Enhancing Reinforcement Learning Through Guided Search

Jérôme Arjonilla
Université Paris Dauphine - PSL
Paris, France
jerome.arjonilla@hotmail.fr

Abdallah Saffidine
University of New South Wales
Sydney, Australia
abdallah.saffidine@gmail.com

Tristan Cazenave
Université Paris Dauphine - PSL
Paris, France
tristan.cazenave@dauphine.psl.eu

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 the performance *i.e.*, whether it is possible to find an expert capable of contributing to performance improvement, and how to incorporate it into our RL agent. Our attention is particularly focused on Monte Carlo Tree Search (MCTS) as expert guidance. 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 an expert guiding 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.

KEYWORDS

Reinforcement Learning, Search algorithm, Monte Carlo Tree Search, Games

1 INTRODUCTION

Reinforcement Learning (RL) is a leading field in artificial intelligence, advancing our grasp of intelligent decision-making in complex environments [2, 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 [30, 35], 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 an expert into our RL agent's.

In our research, we begin by analyzing the various online algorithms available to serve as an expert. In the literature, there exist algorithms that already use experts to improve their performance. As an example, in Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO), Asynchronous Advantage Actor Critic (A3C) [17, 18, 32, 39], the RL agent is combined with an entropy term to facilitate exploration. This entropy, in another formulation, is a measure of the distance between the current policy and the policy of an expert, of which this expert happens to be a random agent.

In our work, we turn our attention to Monte Carlo Tree Search (MCTS) [7, 45] as an expert to guide the RL agent. MCTS is a search algorithm that obtains state-of-the-art performance in many games and converges towards equilibrium with one and two players. When MCTS is employed as an expert, a significantly performance improvement is observed. Analyzing the data, it becomes evident that, in the majority of cases, integrating MCTS as an expert leads to enhanced performance, and in instances where this is not the case, the algorithms achieve the best outcome between the two algorithms. The combination of an RL algorithm with MCTS as an expert leverages RL's inherent generalization and learning capabilities, while also benefiting from MCTS's optimal online capabilities. Additionally, we enhance our investigation by examining various hyperparameters, particularly focusing on the extent to which we integrate the expert's input. Moreover, by reducing the number of time the expert is used, we show the potential to mitigate the overhead associated with expert guidance while preserving performance enhancements.

In Section 2 we present the different formalism and notation used throughout the paper. In Section 3, we present the strengths and weaknesses of multiple online experts and explain how to integrate the expert into the RL agent. Particularly when using MCTS as an expert it is possible to guide the RL agent in several key notions: the actor and the critic components within the RL algorithm. In Section 4, we experiment with different experts on the Atari100k benchmark. In Section 5, we present the related work, and lastly, in Section 6, we summarize our work and future work.

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.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2024), Avalos, Milec, Müller, Wang, Yates (eds.), May 6-7, 2024, Online, https://ala2024.github.io/. 2024.

Addressing an exact Markov Decision Process (MDP) can pose significant computational demands. To alleviate this, employing a world model [19–21, 37] proves advantageous. A world model serves as an approximation of $\mathcal{M}$, encompassing approximations of state representations, dynamics, and rewards. Leveraging the world model for information retrieval not only accelerates computation compared to exact methods but also facilitates parallel processing of state batches, especially when computing intricate tools like N-step bootstrapped λ-returns or employing Monte Carlo Tree Search (MCTS). This parallel processing is often executed on GPUs rather than CPUs, thereby further enhancing computational efficiency.

2.2 Reinforcement Learning

Reinforcement learning confronts the problem of learning to control the MDP, where the agent aims to maximize the expected cumulative reward, *i.e.* the agent try to acquire a policy π, which is defined as a distribution over actions conditioned on states $\pi(a|s)$, that maximizes the long-term discounted cumulative reward defined as follow :

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

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, which 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 or state-action pairs :

$$V_\pi(s^t) = \mathop{\mathbb{E}}_{a^t\sim\pi(\cdot|s^t)}\left[\mathrm{Q}^\pi(s^t, a^t)\right] \tag{2}$$

$$\mathrm{Q}^\pi(s^t, a^t) = r^t + \gamma \mathop{\mathbb{E}}_{s^{t+1}\sim\mathcal{T}(\cdot|s^t,a^t)}\left[V_\pi(s^{t+1})\right] \tag{3}$$

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

$$\mathcal{L}_\theta^C = \mathop{\mathbb{E}}_{s\sim\mathrm{D}}\left[\mathcal{L}_\theta^{C,Sub}(s)\right] \tag{4}$$

