Action Sampling Strategies in Sampled MuZero for Continuous Control

A JAX-Based Implementation with Evaluation of Sampling Distributions and Progressive Widening

Vaclav Kubon[1]

Supervisor(s): Frans A. Oliehoek [1], **Jinke He**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Vaclav Kubon
Final project course: CSE3000 Research Project
Thesis committee: Frans A. Oliehoek, Jinke He, Michael Weinmann

Abstract

This work investigates the impact of action sampling strategies on the performance of Sampled MuZero, a reinforcement learning algorithm designed for continuous control settings like robotics. In contrast to discrete domains, continuous action spaces require sampling from a proposal distribution β during Monte Carlo Tree Search (MCTS), a process that is underexplored despite being central to the algorithm's effectiveness. We systematically study how performance is influenced by (1) the choice of β distribution and (2) the use of progressive widening, an MCTS augmentation that samples additional actions for frequently visited search tree nodes. Our JAX-based implementation[1] of Sampled MuZero is evaluated on the Brax HalfCheetah environment, testing β as either a uniform distribution or the agent's policy distribution. Additionally, we examine how different progressive widening parameters affect planning depth and computational efficiency. Results show that while temperature modulation provides marginal benefits under specific conditions, progressive widening with properly calibrated parameters can improve planning depth and episode returns.

1 Introduction

Model-based reinforcement learning has garnered considerable interest due to its ability to significantly enhance sample efficiency by enabling an agent to internalize a representation of its environment. Such internal models empower agents to plan multiple steps into the future, reasoning through potential outcomes of their actions without relying on immediate external feedback. This is exemplified by the success of models like AlphaZero [30] in games such as chess and Go, which use Monte Carlo Tree Search (MCTS) [5] to plan by simulating possible action trajectories. However, building such trees requires knowledge of the environment's dynamics, limiting generality. MuZero [26] addresses this by learning the environment's rules jointly with policies and value functions, while still achieving state-of-the-art results.

However, in many challenging real-world tasks, the actions an agent can take are not discrete (such as moving a chess piece to a square) but instead are real-valued and multidimensional—for example, selecting torques to apply to each of a robot's joints. In such continuous control domains, MuZero's standard approach of treating each action as a separate node in the search tree becomes intractable.

Sampled MuZero [17] addresses this limitation by introducing action sampling during tree search. Instead of exhaustively representing all possible actions, it samples a fixed number K of actions from the action space according to a proposal distribution β at each node expansion. Together with other modifications to the MCTS algorithm, this enables computation of an unbiased estimate of the improved policy even in continuous action spaces. While the algorithm supports arbitrary β distributions, the Sampled MuZero paper does not investigate how the choice of β impacts learning and performance in practice, which could leave performance on the table. Furthermore, we propose to enhance the algorithm by making the number of sampled actions adaptive with progressive widening [4], which samples additional actions for frequently visited MCTS nodes. This paper aims to answer two key questions:

1. How does the choice of the proposal distribution β affect learning and performance of Sampled MuZero?
2. Can progressive widening lead to greater depth of exploration and better performance?

To answer these, we implement Sampled MuZero in JAX [10], a high-performance numerical computing library, and evaluate it on continuous control robotics tasks using the JAX-based Brax [9] simulation library. We systematically test various formulations of β. We try sampling actions from either a uniform distribution or from the agent's policy distribution, possibly temperature-modulated to promote additional exploration. Furthermore, we compare agents trained with and without progressive widening. Our goal is to identify sampling strategies that maximize cumulative reward.

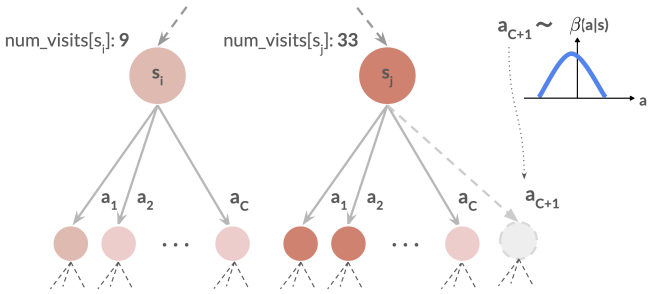


Figure 1: **Illustration of progressive widening in MCTS.** Each node starts with C children and as the number of visits to a node increases (darker red), additional actions are sampled from a proposal distribution $\beta(a|s)$, expanding the search space adaptively.

The paper proceeds as follows. Section 2 provides background on the reinforcement learning setting and the Sampled MuZero algorithm. Section 3 reviews related work in continuous control. Section 4 motivates and describes the chosen action sampling strategies, including progressive widening. Section 5 outlines the experimental setup, and Section 6 presents and discusses the results of our experiments. Section 7 describes the work's limitations and future research directions. Finally, Section 8 discusses the reproducibility and ethical considerations of our work.

2 Background

This section reviews how continuous control differs from discrete reinforcement learning setups and explains how Sampled MuZero extends the base MuZero algorithm.

2.1 RL for Continuous Control

Sampled MuZero [17] considers a standard reinforcement learning setup where an environment is modeled as a *Markov*

[1] Repository: gitlab.tudelft.nl/jinkehe/bachelor-research-project

Decision Process (MDP) with action space $\mathcal{A}$, state space $\mathcal{S}$, and reward function $r : \mathcal{S} \times \mathcal{A} \to R$. We define the return of a state as the sum of discounted future rewards: $G_t = \sum_{i=0}^{\infty} \gamma^i r(s_{t+i}, a_{t+i})$ where $\gamma \in [0, 1]$ is the discount factor. The return is conditioned on the agent's policy function $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, which assigns higher probabilities to actions the agent believes are favorable in a given state, thereby determining the robot's behavior. The goal of the agent is to learn a policy that maximizes the expected return. The MDP can be either *fully observable*, in which case the agent's state contains all necessary information to make optimal decisions, or *partially observable*, where the agent must act based on imperfect information.

In the context of robotics continuous control environments, an action corresponds to selecting torque forces to apply to each of the robot's n joints, so that $a \in R^n$. We can give the agent access to a set of features about its internal state, such as joint velocities or distance from the ground (fully observable setting), or make the agent observe only pixels (partially observable). MuZero was shown to handle both cases, so we focus on the fully observable case, which in practice requires less computational resources [17]. The reward an agent receives depends on the task we are trying to learn. When learning to walk, for instance, we can provide positive reward for moving in the correct direction and negative reward for applying large forces to the joints to promote saving of energy.

