes within multiple episodes (Chandak et al. 2020b).

Users can configure notifications the agent receives about changes in the NS-MDP at three distinct levels:[1]

1. **Basic Notification:** The agent receives a boolean flag indicating a change in an environment parameter.
2. **Detailed Notification:** In addition to the boolean flag, the agent is informed of the magnitude of the change.
3. **Full Environment Model:** Additionally, if the agent requires an environmental model for planning purposes (such as in Monte Carlo tree search), NS-Gym can provide a stationary snapshot of the environment. This snapshot aligns with the basic or detailed notification settings configured by the user. If the user seeks a model without detailed notification, the planning environment is a stationary snapshot of the base environment. Conversely, if detailed notifications are enabled, the agent receives the most up-to-date version of the environment model (but not any future evolutions).

Custom Observation for NS-MDPs

In building on top of Gymnasium, users familiar with the existing Gymnasium API can easily adapt to NS-Gym with minor modifications. Like Gymnasium, the agent-environment interaction consists of a sequence of steps where, at each step, the agent receives an *observation* and *reward*. In NS-Gym, we return custom observation and reward data types to accommodate information unique to non-stationary environments. Table 2 outlines the NS-Gym observation type. The observation type encapsulates information regarding the NS-MDP state and basic and detailed notification. The custom observation type consists of four fields: `state`, `env_change`, `delta_change`, and `relative_time`. The `state` field encodes the current state of the environment. The `env_change` field is a dictionary of boolean flags indicating what environment parameter has changed. The `delta_change` reports the amount of change in each environment parameter. By default, NS-gym returns the difference in value for scalar parameters and the Wasserstein distance for probability distributions. The `relative_time` is the number of decision epochs that have lapsed since the start of the environment episode. The reward type is similarly constructed, but instead of the `state` field, we have a `reward`.

Schedulers and Parameter Update Functions

We recognize that users may need to model non-stationarity differently depending on the specific problem settings. To accommodate this, NS-Gym allows users to specify which parameters change, when they change, and how they change through "schedulers" and parameter "update functions." We decouple the timing (and thereby, the frequency) and the manner of parameter changes, providing users with greater flexibility in designing experiments.

[1]Note that the *user* refers to the programmer using NS-Gym, as opposed to the autonomous *agent* that is being configured.

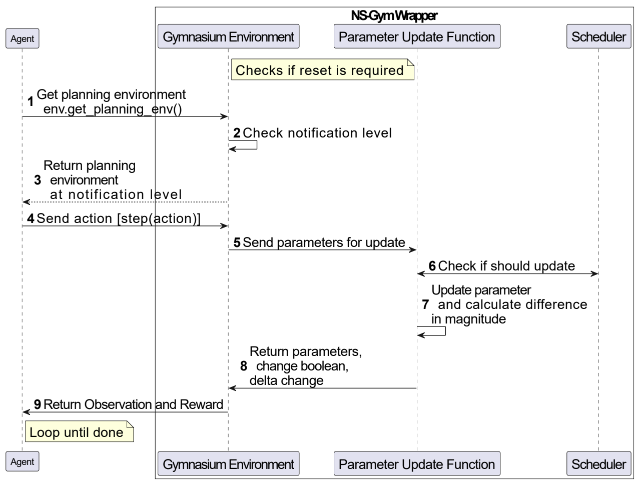
NS-Gym Wrapper
Agent
Gymnasium Environment
Parameter Update Function
Scheduler
Checks if reset is required
1 Get planning environment env.get_planning_env()
2 Check notification level
3 Return planning environment at notification level
4 Send action [step(action)]
5 Send parameters for update
6 Check if should update
7 Update parameter and calculate difference in magnitude
8 Return parameters, change boolean, delta change
9 Return Observation and Reward
Loop until done
Agent
Gymnasium Environment
Parameter Update Function
Scheduler

Figure 2: A sequence diagram of the agent-environment interaction in NS-Gym. Steps 4–9 in the diagram show how parameters are updated. Step 6 checks the current MDP time step and notifies if the parameter should be updated. Step 9 returns Observation and Reward types outlined in Table 2.

Observation Type

Field Name	**Data Type**
`state`	`Union[array,int]`
`env_change`	`Union[dict[str,bool],None]`
`delta_change`	`Union[dict[str,float],None]`
`relative_time`	`Union[int,float]`

Table 2: The custom observation types of NS-Gym capture essential components of NS-MDPs.

Schedulers in NS-Gym are a collection of functions that return a boolean flag at a given time step indicating whether environmental conditions *should* change. If a scheduler returns `True`, the update functions modify the specified parameter accordingly. NS-Gym includes schedulers that trigger continuous, stepwise, random, and periodic time steps. Users can easily implement custom schedulers by inheriting them from the base NS-Gym scheduler class. The parameter update functions determine how parameters change at time steps specified by the scheduler. Example update functions include a random walk with a budget-bounded constraint or a change bounded by Lipschitz Continuity.

Experimental Pipeline

This section illustrates the straightforward integration of the NS-Gym with the typical Gymnasium training pipeline. The general experimental setup procedure is: 1) *Create a Standard Gymnasium Environment:* Begin by making a standard Gymnasium environment. 2) *Define Parameters to Update:* Identify which environmental parameters will be updated during the experiment episode. 3) *Map Parameters to Schedulers and Update Functions:* Assign each parameter a scheduler and an update function. 4) *Generate a Non-Stationary Environment:* Pass the standard Gymnasium environment, along with the parameter mappings and update functions, into the NS-Gym wrapper to create a non-stationary Gymnasium-style environment.

Consider that the user seeks to model a non-stationary environment in the classical CartPole environment, where the pole's mass increases by 0.1 units at each time step, and the system's gravity increases through a random walk every three time steps. Furthermore, we want the decision-making agent to have a basic notification level. The following code snippet shows the general experimental setup in this CartPole Gymnasium environment using NS-Gym.

The first step involves importing the necessary modules from `ns_gym`, i.e.,

```
import gymnasium as gym
from ns_gym.wrappers import *
from ns_gym.schedulers import *
from ns_gym.update_functions import *
```

Next, we create the base gymnasium environment, i.e.,

```
env = gym.make("CartPole-v1")
```

Next, to describe the evolution of the non-stationary parameters, we define the two schedulers and update functions that model the semi-Markov chain over the relevant parameters.

```
scheduler_1 = ContinuousScheduler()
scheduler_2 = PeriodicScheduler(period =
     3)
U_Fn_1 = IncrementUpdate(scheduler_1,k =
     0.1)
U_Fn_2 = RandomWalk(scheduler_2)
```

Next, we map the parameters to the update functions, i.e.,

```
tunable_params = {"masspole":U_Fn_1,"
    gravity": U_Fn_2}
```

Then, we set the notification level and pass the parameters and environment into the wrapper.

```
ns_env = NSClassicControlWrapper(env,
tunable_params,
change_notification=True)
obs,info = ns_env.reset()
```

Finally, we grab an environment model for planning, i.e.,

```
planning_env = ns_env.get_planning_env()
```

The supplementary material includes a detailed tutorial for users to interact with NS-Gym.

Non-Stationary Environment Details

Below, we describe environments included in NS-Gym and how we induce non-stationarity. We focus on observable parameters θ here (available to the NS-Gym wrapper) and present descriptions of the environments in the appendix.

1) **CartPole** Changes in gravity, the mass of the cart, the mass of the pole, the length of the pole, or the magnitude of the force applied to the cart can be made to create a non-stationary MDP. 2) **Mountain Car and Continuous Mountain Car** NS-Gym induces non-stationary effects by changing the gravity and force applied to the car. 3) **Acrobot** NS-Gym induces non-stationarity by altering the link lengths, link masses, center of mass position, and

link moment of inertia. 4) **Pendulum** NS-Gym induces non-stationarity by increasing the link mass or length. 5) **Frozen-Lake** NS-Gym induces non-stationarity by modifying the probability distribution over actions. 6) **CliffWalker** NS-Gym induces non-stationary by introducing stochastic transitions that vary with time. 7) **Bridge** Originally proposed by Lecarpentier and Rachelson (2019), the Bridge environment has two probability distributions for the left and right halves of the grid world. NS-Gym at each decision epoch can make either or both halves of the map more or less slippery.

