PARTICLE MONTE CARLO TREE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Monte Carlo Tree Search (MCTS) is a widely used approach for policy improvement and action selection in Reinforcement Learning. Due to its sequential and deterministic nature, principled runtime-scaling of MCTS with parallel compute remains a major challenge. We introduce Particle MCTS (PMCTS), a parallel MCTS algorithm which is suited for neural network evaluations, designed for GPU-acceleration with batch-parallelization and retains MCTS's principled approximate policy improvement interpretation. Empirically, PMCTS scales well with parallel compute and consistently outperforms or compares well to the popular heuristic-based baselines across a range of popular discrete- and continuous-action benchmark domains, including Chess, 19x19 Go, 9x9 Go, Gardner Chess, Snake, classical control environments from Brax and LLM reasoning in Sokoban.

1 INTRODUCTION

Monte Carlo Tree Search (MCTS, Kocsis & Szepesvári, 2006) is a popular search algorithm that plays a central role in modern reinforcement learning (RL). MCTS performs *policy improvement* and action selection through lookahead and evaluation (i.e., *search*) in a model of the dynamics of the environment. This paradigm has been popularized in RL by the AlphaZero family of algorithms (Silver et al., 2016; 2018) and has since been successfully applied to a wide range of domains, including zero-sum games (Silver et al., 2018), discrete-action, single-agent tasks (Schrittwieser et al., 2020; Fawzi et al., 2022; Mankowitz et al., 2023), continuous control (Wang et al., 2024) and LLM inference scaling (Hao et al., 2023; Wan et al., 2024; Shi et al., 2025). This wide applicability of MCTS makes its inference-time scaling with parallel compute a critical challenge and the subject of significant attention (Cazenave & Jouandeau, 2007; Chaslot et al., 2008; Schäfers, 2014; Liu et al., 2020; Yang et al., 2021; Cazenave, 2022).

Parallelizing MCTS is challenging for two main reasons: (I) In each iteration, MCTS selects a trajectory for evaluation using conventionally deterministic selection policies that depend on statistics from previous iterations in a deterministic dynamics model (Kocsis & Szepesvári, 2006; Rosin, 2011; Danihelka et al., 2022). This makes the *principled* selection of multiple *unique* trajectories in parallel a significant challenge, because parallel selection steps result in the same trajectory. (II) Modern MCTS methods use a deterministic deep neural network (DNN) for evaluation. This makes scaling the evaluation of a *single* trajectory with parallel compute a significant challenge, because repeated evaluations of the same trajectory result in duplicate evaluations rather fresh evaluations which average out error. Together, these properties also make independent instances of MCTS at the same state (*root parallelization*) expand similar trees and return similar search results, renders root parallelization as ineffective for inference-time scaling and makes the selection of multiple unique trajectories in parallel an important challenge.

To overcome these challenges popular methods use heuristic selection policies such as *Virtual Losses* (Chaslot et al., 2008) and *Virtual Means* (Cazenave, 2022) which rely on communication between processes during selection. These methods suffer from three main limitations: First, their heuristic selection policy does not have a clear, principled interpretation as an improved policy or as tracking an improved policy. Second, communication during selection hamstrings parallelization and precludes batch-parallelization of the complete search. Batched-parallelization is an effective and popular approach for GPU-acceleration in RL specifically and Deep Learning in general (Bradbury et al., 2018). Third, they do not account for the effect of duplicate trajectories on the evaluation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To address these limitations, in this work we introduce *Particle MCTS* (PMCTS), a novel parallel MCTS algorithm which retains modern MCTS's principled approximate policy improvement interpretation (Grill et al., 2020), is well suited for GPU acceleration through batch-parallelization and explicitly accounts for the presence of duplicate trajectories. PMCTS samples multiple trajectories independently in parallel from a theoretically motivated stochastic selection policy; explicitly controls the entropy of the selection policy to increase trajectory diversity; uses Sequential Monte Carlo (SMC, see Chopin & Papaspiliopoulos, 2020) within the framework of MCTS to maintain principled estimates of the expected return with respect to the intended target policy and account for the presence of remaining duplicate trajectories; uses the Effective Sample Size (ESS, see Chopin & Papaspiliopoulos, 2020) to estimate the variance of the SMC expectation estimator and maintain principled updates to the search tree; and last, uses importance sampling to mitigate the effect of catastrophic returns observed during evaluation of new nodes in the tree.

We show that Simple PMCTS and PMCTS retain MCTS's principled approximate policy-improvement interpretation and characterize the manner in which increased parallelization improves evaluation. Empirically, PMCTS demonstrates strong inference-time scaling, outperforming or comparing well to the popular, heuristic-based baselines across a range of popular benchmark domains, including hard zero-sum games (chess, 19x19 Go, 9x9 Go, Gardner chess), discrete-action, single-agent tasks (Snake and RubiksCube (Bonnet et al., 2024)), classical continuous control (Ant, HalfCheetah and Humanoid (Freeman et al., 2021)) and an LLM inference scaling task, scaling the capacity of the LLM to play the game of Sokoban (Bonnet et al., 2024). PMCTS provides a promising new direction for scaling runtime inference with MCTS and parallel compute, which is principled for policy improvement and action selection and well suited for modern GPU acceleration.

2 BACKGROUND

In RL, the environment is represented by a Markov Decision Process (MDP, Bellman, 1957) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \rho, R, P, \gamma \rangle$. $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, both of which we consider finite for presentation and analysis simplicity. ρ is an initial state distribution, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a bounded possibly stochastic reward function and P is a transition distribution $s' \sim P(\cdot|s, a)$. The policy of the agent $\pi \in \Pi$ is defined as a distribution over actions $a \sim \pi(s)$ with probabilities specified by $\pi(a|s)$. The objective J_π is the maximization of the expected discounted return, or state-value $V^\pi(s)$:

$$J_\pi = \mathbb{E}_\rho\big[V^\pi(s_1)\big] = \mathbb{E}\Big[\textstyle\sum_{t=1}^{H} \gamma^{t-1} R(s_t, a_t)\Big], \quad s_1 \sim \rho,\ a_t \sim \pi(s_t),\ s_{t+1} \sim P(s_t, a_t), \quad (1)$$

over the initial state distribution ρ. The discount factor $0 < \gamma < 1$ is used when $H \to \infty$ to guarantee that V^π remains bounded, or for stability; otherwise, usually $\gamma = 1$. A state-action *Q-value function* is defined as follows: $Q^\pi(s, a) = \mathbb{E}[R(s, a) + \gamma V^\pi(s')]$.

2.1 SEARCH FOR POLICY IMPROVEMENT

RL algorithms which have access to a dynamics model of the environment $\mathcal{M} = (P, R)$ (exact or approximate) often use *search* to drive the iterative improvement of the acting policy (Moerland et al., 2023). Search refers to the process of look-ahead in the model from a state s, often using a DNN-based prior policy π_θ and state value function v_ϕ, in order to extract an improved policy $\pi_{\text{search}}(s)$ at the current state s. A popular example is AlphaZero which uses MCTS for that purpose.

For exact action values Q^π, a policy improvement operator $\mathcal{I} : \Pi \times \mathcal{Q} \to \Pi$ maps π to a policy satisfying $V^{\mathcal{I}(\pi, Q^\pi)}(s) \geq V^\pi(s)$ at every state (i.e. policy improvement). We define $\mathcal{Q}$ as the set of bounded state-action functions, to indicate that $\mathcal{I}$ can also be applied to approximate search values $q \in \mathcal{Q}$; improvement in the search objective with such values does not by itself guarantee improvement in true MDP value however.

Search algorithms generally return: (I) A selected action a, (II) An improved policy $\pi_{\text{search}}(s)$ and (III) A value $V_{\text{search}}(s) \approx V^{\pi_{\text{search}}}(s)$. $\pi_{\text{search}}(s)$ is used to train π_θ with cross entropy loss and $V_{\text{search}}(s)$ is used to produce bootstraps for TD-targets (Schrittwieser et al., 2020) or even value targets directly (Oren et al., 2025a).

2.2 MONTE CARLO TREE SEARCH (MCTS)

The MCTS algorithm maintains a search tree over states with its root $s =: s_1$ representing the current state s in the environment. Traditionally, MCTS is used for search with an exact, deterministic model $\mathcal{M}$ and finite state-action environments. For simplicity of presentation, we will maintain these assumptions hence. However, MCTS has been used successfully for policy improvement and action selection with learned (Schrittwieser et al., 2020) and stochastic transition (Antonoglou et al., 2022) models as well as in continuous state-action spaces (Hubert et al., 2021; Wang et al., 2024). Our experiments also use sampled continuous actions, whereas the formal results in Section 4 assume finite actions and exact deterministic dynamics. The tree is constructed by iterating (I) *selection*, (II) *expansion* and (III) *backpropagation* M times (often called *number of expansions*, *simulations* or the *search budget*):

Selection: At iteration i, a trajectory $\tau_T^i = (s_1^i, a_1^i, \dots, s_T^i, a_T^i)$ in the tree is selected using a *selection policy* π_i. Traditionally, MCTS uses deterministic selection policies such as π_i^{UCT} or π_i^{PUCT} (Kocsis & Szepesvári, 2006; Rosin, 2011; Silver et al., 2016, see Appendix H.1 for additional detail). π_i uses the current state-action evaluation q_i, the number of prior backpropagation steps $M_i(s_t)$ through each node s_t (and $M_i(s_t, a)$ which can be computed using $M_i(s_{t+1})$ of the node associated with the transition (s_t, a)) and often a *prior policy* π_θ. Grill et al. (2020) show that the visit distributions induced by UCT and PUCT track solutions of regularized policy-optimization problems defined by the search values q_i. They also evaluate selecting directly from the corresponding improved policy, obtaining better performance in their experiments. A canonical modern version, GumbelMCTS (Danihelka et al., 2022) uses the operator $\mathcal{I}_{\text{reg}}$:

$$\pi_i(a|s_t) = \mathcal{I}_{\text{reg}}(\pi_\theta, q_i)(a|s_t) \propto \exp(\log \pi_\theta(a|s_t) + \beta_i(s_t) q_i(s_t, a)). \tag{2}$$

GumbelMCTS uses the heuristic $\beta_i(s_t) = (c_{\text{visit}} + \max_{a \in \mathcal{A}} M_i(s_t, a)) c_{\text{scale}}$, inspired by π_i^{PUCT}, which uses the number of visits $M_i(s_t, a)$ as an estimate for the inverse-variance of the evaluation. To compute π_i, an evaluation $q_i(s_t, a)$ is needed for *all* actions, including those that have not yet been searched. To achieve this, GumbelMCTS uses completed Q values, assigning $q_i(s_t, a) = v_{\text{mix}}(s_t)$ when $M_i(s_t, a) = 0$ (Danihelka et al., 2022). The mixed value interpolates $v_\phi(s_t)$ with the π_θ-weighted mean of visited action values according to the total visit count, as defined in Appendix H.2. In our experiments, we adhere to the design choices of GumbelMCTS whenever possible and unless specified otherwise.

Expansion: When selection arrives at an unobserved transition (s_T, a_T) (a *leaf*), it is *expanded*. This gathers information from the model: $r_T = \mathbb{E}[R(s_T, a_T)], s_{T+1} \sim P(s_T, a_T), v_{i+1}(s_{T+1}) \approx V^{\pi_\theta}(s_{T+1})$. In classical MCTS, the new node is evaluated using the average returns of K rollouts in the model, sampled i.i.d. from state s_{T+1} using a *rollout policy*. Following AlphaGo (Silver et al., 2016), modern variants use a DNN $v_{i+1}(s_{T+1}) := v_\phi(s_{T+1})$ instead.

Backpropagation: The bootstrapped return of the selection trajectory $\nu_i(s_t) = \sum_{j=0}^{T-t} \gamma^j r_j + \gamma^{T-t+1} v_{i+1}(s_{T+1})$ is propagated backwards through all nodes s_t along the trajectory. The nodes maintain the incoming reward $r(s_{t-1}, a_{t-1})$ and the mean of returns from simulations that visited it:

$$v_{i+1}(s_t) = \frac{1}{M_{i+1}(s_t)} \sum_{j=1}^{M_{i+1}(s_t)} \nu_j(s_t), \quad q_{i+1}(s_t, a_t) = r(s_t, a_t) + \gamma v_{i+1}(s_{t+1}). \tag{3}$$

MCTS returns an improved policy, an action to take in the environment and additional statistics such as the value at the root and of actions at the root, as used by the learning algorithm. The exact improved policy, action selection and value targets used by different MCTS variants, as well as PMCTS, are specified in Appendix H.2.

2.3 SEQUENTIAL MONTE CARLO FOR SEARCH FOR POLICY IMPROVEMENT IN RL

SMC methods use *sequential importance sampling* (SIS) to approximate a sequence of *target distributions* using *proposal distributions* (Chopin & Papaspiliopoulos, 2020). In RL, SMC methods have been adapted for search for policy improvement (Piché et al., 2019) by formulating the proposal and target as distributions over trajectories τ_t using π_θ as the *proposal policy* and $\pi'(s_t) = \mathcal{I}(\pi_\theta, Q^{\pi_\theta})(s_t)$ as the *target policy* (Oren et al., 2026). At each time step $t \in \{1, \dots, T\}$, N particles, each comprising a trajectory and a weight $\{\tau_t^n, w_t^n\}_{n=1}^N$, are updated via *sampling* and *correction*. *Sampling*: each trajectory τ_{t-1}^n is extended by sampling $s_{t+1}^n \sim P(\cdot|s_t^n, a_t^n), a_t^n \sim \pi_\theta(\cdot|s_t^n), s_1^n = s$, where s is the

current state in the environment. *Correction*: The weights are corrected using SIS (left) such that the weighted particles $\{\tau_t^n, w_t^n\}_{n=1}^N$ provide a consistent approximation of expectations under the target:

$$w_t^n = w_{t-1}^n \frac{\pi'(a_t^n|s_t^n)}{\pi_\theta(a_t^n|s_t^n)}, \quad \frac{1}{\sum_{n=1}^N w_t^n} \sum_{n=1}^N w_t^n \left(\mathcal{R}(\tau_t^n) + \gamma^t V^{\pi_\theta}(s_{t+1}^n)\right) \xrightarrow[N\to\infty]{\mathbb{P}} V^{\pi'}(s_1). \quad (4)$$

We refer to Oren et al. (2026) for derivation and Appendix C.3 and Chopin & Papaspiliopoulos (2020) for more detail.

3 Particle Monte Carlo Tree Search

We design Particle Monte Carlo Tree Search (PMCTS) to achieve three desiderata: (I) Theoretically motivated selection and evaluation to approximate principled policy improvement. (II) Fully independent parallel selection of multiple trajectories, while encouraging trajectory diversity. (III) Correctly account for the presence of remaining duplicate trajectories, to provide principled updates to the tree. We begin by designing a simple, principled, particle-based batch-parallel variant of MCTS, which retains modern MCTS's interpretation of approximate policy improvement (Grill et al., 2020) under the classical assumptions of exact deterministic dynamics and conditionally independent, unbiased, finite-variance leaf evaluations (Kocsis & Szepesvári, 2006). Next, we identify the challenges that arise from using DNN evaluations which do not satisfy these classical assumptions. We then proceed to present PMCTS, which addresses these challenges and achieves the set desiderata.

3.1 A simple, parallel, particle based Monte Carlo Tree Search algorithm

We begin with an observation: the principled stochastic selection policies proposed by Grill et al. (2020), such as π_i (Equation 2), permit multiple trajectories to be sampled independently in parallel which can produce distinct trajectories without communication between particles. Motivated by this observation, we design a simple particle-based MCTS algorithm which preserves the policy improvement interpretation of MCTS and reduces evaluation variance. The algorithm operates in the same way as baseline MCTS, except that each step is now batch-parallelized over particles.

Particle-based selection: At each iteration i, N trajectories are sampled independently in parallel from the stochastic selection policy π_i (Equation 2).

