# SMX: Sequential Monte Carlo Planning for Expert Iteration

**Edan Toledo** [*1] **Matthew V Macfarlane** [*†1 2]
**Donal Byrne** [1] **Siddarth Singh** [1] **Paul Duckworth** [1] **Alexandre Laterre** [1]

## Abstract

Developing agents that can leverage planning abilities during their decision and learning processes is critical to the advancement of Artificial Intelligence. Recent works have demonstrated the effectiveness of combining tree-based search methods and self-play learning mechanisms. Yet, these methods typically face scaling challenges due to the sequential nature of their search. While practical engineering solutions can partly overcome this, they still demand extensive computational resources, which hinders their applicability. In this paper, we introduce SMX, a model-based planning algorithm that utilises scalable Sequential Monte Carlo methods to create an effective self-learning mechanism. Grounded in the theoretical framework of control as inference, SMX benefits from robust theoretical underpinnings. Its sampling-based search approach makes it adaptable to environments with both discrete and continuous action spaces. Furthermore, SMX allows for high parallelisation and can run on hardware accelerators to optimise computing efficiency. SMX demonstrates a statistically significant improvement in performance compared to AlphaZero, as well as demonstrating its performance as an improvement operator for a model-free policy, matching or exceeding top model-free methods across both continuous and discrete environments.

## 1. Introduction

The quest to imbue machines with human-like intelligence has led to the advent of AI capable of mastering complex decision-making processes. Specifically, the development of sequential decision-making in AI-based systems has produced significant breakthroughs ranging from strategic game-playing to practical, real-time control systems (Mnih et al., 2015; Christiano et al., 2017; Silver et al., 2017; Vinyals et al., 2019; Degrave et al., 2022).

A primary factor in these advancements is the integration of neural networks with planning and Reinforcement Learning (RL) methods. Such systems display sophisticated decision-making, as seen in applications such as chess (Silver et al., 2018; Schrittwieser et al., 2020), matrix multiplication (Fawzi et al., 2022), as well as in the domains of combinatorial optimization and language modelling (Kool et al., 2018; Hottung et al., 2021; Wei et al., 2022; Yao et al., 2023; Chalumeau et al., 2023). Methods like Expert Iteration have been shown to effectively leverage planning not just at test time but during training using iterative imitation learning (Anthony et al., 2017; Silver et al., 2018; Schrittwieser et al., 2020; Danihelka et al., 2021; Antonoglou et al., 2021; Fawzi et al., 2022). This family of approaches uses planning to create expert targets for model-free policies, resulting in a self-learning mechanism that continuously improves the performance of the search and model-free policy. However, popular planning-based policy improvement operators like Neural MCTS face challenges in low-budget planning and adaptability to large or continuous action spaces. MCTS is also sequential by nature which limits scalability, making execution expensive. This has motivated a large corpus of variations to tackle these challenges (Moerland et al., 2018; Hamrick et al., 2019; Danihelka et al., 2021; Yang et al., 2020; Hubert et al., 2021). These limitations underscore the need for scalable and general algorithms to make such methods more widely applicable.

In this context, we introduce SMX, a model-based planning algorithm utilising scalable Sequential Monte Carlo (SMC) methods. SMX, grounded in the *control as inference* theoretical framework, offers a novel self-learning mechanism applicable to both discrete and continuous action space environments. Its sampling-based approach is inherently parallelisable and therefore, can efficiently make use of hardware accelerators. It also does not require the full search tree to be maintained reducing memory costs.

SMX offers four key contributions to the field of RL:

[*]Equal contribution , [†]Work done during internship [1]InstaDeep, London, United Kingdom [2]University of Amsterdam. Correspondence to: Edan Toledo <e.toledo@instadeep.com>, Matthew V Macfarlane <m.v.macfarlane@uva.nl>.

*Workshop on Foundations of Reinforcement Learning and Control at the $41^{st}$ International Conference on Machine Learning*, Vienna, Austria. 

1. We show that the proposed SMX algorithm can act as an efficient and high-performing policy improvement operator, which outperforms established AlphaZero (Silver et al., 2018) and model-free methods on selected baselines.

2. SMX can be applied to environments with both continuous and discrete action spaces, without needing specific adjustments. It consistently delivers high performance in both settings, making it a widely applicable method across domains.

3. SMX presents favourable scaling behaviours, showing improved performance with additional search budgets at inference time.

4. Finally, SMX benefits from the parallelisable nature of its sampling-based search approach, resulting in faster wall-clock times for equivalent search budgets. This is a significant advantage over the sequential process of MCTS-based methods (Coulom, 2006), which is particularly costly if search is used during training.

## 2. Background

### 2.1. Reinforcement Learning

Reinforcement Learning is often formalised within the Markov Decision Processes (MDP) framework. An MDP models the environment using a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma, \mu)$. In this framework, an actor takes a series of actions in an environment with the goal of maximising the expected discounted return. Here, $\mathcal{S}$ is the set of possible states an agent can visit, $\mathcal{A}$ is the set of actions an agent can take, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the state transition probability function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, and $\mu$ is the initial state distribution. An agent's behaviour can then be modeled using a policy, which is a distribution over actions given a state $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. The agent aims to find a policy $\pi^*$ that maximises the expected discounted return:

$$\pi^* \in \arg\max_{\pi \in \Pi} \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right], \tag{1}$$

where $r_t$ is the reward received at time $t$ and $\Pi$ is the set of realisable policies. A key concept in RL is the value function $v^\pi(s_t) = \mathbb{E}_\pi \left[ \sum_{t=t}^{\infty} \gamma^t r_t | s_t \right]$, which maps $s_t$ the expected discounted sum of future rewards from a particular state, given actions sampled according to $\pi$. We refer to the value function using $v$ for simplicity.

### 2.2. Policy Iteration

A classic framework in RL for solving Equation 1 is policy iteration, as described in Sutton & Barto (2018). The framework operates through two core mechanisms: policy evaluation, where the value function $v$ is updated under the current policy $\pi$, and policy improvement, where the policy is updated to select actions that achieve higher return based on the revised value function. There are two main approaches for policy iteration: model-free and model-based. Model-free methods do not utilise any explicit knowledge about the environment, while useful, they are known for their high variance and instability (Henderson et al., 2018).

Model-based methods are an alternative to model-free methods for performing policy iteration. Models can be used in a variety of ways (Berkenkamp et al., 2017; Sekar et al., 2020; Hafner et al., 2023) but one highly successful use case is for planning (Silver et al., 2018; Weber et al., 2017). Planning can either be performed using a perfect model of the environment (Silver et al., 2018) if available or a learned model (Schrittwieser et al., 2020). Performing planning can be beneficial as it allows for counterfactual reasoning and, thus, the comparison of different alternatives from the same state.

**Expert iteration:** Expert iteration (ExIt) leverages planning during training to perform policy improvement. The traditional approach of performing rollouts of actions within the MDP in a linear fashion is refereed to as the *outer process* (Hubert et al., 2021). Planning is used to perform another layer of action rollouts for each state seen in the outer process, this is known as the *inner process*. In ExIt, a planning algorithm employs a parameterised model-free policy, $\pi_\theta$, and its associated value function, $v_\theta$, to generate a policy $\pi'$ with higher expected return for a given state. This new policy is then used to progressively refine $\pi_\theta$, creating a feedback loop of continuous improvement. We can formalise this by defining planning as a policy improvement operator (Ghosh et al., 2020). Given policy $\pi$, we define an improvement operator $\mathcal{I}$ such that $\mathcal{I}\pi$ is also a policy i.e. $\mathcal{I}\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, and satisfies that $\forall s \in \mathcal{S}, v^{\mathcal{I}\pi}(s) \geq v^\pi(s)$. In order to update $\pi$, a projection operator is used that maps the improved policy back to the space of realisable policies $\Pi$. A practical method to execute the projection step involves choosing $\pi_{\theta'}$ that minimizes the forward Kullback-Leibler divergence from the improved policy, expressed as:

$$\pi_{\theta'} \in \operatorname*{argmin}_\theta \mathrm{KL}(\pi' || \pi_\theta) \tag{2}$$

Training in this way can allow algorithms to locally smooth out errors, leading to stable updates when using function approximation, as suggested by Moerland et al. (2023). Additionally, instead of only performing policy improvement for executed actions in the outer process, planning can perform policy improvement for multiple actions, potentially even the whole action space. We utilise expert iteration in our proposed framework (Section 3.3) where we leverage Sequential Monte Carlo Planning as an improvement operator.

### 2.3. Control as Inference

The control as inference framework (Toussaint & Storkey, 2006; Kappen et al., 2012; Levine, 2018), re-imagines RL as performing inference within a probabilistic graphical model. If we consider the finite horizon $T$ case, trajectories $\tau$ are distributed according to the transition dynamics of the system and a uniform action prior.

$$p(s_0, a_0, \dots, s_T) = \mu(s_0) \prod_{t=0}^{T} p(a_t) \mathcal{T}(s_{t+1}|s_t, a_t). \quad (3)$$

However, this distribution is not particularly interesting in itself as it does not take into account the rewards received at each timestep. Accounting for rewards is critical for control problems since we are interested in sampling high reward trajectories. Therefore, the control as inference framework introduces an auxiliary binary variable $\mathcal{O}_t$ that indicates a notion of optimality at timestep $t$, such that

$$p(\mathcal{O}_t = 1|s_t, a_t) \propto \exp(r(s_t, a_t)). \quad (4)$$

We can thus formulate a target distribution, of optimal trajectories as $p(\tau|\mathcal{O}_{1:T})$ which assumes optimality at all timesteps. This is an intuitive definition for the distribution of $\mathcal{O}_t$ as the constructed target distribution assigns higher probabilities to trajectories with higher reward and lower probabilities given lower rewards. The power of viewing RL in this way is that it enables an array of existing inference algorithms to be used for control. In this paper we investigate using Sequential Monte Carlo to perform such inference.