Previous studies have emphasized the benefits of employing cross-entropy over a discrete representation in reinforcement learning [5, 6, 14, 21, 37]. This method involves the critic to learn a discrete weight distribution $p_\theta = p_1, ..., p_B \in R^B$ instead of learning the mean of the distribution, and a function $y()$ that convert a target value into another weight distribution of size B. This leads to the following sub-loss for the critic:

$$\mathcal{L}_\theta^{C,Sub}(s) = y(\bar{V}_\theta(s))^T \log p_\theta \tag{5}$$

The value target often corresponds to the Q-Value, yet, to enhance stability, an alternative approach involves using the N-step bootstrapped λ-returns [21, 44]. These returns incorporate predicted rewards and values [38, 44] over a depth of N, which allows to achieve improved stability of convergence, composed as follows:

$$\begin{cases} V_\theta(s^t) & \text{if N} = 0 \\ r_+^t \gamma \left((1-\lambda) V_\theta(s^{t+1}) + \lambda V_\theta^{\lambda,\mathrm{N}-1}(s^{t+1})\right) & \text{if N} > 0 \end{cases} \tag{6}$$

2.2.2 Actor. The actor's loss function, denoted as $\mathcal{L}_\theta^A$, is trained to maximize the batch of actions that lead to states that maximize the critic output.

$$\mathcal{L}_\theta^A = \mathop{\mathbb{E}}_{s\sim\mathrm{D}}\left[\mathcal{L}_\theta^{A,Sub}(s)\right] \tag{7}$$

In the context of Atari Benchmarks, as observed in [19], authors have found it more advantageous to employ the Reinforce [46] 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:

$$\mathcal{L}_\theta^{A,Sub}(s) = \mathop{\mathbb{E}}_{a\sim\pi_\theta(\cdot|s)}\left[-\ln \pi_\theta(a|s)\left(\frac{\bar{V}_\theta(s) - V_\theta(s)}{S_\theta}\right)\right] \tag{8}$$

where S_θ refers to the normalization 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.

$$S_\theta = \max\left(1, \mathrm{Per}_{95}\left(\bar{V}_\theta(\cdot)\right) - \mathrm{Per}_5\left(\bar{V}_\theta(\cdot)\right)\right) \tag{9}$$

2.3 Behavior Cloning

Behavior Cloning (BC) [23] 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.

$$\pi_{BC} = \max_{\pi} \mathop{\mathbb{E}}_{(a,s)\sim\mathrm{D}}\left[\log \pi(a|s)\right] \tag{10}$$

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 bugdet in the given state that they wish to solve, and during the bugdet 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 multi-player, 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 [12, 27] 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 [31], Information Set Monte Carlo Tree Search [13], or Counterfactual Regret Minimization based method [33] 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], 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. More specifically, when selecting the best path of node, MCTS employs PUCT [41], *i.e.* Upper Confident Bound[3] on tree [37] with deep neural network [42].

$$\mathrm{Q}(s,a) + \pi_\theta(a|s)\frac{\sqrt{\mathrm{N}(s)}}{1+\mathrm{N}(s,a)}\left(c_1 + \log(\frac{\mathrm{N}(s)+c_2+1}{c_2})\right) \quad (11)$$

where $\pi_\theta(a|s)$ is the neural network policy assisting MCTS in prioritizing exploration of promising branches, it can perturbed by noise to facilitate exploration. Additionally, $\mathrm{N}(s)$ and $\mathrm{N}(s,a)$ denotes the number of times state s the pair state-action s,a have been visited, encouraging the search to explore less-visited nodes compared to other siblings. Finally, c_1 and c_2 are variables that control the balance between exploration and exploitation.

The sub-actor loss $\mathcal{L}_\theta^{A,Sub}(s)$ is defined as the error between the actor policy and the search policy, and the sub-critic loss $\mathcal{L}_\theta^{C,Sub}(s)$ is defined as the error between the critic value and the search value.

$$\mathcal{L}_\theta^{C,Sub}(s) = y(\bar{V}_{MCTS}(s))^T \log p_\theta \quad (12)$$

$$\mathcal{L}_\theta^{A,Sub}(s) = \pi_{MCTS}(\cdot|s)^T \log \pi_\theta(\cdot|s) \quad (13)$$

3 EXPERT

As mentioned in the introduction, our aim is 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 experts, and then we will explain how to integrate the expert into the RL agent.

3.1 Analysis of the various experts

To thoroughly assess the efficacy of different experts 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 experts discuted 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 experts is often associated with the use of human experts, whether for learning to drive [22, 47], for conversing with other humans [24] 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 [21], 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 an expert, of which this expert happens to be a random agent. The choice of a random agent as an expert 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 an expert 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 expert 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 search algorithm as an expert stands as a reasonable choice given its constant availability in each state and its relatively low cost compared to human expertise. Particularly, in contrast to employing either a random expert or an expert relying solely on past data, search-based algorithms holds 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 expertise, it 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 an expert into the RL agent

Offline RL [30, 35] 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 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.

Table 1: Advantage and Inconvenient of using each expert.