Particle-based expansion: N nodes s_{T+1}^n are expanded in batch-parallel:

$$\left\{s_{T+1}^n \sim P(\cdot|s_T^n, a_T^n), \quad r(s_T^n, a_T^n) = \mathbb{E}[R(s_T^n, a_T^n)], \quad v_i(s_{T+1}^n), \quad \pi_\theta(s_{T+1}^n)\right\}_{n=1}^N. \quad (5)$$

Particle-based backpropagation: We now need to compute the particle-based return ν_i and update to the tree v_{i+1}. Under the classical assumptions, the returns through s_t are identically distributed and independent, with mean $V^{\pi_i}(s_t)$ and finite variance $\sigma_i^2(s_t)$. Thus the mean $\nu_i(s_t)$ is conditionally unbiased and has variance $\sigma_i^2(s_t)/N(s_t)$. We weight the update to the tree by $N(s_t)$ to maintain v_{i+1} 's interpretation as an estimator of the mean across evaluations.

$$\nu_i(s_t) = \frac{1}{N(s_t)} \sum_{n=1}^N \mathbb{1}_{s_t \in \tau_T^n} \nu^n(s_t), \quad v_{i+1}(s_t) = v_i(s_t) + \frac{(\nu_i(s_t) - v_i(s_t))N(s_t)}{M_{i+1}(s_t)}, \quad (6)$$

and $M_{i+1}(s_t) = M_i(s_t) + N(s_t)$. We summarize Simple PMCTS in Algorithm 1 in Appendix B. Action selection and policy improvement follow the description in Section 2, see Appendix H.2 for more detail. Simple PMCTS is principled and batch-parallelizable, addressing the first and second limitations of prior methods, but does not yet account for the combined effect of duplicate trajectories and DNN evaluations.

3.2 Enter Deep Neural Networks

With DNN evaluations, the returns $\nu_i^n(s_t)$ cannot be assumed to be entirely uncorrelated or unbiased. This is a general problem in search algorithms such as MCTS, SMC for search (Piché et al., 2019; Oren et al., 2026) and MPPI (Hansen et al., 2024). Despite this, these algorithms are extremely successful in practice, implicitly assuming that *unique* evaluations are *sufficiently* uncorrelated and

unbiased. In discrete-action, parallel search algorithms this problem is significantly exacerbated. *Multiple* parallel selection trajectories can arrive at *the same* leaf s_{T+1}, even when π_i is stochastic. In this case, all trajectories where $s^n_{T+1} = s_{T+1}$ will have *fully* correlated returns. This causes two main problems: (I) Duplicate leaf evaluations do not reduce the evaluation's variance and waste compute. (II) The variance $\mathbb{V}[\nu_i(s_t)]$ cannot be assumed to be proportional to $1/N(s_t)$ resulting in *incorrectly weighted backpropagation*. We proceed to extend Simple PMCTS with mechanisms that address problems (I-II) and design the complete PMCTS algorithm.

3.3 Weighted Particle Monte Carlo Tree Search (PMCTS)

Addressing problems (I-II) requires the algorithm to reduce the number of duplicate trajectories while accounting for the remaining duplicate trajectories and estimating V^{π_i} before retrospective reweighting and the induced post-expansion trajectory target afterward. To achieve that, we use SMC within PMCTS, with four mechanisms: *higher-entropy sampling* to increase the number of unique trajectories; *deduplication* to mitigate the effect of remaining duplicate trajectories and maintain a principled estimate of the selection policy's value; and the *Effective Sample Size (ESS)* (Chopin & Papaspiliopoulos, 2020) for principled backpropagation. In addition, SMC allows the algorithm to address a more general problem in MCTS, over-commitment to catastrophic returns, through a fourth mechanism we call *retrospective reweighting*. The resulting algorithm is PMCTS, which is described below. PMCTS maintains the same structure of particle-based selection, expansion, backpropagation, except that each particle n is now associated with a weight w^n_t which is updated during selection.

Weighted particle-based selection with *higher-entropy sampling*: To increase trajectory diversity, actions are sampled from a higher-entropy proposal policy $\hat{\pi}_i$ rather than π_i. The particle's weight is computed using SIS to maintain a principled estimate of the expectation with respect to π_i:

$$w^n_t = w^n_{t-1} \frac{\pi_i(a^n_t \mid s^n_t)}{\hat{\pi}_i(a^n_t \mid s^n_t)}, \qquad \hat{\pi}_i(a_t|s_t) \propto \pi_i(a_t|s_t)^{\frac{1}{\eta(s_t)}}. \tag{7}$$

This mechanism allows the algorithm to trade off between two sources of variance: the effect of $\hat{\pi}_i$ diverging from π_i and the effect of reduced duplicates to minimize the total variance of the update.

For simplicity, in this work we use a constant rather than a dynamic $\eta(s) := \eta$. In general, any policy $\hat{\pi}_i$ whose support contains the target π_i's can be used. If particles are allowed to communicate at the cost of parallelization, the temperature can be chosen based on the number of particles per node to reduce variance. SMC resampling techniques can additionally be incorporated for further variance reduction (Chopin & Papaspiliopoulos, 2020). However, resampling requires communication and can reduce trajectory diversity; we leave its investigation in PMCTS to future work.

Weighted particle-based backpropagation with *deduplication* and *ESS*: After expansion, PMCTS aggregates all duplicate trajectories into a single particle while preserving the correct total weight: $w^m_T(s_{T+1}) = \sum_{n=1}^N w^n_T \mathbb{1}_{s_{T+1}=s^n_{T+1}}$, which we call *deduplication*. This results in a new set of particles where all trajectories are unique while correctly preserving the estimate of the expectation with respect to π_i. We will refer to this new particle set using the same notation $\{\tau^n_T, w^n_T\}_{n=1}^N$. Deduplication is naturally only used when duplicate trajectories have duplicate evaluations, such as with deterministic DNN evaluations. The backup $\nu_i(s_t)$ is computed using the particles' normalized weights $\bar{w}^n_T(s_t)$ and returns $\nu^n(s_t)$:

$$\nu_i(s_t) = \sum_{n=1}^N \underbrace{\bar{w}^n_T(s_t)\nu^n(s_t)\mathbb{1}_{s_t\in\tau^n_T}}_{\text{Weighted particle returns}}, \qquad \bar{w}^n_T(s_t) = \underbrace{\frac{w^n_T \mathbb{1}_{s_t\in\tau^n_T}}{\sum_{m=1}^N w^m_T \mathbb{1}_{s_t\in\tau^m_T}}}_{\text{Normalized per node}}. \tag{8}$$

To update v_{i+1} such that it remains an inverse-variance weighted estimator of the mean, we need to estimate the variance $\mathbb{V}[\nu_i(s_t)]$. A standard choice is the inverse ESS (Chopin & Papaspiliopoulos, 2020) $\mathbb{V}\left[\nu_i(s_t)\right] \approx \frac{1}{\text{ESS}(s_t)} = \sum_{n=1}^N (\bar{w}^n_T(s_t))^2$. Thus:

$$v_{i+1}(s_t) = v_i(s_t) + \frac{(\nu_i(s_t) - v_i(s_t))\text{ESS}(s_t)}{M_{i+1}(s_t)}, \qquad M_{i+1}(s_t) = M_i(s_t) + \text{ESS}(s_t), \tag{9}$$

and $M(s_{T+1}) = 1$. See Appendix C.2 for more detail and derivation.

Retrospective reweighting: SMC additionally enables PMCTS to address a third problem, general to MCTS: over-commitment to catastrophic returns (see Jaldevik, 2024). At parent s_T, MCTS samples $a_T \sim \pi_i(s_T)$ leading to evaluation $q_i(s_T, a_T) = r_T + \gamma v_\phi(s_{T+1})$. The action a_T may be catastrophic in terms of its true value $Q^{\pi_\theta}(s_T, a_T)$ (for example, falling off a cliff), which cannot be known in advance: the purpose of the expansion is to gather new information. In standard MCTS variants, this catastrophic evaluation will then be backpropagated to the root, unnecessarily contaminating all nodes along the trajectory even if the catastrophic action is entirely avoidable. To address this problem, after expansion and before backpropagation, we compute the "retrospective" post-expansion policy at each expanded parent s_T and replace the final action's importance ratio using $\pi_{i+1}(a_T \mid s_T)$:

$$w_T^n = w_{T-1}^n \frac{\pi_{i+1}(a_T^n \mid s_T^n)}{\hat{\pi}_i(a_T^n \mid s_T^n)}, \quad \pi_{i+1}(a_T|s_T) \propto \pi_\theta(a_T|s_T) \exp\big(\underbrace{\beta_{i+1}(s_T) q_{i+1}(s_T, a_T)}_{\text{Post-expansion.}}\big). \tag{10}$$

The corrected weights account for the new action evaluations, induce the post-expansion target and downweigh catastrophic trajectories when their associated action becomes less probable relative to the alternatives represented in the batch.

This completes the PMCTS algorithm, which remains principled and batch parallelizable while accounting for the effects of duplicate trajectories. We summarize PMCTS in natural language in Appendix B.2 for added clarity, along with pseudocode in Algorithm 2.

4 Theoretical Analysis

The main use of MCTS in RL is for policy improvement: search aims to select an action from the best policy it can construct at its root, and, in an AlphaZero-style RL loop, train the prior policy π_θ with this policy as a target. Grill et al. (2020) give MCTS a principled interpretation as approximating policy improvement, showing that canonical selection policies yield visit distributions that track policies improved with respect to tree values q_i and the prior π_θ, and that selection can instead use the tracked policy $\pi_i = \mathcal{I}(\pi_\theta, q_i)$ directly.

Following this interpretation, we establish three theoretical results for PMCTS algorithms. First, that Simple PMCTS and PMCTS retain MCTS's policy-improvement interpretation: each parallel selection and backpropagation evaluates the improved policy π_i directly or by importance weighting, while PMCTS's retrospective correction evaluates the policy improved with respect to the post-expansion evaluation. Second, that with a common selection policy, MCTS is the single-particle case of both algorithms, and the additional particles increase the accuracy of the evaluation. And third, a characterization of the conditions under which PMCTS provides benefits over Simple PMCTS, which relax the classical assumptions specified in Section 3 (namely, unbiased finite variance evaluations and exact, deterministic model).

Theorem 1 (Improved-policy evaluation (informal))**.** *At each iteration, under the classical assumptions, Simple PMCTS unbiasedly estimates V^{π_i}. PMCTS consistently estimates V^{π_i} before retrospective reweighting, and the value of its post-expansion trajectory policy afterward.*

Intuition. Simple PMCTS samples from π_i. PMCTS samples from $\hat{\pi}_i$ and importance-corrects to π_i; its final terminal-action correction changes the policy whose value is backed up.

Theorem 2 (MCTS with increased evaluation accuracy (informal))**.** *With the same stochastic selection policy π_i, MCTS, Simple PMCTS, and PMCTS have the same single-particle update at $N = 1$. Under the classical assumptions, Simple PMCTS's conditional node-evaluation variance is $\sigma_i^2(s_t)/N_i(s_t)$. PMCTS's errors relative to its pre- and post-retrospective targets are $O_{\mathbb{P}}(N^{-1/2})$.*

Intuition. One particle's normalized weight is one, even after retrospective reweighting. Additional particles evaluate the same search policy more accurately. PMCTS uses importance weights to account for sampling from $\hat{\pi}_i$.

Theorem 3 (Benefit of distinct evaluations (informal))**.** *Suppose duplicate trajectories share an evaluation, while the evaluation errors of distinct trajectories are uncorrelated and have equal variance. Before retrospective reweighting, PMCTS has lower evaluation-noise variance than Simple PMCTS exactly when its weighted ESS over distinct trajectories is larger. After reweighting, PMCTS*

also has lower evaluation-noise MSE and variance than Simple PMCTS when its corrected ESS advantage exceeds the effect of adapting its weights to those same evaluations.

Intuition. Repeating a deterministic DNN evaluation of the same trajectory adds no independent information. PMCTS's proposal can reach more distinct trajectories. Its retrospective correction uses their evaluations to change their weights; the formal bound accounts for this dependence, and deduplication preserves the corrected backup.

Together, the results retain MCTS's regularized policy-optimization interpretation while making the parallel backups' targets and evaluation uncertainty explicit. For formal Theorems and proofs see Appendix D. We additionally include a runtime complexity analysis in Appendix E.

5 Related Work

Runtime-inference scaling of MCTS has been a long-standing objective of AI research (Cazenave & Jouandeau, 2007). The canonical approaches are (I) *root parallelization* (running different instances of MCTS in parallel) and (II) *leaf parallelization* (scaling the evaluation of leaves with parallel compute). Both rely on stochasticity in the selection and/or evaluation to scale and are thus unsuited for runtime inference scaling with modern DNN evaluations and deterministic dynamics and selection. (III) *Tree parallelization* (parallelizing the search itself), which canonically uses the *virtual visits* heuristic to incentivize the selection of multiple unique trajectories and circumvent these challenges. When a process arrives at action a in node s, it increments a *virtual visit* $M_i^{\text{virtual}}(s, a)$ (Chaslot et al., 2008). When another process arrives at s, it computes the deterministic selection policy $\pi_i^{\text{PUCT}}(s)$ using $M_i^{\text{virtual}}(s, a)$ and a heuristic which effectively *predicts* the evaluation of the previous processes in the hope that π_i^{PUCT} selects a different action. The selected trajectories are then evaluated in parallel, usually batched, and backpropagated, whether they are unique or not, so repeated deterministic leaf evaluations increase backed-up visit counts without adding independent evaluation information and the resulting visit distribution does not retain the interpretation of approximating policy improvement with respect to the tree's value. In addition, to work as intended this selection requires the processes to operate *sequentially* at each *shared* node, both preventing effective batched-parallelization as well as in the worst case resulting in entirely sequential–rather than parallel–selection (See Appendices F and H.3 for more detail). In contrast, PMCTS selects trajectories independently in parallel and retains the explicit policy-improvement interpretation, established in Theorems 1–3.

Adjacent to MCTS and PMCTS, SMC has been used directly for search for policy improvement (Piché et al., 2019; de Vries et al., 2025; Oren et al., 2026). The resulting SMC-based search algorithms are fundamentally different to MCTS: They maintain statistics only over N particles rather than memorize the entire search tree. As a result, they can only converge to the value of an *improved policy* in the limit of infinite search horizon (Oren et al., 2026), rather than the optimal policy in the case of MCTS (Kocsis & Szepesvári, 2006); their estimates can deteriorate with search depth and limited particle budgets (Oren et al., 2026) as we also demonstrate in Appendix G; and they are not designed to account for duplicate trajectories. In contrast, PMCTS memorizes the entire tree and scales well across search budgets and particle budgets (see Section 6). PMCTS is additionally designed to account for the presence of duplicate trajectories with three mechanisms, *increased-entropy selection*, *deduplication*, and *the ESS*, which are novel to this line of work.

6 Experiments

Our objective in this work is to present a novel and effective tree-parallel MCTS algorithm which is both principled as well as suited for batch-parallelization and runtime scaling on the GPU. In this section, we evaluate PMCTS empirically across three popular MCTS and RL benchmark domains and one LLM inference scaling experiment. These encompass: (I) Zero-sum board games (*chess, 19x19 Go, 9x9 Go* and *Gardner chess)* (Koyamada et al., 2023); (II) Popular (de Vries et al., 2025; Oren et al., 2026) discrete-action single-agent environments (*Snake* and *RubiksCube* (Bonnet et al., 2024)); (III) Classical continuous control environments (*Ant, HalfCheetah* and *Humanoid* (Freeman et al., 2021)), where MCTS is extended to continuous actions in the manner of Hubert et al. (2021) (see Appendix I.5 for more detail); and (IV) scaling the inference of an LLM solving different *Sokoban* puzzles. As performance metrics, we use the popular BayesElo (Coulom, 2005) metric (measures

strength of play in zero-sum games), the observed averaged return, and the number of Sokoban instances solved, where relevant. To aggregate across different environments, we take the average BayesElo, average normalized return, and average number of instances solved, respectively.

To evaluate PMCTS as a tree-parallel MCTS algorithm, we begin by comparing its inference-time scaling with number of particles N to the popular tree-parallel MCTS variants Virtual Means and Virtual Losses. In each environment, all agents use the same π_θ and v_ϕ (when applicable) DNNs in the search and the default, popular MCTS hyperparameters (Danihelka et al., 2022; DeepMind et al., 2020). The only new parameter introduced with this work is the temperature η, whose effect is investigated in Figure 3, based on which we choose $\eta = 1.5$ which is used in all experiments. For additional experimental and implementation details see Appendices H and I.