### 2.4. Sequential Monte Carlo Methods

#### 2.4.1. Overview

Sequential Monte Carlo (SMC) methods (Gordon et al., 1993) are designed to sample from an intractable distribution $p(x)$ which we refer to as the *target* by using a tractable proposal distribution $q(x)$. *Importance Sampling* does this by sampling from $q$ and weighting samples by $p(x)/q(x)$. Practically a set of $N$ particles are used $\{x^{(n)}\}_{n=1}^{N}$, where $x^n$ represents a sample from the support that the target and proposal distributions are defined over. Each particle uses a sample from the proposal distribution to estimate the target, increasing $N$ naturally improves the estimation of $p(x)$ (Bain & Crisan, 2009). After importance weights are calculated, the estimation of the target distribution is

$$\sum_{n=1}^{N} \bar{w}^{(n)} \delta_{x^{(n)}}(x), \quad (5)$$

where $\bar{w}$ is the normalised importance sample weight and $\delta_{x^{(n)}}$ is the dirac measure located at $x^{(n)}$ . *Sequential Importance Sampling* generalises this to the sequential case where $x = (x_1, \dots, x_T)$ and importance weights can be calculated sequentially using

$$w_t(x_{1:t}) = w_{t-1}(x_{1:t-1}) \cdot \frac{p(x_t|x_{1:t-1})}{q(x_t|x_{1:t-1})}. \quad (6)$$

However, this method can suffer from a phenomenon known as *weight degeneracy* (Maskell & Gordon, 2001) where most of the trajectories become very unlikely under the target distribution and once normalised, one importance weight dominates the rest. This results in most samples having negligible weights. This is an unavoidable problem as the variance of particle weights is guaranteed to increase (Doucet et al., 2000) as they are sequentially updated. If a majority of particles do not contribute to the estimation of $p(x)$ this represents a significant waste of compute.

*Sequential Importance Resampling* (Liu et al., 2001) helps to alleviate this by resampling a new set of particles periodically according to the current weights, after which the weights are reset. Resampling can introduce its own problems of *sample impoverishment* (Maskell & Gordon, 2001), where resampling allocates most or even all of its budget to a single particle. One way of controlling this, which we explore in this work, is using a temperature $\tau_s$ in the resampling Softmax distribution.

## 3. Expert Iteration with Sequential Monte Carlo

### 3.1. Sequential Monte Carlo for planning

SMC methods have shown promise for their use in performing inference to solve continuous control problems (Toussaint & Goerick, 2007; Lazaric et al., 2007; Piché et al., 2018; Lioutas et al., 2022). In this work we focus on SMC methods in order to estimate the following target distribution of trajectories up to a planning horizon $h$:

$$p(a_t, s_{t+1}, \dots, s_{t+h}, a_{t+h}|\mathcal{O}_{1:T}, s_t). \quad (7)$$

Intuitively this can be thought of as the distribution over trajectories of length $h$ given acting optimality.

The following sequential weight update can be derived to sample from this distribution using a proposal $\pi_\theta$ Piché et al. (2018),

$$w_t \propto w_{t-1} \cdot \mathbb{E}_{s_{t+1}|s_t,a_t} \left[\exp(A(s_t, a_t, , r_t, s_{t+1}))\right], \quad (8)$$

$$A(s_t, a_t, r_t, s_{t+1}) = r_t + V(s_{t+1}) - \log \pi_\theta(a_t|s_t) - \log \mathbb{E}_{s_t|s_{t-1},a_{t-1}} \left[\exp(V(s_t))\right], \quad (9)$$

where $V$ is the optimal value function. Intuitively, trajectories that are better than expected accumulate larger weights and trajectories worse than expected accumulate

lower weights. In practice $V$ is not known, previously existing algorithms have been used to train this such as SAC (Haarnoja et al., 2018) (Piché et al., 2018).

Aside from the weight update, an important consideration for SMC is what proposal distribution $q(x)$ will be used to sample trajectories (in our method we refer to this as the *prior* to align with the Expert Iteration literature (Anthony et al., 2017), although in SMC literature it is usually referred to as a proposal). This can range from a random Gaussian prior or human behaviour prior both used in Lioutas et al. (2022), to a fully pre-trained policy from a different training method. The prior that is used can be critical to the performance of SMC. An uninformative prior can make SMC estimates of the posterior extremely poor, requiring large particle counts to compensate, which is computationally costly. We note that in previous works (Piché et al., 2018) the particle counts are set to 2500 which is a planning cost that is prohibitively costly for most practical problems, especially if planning is integrated during training.

### 3.2. Sequential Monte Carlo Expert Iteration

Unlike previous work that leverage existing algorithms to train proxy prior and value functions, we recognise that posterior estimates of the target distribution (Equation 7), generated by SMC can be used to iteratively refine both a parameterised policy $\pi_\theta$ and the estimate of the optimal value function $v_\theta$, as shown in Figure 1.

If we iterate this process using $\pi_\theta$ and $v_\theta$ for SMC planning, we create a cycle of improvement. With a better prior and value estimate, the posterior estimate is more accurate, further leading to better training targets for the prior. Our method can, therefore, be viewed as leveraging Sequential Monte Carlo planning methods introduced in Section 3.1 with the ExIt in 2.2. First, we detail the exact Sequential Monte Carlo algorithm that we use to perform policy improvement and discuss key features that stabilise training to enable its application to different environments unmodified, then we discuss details of our implementation of ExIt as the core training method for SMX.

We use neural-guided SMC planning as the inner process within ExIt training and refer to this as SMX going forward. Figure 1 visualises this search for a single timestep $t$. SMX planning for a given state $s_t$ consists of performing $N$ trajectory rollouts in parallel to a horizon length $h$ with individual trajectories represented using $x_{t:t+h} = (s_t, a_t, s_{t+1}, \dots, s_{t+h})$. Particles are initialized to $\{x_t^{(n)}\}_{n=1}^N$, through sequential rollouts, each particle progressively aggregates states and actions, culminating in the formation of a trajectory spanning length $h$. Every step, importance sampling weights $\{w_i^{(n)}\}_{n=1}^N$ are sequentially updated, see Section 3.1 for how these weights are updated.

**Algorithm 1** SMX Planning (timestep $t$)

1: Initialize $\{s_t^{(n)} = s_t\}_{n=1}^N$
2: Set $\{w_t^{(n)} = 1\}_{n=1}^N$
3: **for** $i \in \{t, \dots, t+h\}$ **do**
4: $\quad \{a_i^{(n)} \sim \pi_\theta(\cdot|s_i^{(n)})\}_{n=1}^N$
5: $\quad \{s_{i+1}^{(n)} \sim \mathcal{T}(s_i^{(n)}, a_i^{(n)})\}_{n=1}^N$
6: $\quad \{r_i^{(n)} \sim r(s_i^{(n)}, a_i^{(n)})\}_{n=1}^N$
7: $\quad \{w_i^{(n)} = w_{i-1}^{(n)} \cdot \exp(A_\theta(s_i^{(n)}, r_i^{(n)}, s_{i+1}^{(n)}))\}_{n=1}^N$
8: $\quad$ **if** $(i - t + 1) \bmod p = 0$ **then**
9: $\quad\quad \{w_i^{(n)}\}_{n=1}^N = \text{MinMaxNorm}\big(\{\log(w_i^{(n)})\}_{n=1}^N\big)$
10: $\quad\quad \{w_i^{(n)}\}_{n=1}^N = \text{SoftMax}\left(\{w_i^{(n)}\}_{n=1}^N, \tau_s\right)$
11: $\quad\quad \{x_{t:i}^{(n)}\}_{n=1}^N \sim \text{Mult}(n; w_i^{(1)}, \dots, w_i^{(N)})$
12: $\quad\quad \{w_i^{(n)} = 1\}_{n=1}^N$
13: $\quad$ **end if**
14: **end for**
15: Define $\{a_t^{(n)}\}$ as the set of first actions from $\{x_{t:t+h}^{(n)}\}_{n=1}^N$
16: $\pi'(a|s_t) = \frac{1}{N}\sum_{n=1}^N \delta_{a_t^{(n)}}(a)$

Note that SMX search practically only requires the current particle states, weights and initial actions to be stored, removing the memory costs of storing large search trees. As discussed in Section 2.4.1 since SMC methods can suffer from weight degeneracy, we make use of resampling (line 11). Lower weight particles may not be sampled at all or higher weight particles sampled multiple times. After resampling, the new set of particles continue their rollouts. If we define $\{a_t^{(n)}\}_{n=1}^N$ as the set of initial actions in each trajectory in the set $\{x_{t:t+h}^{(n)}\}_{n=1}^N$, the improved policy generated by SMX is the marginal distribution of initial actions over the approximated posterior, where all but the first action is marginalised from the distribution over trajectories implied by the particles, using a sum of diracs centered on initial actions (line 16).

