Decision Making in Non-Stationary Environments with Policy-Augmented Search

Ava Pettet[†]
Vanderbilt University
Nashville, USA
ava.pettet@vanderbilt.edu

Yunuo Zhang[†]
Vanderbilt University
Nashville, USA
yunuo.zhang@vanderbilt.edu

Baiting Luo
Vanderbilt University
Nashville, USA
baiting.luo@vanderbilt.edu

Kyle Wray
Stanford University
Stanford, USA
kyle.hollins.wray@gmail.com

Hendrik Baier
Eindhoven University of Technology
Eindhoven, Netherlands
h.baier@tue.nl

Aron Laszka
Pennsylvania State University
University Park, USA
aron.laszka@psu.edu

Abhishek Dubey
Vanderbilt University
Nashville, USA
abhishek.dubey@vanderbilt.edu

Ayan Mukhopadhyay
Vanderbilt University
Nashville, USA
ayan.mukhopadhyay@vanderbilt.edu

ABSTRACT

Sequential decision-making under uncertainty is present in many important problems. Two popular approaches for tackling such problems are reinforcement learning and online search (e.g., Monte Carlo tree search). While the former learns a policy by interacting with the environment (typically done before execution), the latter uses a generative model of the environment to sample promising action trajectories at decision time. Decision-making is particularly challenging in non-stationary environments, where the environment in which an agent operates can change over time. Both approaches have shortcomings in such settings—on the one hand, policies learned before execution become stale when the environment changes and relearning takes both time and computational effort. Online search, on the other hand, can return sub-optimal actions when there are limitations on allowed runtime. In this paper, we introduce *Policy-Augmented Monte Carlo tree search* (PA-MCTS), which combines action-value estimates from an out-of-date policy with an online search using an up-to-date model of the environment. We prove theoretical results showing conditions under which PA-MCTS selects the one-step optimal action and also bound the error accrued while following PA-MCTS as a policy. We compare and contrast our approach with AlphaZero, another hybrid planning approach, and Deep Q Learning on several OpenAI Gym environments. Through extensive experiments, we show that under non-stationary settings with limited time constraints, PA-MCTS outperforms these baselines.

KEYWORDS

Sequential Decision-Making, Non-Stationary Environments, Monte Carlo Tree Search

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). This work is licenced under the Creative Commons Attribution 4.0 International (CC-BY 4.0) licence.

ACM Reference Format:
Ava Pettet[†], Yunuo Zhang[†], Baiting Luo, Kyle Wray, Hendrik Baier, Aron Laszka, Abhishek Dubey, and Ayan Mukhopadhyay. 2024. Decision Making in Non-Stationary Environments with Policy-Augmented Search. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 29 pages.

1 INTRODUCTION

Sequential decision-making is present in many important problem domains, such as autonomous driving [3], emergency response [17], and medical diagnosis [2]. An open challenge in such settings is non-stationary environments, where the dynamics of the environment can change over time. A decision agent must adapt to these changes to avoid taking sub-optimal actions. Two well-known approaches for sequential decision-making are reinforcement learning (RL) and online planning [13]. In RL approaches, an agent learns a policy π, i.e., a mapping from states to actions, through interacting with the environment. The learning can also take place before execution using environmental models. Once a policy is learned, it can be invoked nearly instantaneously at decision time. Deep RL methods, which use a neural network as a function approximator for the policy, have achieved state-of-the-art performance in many applications [13, 22]. However, when faced with non-stationary environments, a policy can become stale and result in sub-optimal decisions. Moreover, retraining the policy on the new environment takes time and considerable computational effort, particularly in problems with complex state-action spaces. While RL algorithms have been designed to operate in non-stationary environments [6, 19], there is a delay between when a change is detected and when the learning framework converges to the updated policy. Depending on the problem setting, such delays might be very expensive.

An alternative approach is using algorithms such as Monte Carlo tree search (MCTS) to perform online planning. These approaches

[†] These authors contributed equally to this work

perform their computation at decision time using high-fidelity models of the current environment to find promising action trajectories. These models can be updated as soon as environmental changes are detected, and such changes can be immediately incorporated into decision-making (assuming that the generative model used to build the search tree can be updated quickly). MCTS has been proven to converge to optimal actions given enough computation time [14], but convergence can be slow for domains with large state-action spaces. The slow convergence is a potential issue for problem settings with tight constraints on the time allowed for decision-making; e.g., in emergency response, when an incident occurs, any time used for decision-making increases the overall response time [20].

As we point out, both RL and MCTS have weaknesses when applied individually to complex decision-making problems in non-stationary environments. We argue that a hybrid decision-making approach that integrates RL and online planning can combine their strengths while mitigating their individual weaknesses in non-stationary environments. The intuition is that if the environment has not changed too much between when an optimal policy was learned and when a decision needs to be made, the policy can still provide useful information for decision-making. Similar hybrid approaches have been explored and have achieved state-of-the-art performance in many domains. For example, AlphaZero [23] integrates a policy and value network within a modified MCTS, and has achieved state-of-the-art performance in games such as Go. However, to the best of our knowledge, such a hybrid approach has not been developed specifically for non-stationary environments.

In this paper, we show how hybrid approaches can combine offline learning and online search for decision-making in non-stationary environments. We present a novel hybrid decision-making approach, called *Policy-Augmented Monte Carlo tree search* (PA-MCTS). Our approach is remarkably simple—it combines a policy's action-value estimates with the returns generated by MCTS *without changing either of the two approaches*, i.e., the combination occurs entirely outside the online search tree. Specifically, we make the following contributions:

(1) Conceptually, our core contribution is the idea that offline planning and online search can be combined for decision-making in non-stationary environments.
(2) We present two algorithms to operationalize the proposed idea. The first approach, our core algorithmic contribution, is a novel algorithm that combines a (relatively) stale policy with MCTS after the search, i.e., completely outside the tree. We present several theoretical results for the proposed approach. Second, we show how existing hybrid approaches, e.g., AlphaZero can also be used for decision-making in non-stationary environments.
(3) We validate our approach using four open-source environments from OpenAI Gym and compare it with other state-of-the-art approaches. We show that PA-MCTS results in two distinct advantages—first, given a specific computational budget, our framework converges to significantly better decisions than standard MCTS; and second, online search makes our approach significantly more robust to environmental changes than standard state-of-the-art approaches.

2 BACKGROUND

Motivating Environments: Consider the problem of proactively allocating ambulances in a city in anticipation of accidents, modeled as an MDP [18]. Now, consider that the city experiences unexpected congestion, changing the underlying MDP's transition function. First responders cannot wait to re-train a new policy, and a sub-optimal policy can waste invaluable time to respond to emergency situations. Abstraction of such a problem can be modeled in simpler settings such as toy environments in OpenAI Gym [4]. For example, in the cartpole environment, a rigid pole is attached to a cart via a hinge. The cart can move freely on a bounded, frictionless horizontal track and the agent (i.e., the controller) can apply a force to the cart parallel to the track in either direction. The goal of the controller is to balance the pole as long as possible. Non-stationarity can be induced by varying the gravitational constant or the mass of the cart. Similarly, the Frozen Lake environment involves an agent trying to cross a frozen lake from a start position to a goal position (pre-determined) without falling into any holes (we show a schematic of the environment in Fig. 1). However, the slippery nature of the surface results in the agent not always moving in the intended direction. Here, non-stationarity can be induced by changing the coefficient of friction on the surface, thereby changing how the agent moves. In all settings, we explore how the agent can quickly adapt to the updated environment.

Monte Carlo Tree Search: Our approach is based on Monte Carlo Tree Search (MCTS), an anytime search algorithm that builds a search tree in an incremental and asymmetric manner [5, 8, 14]. MCTS represents environment states by nodes, with the current state being the root node. Actions that result in a transition from one state to another are represented as edges between nodes. The fundamental idea of MCTS is to bias the search toward actions that appear promising. To determine which actions are promising, MCTS uses a model $\mathcal{M}_t$ of the environment at the current decision epoch t to simulate a *rollout* (commonly a randomly sampled possible future trajectory) to the end of the planning horizon. A *tree policy*, such as the commonly used Upper Confidence bound for Trees (UCT) algorithm [14], then uses the averaged returns of these rollouts as estimates $\overline{G}_t(s,a)$ for the value of each action a at each state s, and biases the exploration of future rollouts towards higher estimated returns. MCTS is proven to converge to the optimal action given infinite time [14]. However, the number of iterations required can become impractical as the state-action space of the environment grows.

AlphaZero: One of the most well-known approaches that combine the strengths of MCTS-based planning and deep learning is AlphaZero [24], which has shown state-of-the-art performance in games such as Chess and Go. AlphaZero consists of a neural network to estimate the value of states and MCTS is used as a means for policy improvement. While AlphaZero was not designed as a framework for decision-making in non-stationary environments, we hypothesize that it is particularly suited to such environments. We refer readers to the seminal work by Silver et al. [24] for a comprehensive description of the AlphaZero framework. While we show experimental results against AlphaZero as a baseline, we reiterate that showing that AlphaZero can be used as a decision-making tool for non-stationary environments is one of our contributions.

3 MARKOV DECISION PROCESSES IN NON-STATIONARY SETTINGS

Markov decision processes (MDP) provide a general framework for sequential decision-making under uncertainty. An MDP can be defined by the tuple $(\mathcal{S}, \mathcal{A}, P(s,a), r(s,a))$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a discrete action-space, $P(s' \mid s, a)$ is the probability of reaching state s' when taking action a in state s, and $r(s,a)$ is the scalar reward when action a is taken in state s. The goal of an agent is to learn a policy π that maps states to actions (or, more generally, states to a distribution over actions) that can maximize a specified utility function. Typically, the utility function is simply the expected cumulative reward, also called the *return*, which is defined as $\mathbb{E}_P[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s))]$, where γ denotes the discount factor that weighs immediate rewards more than future rewards. We are interested in settings where the environment with which the agent interacts changes over time. Specifically, we consider an agent that is *trained* on a specific task given a particular setting, but is *executed* in an environment where the underlying MDP's transition function has been modified; e.g., consider the problem of emergency response described in Section 2).

We point out that our interest lies in problem settings where *learning* a new policy immediately is infeasible in practice. In complex real-world tasks, learning a new policy takes time, and decisions must be made as the policy is being updated based on the new environmental conditions. We are interested in understanding how agents can optimize decision-making during such delays, i.e., between when a change is detected in the environment and when the agent learns a new (near-optimal) policy. One way to model such a problem is a non-stationary Markov decision process (NSMDP) [15]. An NSMDP, as defined by Lecarpentier and Rachelson [2019], can be viewed as a stationary MDP where the state space is extended with a temporal dimension. Depending on whether the task is episodic and whether the agent is allowed to explore along the temporal axis, this enhancement can be trivial or difficult [15].

The rate of change in the transition function is typically bounded in an NSMDP by assuming Lipschitz continuity [15], which assumes that the function changes smoothly over time. However, some situations can cause abrupt, large changes to the dynamics of the environment. We propose a new approach to bound the non-stationarity of an MDP by instead considering an upper bound on the total change in the transition function between training and execution; we refer to such a setting as *transition-bounded non-stationary Markov decision processes* (T-NSMDP). Consider the transition probability function $P_t(s' \mid s, a)$, where the subscript t denotes the time step under consideration. Now, consider that the environment undergoes *some* change between time steps 0 and t. We assume that:

$$\forall s, a : \sum_{s' \in S} \left| P_t(s' \mid a, s) - P_0(s' \mid a, s) \right| \leq \eta \tag{1}$$

where $t \in \mathcal{T}$ (i.e., some point in time after the original policy was learned), and $\eta \in \mathbb{R}^+$ is a scalar bound. In other words, the total change in the transition function (cumulated over all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$) between the updated environmental conditions and the original environmental conditions is bounded by η. We also point out that while our problem definition is agnostic to whether the change is continuous or discrete (we could define an analogous setting for continuous-time MDPs), our algorithm only tackles discrete changes for now.

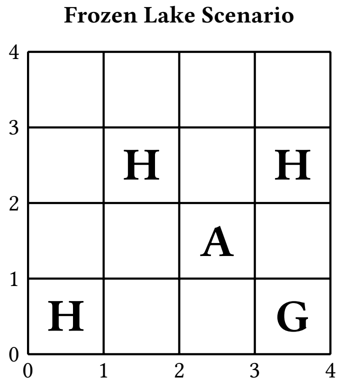

Figure 1: The Frozen Lake environment involves an agent trying to cross a frozen lake from a start position to a goal position (predetermined) without falling into any holes

We demonstrate how Eq. (1) can be applied in practice using the Frozen Lake environment described earlier. At each state, the agent has four actions, i.e., the direction in which it chooses to move. Consider that in the original environment at time 0, the lake is only marginally slippery such that the agent can always go in the direction it intends. Let us consider an arbitrary state $s_{(2,1)}$ which denotes that the agent is in cell $(2,1)$ (refer to the Fig. 1 on the left where each cell is marked with a number). In the original environment, if the agent seeks to go right (takes action a_{right}), then $P_0(s_{(3,1)} \mid a_{\text{right}}, s_{(2,1)}) = 1$, i.e., the agent always moves right if it intends to do so. Now, at time t, consider that lake becomes more slippery, and the agent may not always move in the intended direction due to the slippery nature of the surface. Let us assume that the agent moves in the intended direction with a probability of 1/3 and in each of its perpendicular directions with the same probability (i.e., from cell (2,1), if the agent seeks to go right, it can now end up in cells (3,1), (2,0), or (2,2) with equal probability). In that case, by equation Eq. (1), we can quantify the *amount of change*[1] to be $\mid P_t(s_{(3,1)} \mid a_{\text{right}}, s_{(2,1)}) - P_0(s_{(3,1)} \mid a_{\text{right}}, s_{(2,1)}) \mid + \mid P_t(s_{(2,0)} \mid a_{\text{right}}, s_{(2,1)}) - P_0(s_{(2,0)} \mid a_{\text{right}}, s_{(2,1)}) \mid + \mid P_t(s_{(3,3)} \mid a_{\text{right}}, s_{(2,1)}) - P_0(s_{(3,3)} \mid a_{\text{right}}, s_{(2,1)}) \mid + \mid P_t(s_6 \mid a_{\text{right}}, s_{(2,1)}) - P_0(s_{(2,2)} \mid a_{\text{right}}, s_{(2,1)}) \mid = 1.33$. A similar bound can be computed for the Cartpole environment where the agent is trained in a setting with a certain mass, but the mass of the cart changes in the updated environment.

Hypothesis: Having described an example of how the bounds in a T-NSMDP can be computed, we now present our key hypothesis. Let $s_0 \in \mathcal{S}$ and $a_0 \in \mathcal{A}$ denote the initial state and an action (respectively) at time step 0. Consider the action-value function Q, represented as $Q^{\pi}(s,a) = \mathbb{E}_P\{R(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t R(s_t, a_t = \pi(s_t))\}$, which measures the value of a state s and an action a under policy

[1] we show this for one particular state as an example, the same bound applies across all state-action pairs.

π. Our key hypothesis is that with small changes in the transition function, the Q function under an optimal policy does not change much, i.e., "good" actions remain valuable, and "bad" actions do not suddenly become promising. It is trivial to see that there are exceptions to our hypothesis and its rather simplified explanation; if the Q function stayed the same as before, there would not be any need for the agent to adapt to changes in the environment. However, we show that the change in Q is bounded with respect to the change in P:

Theorem 1. *If $\forall s, a\colon \sum_{s' \in S} |P_t(s' \mid a, s) - P_0(s' \mid a, s)| \leq \eta$, and $\forall s, a\colon |r(s, a)| \leq R$, and the discount factor $\gamma < 1$, then $|Q_0^{\pi_0^*}(s, a) - Q_t^{\pi_t^*}(s, a)| \leq \epsilon\ \forall s, a$, where $\epsilon = \frac{\gamma \cdot \eta \cdot R}{(1-\gamma)^2}$ (The proof is presented in the appendix).*

Theorem 1 considers two environments, one at time 0 and the other at time t; then, under the assumptions that Eq. (1) holds, the rewards are bounded, and $\gamma < 1$, the difference in the optimal Q values (in the old and new environment) is also bounded. While this bound is admittedly loose and can be large in many settings, it serves as the basis for algorithmic decision-making in non-stationary settings if the environment has not changed much. Specifically, if the optimal Q has not changed significantly, then we can leverage it to inform decision-making even when the environment has changed.

Problem Statement: Our goal is to find optimal actions (i.e., actions that maximize the temporally discounted sum of future rewards) at execution time t assuming the following: (1) we are given an optimal action-value function $Q_0^{\pi_0^*}(s, a)$ for time step 0, which might have been learned under different environmental conditions than those at execution time t; (2) any such change in the transition probabilities between when $Q_0^{\pi_0^*}(s, a)$ was learned and time t are bounded by η (Eq. (1)); (3) we are given a black-box simulator that is updated to accurately model the environment at execution time t (we relax this assumption in our experiments); and (4) there is a limited computational budget at execution time that prevents learning an optimal policy.