Benchmark Experiments

In this section, we demonstrate the utility of this package by evaluating decision-making algorithms in environments built using the NS-Gym library. Our experimental setup is designed to assess agent performance across multiple dimensions, providing insights into which decision-making agents are best suited for practical challenges. Consider a system modeled as a known MDP, where an exogenous force induces changes in the MDP's transition function. Specifically, we seek to explore the following questions: how effectively can an agent adapt when this change is known or unknown? What if the system undergoes continuous evolution? How well can an agent handle frequent updates?

We benchmark six algorithms across four base environments. We consider settings where the MDP transition function changes at a single discrete instance and for cases in which the transition function changes from some continuous sequence of time steps. Additionally, for each environment and agent pair, we consider instances with no notification and access to either the up-to-date environment model or a basic notification level. We evaluate agent performance by comparing cumulative undiscounted episodic rewards.

In this paper, we benchmark the CartPole, FrozenLake, CliffWalker, and Bridge environments. For the CartPole environment, we vary the mass of the cart's pole in single and continuous experiments. In the three grid-world environments, we adjust the probability of moving in the intended direction, with corresponding updates to the probabilities of moving in other directions. In the single experiments, the probability shifts from a default value to either 0.4, 0.6, or 0.8. In the continuous change case, the probability decreases by a fixed constant at each decision epoch until a lower threshold is met. Additional details on environment setup are provided in the appendix.

Baseline Algorithms

We evaluate the non-stationary environment across six different decision-making agents: Monte Carlo tree search (MCTS), double deep Q learning (DDQN), AlphaZero, adaptive Monte Carlo tree search, risk-averse tree search (RATS), and policy-augmented Monte Carlo tree search (PA-MCTS). Note that our work is the *first effort to benchmark approaches for tackling non-stationarity on standardized problem settings*. We briefly describe the benchmark approaches below. For all environments, we provide the algorithms that require a model of the environment with a stationary snapshot of the model for planning according to the appropriate notification level.

1) **MCTS** is an anytime online search algorithm that uses a model of the environment to select optimal actions. We use the Upper Confidence bound for Trees (UTC) algorithm (Kocsis and Szepesvári 2006) with random rollouts.
2) The **AlphaZero** algorithm (Silver et al. 2017) is a general game-playing algorithm that combines tree search with a deep value and policy neural network. The policy network is learned through self-play. We train the AlphaZero policy network on a stationary version and the environment but evaluate the agent on an NS-MDP. At each decision epoch the AlphaZero agent receives an environment model for planning at the appropriate notification level.
3) We include the widely popular **DDQN** approach as a pure reinforcement learning method (van Hasselt, Guez, and Silver 2015). In the "with notification" experiments, we let the DDQN do *some* gradient update steps using the most up-to-date model of the MDP (to resemble the baseline setting used by (Pettet et al. 2024)).
4) **ADA-MCTS** as a heuristic tree search algorithm that learns the environmental dynamics and *acts as it learns* Luo et al. (2024). ADA-MCTS uses a risk-averse strategy to explore the environment safely by balancing epistemic and aleatoric uncertainties. In our experiments, we only benchmarked ADA-MCTS when the updated environmental parameters are unavailable, as its core lies in learning about the updated change through environmental interactions.
5) The **RATS** algorithm proposed by Lecarpentier and Rachelson (2019) uses a minimax search strategy to act in a risk-averse manner to future environmental changes. The approach was originally designed against changes bounded by Lipschitz continuity.
6) We benchmark the **Policy-Augmented-MCTS** algorithm from Pettet et al. (2024), which computes a convex combination of returns generated through online search and a stale policy. Crucially, this combination occurs *outside the tree* (as opposed to the AlphaZero algorithm). Using the estimates outside the tree stabilizes the search under non-stationarity and has faster convergence. We consider PAMCTS performance across three α values, 0.25, 0.5, and 0.75, which control the extent to which the stale policy is preferred over online search.

Results

Table 3 shows results from the single change experiments without notifications, and Table 4 reports agent performance in the continuous experiment setting with and without notification. We provide a complete table of experimental results and figures in the supplemental materials. Building on the unified design of NS-Gym and the benchmark results, we have derived some key insights about how different strategies perform under varying conditions. This analysis provides a clearer understanding of how algorithms respond to dynamic environmental changes.

Impact of Detailed Notification on Performance with Single Transition Change: The presence of detailed notifications generally enhances the performance of most methods. AlphaZero, MCTS, PA-MCTS, and RATS demonstrate marked improvements when notifications are available in

		MCTS	AlphaZero	DDQN	PAMCTS	PAMCTS	PAMCTS	ADA-MCTS	RATS
					0.25	**0.5**	**0.75**		
	0.4	-0.58 ± 0.47	-0.26 ± 0.56	-0.82 ± 0.33	-0.58 ± 0.27	-0.20 ± 0.33	**-0.16** ± **0.33**	-0.54 ± 0.07	-0.98 ± 0.02
Bridge	0.6	-0.18 ± 0.57	**0.58** ± **0.47**	-0.78 ± 0.36	0.46±0.33	0.46 ± 0.3	0.38 ± 0.31	-0.16 ± 0.09	0.05 ± 0.08
	0.8	0.64 ± 0.45	**0.92** ± **0.23**	-0.72 ± 0.4	0.4 ± 0.31	0.72 ± 0.23	0.8 ± 0.2	0.46 ± 0.09	-0.01 ± 0.01
Frozen	0.4	0.11 ± 0.18	0.06 ± 0.02	0.22 ± 0.17	0.15 ± 0.04	0.16 ± 0.03	0.12 ± 0.03	**0.67** ± **0.05**	0.6 ± 0.05
	0.6	0.25 ± 0.25	0.25 ± 0.04	0.66 ± 0.19	0.3 ± 0.05	0.33 ± 0.05	0.27 ± 0.04	0.56± 0.05	**0.88** ± **0.03**
Lake	0.8	0.53 ± 0.29	0.39 ± 0.05	0.91 ± 0.12	0.74 ± 0.04	0.68 ± 0.05	0.54 ± 0.05	0.49 ± 0.05	**0.97** ± **0.02**
Cliff	0.4	-1593.89 ± 68.9	-543.94 ± 45.98	-1742.54 ± 91.29	-1572.21 ± 60.82	**-477.50** ± **54.66**	-1382.04 ± 77.88	-1503.34 ± 53.57	-777.55 +/- 31.19
	0.6	-1216.72 ± 63.68	**6.97** ± **8.2**	-1018.27 ± 96.95	-1159.77 ± 53.85	-374.64 ± 44.31	-477.50 ± 54.65	-1019.72 ± 35.99	-314.84 ± 12.8
Walking	0.8	-773.62 ± 54.67	**64.41** ± **3.44**	-287.17 ± 40.55	-790.60 ± 46.66	-54.22 ± 14.25	-109.08 ± 25.99	-523.73 ± 23.79	-231.86 ± 4.22
Cart	1	**600.90** ± **47.68**	441.1 ± 51.96	135.53 ± 0.28	525.98 ± 31.91	120.48 ± 0.57	135.41 ± 0.32	–	–
Pole	1.5	**641.28** ± **50.47**	272.82 ± 21.25	139.19 ± 0.27	467.35 ± 25.11	117.60 ± 1.24	135.42 ± 0.34	–	–

Table 3: Mean episode reward with standard error for an agent in an environment with a single exogenous change without notification. The best-performing agents are in bold. Blanks denote settings where the algorithm is not applicable.