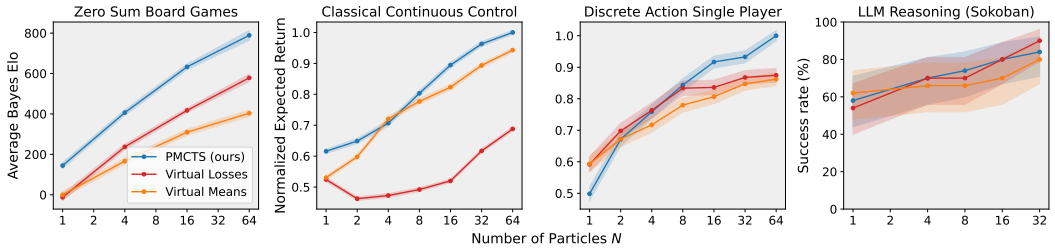


Figure 1: Scaling of tree-parallel MCTS variants with number of particles N across different domains. BayesElo with 95% confidence intervals for board games, Gaussian 95% confidence intervals for discrete and continuous control, and Wilson 95% confidence intervals for Sokoban.

PMCTS scales well with N and achieves consistently strong performance relative to both baselines across all four domains. We additionally investigate the scaling of the different tree-parallel algorithms in an example environment across different budgets M in Figure 4 in Appendix G. PMCTS scales well across increasing M and outperforms both baselines at every budget pair M and N.

To validate that PMCTS indeed provides effective runtime scaling with batch parallelization, we investigate the wall-clock runtime scaling of the different tree-parallel MCTS variants with vectorized, synchronously parallel implementations suited for batch-parallelization, in Figure 2 left and center. PMCTS's runtime scales very well with additional particles, as expected from an algorithm suited for batch parallelization. In contrast, the synchronous implementations of Virtual Means and Virtual Losses do not scale well, as expected, due to their inability to select multiple unique trajectories fully independently in parallel and subsequent collapse to sequential rather than parallel selection (see Appendix F.1 for additional detail).

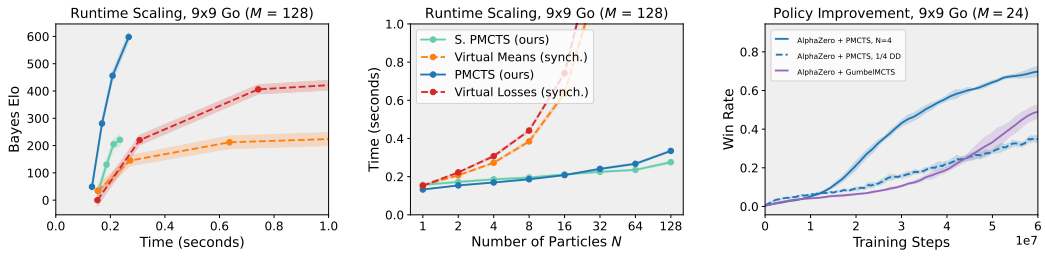


Figure 2: **Left:** Runtime scaling, BayesElo with 95% CI, $N = (1, 4, 16, 64)$ plotted. **Center:** Runtime scaling, 95% confidence interval across repeated evaluations. **Right:** Win rate vs. frames during training of AlphaZero with PMCTS and Gumbel MCTS, mean and 95% CI across 3 seeds.

To investigate the benefits of PMCTS in an RL training loop, we study scaling policy improvement during training with PMCTS, in Figure 2 right. We compare to GumbelMCTS (Danihelka et al., 2022), a modern non-parallel MCTS baseline. PMCTS with $N = 4$ outperforms baseline Gumbel MCTS in sample efficiency, demonstrating better policy improvement. During training, one may trade off parallel compute for *increased data diversity* (root-parallelization across different states) or for *better policy targets* (tree-parallelization in the same state). We include an example of this tradeoff with the dashed-line agent, which trades off increased $N = 4$ against generating $1/4$ as much data during training. All other training parameters, which were tuned for the baseline agent, are kept the same. This agent performs similarly in sample efficiency to the baseline in this example.

To validate that each mechanism developed in Section 3.3 indeed contributes to the overall performance of the algorithm, we ablate the different mechanisms in sequence on 9x9 Go ($M = 128$), in Figure 3 left. A very large number of mechanism combinations is possible. To maintain an informative and reasonable number of experiments, we compare from Simple PMCTS all the way to full PMCTS, with the different mechanisms activated in order, resulting in 6 different agents, each having one more mechanism activated than the previous. These are: (I) **S**imple **PMCTS** (Algorithm 1, shortened to **S**). (II) + **D**eduplication which removes duplicate trajectories. Deduplication computes the weight of the update (Equation 6, right) using the number of *unique* trajectories instead of $N(s_t)$. (III) + **ESS** which weights the update with 1 / ESS as a more principled estimate of the evaluation's variance (Equation 9). (IV) + **T**emperature, which increases the number of unique trajectories by increasing the entropy of the selection policy using the temperature η (Equation 7, right). (V) + weight **C**orrection which uses SIS (Equation 7, left) to estimate the expected return with respect to π_i rather than $\hat{\pi}_i$. (VI) The **PMCTS** algorithm, which uses retrospective reweighting in addition to all prior mechanisms. Performance increases consistently with every additional feature.

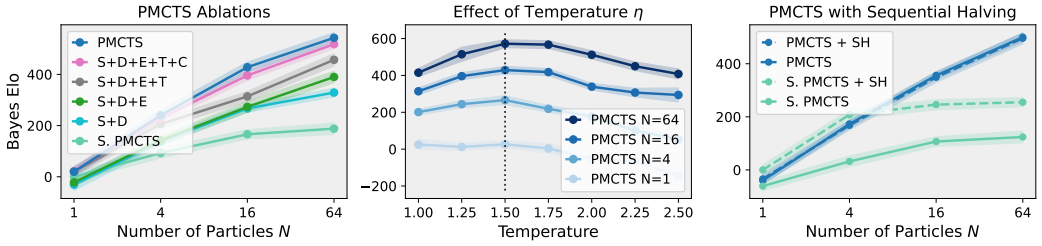


Figure 3: Ablations and hyperparameter evaluation on 9x9 Go ($M = 128$). BayesElo with 95% CI.

To investigate the effect of the introduced temperature hyperparameter η, we evaluate the effect of different η on the performance of PMCTS across different N in Figure 3 center. The best-performing temperature is 1.5, the temperature used in all other experiments. Sequential Halving (SH, Karnin et al. (2013)) has been used by Danihelka et al. (2022) and Oren et al. (2026) to improve search at the root. We investigate its effect on PMCTS, in Figure 3 right. SH, which enforces particle diversity at the root, has a strong positive effect on *Simple* PMCTS. However, when combined with PMCTS, it has no additional effect on the scaling. To reduce unnecessary complexity, we use PMCTS without SH in all experiments.

We include several additional results in Appendix G, including the individual results aggregated in Figure 1; an comparison to *root-parallelization*, to demonstrate that root-parallelization is indeed not effective for runtime scaling with DNNs, as discussed in Sections 1 and 5; a comparison to the non-MCTS, SMC-based search variants, demonstrating their limitations compared to parallel-MCTS variants; and an empirical investigation of the trade-off induced by η between trajectory diversity, divergence of the proposal $\hat{\pi}_i$ from the target π_i and the resulting inverse-ESS variance estimator.

7 Conclusions and Future Work

In this work, we introduced *Particle MCTS* (PMCTS), a principled, tree-parallel MCTS algorithm suited for runtime scaling through parallel compute, batch-parallelization and GPU acceleration. PMCTS uses stochastic selection policies and Sequential Monte Carlo (SMC) to select and evaluate multiple trajectories independently in parallel while explicitly encouraging and accounting for trajectory diversity and maintaining principled evaluation with respect to the target policy. Theoretically, we show that PMCTS retains MCTS's approximate policy improvement interpretation. Empirically, PMCTS scales well with parallel compute and outperforms or performs comparably to the virtual-visit baselines at matched (M, N) budgets across board games, single-player discrete-action tasks, and continuous control. In the 9x9 Go wall-clock comparison using matched JAX/GPU implementations, PMCTS demonstrates the benefits of independently parallel selection. PMCTS opens up multiple exciting avenues for future work such as more advanced temperature schedules for better proposal policies, incorporating proposals that sample without replacement to increase the number of unique trajectories and resampling methods (Chopin & Papaspiliopoulos, 2020) for different variance tradeoffs.

AI USE STATEMENT

In this work, we used generative AI tools for assistance in writing and verifying correctness of proofs, assistance in deploying experiments and writing code to analyze and plot results, and assistance in literature research. We have not used generative AI tools for other tasks with required disclosure. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ioannis Antonoglou, Julian Schrittwieser, Sherjil Ozair, Thomas K. Hubert, and David Silver. Planning in stochastic environments with a learned model. In *The Tenth International Conference on Learning Representations*, 2022.

Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. A difference in efficiency between synchronous and asynchronous systems. In *The 13th Annual ACM Symposium on Theory of Computing*, 1981. doi: 10.1145/800076.802466.

Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957.

G.E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11): 1526–1538, 1989. doi: 10.1109/12.42122.

Clément Bonnet, Daniel Luo, Donal John Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Illing Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries Petrus Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn Neil Waters, Mohamed Ali Ali Mimouni, Ulrich Armel Mbou Sob, Ruan John de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in JAX. In *The Twelfth International Conference on Learning Representations*, 2024.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

Gabriel Cardoso, Sergey Samsonov, Achille Thin, Eric Moulines, and Jimmy Olsson. BR-SNIS: bias reduced self-normalized importance sampling. *The 35th Annual Conference on Advances in Neural Information Processing Systems*, 2022.

Tristan Cazenave. Batch monte carlo tree search. In *Computers and Games - International Conference*, 2022. doi: 10.1007/978-3-031-34017-8_13.

Tristan Cazenave and Nicolas Jouandeau. On the parallelization of UCT. In *Computer Games Workshop 2007 (CGW07)*, 2007.

Guillaume Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel monte-carlo tree search. In *6th International Conference on Computers and Games (CG 2008)*, 2008.

Scott Cheng, Mahmut T. Kandemir, and Ding-Yong Hong. Speculative monte-carlo tree search. In *The 38th Annual Conference on Neural Information Processing Systems*, 2024.

Nicolas Chopin and Omiros Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer Series in Statistics. Springer, Cham, 1st edition, 2020. doi: 10.1007/978-3-030-47845-2.

Rémi Coulom. Bayesian Elo Rating. `https://www.remi-coulom.fr/Bayesian-Elo/`, 2005. [Online; accessed 02-05-2024].

Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with Gumbel. In *The Tenth International Conference on Learning Representations*, 2022.

Joery A. de Vries, Jinke He, Yaniv Oren, and Matthijs T. J. Spaan. Trust-Region Twisted Policy Improvement. In *The 42nd International Conference on Machine Learning*, 2025.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `https://github.com/google-deepmind`.

Markus Enzenberger and Martin Müller. A lock-free multithreaded monte-carlo tree search algorithm. In *The 12th International Conference on Advances in Computer Games*, 2009. doi: 10.1007/978-3-642-12993-3_2.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. doi: 10.1038/s41586-022-05172-4.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Remi Munos. Monte-Carlo Tree Search as Regularized Policy Optimization. In *The 37th International Conference on Machine Learning*, 2020.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, 2024.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.507.

Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and Planning in Complex Action Spaces. In *The 38th International Conference on Machine Learning*, 2021.

David R Hunter. Mm algorithms for generalized bradley-terry models. *The annals of statistics*, 32(1): 384–406, 2004.

Albin Jaldevik. General tree evaluation for AlphaZero. Master's thesis, Delft University of Technology, 2024.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost Optimal Exploration in Multi-Armed Bandits. In *The 30th International Conference on Machine Learning*, 2013.

Juhwan Kim, Byeongmin Kang, and Hyungmin Cho. Specmcts: Accelerating monte carlo tree search using speculative tree traversal. *IEEE Access*, 9:142195–142205, 2021. doi: 10.1109/ACCESS.2021.3120384.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *The 17th European Conference on Machine Learning*, 2006. doi: 10.1007/11871842_29.

Peter M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, C-22(8):786–793, 1973. doi: 10.1109/TC.1973.5009159.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *The 36th Annual Conference on Advances in Neural Information Processing Systems*, 2023.

Li-Cheng Lan, Wei Li, Ting-Han Wei, and I-Chen Wu. Multiple policy value monte carlo tree search. In *The Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019. doi: 10.24963/IJCAI.2019/653.

Anji Liu, Yitao Liang, Ji Liu, Guy Van den Broeck, and Jianshu Chen. On effective parallelization of monte carlo tree search. *CoRR*, abs/2006.08785, 2020.

Emil Malmsten and Wendelin Böhmer. Transzero: Parallel tree expansion in muzero using transformer networks. *arXiv preprint arXiv:2509.11233*, 2025.

Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Koppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023. doi: 10.1038/s41586-023-06004-9.

S. Ali Mirsoleimani, Aske Plaat, H. Jaap van den Herik, and Jos Vermaseren. An analysis of virtual loss in parallel MCTS. In *The 9th International Conference on Agents and Artificial Intelligence*, 2017. doi: 10.5220/0006205806480652.

S. Ali Mirsoleimani, H. Jaap van den Herik, Aske Plaat, and Jos Vermaseren. A lock-free algorithm for parallel MCTS. In *The 10th International Conference on Agents and Artificial Intelligence*, 2018.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based Reinforcement Learning: A Survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023. doi: 10.1561/2200000086.

Yaniv Oren, Viliam Vadocz, Matthijs T. J. Spaan, and Wendelin Boehmer. Epistemic Monte Carlo Tree Search. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Yaniv Oren, Moritz A Zanger, Pascal R Van der Vaart, Mustafa Mert Çelikok, Matthijs TJ Spaan, and Wendelin Boehmer. Value Improved Actor Critic Algorithms. In *The 39th Annual Conference on Neural Information Processing Systems*, 2025b.

Yaniv Oren, Joery A de Vries, Pascal R van der Vaart, Matthijs TJ Spaan, and Wendelin Böhmer. Twice sequential monte carlo for tree search. *The 43rd International Conference on Machine Learning*, 2026.

Alexandre Piché, Valentin Thomas, Cyril Ibrahim, Yoshua Bengio, and Chris Pal. Probabilistic planning with sequential monte carlo methods. In *The 7th International Conference on Learning Representations*, 2019.

Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

Lars Schäfers. *Parallel Monte-Carlo tree search for HPC systems and its application to computer go*. PhD thesis, University of Paderborn, 2014.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588 (7839):604–609, 2020. doi: 10.1038/s41586-020-03051-4.

Zijing Shi, Meng Fang, and Ling Chen. Monte carlo planning with large language model for text-based game agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den 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(7587):484–489, 2016. doi: 10.1038/nature16961.

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. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning*, 2024.

Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. Efficientzero V2: mastering discrete and continuous control with limited data. In *Forty-first International Conference on Machine Learning*, 2024.

Xiufeng Yang, Tanuj Kr Aasawat, and Kazuki Yoshizoe. Practical massively parallel monte-carlo tree search applied to molecular design. In *The 9th International Conference on Learning Representations*, 2021.

