Enhancing Reinforcement Learning Through Guided Search

Jérôme Arjonilla[a,*], **Abdallah Saffidine**[b] **and Tristan Cazenave**[a]

[a]PSL University - Dauphine, Paris, France
[b]Potassco Solutions, Potsdam, Germany
ORCID (Jérôme Arjonilla): https://orcid.org/0000-0002-0082-1939, ORCID (Abdallah Saffidine): https://orcid.org/0000-0001-9805-8291, ORCID (Tristan Cazenave): https://orcid.org/0000-0003-4669-9374

Abstract. With the aim of improving performance in Markov Decision Problem in an Off-Policy setting, we suggest taking inspiration from what is done in Offline Reinforcement Learning (RL). In Offline RL, it is a common practice during policy learning to maintain proximity to a reference policy to mitigate uncertainty, reduce potential policy errors, and help improve performance. We find ourselves in a different setting, yet it raises questions about whether a similar concept can be applied to enhance performance *i.e.*, whether it is possible to find a guiding policy capable of contributing to performance improvement, and how to incorporate it into our RL agent. Our attention is particularly focused on algorithms based on Monte Carlo Tree Search (MCTS) as a guide. MCTS renowned for its state-of-the-art capabilities across various domains, catches our interest due to its ability to converge to equilibrium in single-player and two-player contexts. By harnessing the power of MCTS as a guide for our RL agent, we observed a significant performance improvement, surpassing the outcomes achieved by utilizing each method in isolation. Our experiments were carried out on the Atari 100k benchmark.

1 Introduction

Reinforcement Learning (RL) is a leading field in artificial intelligence, advancing our grasp of intelligent decision-making in complex environments [3, 44]. Despite the remarkable progress, the pursuit of optimizing RL algorithms remains a central focus. In this pursuit, we turn our attention to a foundational concept within the realm of RL. In Offline RL [32, 36], the primary objective is to derive the best possible policy solely from a dataset originating from an auxiliary policy, without interacting with the environment. The prevalent idea is to align the new policy closely with the auxiliary policy to enhance performance. This strategy derives from the principle that deviating from the limits of the auxiliary policy often leads to uncertainty which leads to erroneous judgments about the policy's efficacy.

Our scenario diverges from this framework and pivots back to a more classical approach where the constraints of an auxiliary policy fade away and we once again interact with the environment. Despite this paradigm shift, we question whether it is possible to preserve the concept of Offline RL *i.e.*, staying as close as possible to an auxiliary policy to enhance performance. Considering our lack of auxiliary policy, we inquire whether it is plausible to use an online algorithm proficient enough to act as our guiding reference, and how to integrate such a guiding agent into our RL agent.

In our investigation, we initially explore various online algorithms that can potentially serve as a guide. The existing literature presents algorithms that already exploit guide knowledge to improve performance. For instance, algorithms such as Soft Actor-Critic (SAC) and Asynchronous Advantage Actor Critic (A3C) [17, 18, 34] integrate an entropy term into the reinforcement learning (RL) agent. This entropy, in another formulation, is a measure of the distance between the current policy and the policy of a guide, of which this guide happens to be a random agent.

In our research, we turn our attention to search algorithms, specifically focusing on Monte Carlo Tree Search (MCTS) as a guiding policy for reinforcement learning (RL) agents. MCTS-based approaches, well-established in game theory literature [7, 45], obtain state-of-the-art performance across a spectrum of games, converging towards equilibrium even in complex scenarios involving one or two players.

Integrating MCTS as a guide yields significant performance improvements. Our analysis reveals that, in the majority of cases, this integration leads to enhanced performance. Even in instances where performance does not improve, the algorithms achieve optimal outcomes when compared individually. By combining an RL algorithm with MCTS as a guide, we harness the generalization and learning capabilities inherent to RL, while also capitalizing on MCTS's optimal online decision-making prowess. Furthermore, we extend our investigation by exploring various hyperparameters, with a keen focus on the degree of integration of the guide's policy. Through experimentation, we are demonstrating that it is possible to reduce the frequency of use of the guide, thereby mitigating associated overhead while retaining performance enhancements.

In Section 2, we establish the formalism and notation employed throughout the paper. Section 3 presents multiple online guides, discussing their respective strengths and weaknesses, and elucidates the process of integrating a guide into the RL agent. Particularly incorporating MCTS-based algorithms as a guide offers valuable guidance to the RL agent in several key points: the actor and the critic components. In Section 4, we conduct experiments using various guides on the Atari100k benchmark. Section 5 provides an overview of related work in the field. Lastly, Section 6 offers a summary of our findings and outlines avenues for future research.

[*] Corresponding Author. Email: jerome.arjonilla@hotmail.fr

2 Formalism and Notation

2.1 *Markov Decision Process*

A dynamic system is typically characterized by a Markov Decision Process (MDP), which is represented as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$. Here, $\mathcal{S}$ denotes the state space where $s \in \mathcal{S}$, $\mathcal{A}$ represents the action space with $a \in \mathcal{A}$, $\mathcal{T}(s^{t+1}|s^t, a^t)$ signifies the transition probability distribution governing the system dynamics, $r(s, a)$ stands for the reward function, and $\gamma \in (0, 1]$ serves as a discount factor.

Dealing with an exact MDP can impose considerable computational burdens. Utilizing an approximation of $\mathcal{M}$, known as a world model [20, 38, 21, 22], can offer significant advantages. Employing the world model for information retrieval not only expedites computations compared to exact methods but also facilitates parallel processing of state batches, particularly when computing complex tools such as N-step bootstrapped λ-returns or employing MCTS. This parallel processing is often performed on GPUs rather than CPUs, further enhancing computational efficiency.

2.2 *Reinforcement Learning*

Reinforcement learning confronts the problem of learning to control the MDP, where the agent tries to acquire a policy π, which is defined as a distribution over actions conditioned on state $\pi(a|s)$ that maximizes the long-term discounted cumulative reward defined as follow:

$$\pi^* = \max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{\mathrm{T}} \gamma^t r^t \right] \tag{1}$$

where $\tau = (s^0, a^0, r^0, \dots)$ is a sequence of states, actions, and rewards generated from the current policy. To maximize the policy π, one of the primary methods utilized is the *actor-critic* approach involves learning a critic and an actor-network. The learning can be conducted online by generating new trajectories or by leveraging a data buffer D, which comprises past trajectories $\tau_0, \tau_1, \dots, \tau_{k-1}$.

2.2.1 *Critic*

The critic aims to estimate the value functions, *i.e.* the expected cumulative rewards an agent can obtain at a state:

$$V_\pi(s^t) = \mathop{\mathbb{E}}_{a^t \sim \pi(\cdot|s^t)} \left[r^t + \gamma \mathop{\mathbb{E}}_{s^{t+1} \sim \mathcal{T}(\cdot|s^t, a^t)} \left[V_\pi(s^{t+1}) \right] \right] \tag{2}$$

The loss function of the critic $\mathcal{L}_\theta^C$ is formulated to minimize the disparity between the value target $\bar{V}_\theta(s)$ and the predicted value $V_\theta(s)$ over a batch of state.

$$\mathcal{L}_\theta^C = \mathop{\mathbb{E}}_{s \sim \mathrm{D}} \left[\mathcal{L}_\theta^{C,Sub}(s) \right] \tag{3}$$

Previous studies have emphasized the benefits of employing cross-entropy over a discrete representation in reinforcement learning [5, 38, 22, 6, 15]. This method involves the critic to learn a discrete weight distribution $p_\theta = \{p_1, ..., p_B\} \in \mathbb{R}^B$ instead of learning the mean of the distribution/ A function $y()$ is used to convert a target value into a corresponding weight distribution of size B. This leads to the following sub-loss for the critic:

$$\mathcal{L}_\theta^{C,Sub}(s) = y(\bar{V}_\theta(s))^T \log p_\theta \tag{4}$$

The value target often corresponds to the Q-Value, yet, to enhance stability, an alternative approach involves using the N-step bootstrapped λ-returns [44, 22]. These returns incorporate predicted rewards and values [39, 44] over a depth of N:

$$\begin{cases} V_\theta(s^t) & \text{if } \mathrm{N} = 0 \\ r_+^t \gamma \left((1-\lambda) V_\theta(s^{t+1}) + \lambda V_\theta^{\lambda, \mathrm{N}-1}(s^{t+1}) \right) & \text{if } \mathrm{N} > 0 \end{cases} \tag{5}$$

2.2.2 *Actor*

The actor's loss function, denoted as $\mathcal{L}_\theta^A$, is designed to maximize the expected reward by optimizing the actions that lead to states with the highest predicted values from the critic.

$$\mathcal{L}_\theta^A = \mathop{\mathbb{E}}_{s \sim \mathrm{D}} \left[\mathcal{L}_\theta^{A,Sub}(s) \right] \tag{6}$$

In the context of Atari Benchmarks, as observed in [21], authors have found it more advantageous to employ the Reinforce [47] algorithm, which is the approach adopted in our work as well. Reinforce maximizes the actor's probability of its own sampled actions weighted by the values of those actions. One can reduce the variance of this estimator by subtracting the state value as a baseline. Therefore, we obtain the following loss for the actor:

$$\mathcal{L}_\theta^{A,Sub}(s) = \mathop{\mathbb{E}}_{a \sim \pi_\theta(\cdot|s)} \left[-\ln \pi_\theta(a|s) \left(\frac{\bar{V}_\theta(s) - V_\theta(s)}{S_\theta} \right) \right] \tag{7}$$

where the term S_θ refers to the normalization factor used to stabilize the scale of returns. The normalization is carried out using an exponentially decaying average, is robust to outliers by taking the returns from the $5th$ to the $95th$ batch percentile, and reduces large returns without increasing small returns.

$$S_\theta = \max\left(1, \mathrm{Per}_{95}\left(\bar{V}_\theta(\cdot)\right) - \mathrm{Per}_5\left(\bar{V}_\theta(\cdot)\right)\right) \tag{8}$$

2.3 *Behavior Cloning*

Behavior Cloning (BC) [24] is a method employed in RL where the objective is to develop an agent capable of executing tasks closely resembling those of the demonstrator. In this approach, the agent's policy, denoted as π_{BC}, undergoes a supervised learning process to closely replicate the actions present in the dataset.

$$\pi_{BC} = \max_{\pi} \mathop{\mathbb{E}}_{(a,s) \sim \mathrm{D}} \left[\log \pi(a|s) \right] \tag{9}$$

2.4 *Search Algorithm*

Search algorithms are algorithms that aim to explore the game tree efficiently to make informed decisions that maximize the chances of winning. To do this, search algorithms are given a larger budget in the given state that they wish to solve, and during the budget they efficiently explore the different possible paths of action, thus obtaining a better estimate of the value function and a better policy in the given state.