		MCTS	AlphaZero	DDQN	PAMCTS	PAMCTS	PAMCTS	ADA-MCTS	RATS
					0.25	**0.5**	**0.75**		
Bridge	WN	0.18 ± 0.1	**0.6** ± **0.08**	-0.44 +- 0.09	0.28 ± 0.56	0.34 ± 0.54	0.08 +/ 0.56	–	0.36 ± 0.09
	WON	0.04 ± 0.10	**1.00** ± **0.00**	-0.84 ± 0.05	-0.02 ± 0.58	0.22 ± 0.57	0.20 ± 0.57	0.08 ± 0.1	0.36 ± 0.09
Frozen	WN	0.15 ± 0.04	0.25 ± 0.04	0.1 ± .04	0.2 ± 0.04	0.15 ± 0.04	0.04 ± 0.02	–	**0.71** ± **0.05**
Lake	WON	0.24 ± 0.04	0.25 ± 0.04	0.27 ± 0.04	0.14 ± 0.03	0.21 ± 0.04	0.08 ± 0.03	0.59 ± 0.05	**0.71** ± **0.05**
Cliff	WN	-847.48 ± 55.83	**77.95** ± **0.40**	-137.89 ± 29.19	-803.94 ± 54.89	-56.56 ± 19.2	-75.06 ± 20.77	–	-932.89 ± 50.55
Walking	WON	-907.67 ± 54.62	**76.0** ± **1.89**	-359.97 ± 42.46	-732.28 ± 53.50	-31.84 ± 14.97	-132.26 ± 26.98	-1144.91 ± 43.83	-707.65 ± 36.33
Cart	WN	702.7 ± 21.95	203.68 ± 1.35	100.78 ± 2.62	**1392.23** ± **65.57**	96.15 ± 2.5	99.95 ± 2.58	–	–
Pole	WON	149.0 ± 1.79	**251.47** ± **5.81**	95.97 ± 2.68	109.39 ± 2.69	55.17 ± 1.7	95.61± 2.73	–	–

Table 4: Mean episode reward with standard error for with agent in an environment with continuous parameter updates. WO and WON denote settings "with notification" and "without notification" respectively. The best-performing approaches are in bold. Blanks denote settings where the algorithm is not applicable.

some environments, effectively leveraging the most up-to-date dynamics to optimize decision-making processes. In contrast, DDQN shows only a modest improvement as it is difficult to adapt to changes in limited time.

Impact of Notification on Performance with continuous Transition Change: Again, the presence of detailed notifications generally improves the performance of most methods across various environments. This highlights the importance of quickly adapting the planning model to the latest dynamics of the environment. For example, methods like MCTS and PAMCTS, which leverage online search, show a consistent performance increase across different environments, emphasizing the effectiveness of an online approach in maintaining robust performance amid continuous changes when notifications are given. We observe that AlphaZero performs exceptionally well with notifications.

Variability in Algorithm Effectiveness: When comparing methods that incorporate risk-averse strategies with those that do not, it is evident that the ones with risk-averse strategies perform differently. In environments like FrozenLake, where the agent is more vulnerable to varying levels of unpredictability compared to other environments, methods like ADA-MCTS and RATS, which incorporate risk-averse strategies, generally perform better with single transition changes and continuous changes. These methods are designed to account for and mitigate the risks brought on by the environment's stochastic nature, leveraging worst-case sampling strategies to make decisions robust to possible changes. This enables them to navigate more effectively and avoid the pitfalls that non-risk-averse methods might encounter. We also point out that in prior work, ADA-MCTS is the only approach that that *learn* the updated environmental parameter through environmental interactions.

Conclusion

We present NS-Gym, the first simulation toolkit and set of standardized problem instances and interfaces explicitly designed for NS-MDPs. NS-Gym incorporates problem types and features from over fifty years of research in non-stationary decision-making. We also present benchmark results using prior work. We will continue to maintain NS-Gym, extend it, and maintain a leaderboard of approaches.

References

Ackerson, G.; and Fu, K. 1970. On state estimation in switching environments. *IEEE transactions on automatic control*, 15(1): 10–17.

Auer, P.; Jaksch, T.; and Ortner, R. 2008. Near-optimal regret bounds for reinforcement learning. *Advances in neural information processing systems*, 21.

Besbes, O.; Gur, Y.; and Zeevi, A. 2014. Stochastic multi-armed-bandit problem with non-stationary rewards. *Advances in neural information processing systems*, 27.

Campo, L.; Mookerjee, P.; and Bar-Shalom, Y. 1991. State estimation for systems with sojourn-time-dependent Markov model switching. *IEEE Transactions on Automatic Control*, 36(2): 238–243.

Chakraborty, T.; and Shettiwar, P. 2024. Non Stationary Bandits with Periodic Variation. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 2177–2179.

Chandak, Y.; Jordan, S.; Theocharous, G.; White, M.; and Thomas, P. S. 2020a. Towards safe policy improvement for non-stationary mdps. *Advances in Neural Information Processing Systems*, 33: 9156–9168.

Chandak, Y.; Theocharous, G.; Shankar, S.; Mahadevan, S.; White, M.; and Thomas, P. S. 2020b. Optimizing for the Future in Non-Stationary MDPs. *Thirty-seventh International Conference on Machine Learning (ICML)*.

Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3): 1–58.

Cheung, W. C.; Simchi-Levi, D.; and Zhu, R. 2020. Reinforcement learning for non-stationary markov decision processes: The blessing of (more) optimism. In *International Conference on Machine Learning*, 1843–1854. PMLR.

Garivier, A.; and Moulines, E. 2011. On upper-confidence bound policies for switching bandit problems. In *International Conference on Algorithmic Learning Theory*, 174–188. Springer.

Hu, Q.; and Yue, W. 2007. *Markov decision processes with their applications*, volume 14. Springer Science & Business Media.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A. A.; Yogamani, S.; and Pérez, P. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6): 4909–4926.

Kochenderfer, M. J.; Wheeler, T. A.; and Wray, K. H. 2022. *Algorithms for decision making*. MIT press.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *Machine Learning: ECML 2006*, 282–293. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.

Lecarpentier, E.; and Rachelson, E. 2019. Non-stationary Markov decision processes, a worst-case approach using model-based reinforcement learning. *Advances in neural information processing systems*, 32.

Li, S.; Yan, Z.; and Wu, C. 2021. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34: 26198–26211.

Luo, B.; Zhang, Y.; Dubey, A.; and Mukhopadhyay, A. 2024. Act as You Learn: Adaptive Decision-Making in Non-Stationary Markov Decision Processes. *arXiv preprint arXiv:2401.01841*.

Mukhopadhyay, A.; Pettet, G.; Vazirizade, S. M.; Lu, D.; Jaimes, A.; El Said, S.; Baroud, H.; Vorobeychik, Y.; Kochenderfer, M.; and Dubey, A. 2022. A review of incident prediction, resource allocation, and dispatch models for emergency management. *Accident Analysis & Prevention*, 165: 106501.

Pendharkar, P. C.; and Cusatis, P. 2018. Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, 103: 1–13.

Pettet, A.; Zhang, Y.; Luo, B.; Wray, K.; Baier, H.; Laszka, A.; Dubey, A.; and Mukhopadhyay, A. 2024. Decision Making in Non-Stationary Environments with Policy-Augmented Search. *arXiv preprint arXiv:2401.03197*.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. ArXiv:1712.01815 [cs].

Towers, M.; Terry, J. K.; Kwiatkowski, A.; Balis, J. U.; Cola, G. d.; Deleu, T.; Goulão, M.; Kallinteris, A.; KG, A.; Krimmel, M.; Perez-Vicente, R.; Pierré, A.; Schulhoff, S.; Tai, J. J.; Shen, A. T. J.; and Younis, O. G. 2023. Gymnasium.

van Hasselt, H.; Guez, A.; and Silver, D. 2015. Deep Reinforcement Learning with Double Q-learning. *CoRR*, abs/1509.06461.

Yu, C.; Liu, J.; Nemati, S.; and Yin, G. 2021. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1): 1–36.

Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced: **yes**
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results: **yes**
- Provides well marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper: **yes**

Does this paper make theoretical contributions? **no**

Does this paper rely on one or more datasets? **no**

Does this paper include computational experiments? **yes**
If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix: **yes**
- All source code required for conducting and analyzing the experiments is included in a code appendix: **yes**
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes: **yes**
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from: **yes**
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results: **yes**
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks: **yes**
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics: **yes**
- This paper states the number of algorithm runs used to compute each reported result: **yes**
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information: **yes**
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank): **no** (not applicable)
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments: **yes**
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting: **yes**

Description of NS-Gym Environments

Below, we provide descriptions for each environment supported by NS-Gym.