In discrete action spaces, the policy function can simply be a categorical distribution, which is easily represented by a neural network having a number of output neurons equal to the number of actions and applying a softmax over their outputs. For high-dimensional continuous actions, however, this approach is not feasible. In practice, two approaches are taken based on *factorization*.

First, we can parameterize the policy by selecting a family of distributions (usually Gaussian) and making the neural network's outputs correspond to the distribution's parameters (mean μ and standard deviation σ). In higher dimensions, we avoid having to predict the entire covariance matrix by factoring the policy—learning a distribution for each dimension independently and obtaining the joint probability as

$$\pi(a|s) = \prod_{i=1}^{n} \pi_i(a_i|s) = \prod_{i=1}^{n} \mathcal{N}_i(a_i; \mu(s), \sigma(s)) \qquad (1)$$

where $\pi_i(a_i|s)$ represents the marginal policy for the i-th action dimension. The second factored policy approach, proposed by [32], involves discretizing the policy into discrete bins, which avoids the restrictions of selecting a family of distributions (e.g., the limitation on the number of distribution modes, which is restricted to one in the Gaussian setting). Both approaches were shown to work with Sampled MuZero [17], but we chose the parameterized Gaussian policy for implementation simplicity.

2.2 Base MuZero

The most common framework for learning a policy is called *policy iteration*, which consists of repeatedly applying two steps. Firstly, we perform *policy evaluation* — learning a value function that estimates the expected return of following the current policy from a state. Secondly, we use the value function for *policy improvement* — learning better policies by increasing the probabilities of actions that lead to higher values. It is useful to think of MuZero as doing policy iteration both during MCTS planning and during the outer loop of acting in the environment.

In MuZero, MCTS is used at each decision step to simulate future trajectories within a learned model of the environment, given by a dynamics function g. Which nodes are expanded is guided by the probabilistic upper confidence tree (PUCT) [29] formula (Equation 2), which balances the prior probabilities from the policy network $P(s, a) = \pi(s, a)$ and the value estimates $Q(s, a)$ from simulated rollouts. This process effectively serves as policy evaluation, estimating the expected return of actions by planning through the model rather than relying on the more costly rollouts in the real environment.

$$\arg\max_a Q(s, a) + c(s) \cdot P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \qquad (2)$$

The perhaps surprising aspect is that after several simulation steps, we obtain a distribution of visit counts at the root node, which forms a more informed policy distribution than the raw policy network. This improved policy, $\mathcal{I}\pi$, is then also used to select actions in the real environment. The improved policy, along with the resulting rewards, is stored in a *replay buffer*. The optimizer samples from this buffer and trains the networks by minimizing the loss:

$$\mathcal{L}(u_t, r_\theta) + \lambda_1 \mathcal{L}(\mathcal{I}\pi_t, \pi_\theta) + \lambda_2 \mathcal{L}(z_t, v_\theta) \qquad (3)$$

Here, u_t is the reward from the environment, and $z_t = \sum_{i=0}^{k-1} \gamma^i u_{t+i} + \gamma^k v_{t+k}$ is the bootstrapped value target. The quantities r, v, and π are outputs of a prediction neural network with trainable weights θ. The dynamics function g_θ is learned as a byproduct of minimizing this loss, even though it is never explicitly supervised to match environment transitions.

2.3 Sampled MuZero

Sampled MuZero [17] extends the base MuZero algorithm with several modifications that allow it to perform unbiased policy iteration and planning in continuous action spaces, even when only a sample of actions is explored.

MCTS Node Expansion. When a node is expanded, instead of considering all $N = |\mathcal{A}|$ actions, we sample a fixed $K \ll N$ actions from a proposal distribution β. In principle, any distribution β can be used. The authors use $\beta = \pi$, which has the effect of sampling actions that are more likely under the current policy. In the experimental section, we test the performance of the algorithm using other distributions.

PUCT Formula. In the original MuZero, the decision of which node to expand is guided by the PUCT formula, using prior probabilities $P(s, a) = \pi(s, a)$, where π is the output of the policy network. However, the authors show that to obtain an unbiased estimate $\hat{\mathcal{I}}\pi$ of the improved policy $\mathcal{I}\pi$ —

as if all actions had been considered — the search must use adjusted prior:

$$P(s,a) = \frac{\hat{\beta}(s,a)\,\pi(s,a)}{\beta(s,a)} \tag{4}$$

Here, $\hat{\beta}(s,a) = \frac{1}{K}\sum_i \delta(a - a_i)$ is the empirical distribution over the K sampled actions, and is non-zero only for actions that were actually sampled. This correction ensures that the improved policy estimate remains unbiased despite sampling only a subset of the continuous action space.

Policy Improvement.

From MCTS, we obtain $\hat{\mathcal{I}}\pi$, a categorical distribution over the K sampled actions, while π_θ denotes a continuous, parameterized policy. To perform policy improvement, we minimize the cross-entropy between the true improved policy $\mathcal{I}\pi$ and π_θ, which we approximate as:

$$\mathcal{L}(\mathcal{I}\pi_t, \pi_\theta) = E_{a\sim\mathcal{I}\pi}\left[-\log \pi_\theta(a)\right] \approx -\sum_{i=1}^{K} \hat{\mathcal{I}}\pi(a_i) \log \pi_\theta(a_i)$$

Since we cannot sample directly from $\mathcal{I}\pi$, we estimate this expectation by reweighting the target policy probabilities output by MCTS using Sampling Importance Resampling (SIR) [25]. Each sampled action $\{a_i\}_{i=1}^{K} \sim \beta$ is assigned a weight $\frac{\pi(a_i)}{\beta(a_i)}$, which is then normalized to form the empirical improved policy $\hat{\mathcal{I}}\pi$ used in the summation above.

3 Related Work

Although this work focuses on comparing variants of Sampled MuZero [17], model-free methods remain important baselines—especially in continuous control domains where learning an accurate model is challenging. These approaches learn policies directly from environment interactions without modeling dynamics, often trading off sample efficiency for simplicity and broad applicability. DDPG [20] exploits the fact that in fully continuous action spaces, the action-value function $Q(s,a)$ is differentiable with respect to the action a, allowing efficient computation of deterministic policy gradients. SAC [11] improves upon this by using a stochastic policy and entropy maximization to encourage exploration and robustness. PPO [28], an on-policy method, stabilizes updates via a clipped surrogate objective and is widely used for its reliability and ease of tuning. MPO [1] and AWR [24] adopt a supervised learning perspective on policy improvement, using KL constraints and advantage-weighted updates to ensure stable and efficient learning.