Search algorithms encompass a diverse range of techniques tailored to handle various game scenarios, from single-player to multiplayer, and from perfect to imperfect information settings. In perfect information games like Chess or Go, where players have complete knowledge of the game state, algorithms like Minimax with Alpha-Beta Pruning [28, 13] or MCTS [7, 45] are widely employed. Conversely, imperfect information games like Poker or Skat pose additional challenges due to hidden information. In such cases, techniques like Perfect Information Monte Carlo [33], Information Set Monte Carlo Tree Search [14], or Counterfactual Regret Minimization based method [35] are utilized.

2.4.1 Monte Carlo Tree Search

MCTS is a tree search algorithm, for perfect information game that converges towards equilibrium with one and two players. At each time step of the budget, MCTS (i) selects the best path of node, (ii) expands the tree by adding a child node, (iii) estimates the child node, (iv) backpropagates the result obtained through the nodes chosen. At the end of the budget, the algorithms return the distribution of actions π_{MCTS} that has been visited, and the value V_{MCTS} obtained when running MCTS.

Starting from AlphaGo/AlphaZero (AZ) series [41, 43, 42], MCTS has been combined with neural networks to enhance performance where an actor-network is used to help the search and a critic network is used to give a better estimate of the new state. We denote π_{AZ}/V_{AZ} the information returned when running MCTS with AlphaZero. This information is then utilized to compute the sub-actor loss $\mathcal{L}_\theta^{A,Sub}(s)$ and the sub-critic loss $\mathcal{L}_\theta^{C,Sub}(s)$.

$$\mathcal{L}_\theta^{C,Sub}(s) = y(\bar{V}_{AZ}(s))^T \log p_\theta \tag{10}$$

$$\mathcal{L}_\theta^{A,Sub}(s) = \pi_{AZ}(\cdot|s)^T \log \pi_\theta(\cdot|s) \tag{11}$$

3 Guide

As mentioned in the introduction, we aim to find an online algorithm that can guide our RL agent to improve its performance. In this objective, we will first investigate the advantages and disadvantages of different guides, and then we will explain how to integrate the guide into the RL agent.

3.1 Analysis of the various guides

To thoroughly assess the efficacy of different guides and determine their suitability for guiding the reinforcement learning algorithm, we conducted a comprehensive evaluation based on multiple criteria. The gathered information is summarized in Table 1. The guides discussed are detailed below and are identified as follows ‘Human’, ‘Random’, ‘BC’, and ‘MCTS’.

The criteria take into account their capacity to be available in each state-action, if they are relevant for exploring/performance, their online and offline cost, if they can reduce the extrapolation error, and if they are time dependent. Time-dependent algorithms are those that require learning before they are operational, for example, learning a neural network. Extrapolation error [16] is an error present in Off-Policy and Offline problems that arise when the target selects actions rarely present in the dataset, affecting the accuracy of the value estimate.

3.1.1 Human

The use of guides is often associated with the use of human guides, whether for learning to drive [23, 48], for conversing with other humans [25] or even for trying to play as much as a human [4]. It is a necessity in scenarios where real-time interaction is either infeasible or the risk is too significant. The initial stages of a game present a valuable opportunity for the incorporation of human policies. During this phase, RL policies may prove ineffective, whereas human policies are directly applicable and advantageous. Unfortunately, the data are available in a restricted subset of all state-action, are expensive and complex to obtain.

3.1.2 Random

In algorithms like SAC [17] and several state-of-the-art counterparts [22], the RL agent is coupled with an entropy term to enhance exploration. In an alternative perspective, this entropy is a measure of the distance between the current policy π and the policy of a guide, of which this guide happens to be a random agent. The choice of a random agent as a guide holds distinct advantages, particularly when exploration of the state space is desired, its minimal computational cost and immediate availability make it an ideal choice in many scenarios. However, reservations emerged when considering the utility of such a guide in enhancing overall performance.

3.1.3 Behavior Cloning

In Offline RL, a common strategy involves approximating closeness to the behavioral policy that underlies the D dataset. Achieving this requires an initial step of estimating the behavioral policy by behavioral cloning. This estimate of the behavior policy is then used as a guide for RL agents. This method yields a significant advantage by minimizing extrapolation errors. By aligning the new policy closely with the behavior policy, the algorithm performs actions for which accurate approximations exist, reducing uncertainties of the new policy. However, several considerations come into play. Firstly, the guide is not inherently well-suited for exploration or enhancing performance. Secondly, the data is confined to a subspace of the state space and depends on the amount of interaction.

3.1.4 Search Algorithm

Leveraging a search algorithm as a guide stands as a reasonable choice given its constant availability in each state and its relatively low cost compared to human guidance. Particularly, in contrast to employing either a random guide or a guide relying solely on past data, search-based algorithms hold greater potential for performance enhancement due to their abilities to explore and converge toward the optimal solution. It is noteworthy, however, that while search algorithms are less expensive than human guidance, they may incur higher costs than alternative methods. Additionally, under constrained resource budgets or insufficient training of neural networks, search algorithms may encounter challenges in converging toward the optimal solution.

3.2 How to integrate a guide into the RL agent

Offline RL [32, 36] domain offers diverse methods for aligning one policy with another, contingent on the degree of closeness desired between them. Possible methods include value penalty where the

Table 1: Advantage and Inconvenient of using each guide. Yes* implies that the algorithm is relevant to improve exploration, but only if the action produced is also relevant to improve performance.

Guide Criteria	π_{Random}	π_{BC}	π^{θ}_{BC}	π_{MCTS}	$\pi_{AlphaZero}$	π_{Human}
Available at each (s,a)	Yes	No	Yes	Yes	Yes	No
Relevant for exploration	Yes	No	No	Yes*	Yes*	Yes*
Relevant for performance	No	No	No	Yes	Yes	Yes
Reduce extrapolation error	No	Yes	Yes	No	No	No
Performance is not time-dependent	Yes	No	No	Yes	No	Yes
Online Cost	Low	Low	Low	Medium	Medium	High
Offline Cost	Low	Low	Medium	Low	Medium	High

penalty term is incorporated into the reward function or policy regularization where the penalty term is incorporated after the calculation of the loss. In our work, we have chosen to implement regularization techniques.

Our approach to incorporating the guide is largely inspired by the work of Shi et al. [40], which, to our knowledge, stands as the sole study employing both policy and critic information. This choice is based on our ability to leverage the information provided by the search algorithms, especially π_{AZ} help to influence the actor policy and V_{AZ} help to shape the critic.

In the subsequent discussion, we adopt general notations that consider the possibility of multiple guides. $\mathrm{E} = \{\mathrm{E}_i\}_{i\in\mathcal{N}}$ denotes the set of guide algorithms, each exerting varying degrees of influence on the decision-making process.

3.2.1 *Critic Incorporation*

By integrating the guide into the critic, our objective is to refine the estimation of the value function by considering the insights provided by the guide. Incorporating a penalty into the critic using value regularization amounts to change from Equation (3) to equation the following new loss function $\mathcal{L}^C_\theta$:

$$\mathop{\mathbb{E}}_{s\sim \mathrm{D}}\left[\mathcal{L}^{C,Sub}_\theta(s) + \sum_{\mathbf{E}_i\in\mathbf{E}} \boldsymbol{\lambda}^{\boldsymbol{C}}_{\mathbf{E}_i}(\boldsymbol{s})\boldsymbol{\mathcal{F}}^{\boldsymbol{C}}_{\mathbf{E}_i}(\boldsymbol{V_\theta}(\boldsymbol{s}), \bar{\boldsymbol{V}}_{\mathbf{E}_i}(\boldsymbol{s}))\right] \tag{12}$$

where $\mathcal{F}^C_{\mathrm{E}_i}(,)$ is the penalty term between the guide target $\bar{V}_{\mathrm{E}_i}(s)$ and the predicted value, and $\lambda^C_{\mathrm{E}_i}(s)$ is the function weight used for regularizing the penalty term. The penalty term can be any function that evaluates the disparity, and in particular, the same function as the critic's sub-loss. Similarly, to enhance stability, one can compute the N-step bootstrapped λ-returns on the target value.