CartPole

The CartPole environment has a discrete action space and a continuous state space. As illustrated in Figure 3, the agent's objective is to keep the pole balanced on top of the cart for as long as possible. The agent receives a reward of +1 for each time step that the pole remains balanced. The state is represented by a four-dimensional vector, which includes the cart's position, cart's velocity, pole's angle, and pole's angular velocity. At each time step, the agent can apply a fixed force to push the cart either left or right.

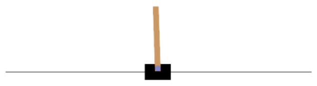

Figure 3: The Gymnasium CartPole environment.

Mountain Car

The MountainCar environment (see Figure 4) is a continuous state but discrete action space environment. In this environment, a car is stuck in a valley, and the agent must apply force to the cart to build momentum so that the car can escape. By default, the agent receives a zero reward for escaping the valley and a -1 reward otherwise. The agent can either push the car to the left, right, or not at all. The continuous Mountain Car environment is similar to the standard Mountain Car environment but with a continuous action space. In the continuous analog, the agent chooses the direction in which to apply the force to the car.

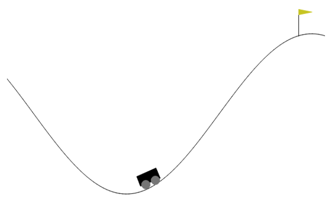

Figure 4: The Gymnasium MountainCar environment.

Acrobot The Acrobot environment is a double pendulum (see Figure 5). The agent can apply torque to the joint connecting the two links of the double pendulum to move the free end above a threshold height. At each time step, the agent can either apply +1, 0, or -1 units of torque.

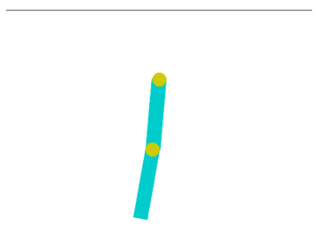

Figure 5: The Gymnasium Acrobot environment.

Pendulum The Pendulum environment is a continuous state and action space environment. The agent aims to keep the pendulum inverted for as long as possible. The agent receives a reward proportional to the pendulum's angle. At each time step, the agent applies some torque magnitude to the pendulum's free end. Figure 6 shows the pendulum environment.

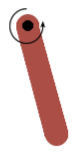

Figure 6: The Gymnasium Pendulum environment.

FrozenLake The FrozenLake environment (Figure 7) is a stochastic, discrete action, and discrete state space grid-world environment. The agent navigates from a starting cell in the top left corner of the map to a jail cell in the bottom right corner while avoiding holes in the "frozen lake." The agent can move in an intended direction, with some probability that it will move in a perpendicular direction instead. The Agent will get a reward of +1 if it reaches the goal and 0 otherwise.

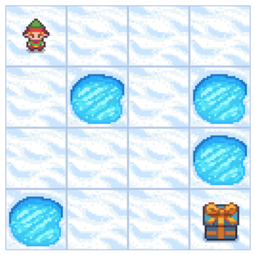

Figure 7: The Gymnasium FrozenLake environment.

CliffWalker The CliffWalking environment (Figure 8) is a deterministic grid-world environment. The agent must navi-

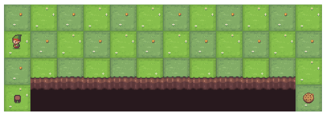

Figure 8: The Gymnasium CliffWalking environment.

gate from the start to the goal cell in the fewest steps. If the agent falls off a ”cliff,” it accrues a reward of -100 and resets at the start cell without ending the episode. The agent accrues -1 reward for each cell that is not a cliff or a goal state. The goal cell is the only terminal state. The agent can move up, down, left, and right.

Bridge The non-stationary bridge environment (Figure 9) is a grid-world setting where the agent must navigate from the starting cell to one of two goal cells. The environment was originally introduced by Lecarpentier and Rachelson (2019). To reach a goal cell, the agent must cross a “bridge” surrounded by terminal cells. The secondary goal cell is farther from the starting location but less risky because fewer holes surround it. Unlike the CliffWalking environment, which has a single global transition probability, the left and right halves of the Bridge map each have separate probability distributions. NS-Gym allows for updates to just the left or right halves of the map or to the global value. Similar to the FrozenLake environment, if the agent moves in some direction, there is some probability that is moves in one of the perpendicular directions instead. The agent receives a +1 reward for reaching a goal cell, a -1 reward for falling into a hole, and a 0 reward otherwise. Our version of the non-stationary bridge environment is not included in the standard Gymnasium Python package. We provide our implementation of the Bridge environment, as described by Lecarpentier and Rachelson (2019), as part of the NS-Gym package.

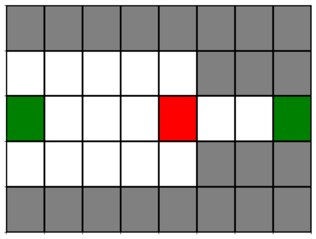

Figure 9: The Bridge environment. The start cell is in red, the two goals are in green, and the terminal ”holes” are in gray.

Experimental Setup

In this section, we elaborate on how we set up the single and continuous change experiments for each environment.

CartPole

- **Single update case**: We initialize the CartPole environment to its default state. After the first decision epoch, we increase the mass of the pole from 0.1 to a value of 1.0 and 1.5.
- **Continuous update case**: We initialize the CartPole environment to its default state. After each decision epoch, we increase the mass of the pole by 0.1.

We truncate the episode after 2500 episode steps if the agent does not reach a terminal state.

FrozenLake

- **Single update case**: We initially set the probability of moving in the intended direction to 0.7 and the probability of moving in each perpendicular direction to 0.15. After the first decision epoch, we change the probability of moving in the intended direction to 0.4, 0.6, or 0.8. We update the chance of moving in a perpendicular direction accordingly.
- **Continuous update case**: We initialize the FrozenLake environment to be completely deterministic. We decrease the chance of moving in the intended direction by 0.2 for the first three decision epochs. We update the chance of moving in a perpendicular direction accordingly.

We truncate the episode after 100 episode steps if the agent does not reach a terminal state.

CliffWalking

- **Single update case**: We initialize the environment to be determenistic. After the first decision epoch, we update the transition probability to a value of 0.8, 0.6, or 0.4. The probability of moving in the perpendicular and reverse directions is updated accordingly.
- **Continuous update case**: We initialize the environment to be deterministic. For the first 10 decision epochs, we decrease the chance of moving in the intended direction by 0.02. The probabilities of moving in the perpendicular and reverse directions are updated accordingly.

In our experimental setup we modify the standard CliffWalking rewards so that the goal state has a reward of +100. Additionally, after 200 decision epochs, if the agent has not found the goal, we truncate the episode.

Bridge

- **Single update case**: We initially set the probability of moving in the intended direction to 0.7 and the probability of moving in each of the perpendicular directions to 0.15. After the first decision epoch, we change the probability of moving in the intended direction to a value of 0.4, 0.6, or 0.8. We update the chance of moving in a perpendicular direction accordingly.
- **Continuous update case**: We initialize the environment to be determenistic. At each decision epoch, the probability of going in the intended direction decreases by 0.1.

We truncate the episode after 200 steps if the agent does not reach a terminal state.

Algorithm Parameters

The Tables 5, 6, 7, 8, 9, and 10 show the parameters used in each experiment.

Single

	Bridge	FrozenLake	CliffWalking	CartPole
m	500	300	1000	300
d	100	100	200	500
c	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$
γ	0.99	0.99	0.999	0.5

Continuous

	Bridge	FrozenLake	CliffWalking	CartPole
m	500	300	1000	300
d	100	100	200	500
c	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$
γ	0.99	0.99	0.999	0.5

Table 5: MCTS parameters for the single and continuous change experiments, where m is the number of MCTS iterations, d is the maximum rollout depth, c is the exploration parameter, γ is the tree discount factor.

	Bridge	FrozenLake	CliffWalking	CartPole
m	500	300	300	500
c	$\sqrt{2}$	1.44	1.44	$\sqrt{2}$
γ	0.99	0.999	0.999	1
layers	3	3	3	2
units	64	64	64	128
α	1	1	5	1
ϵ	0	0	0.75	0

Table 6: AlphaZero parameters for the single and continuous change experiments, where m is the number of MCTS iterations, c is the exploration parameter, γ is the tree discount factor, layers are the number of hidden layers in the neural network, and units are the number of units in each hidden layer. The parameter α is the concentration parameter for the Dirichlet noise added to the priors in the root node of the search tree. The parameter ϵ controls the amount of noise added to the priors.