4 POLICY AUGMENTED MONTE CARO TREE SEARCH

Recall that MCTS is proven to converge to the optimal action given infinite time [14]. The convergence also holds true for an environment that has changed—this consideration is actually meaningless for a purely online approach such as MCTS, assuming that the generative model that is used to build the tree has been updated based on environmental changes. On the other hand, a trained policy has exactly the opposite advantage (and disadvantage): while it can be invoked in constant time during decision-making, it cannot accommodate environmental changes without re-training, which is computationally expensive. We raise the following question: *can the advantages of online search procedures (e.g., MCTS) be combined with those of policy learning (e.g., reinforcement learning) to tackle discrete changes in the environment?*

Our approach presents a natural solution to this question. *Policy-Augmented Monte Carlo Tree Search* (PA-MCTS) addresses this challenge by integrating an online search with Q-values learned on the environment at an earlier decision epoch, even if the environment has changed. Rather than selecting an action based on the highest expected return estimated by the online search, PA-MCTS instead chooses the action that maximizes a convex combination of the previously learned Q-values and the MCTS estimates $\overline{G}$:

$$\underset{a \in \mathcal{A}_s}{\arg\max} \quad \alpha Q_0^{\pi_0^*}(s, a) + (1 - \alpha)\overline{G}_t(s, a) \tag{2}$$

where $Q_0^{\pi_0^*}(s, a)$ is the optimal[2] Q-function previously learned by the decision agent. Note that this combination happens entirely *outside the tree.*[3] The hyper-parameter α, chosen such that $0 \leq \alpha \leq 1$, controls the tradeoff between the learned Q-values and the expected returns generated through MCTS estimates: if $\alpha = 1$, PA-MCTS reduces to the standard Q-learning action selection policy using the previous Q values: $\arg\max_{a \in \mathcal{A}_s} Q_0^{\pi_0^*}(s, a)$. If $\alpha = 0$, it reduces to standard MCTS. We essentially seek to balance the dichotomy between using low-variance but biased estimates through $Q_0^{\pi_0^*}$ (generated using an older environment) and potentially high-variance but unbiased estimates through $\overline{G}_t$ (generated using the current environment). If $\alpha \in (0, 1)$, then both estimates are considered. Note that we refer to the action-value from the Q-function as "estimates" due to the change in the environment. We hypothesize that when the error in Q-values is bounded by ϵ (as described in Theorem 1), $Q_0^{\pi_0^*}$ likely embeds useful information about the updated environment. The PA-MCTS algorithm essentially uses Eq. (2) for decision-making. We present the algorithm in the appendix.

4.1 Theoretical Analysis

We prove three properties of PA-MCTS: (1) the conditions under which PA-MCTS will return the optimal one-step action, (2) the conditions under which PA-MCTS will choose an action with a higher estimated return than either MCTS or selection using Q-values $Q_0^{\pi_0^*}$, and (3) a bound on the total deviation of the expected return from an optimal (updated) policy when following PA-MCTS. We begin by defining some additional notation and detailing our assumptions.

Let $a_t^* := \arg\max_{a \in \mathcal{A}_s} Q^{\pi_t^*}(s, a)$ be the optimal action at an arbitrary time step t. Let $a_t' := \arg\max_{a \in \mathcal{A}_s \setminus \{a_t^*\}} Q_t^{\pi_t^*}(s, a)$ denote the second best action at an arbitrary time step t (we assume that there are no ties in Q-values for any actions at a given state). Let $\psi_t(s) := Q_t^{\pi_t^*}(s, a_t^*) - Q_t^{\pi_t^*}(s, a_t')$; for a state s, $\psi_t(s)$ denotes the difference in Q values when taking actions a_t^* and a_t' at time t and following the optimal policy π_t^* thereafter.[4] Finally, while MCTS is guaranteed to converge to the optimal expected returns for a given state and action given infinite time, actions must be taken after a limited time in practice. Let δ denote the bound on the error of the values estimated by MCTS when it is stopped, i.e., $|Q_t^{\pi_t^*}(s, a) - \overline{G}_t(s, a)|_\infty \leq \delta \quad \forall\ t \in \mathcal{T}$.

[2] In principle, we do not require the optimal Q-function. As we show in the experiments, an approximation also works well in practice.

[3] We also explored approaches to use the learned Q-values inside the tree, but the proposed approach performed at least as well and simplifies (to some degree) the theoretical analysis.

[4] Recall that the optimal policy is indexed with the time step as the environment might change in our setting, i.e., π_t^* denotes the optimal policy given the environmental conditions at time step t.

We first present Theorem 2, which describes the conditions with respect to ϵ, δ, and $\psi_t(s)$ under which PA-MCTS returns the optimal one-step action.

Theorem 2. *If $\alpha\epsilon + (1-\alpha)\delta \leq \frac{\psi_t(s)}{2}$, PA-MCTS is guaranteed to select the optimal action at time step t in state s. (The proof is presented in the appendix).*

Theorem 2 essentially shows that PA-MCTS is guaranteed to choose the optimal action if the sum of α-weighted errors is low enough that the decision agent can differentiate between the best and the next best action. While $\psi_t(s)$ is not observable at decision time, we can find the relationship between $\psi_t(s)$ and $\psi_0(s)$:

Corollary 2.1. $\psi_t(s) \leq \psi_0(s) + 2\epsilon$

Using Corollary 2.1, we substitute $\psi_t(s)$ in terms of $\psi_0(s)$ (which can be computed at decision time using known $Q_0^{\pi_0^*}$ values) in Theorem 2. Then, we can determine when PA-MCTS will select the optimal action using information that can be computed at decision time:

Corollary 2.2. *If $\alpha\epsilon + (1-\alpha)\delta \leq \frac{\psi_0(s)}{2} + \epsilon$, PA-MCTS is guaranteed to select the optimal action at decision epoch t.*

Using $0 \leq \alpha \leq 1$, Corollary 2.2 can be rearranged to solve for α:

Corollary 2.3. *PA-MCTS will choose the optimal one-step action if*

$$\begin{cases} \frac{-\delta}{\epsilon-\delta} \leq \alpha \leq \frac{\psi_0(s)}{2(\epsilon-\delta)} + 1 & \text{if } \epsilon > \delta \\ \frac{-\delta}{\epsilon-\delta} \geq \alpha \geq \frac{\psi_0(s)}{2(\epsilon-\delta)} + 1 & \text{if } \epsilon < \delta. \end{cases}$$

Therefore, given ϵ and δ, a decision agent can determine what values of the hyperparameter α (if any) would guarantee that PA-MCTS chooses the optimal action at decision time.

Theorem 2 guarantees the conditions under which PA-MCTS will choose the optimal action. We now seek to answer a more practical question: when does PA-MCTS choose a one-step action with a higher $Q_t^{\pi_t^*}$ value than MCTS or the learned policy in isolation? First, we list conditions under which PA-MCTS chooses a better action than pure MCTS:

Proposition 1. *If PA-MCTS and MCTS choose different actions, PA-MCTS's chosen action will have a higher $Q_t^{\pi_t^*}$ value than MCTS if $2\epsilon \leq \zeta_t^m$, where*[5] *$\zeta_t^m := (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_m) + \overline{G}_t(s,a_m))$, $\tilde{a} := \arg\max_{a\in\mathcal{A}_s} \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a)$, and $a_m := \arg\max_{a\in\mathcal{A}_s} \overline{G}_t(s,a)$. (The proof is presented in the appendix).*

Since ζ_t^m is constructed from terms that can be computed at decision time, Proposition 1 can be used by the decision agent to determine if it should use PA-MCTS or MCTS when they choose different actions at decision time. A similar statement can be made regarding action selection using $Q_0^{\pi_0^*}$.

Proposition 2. *If the action chosen by PA-MCTS is different than $\arg\max_a Q_0^{\pi_0^*}(s,a)$ at state s, then PA-MCTS's chosen action will have a higher $Q_t^{\pi_t^*}$ value than selection through $Q_0^{\pi_0^*}$-values if $2\delta \leq \zeta_t^q$ where*[6] *$\zeta_t^q := (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_g) + \overline{G}_t(s,a_g))$, $\tilde{a} := \arg\max_{a\in\mathcal{A}_s} \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a)$, and $a_g := \arg\max_{a\in\mathcal{A}_s} Q_0^{\pi_0^*}(s,a)$. (The proof is presented in the appendix).*

Using Propositions 1 and 2, we can determine the conditions under which PA-MCTS chooses an action that is better than either of its constituent policies. The above propositions consider only one decision epoch. However, typically, training a new policy takes time, which might consist of several decision epochs. Therefore, we compute the total error in the expected return when following PA-MCTS compared to an optimal (updated) policy.

Theorem 3. *When PA-MCTS is used for sequential decision making, the maximum difference between the return from an optimal policy and the return from following PA-MCTS is at most $\frac{2(\alpha\epsilon-\alpha\delta+\delta)}{1-\gamma}$. (The proof is presented in the appendix).*

4.2 Experiment Setting

Environments: We use the following four open-source environments from OpenAI Gym [4] to validate our approach:

1. **Cart Pole**: In this classic control problem, a pole is attached by an un-actuated joint to a cart, which moves on a frictionless track. A pendulum is placed upright on the cart. The agent's goal is to balance the pole by applying forces either to the left or the right direction on the cart. For the cart pole environment, we set our original environment with gravitational constant g as $9.8\,\mathrm{m/s^2}$ and the mass of the pole as $0.1\,\mathrm{kg}$. Both parameters are then varied to induce non-stationarity in the state-action transition function.
2. **Frozen Lake**: In the frozen lake problem, an agent must traverse from a starting position to the goal position by avoiding holes. The agent moves on a slippery surface (we describe the setting in section 3). We introduce non-stationary by making the surface more slippery; in our experiments, we refer to a problem instance of the frozen lake environment by the vector p_1, p_2, p_3, where p_1 denotes the probability with which the agent moves in the intended direction, and p_2 and p_3 denote the probabilities of moving in directions perpendicular to the intended direction. For the frozen lake environment, we treat the "original environment" as one with perfect agent movement, i.e., deterministic movement.
3. **Cliff Walking**: In the cliff walking environment, an agent must find a path from the start position to the goal position from start to goal while avoiding falling off a cliff. While the setting is similar to the frozen lake environment, here, the agent incurs a penalty for each step that it takes, thereby presenting the explicit dichotomy of choosing between an optimal path (with respect to discounted rewards) and a safe path. Similar to the frozen lake environment, the original environment involves deterministic agent movement.
4. **Lunar Lander**: This environment presents a rocket trajectory optimization problem. The agent must control a rocket and land safely on a known landing pad. While landing

[5] the superscript m in ζ_t^m represents the bounds with respect to MCTS and is not an index.

[6] Again, the superscript q only denotes bounds with respect to the Q-values and is not an index.

outside the pad is possible, the agent incurs a penalty. The agent's action at each time step involves firing one of the three engines (main, right, and left) at full throttle or not doing anything. Here, the original environment does not involve any wind. We induce wind to simulate non-stationarity; the agent, therefore, must adjust its actions according to the power of the wind.

Baseline approaches and implementation: We implement AlphaZero, a reinforcement learning (RL) agent using a Double Deep Q-Network (DDQN) [26], and standard MCTS as our baseline approaches. We reiterate that we do not treat AlphaZero as a *baseline* per se since our broader conceptual idea of using hybrid approaches to combat non-stationarity subsumes it. Nonetheless, we compare PA-MCTS with AlphaZero, and show that our proposed approach outperforms AlphaZero. We implement MCTS and PA-MCTS using UCT [14] as the tree policy. Our implementation is in Python [27] and available at `https://github.com/scope-lab-vu/PAMCTS`

Hardware: We run experiments on the Chameleon testbed [11] on 4 Linux systems with 32-96 logical processors and 528 GB RAM. We train AlphaZero using the Google Cloud Platform on a Linux system with 128 logical processors and 528 GB RAM.

Hyper-Parameters: We implement the neural networks in TensorFlow [1]. For the RL agent, the Double DQN has 3 hidden layers with ReLU activation, and the output layer has a linear activation. We use a Boltzmann Q Policy and train the network with the Adam Optimizer[12] with a learning rate of 10^{-3} for 300,000 steps while minimizing the mean absolute error of the network. For AlphaZero, we use 5 hidden layers with ReLU activation and appropriate output layers (e.g., softmax for the policy and dense layer for the value function). For both PA-MCTS and AlphaZero, we set the discount factor as 0.99, maximum tree depth as 500 (though this is rarely reached), and the exploration parameter $c = 50$ (after tuning on holdout data). A more detailed description of the hyper-parameters is presented in the appendix.

4.3 Results

Comprehensive results for all environments and all settings we use to induce non-stationarity are presented in Table 1. Below, we dive deeper into the cart pole environment to analyze the results in depth and evaluate the nuances of PA-MCTS. Similar analysis for the other environments can be found in the appendix.

Stationary setting: We train both the AlphaZero network and the DDQN with $g = 9.8\,\text{m/s}^2$ and $m = 1.0\,\text{kg}$ and evaluate all the approaches in the same environmental settings. We present PA-MCTS and DDQN results in Fig. 2 (different color lines represent different α used by PA-MCTS) and the comparison with AlphaZero in Fig. 5 (the left-most sub-graph shows the stationary setting). We observe that under stationary conditions, both AlphaZero and the DDQN (which is a special case of PA-MCTS with $\alpha = 1.0$) outperform standard MCTS and PA-MCTS with $\alpha \in \{0.25, 0.5, 0.75\}$. This result is expected, as offline learning is able to leverage potentially thousands of training episodes to learn an optimal policy. However, we do note that PA-MCTS (for $0 < \alpha < 1$) is able to leverage the information stored in the policy to converge significantly faster than standard MCTS. We compute each result by averaging across 30 samples.

Non-stationary setting: To add non-stationarity to the cart pole environment, we modify g from $9.8\,\text{m/s}^2$ to $20, 50$, and $500\,\text{m/s}^2$ (we present results in Fig. 2; additional experiments which modify the mass of the cart are shown in the appendix). We have several important findings. First, we observe as the change in the environment increases, the performance of the DDQN policy in isolation (trained on the original environment) degrades as hypothesized—in fact, it achieves almost 0 utility for $g \in \{50, 500\}$. Second, we again observe that PA-MCTS converges significantly faster than standard MCTS for all values of g (with appropriate α); it actually achieves optimal performance except when $g = 500\,\text{m/s}^2$. The reader might wonder about how to choose the optimal α and the performance of AlphaZero, which we show below.

Choosing the optimal α: During execution, the agent must choose actions relatively quickly. Therefore, even when updated generative models of the system are available, it is unlikely that the agent would be able to afford the time to compute extensive simulations across a range of α values to select the optimal one. We provide an empirical approach to alleviate this concern. First, we run simulations for various iteration counts of MCTS in a wide array of environmental conditions. For each setting, we show the optimal α in Fig. 3. For example, the top rows of the heatmap show that when the environment has not changed much, it is better for the agent to rely more on the trained policy (i.e., α is typically above 0.5). On the other hand, consider the bottom-right cell in the heatmap, which shows that if the environment changes a lot, the agent must rely more on the online search, as expected. We point out another interesting finding—it turns out that PA-MCTS, with limited MCTS iterations, is able to approximate the optimal choice of α fairly well, i.e., observe that the column with iterations=25 (or even with iterations=10) is almost similar to the column with iterations=100. We use this finding to guide the selection of α. We hypothesize that the agent can use the updated generative models to perform a hyper-parameter sweep of α with a small number of MCTS iterations; the sweep can be run in parallel for each α and in our setting takes only 80 seconds (much smaller than re-training the policy, as we show below). Then, for execution, the agent can choose the α value that provides the highest expected discounted return.

Comparison with AlphaZero: We use the proposed selection strategy to tune α and compare our results with AlphaZero (presented in Fig. 5). First, we observe that except for a drastic environmental change ($g = 500\,\text{m/s}^2$), both AlphaZero and PA-MCTS (with α-selection) converge to the optimal policy. However, we see that PA-MCTS (with α-selection) converges significantly faster than AlphaZero; indeed, when $g = 50\,\text{m/s}^2$, PA-MCTS takes about a third of the number iterations that AlphaZero takes to converge. In real-world settings, faster convergence is potentially invaluable.