3.2.2 *Actor Incorporation*

To incorporate the guide on the actor, we used information from the guide on the actor and the critic. The use of the critic allows us to increase guidance when states are promising or have high potential. Incorporating a penalty into the actor using regularization amounts to change from Equation (6) to the following loss function of the actor $\mathcal{L}^A_\theta$:

$$\mathop{\mathbb{E}}_{s\sim \mathrm{D}}\left[\frac{\mathcal{L}^{A,Sub}_\theta(s)}{\mathbb{E}\left[|\boldsymbol{\mathcal{L}}^{\boldsymbol{A},\boldsymbol{Sub}}_{\boldsymbol{\theta}}(\cdot)|\right]} + \sum_{\mathbf{E}_i\in\mathbf{E}} \boldsymbol{\alpha}^{\boldsymbol{A}}_{\mathbf{E}_i}(\boldsymbol{s})\boldsymbol{\mathcal{F}}^{\boldsymbol{A}}_{\mathbf{E}_i}(\boldsymbol{\pi_\theta}(\cdot|\boldsymbol{s}), \boldsymbol{\pi}_{\mathbf{E}_i}(\cdot|\boldsymbol{s}))\right] \tag{13}$$

where $\mathcal{F}^A_{\mathrm{E}_i}(,)$ represents the penalty term between the actor-network and the target policy, and $\alpha^A_{\mathrm{E}_i}(s)$ is a function determining the penalty weight based on the current state. The penalty term can be any function that evaluates the disparity, yet, in Offline RL, the penalty term is often the KL divergence [49].

The loss of the actor-network significantly depends on the scale of the internal loss values. To address this, we normalize $\mathcal{L}^{A,Sub}_\theta(s)$ by the average absolute value of $\mathcal{L}^{A,Sub}_\theta(\cdot)$. This mean term is estimated over mini-batches and is solely used for scaling purposes. The weight $\alpha^A_{\mathrm{E}_i}(s) \in [\lambda^A_{\mathrm{E}_i}(s), \lambda^A_{\mathrm{E}_i}(s)\cdot\lambda^{Max}_{\mathrm{E}_i}]$ is a function that serves to emphasize the increased penalty on high-quality state *i.e.*, more weight is given to states that perform better than the target, which results in more attention toward the policy given by the guide.

$$\lambda^A_{\mathrm{E}_i}(s)\cdot\text{Clip}\left[\exp\left(\tau_{\mathrm{E}_i}\frac{\bar{V}_{\mathrm{E}_i}(s)-\bar{V}_\theta(s)}{S_{\mathrm{E}_i}}\right), (1, \lambda^{Max}_{\mathrm{E}_i})\right] \tag{14}$$

In this equation, the state's quality is assessed through the term $V_{\mathrm{E}_i}(s) - V_\theta(s)$ normalized by S_{E_i}. The normalization is carried out using an exponentially decaying average, robust to outliers by taking the returns from the $5th$ to the $95th$ batch percentile, and reduces large returns without increasing small returns.

$$S_{\mathrm{E}_i} = \max\left(1, \text{Per}_{95}\left(V_{\mathrm{E}_i}(\cdot)\right) - \text{Per}_5\left(V_{\mathrm{E}_i}(\cdot)\right)\right) \tag{15}$$

4 Experimentation

4.1 *Experimental Information*

4.1.1 *Benchmarks*

Atari 100k [26] serves as a comprehensive benchmark comprising 26 Atari games, providing a diverse range of challenges to assess various algorithms' performance. In this benchmark, agents train for 100k steps, equivalent to 400k frames (considering a frameskip of 4). Each block of 100k steps approximately aligns with 2 hours of real-time gameplay per environment.

4.1.2 *Algorithms*

The algorithms used are namely (i) AlphaZero (AZ) [42]; (ii) A2C (Advantage Actor-Critic) [34]; (iii) A2C with random agent as a guide, noted as A2C-Rand (similar to SAC); (iv) A2C with behavior cloning as an guide, noted as A2C-BC; (v) A2C agent with AlphaZero as an guide, noted as A2C-AZ or A2C-AZ* where A2C-AZ uses a fixed hyperparameter λ^A for all games and A2C-AZ* uses a fine-tuned λ^A for each game.

Given the novelty of our approach, we conducted experiments with a single guide ($|E| = 1$) and uniform weights assigned across all states ($\forall s, \lambda^C_{E_i}(s) = \lambda^C_{E_i}$ and $\forall \lambda^A_{E_i}(s) = \lambda^A_{E_i}$). Furthermore, in our study, we used A2C as the reinforcement learning algorithm and AlphaZero as the guide. However, our implementation is not limited to these specific algorithms. Various other RL and search algorithms could have been explored as alternative options for experimentation.

4.1.3 *Actor/Critic*

All the algorithms use a critic and an actor network, composed of a two-layered MLP network of 512 hidden units. As defined in the introduction, the critic loss sub $\mathcal{L}^{C,Sub}_\theta()$ uses a cross-entropy based on a discrete representation [38, 22] and the actor loss sub $\mathcal{L}^{A,Sub}_\theta()$ uses reinforce with an advantage baseline to reduce the variance.

The distance function $\mathcal{F}^A(,)$ used for the actor is a KL-divergence function and the distance function $\mathcal{F}^C(,)$ used for the critic is a cross-entropy. The weight of the guide penalty λ^A is fixed at 0.08 for behavior cloning and at 0.03 for random (both where chosen between $[0.03, 0.08, 0.3]$), and unless otherwise stated, set at 0.7 for MCTS. For A2C-AZ, the weight for the critic is fixed at 0.05. For enhancing stability, the guide value target $\bar{V}_{E_i}()$ and the value target $\bar{V}_\theta()$ use the N-step bootstrapped λ-returns.

4.1.4 *Monte Carlo Tree Search*

A2C-AZ utilizes the actor and critic networks of the A2C agent, ensuring that it does not deviate significantly from it. Our implementation of MCTS in A2C-AZ and AlphaZero is built on previous famous MCTS implementations [41, 42, 38, 50]. It uses a search budget of 50, PUCT in the selection and Dirichlet noise distribution to help explore. However, three differences should be noted (i) we do not use Re-Analyse; (ii) we do not use prioritized experience replay [37]; (iii) we do not use the search algorithm in the test phase. These differences were made to effectively compare the different algorithms.

4.1.5 *Metrics*

We report the raw performance on each game, the human normalized score, as well as the Interquartile Mean (IQM) and Optimality Gap. The IQM and the Optimality Gap are metrics recommended for Atari100K benchmarks [1] where the authors recommend using IQM instead of the Median, and Optimality Gap instead of Mean, as both methods are more robust. IQM calculates the average over the data, removing the top and bottom 25%. Optimality Gap computes the amount by which the algorithm fails to meet a minimum score. A higher score is better for the IQM and a lower score is better for the optimality gap.

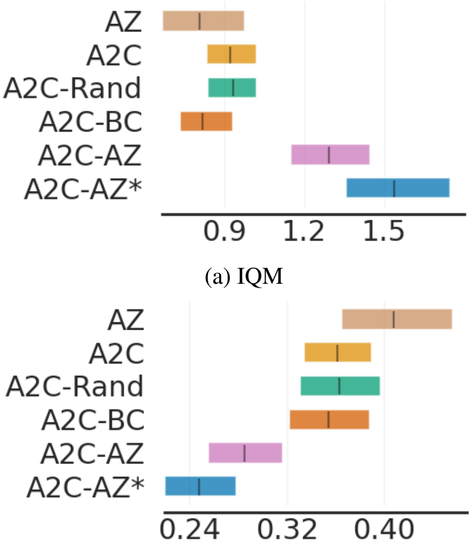
4.1.6 *Other*

Each agent uses a single environment instance with a single *NVIDIA V100 GPU*. Each algorithm is run using 5 seeds, we evaluated performance every 10k training step with 10 independent run of the game. To mitigate training expenses, we conducted our experiments by using a world model. We employed the fixed-trained weights from the Dreamer algorithm [22], a state-of-the-art model-based technique trained over $50,000$k steps. The world model is used to compute the N-step bootstrapped λ-returns for A2C algorithms and facilitating MCTS in A2C-AZ and AlphaZero. Additionally, to enhance cost-effectiveness and stability, we restricted our experimentation to 21 out of 26 games, excluding those where the world model demonstrated poorer performance in terms of mean human-normalized scores. Additionally information and experiments are available in the supplementary material [2].

4.2 *Experiments*

Initially, we will examine the overall impact of the various algorithms and guides. Subsequently, we narrow our focus on MCTS as a guide, analyzing the experiments in greater detail. Finally, we analyze the impact of the guide's weight, by testing several weights and trying to observe the impact when the guide is called less often.

4.2.1 *Overall analysis*



(a) IQM

(b) Optimality Gap

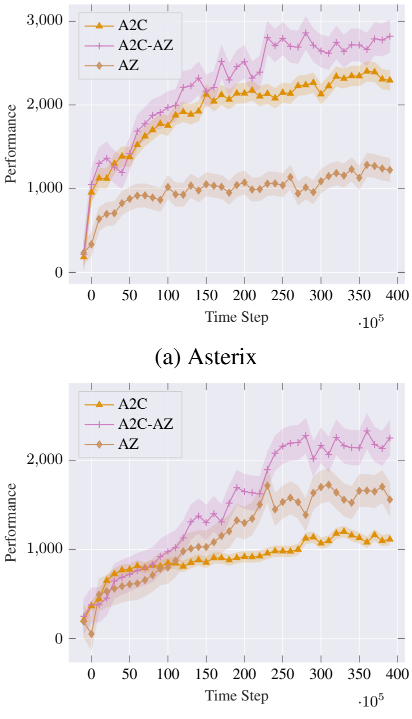
Figure 1: Aggregate performance. Shaded area shows 95% stratified bootstrap confidence interval. The x-axis represents the human normalized score.

Figure 1 analyzes the overall performance using the IQM and Optimality Gap metrics for all the different guides considered. We notice a significant enhancement in performance when utilizing AZ as a guide across both metrics. Specifically, A2C-AZ with a fixed weight surpasses A2C by over 0.4 on the IQM and 0.1 on the Optimality Gap. Fine-tuning the weight further improves performance, with IQM increasing from 1.3 to 1.5 and the Optimality Gap decreasing from 0.28 to 0.24.

4.2.2 *MCTS as a guide*

In Figure 3, we observe the percentage improvement of A2C-AZ*/A2C-AZ/AlphaZero over A2C. Furthermore, Figure 2 displays a series of learning curves for A2C-AZ, A2C, and AlphaZero, forming the foundation for our subsequent analysis.