	Bridge	FrozenLake	CliffWalking	CartPole
layers	3	2	2	2
units	64	64	128	64
time	0.4	0.4	0.4	0.4

Table 7: DDQN parameters for both the single and continuous change experiments. The parameter layers are the number of hidden layers in the DDQN network. The parameter units are the number of units in each layer. In the "with" notification experiments, the time is the number of seconds the agent has to collect data and do gradient updates.

	Bridge	FrozenLake	CliffWalking	CartPole
m	500	1000	1000	300
d	200	500	200	500
c	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$
γ	0.99	0.99	0.999	1
layers	3	2	2	2
units	64	64	128	64

Table 8: PAMCTS experiment parameters for single and continuous experiments, where m is the number of MCTS iterations, d is the MCTS search depth, c is the exploration parameter, γ is the discount factor, layers are the number of layers in the DDQN, and units are the number of units in each hidden layer.

	Bridge	FrozenLake	CliffWalking
γ	0.99	0.99	0.99
d	3	3	3

Table 9: RATS algorithm parameters. γ is the discount factor and d is the tree search depth.

	Bridge	FrozenLake	CliffWalking
γ	0.99	0.99	0.99
m	3000	100	3000

Table 10: ADA-MCTS algorithm parameters. γ is the discount factor and m is the number of iterations.

Experimental Results

In this section, we include additional experimental results and figures. Table 11 shows the complete results for the single change with and without notification experiments. Figures 10 , 11, 12, and 13 show the comparative performance of each decision-making agent in the single change experiments. Figures 14, 15, 16, and 17 show the comparative performance between all agents in the continuous change case.

Single Transition Change With and Without Notification

		MCTS	**AlphaZero**	**DDQN**	**PAMCTS**	**PAMCTS**	**PAMCTS**	**ADA-MCTS**	**RATS**
					0.25	**0.5**	**0.75**		
					With Notification				
	0.4	-0.28 ± 0.56	-0.18 ± 0.1	-0.82 ± 0.33	-0.52 ± 0.29	-0.12 ± 0.33	-0.02 ± 0.33	–	**0.34** ± **0.09**
Bridge	0.6	-0.32 ± 0.55	**0.8** ± **0.06**	-0.80 ± 0.35	-0.10 ± 0.33	0.3 ± 0.32	0.46 ± 0.3	–	0.30 ± 0.09
	0.8	0.32 ± 0.55	**0.98** ± **0.02**	-0.90 ± 0.25	0.32 ± 0.2	0.84± 0.18	0.8 ± 0.2	–	0.08 ± 0.03
	0.4	0.09 ± 0.17	0.1 ± 0.03	0.2 ± 0.04	0.13±0.08	0.01 ± 0.02	0.07 ± 0.06	–	0.61 ± 0.05
FrozenLake	0.6	0.31 ± 0.27	0.21 ± 0.04	0.47 ± 0.05	0.34 ± 0.11	0.28 ± 0.11	0.35 ± 0.11	–	**0.86** ± **0.04**
	0.8	0.53 ± 0.29	0.51 ± 0.05	0.53 ± 0.05	0.62 ± 0.11	0.78 ± 0.10	0.66 ± 0.11	–	**0.97** ± **0.02**
	0.4	-1767.75 ± 61.69	**-588.23** ± **46.46**	-912.50 ± 42.39	-1668.47 ± 64.08	-1285.94 ± 71.43	-1419.56 ± 68.83	–	-1077.98 ± 48.82
CliffWalking	0.6	-1162.91 ± 62.46	**-0.48** ± **10.77**	-246.48 ± 2.08	-1184.65 ± 57.88	-495.81 ± 50.71	-543.45 ± 54.80	–	-400.72 ± 26.59
	0.8	-846.64 ± 53.13	**63.11** ± **3.53**	-20.89 ± 10.44	-852.95 ± 56.15	-43.06 ± 50.9	-136.81 ± 25.46	–	-245.54 ± 9.27
CartPole	1	633.62 ± 49.27	230.81 ± 1.06	92.8 ± 33.38 2	740.84 ± 43.23	122.89 ± 0.5	136.07 ± 0.29	–	–
	1.5	678.58 ± 51.13	**902.05** ± **83.01**	230.57 ± 21.39	702.58 ± 43.60	124.29 ± 0.47	135.22 ± 0.3	–	–
					Without Notification				
	0.4	-0.58 ± 0.47	-0.26 ± 0.56	-0.82 ± 0.33	-0.58 ± 0.27	-0.20 ± 0.33	**-0.16** ± **0.33**	-0.54 ± 0.07	-0.98 ± 0.02
Bridge	0.6	-0.18 ± 0.57	**0.58** ± **0.47**	-0.78 ± 0.36	0.46±0.33	0.46 ± 0.3	0.38 ± 0.31	-0.16 ± 0.09	0.05 ± 0.08
	0.8	0.64 ± 0.45	**0.92** ± **0.23**	-0.72 ± 0.4	0.4 ± 0.31	0.72 ± 0.23	0.8 ± 0.2	0.46 ± 0.09	-0.01 ± 0.01
	0.4	0.11 ± 0.18	0.06 ± 0.02	0.22 ± 0.17	0.15 ± 0.04	0.16 ± 0.03	0.12 ± 0.03	**0.67** ± **0.05**	0.6 ± 0.05
FrozenLake	0.6	0.25 ± 0.25	0.25 ± 0.04	0.66 ± 0.19	0.3 ± 0.05	0.33 ± 0.05	0.27 ± 0.04	0.56± 0.05	**0.88** ± **0.03**
	0.8	0.53 ± 0.29	0.39 ± 0.05	0.91 ± 0.12	0.74 ± 0.04	0.68 ± 0.05	0.54 ± 0.05	0.49 ± 0.05	**0.97** ± **0.02**
	0.4	-1593.89 ± 68.9	-543.94 ± 45.98	-1742.54 ± 91.29	-1572.21 ± 60.82	**-477.50** ± **54.66**	-1382.04 ± 77.88	-1503.34 ± 53.57	-777.55 +/- 31.19
CliffWalking	0.6	-1216.72 ± 63.68	**6.97** ± **8.2**	-1018.27 ± 96.95	-1159.77 ± 53.85	-374.64 ± 44.31	-477.50 ± 54.65	-1019.72 ± 35.99	-314.84 ± 12.8
	0.8	-773.62 ± 54.67	**64.41** ± **3.44**	-287.17 ± 40.55	-790.60 ± 46.66	-54.22 ± 14.25	-109.08 ± 25.99	-523.73 ± 23.79	-231.86 ± 4.22
CartPole	1	**600.90** ± **47.68**	441.1 ± 51.96	135.53 ± 0.28	525.98 ± 31.91	120.48 ± 0.57	135.41 ± 0.32	–	–
	1.5	**641.28** ± **50.47**	272.82 ± 21.25	139.19 ± 0.27	467.35 ± 25.11	117.60 ± 1.24	135.42 ± 0.34	–	–

Table 11: Table of mean rewards and standard error across for the single change environmental parameter change experiment. The best-performing agents for each environment are in bold.