Runtime Comparison: We also explore whether re-training the policies by using the updated environmental model is computationally cheaper than executing PA-MCTS. Table 2 shows the calculated average PA-MCTS execution time for different iteration counts (with α-selection), and the retraining time for the DDQN network and the Alphazero network. For a fair comparison, we do not re-train DDQN and AlphaZero from scratch and update the existing networks, which is considerably faster than training from scratch,

Environment	Setting	DDQN	MCTS	AlphaZero	PA-MCTS
Cartpole (varying g)	g = 9.8	**2500.0 ± 0.0**	846.456±43.228	2403.261±35.946	**2500.0±0.0**
	g = 20	**2500.0±0.0**	918.022±46.554	2278.90±52.9	**2500.0±0.0**
	g = 50	22.061±0.729	778.511±44.90	1920.261±78.547	**2500.0±0.0**
	g = 500	7.083±0.126	111.578±17.705	626.178±80.091	**954.656±90.150**
Cartpole (varying m)	m = 0.1	**2500.0 ± 0.0**	846.456±43.228	2403.261±35.946	**2500.0±0.0**
	m = 1.0	**500.0±0.0**	892.956±42.674	1340.006±86.713	**2500.0±0.0**
	m = 1.2	1249.661±90.587	802.905±30.748	**2093.411±65.519**	1715.783±68.101
	m = 1.3	475.35±67.96	904.078±38.404	**1346.433±86.238**	904.078±38.404
	m = 1.5	**2.0±0.0**	**2.0±0.0**	**2.0±0.0**	**2.0±0.0**
Frozen Lake	[1.000, 1.000, 1.000]	**1.0 ± 0.0**	**1.0 ± 0.0**	**1.0 ± 0.0**	**1.0±0.0**
	[0.833, 0.083, 0.083]	**0.830 ± 0.012**	0.806 ± 0.012	0.809±0.013	**0.830±0.012**
	[0.633, 0.183, 0.183]	0.522 ± 0.016	0.56 ± 0.017	0.523±0.017	**0.587±0.016**
	[0.433, 0.283, 0.283]	0.26 ± 0.014	0.764 ± 0.013	0.235±0.014	**0.796±0.013**
	[0.333, 0.333, 0.333]	0.12 ± 0.01	0.866 ± 0.01	0.114±0.011	**0.936±0.009**
Cliff Walking	0.0	**0.88 ± 0.00**	0.869 ± 0.001	0.874 ± 0.00	**0.88 ± 0.00**
	0.1	0.291 ± 0.016	**0.7 ± 0.011**	0.27 ± 0.014	**0.7 ± 0.011**
	0.2	0.102 ± 0.011	**0.459 ± 0.015**	0.081 ± 0.009	**0.459 ± 0.015**
	0.3	0.034 ± 0.006	**0.173 ± 0.012**	0.012 ± 0.004	**0.173 ± 0.012**
Lunar Lander	0.0	**256.034±2.939**	-76.867±3.792	-286.325 ± 16.759	**256.034 ± 2.939**
	10.0	**241.676±4.956**	-79.745±4.697	-250.502 ± 14.967	**241.676 ± 4.955**
	15.0	**192.877±9.25**	-94.906±6.440	-255.377 ± 13.234	**192.877 ± 9.25**
	20.0	**113.431±14.113**	-119.177±8.654	-253.276 ± 11.086	**113.43 ± 14.113**

Table 1: Results for all four environments with varying levels of non-stationarity. For each environment, the degree of *change* increases from top to bottom. We observe that PA-MCTS comprehensively outperforms the baseline approaches, including AlphaZero.

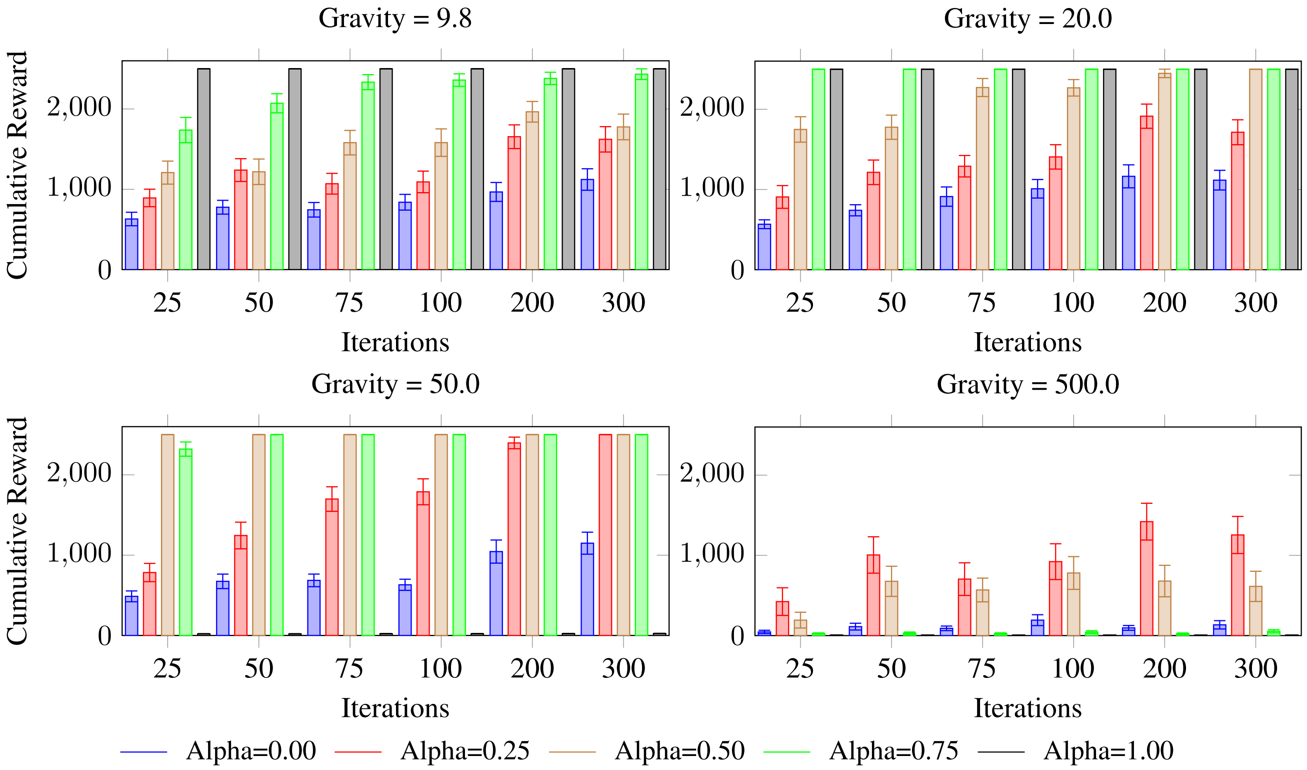

Figure 2: We show the cumulative reward obtained by PA-MCTS, standard MCTS ($\alpha = 0$), and DQN ($\alpha = 1$) for different environmental changes. Note that as the environment changes, DQN achieves close to 0 rewards. PA-MCTS outperforms both baseline approaches.

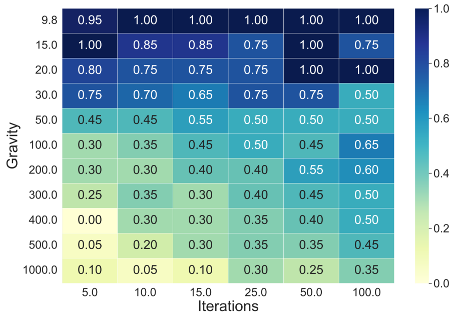

Figure 3: We show the optimal α for varying MCTS iterations and environmental conditions.

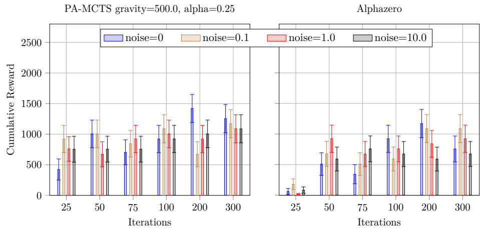

Figure 4: The performance of PA-MCTS and AlphaZero under varying levels of noise with $g = 500\,\mathrm{m/s^2}$. PA-MCTS outperforms AlphaZero in most settings.

given the updated environment. We show that it is significantly faster to run PA-MCTS than re-training DDQN or AlphaZero.

Noisy settings: In practice, it is difficult to update the environmental model to reflect the exact changes in the non-stationary environment. While the generative models can be updated online using updated sensor measurements, such measurements can be noisy, thereby resulting in a noisy model. To simulate such conditions, we repeat the entire pipeline by adding noise in the updated environmental model, e.g., the agent perceives that $g = g_{\text{true}} \pm \mathcal{N}(0, \sigma)$, (where $\mathcal{N}$ denotes a Gaussian distribution) for different values of σ. Note that during execution, the agent must operate under g_{true}, thereby making this situation particularly challenging. We show extensive results for this setting in the appendix but present the key results in Fig. 4. We see that even under noisy settings, PA-MCTS is able to leverage the old policy and outperforms AlphaZero (it also outperforms standard MCTS and DDQN by a large margin, which we show in the appendix).

5 RELATED WORK

Sequential decision-making in non-stationary environments has been studied from several perspectives. Satia and Lave Jr [21] and White III and Eldeib [28] consider transition matrices constrained within a pre-specified polytope. One of the earliest works in this domain, by Satia and Lave [21], constrains the state transition function among a set of predefined distributions. White and Eldeib [28] consider imprecise transition probabilities by representing the transition function by a finite set of linear inequalities. In both formulations, the transition matrix is constrained within a pre-specified polytope. However, as pointed out by Iyengar [10], they do not discuss how such polytopes can be constructed. Iyengar [10] introduces the idea of robust MDPs using the concept of uncertain priors, where the transition function can change within a set of functions due to uncertainty. Later, Lecarpentier and Rachelson [15] introduce Non-Stationary Markov Decision Processes (NSMDP), which extend robust MDPs by allowing uncertainty in the reward model in addition to the transition function and use a stronger Lipschitz formulation for the set of possible transition and reward functions. Choi et al. [7] introduce hidden-mode MDPs, which consider a formal model for representing changes in the environment, confined with a set of modes. A broader notion of non-stationarity is introduced by Lecarpentier and Rachelson [15], who consider that both the reward function and the transition function can change over time and that the rate of change is bounded through Lipschitz continuity. Our formulation is inspired by that of Lecarpentier and Rachelson [15]. The key difference in our formulation is the representation of that change.

We also point out that decision-making by an agent that is trained on one task (or environmental conditions) and subsequently provided with another task has also been explored in the domains of transfer learning [16] and lifelong reinforcement learning [25]. However, we specifically look at settings where "learning" a new policy is not feasible, or even when it is feasible, decisions must be taken while the updated policy is being learned. Finally, we recognize that approaches combining model-based online search with learning methods have been explored—for example, AlphaZero integrates MCTS with a policy iteration framework [24], while the Search with Amortized Value Estimates (SAVE) algorithm combines model-free Q-learning with MCTS [9]. To the best of our knowledge, our work is the first that develop such an approach for non-stationary environments.

6 CONCLUSION

We explore sequential decision-making in non-stationary environments, where the decision-maker faces the dilemma of choosing between accurate but obsolete state-action values from a learning-based approach and unbiased but high-variance estimates from an online search. We present a novel approach, *Policy Augmented*

Method	Iterations	Execution Time (in seconds)
PA-MCTS	5	2.57
PA-MCTS	10	4.35
PA-MCTS	15	3.51
PA-MCTS	25	9.60
PA-MCTS	50	25.50
PA-MCTS	100	48.73
PA-MCTS	200	100.0
PA-MCTS	300	248.8
Re-Train DQN	200,000 steps	430.0
Re-Train AlphaZero	NA	31,800

Table 2: We show the computation time for running PA-MCTS (with α-selection and re-training DQN and AlphaZero (by updating the existing networks instead of training from scratch for a fair comparison). Even under these settings, we show that PA-MCTS is significantly faster.

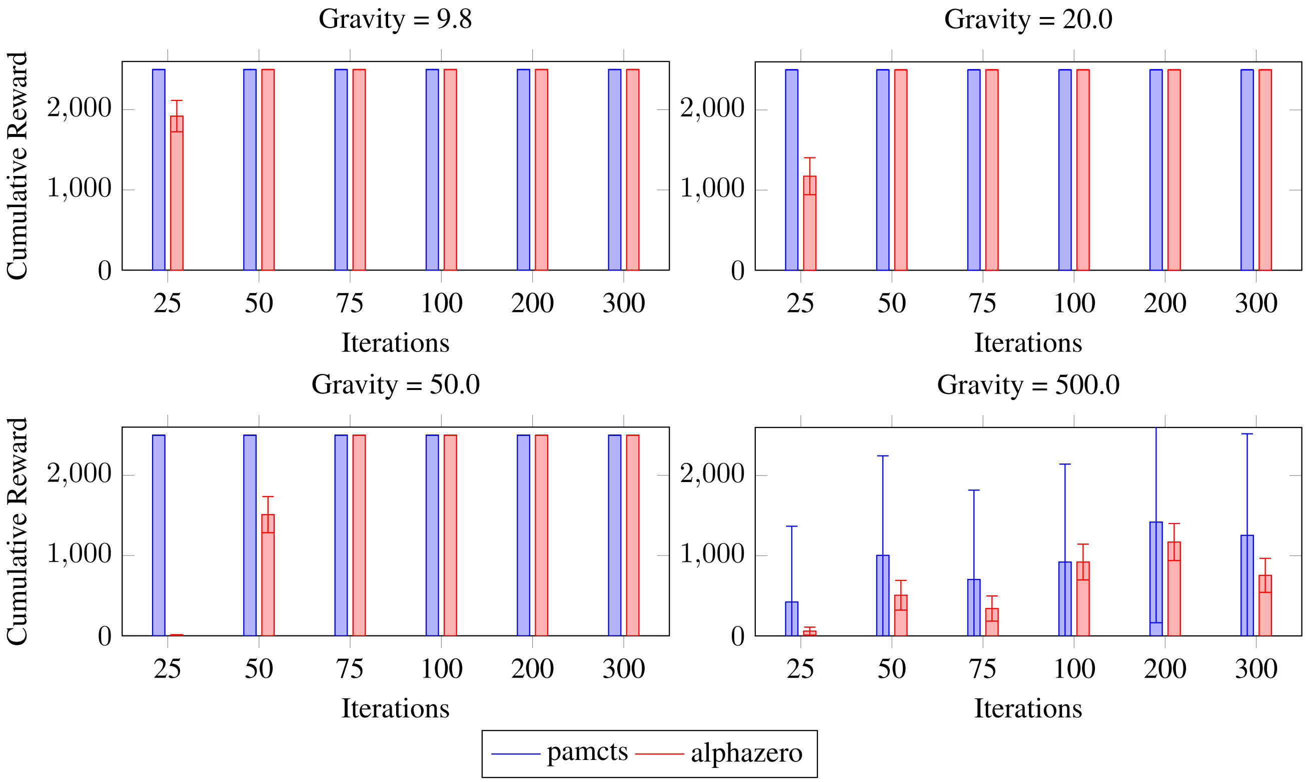

Figure 5: We compare PA-MCTS with α-selection with AlphaZero. While both approaches find the optimal policy, PA-MCTS converges significantly faster.

MCTS (PA-MCTS), that combines the strengths of reinforcement learning and online planning in non-stationary environments. We present theoretical results characterizing the performance of MCTS, demonstrating that there is a range of values that will work well in practice. We also present bounds on the error accrued by following PA-MCTS as a policy for sequential decision-making. Through extensive experiments on open-source environments, we show that PA-MCTS outperforms the baselines in terms of performance and convergence time.

ACKNOWLEDGMENTS

This material is based upon work sponsored by the National Science Foundation (NSF) under Grant CNS-2238815 and by the Defense Advanced Research Projects Agency (DARPA). We also acknowledge Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, or the DARPA. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Turgay Ayer, Oguzhan Alagoz, and Natasha K Stout. 2012. OR Forum—A POMDP approach to personalize mammography screening decisions. *Operations Research* 60, 5 (2012), 1019–1034.

[3] Maxime Bouton, Alireza Nakhaei, Kikuo Fujimura, and Mykel J Kochenderfer. 2019. Safe reinforcement learning with scene decomposition for navigating complex urban environments. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1469–1476.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* (2016).

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43.

[6] Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. 2019. Non-stationary reinforcement learning: The blessing of (more) optimism. *Available at SSRN 3397818* (2019).

[7] Samuel PM Choi, Dit-Yan Yeung, and Nevin L Zhang. 2000. Hidden-mode Markov decision processes for nonstationary sequential decision making. In *Sequence Learning*. Springer, 264–287.

[8] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games (CG)*. Springer, 72–83.

[9] Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W Battaglia. 2019. Combining Q-Learning and Search with Amortized Value Estimates. In *International Conference on Learning Representations*.

[10] Garud N Iyengar. 2005. Robust dynamic programming. *Mathematics of Operations Research* 30, 2 (2005), 257–280.

[11] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *USENIX Annual Technical Conference*. USENIX Association.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. 2022. *Algorithms for decision making*. MIT Press.

[14] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning (ECML)*. Springer, 282–293.

[15] Erwan Lecarpentier and Emmanuel Rachelson. 2019. Non-stationary Markov decision processes, a worst-case approach using model-based reinforcement learning. *Advances in Neural Information Processing Systems* 32 (2019), 7216–7225.

[16] Leandro L Minku. 2019. Transfer learning in non-stationary environments. In *Learning from Data Streams in Evolving Environments*. Springer, 13–37.

[17] Ayan Mukhopadhyay, Geoffrey Pettet, Chinmaya Samal, Abhishek Dubey, and Yevgeniy Vorobeychik. 2019. An online decision-theoretic pipeline for responder dispatch. In *ACM/IEEE International Conference on Cyber-Physical Systems*. 185–196.

[18] Ayan Mukhopadhyay, Zilin Wang, and Yevgeniy Vorobeychik. 2018. A decision theoretic framework for emergency responder dispatch. In *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 588–596.

[19] Ronald Ortner, Pratik Gajane, and Peter Auer. 2020. Variational regret bounds for reinforcement learning. In *35th Uncertainty in Artificial Intelligence Conference*, Vol. 115. 81–90.

[20] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J. Kochenderfer, and Abhishek Dubey. 2021. Hierarchical Planning for Dynamic Resource Allocation in Smart and Connected Communities. *ACM Transactions on Cyber-Physical Systems* (2021). arXiv:2107.01292 [cs.MA]

[21] Jay K Satia and Roy E Lave Jr. 1973. Markovian decision processes with uncertain transition probabilities. *Operations Research* 21, 3 (1973), 728–740.

[22] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (Jan. 2016), 484–489.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144. https://doi.org/10.1126/science.aar6404 arXiv:https://www.science.org/doi/pdf/10.1126/science.aar6404

[24] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[25] Daniel L Silver, Qiang Yang, and Lianghao Li. 2013. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI Spring Symposium Series*. 49–55.