We begin our analysis by comparing the performance of AZ and A2C agents independently. Each figure represents distinct scenarios: one where AZ outperforms A2C (Figure 2.a) and another where A2C outperforms AZ (Figure 2.b). These figures provide an initial glimpse into the broader performance trends.



(a) Asterix

(b) Assault

Figure 2: Learning curves on 2 different game of Atari100k benchmarks with 3 algorithms presented. The shaded area shows 95% confidence interval.

Overall, A2C demonstrates superior performance in 12 games, while AZ surpasses A2C in 8 games, with 1 game showing equivalent performance. Despite the general advantage of A2C, it is essential to highlight instances where A2C falls short, indicating the potential benefits of integrating AZ as a guide.

When considering the incorporation of AZ as a guide, several critical questions arise: can this integration elevate the agent's performance to at least match the best of the two individual agents? Is it possible to create an agent superior to the best individual performer, or might utilizing the guide lead to a weakened agent?

Our analysis across games shows that, compared to A2C, 12/17 games exhibit performance improvements, 4/2 show equivalent performance, and 5/2 show lower performance when using the combined approaches A2C-AZ and A2C-AZ*. Interestingly, in the subset of 8 games where AZ outperformed A2C in isolation, integrating AZ as a guide resulted in superior performance in 6/7 of those instances.

Although not visible in the figure, but observable in supplementary material. Our observations indicate that the combined algorithm outperforms both individual algorithms in 9/11 instances, achieves the performance of the better of the two methods in 7/9 instances, is lower than the best but bounded by the two algorithms in 4/1 instances, and shows lower performance than both in only 1/0 instance.

4.2.3 *Weight of the guide*

In Figure 4, we explore the impact of the weight parameter, λ^A, on performance. We compare several fixed values of λ^A ranging from 0.1 to 0.7, alongside the optimal weight selected for each game. Each variant of A2C-AZ is denoted by A2C-AZ-X, where X represents the specific weight used. For instance, A2C-AZ-0.3 employs AZ as a guide with a weight of 0.3.

Upon examination, we find that the optimal fixed weight is 0.7, resulting in an IQM of 1.29 and an Optimality Gap of 0.28. Notably, reducing the weight significantly leads to performance outcomes closely resembling those of A2C alone.

4.2.4 *Cost of using a guide*

Throughout our previous experiments, we have observed a significant advantage in using AZ as a guide. However, as indicated in Table 1, there is an overhead cost associated with employing AZ.

In practice, several MCTS methods can significantly reduce this cost, such as batch MCTS [8] and various parallelization techniques (leaf [9], root [11], and tree [10]). Additionally, many implementations utilize extensive computational resources to better distribute the workload. For instance, the basic version of AlphaGo uses 40 search threads, 48 CPUs, and 8 GPUs.

In the following experiment, we demonstrate another way to reduce the cost of incorporating a guide, which is quite natural in our context. Currently, the guide is executed at every iteration. However, our goal is to avoid deploying the guide in every situation. We aim to activate the guide only when necessary—specifically, in scenarios where our RL agent faces challenges or when the guide is known to excel.

Figure 5 shows the impact of using the guide less frequently. Instead of employing the guide at each iteration, we use it at every N iterations. We introduce the notation A2C-AZ-X, where X indicates how often the guide is called. For example, A2C-AZ-3 uses MCTS every three steps. Additionally, Table 2 presents the overhead cost of using AZ as a guide according to different values of N.

Table 2: Runtime on Atari100k Benchmarks.

Algorithm	A2C	A2C-AZ-1	A2C-AZ-2	A2C-AZ-3
Time	4:00	18:00	11:00	8:30

As observed, when running AZ at every iteration, the algorithm will achieve the best performance but take 18 hours to complete instead of the 4 hours required by A2C alone. However, by reducing the frequency to every two iterations, the runtime is reduced by half while still achieving a performance close to the best.

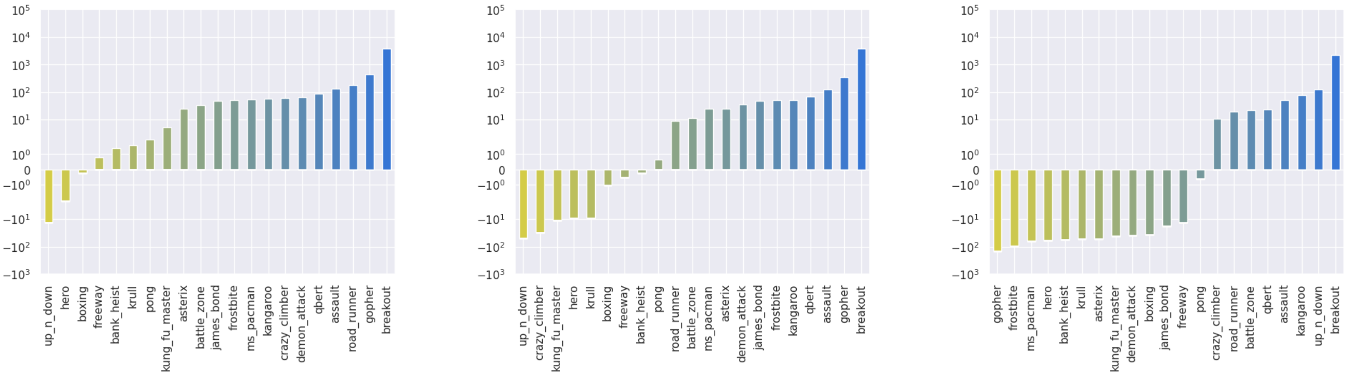
5 Related Work

5.1 *Offline Reinforcement Learning*

Our work is strongly linked to the field of Offline RL as inspired by one of the key methods in the field. In our case, we have chosen to use regularization methods to align the policy and the value function with the guide. Yet, within the realm of regularization methods, there exist many methods, these include penalties applied within the reward function [49] or regularization penalties applied after its computation of the loss [30, 29]. Additionally, the calculation of the penalty can be accomplished by using various functions including KL divergence, Maximum Mean Discrepancy [49], or even Fisher information [29].

5.2 *Monte Carlo Tree Search*

MCTS [7, 45] stands as a state-of-the-art algorithm that has significantly enhanced performance and tackled complex problems. In recent years, MCTS has been integrated with neural networks to boost

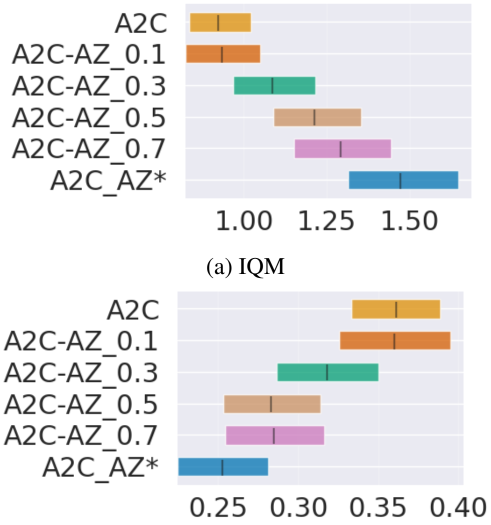


(a) X is A2C-AZ*, Y is A2C agent

(b) X is A2C-AZ, Y is A2C agent

(c) X is AlphaZero, Y is A2C agent

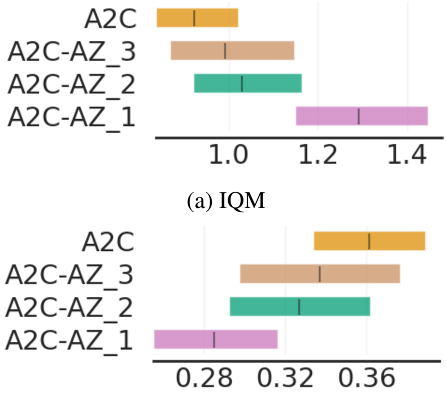
Figure 3: Percentage improvement of algorithm X compared to algorithm Y on Atari100k Benchmarks. Improvement is measured as a percentage of mean human-normalized return.



(a) IQM

(b) Optimality Gap

Figure 4: Aggregate performance metrics according to the weight. The shaded area shows 95% stratified bootstrap confidence interval. The x-axis represents the human normalized score.



(a) IQM

(b) Optimality Gap

Figure 5: Aggregate performance according to the number of calls made to the guide. The shaded area shows 95% stratified bootstrap confidence interval. The x-axis represents the human normalised score.

its performance [41, 43, 42], however, in most scenarios, neural networks are employed to predict the outcomes generated by MCTS.

To our knowledge, no prior work has attempted to utilize MCTS as a guide while retaining the RL module. The most closely related study we encountered is [27], where the authors incorporated A3C with K workers, among which MCTS was one, resulting in notable performance enhancements. However, unlike their method, our approach utilizes MCTS as a guide in every state. Furthermore, we consider not only the policy distribution but also the value returned by MCTS to enrich learning. Additionally, we introduce an adaptive weight for actor learning and conduct a more comprehensive set of experiments.

6 Conclusion

In this paper, we investigate the influence of leveraging online algorithms as a guide to enhance the learning process of RL algorithms. Inspired by techniques in Offline RL, we adapt these methodologies to the context of using an online algorithm, as a guide. Our approach involves regularizing the loss functions for both the actor and the critic to incorporate the information provided by the guide effectively.

Among the array of online algorithms explored from existing literature, our focus lies on Monte Carlo Tree Search (MCTS), a cutting-edge planning algorithm renowned for its convergence capabilities in both single-player and two-player scenarios. Notably, employing MCTS as a guide yields superior results compared to employing either of the two methods in isolation. Furthermore, fine-tuning just one hyperparameter can extend performance gains. Additionally, reducing the frequency of the guide calls can mitigate the cost associated with it while still resulting in enhanced performance.