Before Sampled MuZero, several model-based approaches attempted to adapt MCTS-style planning to continuous control [22; 12]. While they demonstrated feasibility in principle, these methods only outperformed model-free baselines on low-dimensional tasks such as the 1D inverted pendulum. The authors of Sampled MuZero compare their method to the Dreamer family of models [13; 14], which, instead of using tree search, rely on a recurrent world model to simulate future trajectories entirely in latent space and apply policy gradients to maximize expected return. Because Dreamer avoids explicit search over actions, it handles continuous and high-dimensional action spaces particularly well. However, its limitations compared to Sampled MuZero include greater sensitivity to model errors, limited interpretability, and currently lower performance than recent MuZero variants [33]. Sampled MuZero itself is one of several extensions of the original algorithm, each targeting a specific limitation—for example, MuZero Unplugged [27] for offline RL, EfficientZero [35] for improved sample efficiency, and Gumbel MuZero [6] for removing reliance on UCB heuristics in MCTS. Although in principle higher performance could be achieved by combining ideas from these variants, we use the base Sampled MuZero to focus purely on relative performance differences.

Sparse sampling algorithms, such as the one introduced by [19], aim to efficiently plan in large Markov Decision Processes (MDPs) by sampling a limited number of trajectories. While effective in complex discrete settings, these methods require evaluating all possible actions from each state, making them impractical for continuous action spaces.

Continuous Upper Confidence Trees [3] introduced progressive widening, a technique that gradually increases the number of actions considered at each MCTS node based on its visit counts. Progressive widening has been used in some MuZero-based algorithms such as [31], which applies a variant of progressive widening that tries to consider a more diverse set of actions by merging similar actions using a Voronoi-based abstraction [21]. Another related method, BetaZero [23], uses progressive widening in belief-state planning and demonstrates its effectiveness in stochastic, partially observable environments. However, none of these approaches systematically benchmark progressive widening in continuous action spaces, leaving open questions about its interaction with action sampling in Sampled MuZero.

4 Action Sampling Strategies

While the original Sampled MuZero paper proposes a general framework for adapting MuZero to continuous action spaces, it leaves largely unexplored the core mechanism that differentiates it from the discrete version: the sampling of actions from a continuous space. Specifically, two interesting design questions remain open: (1) *how* actions should be sampled (i.e., from what distribution) and (2) *how many* should be sampled at each node during tree search.

For the first question, the original paper only investigates a single strategy, setting the sampling distribution β equal to the current policy π. In this work, we explore alternative formulations for β, aiming to test whether other distributions might yield better performance.

For the second question, the original paper evaluates fixed values of K, i.e., a constant number of actions sampled per node. In contrast, we examine whether progressive widening [4], a technique that allows the number of sampled actions to grow dynamically with node visitation, can lead to more effective planning in continuous domains.

This section introduces the conceptual background behind these strategies and describes our methodology for implementing them. We benchmark the strategies in Section 6.

4.1 Proposal Distributions β

The Sampled MuZero framework is built for an arbitrary action proposal distribution β, but [17] exclusively uses $\beta = \pi$ without testing other options.

Uniform Distribution. First, we propose setting $\beta = U(-1, 1)$, meaning sampling the action space uniformly. With a probability density function of $\beta(a) = 1/K$, this cancels out with $\hat{\beta}$ in the prior in the PUCT formula (Equation 2), which simplifies to $P = \pi$. This is the same prior used in the base MuZero. With this setting, the bias towards exploration of promising actions according to the current policy would not happen during sampling but during MCTS.

Temperature-modulated Policy Distribution. Secondly, the authors mention the possibility of setting $\beta = \pi^{1/\tau}$, i.e., temperature-modulating the policy with temperature hyperparameter τ. The prior in the PUCT can then be written as $P = \hat{\beta}\pi^{1-1/\tau}$. If $\tau = 1$, then $\beta = \pi$, equaling the setup used in the original paper [17] and resulting in a uniform prior $P = \hat{\beta}$. This means the sampling of actions is guided by π, but then the policy does not play a role in determining which actions to search first in MCTS—the reverse of the case when β is a uniform distribution.

We experiment with different values for τ. Since we chose to use a factored Gaussian policy, we temperature-modulate the parameters of the policy distribution as

$$\pi_\theta^{1/\tau} = \prod_{i=1}^{n} \mathcal{N}_i(\mu_\theta, \sqrt{\tau} \cdot \sigma_\theta) \tag{5}$$

If $\tau > 1$, the MCTS search will evaluate more diverse samples but is guided by more peaked probabilities $\hat{\beta}\pi^{1-1/\tau}$. If $\tau < 1$, the opposite is true.

A motivation for why temperature modulation could be promising is that it could be a substitute for the Dirichlet noise, which in the base Muzero [26] is added to the probabilities of the root actions to promote additional exploration. We can't apply noise to π in Sampled MuZero because of the PUCT prior formula, where π and, therefore, the applied noise too, can cancel out with the logits β when using $\beta = \pi$.

4.2 Progressive Widening

Progressive widening [4] is an augmentation for MCTS that allows the tree to dynamically adjust the number of node children considered, enabling it to grow arbitrarily wide. During the selection step in default MCTS, we descend the tree starting at the root node by recursively performing action selection, i.e., choosing among the K node children with the PUCT formula. In MCTS with progressive widening, instead of selecting among K actions, the number of actions we consider depends on the number of visits the parent node has received, as illustrated in Figure 1. This aims to provide a more nuanced exploration of promising tree paths. More concretely, we show the modified action selection procedure in Algorithm 1. On line 6, we see that when the algorithm samples additional actions and widens the tree is controlled by hyperparameters C and α. C corresponds to the base number of actions we consider during the first node visit. The higher the parameter α, the fewer node visits it takes to sample additional actions, widening the tree faster. To illustrate how these parameters influence the progressive widening algorithm, Figure 3 plots the number of child nodes considered as the node's visit count increases. In the extremes, when $\alpha = 1$, we sample additional actions every time a node is visited, and when $\alpha = 0$, no additional actions beyond C can be sampled, corresponding to default MCTS. This is shown for various α values, which are later used in our experiments.