Criteria \ Expert	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

Yes* implies that the algorithm is relevant to improve exploration, but only if the action produced is also relevant to improve performance.

Our approach to incorporating the expert 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 to guide the search process. This choice is based on our ability to leverage the information provided by MCTS. Specifically, we utilize the probability distribution π_{MCTS} to influence the actor policy and the state value V_{MCTS} to shape the critic.

In the subsequent discussion, we adopt general notations that consider the possibility of multiple experts, each exerting varying degrees of influence on the decision-making process. Ideally, we aim to incorporate weights based on the games and the current state. However, as our work is novel in this direction, we initially experiment with a single expert ($|\mathrm{E}| = 1$) and uniform weights across all states ($\forall s, \lambda^{C}_{\mathrm{E}_i}(s) = \lambda^{C}_{\mathrm{E}_i}, \forall \lambda^{A}_{\mathrm{E}_i}(s) = \lambda^{A}_{\mathrm{E}_i}$). Throughout our discussion, E_i represents the ith expert algorithm, and $\mathrm{E} = \{\mathrm{E}_i\}_{i \in \mathcal{N}}$ denotes the set of expert algorithms.

3.2.1 Expert Critic Incorporation. By integrating the expert into the critic, our objective is to refine the estimation of the value function by considering the insights provided by the expert. Incorporating a penalty into the critic using value regularization amounts to change from Equation (4) to equation the following new loss function $\mathcal{L}^{C}_{\theta}$:

$$\mathop{\mathbb{E}}_{s\sim \mathrm{D}}\left[\mathcal{L}^{C,Sub}_{\theta}(s) + \sum_{\mathbf{E}_i \in \mathbf{E}} \boldsymbol{\lambda}^{C}_{\mathbf{E}_i}(\mathbf{s})\mathcal{F}^{C}_{\mathbf{E}_i}(V_{\theta}(\mathbf{s}), \bar{V}_{\mathbf{E}_i}(\mathbf{s}))\right] \tag{14}$$

where $\mathcal{F}^{C}_{\mathrm{E}_i}(,)$ is the penalty term between the target expert $\bar{V}_{\mathrm{E}_i}(s)$ 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 Expert Actor Incorporation. For incorporating the expert on the actor, we have been using both the expert guidance from 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 (7) to the following loss function of the actor $\mathcal{L}^{A}_{\theta}$:

$$\mathop{\mathbb{E}}_{s\sim \mathrm{D}}\left[\frac{\mathcal{L}^{A,Sub}_{\theta}(s)}{\mathbb{E}\left[|\mathcal{L}^{A,Sub}_{\theta}(\cdot)|\right]} + \sum_{\mathbf{E}_i \in \mathbf{E}} \boldsymbol{\alpha}^{A}_{\mathbf{E}_i}(\mathbf{s})\mathcal{F}^{A}_{\mathbf{E}_i}(\boldsymbol{\pi}_{\theta}(\cdot|\mathbf{s}), \boldsymbol{\pi}_{\mathbf{E}_i}(\cdot|\mathbf{s}))\right] \tag{15}$$

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 [48].

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 expert. It is calculated as follows:

$$\lambda^{A}_{\mathrm{E}_i}(s) \cdot \mathrm{Clip}\left[\exp\left(\tau_{\mathrm{E}_i}\frac{\bar{V}_{\mathrm{E}_i}(s) - \bar{V}_{\theta}(s)}{S_{\mathrm{E}_i}}\right), (1, \lambda^{Max}_{\mathrm{E}_i})\right] \tag{16}$$

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.

$$S_{\mathrm{E}_i} = \max\left(1, \mathrm{Per}_{95}\left(V_{\mathrm{E}_i}(\cdot)\right) - \mathrm{Per}_{5}\left(V_{\mathrm{E}_i}(\cdot)\right)\right) \tag{17}$$

4 EXPERIMENTATION

4.1 Experimental Information

In the following, we incorporate several strategies outlined from [21, 37, 49], notably aimed at enhancing stability and generalization in the context of our benchmarks.

4.1.1 Benchmarks. Atari 100k [25] 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) [32]; (iii) A2C with random action as an expert, noted as A2C-Rand (similar to SAC); (iv) A2C with behavior cloning as an expert, noted as A2C-BC; (v) A2C agent with MCTS as an expert, noted as A2C-AZ or A2C-AZ* where A2C-AZ is a general version where the hyperparameter λ^A is fixed for all games and A2C-AZ* is a fine-tuned version where the λ^A if tailored for each game.