While implementing an SMC algorithm with a static prior is fairly simple, ensuring that SMC continues to act as a policy improvement operator during ExIt as both $\pi_\theta$ and $v_\theta$ are updated is not trivial. Due to the fact that our search policy is always improving, the range of the value function is constantly changing. This directly impacts the scale of the advantage weight update. Balancing the scale between the advantage and $-\log(\pi_\theta(a_t|s_t))$ is difficult, and we find that setting this to a low value is important to maintain good performance when used within ExIt, for our main results 0 is used. A primary objective in this work was developing an SMC algorithm that was general and could be applied across environments unaltered. We identified three crucial, practical improvements to our SMX algorithm that are important for its application as a policy improvement

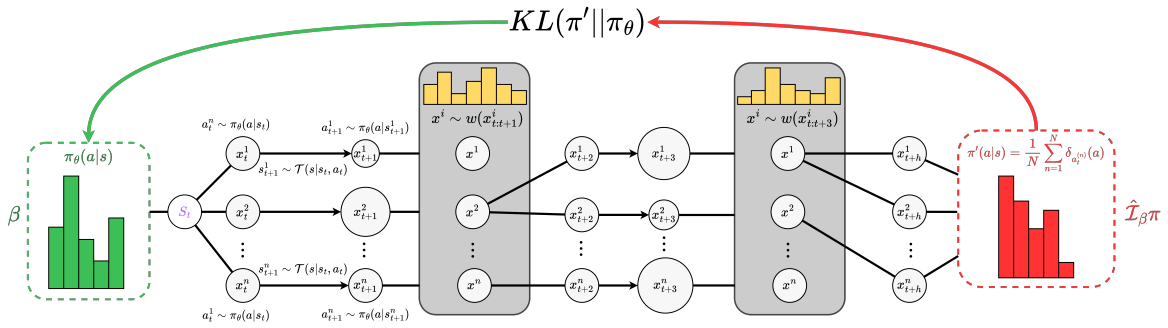



Figure 1: Diagram depicting a representation of SMX search from left to right. $N$ Rollouts are executed in parallel according to $\pi_\theta$ (the sampling policy $\beta$). At each step in the environment the particle weights are adjusted, indicated by the particle sizes. We depict two resampling zones where particles are resampled (favouring higher weights) and weights are reset. Finally an improved policy $\pi' = \hat{\mathcal{I}}_\beta \pi$ is constructed from the initial actions from the remaining particles, furthest to the right. This improved policy is then used to update $\pi_\theta$.

operator within ExIt.

**Normalising advantage updates:** We find Sequential Monte Carlo methods to be sensitive to reward scale, resulting in sample impoverishment at resampling steps. Scaling the advantage estimates with min-max normalisation before resampling (line 9) helps to alleviate this problem. This is analogous to action-value normalisation in MuZero. In Appendix B we show results for SMX without this modification.

**Resampling Temperature:** Too high of a resampling temperature can limit the impact of weight updates and subsequently policy improvement. Too low of a resampling temperature can cause sample impoverishment. We, therefore, introduce temperature $\tau_s$ within the softmax distribution (line 10). Paired with our normalisation method, a temperature of $\tau_s = 0.1$ works well across all environments investigated and ensures that horizon length can be utilised.

**Resampling every $p$ steps:** Resampling is used to prevent one importance weight from dominating the remaining weights i.e. weight degeneracy. However, practically, it can lead to an over-reliance on a small amount of trajectory information and cause sample impoverishment. To alleviate this, we perform particle resampling every $p$ planning steps (line 8), and find empirically that periods 2 and 4 perform well.

### 3.3. Expert Iteration

We adopt the ExIt training methodology as outlined by Anthony et al. (2017) and Silver et al. (2018). At each timestep $t$, the agent performs an SMX search and computes an improved policy $\pi'(a|s_t)$ and a value target $v'(s_t)$ which are both added to a buffer $B$. After performing $m$ steps as part of the outer process, as discussed in Section 2.2, and collecting environment trajectory data, $\pi_\theta$ and $v_\theta$ are updated. To update $\pi_\theta$, we use the forward KL divergence[1] to minimise the distance from the improved policy. For the value function, we utilise Generalised Advantage Estimation (Schulman et al., 2016) in order to calculate the final value targets and train a distributional value function $v_\theta$ with cross entropy loss. See Appendix C for details.

## 4. Results

We focus on three main areas for our analysis of SMX. First, we look at the performance of SMX relative to other model-free and model-based algorithms on a variety of environments with both discrete and continuous action spaces. We then focus on the scaling behaviour of SMX, looking at how asymptotic performance scales with expert search budget. Lastly, we analyse SMX from a practical perspective, comparing AlphaZero (Silver et al., 2018) and SMX speeds on hardware accelerators. To ensure the robustness of the conclusions, we follow the evaluation methodology of Agarwal et al. (2021). This evaluation methodology groups performance across environments, enabling clearer conclusions over the significance of learning algorithms. We include individual results in Appendix B, along with additional summary plots that measure the statistical significance of learning algorithm performance.

### 4.1. Experimental Setup

We evaluate on Boxoban (Guez et al., 2018a), a discrete environment commonly used to compare planning and reasoning methods (Guez et al., 2018b), and Rubik's Cube, a sparse reward environment with a large combinatorial state-action

[1]The reverse Kullback-Leibler divergence has also been motivated for use in continuous implementations of AlphaZero (Silver et al., 2018) however we consistently found it performed worse than the forward case for all ExIt baselines, this could be due to its mode seeking behaviour, limiting exploration.

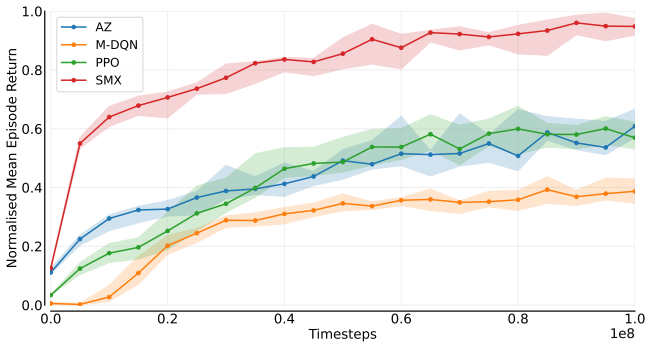



Figure 2: **Discrete Environments:** Amalgamated results for Rubiks Cube 7 and Boxoban Hard. The Y-axis shows the Interquartile mean of min-max normalised scores of all evaluation episodes. Evaluations are performed throughout training to measure performance as a function of environment transitions. Shading indicates 95% confidence intervals.

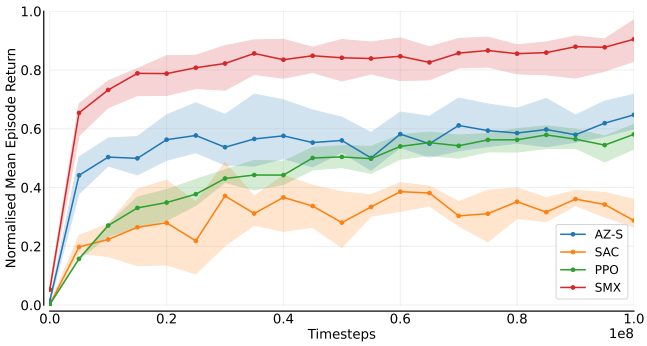



Figure 3: **Continuous Environment:** Amalgamated results for Ant, Half Cheetah and Humanoid. Y-axis shows the Interquartile mean of min-max normalised scores of all evaluation episodes. Evaluations are performed throughout training to measure performance as a function of environment transitions. Shading indicates 95% confidence intervals.

space. We train on a uniform distribution of scrambles away from the solution state in the range [3,7] and evaluate on 7 scrambles. We leverage Jumanji (Bonnet et al., 2023) for both implementations. We compare performance to AlphaZero with modern improvements to the search method introduced in MuZero (Schrittwieser et al., 2020). Our AlphaZero benchmark uses the official open-source implementation[2] to ensure a fair comparison, ensuring the use of default parameters that have been found to perform well (Danihelka et al., 2021). Our model-free baselines include PPO (Schulman et al., 2017) and Munchausen-DQN (Vieillard et al., 2020), an extension to DQN that fits within the maximum entropy RL framework (Ziebart et al., 2008).

For continuous control we evaluate on the Brax (Freeman et al., 2021a) benchmark focusing on Ant, Half Cheetah, and Humanoid. We baseline our results to Sampled MuZero (Hubert et al., 2021), a modern extension to MuZero for large or continuous action spaces. We utilise a true environment model making our implementation of Sampled MuZero more akin to AlphaZero. For model-free methods we compare to PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018), to validate a method within the maximum entropy framework.

In our experiments, we adhere to standardised parameters for AlphaZero, employing a search budget of 64 simulations. For our primary findings, we deploy SMX, configured with 16 particles and a horizon of 4. For remaining parameters see Appendix D. We parameterise the prior and value functions using environment-dependent neural networks. For discrete environments, we use multi-layer networks with skip connections and separate readout policy and value heads. For continuous environments we parameterise policies with a Gaussian distribution using separate networks for the policy and value prediction. See Appendix A for further details and Appendix B.4 for hardware details.

[2]Open-source implementation available at mctx.