APPENDIX CONTENTS

A Acronym and Symbols List **16**

B Pseudocode **16**

B.1 Simple PMCTS 16

B.2 PMCTS 17

C Derivations **18**

C.1 Derivation of the numerically stable weighted average 18

C.2 Derivation of the particle-based backpropagation step in PMCTS 18

C.3 Approximating expectations under the target in Sequential Monte Carlo 19

C.4 Unbiased estimates of the value at any depth 20

D Formal Theoretical Analysis **20**

D.1 Improved-policy evaluation 20

D.2 Single-particle MCTS and evaluation accuracy 22

D.3 Distinct evaluations with DNNs 22

E Worst-Case Complexity Analysis **24**

E.1 MCTS Worst Case Complexity Analysis 24

E.2 Simple PMCTS Worst Case Complexity 24

E.3 PMCTS Worst Case Complexity 25

F MCTS and Parallelization: A Brief Survey **25**

F.1 Synchronous vs. asynchronous parallelism 25

F.2 Parallel MCTS: A brief survey 25

G Additional Results **26**

H Implementation Details **29**

H.1 The search policies PUCT and UCT 29

H.2 Improved Policy, Value and Action Selection - A Brief Survey of Design Choices . 29

H.3 The Virtual Losses and Virtual Means Heuristics 31

H.4 Sequential Halving with P/MCTS 31

H.5 Baselines 31

I Experimental Details **32**

I.1 Environments and Scaling Experiments Across Domains 32

I.2 Opening Book 32

I.3 Acting in the Environment 32

I.4 Evaluation Metrics 33

I.5 Discrete vs. Continuous Action Spaces 33
I.6 Runtime Experiments 33
I.7 Policy Improvement Experiment 33
I.8 LLM inference scaling on Sokoban 33

J Hyperparameters **34**
J.1 Hyperparameter tuning 34
J.2 Network architectures 35
J.3 Hyperparameters 35

A ACRONYM AND SYMBOLS LIST

Variable	Meaning
γ	discount
β_i	Q-value scale at iteration i (inverse-temperature)
s_t	state at step t
τ_t^n	trajectory at step t for particle n
T	last step
$\pi_\theta(s_t)$	prior policy network at state s_t
$v_\phi(s_t)$	value network prediction at state s_t
$\pi'(a_t\|s_t)$	target improved-policy for action a_t at state s_t
$\pi_i(a_t\|s_t)$	selection policy at iteration i for action a_t at state s_t
$\hat{\pi}_i(a_t\|s_t)$	sampling (e.g. *proposal*) policy at state s_t and iteration i
$M_i(s_t)$	visit count (or effective visit count in PMCTS) in the tree for state s_t at iteration i
$v_i(s_t)$	value estimate in the tree for state s_t at iteration i
$v(s_{T+1})$	value prediction at leaf s_{T+1} in the tree during expansion
$q_i(s_t, a_t)$	Q-value estimate in the tree for state s_t and action a_t at iteration i
$N(s_t)$	number of particles passing through state s_t
$ESS(s_t)$	effective sample size for particles that passed through state s_t
w_t^n	weight of particle n at step t
$\bar{w}_T^n(s_t)$	weight of particle n at step T normalized for state s_t
$\nu^n(s_t)$	value update at state s_t by particle n
$\nu(s_t)$	value update at state s_t across all particles
$\eta(s_t)$	possibly state-dependent temperature

B PSEUDOCODE

B.1 SIMPLE PMCTS

Algorithm 1 Simple PMCTS

Require: Number of particles N, Number of iterations M, Root state s_1, Environment model $\mathcal{M}$, DNNs π_θ, v_ϕ.

1: Initialize a tree with capacity for $NM + 1$ nodes.
2: Initialize root using predictions $\pi_\theta(s_1)$ and $v_\phi(s_1)$.
3: **for** $i \leftarrow 1$ to M **do**
4: Sample $\{\tau_T^n = (s_1, a_1^n, ..., s_T^n, a_T^n)\}_{n=1}^N$ from $\pi_i, \mathcal{M}$ independently in parallel. ▷ Selection.
5: Expand and add to the tree up to N unique leaves in parallel: ▷ Expansion.

$$\left\{s_{T+1}^n \sim P(s_T^n, a_T^n),\, r(s_T^n, a_T^n) = \mathbb{E}[R(s_T^n, a_T^n)],\, v(s_{T+1}^n),\, \pi_\theta(s_{T+1}^n)\right\}_{n=1}^N.$$

6: For each node s_t compute $\nu_i(s_t)$ (Equation 6, left): ▷ Backpropagation.

$$\nu_i(s_t) = \frac{1}{N(s_t)} \sum_{n=1}^{N} \mathbb{1}_{s_t \in \tau_T^n} \nu^n(s_t),$$

7: And update the tree (Equation 6, right):

$$v_{i+1}(s_t) = v_i(s_t) + \frac{(\nu_i(s_t) - v_i(s_t))N(s_t)}{M_i(s_t) + N(s_t)}, \quad M_{i+1}(s_t) = M_i(s_t) + N(s_t).$$

8: **end for**
9: **return** : ▷ See Appendix H.2, Equation 50

$$\pi_{search}(a|s_1) = \mathcal{I}(\pi_\theta, q_{M+1}), \quad V_{search}(s_1) = \sum_{a \in \mathcal{A}} \bar{\pi}(a|s_1) q_{M+1}(s_1, a), \quad a \sim \bar{\pi}(s_1).$$

$$\bar{\pi}(a|s_1) \propto \begin{cases} \pi_{search}(a|s_1) & \text{if: } M(s_1, a) > 0 \\ 0 & \text{if: } M(s_1, a) = 0 \end{cases}$$

B.2 PMCTS

For added clarity, we include a brief summary of PMCTS in natural language in addition and following the pseudocode.

Algorithm 2 PMCTS

Require: Number of particles N, Number of iterations M, Root state s_1, Environment model $\mathcal{M}$, DNNs π_θ, v_ϕ.

Initialize a tree with capacity for $NM + 1$ nodes.
Initialize root using predictions $\pi_\theta(s_1)$ and $v_\phi(s_1)$.
for $i \leftarrow 1$ to M **do**
  for $t \leftarrow 1$ to T for $n \in N$ particles independently in parallel **do** ▷ Weighted Selection
    Sample and append to trajectory τ_{t-1}: ▷ *SMC's Sampling*

$$a_t^n \sim \hat{\pi}_i(a_t^n|s_t^n), \quad s_{t+1}^n \sim P(\cdot|s_t^n, a_t^n),$$

    Update particle weights (Equation 7): ▷ *SMC's Correction*

$$w_t^n = w_{t-1}^n \frac{\pi_i(a_t^n \mid s_t^n)}{\hat{\pi}_i(a_t^n \mid s_t^n)},$$

  end for
  Expand and add to the tree up to N unique leaves in parallel: ▷ Expansion.

$$\left\{s_{T+1}^n \sim P(s_T^n, a_T^n),\, r(s_T^n, a_T^n) = \mathbb{E}[R(s_T^n, a_T^n)],\, v(s_{T+1}^n),\, \pi_\theta(s_{T+1}^n)\right\}_{n=1}^N.$$

  For every expanded pair (s_T^n, a_T^n), with updated normalizer Z_{i+1}, compute:

$$\pi_{i+1}(a_T^n \mid s_T^n) = \pi_\theta(a_T^n \mid s_T^n) \exp(\beta_{i+1}(s_T^n) q_{i+1}(s_T^n, a_T^n)) \,/ Z_{i+1}$$

  Update weights w_T^n with respect to $\pi_{i+1}(s_T)$ (Equation 10): ▷ Retrospective reweighting

$$w_T^n \leftarrow w_T^n \frac{\pi_{i+1}(a_T^n \mid s_T^n)}{\pi_i(a_T^n \mid s_T^n)}$$

  Merge duplicates, preserving weight $w_T^n(s_{T+1}) = \sum_{m=1}^N w_T^m \mathbb{1}_{s_{T+1}=s_{T+1}^m}$ per unique trajectory and setting duplicates to zero.
  Normalize weights with respect to each other at all nodes s_t:

$$\bar{w}_T^n(s_t) = \frac{w_T^n \mathbb{1}_{s_t \in \tau_T^n}}{\sum_{m=1}^N w_T^m \mathbb{1}_{s_t \in \tau_T^m}}.$$

  Compute $\nu_i(s_t)$ (Equation 8, left) and the ESS for each node s_t: ▷ Weighted Backprop.

$$\nu_i(s_t) = \sum_{n=1}^N \bar{w}_T^n(s_t) \mathbb{1}_{s_t \in \tau_T^n} \nu^n(s_t), \quad ESS(s_t) = \frac{1}{\sum_{n=1}^N (\bar{w}_T^n(s_t))^2}$$

  Update every modified node s_t in the tree (Equation 9):

$$v_{i+1}(s_t) = v_i(s_t) + \frac{(\nu_i(s_t) - v_i(s_t))\text{ESS}(s_t)}{M_i(s_t) + \text{ESS}(s_t)}, \quad M_{i+1}(s_t) = M_i(s_t) + \text{ESS}(s_t).$$

end for
return : ▷ See Appendix H.2, Equation 50

$$\pi_{search}(a|s_1) = \mathcal{I}(\pi_\theta, q_{M+1}), \quad V_{search}(s_1) = \sum_{a \in \mathcal{A}} \bar{\pi}(a|s_1) q_{M+1}(s_1, a), \quad a \sim \bar{\pi}(s_1).$$

$$\bar{\pi}(a|s_1) \propto \begin{cases} \pi_{search}(a|s_1) & \text{if:} \quad M(s_1, a) > 0 \\ 0 & \text{if:} \quad M(s_1, a) = 0 \end{cases}$$

PMCTS summarized: At each parallel MCTS iteration i, N particles sample trajectories independently in parallel from a higher-entropy stochastic-selection policy $\hat{\pi}_i$ (Equation 7, right). At each step during selection, the particles' weights are updated with SIS, maintaining a principled estimate of the distribution over trajectories with respect to π_i (Equation 7, left). When all particles arrive at a leaf, up to N leaves are expanded in on batch parallel pass of batch size N, as in Simple PMCTS. In our practical implementation, we expand all N leaves, but only keep track of each unique trajectory once, so the tree does not have duplicate leaves. Retrospective reweighting is then applied, updating the particles' weights to estimate the distribution with respect to π_{i+1} rather than π_i (Equation 10). Deduplication is applied, removing duplicate trajectories while maintaining the correct importance weight per unique trajectory. Backpropagation is applied, computing the returns ν_i and the ESS and using them to update v_{i+1} (Equations 8 and 9 respectively). At this point, the iteration is complete, and the algorithm proceeds to the next iteration.

C Derivations

C.1 Derivation of the numerically stable weighted average

$$v_{i+1}(s_t) = \frac{\nu(s_t)N_i(s_t) + v_i(s_t)M_i(s_t)}{M_i(s_t) + N(s_t)} \tag{11}$$

$$= \frac{\nu(s_t)N_i(s_t) + v_i(s_t)M_i(s_t) + v_i(s_t)N_i(s_t) - v_i(s_t)N_i(s_t)}{M_i(s_t) + N(s_t)} \tag{12}$$

$$= \frac{v_i(s_t)M_i(s_t) + v_i(s_t)N_i(s_t)}{M_i(s_t) + N_i(s_t)} + \frac{\nu(s_t)N_i(s_t) - v_i(s_t)N_i(s_t)}{M_i(s_t) + N_i(s_t)} \tag{13}$$

$$= v_i(s_t) + \frac{(\nu(s_t) - v_i(s_t))N_i(s_t)}{M_i(s_t) + N_i(s_t)}. \tag{14}$$

C.2 Derivation of the particle-based backpropagation step in PMCTS

C.2.1 Derivation of the variance estimator

We analyze the variance of the PMCTS estimator and construct an estimator of it by considering the set of N particles contributing to the update of node s_t. The estimator is defined as the weighted sum:

$$\mathbb{V}[\nu(s_t)] = \mathbb{V}\left[\sum_{n=1}^{N} \bar{w}_T^n \nu^n(s_t)\right] \tag{15}$$

We assume that we have merged all duplicate particles into one particle with the total weight and all trajectories are unique. After deduplication, each retained particle represents one unique trajectory with its aggregate importance weight. Before normalization, the proposal trajectories are sampled independently in our SIS construction. Their self-normalized contributions are nevertheless coupled through the common random denominator, and evaluation errors of distinct trajectories need not be independent. As a practical approximation, we treat the normalized weights as fixed and neglect cross-trajectory covariance. Under this assumption, and by treating the weights as approximately constant relative to the variance of the returns, we obtain:

$$\mathbb{V}[\nu(s_t)] \approx \sum_{n=1}^{N} \mathbb{V}[\bar{w}_T^n \nu^n(s_t)] \approx \sum_{n=1}^{N} (\bar{w}_T^n)^2 \mathbb{V}[\nu^n(s_t)] \tag{16}$$

This formulation mirrors the derivation of the Effective Sample Size (ESS) (Chopin & Papaspiliopoulos, 2020), where the variance of a weighted importance sampling estimator is shown to scale with $\sum(\bar{w}_t^n)^2$. The ESS approximation is widely used as a proxy for estimator quality when the true covariance structure is intractable.

Assuming the variance of the return for any single trajectory is bounded by $\mathbb{V}[\nu^n(s_t)] \leq \sigma^2$, we obtain the following bound within the fixed-weight, negligible-covariance approximation:

$$\mathbb{V}[\nu(s_t)] \approx \sum_{n=1}^{N} (\bar{w}_T^n)^2 \mathbb{V}[\nu^n(s_t)] \leq \sigma^2 \sum_{n=1}^{N} (\bar{w}_T^n)^2 = \sigma^2 \frac{1}{ESS(s_t)} \tag{17}$$

This suggests that the variance of the node update is approximately proportional to the sum of the squared weights, $\sum_{n=1}^{N}(\bar{w}_t^n)^2$, allowing us to use this term as a variance proxy in the tree search.

C.2.2 Derivation of the node value update

The weight of the update to the variance-minimizing unbiased estimator of the mean is the inverse variance. Under the approximation in Equation (17), and after removing the common scale σ^2, the variance estimator is:

$$\sum_{n=1}^{N}\left(\bar{w}_T^n(s_t)\right)^2,$$

and its inverse is

$$\frac{1}{\sum_{n=1}^{N}\left(\bar{w}_T^n(s_t)\right)^2} = \mathrm{ESS}(s_t).$$

We therefore use $\mathrm{ESS}(s_t)$ to approximate the inverse-variance of the update and weight the particle-based update.

We can now derive the update:

$$v_{i+1}(s_t) = \frac{\nu(s_t)\mathrm{ESS}(s_t) + v_i(s_t)M_i(s_t)}{M_i(s_t) + \mathrm{ESS}(s_t)} \tag{18}$$

$$= \frac{\nu(s_t)\mathrm{ESS}(s_t) + v_i(s_t)M_i(s_t) + v_i(s_t)\mathrm{ESS}(s_t) - v_i(s_t)\mathrm{ESS}(s_t)}{M_i(s_t) + \mathrm{ESS}(s_t)} \tag{19}$$

$$= \frac{v_i(s_t)M_i(s_t) + v_i(s_t)\mathrm{ESS}(s_t)}{M_i(s_t) + \mathrm{ESS}(s_t)} + \frac{\nu(s_t)\mathrm{ESS}(s_t) - v_i(s_t)\mathrm{ESS}(s_t)}{M_i(s_t) + \mathrm{ESS}(s_t)} \tag{20}$$

$$= v_i(s_t) + \frac{(\nu(s_t) - v_i(s_t))\mathrm{ESS}(s_t)}{M_i(s_t) + \mathrm{ESS}(s_t)}. \tag{21}$$

C.3 Approximating expectations under the target in Sequential Monte Carlo

There are multiple approaches to approximating expectations under the target using the particles $\{\tau_t^n, w_t^n\}_{n=1}^N$ of SMC, for some example function $\mathcal{R}$ (in RL terms, we can think of $\mathcal{R}(\tau_T^n)$ as the return along the trajectory).

The standard approach is to use the *self-normalizing* estimator:

$$\frac{1}{\sum_{n=1}^{N} w_t^n}\sum_{n=1}^{N} w_t^n \mathcal{R}(\tau_t^n) \approx \mathbb{E}_{\pi'}[R(\tau_t^n)] = V^{\pi'}(s_1). \tag{22}$$

It normalizes with respect to the sum of the *weights* of the particles $\sum_{n=1}^{N} w_t^n$. It is the more common of the options, as it allows for SIS even with access only to the probability mass / density functions $\pi'(a|s), \pi_\theta(a|s)$ up to a proportionality constant. This estimator however is biased (although consistent), introducing bias of order $\mathcal{O}(1/N)$ (Cardoso et al., 2022), because the ratio of two unbiased estimators: $\sum_{n=1}^{N} w_t^n \mathcal{R}(\tau_t^n) / \sum_{n=1}^{N} w_t^n$, is not itself unbiased (Chopin & Papaspiliopoulos, 2020).

However, when we have access to the full densities / masses $\pi'(a|s), \pi_\theta(a|s)$, we can also use the estimator (Equation 4):

$$\frac{1}{N}\sum_{n=1}^{N} w_t^n \mathcal{R}(\tau_t^n) \approx \mathbb{E}_{\pi'}[R(\tau_t^n)] = V^{\pi'}(s_1). \tag{23}$$

This is an *unbiased* estimator of expectations under the function $\mathcal{R}$.

In practice, the first estimator (Equation 22) is often preferred in practice because it is numerically stable and can have lower variance, at the cost of an $O(1/N)$ bias under standard regularity conditions, and we use it in the practical algorithm.