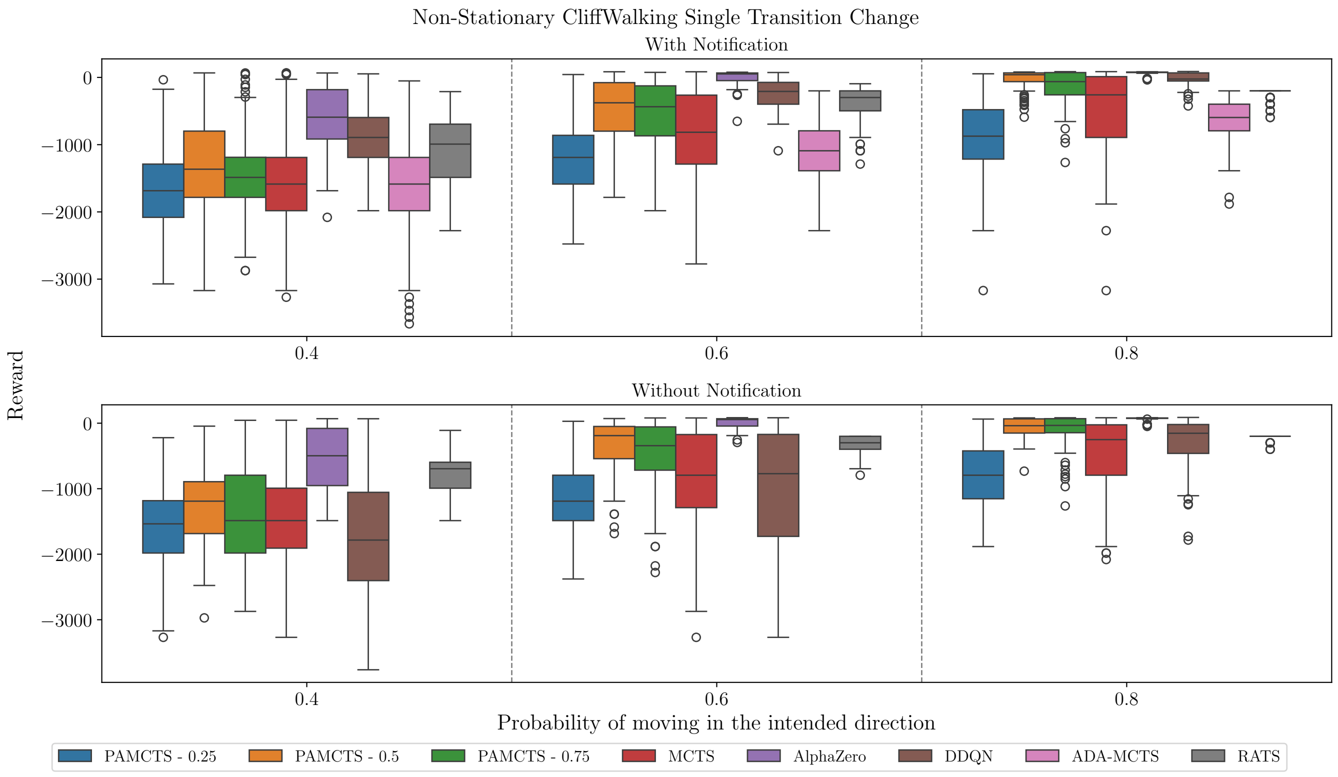

Figure 10: Distribution of rewards for the CliffWalking experiments with a single change.

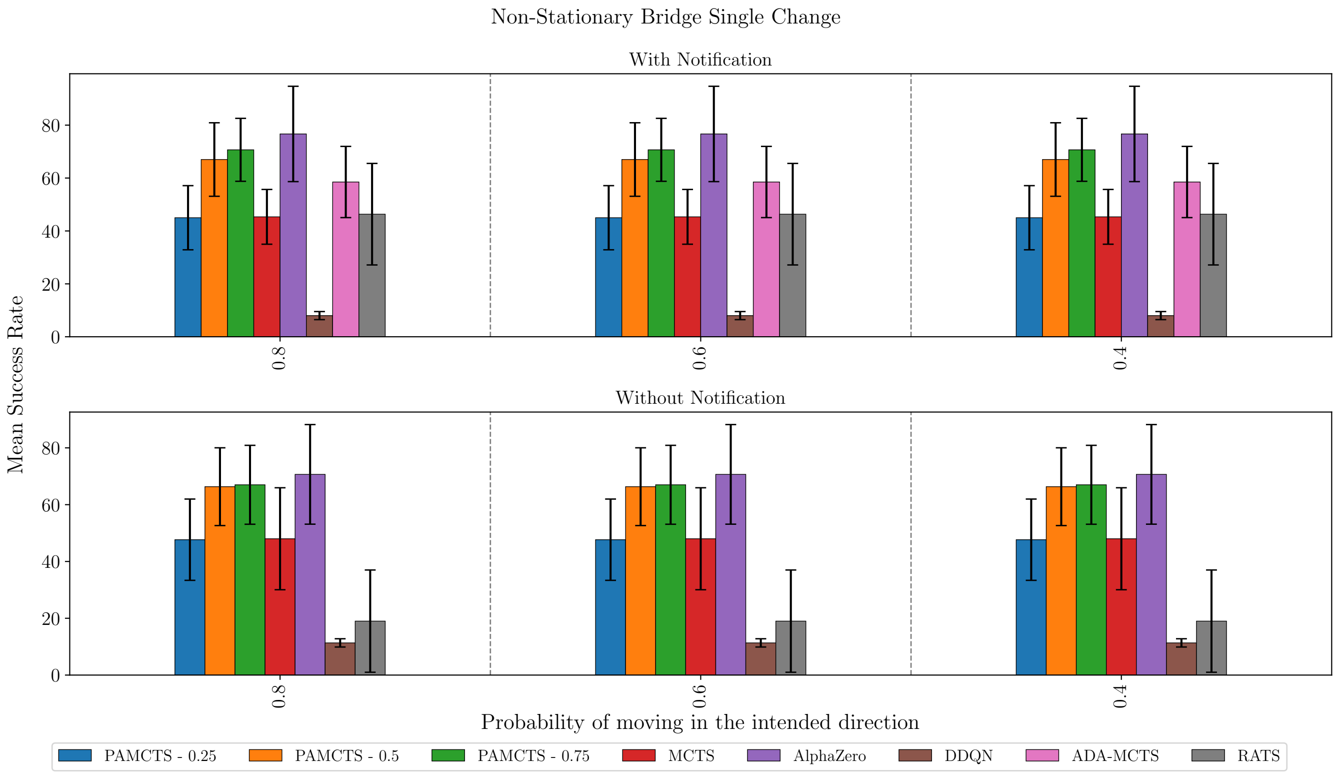

Figure 11: Average success rate (i.e., the agent finds the goal state) for each agent in the single change experiments.

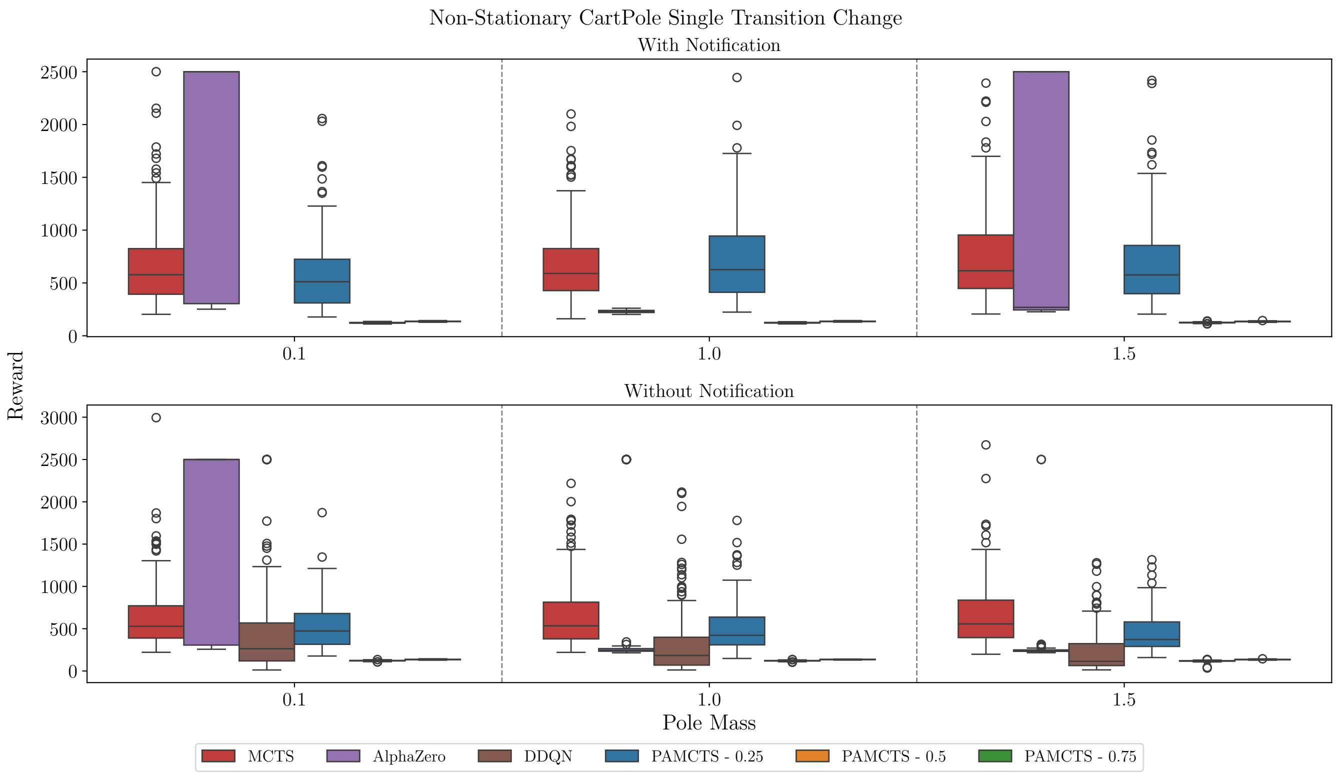

Figure 12: Distribution of episode rewards for each agent tested on non-stationary CartPole environment with and without notification.