Algorithm 1 MCTS with Progressive Widening

```
1:  procedure SELECT_ACTION(s)
2:      if LEAF_NODE(s) then
3:          return s
4:      end if
5:      t ← num_visits[s]
6:      if num_children[s] < C t^α then
7:          a' ~ β(s)
8:          s' ← g_θ(s, a')
9:          children[s].add(s')
10:     end if
11:     SELECT_ACTION(arg max_{s' ∈ children[s]} PUCT(s'))
12: end procedure
```

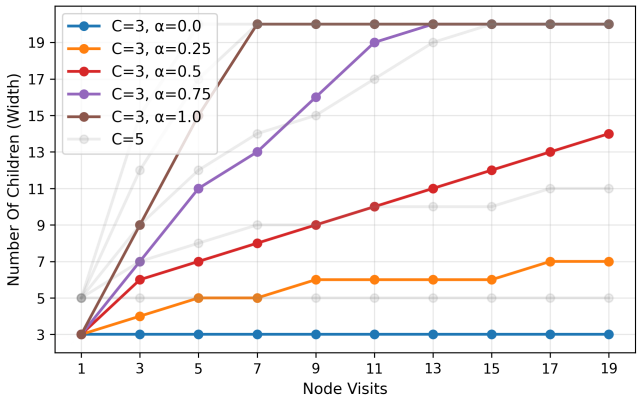


Figure 2: **Effect of C and α parameters on progressive widening.** C corresponds to the base number of actions we consider. Higher α values mean the algorithm requires fewer node visits to sample additional actions and making the MCTS tree branch wider.

We chose to try progressive widening following our observation that the search trees built by Sampled MuZero are relatively shallow. This likely occurs because when $\beta = \pi$ (as in the original Sampled MuZero paper), the prior P becomes uniform. Therefore, there is no prior knowledge to guide MCTS in ignoring poor actions early and instead allocating the simulation budget to more promising actions for deeper exploration. Since progressive widening does not expand all K actions at once, the algorithm has fewer choices at each level, which may enable deeper tree construction. Subsequently, it can widen these levels for the most relevant actions using the refined Q values in the PUCT formula.

5 Experimental Setup

The following section details the specific settings used for our training environment and the Sampled MuZero agent, which serve as a common base throughout all experiments.

5.1 Training Envrionment

Several popular physics engines are available for testing continuous control agents [18]. We chose Brax [9] as our framework, primarily because it is the only physics engine written in JAX. This choice complements our Sampled MuZero implementation, which is also in JAX, allowing for greater computational performance. The completely JAX-based setup proved highly efficient, enabling us to run 128 environments in parallel and significantly accelerating training.

However, computational constraints limited our experiments. Training was restricted to university cluster DAIC [8], which prioritizes jobs by time allocation. Since continuous control requires orders of magnitude more steps than discrete benchmarks like Atari, and while the original Sampled MuZero allows up to 20 million steps, we limited training to 1 million steps (approximately 15 hours), resulting in longer resource queues.

Given these constraints, we needed an environment showing learning progress within this shorter duration. We selected HalfCheetah [34], featuring a two-legged robot with 9 links and 6 controllable joints. The objective is to apply torque for maximum forward speed, with rewards for forward distance and penalties for backward movement. We used the fully observable variant for faster learning [17], providing the agent a 17-dimensional observation vector (link angles, center of mass locations, joint velocities) to output 6-dimensional actions.

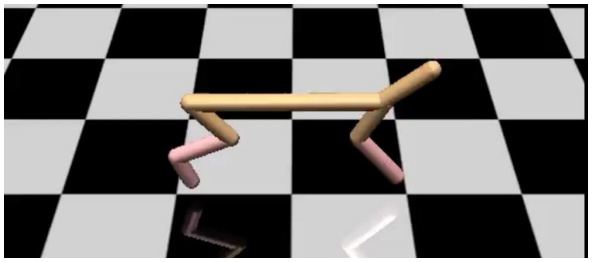


Figure 3: **HalfCheetah training envrionment.** We train our agent to control a 2D bipedal robot, receiving positive rewards for forward movement while minimizing energy consumption.

5.2 Agent Architecture

As a starting point, we were provided with a repository implementing base MuZero for Atari, which uses DeepMind's mctx library [7] to perform MCTS search. We extended the mctx library to handle multidimensional actions and implemented new policy and action selection functions according to the Sampled MuZero paper [17]. Following the Sampled MuZero paper specifications, we implemented additional network architectures and modified the code to work with vector inputs rather than just images used in Atari, allowing users to switch between input types. This work provides the first scalable open-source JAX implementation of Sampled MuZero, representing another contribution of our research.

Following Sampled MuZero, the representation and dynamics functions are processed by a ResNet v2 [15] style pre-activation residual tower, with each block containing two fully connected layers with leaky ReLU activations and layer normalization [2]. We used a smaller network with 4 blocks and a hidden dimension of 512.

The policy network uses a factored Gaussian distribution. To keep sampled actions within the action bounds (-1, 1), we initially tried clipping, which accumulates density near the cutoff points and results in incorrect log probabilities and poor performance. In the end, we used the tanh function to squash the actions and used the change of variable formula to compute action log likelihood following [11]. To improve stability, we use entropy regularization in the policy loss with a coefficient of $5 \cdot 10^{-3}$.

We train with $K = 10$ sampled actions and a search budget of 50 simulations per move, as the authors found that performance gains become minimal beyond this point in practice. We use the Adam optimizer with a batch size of 256, initial learning rate of 0.003, and weight decay of $2 \cdot 10^{-5}$. We employ n-step bootstrapping with n=5 and a discount factor of 0.99, following [16]. Like MuZero, we use a categorical representation of reward and value predictions with 51 bins. All hyperparameter settings for the agent and the environment are listed in Appendix A.

6 Results

In this section, we describe our experimental design, present our results and discuss their implications.

6.1 Comparison of Proposal Distributions β

Firstly, we compare the performance measured by the agent's mean episode return across the different sampling distributions. As explained in Section 4.1, we choose to test $\beta = U(-1, 1)$, which makes MCTS use a prior from the base MuZero. Furthermore, we tested using temperature-modulated policy $\pi^{1/\tau}$ as β. We test $\tau \in \{0.7, 0.85, 1.15, 1.3\}$ and compare it with $\tau = 1$, which is the setting from the Sampled MuZero paper that provides no additional exploration. $\tau > 1$ should promote exploration during action sampling and $\tau < 1$ should reduce it.