[26] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-learning. In *30th AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 30. 2094–2100.

[27] Guido Van Rossum and Fred L Drake Jr. 1995. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

[28] Chelsea C White III and Hany K Eldeib. 1994. Markov decision processes with imprecise transition probabilities. *Operations Research* 42, 4 (1994), 739–749.

[29] Kyle Hollins Wray. 2019. *Abstractions in reasoning for long-term autonomy*. Ph.D. Dissertation. University of Massachusetts Libraries.

Appendices

A EXTENDED THEORETICAL ANALYSIS

This appendix contains detailed proofs for each of the theorems, propositions, and corollaries described in Section 4.2 of the main text.

Theorem 1. If $\forall s, a\colon \sum_{s' \in S} |P_t(s' \mid a, s) - P_0(s' \mid a, s)| \leq \eta, \forall s, a\colon |r(s, a)| \leq R$, and the discount factor $\gamma < 1$, then $|Q_0^{\pi_0^*}(s, a) - Q_t^{\pi_t^*}(s, a)| \leq \epsilon \ \forall s, a$, where $\epsilon = \frac{\gamma \cdot \eta \cdot R}{(1-\gamma)^2}$.

Proof. Our goal is to find an upper bound ϵ on the difference between the action values before a change in the environment's transition probabilities and after, i.e.:

$$\forall s, a : \left|Q'(s, a) - Q(s, a)\right| \leq \epsilon$$

where Q compactly represents the action-value function in the original environment using an optimal policy (i.e., $Q := Q_0^{\pi_0^*}$) and Q' represents the action-value function in an updated environment at time step $t \in \mathcal{T}$ (i.e., $Q' := Q_t^{\pi_t^*}$).

Let $V_n(s)$ be the value of state s if we terminate the environment after n steps (following the optimal policy until step n). Similarly, let $Q_n(s, a)$ be the $Q(s, a)$ if we terminate the environment after n steps.

For $n = 1$, $|V_n'(s) - V_n(s)| = 0$ and $|Q_n'(s, a) - Q_n(s, a)| = 0$, since we can assume immediate rewards are the same.

Let $\kappa_{n-1} = \max_s |V_{n-1}'(s) - V_{n-1}(s)|$ (i.e., κ_{n-1} is the upper bound in the difference of the value).

For an arbitrary n, we can bound the action-value function Q as follows

$$\begin{aligned}
\left|Q_n'(s, a) - Q_n(s, a)\right| &= \left|\left[r(s, a) + \gamma \sum_{s'} T'(s, a, s') \cdot V_{n-1}'(s')\right] - \left[r(s, a) + \gamma \sum_{s'} T(s, a, s') \cdot V_{n-1}(s')\right]\right| \\
&= \gamma \left|\sum_{s'} T'(s, a, s') \cdot V_{n-1}'(s') - T(s, a, s') \cdot V_{n-1}(s')\right| \\
&= \gamma \left|\sum_{s'} T'(s, a, s') \cdot \left(V_{n-1}'(s') - V_{n-1}(s')\right) + T'(s, a, s') \cdot V_{n-1}(s') - T(s, a, s') \cdot V_{n-1}(s')\right| \\
&= \gamma \left|\sum_{s'} T'(s, a, s') \cdot \left(V_{n-1}'(s') - V_{n-1}(s')\right) + V_{n-1}(s') \cdot \left(T'(s, a, s') - T(s, a, s')\right)\right| \\
&= \gamma \left|\sum_{s'} T'(s, a, s') \cdot \left(V_{n-1}'(s') - V_{n-1}(s')\right) + \sum_{s'} V_{n-1}(s') \cdot \left(T'(s, a, s') - T(s, a, s')\right)\right| \\
&\leq \gamma \left|\sum_{s'} T'(s, a, s') \cdot \left(V_{n-1}'(s') - V_{n-1}(s')\right)\right| + \gamma \left|\sum_{s'} V_{n-1}(s') \cdot \left(T'(s, a, s') - T(s, a, s')\right)\right| \\
&\leq \gamma \sum_{s'} \left|T'(s, a, s') \cdot \left(V_{n-1}'(s') - V_{n-1}(s')\right)\right| + \gamma \sum_{s'} \left|V_{n-1}(s') \cdot \left(T'(s, a, s') - T(s, a, s')\right)\right|
\end{aligned}$$

$$\begin{aligned}
&= \gamma \sum_{s'} \left|T'(s, a, s')\right| \cdot \left|V_{n-1}'(s') - V_{n-1}(s')\right| + \gamma \sum_{s'} \left|V_{n-1}(s')\right| \cdot \left|T'(s, a, s') - T(s, a, s')\right| \\
&\leq \gamma \sum_{s'} T'(s, a, s') \cdot \kappa_{n-1} + \gamma \sum_{s'} \frac{R}{1-\gamma} \cdot \left|T'(s, a, s') - T(s, a, s')\right| \\
&= \gamma \cdot \kappa_{n-1} \sum_{s'} T'(s, a, s') + \gamma \frac{R}{1-\gamma} \sum_{s'} \left|T'(s, a, s') - T(s, a, s')\right| \\
&\leq \gamma \cdot \kappa_{n-1} \cdot 1 + \gamma \frac{R}{1-\gamma} \epsilon \\
&= \gamma \left[\epsilon \frac{R}{1-\gamma} + \kappa_{n-1}\right]
\end{aligned}$$

We can also bound the value function V. First, assume that $V_n'(s) \geq V_n(s)$, then for an arbitrary n, we have

$$\begin{aligned}
\left|V_n'(s) - V_n(s)\right| &= V_n'(s) - V_n(s) \\
&= \max_a Q_n'(s,a) - \max_a Q_n(s,a) \\
&= \max_a \left[Q_n'(s,a) - Q_n(s,a) + Q_n(s,a)\right] - \max_a Q_n(s,a) \\
&\leq \max_a \left[\gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right] + Q_n(s,a)\right] - \max_a Q_n(s,a) \\
&= \gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right] + \max_a Q_n(s,a) - \max_a Q_n(s,a) \\
&\leq \gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right]
\end{aligned}$$

Second, assume that $V_n'(s) < V_n(s)$, then for an arbitrary n, we have

$$\begin{aligned}
\left|V_n'(s) - V_n(s)\right| &= V_n(s) - V_n'(s) \\
&= \max_a Q_n(s,a) - \max_a Q_n'(s,a) \\
&= \max_a Q_n(s,a) - \max_a \left[Q_n'(s,a) - Q_n(s,a) + Q_n(s,a)\right] \\
&\leq \max_a Q_n(s,a) - \max_a \left[-\gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right] + Q_n(s,a)\right] \\
&= \max_a Q_n(s,a) + \gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right] - \max_a Q_n(s,a) \\
&\leq \gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right]
\end{aligned}$$

Thus, $|V_n'(s) - V_n(s)| \leq \gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right]$ in both cases.

Therefore,

$$\kappa_n \leq \gamma\left[\epsilon\frac{R}{1-\gamma} + \kappa_{n-1}\right]$$

$$\kappa_n \leq \underbrace{\gamma\left(\epsilon\frac{R}{1-\gamma} + \gamma\left(\epsilon\frac{R}{1-\gamma} + \gamma\left(\epsilon\frac{R}{1-\gamma} + \dots\right)\right)\right)}_{n \text{ terms, plus } \kappa_0=0}$$

Hence,

$$\begin{aligned}
\kappa_n &\leq \sum_{i=1}^{n} \gamma^i \epsilon\frac{R}{1-\gamma} \\
&= \gamma \cdot \epsilon\frac{R}{1-\gamma}\sum_{i=0}^{n-1}\gamma^i \\
&\leq \gamma \cdot \epsilon\frac{R}{1-\gamma}\sum_{i=0}^{\infty}\gamma^i \\
&= \gamma \cdot \epsilon\frac{R}{1-\gamma} \cdot \frac{1}{1-\gamma} \\
&= \frac{\gamma \cdot \epsilon \cdot R}{(1-\gamma)^2}
\end{aligned}$$

□

Theorem 2. *If $\alpha\epsilon + (1-\alpha)\delta \leq \frac{\psi_t}{2}$, PA-MCTS is guaranteed to select the optimal action at time step t.*

Proof. PA-MCTS is guaranteed to select the optimal action when the following inequality holds:

$$\alpha Q_0^{\pi_0^*}(s, a_t') + (1-\alpha)\overline{G}_t(s, a_t') \leq \alpha Q_0^{\pi_0^*}(s, a_t^*) + (1-\alpha)\overline{G}_t(s, a_t^*) \tag{3}$$

Recall that δ denotes the bound on the error of the values estimated by MCTS when it is stopped, i.e., $|Q_t^{\pi_t^*}(s,a) - \overline{G}_t(s,a)|_\infty \leq \delta \quad \forall\ t \in \mathcal{T}$. Using this definition and Theorem 1, $Q_t^{\pi_t^*}(s,a)$ can be bounded with respect to the estimates $Q_0^{\pi_0^*}(s,a)$ and $\overline{G}_t(s,a)$:

$$\begin{aligned} Q_0^{\pi_0^*}(s,a) - \epsilon \leq Q_t^{\pi_t^*}(s,a) \leq Q_0^{\pi_0^*}(s,a) + \epsilon \\ \overline{G}_t(s,a) - \delta \leq Q_t^{\pi_t^*}(s,a) \leq \overline{G}_t(s,a) + \delta \end{aligned} \tag{4}$$

By substituting $Q_t^{\pi_t^*}(s,a)$ for $Q_0^{\pi_0^*}$ and $\overline{G}_t$ in Eq. (3) by using inequalities in Eq. (4) and rearranging, we can find the conditions under which PA-MCTS chooses the optimal action (recall that $\psi_t(s) := Q_t^{\pi_t^*}(s,a_t^*) - Q_t^{\pi_t^*}(s,a_t'))$:

$$\begin{aligned} \alpha Q_0^{\pi_0^*}(s,a_t') + (1-\alpha)\overline{G}_t(s,a_t') &\leq \alpha Q_0^{\pi_0^*}(s,a_t^*) + (1-\alpha)\overline{G}_t(s,a_t^*) \\ \Rightarrow \alpha(Q_t^{\pi_t^*}(s,a_t') + \epsilon) + (1-\alpha)(Q_t^{\pi_t^*}(s,a_t') + \delta) &\leq \alpha(Q_t^{\pi_t^*}(s,a_t^*) - \epsilon) + (1-\alpha)(Q_t^{\pi_t^*}(s,a_t^*) - \delta) \\ \Rightarrow 2(\alpha\epsilon + (1-\alpha)\delta) &\leq Q_t^{\pi_t^*}(s,a_t^*) - Q_t^{\pi_t^*}(s,a_t') \\ \Rightarrow 2(\alpha\epsilon + (1-\alpha)\delta) &\leq \psi_t \\ \Rightarrow \alpha\epsilon + (1-\alpha)\delta &\leq \frac{\psi_t}{2} \end{aligned}$$

□

Corollary 2.1. $\psi_t \leq \psi_0 + 2\epsilon$

Proof. It is important to note that the optimal and second best actions with respect to $Q_0^{\pi_0^*}$ and $Q_t^{\pi_t^*}$ may not be the same, i.e., it is possible that $a_0^* \neq a_t^*$ and $a_0' \neq a_t'$. There exist the following cases depending on the relationship between a_0' and a_t':

- If $a_0' = a_t'$, we first upper bound $Q_t^{\pi_t^*}(s,a_t^*)$. We know that for any action $a \in \mathcal{A}$

  $$Q_t^{\pi_t^*}(s,a) \leq Q_0^{\pi_0^*}(s,a_0^*) + \epsilon$$

  Therefore,

  $$Q_t^{\pi_t^*}(s,a_t^*) \leq Q_0^{\pi_0^*}(s,a_0^*) + \epsilon \tag{5}$$

  We can then simply use the definition of ϵ to lower bound $Q_t^{\pi_t^*}(s,a_t')$:

  $$Q_t^{\pi_t^*}(s,a_t') \geq Q_0^{\pi_0^*}(s,a_0') - \epsilon \tag{6}$$

  Finally, by combining Eq. (5) and Eq. (6), we can upper bound ψ_t

  $$\begin{aligned} Q_t^{\pi_t^*}(s,a_t^*) - Q_t^{\pi_t^*}(s,a_t') &\leq (Q_0^{\pi_0^*}(s,a_0^*) + \epsilon) - (Q_0^{\pi_0^*}(s,a_0') - \epsilon) \\ \Rightarrow Q_t^{\pi_t^*}(s,a_t^*) - Q_t^{\pi_t^*}(s,a_t') &\leq Q_0^{\pi_0^*}(s,a_0^*) - Q_0^{\pi_0^*}(s,a_0') + 2\epsilon \\ \Rightarrow \psi_t &\leq \psi_0 + 2\epsilon \end{aligned}$$

- If $a_0' = a_t^*$, i.e., the second best action at time step 0 became the optimal action at time step t, then this implies that $a_0^* \neq a_t^*$. This implies:

  $$\psi_t \leq 2\epsilon$$

  as it would otherwise be impossible for the optimal action at timestep 0 to become sub-optimal at timestep t. Since ψ_0 is positive by definition, this implies that $\psi_t \leq \psi_0 + 2\epsilon$

- If $a_0' \notin \{a_t^*, a_t'\}$, then it follows from the definition of a_t' that $Q_0^{\pi_0^*}(s,a_t') \leq Q_0^{\pi_0^*}(s,a_0')$ and $Q_t^{\pi_t^*}(s,a_t') \geq Q_t^{\pi_t^*}(s,a_0')$.

  Let us assume that $Q_t^{\pi_t^*}(s,a_t') < Q_0^{\pi_0^*}(s,a_0') - \epsilon$. Then, by substituting $Q_0^{\pi_0^*}(s,a_0') - \epsilon$ for $Q_t^{\pi_t^*}(s,a_0')$ using Eq. (4), this implies that $Q_t^{\pi_t^*}(s,a_t') < Q_t^{\pi_t^*}(s,a_0')$. This is a contradiction given the above inequalities. Therefore, through proof by contradiction, $Q_t^{\pi_t^*}(s,a_t') \geq Q_0^{\pi_0^*}(s,a_0') - \epsilon$. Similar to the $a_0' = a_t'$ case above, we can combine this with the upper bound in Eq. (5) to find that

  $$\psi_t \leq 2\epsilon$$

In each case, $\psi_t \leq 2\epsilon$. □

Corollary 2.2. *If $\alpha\epsilon + (1-\alpha)\delta \leq \frac{\psi_0}{2} + \epsilon$, PA-MCTS is guaranteed to select the optimal action at decision epoch t.*

Proof. Using Corollary 2.1, we substitute ψ_t in terms of ψ_0 in Theorem 2.

$$\begin{aligned} \alpha\epsilon + (1-\alpha)\delta &\le \frac{\psi_t}{2} \\ \Rightarrow \alpha\epsilon + (1-\alpha)\delta &\le \frac{\psi_0 + 2\epsilon}{2} \\ \Rightarrow \alpha\epsilon + (1-\alpha)\delta &\le \frac{\psi_0}{2} + \epsilon \end{aligned}$$

□

Corollary 2.3. *PA-MCTS will choose the optimal one-step action if*

$$\begin{cases} \frac{-\delta}{\epsilon-\delta} \le \alpha \le \frac{\psi_0}{2(\epsilon-\delta)} + 1 & \text{if } \epsilon > \delta \\ \frac{-\delta}{\epsilon-\delta} \ge \alpha \ge \frac{\psi_0}{2(\epsilon-\delta)} + 1 & \text{if } \epsilon < \delta. \end{cases}$$

Proof. Using $0 \le \alpha \le 1$, Corollary 2.2 can be rearranged to solve for α:

$$\begin{aligned} 0 \le \alpha\epsilon + (1-\alpha)\delta &\le \frac{\psi_0}{2} + \epsilon \\ \Rightarrow 0 \le \alpha\epsilon - \alpha\delta + \delta &\le \frac{\psi_0}{2} + \epsilon \\ \Rightarrow -\delta \le \alpha\epsilon - \alpha\delta &\le \frac{\psi_0}{2} + \epsilon - \delta \\ \Rightarrow &\begin{cases} \frac{-\delta}{\epsilon-\delta} \le \alpha \le \frac{\psi_0}{2(\epsilon-\delta)} + 1 & \text{if } \epsilon > \delta \\ \frac{-\delta}{\epsilon-\delta} \ge \alpha \ge \frac{\psi_0}{2(\epsilon-\delta)} + 1 & \text{if } \epsilon < \delta. \end{cases} \end{aligned}$$

□

Proposition 1. *If PA-MCTS and MCTS choose different actions, PA-MCTS's chosen action will have a higher $Q_t^{\pi_t^*}$ value than MCTS if $2\epsilon \le \zeta_t^m$, where*[7] *$\zeta_t^m := (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_m) + \overline{G}_t(s,a_m))$, $\tilde{a} := \arg\max_{a\in\mathcal{A}_s} \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a)$, and $a_m := \arg\max_{a\in\mathcal{A}_s} \overline{G}_t(s,a)$.*

Proof. PA-MCTS's chosen action $\tilde{a}$ has a higher $Q_t^{\pi_t^*}$ value than MCTS's chosen action a_m when

$$0 \le Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m) \tag{7}$$

If PA-MCTS and MCTS choose different actions, then it must be that $\overline{G}_t(s,a_m) \ge \overline{G}_t(s,\tilde{a})$ and $\alpha Q_0^{\pi_0^*}(s,a_m) + (1-\alpha)\overline{G}_t(s,a_m) \le \alpha Q_0^{\pi_0^*}(s,\tilde{a}) + (1-\alpha)\overline{G}_t(s,\tilde{a})$. Then, we can bound $Q_t^{\pi_t^*}$ with respect to $Q_0^{\pi_0^*}$ and ϵ:

$$Q_0^{\pi_0^*}(s,\tilde{a}) - Q_0^{\pi_0^*}(s,a_m) - 2\epsilon \le Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m)$$

Next, we rewrite this bound in terms of ζ_t^m and ϵ:

$$\begin{aligned} Q_0^{\pi_0^*}(s,\tilde{a}) - Q_0^{\pi_0^*}(s,a_m) - 2\epsilon &\le Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m) \\ \Rightarrow Q_0^{\pi_0^*}(s,\tilde{a}) - Q_0^{\pi_0^*}(s,a_m) - 2\epsilon + \overline{G}_t(s,a_m) - \overline{G}_t(s,a_m) &\le Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m) \quad (8) \\ \Rightarrow Q_0^{\pi_0^*}(s,\tilde{a}) - Q_0^{\pi_0^*}(s,a_m) - 2\epsilon + \overline{G}_t(s,\tilde{a}) - \overline{G}_t(s,a_m) &\le Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m) \quad (9) \\ \Rightarrow (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_m) + \overline{G}_t(s,a_m)) - 2\epsilon &\le Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m) \quad (10) \\ \Rightarrow \zeta_t^m - 2\epsilon &\le Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_m) \end{aligned}$$

where in Eq. (8) we add and subtract $\overline{G}_t(s,a_m)$ to the left hand side, in Eq. (9) we use $\overline{G}_t(s,a_m) \ge \overline{G}_t(s,\tilde{a})$ to substitute $\overline{G}_t(s,\tilde{a})$ for $\overline{G}_t(s,a_m)$, and in Eq. (10) we rearrange the left hand side.