### 4.2. Performance and Comparisons

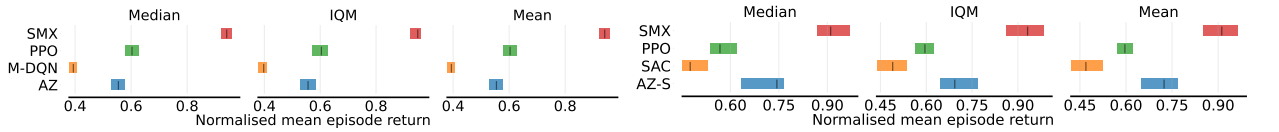
**Policy Improvement:** Our primary results are presented in Figures 2 and 3, here we show empirical evidence that the SMX outperforms all other methods across both discrete and continuous benchmarks. This conclusion holds for all per-environment results reported in Appendix B. This also shows that leveraging SMC within Expert iteration enables performance with much lower search budgets than were used for SMC in Piché et al. (2018), i.e. particles: 16 vs 2500 and horizon: 4 vs 10 / 20.

Following Agarwal et al. (2021), Colas et al. (2018) and Gorsane et al. (2022), in addition to analysing the sample efficiency of different algorithms we also perform a final evaluation using parameters that achieved the best performance throughout intermediate evaluation runs during training. This final evaluation uses 1280 episodes. Figure 4 shows SMX achieves statistically significant improvements in all point estimates in both discrete and continuous environments.

**SMX vs AlphaZero:** Comparing the model-based methods, SMX performs well on Boxoban, marginally above the performance of AlphaZero, with both solving over 95% of problems. For Rubik's cube with no change in parameters, SMX outperforms all baselines, including AlphaZero, by a large margin. Likewise for continuous control SMX significantly outperforms AlphaZero, which maintains stable but low performance.

### 4.3. Scaling Behaviour of Search

Figure 5 shows how SMX training scales with particle counts and horizon length. We evaluate on the subset of hard boxoban problems, as all versions of SMX solve al-




**Discrete** **Continuous**

Figure 4: **Point Aggregates:** Absolute evaluation using the best checkpoint found during training. Results include 95% confidence intervals generated from stratified bootstrapping.

most all of the problems on the unfiltered dataset, making comparison difficult. SMX utilises the capabilities of modern hardware accelerators to efficiently parallelise the particle steps over the chosen search horizon. Empirically we see that scaling the number of particles provides a greater increase in performance when compared to scaling the horizon length. We also see that particle count and horizon length should be scaled together, otherwise performance can plateau or even drop. There is also scaling in the asymptotic performance of the prior policy. For high particle counts and horizons lengths this benefit plateaus, likely due to the constraint over what policies the parameterised prior can represent, an issue highlighted in Section 4.2.

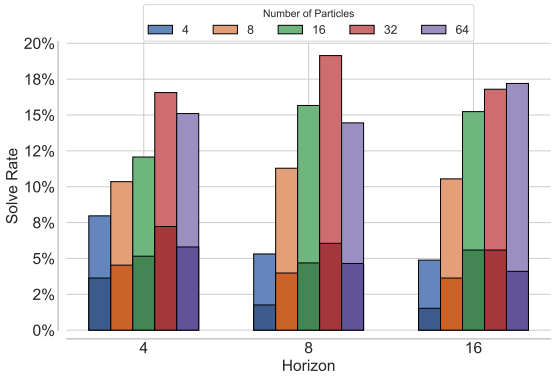



Figure 5: **Scaling:** Performance on Hard Boxoban after $10^8$ environment steps with varying particle numbers and search horizon for SMX during training. Full bar performance showing search performance and shaded regions showing the prior performance.

### 4.4. Speed

Finally, in Figure 6 we demonstrate how the performance of the search scales with additional search budget, measured in wall clock time and benchmarked on a TPUv3-8. While we recognise that such analysis can be difficult to perform due to implementation discrepancies, we have used the official Jax implementation of AlphaZero (MCTX) and compare this to SMX. Such analysis is crucial for understanding the practicalities of using search, both within the ExIt framework and at test time. We perform scaling on a fixed prior and value function (the highest performing model from our main ExIt training results[3]). AlphaZero scales much more slowly when compared to SMX for any horizon length. We also see, for very low compute, SMX is able to generate very strong performance compared to AlphaZero. For SMX, depending on the compute availability, different settings of search are preferable. For limited compute, low horizon lengths and relatively higher particle counts provide the best performance, but as compute availability increases, the gains from increasing particle counts decrease and instead horizon length $h$ should be increased.

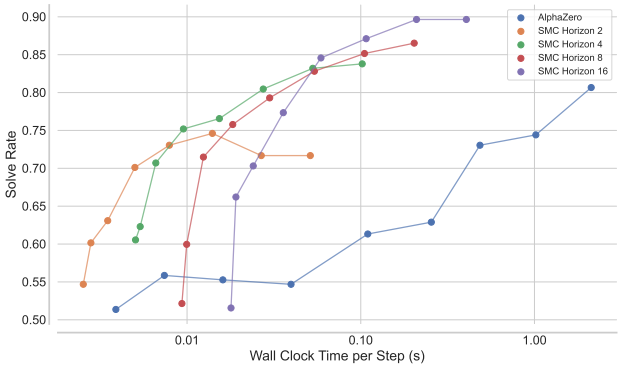



Figure 6: **Wall Clock Time Comparison:** A comparison of AlphaZero and SMX performance over 512 Rubik's Cube episodes with 10 scrambles. We scale search computation, measured in wall-clock time and displayed on a logarithmic scale, plotted against the solve rate. To scale search for SMX, horizon is fixed and the number of particles are scaled in the range [4-512]. AlphaZero's search budget is scaled in the range [4-1024] (we limit to 1024 at which point further scaling becomes prohibitively expensive).

## 5. Related Work

**Sequential Monte Carlo Methods:** Sequential Monte Carlo (Gordon et al., 1993; Kitagawa, 1996; Liu & Chen, 1998) methods, also referred to as particle filters (Pitt & Shephard, 2001; Andrieu et al., 2004), have been applied in

[3]We investigated using AlphaZeros's prior and value function for evaluating its scaling, but its performance was too low and did not provide an accurate representation of AlphaZero scaling.

continuous control settings. SMC-Learning (Lazaric et al., 2007) was used to perform control in a boat control scenario, updating particle weights using a Boltzmann exploration strategy, according updates in Q-values estimations. Prior to the major advancements in deep learning, they leverage a weighted average of kernel densities to parameterise the prior and use a tabular critic. Our work builds on this, parameterising these functions with large neural networks, enabling them to be much more expressive. This approach provides a far better estimation of the posterior and allows for the use of neural advantage estimates for particle updates. Piché et al. (2018) derive a maximum entropy (Ziebart et al., 2010) advantage weight update for particles to estimate the posterior over trajectories given optimality, framed within control as inference. They applied this methodology to the Mujoco continuous control benchmark (Todorov et al., 2012). To perform rollouts and to estimate the advantage function they use a SAC (Haarnoja et al., 2018) policy and value function as proxies. We also utilise this advantage update but instead, train a progressively improving policy and value function using SMC posterior estimates. CriticSMC (Lioutas et al., 2022) utilises a soft action value function (Haarnoja et al., 2017) to score particles, eliminating the need to explicitly perform a step in the environment. This enables more efficient exploration of trajectories, demonstrating performance on lower SMC budgets compared with Piché et al. (2018), however their method is still slower in wall clock time. By directly parameterising our policy, we can perform such computations within the networks themselves, which also enables low SMC budgets. While not evaluated in control settings, Gu et al. (2015) explore the parameterisation of the proposal in SMC using LSTMs (Hochreiter & Schmidhuber, 1997), applying it to music generation and positional inference in a cart and pole system.

**Model-based Reinforcement Learning:**

Model-based methods that integrate planning have achieved significant success within RL. AlphaZero (Silver et al., 2018) notably exhibited robust asymptotic performance in Go, Chess, and Shogi. It was able to outperform human players purely through self-play, leveraging Expert Iteration (Anthony et al., 2017), also known as Dual Policy Iteration (Sun et al., 2018). Neural variants of Monte Carlo Tree Search (Auer et al., 2002; Coulom, 2006) are the most explored method for planning within ExIt. Our work can also be viewed as performing ExIt, but using Neural SMC for planning, which is a scalable and parallelisable algorithm. Grill et al. (2020) show that using MCTS within the ExIt framework can be framed as regularised policy optimization. A large amount of research into ExIt focuses on iterating upon early neural MCTS variants, applying it to various domains such as stochastic environments (Antonoglou et al., 2021) or continuous control (Moerland et al., 2018; Yang et al., 2020; Hubert et al., 2021). In contrast, SMC planning can be flexibly applied to both discrete and continuous control without the need for modification. MuZero (Schrittwieser et al., 2020) shows that planning can be performed using a learned model. Gumbel MuZero (Hamrick et al., 2019; Danihelka et al., 2021) explores planning in low-budget regimes.

## 6. Conclusion

Planning-based policy improvement operators have proven to be powerful methods to enhance the learning of policies for complex environments when compared to model-free methods. Despite the success of these methods, they are still limited in their usefulness, due to the requirement of high planning budgets and the need for algorithmic modifications to adapt to large or continuous action spaces. In this paper, we introduce a modern implementation of Sequential Monte Carlo planning as a policy improvement operator, within the Expert Iteration framework. This results in a general training methodology with the ability to scale in both discrete and continuous environments.