Figure 5 shows the effect the temperature parameter has on the entropy of the policy predicted by the agent's network. As expected, for $\tau > 1$, the policy entropy increases, meaning we give a chance to a wider range of actions. Interestingly, the temperature does not seem to just scale the standard deviation but rather change the overall trend of how the policy evolves over time. For $\tau < 1$, it seems to promote faster policy convergence by assigning high probabilities to a smaller range of actions. When we sample actions uniformly, the entropy of the policy appears to stay constant, which could mean the uniformly selected actions might not provide enough learning signal for the policy to change. This would also explain this strategy's poor performance, as discussed later.

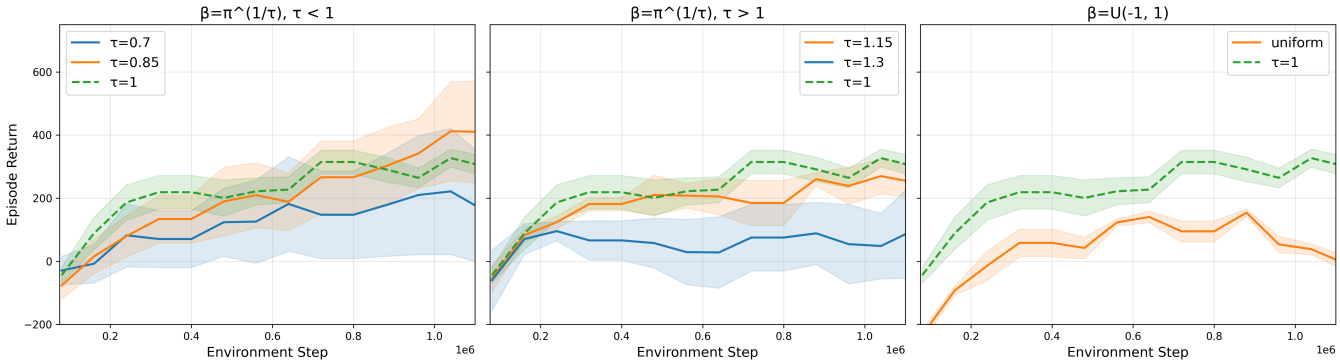


Figure 4: **Performance of different proposal distributions β.** We compare uniform sampling and temperature-modulated sampling, with $\tau = 1$ corresponding to baseline. Only moderate temperature adjustment ($\tau = 0.85$) matches baseline performance. Results show the mean and standard error of 5 seeds per experiment.

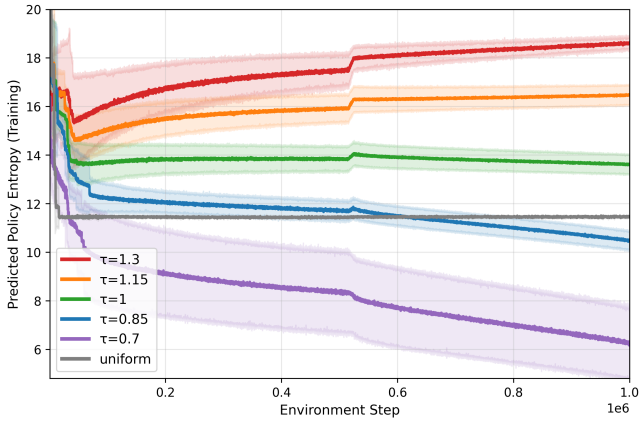


Figure 5: **Policy network entropy across β distributions**. Higher temperatures ($\tau > 1$) maintain elevated entropy throughout training, promoting exploration of diverse actions. Lower temperatures ($\tau < 1$) lead to rapid entropy decay, indicating faster convergence to peaked action distributions. Uniform sampling maintains constant entropy. Shaded regions show standard error over 5 seeds.

In Figure 4, we plot the achieved episode return during evaluation for each β option. We can see that no distribution clearly beats the baseline on the first million environmental steps. Only $\tau = 0.85$, which reduces exploration during sampling and increases it during MCTS, matched the baseline performance. This is consistent with [26], which also increases exploration during MCTS but through adding Dirichlet noise to the root priors. However, we can see that $\tau > 1$ settings have much larger variance than others, so clearer results might need more computational resources. We can also see that moving further in either direction from $\tau = 1$ (from 1.15 to 1.3 or from 0.85 to 0.7) worsens the results, showing that too much added noise prevents the algorithm from learning. The results for the uniform distribution show it does not learn at all, proving the need for the modified Sampled MuZero we introduced.

We then tested two additional ideas for making the temperature modulation work better, but they did not result in improvement. Firstly, it is possible that different temperatures might be needed at different points in training. For instance, when using $\pi = \beta$, at the start of training, the policy might be poor and overly narrow—failing to explore potentially useful actions. Introducing noise early might reduce reliance on the initial policy's quality, but later in training, it might have a negative effect. To test this, we tried creating a time-dependent schedule, starting at high noise of $\tau = 1.3$ at the start and lowering the temperature by 0.08 after 125K steps. Secondly, we applied temperature modulation to the β probabilities not only at the root of the MCTS tree (as is done with Dirichlet noise in standard MuZero), but to all actions throughout the tree. Neither of the modifications managed to beat their respective baseline. We included the plotted results in Appendix B.

In summary, this section demonstrates that while $\beta = \pi^{1/\tau}$ shows some promise, particularly with $\tau = 0.85$, which reduces action sampling exploration, the benefits are marginal and come with increased variance. The experiments reveal that excessive modulation in either direction harms performance, and that the uniform distribution completely fails to learn, validating the necessity of the modified Sampled MuZero approach.

6.2 Effect of Progressive Widening

In Section 4.2, we explained how progressive widening has the potential to improve performance by making the search tree deeper when necessary. To evaluate our progressive widening approach, we test configurations with $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$. After experimenting with different settings for the base number of children C we set it to 3, although the performance impact of this parameter was small (we plot these results in Appendix C). We also limit the maximum number of children per node to 20 for every experiment. As a baseline, we include an agent that can search all actions from the start (i.e., it uses no progressive widening with $K = 20$).