In the future, there exist promising avenues for further exploration. Experimenting with diverse hyperparameters, such as alternative distance functions, different search algorithms or different reinforcement learning algorithms, could illuminate nuanced insights. Additionally, exploring the integration of multiple guides could broaden the range of possibilities, incorporating different perspectives from various guides. Finally, investigating the utilization of an automatic weight, potentially based on neural networks, could provide a more adaptive, efficient, and general approach.

References

[1] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[2] J. Arjonilla, A. Saffidine, and T. Cazenave. Enhancing reinforcement learning through guided search, 2024. URL https://arxiv.org/abs/2408.10113.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[4] A. Bakhtin, D. J. Wu, A. Lerer, J. Gray, A. P. Jacob, G. Farina, A. H. Miller, and N. Brown. Mastering the game of no-press diplomacy via human-regularized reinforcement learning and planning. *arXiv preprint arXiv:2210.05492*, 2022.

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

[6] M. G. Bellemare, W. Dabney, and M. Rowland. *Distributional reinforcement learning*. MIT Press, 2023.

[7] C. Browne, E. J. Powley, D. Whitehouse, S. M. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. P. Liebana, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 2012.

[8] T. Cazenave. Batch Monte Carlo Tree Search. *ArXiv*, abs/2104.04278, 2021.

[9] T. Cazenave and N. Jouandeau. On the parallelization of uct. In *Computer games workshop*, 2007.

[10] T. Cazenave and N. Jouandeau. A parallel monte-carlo tree search algorithm. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 72–80. Springer, 2008.

[11] G. M. B. Chaslot, M. H. Winands, and H. J. van Den Herik. Parallel monte-carlo tree search. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*, pages 60–71. Springer, 2008.

[12] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[13] Q. Cohen-Solal and T. Cazenave. Minimax strikes back. *arXiv preprint arXiv:2012.10700*, 2020.

[14] P. I. Cowling, E. J. Powley, and D. Whitehouse. Information Set Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:120–143, 2012.

[15] J. Farebrother, J. Orbay, Q. Vuong, A. A. Taïga, Y. Chebotar, T. Xiao, A. Irpan, S. Levine, P. S. Castro, A. Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint arXiv:2403.03950*, 2024.

[16] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.

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

[18] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[19] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[20] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *CoRR*, abs/1912.01603, 2019. URL http://arxiv.org/abs/1912.01603.

[21] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

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

[23] Z. Huang, J. Wu, and C. Lv. Efficient deep reinforcement learning with imitative expert priors for autonomous driving. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[24] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2): 1–35, 2017.

[25] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[26] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[27] B. Kartal, P. Hernandez-Leal, and M. E. Taylor. Action guidance with mcts for deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 15, pages 153–159, 2019.

[28] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(75)90019-3. URL https://www.sciencedirect.com/science/article/pii/0004370275900193.

[29] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pages 5774–5783. PMLR, 2021.

[30] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[32] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[33] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak. Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search. In *AAAI*, 2010.

[34] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[35] T. W. Neller and M. Lanctot. An introduction to counterfactual regret minimization. In *Proceedings of model AI assignments, the fourth symposium on educational advances in artificial intelligence (EAAI-2013)*, volume 11, 2013.

[36] R. F. Prudencio, M. R. Maximo, and E. L. Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[40] L. Shi, R. Dadashi, Y. Chi, P. S. Castro, and M. Geist. Offline reinforcement learning with on-policy q-function regularization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 455–471. Springer, 2023.

[41] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

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

[43] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362:1140 – 1144, 2018.

[44] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, Cambridge, MA, USA, 2018. ISBN 0262039249.

[45] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

[46] Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, and L. Zitnick. Elf opengo: An analysis and open reimplementation of alphazero. In *International conference on machine learning*, pages 6244–6253.

PMLR, 2019.

[47] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[48] J. Wu, Z. Huang, Z. Hu, and C. Lv. Toward human-in-the-loop ai: Enhancing deep reinforcement learning via real-time human guidance for autonomous driving. *Engineering*, 21:75–91, 2023.

[49] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, abs/1911.11361, 2019. URL https://api.semanticscholar.org/CorpusID:208291277.

[50] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

A Implementation Details

A.1 Discrete representation for the critic

The critic's loss function, denoted as $\mathcal{L}_{\theta}^{C,Sub}(s)$, is formulated to minimize the disparity between the value target $\bar{V}_{\theta}(s)$ and the predicted value $V_{\theta}(s)$ at a specific state s. Commonly, the disparity is computed with the mean squared error or the cross-entropy over a discrete representation.

Previous studies have emphasized the benefits of employing cross-entropy over a discrete representation in reinforcement learning [5, 38, 22, 6, 15]. This method involves the critic to learn a discrete weight distribution $p_{\theta} = \{p_1, ..., p_B\} \in \mathbb{R}^B$ instead of learning the mean of the distribution. A function $y()$ that converts a target value into a corresponding weight distribution of size B. This leads to the following sub-loss for the critic:

$$\mathcal{L}_{\theta}^{C,Sub}(s) = y(\bar{V}_{\theta}(s))^T \log p_{\theta} \tag{16}$$

More specifically, transforming (function y) the reward/target into a discrete representation is done function by a method called two-hot encoding. The two-hot encoding is a generalization of the one-hot encoding where all elements are 0 except for the two entries closest to y at positions m and m + 1. These two entries sum up to 1, with more weight given to the entry that is closer to y:

$$y(x) = twohot(x)_i = \begin{cases} |b_{m+1} - x|/|b_{m+1} - bm| & \text{if } i = m \\ |b_m - x|/|b_{m+1} - b_m| & \text{if } i = m+1 \\ 0 & \text{else} \end{cases}$$

Importantly, two-hot encoding can predict any continuous value in the interval because its expected bucket value can fall between the buckets.

B Monte Carlo Tree Search-Detailed

Below, we provide a comprehensive overview of the Monte Carlo Tree Search (MCTS) algorithm, drawing from previous research on MCTS [43, 38, 50]. Notably, in Schrittwieser et al. [38], it was demonstrated that a budget of 50 is adequate for resolving the Atari100K benchmark, hence informing our decision in this regard.

Monte Carlo Tree Search (MCTS) [7] is the state-of-the-art in the perfect information game. MCTS converges asymptotically to the optimal policy in single-agent domains and to the minimax value function in zero-sum games. Starting from the AlphaGo [41, 43, 42], MCTS has been combined with an offline neural network to enhance performance.

$MCTS(s, budget)$ is an online tree search algorithm that runs at s for a budget of $budget$ and works as follows (i) **selection** — selects a path of nodes until a leaf node; (ii) **expansion** — expands the tree by adding a new child node; (iii) **backpropagation** — backpropagates the result obtained through the nodes chosen during the selection phase; (iv) repeats step (i) to (iii) until the budget $budget$ is finished; (v) returns the distribution of actions π_{MCTS} that has been visited, and the value V_{MCTS} obtained when running MCTS.

Every node of the search tree is associated with a state s and each node stores the statistics estimating the value of the node $V(s)$. For every action a at s, there exists an associated edge represented as (s, a). These edges stores a set of statistics $\{$ $\mathrm{N}_t(s, a)$, $\mathrm{Q}^t(s, a)$, $\pi(a|s)$, $r(s, a)$, $\mathcal{T}(\cdot|s, a)$ $\}$, respectively representing visit counts N, mean Q-Value observed, policy π, reward r, and state transition $\mathcal{T}$.

B.1 Selection

In the selection, a simulation begins at the internal root state, denoted as s^0, and progresses until it reaches a leaf node represented as s^l. Throughout this selection, actions a^k for $k = 1, \cdots, l$ are selected using the PUCT formula. This formula strikes a balance between exploration and exploitation, guiding the choice of actions during the simulation.

$$\mathrm{Q}(s,a) + \pi(a|s)\frac{\sqrt{\mathrm{N}(s)}}{1+\mathrm{N}(s,a)}\left(c_1 + log(\frac{\mathrm{N}(s)+c_2+1}{c_2})\right)$$

The best action is the one that maximizes where $\mathrm{N}(s)$ represents the number of times that the state s has been visited, c_1 and c_2 are variables that help to control the exploration.

B.2 Expansion

Following the selection phase, at step l of the simulation, the environment dynamics function $f(s^{l-1}, a^l)$ computes the reward $r(s^{l-1}, a^l)$ and the ensuing state s^l. The policy network generates $\pi_\theta(\cdot|s^l)$ and the value network estimates $V_\theta(s^l)$. For every action a within s^l, a corresponding edge (s^l, a) is established. This edge is initialized with values: $\mathrm{N}(s^l, a) = 0$, $\mathrm{Q}(s^l, a) = 0$, and $\pi(a|s^l) = \pi_\theta(a|s^l)$.

Initially, the default Q for unvisited nodes is set to 0, indicating the worst possible state. To enhance the Q estimation for unvisited nodes, a mean Q mechanism is implemented during each simulation for tree nodes. This approach, similar to the one employed in Elf OpenGo [46, 50], provides a more accurate estimation of Q for unvisited nodes. This evaluation is performed iteratively for $k = 0, \cdots, l$.

$$\widehat{\mathrm{Q}s^0}() = 0$$

$$\widehat{\mathrm{Q}s}() = \frac{\mathrm{Q}\widehat{s^{parent}}(+)\sum_b \mathbb{1}_{\mathrm{N}(s,a)>0}\mathrm{Q}(s,b)}{1+\mathbb{1}_{\mathrm{N}(s,a)>0}}$$

$$\mathrm{Q}(s,a) = \begin{cases}\mathrm{Q}(s,a) & \mathrm{N}(s,a) > 0 \\ \widehat{\mathrm{Q}s}() & \mathrm{N}(s,a) = 0\end{cases}$$

where $\widehat{\mathrm{Q}s}()$ is the estimated Q value for unvisited nodes.