Therefore, it follows from Eq. (7) that PA-MCTS chooses an action with higher $Q_t^{\pi_t^*}$ value when $0 < \zeta_t^m - 2\epsilon$, i.e., $2\epsilon \le \zeta_t^m$. □

[7] The superscript m in ζ_t^m represents the bounds with respect to MCTS and is not an index.

Proposition 2. *If the action $\tilde{a}$ chosen by PA-MCTS is different than $a_g := \arg\max_{a \in \mathcal{A}_s} Q_0^{\pi_0^*}(s,a)$ at state s, then PA-MCTS's chosen action $\tilde{a}$ will have a higher $Q_t^{\pi_t^*}$ value than action a_g if $2\delta \ \leq \zeta_t^q$ where*[8] $\zeta_t^q := (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_g) + \overline{G}_t(s,a_g))$, $\tilde{a} := \arg\max_{a \in \mathcal{A}_s} \alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a)$, *and* $a_g := \arg\max_{a \in \mathcal{A}_s} Q_0^{\pi_0^*}(s,a)$.

Proof. PA-MCTS's chosen action $\tilde{a}$ has a higher $Q_t^{\pi_t^*}$ value than the action selected through $Q_0^{\pi_0^*}$-values a_g when

$$0 \leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_g) \tag{11}$$

If PA-MCTS and action selection through $Q_0^{\pi_0^*}$-values choose different actions, then it must be that $Q_0^{\pi_0^*}(s,a_g) \geq Q_0^{\pi_0^*}(s,\tilde{a})$ and $\alpha Q_0^{\pi_0^*}(s,a_g) + (1-\alpha)\overline{G}_t(s,a_g) \leq \alpha Q_0^{\pi_0^*}(s,\tilde{a}) + (1-\alpha)\overline{G}_t(s,\tilde{a})$. Then, we can bound $Q_t^{\pi_t^*}$ with respect to $\overline{G}_t$ and δ:

$$\overline{G}_t(s,\tilde{a}) - \overline{G}_t(s,a_g) - 2\delta \leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_g)$$

Next, we can rewrite this bound in terms of ζ_t^q and δ:

$$\begin{aligned}
\overline{G}_t(s,\tilde{a}) - \overline{G}_t(s,a_g) - 2\delta &\leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_g) \\
\Rightarrow \overline{G}_t(s,\tilde{a}) - \overline{G}_t(s,a_g) - 2\delta + Q_0^{\pi_0^*}(s,a_g) - Q_0^{\pi_0^*}(s,a_g) &\leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_g) \qquad (12) \\
\Rightarrow \overline{G}_t(s,\tilde{a}) - \overline{G}_t(s,a_g) - 2\delta + Q_0^{\pi_0^*}(s,\tilde{a}) - Q_0^{\pi_0^*}(s,a_g) &\leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_g) \qquad (13) \\
\Rightarrow (Q_0^{\pi_0^*}(s,\tilde{a}) + \overline{G}_t(s,\tilde{a})) - (Q_0^{\pi_0^*}(s,a_g) + \overline{G}_t(s,a_g)) - 2\delta &\leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_g) \qquad (14) \\
\Rightarrow \zeta_t^g - 2\delta &\leq Q_t^{\pi_t^*}(s,\tilde{a}) - Q_t^{\pi_t^*}(s,a_g)
\end{aligned}$$

where in Eq. (12) we introduce $Q_0^{\pi_0^*}(s,a_g)$ by adding zero to the left hand side, in Eq. (13) we use $Q_0^{\pi_0^*}(s,a_g) \geq Q_0^{\pi_0^*}(s,\tilde{a})$ to substitute $Q_0^{\pi_0^*}(s,\tilde{a})$ for $Q_0^{\pi_0^*}(s,a_g)$, and in Eq. (14) we rearrange the left hand side.

Therefore, it follows from Eq. (11) that PA-MCTS chooses an action with higher $Q_t^{\pi_t^*}$ value when $0 < \zeta_t^g - 2\delta$, i.e., $2\delta \ \leq \zeta_t^g$. □

Theorem 3. *When PA-MCTS is used for sequential decision making, the maximum difference between the return from an optimal policy and the return from following PA-MCTS is at most* $\frac{2(\alpha\epsilon - \alpha\delta + \delta)}{1-\gamma}$.

Proof. The proof closely follows prior work by Wray [29]. Let $\mathcal{A}'_s$ denote the set of actions that can be taken by PA-MCTS in state s at any decision epoch. Given ϵ, δ, and $a \in \mathcal{A}'_s$, it must be that

$$\alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a) \geq \alpha Q_0^{\pi_0^*}(s,a_t^*) + (1-\alpha)\overline{G}_t(s,a_t^*) \tag{15}$$

for a to be chosen by PA-MCTS. We can substitute $Q_t^{\pi_t^*}(s,a)$ for $Q_0^{\pi_0^*}$ and $\overline{G}_t$ in Eq. (15) to get

$$\begin{aligned}
\alpha Q_0^{\pi_0^*}(s,a) + (1-\alpha)\overline{G}_t(s,a) &\geq \alpha Q_0^{\pi_0^*}(s,a_t^*) + (1-\alpha)\overline{G}_t(s,a_t^*) \\
\Rightarrow \alpha(Q_t^{\pi_t^*}(s,a) + \epsilon) + (1-\alpha)(Q_t^{\pi_t^*}(s,a) + \delta) &\geq \alpha(Q_t^{\pi_t^*}(s,a_t^*) - \epsilon) + (1-\alpha)(Q_t^{\pi_t^*}(s,a_t^*) - \delta) \\
\Rightarrow 2(\alpha\epsilon + (1-\alpha)\delta) &\geq Q_t^{\pi_t^*}(s,a_t^*) - Q_t^{\pi_t^*}(s,a) \\
\Rightarrow 2(\alpha\epsilon + (1-\alpha)\delta) &\geq V_t^{\pi_t^*}(s) - Q_t^{\pi_t^*}(s,a)
\end{aligned}$$

where

$$V_t^{\pi}(s) = R_t(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P_t(s'|s,a) V^{\pi}(s') \tag{16}$$

where $R_t(s,a)$ is the reward for taking action a at s at time step t, $\pi(s)$ is policy π's distribution over actions given s, γ is discount factor, and $P_t(s'|s,a)$ is the probability of reaching state s' when taking a at s at time step t. From Eq. (16), the optimal value function can be expressed as $V_t^{\pi_t^*}(s) = \arg\max_a Q_t^{\pi_t^*}(s,a)$.

Therefore

$$\mathcal{A}'_s = \{a \in \mathcal{A}_s \mid V_t^{\pi_t^*}(s) - Q_t^{\pi_t^*}(s,a) \leq 2(\alpha\epsilon + (1-\alpha)\delta)\}$$

Now, let $V_t^{\tilde{\pi},k}(s)$ be k applications of Eq. (16) following PA-MCTS's policy $\tilde{\pi}$, i.e., $\tilde{\pi} \in \Pi'$ where $\Pi' = \{\pi | \pi(s) \in \mathcal{A}'_s, \forall s \in \mathcal{S}\}$. Also let $Q_t^{\pi_t^*,k}(s,\tilde{\pi}(s))$ be a one-step action deviation following $\tilde{\pi}$, after which an optimal policy π_t^* is followed for the remaining $k-1$ iterations.

[8] Again, the superscript q only denotes bounds with respect to the Q-values and is not an index.

Next, we show that for any state $s \in \mathcal{S}$:

$$V_t^{\tilde{\pi},k}(s) \geq V_t^{\pi_t^*,k}(s) - \sum_{\tau=0}^{k} \gamma^\tau 2(\alpha\epsilon + (1-\alpha)\delta) \tag{17}$$

by induction on k iterations of Eq. (16).

Base Case: At $k = 0$, by the definition of $\tilde{\pi}(s) \in \mathcal{A}'_s$, we have:

$$\begin{aligned}
&V_t^{\pi_t^*,0}(s) - Q_t^{\pi_t^*,0}(s, \tilde{\pi}(s)) \leq 2(\alpha\epsilon + (1-\alpha)\delta) \\
\Rightarrow\ &V_t^{\pi_t^*,0}(s) - R_t(s, \tilde{\pi}(s)) \leq 2(\alpha\epsilon + (1-\alpha)\delta) \\
\Rightarrow\ &V_t^{\pi_t^*,0}(s) - V_t^{\tilde{\pi},0}(s) \leq 2(\alpha\epsilon + (1-\alpha)\delta) \\
\Rightarrow\ &V_t^{\tilde{\pi},0}(s) \geq V_t^{\pi_t^*,0}(s) - 2(\alpha\epsilon + (1-\alpha)\delta)
\end{aligned}$$

Induction Step: Assume for $k-1$ the induction hypothesis:

$$V_t^{\tilde{\pi},k-1}(s) \geq V_t^{\pi_t^*,k-1}(s) - \sum_{\tau=0}^{k-1} \gamma^\tau 2(\alpha\epsilon + (1-\alpha)\delta)$$

is true. We now show that Eq. (17) is true for k:

$$V_t^{\tilde{\pi},k}(s) = R_t(s, \tilde{\pi}(s)) + \gamma \sum_{s' \in \mathcal{S}} P_t(s'|s,a) V^{\tilde{\pi},k-1}(s') \tag{18}$$

$$\Rightarrow V_t^{\tilde{\pi},k}(s) \geq R_t(s, \tilde{\pi}(s)) + \gamma \sum_{s' \in \mathcal{S}} P_t(s'|s,a)(V^{\pi_t^*,k-1}(s') - \sum_{\tau=0}^{k-1} \gamma^\tau 2(\alpha\epsilon + (1-\alpha)\delta)) \tag{19}$$

$$\Rightarrow V_t^{\tilde{\pi},k}(s) \geq (R_t(s, \tilde{\pi}(s)) + \gamma \sum_{s' \in \mathcal{S}} P_t(s'|s,a) V^{\pi_t^*,k-1}(s')) - \sum_{\tau=0}^{k-1} \gamma^{\tau+1} 2(\alpha\epsilon + (1-\alpha)\delta)) \tag{20}$$

$$\Rightarrow V_t^{\tilde{\pi},k}(s) \geq Q_t^{\pi_t^*,k}(s, \tilde{\pi}(s)) - \sum_{\tau=0}^{k-1} \gamma^{\tau+1} 2(\alpha\epsilon + (1-\alpha)\delta)) \tag{21}$$

$$\Rightarrow V_t^{\tilde{\pi},k}(s) \geq Q_t^{\pi_t^*,k}(s, \tilde{\pi}(s)) - \sum_{\tau=1}^{k} \gamma^{\tau} 2(\alpha\epsilon + (1-\alpha)\delta))$$

where Eq. (18) is by Eq. (16), Eq. (19) is by the induction hypothesis, Eq. (20) is by rewriting and normalizing, and Eq. (21) is by the definition of $Q_t^{\pi_t^*,k}$.

By the definition of $\mathcal{A}'_s$, we have $V_t^{\pi_t^*,k}(s) - Q_t^{\pi_t^*,k}(s, \tilde{\pi}(s) \leq 2(\alpha\epsilon + (1-\alpha)\delta)$. By rearranging this, applying to $Q_t^{\pi_t^*,k}$ and grouping $2(\alpha\epsilon + (1-\alpha)\delta)$ in the sum, we find

$$\begin{aligned}
V_t^{\tilde{\pi},k}(s) &\geq V_t^{\pi_t^*,k}(s) - 2(\alpha\epsilon + (1-\alpha)\delta) - \sum_{\tau=1}^{t} \gamma^\tau 2(\alpha\epsilon + (1-\alpha)\delta) \\
&= V_t^{\pi_t^*,k}(s) - \sum_{\tau=0}^{t} \gamma^\tau 2(\alpha\epsilon + (1-\alpha)\delta)
\end{aligned}$$

Thus, by induction on k, we have shown that Eq. (17) is true for all k. Finally let $k \longrightarrow \infty$ and evaluate Eq. (17):

$$V_t^{\tilde{\pi},k}(s) \geq V_t^{\pi_t^*,k}(s) - \sum_{\tau=0}^{t} \gamma^\tau 2(\alpha\epsilon + (1-\alpha)\delta) \tag{22}$$

$$\geq V_t^{\pi_t^*,k}(s) - \sum_{\tau=0}^{\infty} \gamma^\tau 2(\alpha\epsilon + (1-\alpha)\delta) \tag{23}$$

$$\geq V_t^{\pi_t^*,k}(s) - \frac{2(\alpha\epsilon + (1-\alpha)\delta)}{1-\gamma} \tag{24}$$

$$V_t^{\pi_t^*,k}(s) - V_t^{\tilde{\pi},k}(s) \leq \frac{2(\alpha\epsilon + (1-\alpha)\delta)}{1-\gamma} \tag{25}$$

where Eq. (22) is by Eq. (17), Eq. (23) is by $\gamma \in [0, 1)$ and $2(\alpha\epsilon + (1 - \alpha)\delta) \geq 0$, Eq. (24) is by geometric series, and Eq. (25) is by rearranging. □

B EXPERIMENTAL RESULTS ON THE CARTPOLE ENVIRONMENT

In the main text, we present results on decision-making in non-stationary settings by varying the gravity parameter in the cartpole environment. To gather more robust results, we also vary the total mass of the cartpole system by increasing the mass of the pole. We fix gravity=9.8m/s^2 and the mass of the *cart* as 1kg, but we vary the mass of the *pole* among {0.1kg, 1.0kg, 1.2kg, 1.3kg, 1.5kg}; note that as the mass of the pole increases, the total mass of the system under consideration (i.e., the mass of the cart and the pole) also increases. We keep all other hyper-parameters the same as the experiments where gravity is changed (shown in the main text).

B.1 Stationary Environment

The results for the stationary Environment are the same as the stationary environment for the gravity experiments. We present the results in the main text (Fig. 2 in the main text). We compute each result by averaging across 30 samples.