C.4 UNBIASED ESTIMATES OF THE VALUE AT ANY DEPTH

The unnormalized estimator provides unbiased estimates of the expectation at the root s_1. However, we cannot simply use the same weights for any node under the root (the way we could for the self-normalizing estimator) and expect unbiased expectations, because the weights will be proportional to the probability of reaching those nodes. As a result, unlikely trajectories will have value estimates close to zero (because their weights will be close to zero) and using them directly to update the tree will effectively set all the values closer to zero.

This can be addressed however, by maintaining an array of weights per particle n, which save a weight per depth t. At each depth t the new-weight to depth t is initialized with 1.0, and is updated in the same manner as all other weights at all depths. As a result, a different weight is available per particle per depth to compute the expectation at each depth (i.e. each node along the trajectory). This can effectively be thought of as "restarting" the SMC process at each depth guaranteeing that its properties apply for each depth and not just the root.

In practice, PMCTS uses the self-normalized estimator. Unlike the unnormalized estimator, it does not require a separate weight array at every depth because the common prefix weight cancels in the node-wise normalization. Bias-reduced variants of the self-normalized estimator also exist (Cardoso et al., 2022).

D FORMAL THEORETICAL ANALYSIS

D.1 IMPROVED-POLICY EVALUATION

We first make explicit the improvement objective used by the regularized instance of Equation 2. Assume a finite action space $\mathcal{A}$, $\pi_\theta(a \mid s_t) > 0$, and $\beta_i(s_t) > 0$. Then $\pi_i = \mathcal{I}_{\mathrm{reg}}(\pi_\theta, q_i)$ uniquely solves

$$\pi_i(\cdot \mid s_t) = \operatorname*{arg\,max}_{\mu \in \Delta(\mathcal{A})} \left\{ \sum_{a \in \mathcal{A}} \mu(a) q_i(s_t, a) - \frac{1}{\beta_i(s_t)} D_{\mathrm{KL}}(\mu \,\|\, \pi_\theta(\cdot \mid s_t)) \right\}. \tag{24}$$

Indeed, setting $Z_i(s_t) = \sum_a \pi_\theta(a \mid s_t) e^{\beta_i(s_t) q_i(s_t, a)}$, the objective at any μ equals

$$\sum_a \mu(a) q_i(s_t, a) - \frac{D_{\mathrm{KL}}(\mu \| \pi_\theta)}{\beta_i(s_t)} = \frac{\log Z_i(s_t) - D_{\mathrm{KL}}(\mu \| \pi_i)}{\beta_i(s_t)}. \tag{25}$$

The right side is uniquely maximized at $\mu = \pi_i$. Comparing its objective with that of π_θ additionally gives

$$\sum_a \big(\pi_i(a \mid s_t) - \pi_\theta(a \mid s_t)\big) q_i(s_t, a) \geq \frac{D_{\mathrm{KL}}(\pi_i \| \pi_\theta)}{\beta_i(s_t)} \geq 0. \tag{26}$$

The divergences in these expressions are evaluated at s_t. This is improvement against the search values q_i; the backup results below do not require a particular choice of the improvement operator $\mathcal{I}$.

Fix a pre-batch tree. Let $\mathcal{T}_i$ be its finite set of complete root-to-frontier or terminal trajectories. The target π_i selects actions in the tree and follows π_θ after the frontier. Write $p_i(\tau)$ and $\hat{p}_i(\tau)$ for the probabilities of $\tau \in \mathcal{T}_i$ under π_i and PMCTS's proposal $\hat{\pi}_i$, respectively. Equation 7 supplies the pre-retrospective importance ratio $\omega_i(\tau) = p_i(\tau)/\hat{p}_i(\tau)$. To describe Equation 10, let $\pi_{i+1}^{(N)}$ be the policy at the newly evaluated parent nodes after expanding a batch of N particles. If the last action (s_T, a_T) of τ was expanded, define its terminal-action correction by

$$d_{i,N}(\tau) = \frac{\pi_{i+1}^{(N)}(a_T \mid s_T)}{\pi_i(a_T \mid s_T)}. \tag{27}$$

Set $d_{i,N}(\tau) = 1$ for trajectories ending at a terminal state. The corrected weights target the trajectory distribution

$$p_{i,N}^{\mathrm{ret}}(\tau) = \frac{p_i(\tau) d_{i,N}(\tau)}{\sum_{\tau' \in \mathcal{T}_i} p_i(\tau') d_{i,N}(\tau')}. \tag{28}$$

Its conditional action probabilities at history-specific tree nodes define $\pi^{\mathrm{ret}}_{i,N}$, followed by π_θ beyond the frontier. Equation 10 corrects only a trajectory's last expanded action; therefore $\pi^{\mathrm{ret}}_{i,N}$ need not be $\pi^{(N)}_{i+1}$ at all expanded nodes.

Theorem 4 (Improved-policy evaluation (formal))**.** *Fix an iteration i and condition on the pre-batch search history. Hold the finite pre-batch tree, π_i, and $\hat{\pi}_i$ fixed as N varies. Assume a true deterministic model, bounded rewards, finite action spaces, and $\hat{\pi}_i(a \mid s) > 0$ wherever $\pi_i(a \mid s) > 0$, with positive target probabilities on searched actions. Each particle independently selects a trajectory under the respective selection policy: π_i for Simple PMCTS, $\hat{\pi}_i$ for PMCTS, without resampling. Conditional on its selected frontier, its evaluation has mean V^{π_θ} and a finite second moment, independently across particles; an exact deterministic evaluation may instead be shared. Every sampled frontier action is expanded. At every pre-existing node s_t with positive reach probability under π_i, for every $n \geq 1$,*

$$\begin{aligned} \mathbb{E}[\nu^{\mathrm{S}}_i(s_t) \mid N^{\mathrm{S}}_i(s_t) = n] &= V^{\pi_i}(s_t), \\ \nu^{\mathrm{P},(N)}_{i,\mathrm{pre}}(s_t) &\xrightarrow[N\to\infty]{\text{a.s.}} V^{\pi_i}(s_t), \\ \nu^{\mathrm{P},(N)}_{i,\mathrm{ret}}(s_t) - V^{\pi^{\mathrm{ret}}_{i,N}}(s_t) &\xrightarrow[N\to\infty]{\text{a.s.}} 0. \end{aligned} \tag{29}$$

Here ν^{S}_i is the new-batch average in Equation (6), and the two PMCTS terms are the new-batch self-normalized backups in Equation (8) before and after Equation (10). The corrected target may depend on the same batch, and neither finite-N PMCTS backup is generally unbiased.

Proof. For Simple PMCTS, let $Z_n = \mathbf{1}_{\{s_t \in \tau^n_T\}}$ and set $Y_n = Z_n \nu^n_i(s_t)$, with $Y_n = 0$ if $Z_n = 0$. The pairs (Y_n, Z_n) are i.i.d. Conditional on $Z_n = 1$, actions are sampled from π_i up to the first frontier action, and the leaf continuation has the correct conditional mean under π_θ. Hence $\mathbb{E}[Y_n \mid Z_n = 1] = V^{\pi_i}(s_t)$. Given $N^{\mathrm{S}}_i(s_t) = n \geq 1$, Equation (6) averages n such returns. This proves its conditional expectation in Equation 29.

For PMCTS, let $m_t(\tau)$ be the true expected return from s_t conditional on $\tau \ni s_t$, including the π_θ continuation. For each $\tau \in \mathcal{T}_i$ define

$$\begin{aligned} C_N(\tau) &= \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}_{\{\tau^n_T = \tau\}}, \\ H_{t,N}(\tau) &= \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}_{\{\tau^n_T = \tau\}} \nu^n_i(s_t) \qquad (s_t \in \tau). \end{aligned} \tag{30}$$

Independence and the finite second moment imply, by the strong law and finiteness of $\mathcal{T}_i$, that almost surely and simultaneously for every such path and node,

$$\begin{aligned} C_N(\tau) &\longrightarrow \hat{p}_i(\tau), \\ H_{t,N}(\tau) &\longrightarrow \hat{p}_i(\tau) m_t(\tau). \end{aligned} \tag{31}$$

The same limits hold for a shared evaluation when that evaluation equals the exact conditional continuation value. Before retrospective correction, the numerator of the self-normalized backup at s_t is $\sum_{\tau \ni s_t} \omega_i(\tau) H_{t,N}(\tau)$, and its denominator is $\sum_{\tau \ni s_t} \omega_i(\tau) C_N(\tau)$. Their limits are $\sum_{\tau \ni s_t} p_i(\tau) m_t(\tau)$ and $\sum_{\tau \ni s_t} p_i(\tau) > 0$, respectively. Their ratio equals $V^{\pi_i}(s_t)$.

After correction, the same two path sums acquire factors $d_{i,N}(\tau)$. These factors can depend on all returns in the batch, but they satisfy $0 \leq d_{i,N}(\tau) \leq 1/\pi_i(a_T \mid s_T)$ for a frontier trajectory, and equal one for a terminal trajectory. Since $\mathcal{T}_i$ is finite, the differences between the corrected empirical sums and their population path sums vanish almost surely, uniformly over the possible values of $\pi^{(N)}_{i+1}$.

The population denominator for s_t is bounded below by a fixed positive constant for all sufficiently large N. To see this, follow a positive-probability path from s_t to a deepest pre-batch node $s^\star$ that is either terminal or has no expanded outgoing action. A terminal trajectory contributes its positive path probability. Otherwise, every action at $s^\star$ has positive proposal probability and therefore occurs in the batch eventually, almost surely. All these actions are then expanded. If ρ is the prefix ending at $s^\star$, their combined corrected target mass is $p_i(\rho) \sum_a \pi^{(N)}_{i+1}(a \mid s^\star) = p_i(\rho) > 0$. Thus both corrected

ratios are well defined asymptotically. By Equation 28, the ratio of the population path sums is exactly $V^{\pi^{\mathrm{ret}}_{i,N}}(s_t)$. Uniform convergence of the sums proves the last limit in Equation 29, despite the data-dependent correction. Finally, self-normalized importance ratios and the data-dependent correction need not be unbiased at finite N. □

D.2 Single-particle MCTS and evaluation accuracy

Theorem 5 (MCTS with increased evaluation accuracy (formal))**.** *Under the assumptions of Theorem 4, use the same stochastic selection policy π_i for MCTS and Simple PMCTS, and set PMCTS's proposal to $\hat{\pi}_i = \pi_i$ for the $N = 1$ comparison. Assume its retrospective factor is positive, as with the full-support regularized policy in Equation 2. Then all three algorithms have the same single-trajectory selection, expansion, and node update at $N = 1$, including PMCTS's retrospective step. For any pre-existing node s_t with positive reach probability, let $N_i^{\mathrm{S}}(s_t)$ denote the number of Simple PMCTS particles visiting it in the current batch (the $N(s_t)$ of Equation (6)) and let $\sigma_i^2(s_t) = \mathbb{V}[\nu_i^1(s_t) \mid s_t \in \tau_T^1]$. For every $n \geq 1$, and for either full-PMCTS proposal satisfying Theorem 4,*

$$\begin{aligned} \mathbb{V}[\nu_i^{\mathrm{S}}(s_t) \mid N_i^{\mathrm{S}}(s_t) = n] &= \frac{\sigma_i^2(s_t)}{n}, \\ \nu_{i,\mathrm{pre}}^{\mathrm{P},(N)}(s_t) - V^{\pi_i}(s_t) &= O_{\mathbb{P}}(N^{-1/2}), \\ \nu_{i,\mathrm{ret}}^{\mathrm{P},(N)}(s_t) - V^{\pi^{\mathrm{ret}}_{i,N}}(s_t) &= O_{\mathbb{P}}(N^{-1/2}). \end{aligned} \tag{32}$$

Moreover, $N_i^{\mathrm{S}}(s_t) \to \infty$ and $\nu_i^{\mathrm{S}}(s_t) \to V^{\pi_i}(s_t)$ almost surely as $N \to \infty$, simultaneously at every such pre-existing node. The two PMCTS rates concern the respective targets identified in Theorem 4.

Proof. At $N = 1$ all three algorithms select the same trajectory when their selection policies agree, and they expand and evaluate its frontier action identically. Simple PMCTS's visiting-node count in Equation 6 is one. PMCTS's per-node normalized weight and ESS in Equations 8-9 are both one; multiplying its sole particle's weight by the retrospective factor in Equation 10 does not change its normalized return or ESS. Both variants therefore perform the same one-trajectory node update as MCTS. This equivalence uses no additional action-selection heuristic.

The i.i.d. pairs (Y_n, Z_n) from the proof of Theorem 4 imply that, conditional on $N_i^{\mathrm{S}}(s_t) = n$, the n visiting returns are independent and identically distributed, with variance $\sigma_i^2(s_t)$. Their average has the variance in Equation 32. Write $p_t = \mathbb{E}[Z_n] > 0$. The strong law gives $N_i^{\mathrm{S}}(s_t)/N \to p_t$ and $\sum_{n=1}^{N} Y_n/N \to p_t V^{\pi_i}(s_t)$ almost surely. Dividing these limits proves the convergence of the batch average. The pre-batch tree is finite, so the conclusions hold simultaneously at its positive-probability nodes.

For PMCTS, each coordinate $C_N(\tau)$ and $H_{t,N}(\tau)$ in Equation 30 is an average of independent variables with finite variance. Hence its difference from the corresponding limit in Equation 31 is $O_{\mathbb{P}}(N^{-1/2})$. There are finitely many paths, so the largest such difference has the same order. Before retrospective correction, the importance ratios $\omega_i(\tau)$ are fixed and bounded, and the limiting node denominator is positive. Taking the ratio gives the first PMCTS rate in Equation 32.

After retrospective correction, the factors $d_{i,N}(\tau)$ may depend on all these averages, but are uniformly bounded over paths and N. Multiplying the coordinate errors by them therefore retains the $O_{\mathbb{P}}(N^{-1/2})$ order without an independence assumption for the factors. The population denominator is bounded below with probability tending to one by the deepest-node argument in the proof of Theorem 4. On this event the empirical denominator is also bounded below with probability tending to one. The difference of the empirical and population ratios is consequently $O_{\mathbb{P}}(N^{-1/2})$, proving the post-retrospective rate. □

D.3 Distinct evaluations with DNNs

Theorem 6 (Benefit of distinct evaluations (formal))**.** *Fix a pre-existing node s_t visited by particles of both algorithms and condition on their pre-batch histories, sampled trajectories, and pre-retrospective particle weights. For $k \in \{\mathrm{S}, \mathrm{P}\}$, let $\mathcal{U}_k(s_t)$ be the distinct trajectories reaching s_t, and let $b_u^k(s_t)$ be the sum of the weights of particles following u, normalized across particles reaching s_t. Simple*

PMCTS has unit weights; PMCTS has the importance weights from Equation 7. Define

$$\mathrm{ESS}^{k}_{\mathrm{unique}}(s_t) = \left(\sum_{u \in \mathcal{U}_k(s_t)} b^k_u(s_t)^2 \right)^{-1}. \tag{33}$$

Suppose duplicate trajectories share the same backpropagated evaluation error ϵ^k_u; across distinct trajectories these conditional errors have mean zero and are uncorrelated. If their conditional variances $\sigma^2_{u,k}$ are bounded by σ^2, then the conditional variance due to evaluation error in the pre-retrospective backup satisfies

$$\mathbb{V}[\nu^k_{i,\mathrm{pre}}(s_t) \mid \textit{trajectories, weights}] = \sum_{u \in \mathcal{U}_k(s_t)} b^k_u(s_t)^2 \sigma^2_{u,k} \le \frac{\sigma^2}{\mathrm{ESS}^{k}_{\mathrm{unique}}(s_t)}. \tag{34}$$

Here $\nu^{\mathrm{S}}_{i,\mathrm{pre}} = \nu^{\mathrm{S}}_i$. If all the conditional error variances equal the same $\sigma^2 > 0$ in both algorithms, equality holds, and PMCTS has lower conditional evaluation-noise variance exactly when $\mathrm{ESS}^{\mathrm{P}}_{\mathrm{unique}}(s_t) > \mathrm{ESS}^{\mathrm{S}}_{\mathrm{unique}}(s_t)$. The errors may have nonzero bias relative to true values. Merging identical trajectories and summing their weights leaves a batch's weighted backup unchanged, both before and after its terminal-action retrospective correction.