Figure 6 partially validates our hypothesis that lower α values result in deeper search trees on average. However, the difference becomes negligible for α values above 0.25, where the depth approaches that of the baseline.

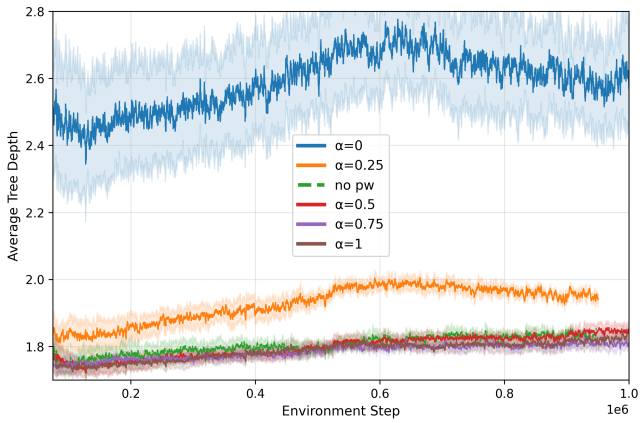


Figure 6: **Average MCTS tree depth across different α settings.** Progressive widening with lower α values produces deeper trees by restricting initial action width, while higher α values converge to depths similar to the no progressive widening baseline. The effect becomes negligible for $\alpha \geq 0.5$

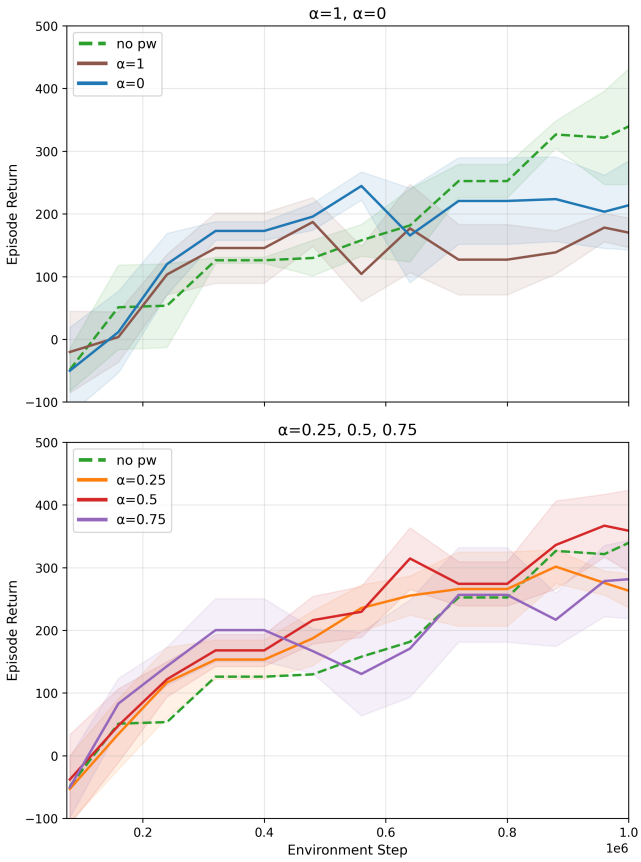


Figure 7: **Episode return across different α settings.** Shaded regions show standard error over 5 random seeds. The dotted line shows the baseline without progressive widening ($K = 20$). Progressive widening with $\alpha = 0.5$ achieves the best performance, outperforming both extreme values and the baseline.

Figure 7 shows the episode returns achieved during evaluation. At the extreme values ($\alpha = 0$ and $\alpha = 1$), progressive widening performs the worst. For $\alpha = 0$, this is likely due to MCTS being stuck with evaluating just $C = 3$ actions at each level, which as experiments in [17] indicate is too narrow of an action selection to choose good actions from. The best results are achieved with $\alpha = 0.5$, which, as illustrated in Figure 3, samples a new action approximately every two node visits. This seems to strike a good balance, not making the width too narrow or adding new nodes too quickly without having the time to evaluate them. This configuration also narrowly outperforms the baseline after the first million environmental steps, providing evidence that progressive widening could improve Sampled MuZero's performance.

We further attempted to improve performance in two ways. The above experiments used a fixed budget of 50 MCTS simulations, following the original Sampled MuZero implementation (where a new node is added during each simulation). We tested performance with 100 simulations to determine whether progressive widening's increased flexibility could better utilize the larger simulation budget. We found that progressive widening offered no improvement beyond what the increased simulation budget alone provided.

Second, we hypothesized that the lack of depth increase resulted from Sampled MuZero exploring actions too uniformly, failing to focus on any single branch and exploit progressive widening's benefits. Therefore, we combined progressive widening with temperature modulation using $\tau > 1$, which should result in more peaked probabilities of promising actions during MCTS. However, this approach worsened performance. Detailed results are presented in Appendix C.

Finally, we argue that progressive widening could improve training efficiency. When expanding a node in Sampled MuZero, all K actions (typically 5-20) must be sampled, even if the node or its children are never revisited, which is computationally wasteful. Figure 8 shows that while progressive widening did not yield dramatic performance improvements in episode return, it requires MCTS to sample significantly fewer actions than even the minimal baseline setting of $K = 5$. For instance, MCTS with $\alpha = 0.25$ performs 30% fewer action samplings.

All in all, these experiments demonstrate that progressive widening can improve Sampled MuZero's performance when properly configured. The results show that $\alpha = 0.5$ strikes a good balance between search depth and breadth, outperforming both extreme values and the baseline. While the performance improvements are modest, the computational efficiency gains suggest that progressive widening is a promising technique, though benefits are sensitive to parameterization.