Our work provides several key contributions. First, we show that SMX is a powerful policy improvement operator, outperforming all the model-based and model-free baselines, while proving to be more sample-efficient. Additionally, SMX is shown to be competitive across both continuous and discrete domains, without additional environment-specific alterations. This illustrates the effectiveness of SMX as a generic approach, in contrast to prior methods, requiring domain-specific enhancements. Furthermore, the parallelisable nature of SMX results in the efficient scaling behaviour of the search method. This allows SMX to achieve a significant boost in performance at inference time by scaling our search budget. Finally, we demonstrate that scaling SMX results in faster wall-clock times compared to previous work utilising tree-based search methods. The presented work culminates in a versatile, neural-guided, sample-based planning method that demonstrates superior performance and scalability over previous benchmarks.

**Limitations & Future Work:** Our work used exclusively perfect world models of the environment in order to isolate the effects of various planning methods on performance. Extending our research to include a learned world model would broaden the applicability of our methods to more complex problems that may not have a perfect simulator and are therefore unsuitable for direct planning. Additionally, we applied the Sequential Monte Carlo (SMC) method only in deterministic settings. Adapting our approach for stochastic environments presents challenges. Notably, the resampling mechanism of SMC search enables the selection of dynamics most favorable to the agent, potentially encouraging risk-seeking behavior for stochastic environments.

## References


Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Andrieu, C., Doucet, A., Singh, S. S., and Tadic, V. B. Particle methods for change detection, system identification, and control. *Proceedings of the IEEE*, 92(3):423–438, 2004.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

Antonoglou, I., Schrittwieser, J., Ozair, S., Hubert, T. K., and Silver, D. Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*, 2021.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Bain, A. and Crisan, D. *Fundamentals of stochastic filtering*, volume 3. Springer, 2009.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pp. 449–458. PMLR, 2017.

Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A. Safe model-based reinforcement learning with stability guarantees. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/766ebcd59621e305170616ba3d3dac32-Paper.pdf.

Bonnet, C., Luo, D., Byrne, D., Surana, S., Coyette, V., Duckworth, P., Midgley, L. I., Kalloniatis, T., Abramowitz, S., Waters, C. N., Smit, A. P., Grinsztajn, N., Sob, U. A. M., Mahjoub, O., Tegegn, E., Mimouni, M. A., Boige, R., de Kock, R., Furelos-Blanco, D., Le, V., Pretorius, A., and Laterre, A. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2023. URL https://arxiv.org/abs/2306.09884.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Chalumeau, F., Surana, S., Bonnet, C., Grinsztajn, N., Pretorius, A., Laterre, A., and Barrett, T. D. Combinatorial optimization with policy adaptation using latent space search. 2023.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. In *International conference on machine learning*, pp. 1039–1048. PMLR, 2018.

Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2021.

DeepMind, Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., Kapturowski, S., Keck, T., Kemaev, I., King, M., Kunesch, M., Martens, L., Merzic, H., Mikulik, V., Norman, T., Papamakarios, G., Quan, J., Ring, R., Ruiz, F., Sanchez, A., Sartran, L., Schneider, R., Sezener, E., Spencer, S., Srinivasan, S., Stanojević, M., Stokowiec, W., Wang, L., Zhou, G., and Viola, F. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Doucet, A., Godsill, S., and Andrieu, C. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10:197–208, 2000.

Dror, R., Shlomov, S., and Reichart, R. Deep dominance-how to properly compare deep neural models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2773–2785, 2019.

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., Ruiz, F. J. R., Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., and Kohli, P. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930): 47–53, 2022. doi: 10.1038/s41586-022-05172-4.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax - a differentiable physics engine for large scale rigid body simulation, 2021a. URL http://github.com/google/brax.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021b.

Ghosh, D., C Machado, M., and Le Roux, N. An operator view of policy gradient methods. *Advances in Neural Information Processing Systems*, 33:3397–3406, 2020.

Gordon, N. J., Salmond, D. J., and Smith, A. F. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, volume 140, pp. 107–113. IET, 1993.

Gorsane, R., Mahjoub, O., de Kock, R. J., Dubb, R., Singh, S., and Pretorius, A. Towards a standardised performance evaluation protocol for cooperative marl. *Advances in Neural Information Processing Systems*, 35:5510–5521, 2022.

Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., and Munos, R. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pp. 3769–3778. PMLR, 2020.

Gu, S. S., Ghahramani, Z., and Turner, R. E. Neural adaptive sequential monte carlo. *Advances in neural information processing systems*, 28, 2015.

Guez, A., Mirza, M., Gregor, K., Kabra, R., Racaniere, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., Wayne, G., Silver, D., Lillicrap, T., and Valdes, V. An investigation of model-free planning: boxoban levels. https://github.com/deepmind/boxoban-levels/, 2018a.

Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., and Silver, D. Learning to search with mctsnets. In *International conference on machine learning*, pp. 1822–1831. PMLR, 2018b.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pp. 1352–1361. PMLR, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Pfaff, T., Weber, T., Buesing, L., and Battaglia, P. W. Combining q-learning and search with amortized value estimates. In *International Conference on Learning Representations*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep Reinforcement Learning that Matters, 2018.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hoffman, M. W., Shahriari, B., Aslanides, J., Barth-Maron, G., Momchev, N., Sinopalnikov, D., Stańczyk, P., Ramos, S., Raichuk, A., Vincent, D., Hussenot, L., Dadashi, R., Dulac-Arnold, G., Orsini, M., Jacq, A., Ferret, J., Vieillard, N., Ghasemipour, S. K. S., Girgin, S., Pietquin, O., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., Henderson, S., Friesen, A., Haroun, R., Novikov, A., Colmenarejo, S. G., Cabi, S., Gulcehre, C., Paine, T. L., Srinivasan, S., Cowie, A., Wang, Z., Piot, B., and de Freitas, N. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979.

Hottung, A., Kwon, Y.-D., and Tierney, K. Efficient active search for combinatorial optimization problems. *arXiv preprint arXiv:2106.05126*, 2021.

Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S., and Silver, D. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, pp. 4476–4486. PMLR, 2021.

Imani, E. and White, M. Improving regression performance with distributional losses. In *International conference on machine learning*, pp. 2157–2166. PMLR, 2018.

Kappen, H. J., Gómez, V., and Opper, M. Optimal control as a graphical model inference problem. *Machine learning*, 87:159–182, 2012.

Kitagawa, G. Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of computational and graphical statistics*, 5(1):1–25, 1996.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

Lazaric, A., Restelli, M., and Bonarini, A. Reinforcement learning in continuous action spaces through sequential monte carlo methods. *Advances in neural information processing systems*, 20, 2007.

Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Levy, H. Stochastic dominance and expected utility: Survey and analysis. *Management Science*, pp. 555–593, 1992.

Lioutas, V., Lavington, J. W., Sefas, J., Niedoba, M., Liu, Y., Zwartsenberg, B., Dabiri, S., Wood, F., and Scibior, A. Critic sequential monte carlo. In *The Eleventh International Conference on Learning Representations*, 2022.

Liu, J. S. and Chen, R. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998.

Liu, J. S., Chen, R., and Logvinenko, T. A theoretical framework for sequential importance sampling with resampling. In *Sequential Monte Carlo methods in practice*, pp. 225–246. Springer, 2001.

Mann, H. B. and Whitney, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pp. 50–60, 1947.

Maskell, S. and Gordon, N. A tutorial on particle filters for on-line nonlinear/non-gaussian bayesian tracking. *IEE Target Tracking: Algorithms and Applications (Ref. No. 2001/174)*, pp. 2–1, 2001.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Moerland, T. M., Broekens, J., Plaat, A., and Jonker, C. M. A0c: Alpha zero in continuous action space. *arXiv preprint arXiv:1805.09613*, 2018.

Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1): 1–118, 2023.

Piché, A., Thomas, V., Ibrahim, C., Bengio, Y., and Pal, C. Probabilistic planning with sequential monte carlo methods. In *International Conference on Learning Representations*, 2018.

Pitt, M. K. and Shephard, N. Auxiliary variable based particle filters. *Sequential Monte Carlo methods in practice*, pp. 273–293, 2001.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8583–8592. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/sekar20a.html.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Sun, W., Gordon, G. J., Boots, B., and Bagnell, J. Dual policy iteration. *Advances in Neural Information Processing Systems*, 31, 2018.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Tang, Y. and Agrawal, S. Discretizing continuous action space for on-policy optimization. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pp. 5981–5988, 2020.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Toledo, E., Midgley, L., Byrne, D., Tilbury, C. R., Macfarlane, M., Courtot, C., and Laterre, A. Flashbax: Streamlining experience replay buffers for reinforcement learning with jax, 2023. URL https://github.com/instadeepai/flashbax/.

Toussaint, M. and Goerick, C. Probabilistic inference for structured planning in robotics. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3068–3073. IEEE, 2007.

Toussaint, M. and Storkey, A. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pp. 945–952, 2006.

Vieillard, N., Pietquin, O., and Geist, M. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4235–4246, 2020.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Weber, T., Racaniere, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Yang, X., Duvaud, W., and Wei, P. Continuous control for searching and planning with a learned model. *arXiv preprint arXiv:2006.07430*, 2020.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

Ziebart, B. D., Bagnell, J. A., and Dey, A. K. Modeling interaction via the principle of maximum causal entropy. 2010.

# A. Environments

## A.1. Sokoban

Table 1: Summary of Boxoban dataset levels

| Difficulty Level | Dataset Size |
|---|---|
| Unfiltered-Train | 900k |
| Unfiltered-Validation | 100k |
| Unfiltered-Test | 1k |
| Medium | 450k |
| Hard | 50k |

We use the specific instance of Sokoban outlined in Guez et al. (2018a) illustrated in Figure 7 with the codebase available at `https://github.com/instadeepai/jumanji`. The datasets employed in this study are publicly accessible at `https://github.com/google-deepmind/boxoban-levels`. These datasets are split into different levels of difficulty, which are categorised in Table 1.