The following comparison also covers PMCTS's implemented retrospective correction, even when its weights depend on the same evaluation errors. Let $\mathcal{F}$ denote the conditioned pre-batch histories, trajectories, and pre-retrospective weights. Write $\mathbf{b}^{\mathrm{P,ret}}$ for the vector of normalized, retrospectively corrected group weights at s_t. Let $\mathbf{b}^{\mathrm{P},0}$ be the weights obtained by repeating the same correction with every new leaf evaluation in the batch replaced by its conditional mean, while holding the trajectories and pre-retrospective weights fixed. Assume both corrected normalizers are positive, as with the full-support regularized policy of Equation 2. Define

$$\begin{aligned} \mathrm{ESS}^{\mathrm{P},0}_{\mathrm{unique,ret}}(s_t) &= \left(\sum_u (b^{\mathrm{P},0}_u(s_t))^2 \right)^{-1}, \\ \delta_{\mathrm{ret}}(s_t) &= \left(\mathbb{E}\left[\|\mathbf{b}^{\mathrm{P,ret}} - \mathbf{b}^{\mathrm{P},0}\|_2^2 \|\boldsymbol{\epsilon}^{\mathrm{P}}\|_2^2 \mid \mathcal{F} \right] \right)^{1/2}. \end{aligned} \tag{35}$$

Let $\Delta^{\mathrm{P}}_{\mathrm{ret}}(s_t) = \sum_u b^{\mathrm{P,ret}}_u(s_t)\epsilon^{\mathrm{P}}_u$ denote the final backup's evaluation-error contribution. Under the common-variance assumption $\sigma^2_{u,k} = \sigma^2 > 0$, its conditional root mean squared error satisfies

$$\left(\mathbb{E}[(\Delta^{\mathrm{P}}_{\mathrm{ret}}(s_t))^2 \mid \mathcal{F}] \right)^{1/2} \le \frac{\sigma}{\sqrt{\mathrm{ESS}^{\mathrm{P},0}_{\mathrm{unique,ret}}(s_t)}} + \delta_{\mathrm{ret}}(s_t). \tag{36}$$

For Simple PMCTS, Equation 34 gives its conditional evaluation-error MSE. PMCTS has a smaller contribution after retrospective correction whenever

$$\frac{\sigma}{\sqrt{\mathrm{ESS}^{\mathrm{P},0}_{\mathrm{unique,ret}}(s_t)}} + \delta_{\mathrm{ret}}(s_t) < \frac{\sigma}{\sqrt{\mathrm{ESS}^{\mathrm{S}}_{\mathrm{unique}}(s_t)}}. \tag{37}$$

This condition also implies lower conditional evaluation-noise variance, since variance cannot exceed mean squared error. The term δ_{ret} accounts for using the same evaluations to determine retrospective weights and to compute the backup; it vanishes if the corrected weights do not change with those errors.

Proof. For each distinct trajectory u, write its return from s_t as its conditional mean plus ϵ^k_u. Conditional on the paths and pre-retrospective weights, the error component of the backup is $\sum_u b^k_u(s_t)\epsilon^k_u$. Uncorrelatedness removes its cross terms, giving the equality in Equation 34. The bound follows from Equation 33; common positive error variance yields equality and the stated comparison. This comparison is conditional on the trajectories and their weights, and concerns the pre-retrospective backup: the post-expansion weights may themselves depend on the new evaluation errors.

For a group G_u of identical trajectories with the same return $\nu_u(s_t)$, its contribution to a weighted numerator is $\sum_{n \in G_u} w_n \nu_u(s_t) = (\sum_{n \in G_u} w_n)\nu_u(s_t)$; its denominator contribution is $\sum_{n \in G_u} w_n$.

Grouping therefore leaves the normalized backup unchanged. Identical trajectories receive the same terminal-action correction $d_{i,N}(u)$, so the same argument applies after retrospective reweighting.

For the final retrospective backup, decompose its evaluation-error contribution as

$$\Delta^{\mathrm{P}}_{\mathrm{ret}}(s_t) = \sum_u b^{\mathrm{P},0}_u(s_t)\epsilon^{\mathrm{P}}_u + \sum_u \big(b^{\mathrm{P},\mathrm{ret}}_u(s_t) - b^{\mathrm{P},0}_u(s_t)\big)\epsilon^{\mathrm{P}}_u. \tag{38}$$

Conditional on $\mathcal{F}$, the first term has root mean square $\sigma/\sqrt{\mathrm{ESS}^{\mathrm{P},0}_{\mathrm{unique,ret}}(s_t)}$ because its weights are fixed and the distinct errors are uncorrelated. Cauchy–Schwarz bounds the absolute value of the second term by $\|\mathbf{b}^{\mathrm{P},\mathrm{ret}} - \mathbf{b}^{\mathrm{P},0}\|_2\|\boldsymbol{\epsilon}^{\mathrm{P}}\|_2$, whose conditional root mean square is $\delta_{\mathrm{ret}}(s_t)$. The triangle inequality in conditional L^2 proves Equation 36, with no independence assumption between the final weights and errors. Simple PMCTS's weights are fixed given $\mathcal{F}$, so its error contribution has conditional MSE $\sigma^2/\mathrm{ESS}^{\mathrm{S}}_{\mathrm{unique}}(s_t)$. Equation 37 therefore gives the stated strict comparison. □

E Worst-Case Complexity Analysis

In this section we analyze the computation complexity of Simple PMCTS (Algorithm 1) and PMCTS (Algorithm 2) under a model with $O(MN)$ parallel workers and a batched model evaluation treated as a single parallel operation.

We begin with a brief complexity analysis of baseline MCTS, for reference.

E.1 MCTS Worst Case Complexity Analysis

MCTS is given a search budget of M *simulations*, i.e. iterations of selection, expansion, and backpropagation. Expansion takes a constant amount of time, and both selection and backpropagation scale linearly with the depth d_i of the newly expanded leaf. $d_i \leq M$ since in each iteration, the depth of the tree increases by at most one. Thus we can bound the worst-case runtime complexity by $\mathcal{O}(M^2)$.

Each expansion step adds a new node to the tree, and the rest of the operations (selection and backup) use a constant amount of space, so in total the space complexity of MCTS is $\mathcal{O}(M)$.

E.2 Simple PMCTS Worst Case Complexity

We analyze the worst-case complexity of Simple PMCTS (and next PMCTS) with respect to M parallel search iterations and N particles per iteration, corresponding to a tree size of at most MN.

Selection: Each particle can do selection completely independently, so the time taken again scales linearly in the maximum depth across particles d_i. Even though we may add up to N nodes in each iteration, the depth still only increases by at most one, so we have $d_i \leq M$ again.

Expansion: Expansion is done as a batch. We compute the value predictions, new prior policies, and add up to N nodes to the tree. This can be done in constant time by having every particle create the new node in a predetermined spot based on its index. To connect the parent nodes to the new children, each particle tries to assign its child index to the parent, and in the case of contention, one arbitrarily wins out. Each particle then checks what was written and reassigns itself to the child node that won.

Backpropagation: Each particle can independently calculate its contribution to each node by tracing its trajectory in the tree. This will take a linear time in d_i. The contributions can then be combined for each state in parallel, either with a linear scan $\mathcal{O}(N)$ or tree reduction $\mathcal{O}(\log(N))$ Kogge & Stone (1973). Thus the time for backpropagation scales as $\mathcal{O}(d_i + \log(N))$.

Total: In each iteration we take $\mathcal{O}(d_i)$ for selection, $\mathcal{O}(1)$ time for expansion, and $\mathcal{O}(d_i+\log(N))$ for backpropagation. Over M iterations this amounts to $\mathcal{O}(M(M + \log(N))) = \mathcal{O}(M^2 + M\log(N))$.

Space: Since each iteration can add at most N new nodes, the space required for the search tree is $O(MN)$. Additional space is required during backpropagation to keep track of the contribution of each particle at each affected node: There are N particles and each affected at most d_i nodes, so $\mathcal{O}(d_iN) \leq \mathcal{O}(MN)$ space.

E.3 PMCTS WORST CASE COMPLEXITY

Selection: Selection is much the same as in Simple PMCTS. The few differences are that we must now compute a sampling policy $\hat{\pi}_i$ to sample from, and the particle weights must be updated. The weight update is constant-time per step, and when using a temperature-based sampling policy as in this work, $\hat{\pi}_i$ can also be computed in constant time (independently by each particle). This gives the same linear time $\mathcal{O}(d_i) \leq \mathcal{O}(M)$ for selection.

Expansion remains identical to Simple PMCTS, so $\mathcal{O}(1)$.

Backpropagation: Most of PMCTS's features come into play in this step. We compute the unweighted returns contributed by each particle in parallel in $O(d_i)$ time. *Retrospective reweighing* can be done in parallel for each particle in constant time. De-duplication of particles can be done using *reduce-by-key* in $\mathcal{O}(\log(N))$ time Blelloch (1989) (i.e. by using a parallel sort, detecting boundaries, and then tree-reducing). Computing ESS can be done for each affected state in parallel in time $\mathcal{O}(\log(N))$ using tree reduction for sums over particle weights, and parallelism across particles for operations such as squaring or dividing weights. Calculating the weighted return similarly takes $\mathcal{O}(\log(N))$ time with tree reduction and can be done in parallel per state. In total the worst-case time complexity for backup is $\mathcal{O}(d_i + \log(N))$.

Total: Similar to Simple PMCTS, the worst-case complexity of one iteration is $\mathcal{O}(M + \log(N))$ and $\mathcal{O}(M^2 + M\log(N))$ the entire algorithm.

Space: The space required for the search tree is $\mathcal{O}(MN)$. Space required for backpropagation is dominated by the weighted return and ESS calculations which need $\mathcal{O}(d_i N) \leq \mathcal{O}(MN)$ as before to store the contribution of each particle.

F MCTS AND PARALLELIZATION: A BRIEF SURVEY

F.1 SYNCHRONOUS VS. ASYNCHRONOUS PARALLELISM

A system of parallel processes is called *synchronous* if all processes run using the same clock, i.e., all operations are executed in lockstep (physically or conceptually). It is called *asynchronous* if each process has its own independent clock, i.e. the different processes execute independently, potentially interleaving arbitrarily Arjomandi et al. (1981).

Asynchronous parallelization is very popular, as it allows multiple processes to run independently in parallel, and is indeed a popular way to implement virtual-visits based heuristic parallelization of MCTS Mirsoleimani et al. (2017).

In contrast, modern GPU acceleration with JAX Bradbury et al. (2018) enables a form of semantically synchronous parallelization, typically expressed via batching or vectorization. In this setting, computations are defined so that elements of a batch are processed independently but conceptually in lockstep, with no observable ordering or interaction between them. This has popularized synchronous parallelized acceleration of the RL framework, based on "batched" operations, for training as well as search de Vries et al. (2025); Oren et al. (2026); Danihelka et al. (2022). As a result, it is much easier to benefit from GPU acceleration with Jax with conceptually synchronous parallelization.

F.2 PARALLEL MCTS: A BRIEF SURVEY

Many approaches for parallelizing MCTS under different practical constraints have been proposed over the years Cazenave & Jouandeau (2007); Chaslot et al. (2008); Enzenberger & Müller (2009); Schäfers (2014); Mirsoleimani et al. (2018); Cazenave (2022); Malmsten & Böhmer (2025). Among them, the three canonical approaches are *root parallelization*, *leaf parallelization* and *tree parallelization* Cazenave & Jouandeau (2007); Chaslot et al. (2008).

Root parallelization parallelizes across multiple instances of MCTS. To achieve runtime inference scaling, all instances run from the same root state and aggregate their results once search has concluded. This approach builds search trees of the same size, and in expectation each tree is the same. It also relies heavily on stochasticity in the selection and/or evaluation and thus is not well suited to modern uses of MCTS that rely on fully deterministic selection and evaluation.

A variant of this approach is very effective for scaling up during training however. Instead of running multiple instances of MCTS in the same state in parallel and aggregating their results, the search can be parallelized across N *different* states. The results are then aggregated in a training batch over these states. This approach is the go-to in JAX-based implementations of training RL agents with MCTS for policy improvement (e.g. AlphaZero variants) DeepMind et al. (2020).

Leaf parallelization parallelizes the evaluation of the same leaf across N parallel processes. This is only effective when evaluation is stochastic, and thus is not suited for modern algorithms either.

Tree parallelization parallelizes the search with multiple parallel, synchronous or asynchronous, *selection, expansion* and *backpropagation* processes operating on the same tree. The immediate challenge these methods run into are duplicate selection trajectories. To overcome this challenge, the canonical approach in practice is the *virtual visits* family of heuristics. To reduce the probability that different processes attempt to expand the same leaf, the main idea is that when a process arrives at a node during selection, it appends a count of "virtual" visits *before* the expansion and backpropagation steps complete. These visits are taken into account by other processes arriving at the same node, *as if* the node was fully updated with backpropagation.

This approach is very popular in practice, and has been very successful for benefiting from parallel compute during inference time. However, it is not principled, in the sense that there are no guarantees over which policy is evaluated by the tree at each iteration.

Additional approaches have iterated on the notion of virtual visits with *informed*-virtual visits, using smaller DNNs or other forms of cheaper evaluations to direct the search, while spending the cost for expensive evaluations only later or only on specific nodes (Cheng et al., 2024; Kim et al., 2021; Lan et al., 2019).

Prior work analyzed this direction theoretically from the perspective of *how much worse than regular MCTS can the decisions induced by virtual visits be* (Liu et al., 2020). In contrast, PMCTS recovers the expected-backpropagation limit of MCTS as $N \to \infty$ and uses parallel sampling to reduce Monte Carlo error under the classical assumptions.

Overall, a large number of approaches have investigated using virtual visits in different implementations with different levels of synchronous and asynchronous parallelization (Chaslot et al., 2008; Cazenave & Jouandeau, 2007; Enzenberger & Müller, 2009; Schäfers, 2014; Mirsoleimani et al., 2018; Cazenave, 2022).

Other approaches for parallelizing MCTS have been proposed. Malmsten & Böhmer (2025) replace the DNN-learned dynamics model of MuZero Schrittwieser et al. (2020) with a transformer, to unroll multiple action trajectories in the tree in a batch. This approach is not suited for MCTS in general, only for DNN-based dynamics models, modifies the backpropagation of MCTS, and does not provide any theoretical analysis or guarantees of policy improvement with respect to the modified selection policies.

G Additional Results

Scaling across M in an example environment is investigated in Figure 4.

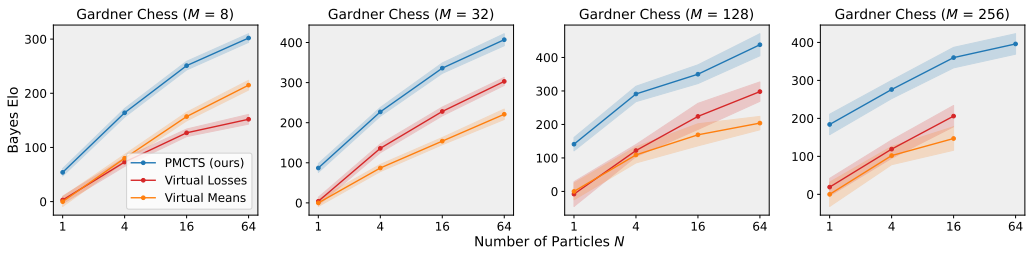


Figure 4: Scaling of parallel MCTS variants with parallel compute (number of particles N) and simulation budgets M in Gardner chess. Bayes Elo with 95% CI.

The individual boardgames results aggregated in Figure 1 are presented in Figure 5.

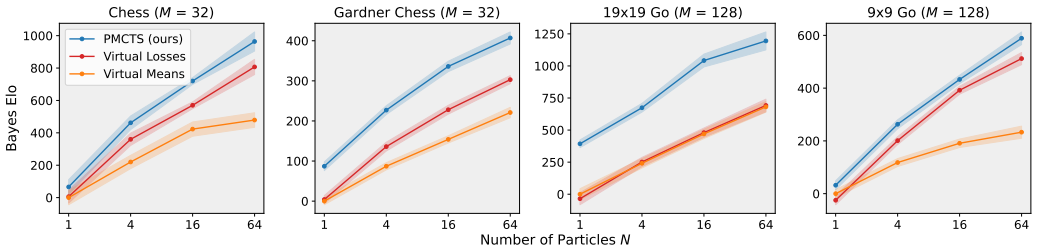


Figure 5: Scaling of tree parallel MCTS variants with parallel compute (number of particles N) in different boardgames. Bayes Elo with 95% CI.

The individual discrete and continuous action results aggregated in Figure 1 are presented in Figure 6.