B.2 Non-Stationary Environment

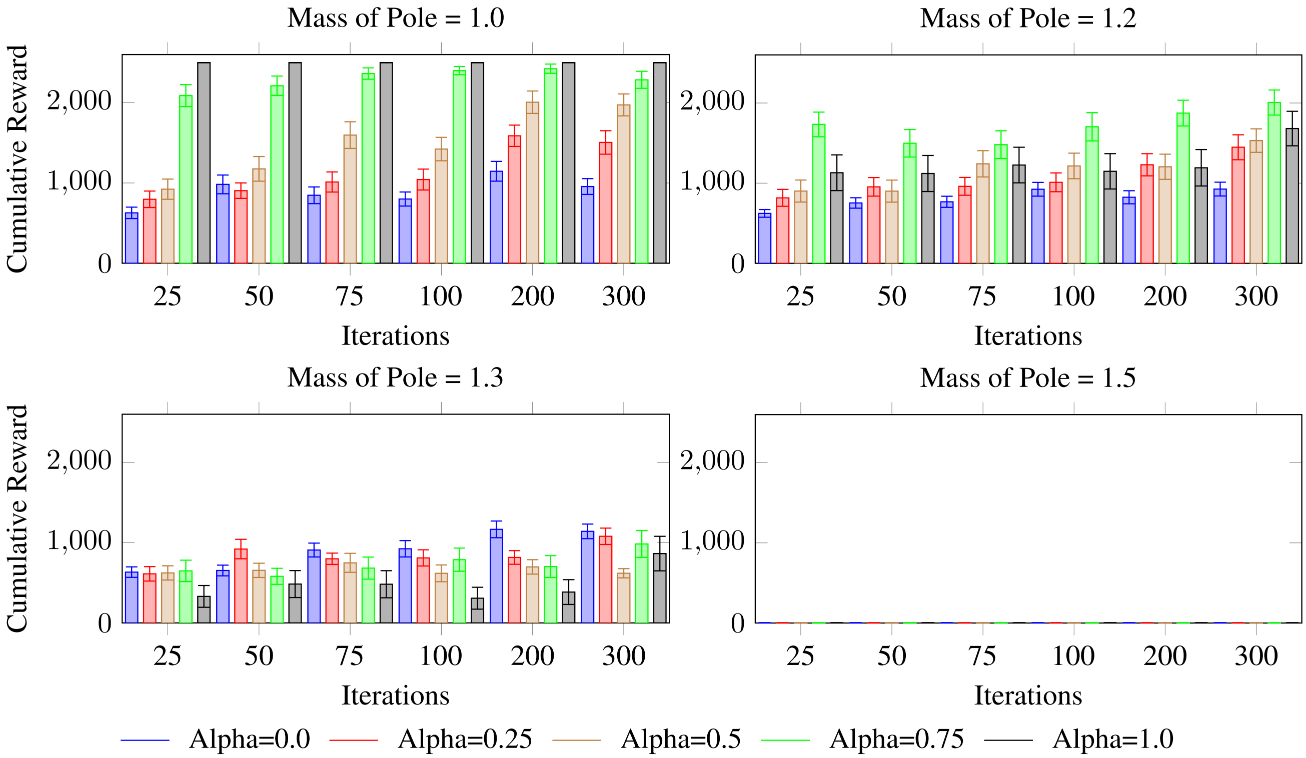

Figure 6: We show the average cumulative reward obtained by PA-MCTS, standard MCTS ($\alpha = 0$), and DDQN ($\alpha = 1$) for different environmental changes. The mass of the pole changes among {0.1kg, 1.0kg, 1.2kg, 1.3kg, 1.5kg}

We represent the non-stationary environment here by changing the mass of the pole on the cart. We show the results in Fig. 6. We vary the mass of the pole among {0.1kg, 1.0kg, 1.2kg, 1.3kg, 1.5kg}. Each α value used for PA-MCTS is represented by a different color. For each setting, we vary MCTS iterations among { 25, 50, 75, 100, 200, 300 } on 30 samples (test episodes). As Fig. 6 shows, DDQN achieves maximum reward in the stationary setting but its performance deteriorates under non-stationarity. We observe that PA-MCTS, given appropriate α, achieves significantly higher reward than both standard MCTS and DDQN under non-stationary settings. Also, as we observe in the setting where gravity is changed, PA-MCTS converges significantly faster than standard MCTS by leveraging the pre-trained policy. Finally, we observe that all approaches (PA-MCTS, standard MCTS, and DDQN) achieve close to 0 utility when the mass of the pole is set to 1.5kg, presumably because it is extremely difficult to balance the pole in this setting.

B.3 Choosing the Optimal α

We propose selecting α as before, i.e., we observe how PA-MCTS performs under different computational constraints set by the number of MCTS iterations. Again, we observe that the optimal α for 100 MCTS iterations is well approximated (almost perfectly) by 25 MCTS iterations. We show the results in Fig. 7.

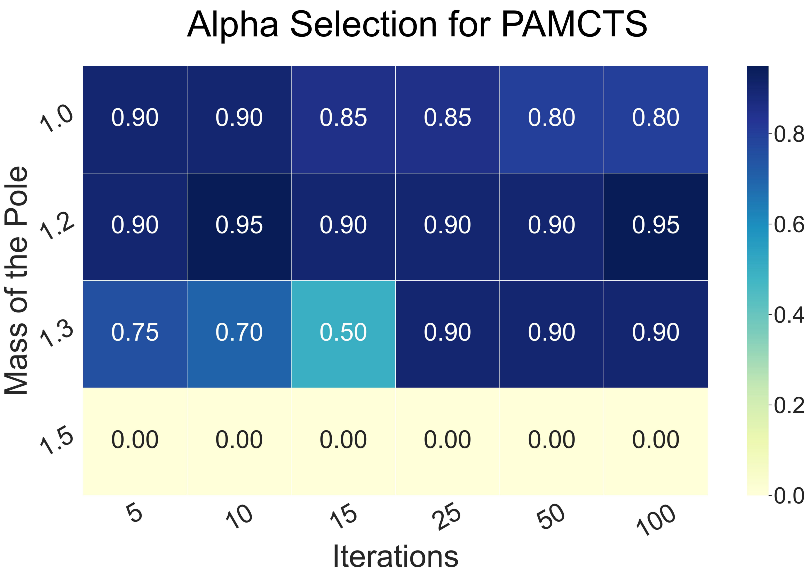

Figure 7: Cartpole Alpha Selection Changing Mass of the Pole among 1.0kg, 1.2kg, 1.3kg, 1.5kg

.

B.4 Comparison with Alphazero

We use the proposed α-selection strategy to find the optimal alpha value for a given non-stationary setting, i.e., for each mass of the pole. Then, we compare PA-MCTS with the optimal α with Alphazero and present results in Fig. 8. Both AlphaZero and PA-MCTS achieve 0 utility when mass=1.5kg. When mass=1.0kg, we see that PA-MCTS outperforms AlphaZero by converging significantly faster. When mass=1.2kg, we see that AlphaZero slightly outperforms PA-MCTS.

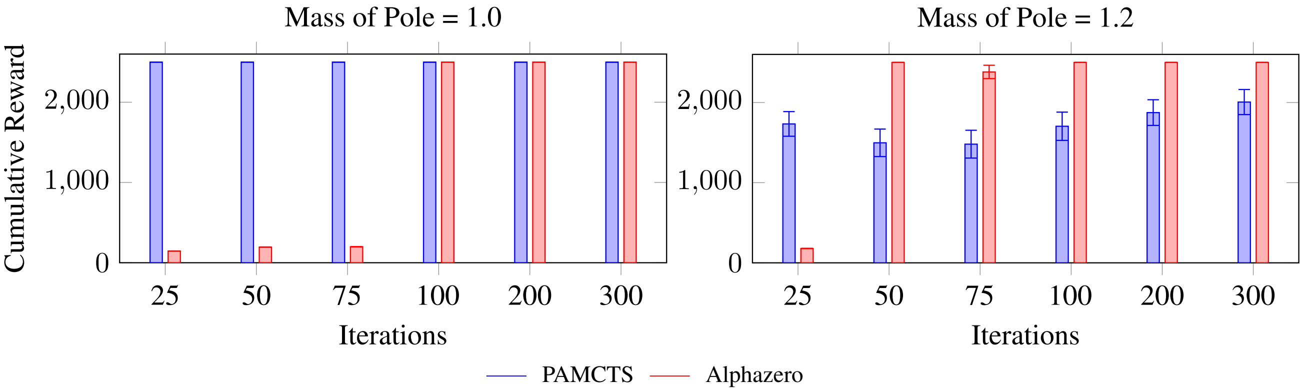

Figure 8: We compare between the results of PA-MCTS with optimal alpha and the results of Alphazero

B.5 Noisy settings

We now consider the setting that in practice, it is difficult to observe the environmental change exactly. As a result, we add noise between the environmental change and how the agent perceives the change. We introduce Gaussian noise (with 0 mean and varying standard deviation)

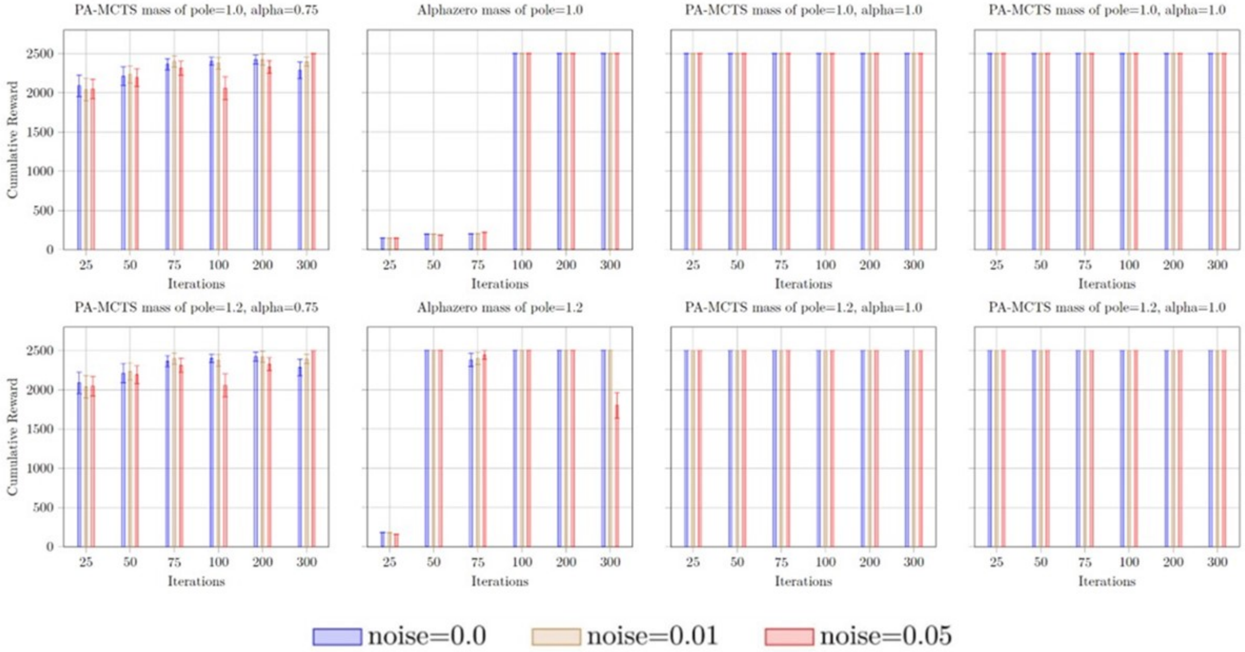

Figure 9: The performance of PA-MCTS and AlphaZero under varying levels of noise with mass of pole = 1.0kg, 1.2kg. PA-MCTS converges faster than Alphazero with small MCTS iterations.

in the mass, i.e., the agent perceives a noisy mass during simulation but the true non-stationary mass is used for execution. We show the results in Fig. 9. We observe that even under noisy settings, PA-MCTS achieves higher rewards than AlphaZero in most cases, especially when the agent can afford low computational time for making decisions.

B.6 Hardware

The experiments were conducted on Chameleon testbed [11] on 4 Linux systems with 32-96 logical processors and 528 GB RAM.

B.7 Extensive Results for Noisy Settings Changing Gravity

In the main text, due to the space limitation, we only showed partial results for the noisy settings where gravity is changed. Here, we present complete results (for each value of updated gravity) with noisy settings. We show the extensive comparison results between PA-MCTS, Alphazero, Standard MCTS, and DDQN (as shown in Fig. 10). We observe that (as in the main text), PA-MCTS outperforms the baselines even under noisy settings.

C EXPERIMENTAL RESULTS ON THE FROZENLAKE ENVIRONMENT

We also conduct experiments using the Frozenlake Environment. The goal of the agent is to travel on a slippery surface (a frozen lake) from a pre-defined starting position to a pre-defined goal position without falling into any holes. Due to the slippery nature of the surface, the agent does not always move in the intended direction. We introduce non-stationarity by changing the probability of moving toward the desired direction. We vary this probability and set the probability of moving in the two perpendicular directions as equal. We represent an environment by a tuple of three probabilities which shows the **[probability to move towards the desired direction, probability to move towards the first perpendicular direction, probability to move towards the second perpendicular direction]**, e.g., $[1, 0, 0]$ refers to the stationary setting, where the surface is not slippery and the agent moves in its desired direction with certainty (i.e., probability of 1) and there are no chances of the agent moving in directions perpendicular to its desired direction (denoted by the two 0 entries). Given this setting, note that we can calculate η (the amount of change in the probability transition function as defined in the main text); $\eta_{min} = 0$, when there is no change, i.e., the updated environment is the same as the stationary environment ($[1, 0, 0]$), and $\eta_{max} = 1.333$ when the updated environment is denoted by [0.333, 0.333, 0.333]. We show the environments we use for testing in Table 3. We choose non-stationary environments by uniformly sampling η value from η_{min} to η_{max}. We use a 3x3 map for Frozen Lake, with position indices from 0 to 8 as

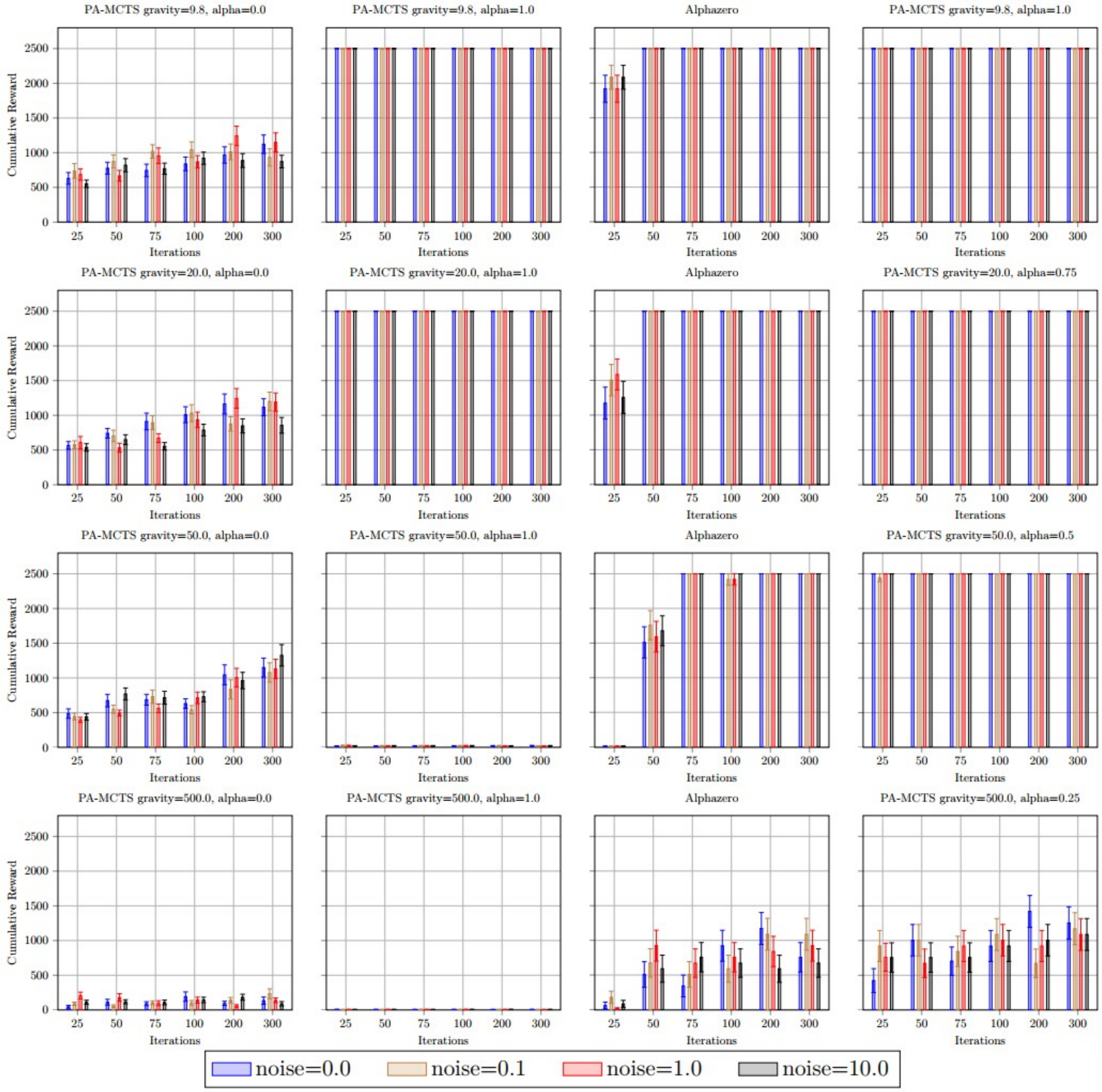

Figure 10: We compare the performance of PA-MCTS, Alphazero, Standard MCTS, DDQN for noise among 0.0, 0.1, 1, 10 with different gravity values among 9.8 m/s^2, 20.0 m/s^2, 50.0 m/s^2, 500.0 m/s^2

.

shown in the main text. The agent starts from cell 0, the goal is set at cell 8, and there are holes in cells 1 and 6. We show a schematic of the environment in Fig. 11.

As Table 3 shows, we choose probability distributions based upon uniformly sampling η value from η_{min} to η_{max}. The map used for this experiment is a 3x3 map, with position index from 0 to 8 and 0 is the start position, 8 is the destination, and position index 1 and 6 are holes.