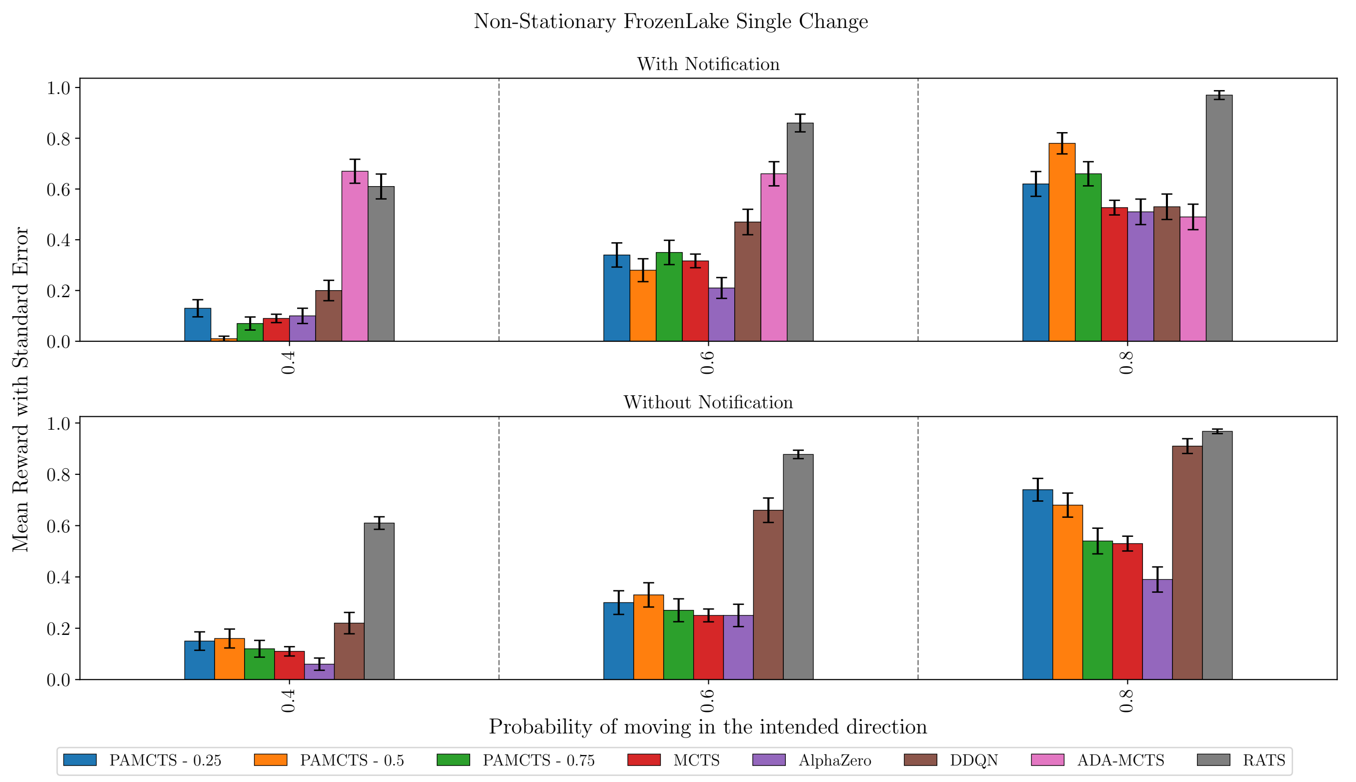

Figure 13: Mean episode reward and standard error for each agent in a non-stationary FrozenLake environment with a single change in its transition function.

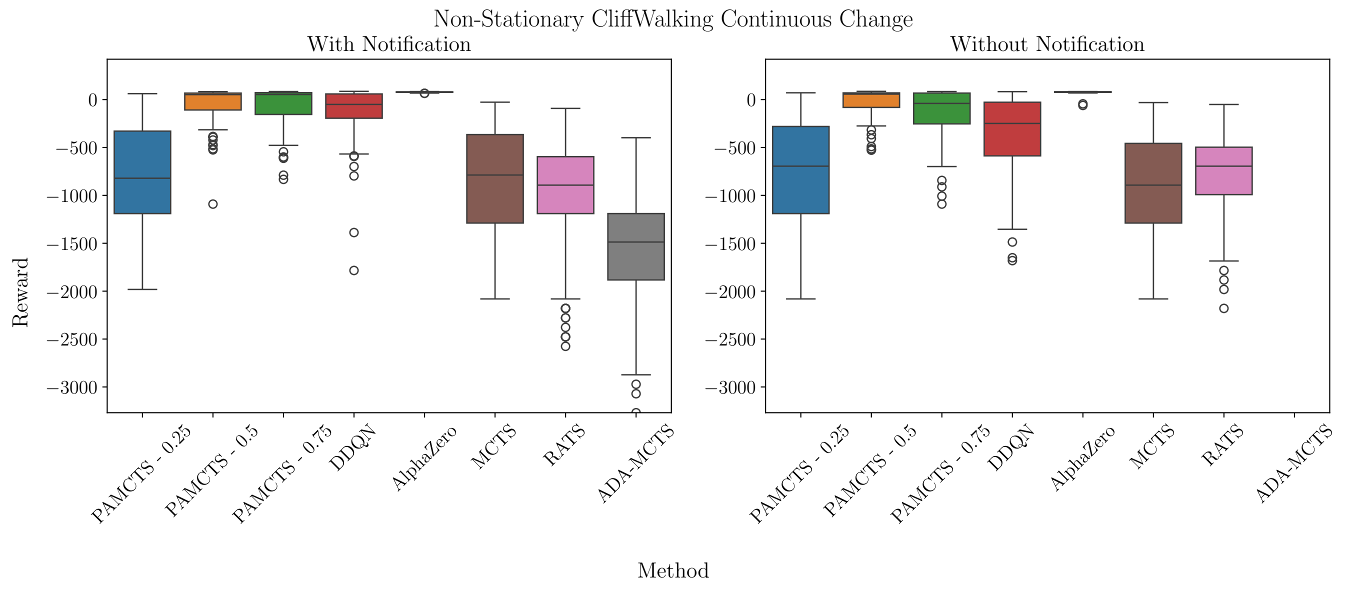

Figure 14: Distribution of episode reward for each agent under the continuous change experiment conditions.