In this research, we always train on the *Unfiltered-Train* dataset. Evaluations are performed on the *Hard* dataset, important for differentiating the strongest algorithms.

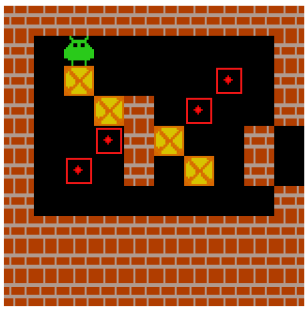


Figure 7: Example of a Boxoban Problem

### A.1.1. Network Architecture

Observations are represented as an array of size (10,10) where each entry is an integer representing the state of a cell. We used a deep ResNet (He et al., 2016) architecture for the torso network to embed the observation. We define a block as consisting of the following layers: [CNN, ResNet, ResNet]. Four such blocks are stacked, each characterized by specific parameters:

- Output Channels: [256, 256, 512, 512]
- Kernel Sizes: [3, 3, 3, 3]
- Strides: [1, 1, 1, 1]

Additionally, the architecture includes two output heads (policy and value), each comprising two layers of size 128 with ReLU activations. The output heads share the same torso network. We use the same architecture for all algorithms.

## A.2. Rubik's Cube

For our Rubik's Cube experiments, we utilize the implementation in Bonnet et al. (2023) available at `https://github.com/instadeepai/jumanji`. Rubik's cube problems can be be made progressively difficult by scaling the number of random actions performed in a solution state to generate problem instances. We always perform training on a uniform distribution sampled from the range [3,7] of scrambled states, followed by evaluation on 7 scrambles. The observation is represented using an array of size (6,3,3). The action space is represented using an array of size (6,3) corresponding to each face and the depth at which it can be rotated.

#### A.2.1. NETWORK ARCHITECTURE

We utilise an embedding layer to first embed the face representations to size (6,3,3,4). The embedding is flattened and we use a torso layer consisting of a two layer residual network with layer size 512 and ReLU activations. Additionally, the architecture includes two output heads (policy and value), each comprising of a single layer of size 512 with ReLU activations. We use the same architecture for all algorithms.

### A.3. Brax

For experimentation on continuous settings, we make use of Brax (Freeman et al., 2021a). Brax is a library for rigid body simulation much like the popular MuJoCo setting (Todorov et al., 2012) simulator. However, Brax is written entirely in JAX (Bradbury et al., 2018) to fully exploit the parallel processing capabilities of accelerators like GPUs and TPUs.

We use the same environmental setup as the original Brax paper. It is important to note, that at the time of writing, there are 4 different physics back-ends that can be used. These back-ends have differing levels of computational complexity and the results between them are not comparable.

Table 2: Observation and action space for Brax environments

| Environment | Observation Size | Action Size |
|---|---|---|
| Halfcheetah | 18 | 6 |
| Ant | 27 | 8 |
| Humanoid | 376 | 17 |

Table 2 contains the observation and action specifications of the scenarios used for our experiments. We specify the dimension size of the observation and action vectors.

#### A.3.1. NETWORK ARCHITECTURE

In practice, we found the smaller networks used in the original Brax publication to limit overall performance. We instead use a 4 layer feed forward network to represent both the value and policy network. Aside from the output layers, all layers are of size 256 and use SiLU non-linearities. We use the same architecture for all algorithms. Unlike Rubik's Cube and Sokoban, we do not use a shared torso to learn embeddings.

## B. Experiments

Table 3: Raw rewards used per environment to compute normalised scores

| Environment | Min Score | Max Score |
|---|---|---|
| Ant | -2401.15 | 11091.63 |
| Half Cheetah | -558.24 | 10827.86 |
| Humanoid | 62.19 | 14573.41 |
| Rubiks Cube | 0.00 | 1.00 |
| Hard Boxoban | -12.00 | 12.60 |
| Unfiltered Boxoban | -12.00 | 13.20 |

Table 3 shows the performance reached by the best model found for each environment. The min and max scores after evaluation are recorded and were used to calculate normalised environment scores.

### B.1. Additional Results

We list additional results including the probability of improvement plots in Figures 8 and 9, and the performance profiles in Figures 10 and 11. We follow the methodology outlined by Agarwal et al. (2021) and Gorsane et al. (2022) by performing the final evaluation using parameters that achieved the best performance throughout intermediate evaluation runs during training

The probability of improvement plots in Figures 8 and 9 illustrates the likelihood that algorithm $X$ outperforms algorithm $Y$

on a randomly selected task. It is important to note that a statistically significant result from this metric is a probability of improvement greater than 0.5 where the CIs do not contain 0.5. The metric utilises the Mann-Whitney U-statistic (Mann & Whitney, 1947). See (Agarwal et al., 2021) for further details. Across both continuous and discrete tasks, the SMX planner consistently outperforms all other methods. The SMX prior also frequently outperforms the AlphaZero (AZ) prior as well as the model-free baselines.

The performance profiles in Figures 10 and 11 illustrate the fraction of the runs over all training environments from each algorithm that achieved scores higher than a specific value (given on the X-axis). Functionally this serves the same purpose as comparing average episodic returns for each algorithm in a tabular form but in a simpler format. From the performance profiles, we can see a clear distinction in the overall ranking of each algorithm across the entire set of tasks. In the discrete case, the SMX planner has consistently higher performance than all other methods and is able to learn a prior that is competitive with existing baselines. In the continuous case, we observe similar results however, interestingly the SMX prior shows poorer performance for middling episode returns when compared to the baselines but shows higher performance for achieving high rewards. This could be a product of the off-policy training, given the model-free network is only trained on high performing trajectories, consisting of a specific distribution of states, if it takes an action that takes it off distribution it likely would perform poorly. Additionally, if an algorithm's curve is strictly greater than or equal to another curve, this indicates "stochastic dominance" (Levy, 1992; Dror et al., 2019). Thus, we can make the claim that both in discrete and continuous environments SMX stochastically dominates the baseline methods. All experiments utilised 3 seeds per task.

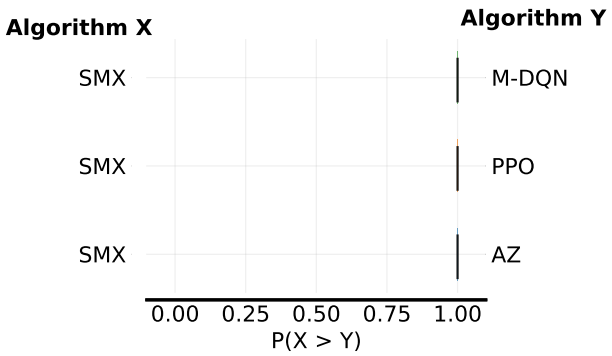



Figure 8: Probability of improvement for discrete environments

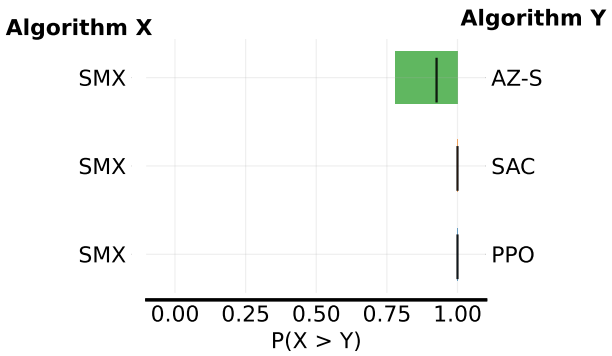



Figure 9: Probability of improvement for continuous environments

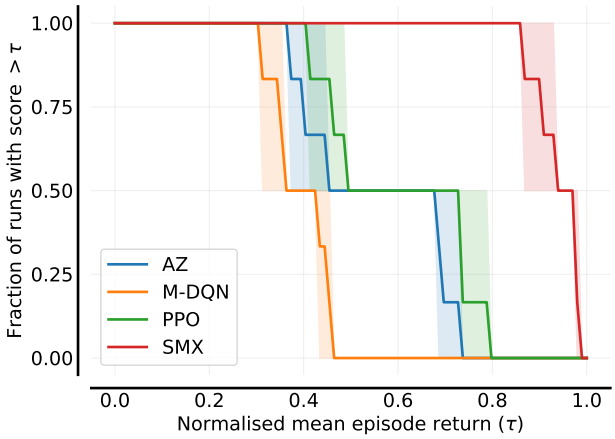



Figure 10: Performance profile for discrete environments

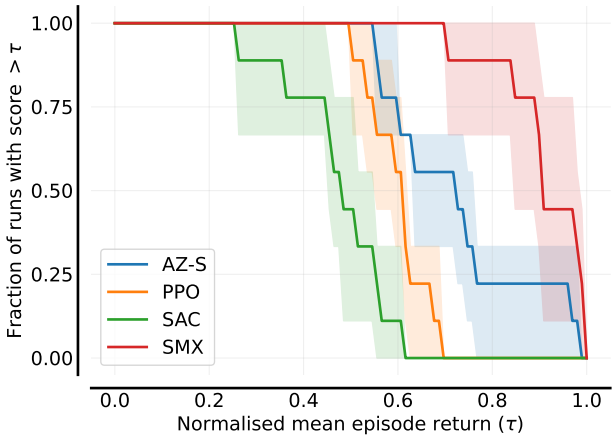



Figure 11: Performance profile for continuous environments

### B.2. Individual Environment Results

For more insight into performance, the performance per environment is illustrated for Brax in Figures 12 to 14, for Sokoban in **??** and Figure 15 and for Rubik's Cube in **??** and Figure 16.