7 Conclusion, Limitations and Future Work

This work addressed a research gap in Sampled MuZero's core action-sampling algorithm. We motivated and tested variations in (1) the proposal distribution β from which actions are sampled and (2) the MCTS algorithm, allowing flexible action sampling during node expansion with progressive widening. For the proposal distribution, we found that adding noise through temperature-modulated policies does not improve performance. For progressive widening, we nar-

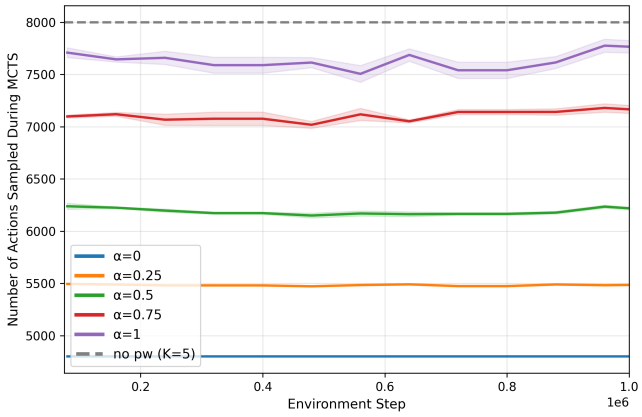


Figure 8: **Action sampling reduction with progressive widening.** Lower α values result in fewer sampled actions during MCTS compared to the baseline without progressive widening ($K = 5$).

rowly outperformed the baseline with proper α parameterization and showed potential for increased training efficiency. Additionally, an important contribution is our open-source JAX-based implementation—the first of its kind for this algorithm—which complements simulation libraries like Brax.

Limitations. Our conclusions are limited by experimental constraints. We tested performance only during the first million environmental steps, potentially before algorithm convergence. More random seeds would reduce variance and strengthen conclusions. We tested only a single, simple environment, limiting generalizability to higher-dimensional action spaces like Humanoid or Spot. We also cannot rule out that better performance could be achieved by further tuning the hyperparameters (e.g., initial number of node children C or temperature τ).

Future Work. Future work could explore several directions. First, investigating temperature modulation effects on alternative policy representations, such as the discretized policy [32]. Second, progressive widening could be analysed more extensively by for instance, studying the distribution of individual branch depths and how it evolves instead of just looking at the average tree depth. Third, Voronoi abstraction could be used to improve progressive widening by avoiding sampling similar actions as in [21]. Lastly, the double progressive widening [4] could be used to extend Sampled MuZero to stochastic settings.

8 Responsible Research

Reproducibility. We believe we provide all the necessary information to reproduce the research in its entirety and validate it. We have published the codebase in a publicly accessible repository. A description of the used architecture in Section 5 further aids in reproducing the research. We also include a complete list of all parameters used during experimentation in Appendix A. For each hyperparameter setting in an experiment, we run several seeds with randomized initial weights to ensure the generalizability of our conclusions, and we plot the standard error to reflect the variance in results. We are also transparent about the limitations of our research in Section 7, which allows future work to build on our findings. Furthermore, to the best of our knowledge, we conducted a thorough review of related literature, properly citing all sources and avoiding any misrepresentation of prior work.

Ethics. The field of continuous control and robotics warrants consideration of the broader ethical and societal impacts of this technology, such as safety or job automation. However, we believe this work does not pose any tangible impact in this regard, as we are adapting existing experimental setups, testing agents only in simulation, and conducting experiments at a relatively small scale. The limited scope of our experiments—with GPU training times under a day—also minimizes the environmental impact of our research.

Use of Generative AI. During the writing of this paper, we used the Claude Sonnet 4 model to help with proofreading and correcting of grammar. Additionally, we used it to help with generating code for plotting our experimental data using Python's matplotlib library.

References

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation, 2018.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[3] Adrien Couetoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION'11)*, Italy, 2011. HAL Id: hal-00542673v1, https://hal.science/hal-00542673v1.

[4] André Couëtoux, Jean-Baptiste Hoock, Nikoleta Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 433–445. Springer, Berlin, Heidelberg, 2011.

[5] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. volume 4630, 05 2006.

[6] Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022.

[7] DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, and Antoine Dedieu. The DeepMind JAX Ecosystem, 2020.

[8] Delft AI Cluster (DAIC). The delft ai cluster (daic), rrid:scr_025091, 2024.

[9] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation, 2021.

[10] Roy Frostig, Matthew Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. 2018.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[12] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

[13] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2022.

[14] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[16] Matthew W. Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, and Danila Sinopalnikov. Acme: A research framework for distributed reinforcement learning, 2022.

[17] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces, 2021.

[18] Michael Kaup, Cornelius Wolff, Hyerim Hwang, Julius Mayer, and Elia Bruni. A review of nine physics engines for reinforcement learning research, 2024.

[19] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2–3):193–208, 2002.

[20] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[21] Michael H. Lim, Claire J. Tomlin, and Zachary N. Sunberg. Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation pomdps, 2021.

[22] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. A0c: Alpha zero in continuous action space, 2018.

[23] Robert J. Moss, Anthony Corso, Jef Caers, and Mykel J. Kochenderfer. Betazero: Belief-state planning for long-horizon pomdps using learned approximations, 2024.

[24] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019.

[25] Donald B. Rubin. The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398):543–546, 1987.

[26] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, and Demis Hassabis. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, December 2020.

[27] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model, 2021.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[29] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[30] 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. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[31] Samuel Sokota, Caleb Ho, Zaheen Farraz Ahmad, and J Zico Kolter. Monte carlo tree search with iteratively refining state abstractions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[32] Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization, 2020.

[33] Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. Efficientzero v2: Mastering discrete and continuous control with limited data, 2024.

[34] Paweł Wawrzyński. A cat-like robot real-time learning to run. In Mikko Kolehmainen, Pekka Toivanen, and Bartlomiej Beliczynski, editors, *Adaptive and Natural Computing Algorithms*, pages 380–390, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

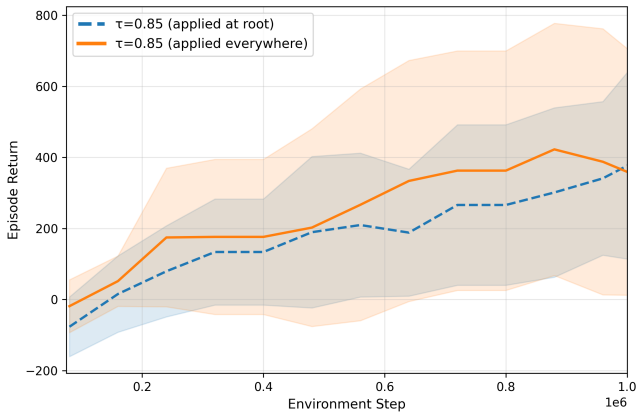
[35] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data, 2021.