In addition, when the root node is expanded, a Dirichlet noise to the policy prior is added. This technique is used for improving the exploration. $\mathcal{N}_D(\epsilon)$ is the Dirichlet noise distribution, ρ, ϵ is set to 0.25 and 0.3 respectively.

$$\pi(a|s) = (1-\rho)\pi(a|s) + \mathcal{N}_D(\epsilon)$$

B.3 Backup

For every step from $k = \{l-1, \ldots, 0\}$, we update the statistics associated with each edge (s^k, a^k) in the simulation path by using the boostrapped λ-returns.

$$\mathrm{Q}(s^{k-1}, a^k) = \frac{\mathrm{N}(s^{k-1}, a^k)\mathrm{Q}(s^{k-1}, a^k) + V^{\lambda, l-k}(s^k)}{\mathrm{N}(s^{k-1}, a^k)+1}$$

$$\mathrm{N}(s^{k-1}, a^k) = \mathrm{N}(s^{k-1}, a^k) + 1$$

Ensuring the Q-Value falls within the range of [0,1] is crucial for employing the PUCT formula, to achieve this, we normalize the Q value by referencing the minimum and maximum values observed in the search tree up to that particular point.

$$\bar{Q}(s^k, a^k) = \frac{\mathrm{Q}(s^k, a^k) - \underset{s,a\in Tree}{min}\mathrm{Q}(s,a)}{max(\underset{s,a\in Tree}{max}\mathrm{Q}(s,a) - \underset{s,a\in Tree}{min}\mathrm{Q}(s,a), \epsilon)}$$

, where ϵ, the threshold to give a smooth range of the min-max bound.

B.4 Additional Information

After the budget *budget* is finished, we obtain the average value V_{AZ} and visit count distributions of the root node. To obtain the policy distribution π_{AZ}, we use a temperature parameter of the visit count; *i.e.*

$$\pi_{AZ}(s,a) = \frac{\mathrm{N}(s,a)^{\frac{1}{\mathrm{T}}}}{\sum_b \mathrm{N}(s,b)^{\frac{1}{\mathrm{T}}}}$$

During the training process, we decay the temperature twice, at 50% of training progress to 0.5, and at the 75% of training progress to 0.25.

C Model-Based

To reduce the cost of training, our experiments were carried out using a world model trained. This means that the game (dynamics, reward, graphical representation, etc.) is approximated by a representation of the world and to obtain one of the game's pieces of information, a call is made to this representation. Using a world model that has already been trained allows us to reduce calculation time. Thanks to this, we already have an abstraction of the worlds and the dynamics, allowing us to concentrate our learning on the critic and the actor-network.

In our case, we used the world model of the Dreamer V3 algorithm [22]. Dreamer is a state-of-the-art algorithm in a model-based setting and constitutes the first agent that achieves human-level performance on the Atari benchmark tasks by learning behaviors inside a separately trained world model. Dreamer takes a world state s^t as an input and returns an abstracted world state $\hat{s}_t \sim s^t$, an expected reward $\hat{r}_t \sim r^t$ and a continue flag c_t, which indicate if the game if finished or not. The world model uses a Recurrent State-Space Model (RSSM) which predicts future information ($\hat{s}_{t+1}$, $\hat{r}_{t+1}$ and c_{t+1}) when given a^t to the current abstract world $\hat{s}_t$. The world model is used to compute the N-step bootstrapped λ-returns for A2C algorithms and facilitating MCTS in A2C-AZ and AlphaZero.

In our implementation, we use the weight of an already trained Dreamer algorithm during $50,000$k training steps. To be precise, the world model is never modified (fixed), so our work is not related to MuZero/EfficientZero/Dreamer algorithms, which train the world model and at the same time learn the policy/critic, but much closer to algorithms such as AlphaZero/A2C that use a fixed world and learn the policy/critic.

D Neural Network

The neural networks of the world model are the same as used in the paper in DreamerV3 [22] in which we used the Model size S. The encoder begins with stride 2 convolutions neural networks (CNN) [31] that progressively increase the depth of the representation until reaching a resolution of 4×4. Subsequently, the data is flattened for further processing. Conversely, the decoder initiates with a dense layer, which is followed by reshaping the data into a $4 \times 4 \times 32$ format. It then effectively reverses the architecture employed in the encoder to reconstruct the original data. The dynamics component is realized through a Recurrent State Space Model (RSSM) [19] utilizing vectors of categorical representations. This implementation comprises a Gated Recurrent Unit (GRU) [12] combined with MLP layers. The reward, critic, actor and continue predictors are also MLPs. Each MLP network is composed of 2 linear network of 512 hidden units.

As clarified in the preceding section, our approach does not involve training the world model as in Dreamer. Instead, we utilize a pre-trained and fixed world model. Consequently, the only neural networks trained in our setup are the critic and actor networks which each are a MLP with 2 linear layers of 512 hidden units.

E Notation and Hyper-parameters

All notations and hyperparameters utilized throughout the paper are summarized in Table 3. The hyperparameters in the World-Model/Actor-Critic section are derived from DreamerV3 [22], while those in the Search section are adapted from EfficientZero [50].

F Additional Experiment

In Tables 4, 5 and 6, we observe the score obtained at the end of the training for the different algorithms tested on the Atari100k benchmarks. In Figure 6, we observe all the learning curves of all the algorithms on the Atari100k benchmarks. In Figure 8, we observe the percentage improvement of the different algorithms. In Figure 7, we observe the score distribution of the different algorithms, following methodology from [1].

G Questions and Answers

A few questions were asked during the review process, which could also be of interest to the reader.

- **Q1:** Links with Offline RL?
- **A1:** There is no direct link with offline RL in our work, but it inspired our approach. Therefore, we discussed it in the Related Work section to highlight alternative methods of penalization for deviations from the expert.
- **Q2:** Extending the Proposed Method to Value-Based Approaches like DQN?
- **A2:** For value-based approaches like DQN, which lack an actor loss function, two solutions are: (i) remove the actor loss function entirely and retain only the critic loss, which still benefits from the expert's guidance, or (ii) retain the actor loss function so that it can still be used by AlphaZero, but only $\mathcal{L}_\theta^{A,Sub}(s)$ of equation 10 would remain.
- **Q3:** Useful/Useless Information from Expert:?
- **A3:** Utilizing a search algorithm as an expert generally avoids harmful guidance. However, as noted, lower performance was observed in one game, though this issue was resolved in A2C_AZ*. In our case, the hyperparameters in Equations 12 and 13 were determined through experimentation with various values. Moving forward, it would be valuable to explore neural network-based weighting methods, which could dynamically adjust the weights based on both the expert's input and the current state. This approach could potentially minimize costs while maximizing the beneficial impact of expert guidance.

Table 3: Hyper parameters used.

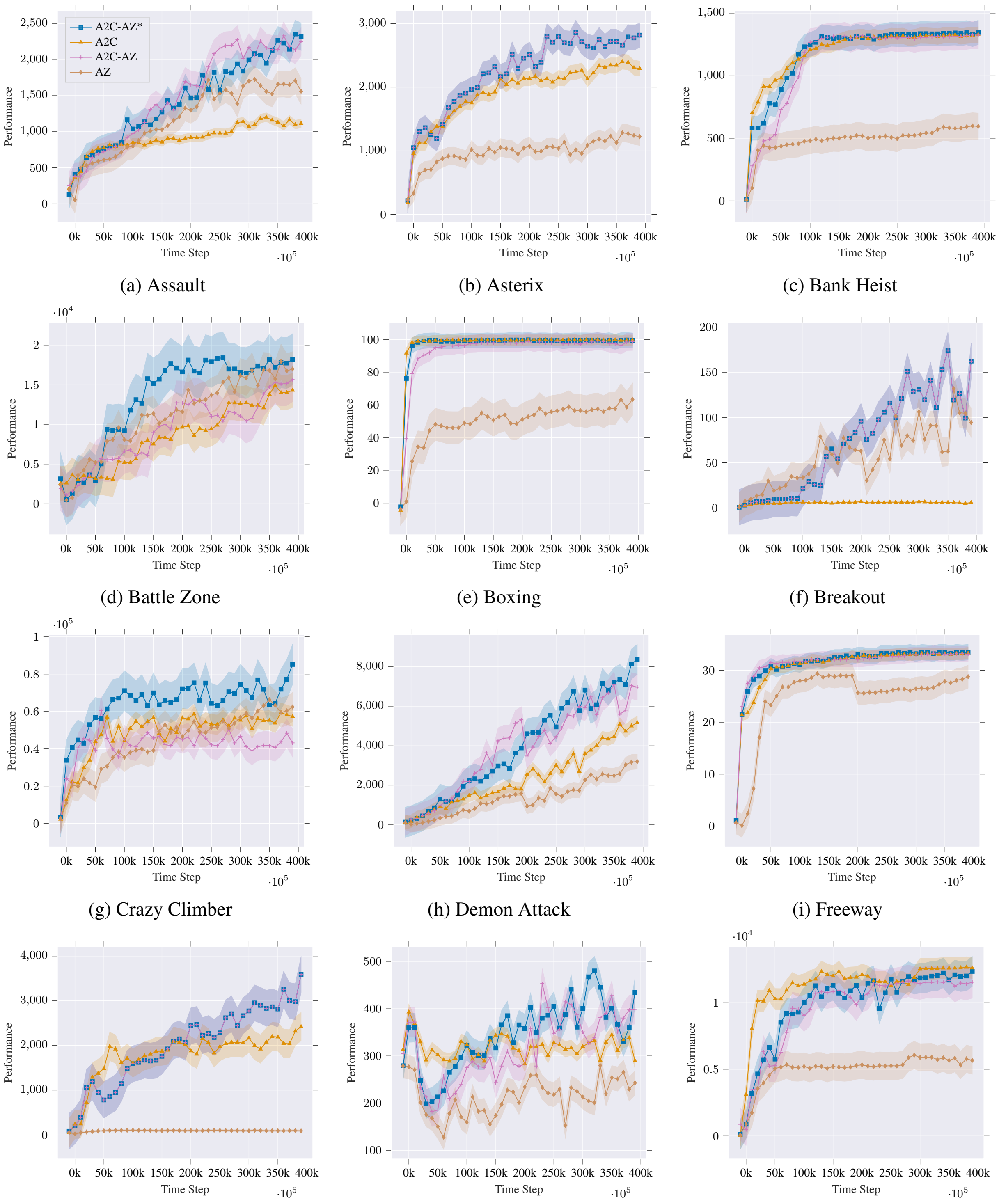
Name	**Notation**	**Value**
General		
Training Step	_	$100,000$
World Model		
Batch size	_	16
Batch length	_	16
Imagination horizon	_	15
Number of latents	_	32
Classes per latent	_	32
Model Size	_	S
Actor Critic		
Discount	λ	0.95
Critic EMA decay	_	0.98
Critic EMA regularizer	_	1.0
Normalization Ter	S	$Per(R, 95) - Per(R, 5)$
Learning rate	_	$3 \cdot 10^{-5}$
Adam epsilon	_	10^{-5}
Gradient clipping	_	100
Guide Actor penalty	λ^{A}	$[0.1, 0.3, 0.5, 0.7]$,
Max Guide Penalty	λ^{Max}	10.0
Tau	τ	5.0
Guide Critic penalty	λ^{C}	0.05
Distance function Actor	$\mathcal{F}^{A}(,)$	KL-divergence
Distance function Critic	$\mathcal{F}^{C}(,)$	Cross-Entropy
Loss function Actor	$\mathcal{L}_{\theta}^{A,Sub}()$	Reinforce/Cross-Entropy for RL/MCTS
Loss function Critic	$\mathcal{L}_{\theta}^{C,Sub}()$	Cross-Entropy
Search		
Exploration constant	c_1	1.25
Exploration constant 2	c_2	19652
Search budget	$budget$	50
Dirichlet Noise	ϵ	0.3
Dirichlet Noise Proba	ρ	0.25
Temperature	T	$[1.0, 0.5, 0.25]$