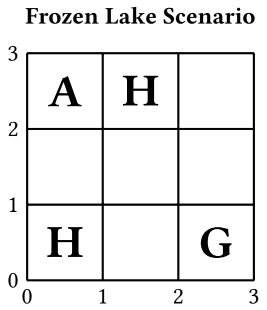

Figure 11: The test environment for the FrozenLake experiments. The agent must traverse from cell (0,2) to cell (2,0) without falling into cell (0,0) and cell (1,2). However, under slippery conditions, the agent does not always move in the desired direction.

probability distribution	η
[1.0, 0.0, 0.0]	0
[0.933, 0.033, 0.033]	0.2
[0.833, 0.083, 0.083]	0.4
[0.733, 0.133, 0.133]	0.6
[0.633, 0.183, 0.183]	0.8
[0.533, 0.233, 0.233]	1.0
[0.433, 0.283, 0.283]	1.2
[0.333, 0.333, 0.333]	1.333

Table 3: Probability distribution of the testing environments and corresponding η

C.1 Stationary Environment

As described above, the stationary environment corresponds to the setting $[1, 0, 0]$, i.e., the lake is not slippery. As we see in Fig. 12 shows, standard MCTS, PA-MCTS (with all values of α) and DDQN can achieve maximum rewards. We compute each result by averaging across 100 samples.

C.2 Non-Stationary Environment

We also show results in non-stationary settings results in Fig. 12. We use different colors to represent different environments. We show results for five settings in the figure to make the plot readable (the results are consistent across the other settings). Like the cartpole environment, we observe that the performance of the DDQN deteriorates as the environment changes.

This behavior is expected, as actions that are favorable in the non-slippery setting can now lead the agent to a hole. We also see that for most settings, PA-MCTS achieves the optimal utility. In some settings, while PA-MCTS does not achieve the highest possible utility, it outperforms both standard MCTS and DDQN given the optimal α (we discuss how to tune α below).

C.3 Choosing the optimal α

As we describe in the main text, during execution, the agent needs to pick an optimal α quickly. Similar to the cartpole environment, we leverage the finding that the agent can approximate the performance of different values of α by using a small number of PA-MCTS iterations. We show the results in Fig. 13; note that PA-MCTS with 25 iterations approximates the optimal α with 500 iterations (naturally, this approach is not optimal; we hypothesize that the α-selection mechanism can be operated in parallel even after an initial α is chosen, and the agent can update the parameters over time).

C.4 Comparison with Alphazero

Finally, we use the proposed α-selection strategy to select the optimal α value for different environment settings. Then, we compare PA-MCTS with Alphazero. Fig. 14 shows that as the non-stationary environment deviates further and further from the stationary environment, Alphazero performs significantly worse than PA-MCTS. We hypothesize this loss of performance is due to the fact that AlphaZero uses a pre-trained DDQN (on the stationary environment) to simulate rollouts, whereas PA-MCTS uses *actual* rollouts as it grows the search tree.

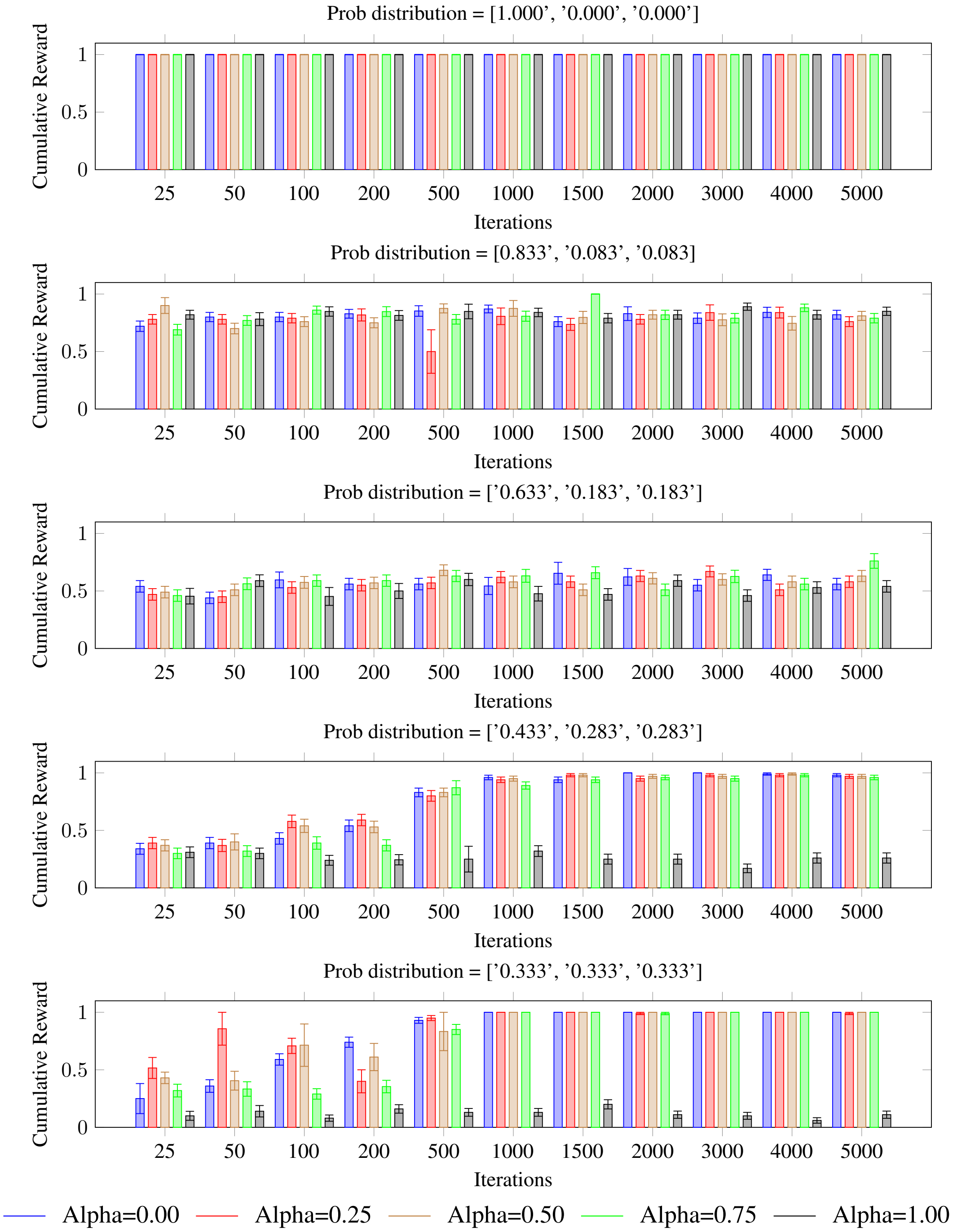

Figure 12: We show the cumulative reward obtained by PA-MCTS, standard MCTS ($\alpha = 0$), and DDQN ($\alpha = 1$ for different environmental changes.

C.5 Time Constraints

Finally, we explore the time taken to run PA-MCTS (with α-selection, i.e., the additional time to tune α is included in the computational time we report) in comparison with the baseline approaches. Again, we point out that for a fair comparison, we re-train both DDQN and AlphaZero starting from the network trained on the stationary setting; we observed that this method significantly reduces the training time

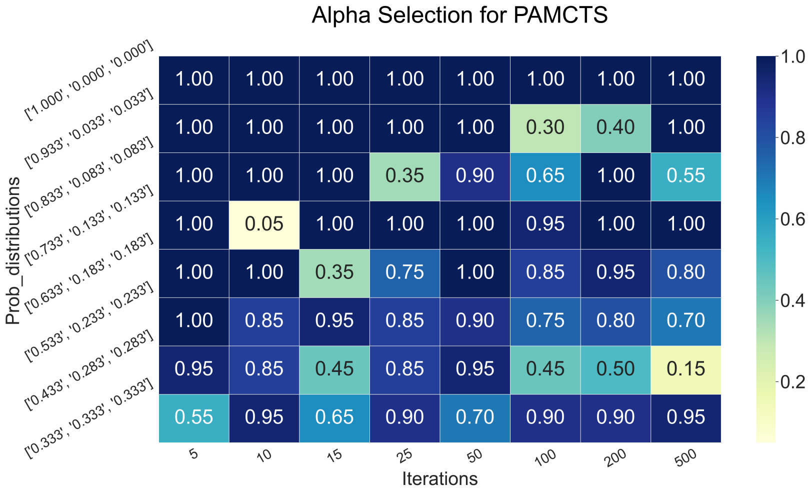

Figure 13: Frozenlake Alpha Selection for different probability distributions

.

than re-training from scratch. We present the results in Table 4. Also, to be fair to the AlphaZero framework, we create 100 data generating processes in parallel to feed into its re-training procedure (we leverage Apache Kafka to achieve this framework). For the DDQN, we use a maximum of 1,000,000 million steps or stop the training if the network converges. For PA-MCTS, we run α-selection in parallel. As we see, it is significantly faster to run PA-MCTS than re-training the networks.

Method	Iterations	Execution Time (in seconds)
PA-MCTS	5	0.001
PA-MCTS	10	0.002
PA-MCTS	15	0.005
PA-MCTS	25	0.006
PA-MCTS	50	0.007
PA-MCTS	100	0.012
PA-MCTS	200	0.029
PA-MCTS	500	0.08
PA-MCTS	1000	0.3
PA-MCTS	1500	0.41
PA-MCTS	2000	0.645
PA-MCTS	3000	0.856
Re-Train DDQN	1,000,000 steps	1865.75
Re-Train AlphaZero	NA	3900

Table 4: Execution time of PA-MCTS (with α-selection) with varying MCTS iterations in comparison with DDQN and AlphaZero. We observe that PA-MCTS is significantly faster.

C.6 Hardware

The experiments were conducted on Chameleon testbed [11] on 4 Linux systems with 32-96 logical processors and 528 GB RAM.

D EXPERIMENTAL RESULTS ON THE CLIFF WALKING ENVIRONMENT

We also conduct experiments on the Cliff Walking Environment. The environment has 48 possible states. Instead of Cliff Walking implemented by OpenAI Gymnasium [4], which only has deterministic transition functions, we implement our own Cliff Walking environment, which has stochastic transition functions (our code is available through the supplementary material). We show a schematic of the environment in Fig. 15; cell 36 is the start position, and state 47 is defined as the goal position. States in 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, shown in brown color in Fig. 15 are defined as cliff region. Unlike OpenAI Gymnasium version [4], our environment ends when the agent falls off to a cliff, or the agent reaches the goal position. The agent receives a reward only if it reaches the goal position, and the corresponding reward is related

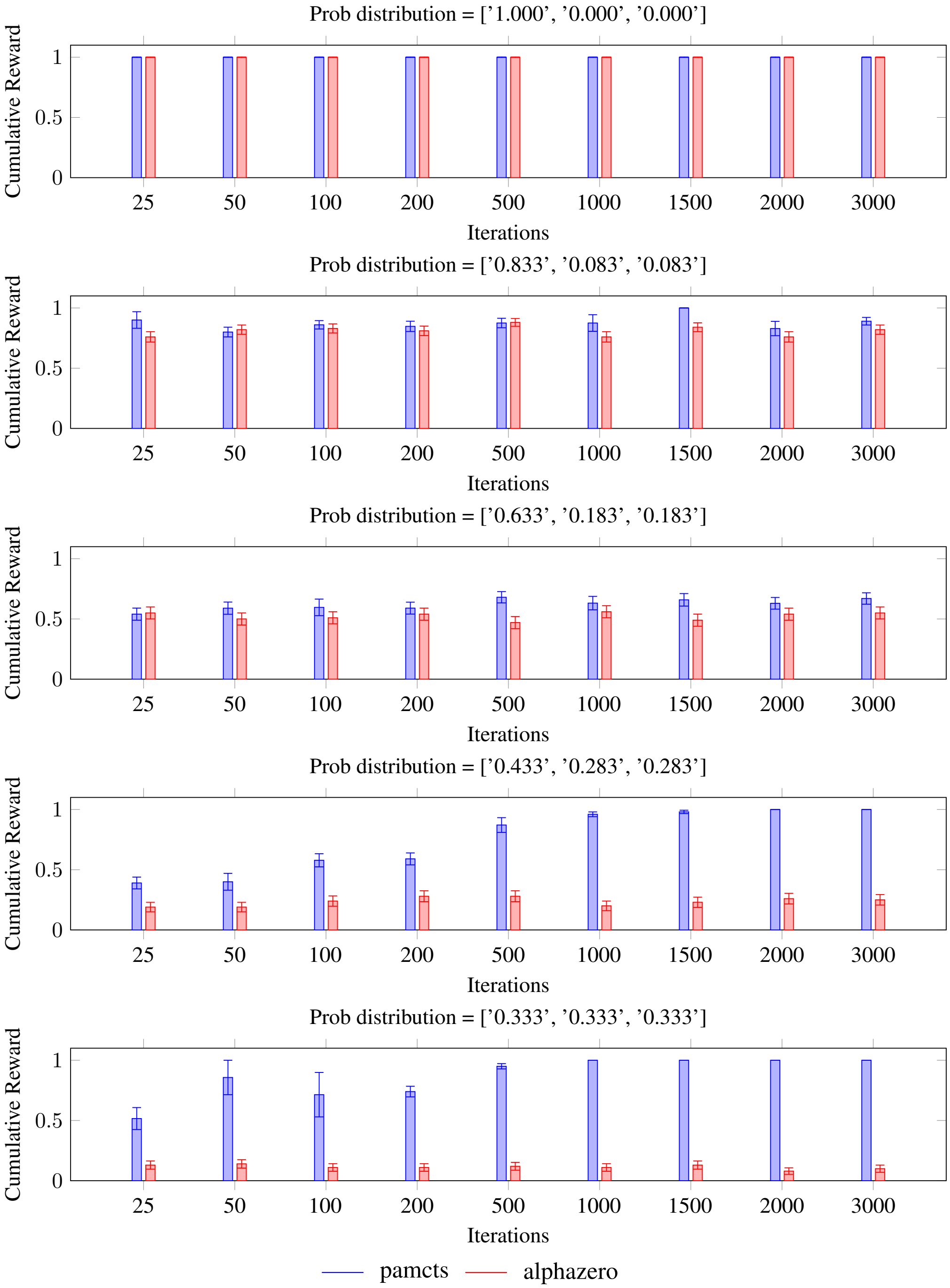

Figure 14: Comparison of PA-MCTS with Alphazero; we see that PA-MCTS comprehensively outperforms Alphazero in non-stationary settings.

to how many steps the agent takes to reach the goal position from the start position. The fewer steps the agent takes, the more reward is received. For instance, the agent receives more reward by choosing path [36, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 47] than choosing path [36, 24, 12, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 23, 35, 47]. We introduce a slippery nature in this environment in the same manner as the Frozen Lake environment, with one critical difference to account for the fact that the hill has a specific location. If the slippery factor is 0.1, then there is a 0.1 probability that the agent goes to the cell below its current location (except when it explicitly wants to go down, in which case it always goes down).

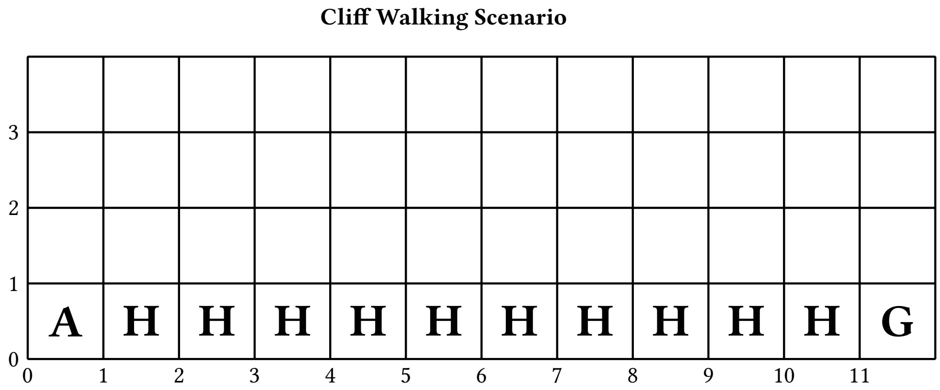

Figure 15: The test environment for the Cliff Walking experiments. The agent must traverse from cell (1,0) to cell (10,0) without falling into the cliff. Under Slippery conditions, the agent does not always go in the desired direction.