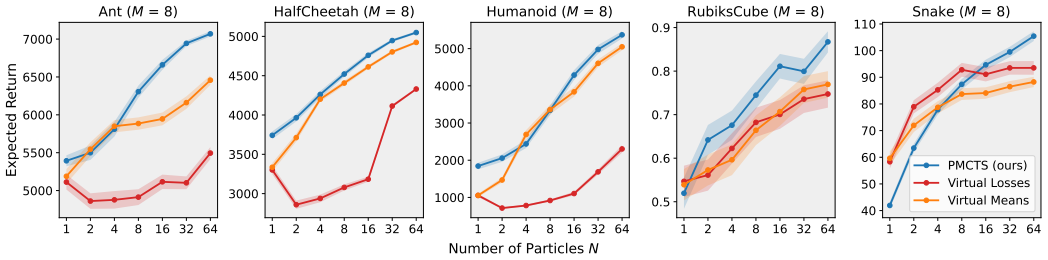


Figure 6: Scaling of parallel MCTS variants with parallel compute (number of particles N) across different environments in the discrete and continuous domains. Mean expected return with 95% Gaussian CI.

A comparison between tree-parallel MCTS with PMCTS and root-parallel MCTS is presented in Figure 7, showing that, in the evaluated deterministic-DNN settings, root-parallel GumbelMCTS does not scale inference at runtime effectively with additional parallel compute. For this comparison, we use a modern version of non-parallel MCTS, GumbelMCTS, and run it vectorized across different instances at the same state, which also allows us to compare the performance of PMCTS to a popular version of non-parallel MCTS.

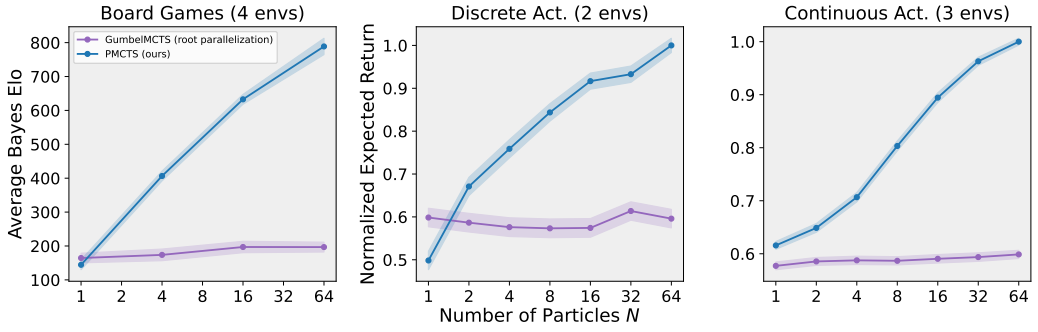


Figure 7: Scaling of tree parallel MCTS with PMCTS vs. root parallel MCTS with GumbelMCTS across boardgames, discrete and continuous environments. Average Bayes Elo or expected return with 95% CI.

In Figure 8 we investigate the effect of different action selection mechanisms across the different baselines. In this representative 9x9 Go setting, the action-selection mechanism used by each baseline in all other experiments is the strongest among the alternatives evaluated for that baseline. *Improved Policy* refers to selecting the action that maximizes the improved policy. *Max. Visits* chooses the action with the most visits. For GumbelMCTS, we compare different ways to aggregate across trees, specifically averaging the improved policy (*Aggregate Policy*), averaging the Q-values (*Aggregate Values*), and choosing the action picked by the most independent searches (*Aggregate Voting*), see Appendices I.3 and H.2 for more detail.

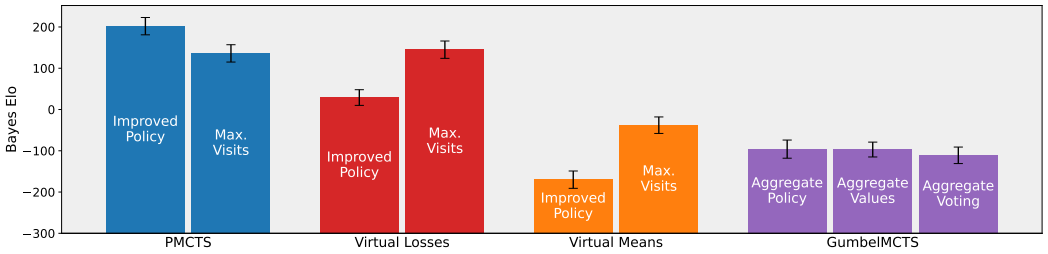


Figure 8: Action selection ablations across the different baselines, in 9x9 Go ($M = 128, N = 16$). Bayes Elo with 95% CI.

We include a comparison to TSMCTS and SMC on Gardner chess and 9x9 Go, demonstrating their challenge to perform well with a larger simulation budgets (and smaller particle budgets, in the case of TSMCTS), in Figure 9.

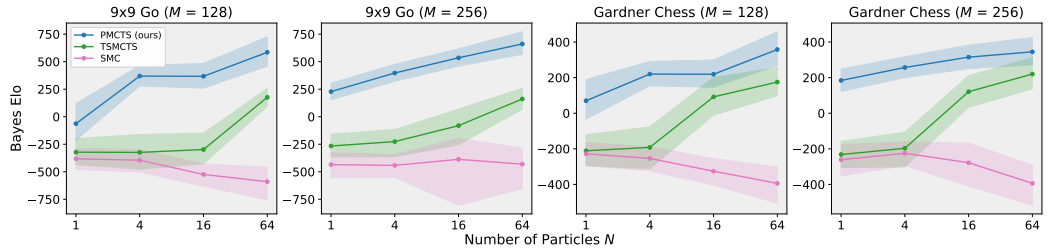


Figure 9: PMCTS vs. TSMCTS and baseline SMC in Gardner chess and 9x9 Go, with larger simulation budgets $M = 128, 256$. Bayes Elo with 95% CI.

To investigate how the temperature η trades off trajectory diversity against divergence from the target π_i, Figure 10 reports search statistics averaged across trees rooted at different states in 9x9 Go, with $M = 128$ and $N = 64$ unless otherwise stated. The left panel shows that the number of unique trajectories increases with η. The second panel shows the corresponding increase in $\mathrm{KL}(\hat{\pi}_i \| \pi_i)$. The third panel reports $1/\mathrm{ESS}$, the estimator of the variance. Among the evaluated temperatures, $\eta = 1.5$ consistently gives the lowest value variance estimate and the right panel shows that the same temperature also achieves the best performance, which supports the inverse ESS as a variance estimator on the one hand, and demonstrates how aiming to minimize the variance of the update results in improved performance on the other hand.

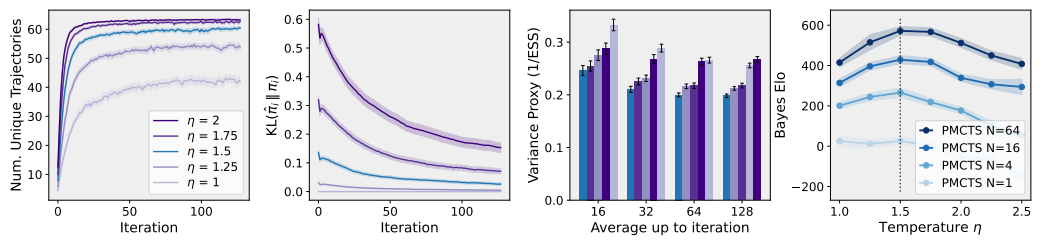


Figure 10: Search statistics for PMCTS on 9x9 Go with $M = 128$. Means and 95% Gaussian or Bayes Elo CIs, where relevant. Among the evaluated temperatures, $\eta = 1.5$ gives the lowest estimated variance (second from right) and the highest observed performance (right).

H Implementation Details

H.1 The search policies PUCT and UCT

$$\pi_i^{\text{UCT}}(s) := \arg\max_{a \in A} q_i(s_t, a) + C\sqrt{\frac{\log M_i(s_t)}{1 + M_i(s_t, a)}}. \tag{39}$$

$$\pi_i^{\text{PUCT}}(s) := \arg\max_{a \in A} q_i(s_t, a) + \pi_\theta(a|s_t)\left(c_{\text{init}} + \log\left(\frac{M_i(s) + c_{\text{base}} + 1}{c_{\text{base}}}\right)\right)\frac{\sqrt{M_i(s_t)}}{1 + M_i(s_t, a)}. \tag{40}$$

H.2 Improved Policy, Value and Action Selection - A Brief Survey of Design Choices

At the end of search, P/MCTS returns an action, a policy target π_{search} for π_θ, and a value target V_{search} for v_ϕ. In this appendix q_M denotes the post-search values after M simulations; Algorithms 1 and 2 denote those same values by q_{M+1} because their loops start at $i = 1$. This can be (and is) done in multiple different ways, which we discuss below.

MCTS for Heuristic Policy Improvement: In AlphaZero (Silver et al., 2016; 2018), MCTS is used to train the policy. As prior to Grill et al. (2020)'s work, MCTS was not theoretically interpretable as a policy improvement operator, it was not obvious what target to extract for the prior policy. Silver et al. (2016) chose the visitations distribution $M(s_1, a)$:

$$\pi_{\text{search}}^{\text{classical}}(a|s_1) = \frac{M(s_1, a)}{\sum_{b \in \mathcal{A}} M(s_1, b)} \tag{41}$$

Although this does not guarantee policy improvement (Danihelka et al., 2022), it has been extremely successful in practice and paved the way for the great influence MCTS-based agents had on RL.

It is natural to then compute the root value and select an action to take as:

$$V_{\text{search}}(s_1) = \sum_{a \in \mathcal{A}} \pi_{\text{search}}^{\text{classical}}(a|s_1) q_M(s_1, a), \quad a \sim \pi_{\text{search}}^{\text{classical}}(s_1). \tag{42}$$

When we want a deterministic action, we can select the action with the most visits $M(s_1, a)$.

MCTS as a Policy Improvement Operator Grill et al. (2020) showed that MCTS *tracks*, *approximates* and therefore *can be interpreted as* a policy improvement operator. More specifically, their results can be summarized as follows: (I) The popular selection policies UCT and PUCT *converge over iterations* (i.e., track) to the solution of an optimization problem of the form:

$$\pi'(s) = \arg\max_{\pi'} \sum_{a \in \mathcal{A}} \pi'(a|s) q_i(s, a) - \lambda(M_i(s)) D(\pi', \pi_\theta). \tag{43}$$

Here, D is some regularization term and $\lambda(M_i(s, a))$ is a function of the visitation counts $M_i(s)$. Further, Grill et al. (2020) showed that PUCT corresponds to the specific regularization term $D(\pi', \pi_\theta) = KL(\pi_\theta, \pi')$, i.e. the reverse KL divergence. We apply the following definition of the $KL(x, y) = \sum_{a \in \mathcal{A}} x(a) \log \frac{x(a)}{y(a)}$.

Next, Grill et al. (2020) showed that the solution π' (Equation 43) can be used directly, for three purposes: First, for selection actions in the environment. Second, for training the policy. And third, for driving the search itself, replacing deterministic PUCT with stochastic π'.

The operator $\pi' = \mathcal{I}(\pi_\theta, q_M)$ which solves Equation 43 is not a very popular operator, and requires solving an optimization problem numerically to find the normalization constant of π'. Danihelka et al. (2022) showed that it can be replaced by $\mathcal{I}_{\text{reg}}$ effectively.

Accordingly, this suggests the following natural return values:

$$\pi_{\text{search}}(a|s_1) = \mathcal{I}(\pi_\theta, q_M), \; V_{\text{search}}(s_1) = \sum_{a \in \mathcal{A}} \pi_{\text{search}}(a|s_1) q_M(s_1, a), \; a \sim \pi_{\text{search}}(s_1). \tag{44}$$

Sequential Halving and Completed Qs: Danihelka et al. (2022) introduced *Sequential Halving* (SH) to search at the root. SH is a bandit algorithm which optimizes for *simple-regret, best action identification* given *known* search budget M. In contrast, PUCT was designed for *anytime, cumulative-regret, best action identification*. SH can be interpreted in two different ways: First, it is just a budget-allocation algorithm, which assigns different actions different action budgets, without a direct influence on the action values themselves. One qualification is that the backed-up values are nonstationary: later simulations evaluate different selection policies as the tree is updated. Kocsis & Szepesvári (2006) model this nonstationarity as drift. Root budget allocation changes which actions are evaluated and therefore the resulting search values q_i; each expected backup still evaluates the selection policy specified for that batch under the assumptions of Appendix D.

Second, SH can be viewed as a *best-action selection* algorithm. At the last iteration of SH, it outputs a recommended-best action. Danihelka et al. (2022) used SH to select actions in the environment. However, they found that in order to retain sufficient exploration in board games, the agents benefited from sampling proportional to $\pi_{search}^{classical}$, the policy induced directly by the visitation counts $M(s,a)$.

Additionally, Danihelka et al. (2022) proposed the completed Q values, which predicts the value $q(s,a)$ of unvisited actions $M(s,a)=0$ with a mixture of the form:

$$q_i(s,a) = \begin{cases} q_i(s,a) & \text{if} M(s,a) > 0 \\ v_{\text{mix}}(s) & \text{if} M(s,a) = 0 \end{cases}, \tag{45}$$

$$v_{\text{mix}}(s) = \frac{1}{1+\sum_a M(s,a)} \left(v_\phi(s) + \frac{\sum_a M(s,a)}{\sum_{a\in A} \pi_\theta(a|s)} \sum_{a\in A} \pi_\theta(a|s) q_i(s,a) \right), \tag{46}$$

$$A = \{a : M(s,a) > 0\}. \tag{47}$$

The value $v_{\text{mix}}(s)$ is intended as a more stable estimator of $V^{\pi_\theta}(s)$ than the single DNN prediction $v_\phi(s)$. Danihelka et al. (2022) showed that using estimators of this form preserves policy improvement.

This distills to the following return values:

$$\pi^{\text{GMZ}}_{\text{search}}(s_1) = \mathcal{I}_{\text{reg}}(\pi_\theta, q_M)(s_1), \quad V_{\text{search}}(s_1) = \sum_{a\in\mathcal{A}} \pi^{\text{GMZ}}_{\text{search}}(a|s_1) q_M(s_1,a), \tag{48}$$

$$a \sim \pi^{\text{classical}}_{\text{search}}(s_1), \quad \text{or} \quad a = \text{SH recommended action}. \tag{49}$$

Putting it all together Recent work from the SMC for search literature (Oren et al., 2026) uses SH only for search budget allocation at the root s_1 to great benefit. Using SH only for root budget allocation changes which values are observed, while the returned policy is still computed by applying $\mathcal{I}$ to the resulting search values q_M.

However, the completed Qs, although sound for policy improvement, may result in large mass for unvisited actions. This results in potentially a very high entropy policy for action selection in the environment, which may be catastrophic in environments where one bad move can terminate the agent (falling off of cliffs, or moves leading to immediate loss in boardgames).

Putting all of this together, we opt for the following:

$$\pi_{\text{search}}(a|s_1) = \mathcal{I}(\pi_\theta, q_M), \quad V_{\text{search}}(s_1) = \sum_{a\in\mathcal{A}} \bar{\pi}(a|s_1) q_M(s_1,a), \quad a \sim \bar{\pi}(s_1). \tag{50}$$

$$\bar{\pi}(a|s_1) \propto \begin{cases} \pi_{\text{search}}(a|s_1) & \text{if:} \quad M(s_1,a) > 0 \\ 0 & \text{if:} \quad M(s_1,a) = 0 \end{cases} \tag{51}$$

The policy $\bar{\pi}$ has mass only on visited actions, and otherwise is proportional to π_{search}. As a result, the root value $V_{\text{search}}(s_1)$ only incorporates q_M of visited actions (e.g. no completed Q values participate in the value computation). Similarly, the acting policy is not over-entropic and only incorporates information from visited actions (e.g. again, no completedQ values participate in the value computation). $\bar{\pi}$ and can be thought of as a more principled way to achieve the sampling used by GumbelMCTS $a \sim \pi^{\text{classical}}_{\text{search}}(s_1)$.

Using V_{search} to generate value targets (Oren et al., 2025a) has been shown to be sound by Oren et al. (2025b), despite V_{search} not being the value of the policy π_θ.