Table 4: Performance obtained on Atari100K benchmarks.

Game	Random	Human	A2C	AZ	A2C-Rand	A2C-_BC	A2C-AZ	A2C-AZ*
Assault	222.4	742.0	1113.6	1559.7	2138.62	1766.1	2249.6	**2249.6**
Asterix	210.0	8503.3	2294.5	1222.8	2225.0	1924.0	**2819.2**	**2819.2**
Bank Heist	14.2	753.1	1328.0	595.1	1285.0	1132.6	1324.4	**1346.0**
BattleZone	2360.0	37187.5	14300.0	16985.7	19660.0	**22660.0**	15642.8	15642.8
Boxing	0.1	12.1	**99.6**	63.4	**99.6**	99.2	98.6	99.2
Breakout	1.7	30.5	5.8	94.4	6.16	8.7	**162.4**	**162.4**
Crazy Climber	10780.5	35829.4	52137.0	62392.8	38264.0	26816.0	43157.1	**75465.7**
Demon Attack	152.1	1971.0	5168.4	3197.7	**11517.5**	5822.4	6967.2	75465.7
Freeway	0.0	29.6	33.3	28.7	**33.6**	**33.6**	33.1	33.1
Frostbite	65.2	4334.7	2420.7	89.4	920.2	2225.2	**3588.1**	**3588.1**
Gopher	257.6	2412.5	289.6	242.8	356.4	287.6	398.5	**434.5**
Hero	1027.0	30826.4	12598.4	5669.0	11093.9	**14549.4**	11531.3	12343.7
Jamesbond	29.0	302.8	671.0	557.8	816.0	947.0	**981.4**	**981.4**
Kangaroo	52.0	3035.0	4084.0	**7288.5**	792.0	4084.0	6117.1	6117.1
Krull	1598.0	2665.5	10150.2	5593.5	9906.6	7699.4	9366.8	**10257.7**
Kung Fu Master	258.5	22736.3	35207.0	20454.2	34592.0	20590.0	31430.0	**37002.8**
Ms Pacman	307.3	6951.6	1656.5	784.4	**2904.0**	2208.6	1985.8	2402.5
Pong	-20.7	14.6	20.1	19.8	20.6	18.5	**20.4**	**20.9**
Qbert	163.9	13455.0	3780.2	4610.0	6569.5	5056.5	6268.9	**6844.6**
Road Runner	11.5	7845.0	7827.0	9341.4	9836.0	8016.0	8521.4	**12178.5**
Up N Down	533.4	11693.2	8622.3	**18857.8**	3922.8	4156.6	4688.7	7570.2
Mean (↗)	0.0	1.0	1.7	1.46	1.89	1.62	2.14	2.31
Median (↗)	0.0	1.0	1.63	1.48	1.87	1.45	2.18	2.23
IQM (↗)	0.0	1.0	0.92	0.81	0.93	0.82	1.29	1.47
Optimality Gap (↘)	1.0	0.0	0.36	0.40	0.36	0.35	0.28	0.25

Table 5: Performance obtained on Atari100K benchmarks. We denote A2C-AZ-X where x refers to the weight of the guide's penalty.

Game	Random	Human	A2C-AZ_0.1	A2C-AZ_0.3	A2C-AZ_0.5	A2C-AZ_0.7
Assault	222.4	742.0	1753.5	1786.4	2154.8	**2249.6**
Asterix	210.0	8503.3	2319.2	2275.0	2500.7	**2819.2**
Bank Heist	14.2	753.1	1342.2	**1346.0**	1303.7	1324.4
BattleZone	2360.0	37187.5	7457.1	10885.7	10328.5	**15642.8**
Boxing	0.1	12.1	98.2	**99.2**	97.6	98.6
Breakout	1.7	30.5	6.9	86.3	84.3	**162.4**
Crazy Climber	10780.5	35829.4	**75465.7**	29448.5	60411.4	43157.1
Demon Attack	152.1	1971.0	5578.0	**7361.7**	6947.2	6967.2
Freeway	0.0	29.6	**33.1**	33.0	**33.1**	33.1
Frostbite	65.2	4334.7	2364.4	1639.8	3113.1	**3588.1**
Gopher	257.6	2412.5	371.7	336.0	**434.5**	398.5
Hero	1027.0	30826.4	9145.7	10719.2	**12343.7**	11531.3
Jamesbond	29.0	302.8	546.4	855.7	793.5	**981.4**
Kangaroo	52.0	3035.0	3248.0	3611.4	3945.7	**6117.1**
Krull	1598.0	2665.5	9711.8	**10257.7**	8366.7	9366.8
Kung Fu Master	258.5	22736.3	25264.0	30807.1	**37002.8**	31430.0
Ms Pacman	307.3	6951.6	**2402.5**	2261.4	2143.7	1985.8
Pong	-20.7	14.6	20.3	**20.9**	20.3	20.4
Qbert	163.9	13455.0	5411.7	6021.7	**6844.6**	6268.9
Road Runner	11.5	7845.0	10070.0	11460.0	**12178.5**	8521.4
Up N Down	533.4	11693.2	**7205.0**	4708.7	7178.7	4688.7
Mean (↗)	0.0	1.0	1.78	1.95	1.95	2.14
Median (↗)	0.0	1.0	1.74	1.91	1.90	2.18
IQM (↗)	0.0	1.0	0.93	1.08	1.21	1.29
Optimality Gap (↘)	1.0	0.0	0.36	0.31	0.28	0.28