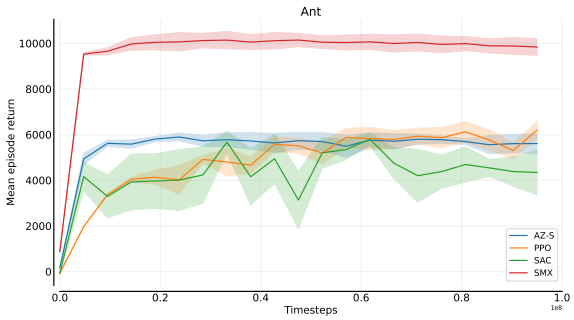



Figure 12: **Ant:** Sample Efficiency Curves

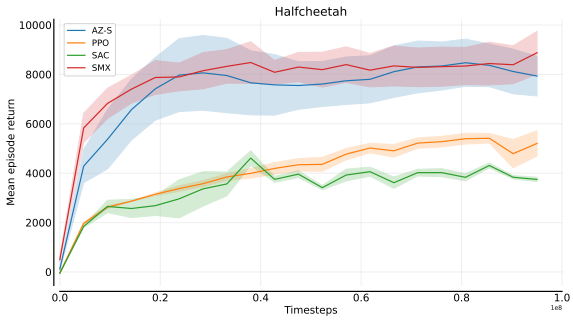



Figure 13: **HalfCheetah:** Sample Efficiency Curves

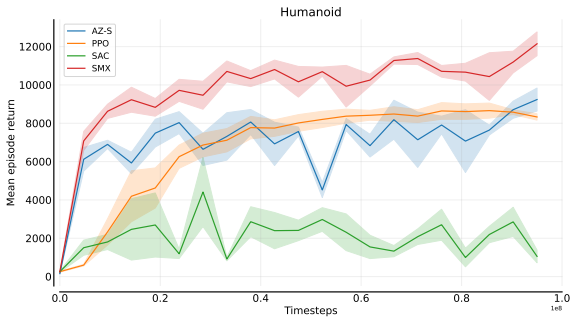



Figure 14: **Humanoid:** Sample Efficiency Curves

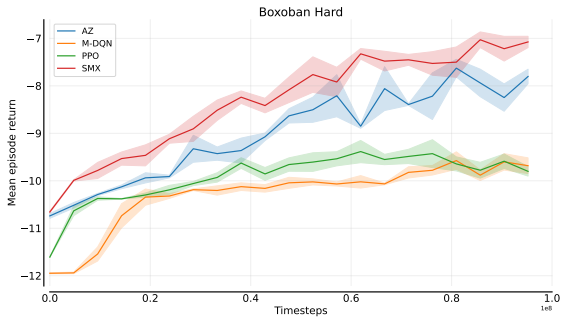



Figure 15: **Boxoban Hard:** Sample Efficiency Curves

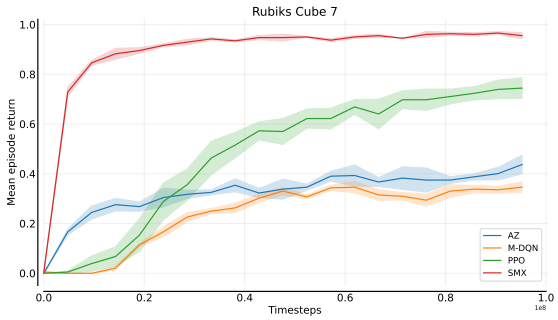



Figure 16: **Rubik's Cube 7 scrambles:** Sample Efficiency Curves

### B.3. Ablations

Table 4: Prior Performance

| Environment | SMX: No Normalisation | SMX: With Normalisation |
|---|---|---|
| Ant | **8314.6** | 6910.1 |
| Humanoid | 9023.6 | **11585.6** |
| Half Cheetah | 1122.1 | **2030.5** |
| Sokoban Unfiltered | 0.00 | **0.89** |
| Rubik's Cube 7 scrambles | 0.00 | **0.71** |

Table 5: Search Performance

| Environment | SMX: No Normalisation | SMX: With Normalisation |
|---|---|---|
| Ant | 8552.7 | **9928.3** |
| Humanoid | **13137.4** | 12255.5 |
| Half Cheetah | **7782.3** | 7070.1 |
| Sokoban Unfiltered | 0.00 | **0.95** |
| Rubik's Cube 7 scrambles | 0.00 | **0.96** |

We perform an ablation where we switch off normalisation and temperature selection for each environment for 1 seed of SMX expert iteration training and show raw performance on 1280 games for the prior policy and search policy. We show the solve rate for Rubik's Cube and Sokoban and the raw return for the Brax environments in Tables 4 and 5.

Performance in continuous environments remains strong without normalisation however for the discrete environment learning completely collapses and we see no performance on Sokoban or Rubik's cube.

### B.4. Hardware

Training was performed using a mixture of Google v4-8 and v3-8 TPUs. Each experiment was run using a single TPU and only v3-8 TPUs were used to compare wallclock time.

## C. Learning Algorithms

### C.1. Expert Iteration

We describe the basics of the practical implementation of ExIt in algorithm 2. In practice, a buffer is initialised as a list to store training data. We do this using Flashbax (Toledo et al., 2023). For data collection, the expert policy $\pi'(\cdot|s_t, \theta)$ is run for a set of environments and the resulting trajectories are recorded. Each batch of data consists of the environment state $s_t$, the expert policy and the GAE value target of the state $v'_t$ for each timestep of the recorded trajectory. After data collection, the parameters of the prior policy are updated based on the training data collected by the expert.

**Algorithm 2** Expert-Guided Bootstrapping Algorithm

$B \leftarrow []$ # Initialize buffer as an empty list
**for** $i \in \{1, \dots, I\}$ **do**
  *# Collect Data and add to Buffer*
  **for** env $\in$ range($n$-envs) **do**
    **for** $t \in \{j, \dots, j+m\}$ **do**
      $a_t \sim \pi'(\cdot|s_t, \theta)$
      $s_{t+1}, r_t \leftarrow \text{env}(s_t, a_t)$
    **end for**
    $v' \leftarrow GAE(\tau, \theta)$
    **for** $t \in \{j, \dots, j+m\}$ **do**
      $B \leftarrow B \cup \{(s_t, \pi'(\cdot|s_t, \theta), v'_t)\}$
    **end for**
  **end for**
  *# Update Parameters using Buffer*
  $D_i \sim B$
  **for** batch $\in D_i$ **do**
    $\theta \leftarrow L(batch, \theta)$
  **end for**
**end for**

### C.1.1. Policy Distillation Loss

For the discrete case: SMX, given a prior policy and value function, constructs an improved target policy. As in AlphaZero, we minimise the Kullback-Leibler (KL) divergence between the improved target policy $\pi'$ and the current policy $\pi_\theta$. The target policy $\pi'$ reflects the normalised frequency of initial actions contained by the final set of particles. The training objective is to minimize the Kullback-Leibler divergence:

$$\mathcal{L}_\pi = \text{KL}(\pi' || \pi_\theta) = \mathbb{E}_{\pi'}[\log(\pi'(x))] - \mathbb{E}_{\pi'}[\log(\pi_\theta(x))]$$

For continuous environments: the policy distillation occurs similarly to Sampled MuZero whereby the current policy is trained to minimise the negative log-likelihood of the action samples of the prior policy weighted by the normalised visit count which in SMX's case is the frequency of $a^i$ in the set of the final $K$ actions outputted by the search:

$$\mathcal{L}_\pi = -\sum_{i=0}^{K} \log(\pi_\theta(a^i)) * f(a^i) = -\mathbb{E}_{\pi'}[\log(\pi_\theta(x))]$$

If we consider the normalised frequency $f(a^i)$ as the probability of sampling action $a^i$ from the target distribution $\pi'(a^i)$ then the gradient derived from this loss is equal to the gradient of the discrete loss since the expectation $\mathbb{E}[\log(\pi'(x))]$ falls away.

### C.1.2. Value Loss

The variability in reward scales across different environments significantly complicates the process of identifying a universally effective set of hyperparameters. A variety of loss functions and the use of normalisation statistics have been implemented to address these challenges, yet they often introduce new complications. In this study, we have adopted the symlog transform as a strategic solution to bypass the necessity for reward scaling or clipping. This approach also enables the use of a consistent critic architecture across all environments. For a detailed understanding of the symlog transform, refer to (Hafner et al., 2023).

Further aligning with the methodologies employed in (Bellemare et al., 2017; Imani & White, 2018; Schrittwieser et al., 2020; Hafner et al., 2023), our approach incorporates discrete regression combined with two-hot encoded targets for the critic's learning process. Prior to the application of the two-hot encoding, we apply the symlog transformation to scalar values. This integration of the symlog transform with two-hot encoding assists the learning efficiency and adaptability of the critic across diverse environment domains. We leverage the same value loss for all algorithms that train a value function or critic.