A Hyperparameters for HalfCheetah Environment

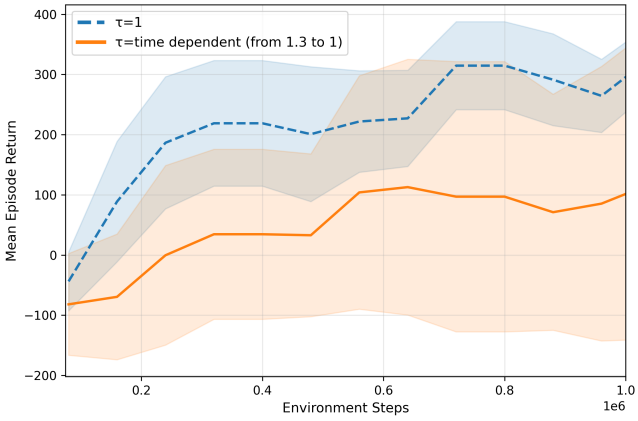
Table 1: Key Hyperparameters Used in Sampled MuZero Training. The architecture, optimizer and MCTS settings are taken from [17].

Category	Hyperparameter Setting
Training	Total environment steps: 1,000,000
	Learn per update step: 1
	Evaluation frequency: every 1250 updates
	Save model every 25,000 updates
	Random seeds: 0, 42, 123, 1, 43
Environment	Environment: Brax HalfCheetah
	Episode length: 1000 steps
	Training envs: 64 (async), Eval envs: 32 (async)
Action Sampling	Action sample size: 10
MCTS (Training)	Simulations: 50
	Temperature schedule: annealed over 4 phases
	Dirichlet noise: $\alpha = 0.3$, fraction = 0.25
	PUCT constants: $c_{\text{init}} = 1.25$, $c_{\text{base}} = 19652$
Network Architecture	ResNet blocks: 4 per module
	Hidden units per layer: 512
	Value transformation: 51 bins over [-150, 150]
Optimization	Optimizer: AdamW
	Learning rate: 0.003, weight decay: 0.0002
	Batch size: 256, max grad norm: 10.0
	Warmup ratio: 10%
Replay Buffer	Size: 524,288, Min size to learn: 8192
	Prioritized replay: $\alpha = 0.6$, $\beta \in [0.4, 1.0]$
Loss Weights	Reward loss coef: 1.0
	Policy loss coef: 1.0, Entropy: 0.005
	Value loss coef: 0.25, TD steps: 5
	Unroll steps: 5

B Additional Experiments for Temperature Modulated Policy β

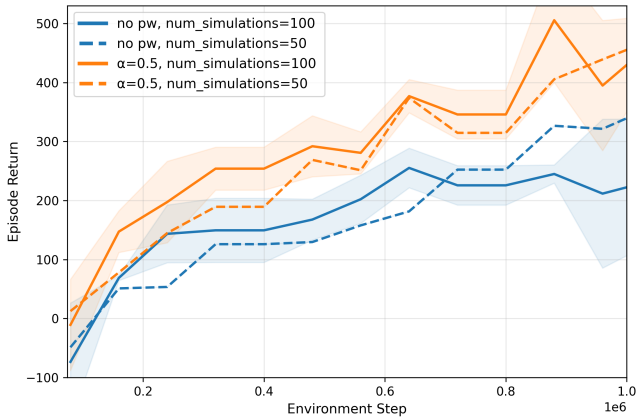


(a) **Noise Applied to Root vs to All Actions.** Episode returns from 5 random seeds comparing temperature modulation applied only to β probabilities of root actions versus applying it to all actions in the entire MCTS tree. The minimal improvement observed is likely due to increased variance from introducing additional noise throughout the search.

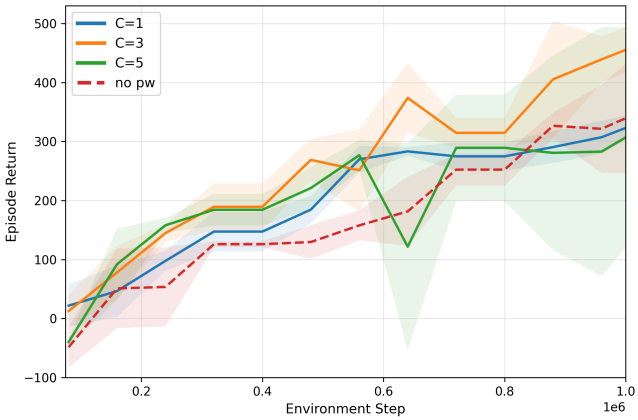


(b) **Time-dependent Temperature Modulation.** Episode returns from 5 random seeds comparing a time-dependent temperature schedule ($\tau = 1.3$ (max noise) to 1.0 (no noise), -0.08 every 125K steps) against baseline. The schedule degrades performance and increases variance, likely due to fine-tuning difficulties and policy instability from frequent temperature changes.

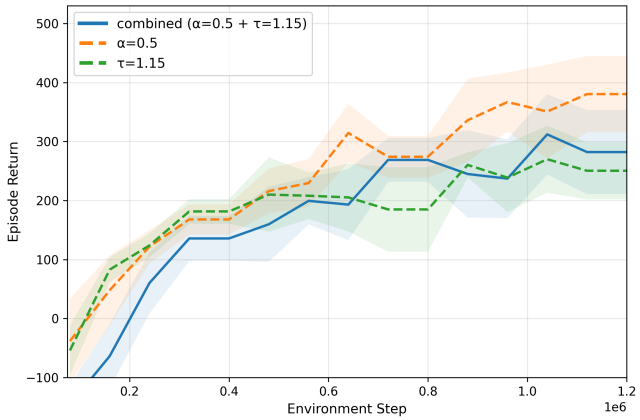
C Additional Experiments for Progressive Widening



(a) **Progressive widening and simulation budget.** Comparison of progressive widening performance with MCTS simulation budgets of 50 (baseline) and 100 (3 seeds each). Increasing the simulation budget with progressive widening yields identical performance gains as without progressive widening, indicating progressive widening does not improve utilization of additional simulations.



(b) **Effect of initial width C on progressive widening performance.** Results averaged across 3 seeds show performance peaks at $C = 3$, with larger initial widths providing no additional benefit.



(c) **Combination of progressive widening with temperature-modulated policy β.** We hypothesized that using $\tau > 1$ to create more peaked action probabilities would guide MCTS to focus exploration on promising branches, thereby increasing search tree depth and improving performance. However, results demonstrate that this combination of progressive widening with temperature modulation actually worsened performance compared to baseline approaches. Shaded regions represent standard error across 5 random seeds.