(a) Assault

(b) Asterix

(c) Bank Heist

(d) Battle Zone

(e) Boxing

(f) Breakout

(g) Crazy Climber

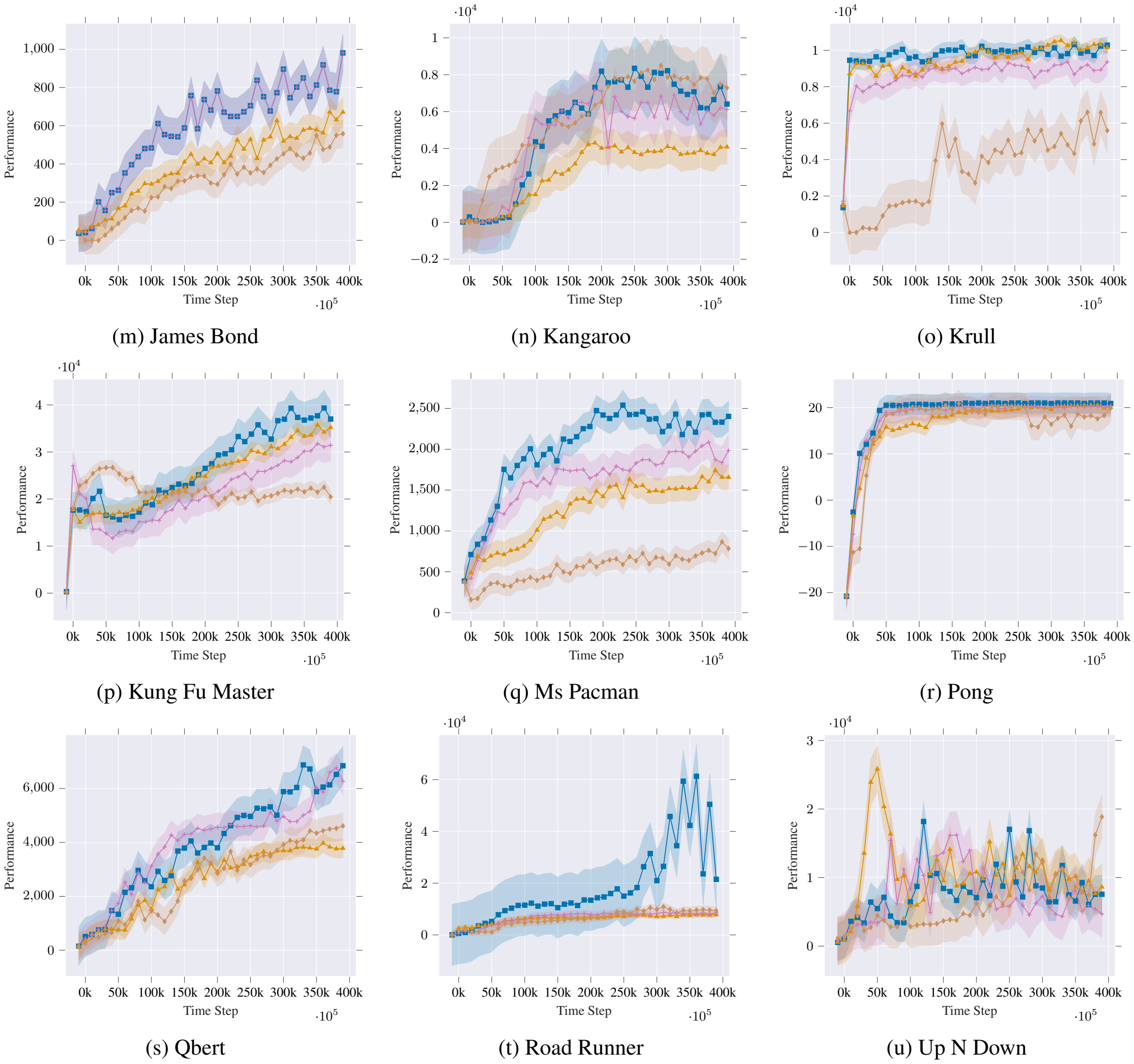
(h) Demon Attack

(i) Freeway

(j) Frostbite

(k) Gopher

(l) Hero



(m) James Bond

(n) Kangaroo

(o) Krull

(p) Kung Fu Master

(q) Ms Pacman

(r) Pong

(s) Qbert

(t) Road Runner

(u) Up N Down

Figure 6: Learning curves on the 26 game of Atari100k benchmarks with 5 algorithms presented. The shaded area shows 95% confidence interval (CI) over 5 seeds.

Table 6: Performance obtained on Atari100K benchmarks. We denote A2C-AZ-X where X indicates how often the guide is called.

Game	Random	Human	A2C-AZ_1	A2C-AZ_2	A2C-AZ_3
Assault	222.4	742.0	**2249.6**	1737.0	1819.3
Asterix	210.0	8503.3	**2819.3**	2431.4	2587.9
Bank Heist	14.2	753.1	**1324.4**	1271.4	1264.7
BattleZone	2360.0	37187.5	**15642.9**	13585.7	12814.3
Boxing	0.1	12.1	**99.2**	**99.2**	98.8
Breakout	1.7	30.5	**162.4**	74.3	47.3
Crazy Climber	10780.5	35829.4	**43157.1**	40022.9	32127.1
Demon Attack	152.1	1971.0	6967.2	**7476.4**	6697.2
Freeway	0.0	29.6	33.0	33.2	**33.1**
Frostbite	65.2	4334.7	**3588.1**	2722.0	3221.3
Gopher	257.6	2412.5	**398.5**	398.3	351.1
Hero	1027.0	30826.4	11531.3	9462.1	**11860.3**
Jamesbond	29.0	302.8	**981.4**	653.2	720.7
Kangaroo	52.0	3035.0	**6117.1**	2787.1	4262.9
Krull	1598.0	2665.5	9366.8	8392.0	**9847.1**
Kung Fu Master	258.5	31430.0	26915.7	31235.4	**32202.9**
Ms Pacman	307.3	6951.6	1985.8	**2312.6**	2176.1
Pong	-20.7	14.6	20.4	**21.0**	20.0
Qbert	163.9	13455.0	**6268.9**	5517.5	6202.1
Road Runner	11.5	7845.0	8521.4	**10144.3**	9850.0
Up N Down	533.4	11693.2	4688.7	4895.3	**6031.6**
Mean (↗)	0.0	1.0	2.14	1.80	1.83
Median (↗)	0.0	1.0	2.18	1.74	1.68
IQM (↗)	0.0	1.0	1.29	1.03	0.99
Optimality Gap (↘)	1.0	0.0	0.28	0.32	0.34

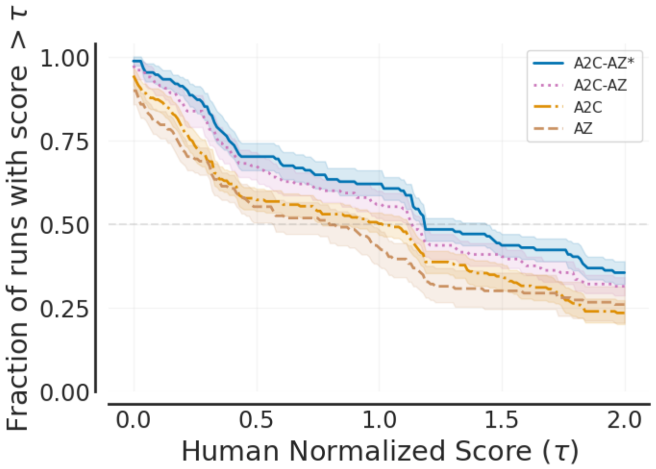
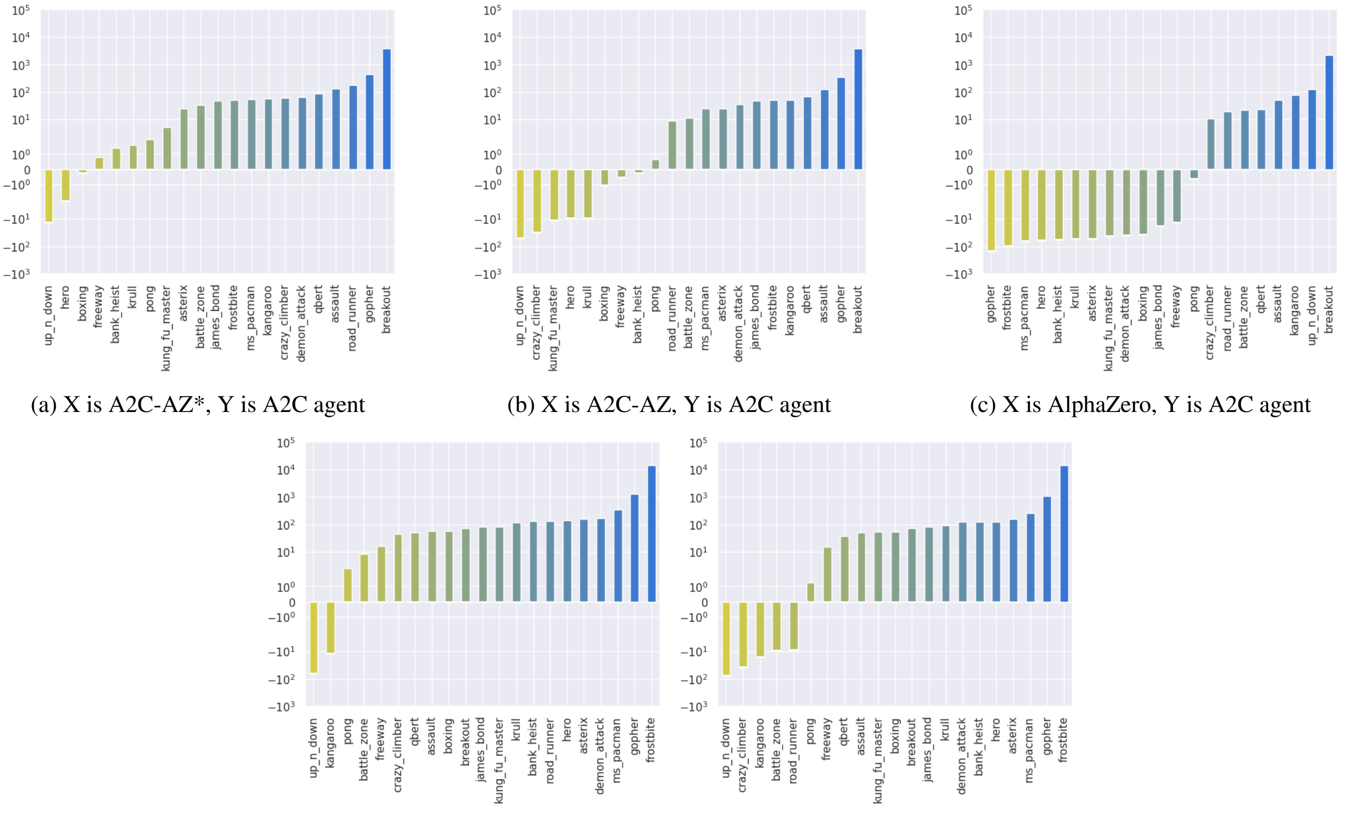


Figure 7: Human Normalised Score distribution on Atari100k Benchmarks. The shaded area shows 95% stratified bootstrap confidence interval (CI) over 5 seeds, following methodology. A higher score is better.



(a) X is A2C-AZ*, Y is A2C agent

(b) X is A2C-AZ, Y is A2C agent

(c) X is AlphaZero, Y is A2C agent

(d) X is A2C-AZ*, Y is AlphaZero

(e) X is A2C-AZ, Y is AlphaZero

Figure 8: Percentage improvement of algorithm X compared to algorithm Y on Atari100k Benchmarks. Improvement is measured as a percentage of mean human-normalized return.