### C.1.3. Hyperparameters

Table 6: Hyperparameters of the general Expert iteration Algorithm for Sokoban, Rubik's Cube, and Brax used for both AlphaZero and SMX

| Parameter | Sokoban | Rubik's Cube | Brax |
|---|---|---|---|
| Learning Rate | $3e^{-4}$ | $3e^{-4}$ | $3e^{-4}$ |
| Discount Factor | 0.99 | 0.99 | 0.99 |
| Lambda | 0.95 | 0.95 | 0.95 |
| Replay Buffer Size | $6.5^4$ | $6.5e^4$ | $6.5e^4$ |
| Batch Size | $6.5^4$ | $6.5^4$ | $6.5e^4$ |
| Mini Batch Size | 1024 | 1024 | 1024 |
| Number of Epochs | 1 | 1 | 1 |
| Number of Envs | 1024 | 1024 | 1024 |
| Number of Steps | 64 | 64 | 32 |
| Observation Normalisation | True | True | True |
| Value Head | Symlog | Symlog | Symlog |

For expert iteration we use the same parameters listed in 6 for both AlphaZero and SMX ensuring a fair comparison to isolate the impact of search on performance. We use the same parameters for discrete environments and only change the rollout steps for the continuous case for stability.

### C.2. Sampled Muzero

Alpha/Mu-Zero (Silver et al., 2017; Schrittwieser et al., 2020) are high-performing model-based RL algorithms, known for their integration of search into training. Despite its success, the standard implementation of both AlphaZero and MuZero primarily targets environments characterized by discrete action spaces. This specificity arises mainly due to its reliance on MCTS. The adaptation of these algorithms to continuous action spaces presents certain challenges. One approach involves discretising the action space (Tang & Agrawal, 2020), but this method is highly limiting and continuous control problems often require precision that would be impossible after discretising. Moerland et al. (2018) introduce a method that involves a variety of modifications to Alpha/Mu-Zero that increase the difficulty of implementation with performance only validated on the Pendulum-v0 toy environment.

To address these challenges, Hubert et al. (2021) proposed Sampled MuZero, a variation of the original MuZero algorithm that effectively operates in continuous action spaces. This adaptation maintains the essence of MuZero but introduces a novel approach: it employs sampled actions for search, thus enabling the algorithm to handle the complexity of continuous spaces. This extension is a significant advancement that makes Sampled MuZero a more suitable and efficient choice for continuous domains.

In our implementation of Sampled MuZero, we sought to maintain a direct comparison with the original framework however we utilized a parameterised Gaussian distribution as the prior policy, denoted as $\pi_\theta$. For the base implementation of MuZero's MCTS search method, we utilized MCTX [4]. As stated in (Hubert et al., 2021), the core modification for the approach involved sampling $K$ actions from the prior policy $\pi_\theta$ during the search phase, treating these sampled actions akin to a discrete set within MCTX. During environment steps, such as node expansion, we indexed into this set of sampled actions. A crucial aspect of our adaptation in MCTX was the adjustment of probabilities in the PUCT formula to align with the sample-based policy. Given that actions sampled from a Gaussian distribution are unique, we adopted a uniform probability distribution for the prior probability values used in the PUCT formula. This is equivalent to the policy $\hat{\pi}_\beta(s, a)$ explained in the original paper. This decision is taken to mitigate any potential double counting in the search algorithm and is stated as necessary in the paper. Lastly, we add noise to our sampled actions at the root node by sampling noise from a truncated Gaussian on an interval of $[-1, 1]$ and adding this noise scaled by an exploration coefficient $n_c = 0.1$.

### C.3. PPO

In this study, we employed a conventional implementation of Proximal Policy Optimization (PPO) (Schulman et al., 2017). For the implementation of the PPO loss, we utilized the version provided by the rlax library (DeepMind et al., 2020), which can be found at `https://github.com/google-deepmind/rlax`. We use the same value loss as in the expert iteration.

Table 7: Hyperparameters of the PPO Algorithm for Sokoban, Rubik's Cube, and Brax

| Parameter | Sokoban | Rubik's Cube | Brax |
|---|---|---|---|
| Learning Rate | $3e^{-4}$ | $3e^{-4}$ | $3e^{-4}$ |
| Discount Factor | 0.99 | 0.99 | 0.99 |
| Lambda | 0.95 | 0.95 | 0.95 |
| Batch Size | $6.5^4$ | $6.5^4$ | 81920 |
| Mini Batch Size | 1024 | 1024 | 5120 |
| Number of Epochs | 4 | 4 | 4 |
| Number of Envs | 1024 | 1024 | 16384 |
| Number of Steps | 64 | 64 | 5 |
| Observation Normalisation | True | True | True |
| Value Head | Symlog | Symlog | Symlog |

The hyperparameters listed in Table 7 were adjusted for the continuous environments, drawing upon previous implementations of PPO on the Brax benchmark (Freeman et al., 2021b).

[4] https://github.com/google-deepmind/mctx/

### C.4. M-DQN

We include Munchausen-DQN (M-DQN) (Vieillard et al., 2020) in our benchmarks in order to validate against a modern method based on the maximum entropy framework for discrete tasks. We base our implementation on the original M-DQN repo found here https://github.com/google-research/google-research/tree/master/munchausen_rl. As our work was carried out using JAX and utilised a fully jittable training loop, we replaced the default replay buffers with JAX-based prioritised experience replay buffers found in (Toledo et al., 2023) which can be found at https://github.com/instadeepai/flashbax.

Table 8: Hyperparameters of the M-DQN Algorithm for Sokoban and Rubik's Cube

| Parameter | Sokoban | Rubik's Cube |
|---|---|---|
| Learning Rate | $7e^{-4}$ | $7e^{-4}$ |
| Discount Factor | 0.95* | 0.97* |
| Replay Buffer Size | $6.5e^{6}$ | $6.5e^{6}$ |
| Batch Size | $6.5^{4}$ | $6.5^{4}$ |
| Mini Batch Size | 2048 | 2048 |
| Burn in iterations | 50 | 50 |
| spi | 1 | 1 |
| number of envs | 1024 | 1024 |
| number of steps | 64 | 64 |
| Observation Normalisation | True | True |

**Note: A discount factor of 0.99 for M-DQN led to highly unstable training. Lower discount factors were found to stabilize learning across seeds.*

M-DQN hyperparameters are listed in Table 8. "Burn in iterations" refers to the number of epochs run to prefill the replay buffer with data before training begins and "spi" refers to the sample-to-insert ratio used during training.

### C.5. SAC

The Soft Actor-Critic (SAC) algorithm is an off-policy, actor-critic reinforcement learning method designed for continuous action spaces. Similar to M-DQN, it adheres to the maximum entropy framework, promoting both effective exploration and robustness, while utilising a replay buffer for enhanced sample efficiency. Our SAC benchmark is based on the (Hoffman et al., 2020) implementation found here https://github.com/google-deepmind/acme/tree/master/acme/agents/jax/sac. Similar to our implementation of M-DQN, we use a JAX-based, jitted replay buffer in order to jit compile our training loop end-to-end.

Table 9: Hyperparameters of the Soft Actor Critic Algorithm for the Brax environments

| Parameter | Brax Environments |
|---|---|
| Learning Rate | $5e^{-4}$ |
| Discount Factor | 0.99 |
| Replay Buffer Size | $1e^{6}$ |
| Batch Size | $8.2e^{4}$ |
| Mini Batch Size | 512 |
| Number of Envs | 256 |
| Number of Steps | 5 |
| Observation Normalisation | True |

SAC hyperparameters are listed in Table 9. "Burn in iterations" were not used.

# D. Search Algorithms

## D.1. SMX

### D.1.1. HYPERPARAMETERS

Table 10: Hyperparameters of SMX for main results

| Parameter | Value |
|---|---|
| Depth | 4 |
| Num particles | 16 |
| Resampling Period | 4 |
| Resampling Temp | 0.1 |
| Policy Temp | 1 |
| Root Policy Temp | 1 |
| Dirichlet Alpha | 0.03 |
| Dirichlet Weight | 0.25 |
| Normalisation | min max |

Table 11: Hyperparameters of SMX for scaling results

| Parameter | Value |
|---|---|
| Depth | [4, 8, 16] |
| Num particles | [4, 8, 16, 32, 64] |
| Resampling Period | 4 |
| Resampling Temp | 0.1 |
| Policy Temp | 1 |
| Root Policy Temp | 1 |
| Dirichlet Alpha | 0.03 |
| Dirichlet Weight | 0.25 |
| Normalisation | min max |

Table 12: Hyperparameters of SMX for wall-clock results

| Parameter | Value |
|---|---|
| Depth | [2, 4, 8, 16] |
| Num particles | [4, 8, 16, 32, 64, 128, 256, 512] |
| Resampling Period | 2 |
| Resampling Temp | 0.1 |
| Policy Temp | 1 |
| Root Policy Temp | 1 |
| Dirichlet Alpha | 0.03 |
| Dirichlet Weight | 0.25 |
| Normalisation | min max |

SMX hyperparameters for the main experiments are listed in Table 10 and parameters for the scaling experiments are listed in Table 12.

## D.2. AlphaZero

We utilized the official JAX-based implementation of MuZero for our baseline, ensuring correctness. We ensure to use of the default, already highly optimized, parameters which have demonstrated effective performance across a range of environments. For more details, see the MuZero implementation at `https://github.com/google-deepmind/mctx`.

Table 13: Hyperparameters of AlphaZero search

| Parameter | Value |
|---|---|
| Search Budget | 50 |
| Dirichlet Fraction | 0.25 |
| Dirichlet Alpha | 0.3 |
| Pb C Init | 1.25 |
| Pb C Base | 19652 |
| Temperature | 1.0 |
| Q-Normalisation | by parent and siblings |

We use the same hyperparameters prescribed in the original MuZero paper as listed in Table 13.