H.3 The Virtual Losses and Virtual Means Heuristics

We use two popular variations of *virtual visits*: Virtual Means (VM) Cazenave (2022) and Virtual Losses (VL) (Mirsoleimani et al., 2017; Chaslot et al., 2008). Under these heuristics, the deterministic search policy is computed as follows:

$$\pi^{\text{PUCT}}_{\text{VL},i}(s) := \arg\max_{a\in A} q_i^{\text{VL}}(s_t,a) + \pi_\theta(a|s_t)\, C_i(s_t) \frac{\sqrt{M_i(s_t) + M_i^{\text{virtual}}(s_t)}}{1 + M_i(s_t,a) + M_i^{\text{virtual}}(s_t,a)}. \tag{52}$$

$$\pi^{\text{PUCT}}_{\text{VM},i}(s) := \arg\max_{a\in A} q_i(s_t,a) + \pi_\theta(a|s_t)\, C_i(s_t) \frac{\sqrt{M_i(s_t) + M_i^{\text{virtual}}(s_t)}}{1 + M_i(s_t,a) + M_i^{\text{virtual}}(s_t,a)}. \tag{53}$$

$$C_i(s_t) := c_{\text{init}} + \log\left(\frac{M_i(s_t) + c_{\text{base}} + 1}{c_{\text{base}}}\right). \tag{54}$$

This is the PUCT formula Rosin (2011) but including the virtual visits added by unevaluated particles / processes. q^{VL} is q_i except each virtual visit is treated as a *loss*, i.e. a return of -1. We keep the implementation of VM and VL consistent with mctx's DeepMind et al. (2020) PUCT implementation: the Q values of unvisited actions are completed with the value prediction $v_\phi(s)$ and normalized after virtual losses are added. After normalization, the normalized Q-values for unvisited actions are set to 0.

H.4 Sequential Halving with P/MCTS

GumbelMCTS Danihelka et al. (2022) modifies the search at the root of the tree with Sequential Halving (SH, Karnin et al. (2013)). The SH algorithm is a budget-allocation algorithm, which takes as input *number of actions to search* K and divides the search into $\log_2 K$ iterations. Each iteration is assigned a search budget $B = M/\log_2 K$. At each iteration $i \geq 1$, the number of searched actions halves $K_i = K/2^{i-1}$, and assigned equal budget: B/K_i.

When implementing SH with particles, the budget is treated as MN instead of M, and divided accordingly. To keep the parallelization across N intact, N particles are used during search at every iteration. See Oren et al. (2026) for more detail.

SH interacts with PMCTS: on the one hand, it allows PMCTS to drive diversity by setting large $K = N$ for example. On the other hand, in later iterations, all particles are forced to search the same actions, which can reduce trajectory diversity. As a result, it's not clear whether SH should increase performance or reduce performance when coupled with PMCTS. In our experiments, we were not able to see benefit from using a parallel, particle-based variation of SH at the root of PMCTS.

H.5 Baselines

All baselines are implemented within our PMCTS framework written in JAX Bradbury et al. (2018). Virtual Means / Losses are thus also implemented as batch-parallel and synchronous. This means that all "particles" (what would be threads or processes) have to select trajectories fully sequentially in order to communicate the virtual losses added. If we attempted to select all actions simultaneously all "particles" would pick the same action since Virtual Means / Losses are deterministic. Their backpropagation is the same as used by PMCTS, but without the PMCTS-specific features such as ESS, de-duplication, etc. As a result, Virtual Means / Losses only parallelize the expansion well, which is not the best case for these algorithms. In more standard implementations of Virtual Means / Losses, selection and backup are done asynchronously with atomics or locks to deal with the situation where multiple threads are at the same node at the same time. Even in these implementations however, in the worst case, when *all* threads are at the same node at the same time, the selection is effectively sequential. This fine-grained shared-tree asynchronous pattern is not expressible within our fully JIT-compiled batched JAX/XLA implementation without leaving the staged array-programming model or introducing custom synchronization primitives.

GumbelMCTS and PUCT MCTS are implemented in our setup, to keep comparison as direct as possible, based on the official implementation DeepMind et al. (2020).

Root-parallelization with GumbelMCTS We conduct independent search on N copies of the state with different random keys and then aggregate the results. In continuous action environments the action space at the root is shared by sampling K actions from π_θ once and treating this as a discrete action space Hubert et al. (2021). From that point on, every search tree is unique. Once search has concluded, we aggregate across search trees by taking either the mean of the Q values (in continuous environments) or the mean of the policy (in discrete environments): $\bar{q}_M(s,a) = \frac{1}{N}\sum_{i=1}^{N} q_M^i(s,a)$ or $\bar{\pi} = \frac{1}{N}\sum_{i=1}^{N} \pi_M^i$.

I Experimental Details

I.1 Environments and Scaling Experiments Across Domains

We evaluate PMCTS empirically across four domains and 10 environments: four two-player zero-sum board games (chess, Gardner chess, 19x19 Go and 9x9 Go) implemented by PGX Koyamada et al. (2023); Two standard discrete-control single-agent problems (Snake and RubiksCube) implemented by Jumanji Bonnet et al. (2024); Three classical continuous control tasks (Ant, Humanoid and HalfCheetah) implemented by Brax Freeman et al. (2021). One LLM-scaling inference in the environment Sokoban Bonnet et al. (2024) described in more detail in Appendix I.8.

We have DNN models (v_ϕ and π_θ pairs) for each environment: In 9x9 Go and Gardner chess we use the baseline models provided by PGX. For full chess and 19x19 Go we trained our own networks as PGX does not supply those (architecture is described in Appendix J.2). In the discrete and continuous control environments we use three models (three pairs), trained independently for different numbers of steps. We use checkpoints prior to convergence to keep the evaluation as meaningful as possible.

For each environment, agent, particle budget N and model, we run multiple evaluation episodes. In the board game environments, this amounted to each variant (agent and particle budget N) playing multiple games against each other variant, and computing Bayes Elo, which also provides a 95% confidence intervals. The games started from positions in an opening book, generation of which is explained in I.2. In the single-player environments, this amounted to each variant (environment, agent, particle budget N and model) running 256 episodes. We then compute the mean and a ± 2-SEM based $> 95\%$ confidence interval. In the single-player environments, every combination of environment, agent, particle budget N, and model was evaluated for 256 episodes. For each environment–agent–N combination, we pool the returns from all three models and report their mean and the Gaussian interval

$$\bar{R} \pm 1.96 \, \frac{s_R}{\sqrt{3 \cdot 256}},$$

where s_R is the standard deviation across the pooled model–episode returns. Thus, these intervals treat model–episode pairs as the evaluation units.

For the aggregated results in Figure 1, each return is first normalized within its environment using zero as the minimum observed return and the maximum observed return across all evaluated agents as the upper endpoint. We then pool the normalized returns across the environments in the corresponding domain and compute the mean and Gaussian interval in the same manner.

I.2 Opening Book

All agents in the board-game scaling experiments are deterministic. Starting every match from the initial position would make all matches identical, so we use an opening book: a set of starting states. Each environment's book contains 20,000 unique positions, generated by sampling from the policy network up to a predetermined move count, then filtering for roughly balanced positions. A position is roughly balanced when that model's value prediction lies in $[-0.3, 0.3]$. The number of moves sampled per environment is given in Table 2.

I.3 Acting in the Environment

The different agents act as follows: PMCTS selects the action that maximizes the visited-action acting policy at the root: $\arg\max_a \bar{\pi}(a \mid s_1)$, with $\bar{\pi}$ defined in Equation (51). Virtual Means / Losses act in the standard manner of this heuristic, by taking the action with the most visitations:

$\arg\max_a M(s_1, a)$. The aggregated GumbelMCTS is implemented as taking the action which maximizes the *aggregated* improved policy (Appendix H.5). See Figure 8 for the effect of action selection for each agent.

In the learning experiment, GumbelMCTS picks an action stochastically from the improved policy in the first 8 moves, and after that picks deterministically the action recommended by SH. Similarly, in the learning experiment PMCTS selects by sampling from $\pi_{search}(s)$ at the first 8 actions, and then deterministically taking the $\arg\max_a \bar{\pi}(a|s)$ (see Appendix H.2).

I.4 Evaluation Metrics

In the standard RL environments, we use the observed averaged return across multiple independent evaluation episodes. In the board game environments, we use BayesElo Coulom (2005), a commonly used metric for comparing the relative strength of agents in two-player zero-sum games. Elo rating difference is related to the win-rate between agents. Bayes Elo finds the maximum likelihood ratings using MM Hunter (2004) where the prior on the Elo distributions is assumed to be uniform. Elo is shift agnostic: only the relative Elo between agents matters, not the absolute score. To compute the average Elo across board games we shift a single agent to 0 in every environment (the $N = 1$ Virtual Means), shift the Elo scores of all other agents in each environment accordingly, and then take the average across environments per agent. For the policy improvement experiment in Go, we plot the win-rate against the PGX baseline agent.

We configure BayesElo with first-player advantage fixed at zero, since whenever we simulate a match between two agents, we play the same opening from both sides. Reported intervals are BayesElo's 95% confidence intervals.

I.5 Discrete vs. Continuous Action Spaces

To extend PMCTS and MCTS to continuous actions we follow the popular approach of SampledMZ (Hubert et al., 2021), which samples C actions from π_θ at each node in the search tree and treats $\{a_1, \ldots, a_C\}$ as a discrete action space with uniform logits.

I.6 Runtime Experiments

We measured the runtime for every algorithm at every particle budget (powers of 2 from 1 to 128) with a simulation budget of 128. Each measurement consisted of 3 warmup runs to remove the overhead of JIT compilation, followed by 10 trials of 10 runs each. Every run used a different input state (generated ahead of time) to reflect real use. Each algorithm was tested using the same hardware (one NVIDIA A100 GPU), neural network, dependency versions, drivers, sequence of input states, etc. We plot the mean across trials, in effect the mean across all 100 runs, and the 95% confidence intervals computed from the trials.

I.7 Policy Improvement Experiment

We use the baseline implementation of an on-policy AlphaZero training loop for the policy improvement experiments on board games by Koyamada et al. (2023). The training hyperparameters (learning rate, batch sizes, etc.) were tuned for the baseline GumbelMCTS based AlphaZero, and were not modified and thus are likely to favor the baseline agent (purple line in Figure 2, right).

I.8 LLM inference scaling on Sokoban

In this experiment, the LLM is playing instances of the game of Sokoban in which we use text-based rendering (ASCII). Each Sokoban instance is a procedurally generated solvable 1-box puzzle. We sample 200 instances which we verify with breadth-first search are solvable within a 20-step window. The model we used was Qwen2.5-14B-Instruct. The LLM supplies the prior policy over the actions to search at each step. The search action space contains five actions: the four moves plus a No-Op. Policy priors are formed from 8 independent completions per node at temperature 1.25, parsed strictly from <action>...</action>; completions that fail to parse are attributed to the No-Op action, so malformed output becomes explicit prior mass rather than being silently discarded. A No-Op

consumes an environment step. For the value approximation at the leaves uniform random rollouts are used with a true model of the environment.

All methods were evaluated on the same 200 puzzle instances. Each environment action used $M = 8$ parallel search iterations and $N \in \{1, 2, 4, 8, 16, 32\}$ particles. Each newly evaluated leaf used 8 independent random rollout(s), each continuing until termination.

The system prompts used were:

```
You are solving a Sokoban puzzle. You control the player (P)
    on a grid.
Your goal: push all boxes (X) onto the targets (O).

Symbols:
  \# wall, \_ floor, O target, P player
  X box, V box on target, S player on target.

Rules:
- Move up, down, left, or right.
- Push a box by moving into it (space behind must be empty).
- You cannot pull boxes or push two at once.
- Boxes cannot be pulled back.

A box pushed into a corner or against a wall may be stuck
    forever - plan multiple steps ahead before pushing.

First reason about which moves are useful inside <think>...</
    think> tags.
Then choose exactly one action inside <action>...</action>
    tags.
Example: <think>The box is above me and the target is above
    the box. Pushing up will place it on the target without
    blocking anything.</think><action>up</action>

Then in each step, the system prompt is concatenated with this
     message:

Step \{step\_count\} of \{max\_steps\}.

\{board\_text\}

Moves so far: \{action, action, ...\} (only present if move\
    _history is non-empty).

Your move?
```

The system prompt additionally contains three worked examples, each showing a small board and the corresponding <think>/<action> response; these are given verbatim in the supplementary code. Each prompt contains the current board and the list of actions taken so far, but not the previous boards or the model's earlier reasoning traces; carrying the full interaction history would substantially increase runtime.

J HYPERPARAMETERS

J.1 HYPERPARAMETER TUNING

All hyperparameters except the temperature parameter η introduced in this work follow Danihelka et al. (2022) as published in DeepMind et al. (2020). Search over η is presented in Figure 3, where $\eta = 1.5$ dominates all other constant temperatures. We leave the investigation of more complex

temperature schedules depending on state, policy entropy, number of particles at the state, etc. to future work.

J.2 NETWORK ARCHITECTURES

Different architectures are used in different environments. In the discrete-action and continuous-action single-agent environments, we use the architectures of de Vries et al. (2025). These are MLPs in all environments except Snake, where a CNN followed by an MLP is used. See de Vries et al. (2025) for a full specification.

In the board game experiments we use the PGX Koyamada et al. (2023) baseline models for 9x9 Go and Gardner chess: small AlphaZero-style ResNets with 6 layers and 128 filters. For chess and 19x19 Go where we trained our own models, we used the same architecture as the PGX models, except for 19x19 Go we used 16 residual blocks instead of 6.

J.3 HYPERPARAMETERS

This subsection covers the hyperparameters used for the experiments. Table 1 gives the hyperparameters shared among experiments: These are the PUCT parameters used by Virtual Means and Virtual Losses, and the Q-value scaling used by both PMCTS and Gumbel MCTS. Table 2 lists hyperparameters specific to the experiments in 9x9 Go and Gardner Chess. Table 4 details the episode length per environment used. The sampled actions refer to how many actions we sample from the continuous action space for search. Finally, Table 5 gives hyperparameters for the policy optimization experiment.

Table 1: Shared Hyperparameters for All Experiments

Category	Hyperparameter	Value
PUCT (used by Virtual Means / Losses)	c_{base}	19652
	c_{init}	1.25
Q-value Scaling for the Improved Policy	β_i	$(c_{\text{visit}} + \max_a M_i(s,a))\, c_{\text{scale}}$
	c_{visit}	50
	c_{scale}	0.1

Table 2: Hyperparameters for Board Game Experiments

Category	**Hyperparameter**	**Value**
Gumbel MCTS	Sampled Actions	16
	Gumbel Scale	0.0
PMCTS	Temperature η	1.5
Model	Model for 9x9 Go	PGX baseline v0
	Model for Gardner Chess	PGX baseline v0
Opening Book	Size per Environment	20000
	Allowed Value Range	$[-0.3, 0.3]$
	Steps for 9x9 Go	12
	Steps for 19x19 Go	12
	Steps for Gardner Chess	8
	Steps for Chess	8

Table 3: Training hyperparameters for the chess and 19x19 Go evaluation networks.

Hyperparameter	Chess	19x19 Go
Search used in self-play	GumbelMCTS	GumbelMCTS
Simulation budget M	24	32
Considered actions	8	16
Discount	0.997	0.997
Self-play steps per iteration	256	512
Self-play batch size	1024	1024
Training batch size	256	512
Learning rate	10^{-3}	10^{-4}
Momentum	-	0.9
Training iterations	250	670

Table 4: Hyperparameters for Continuous and Discrete Environments

Environment	**Type**	**Episode Length**	**Num. Sampled Actions**
Snake	Discrete (jumanji)	4000	-
RubiksCube	Multi-Discrete (jumanji)	20	-
Ant	Continuous (brax)	1000	30
HalfCheetah	Continuous (brax)	1000	30
Humanoid	Continuous (brax)	1000	30

Table 5: Hyperparameters for Policy Optimization Experiment (AlphaZero Training)

Category	**Hyperparameter**	**Value**
Network	Architecture	ResNet V2
	Number of Filters	128
	Number of Residual Blocks	6
Self-Play	Batch Size	1024
	Simulations per Step	24
	Max Number of Steps	256
	Random Plies	8
Gumbel MCTS	Sampled Actions	8
	Gumbel Scale	1.0
PMCTS	Number of Particles	4
	Temperature η	1.5
Training	Batch Size	256
	Discount	0.997
	Optimizer	Adam
	Learning Rate	0.001