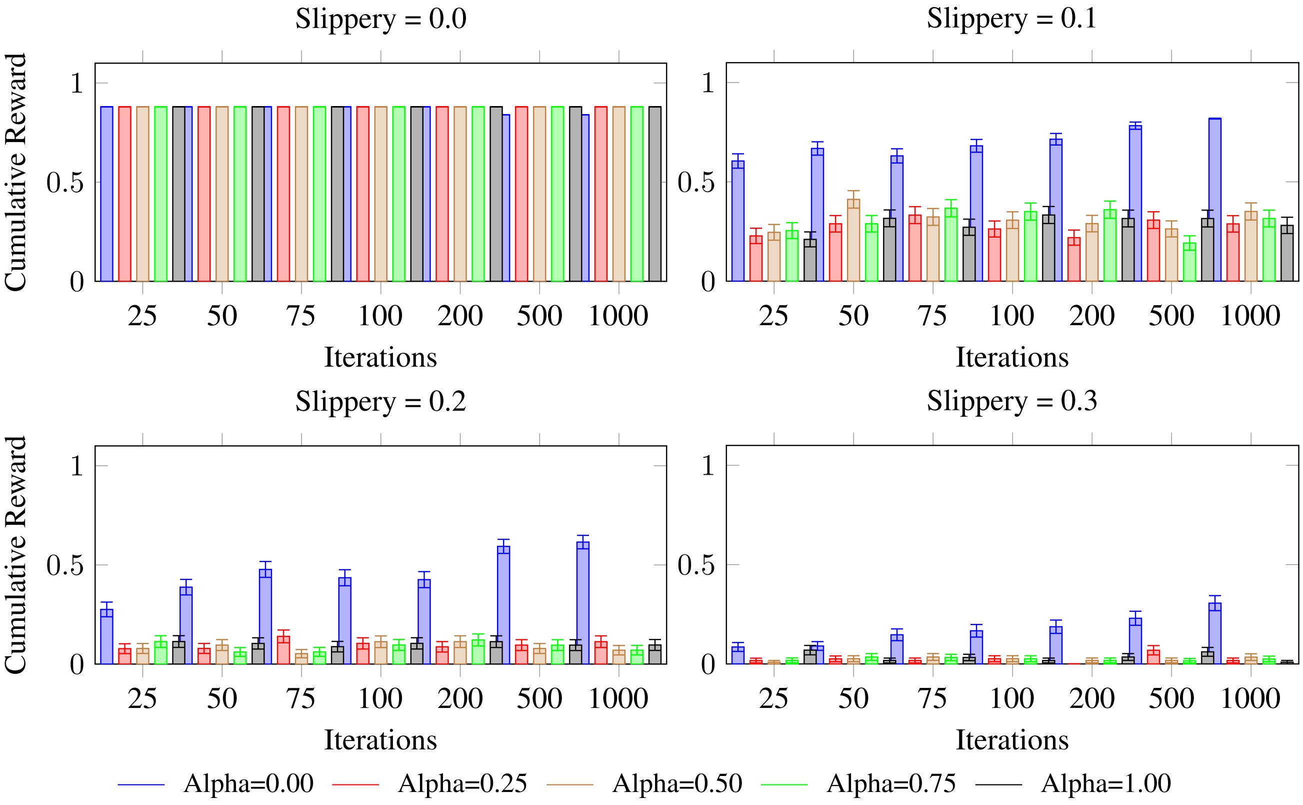

Figure 16: We show the cumulative reward obtained by PA-MCTS, standard MCTS ($\alpha = 0$), and DDQN ($\alpha = 1$) for different slippery levels among 0.0, 0.1, 0.2, 0.3

D.1 Stationary Environment

The stationary environment is defined as the slippery factor is 0, i.e. the environment is not slippery. As we see in Fig. 16, both PA-MCTS and DDQN can achieve maximum rewards. There is some noise in the performance of standard MCTS. We believe this randomness could be alleviated by introducing more iterations.We compute each result by averaging across 100 samples.

D.2 Non-Stationary Environment

In Fig. 16, we also show results in non-stationary settings. We use different colors to represent different α values used for PA-MCTS. Similar to the Frozen Lake environment, the performance of the DDQN deteriorates as the environment changes. This behavior is expected, as actions that are favorable in the non-slippery setting can now lead the agent to the cliff region. As the surface becomes more slippery, we observe that using standard MCTS is better than leveraging learned Q-values. Indeed, PA-MCTS selects 0 as the optimal α, disregarding the learned Q-values. This shows the robustness of PA-MCTS, i.e., when the learned Q-values are not useful, it reverts to a fully online search.

D.3 Choosing the optimal α

During execution, the agent needs to pick an optimal α quickly. Similar to previous environments, we utilize the observation that the agent can emulate the performance outcomes from various α values by executing a limited count of PA-MCTS iterations. The outcomes are displayed in Fig. 17. For a stationary environment, counting on DDQN could give a maximum cumulative reward. In non-stationary environments, the selection is more biased towards $alpha = 0.0$, as the optimal path chosen by DDQN becomes risky. Given 50 iterations, MCTS results become more reliable.

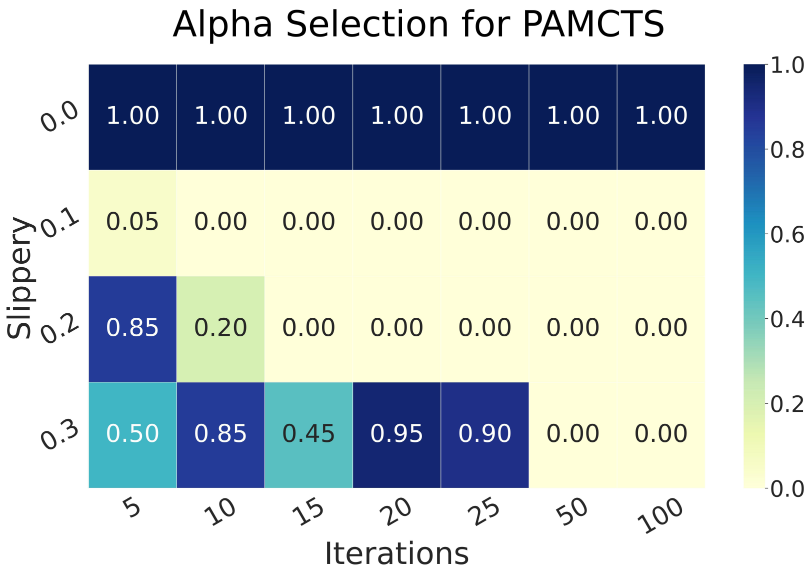

Figure 17: Cliff Walking Alpha Selection for different Slippery Levels

.

D.4 Comparison with Alphazero

We compare PA-MCTS with Alphazero after we use the proposed α-selection strategy to find the optimal α value for different environment settings. In Fig. 18, as slippery becomes nonzero, alphazero performance becomes significantly worse than PA-MCTS. This is reasonable as Alphazero selects actions in UCT by selecting actions preferred by those in stationary settings.

D.5 Time Constraints

As in previous environments, we explore the time taken to run PA-MCTS (with α-selection) and compare it with the time to retrain DDQN and Alphazero. We show the results in Table 5. We retrain DDQN and Alphazero that were previously trained in the stationary environment setting. For alphazero network, we create 60 data-generating processes in parallel to feed into its re-training procedure. For DDQN, we use a maximum of 100,000 steps to stop the training. As we did previously, we run α-selection in parallel, and running PA-MCTS is significantly faster than retraining the networks.

D.6 Hardware

The experiments were conducted on Chameleon testbed [11] on 4 Linux systems with 32-96 logical processors and 528 GB RAM.

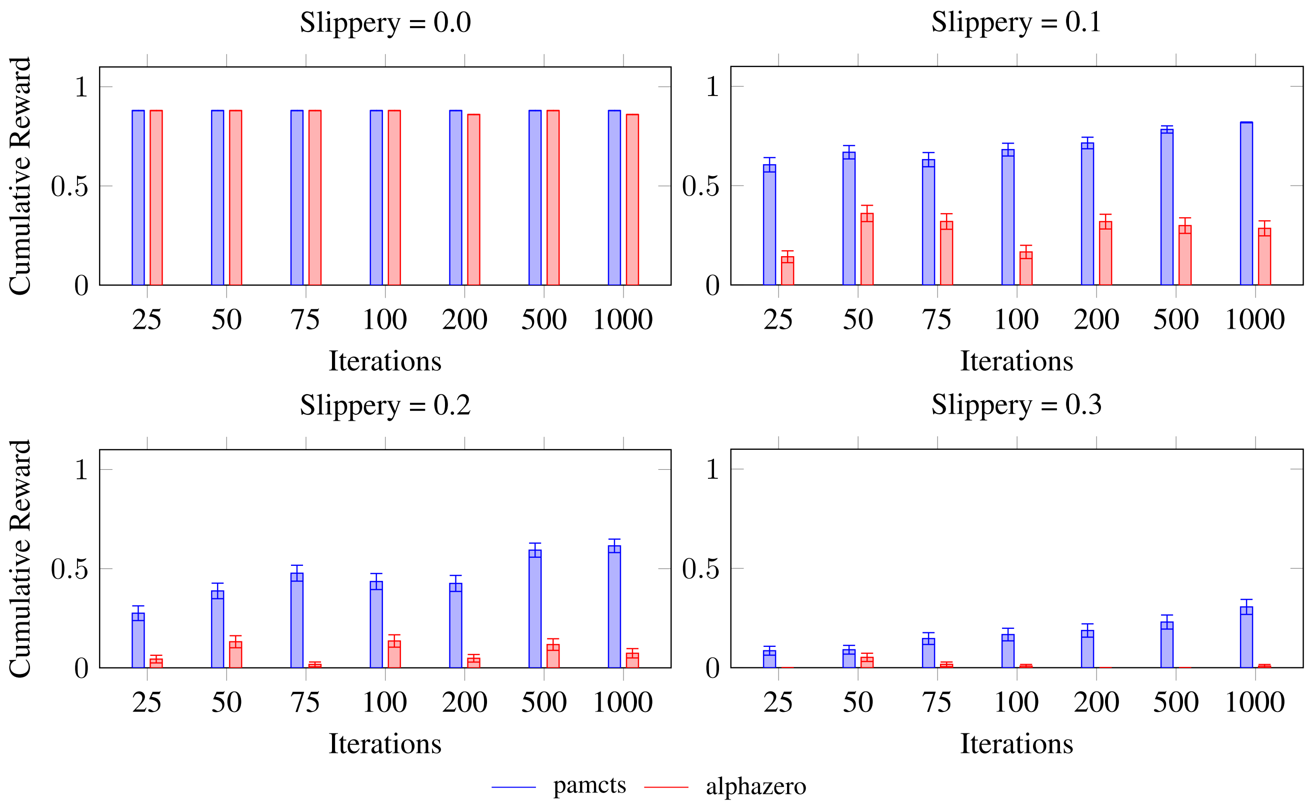

Figure 18: Comparison of PA-MCTS with Alphazero; we see that PA-MCTS comprehensively outperforms Alphazero in non-stationary settings.

Method	Iterations	Execution Time (in seconds)
PA-MCTS	5	0.019
PA-MCTS	10	0.032
PA-MCTS	15	0.038
PA-MCTS	20	0.392
PA-MCTS	25	0.401
PA-MCTS	50	0.071
PA-MCTS	100	0.113
PA-MCTS	200	0.218
PA-MCTS	500	0.570
PA-MCTS	1000	0.919
Re-Train DDQN	100,000 steps	2916.864
Re-Train AlphaZero	NA	6097.156

Table 5: Execution time of PA-MCTS (with α-selection) with varying MCTS iterations in comparison with DQN and AlphaZero. We observe that PA-MCTS is significantly faster.

E EXPERIMENTAL RESULTS ON THE LUNAR LANDER ENVIRONMENT

We also evaluated our approach on the Lunar Lander Environment from OpenAI Gymnasium [4]. In this setting, the agent must control the lunar module by firing the main engine upwards or the side engines to steer left or right. We use the discrete version of the environment. The state is an 8-dimensional vector: the coordinates of the lander, its linear velocities in its angle, its angular velocity, and two booleans that represent whether each leg is in contact with the ground or not. We change the windpower parameter in order to represent uncertainty in this environment.

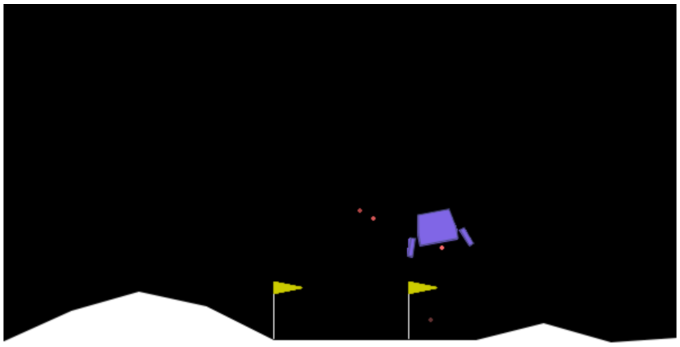

Figure 19: The test environment for the Lunar Lander Environment. The agent must land within a given viewpoint without crashing the body.

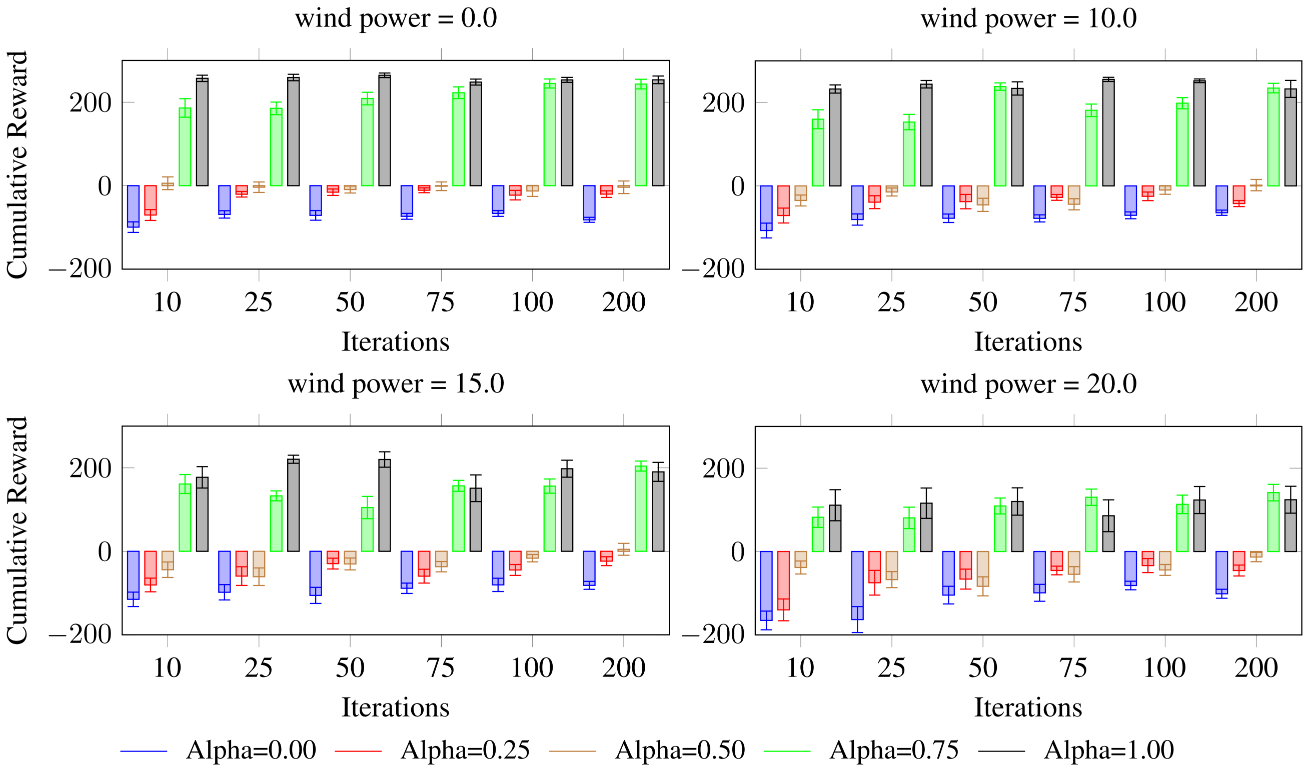

Figure 20: We show the cumulative reward obtained by PA-MCTS, standard MCTS ($\alpha = 0$), and DDQN ($\alpha = 1$) for changing wind power among {0.0, 10.0, 15.0, 20.0}.

E.1 Stationary Environment

The stationary setting in this environment has no wind, i.e., the wind power is set to 0. As Fig. 20 shows, both DDQN and PA-MCTS with higher α value can achieve relatively high cumulative reward. We compute each result by averaging across 30 samples.

E.2 Non-Stationary Environment

We show results for different wind power values in Fig. 20 as well. We use different colors to represent different α values used for PA-MCTS. As the results suggest, it is challenging to use MCTS with a limited number of iterations to find near-optimal actions. This environment serves as the other end of the spectrum (with respect to cliff walking), where MCTS is not particularly useful given limited computational budget. In this case, relying on learned Q-values is the best course of action. Below, we show that PA-MCTS indeed selects a very high α, therefore relying more on the learned Q-values.

E.3 Choosing the optimal α

As we expected, since the limited number of iterations can not fully explore the future trajectory of the agent, PA-MCTS relies more on the predictions from trained DDQN.

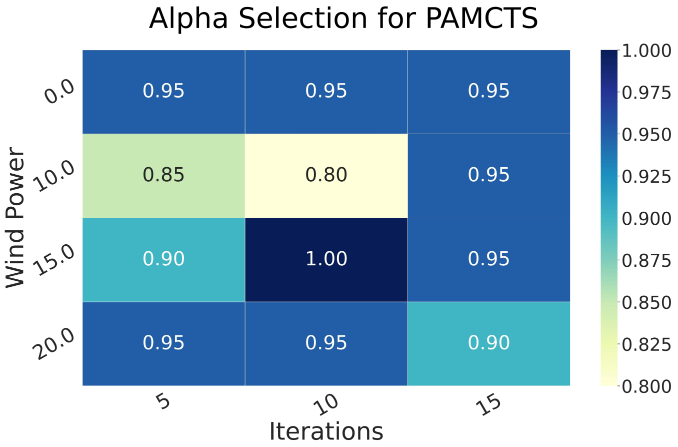

Figure 21: Lunar Lander Alpha Selection for different Wind Power

.

E.4 Comparison with Alphazero

We deployed Alphazero training with 90 parallel running data-generating processes that feed data to the training process for 96 hours. However, in our setting, the AlphaZero network, even with multiple hyperparameters, did not converge and showed worse results than DDQN

E.5 Hardware

The experiments were conducted on Chameleon testbed [11] on 4 Linux systems with 32-96 logical processors and 528 GB RAM.