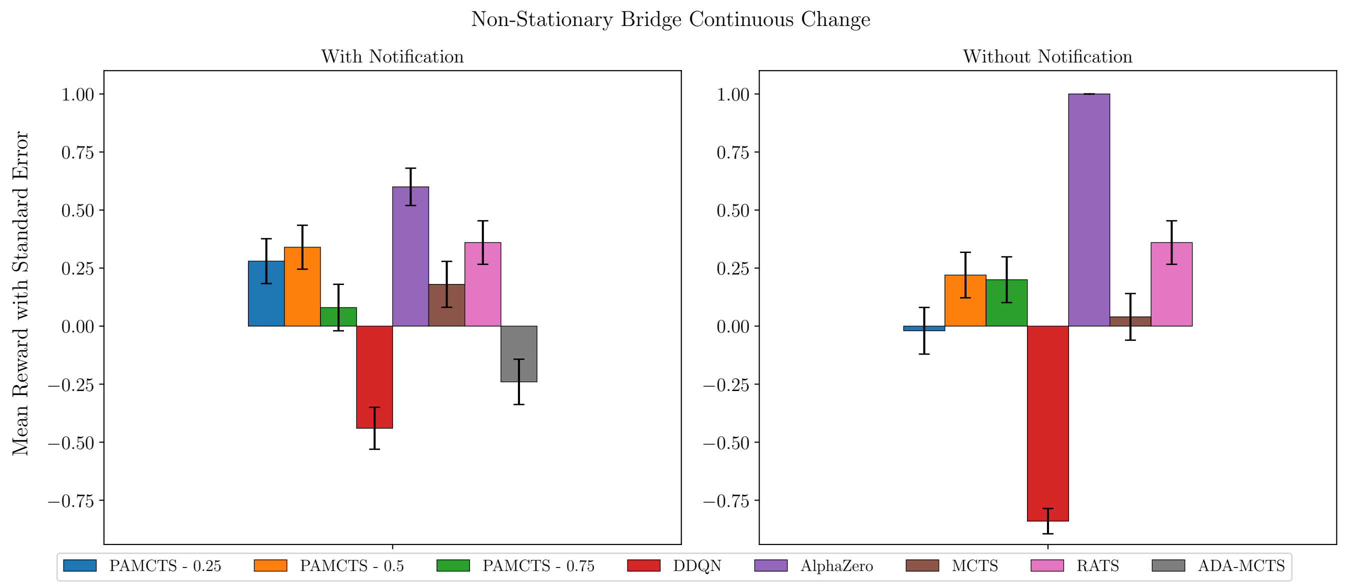

Figure 15: Mean reward and standard error for agents in the non-stationary Bridge environment under the continuous change conditions.

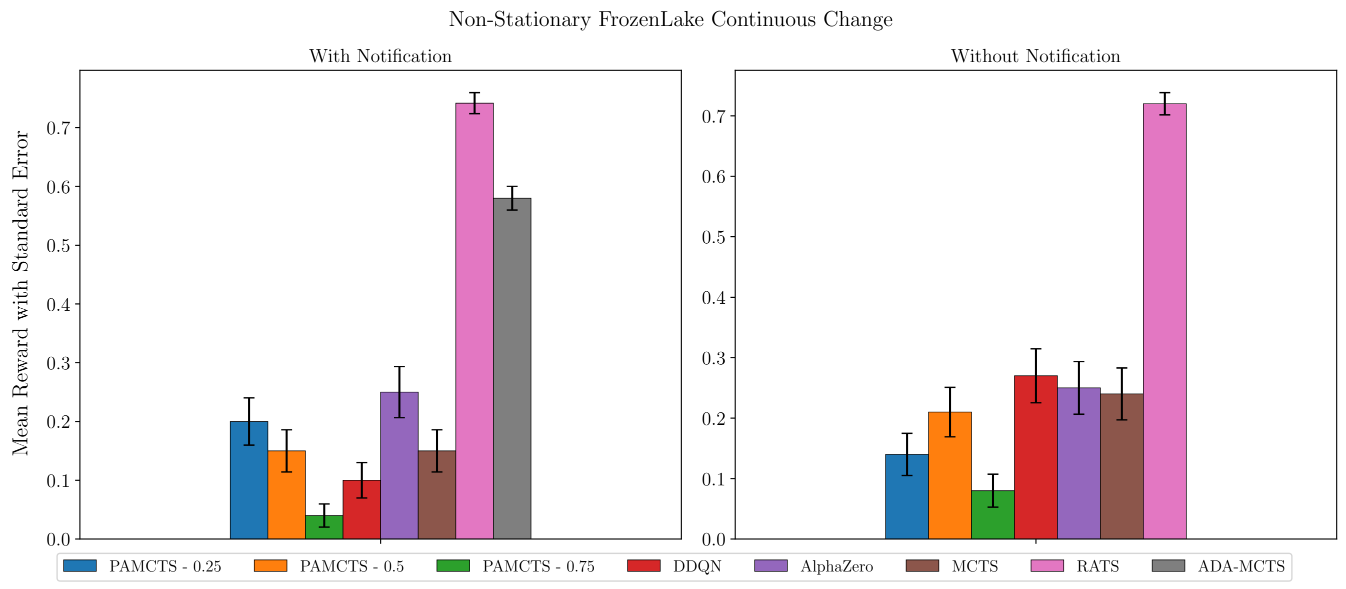

Figure 16: Mean reward and standard error for agents in the non-stationary FrozenLake environment under continuous change conditions.

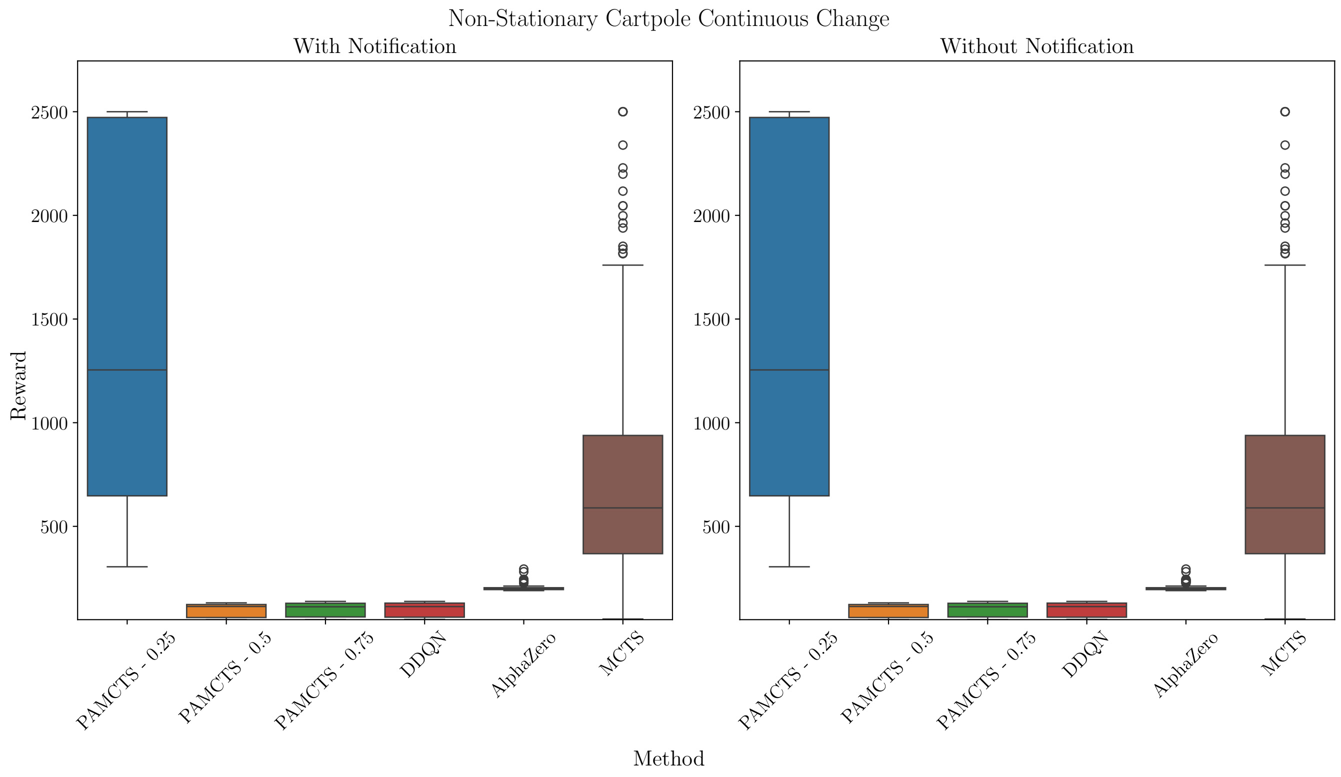

Figure 17: Distribution of episode rewards for agents in the continuous non-stationary CartPole environment.