In our study, we employed A2C as our chosen RL algorithm; however, it's important to note that our implementation is not restricted to this specific algorithm. Numerous other RL algorithms could have been explored as alternatives. Likewise, although we selected MCTS for our search algorithm, there are numerous other search algorithms that could have been viable options for our 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 $\mathcal{L}_\theta^{C,Sub}()$ uses a cross-entropy based on a discrete representation [21, 37] and the actor loss sub $\mathcal{L}_\theta^{A,Sub}()$ uses reinforce with a 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 expert penalty λ^A is fixed at 0.08 for random and behavior cloning, 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 expert 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 [37, 41, 42, 49]. 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 [36]; (iii) we do not use the search algorithm in the test phase. These differences were made in order 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 being 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.

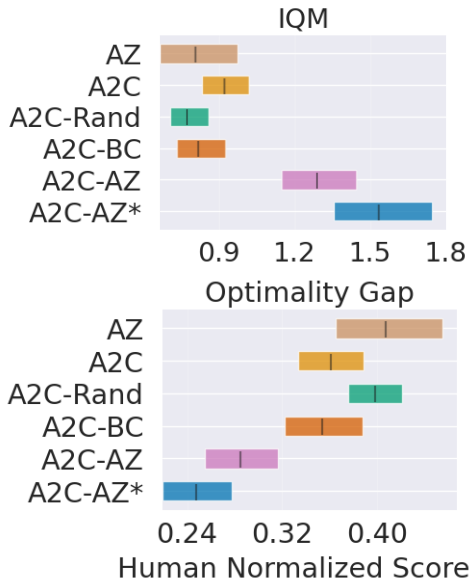


Figure 1: Aggregate performance. Shaded area shows 95% stratified bootstrap confidence interval.

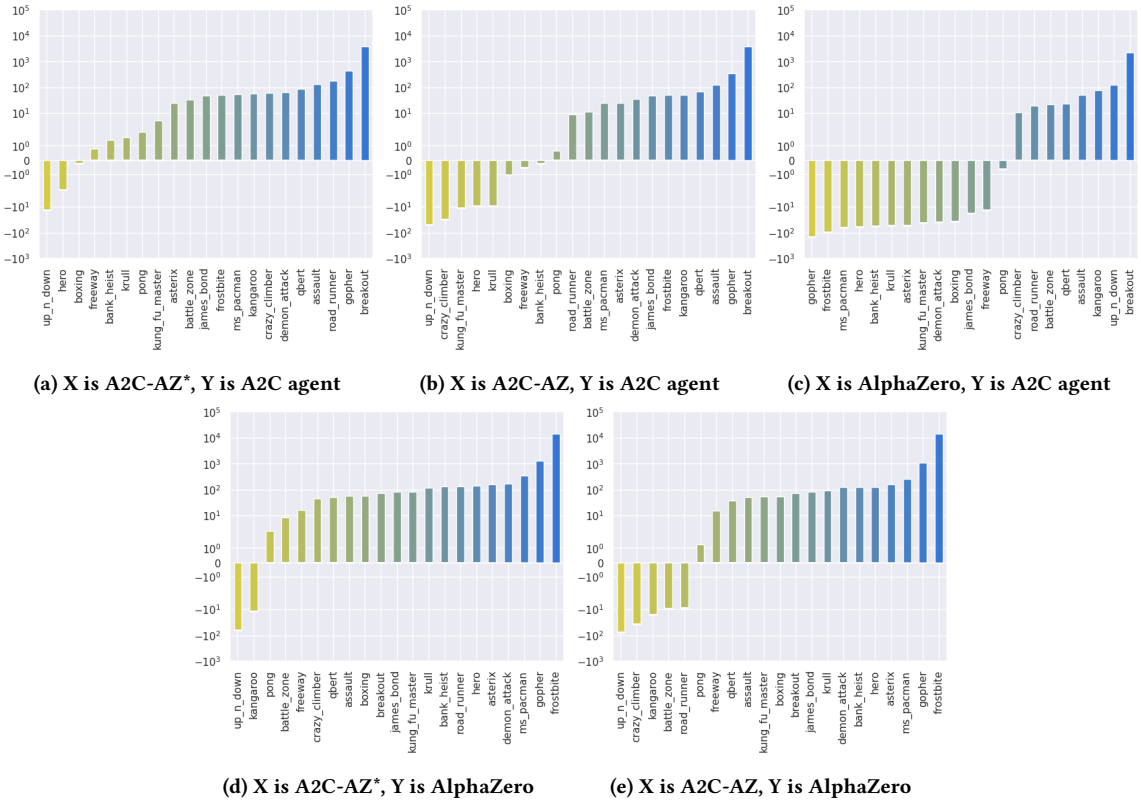
4.1.6 Other. Each agent uses a single environment instance with a single *NVIDIA V100 GPU*. Each algorithm is run using 5 seeds, we evaluated performance every 10k training steps with 10 independant run of the game. To mitigate training expenses, we conducted our experiments bu using a world model. We employed pretrained weights from the Dreamer algorithm [21], a state-of-the-art model-based technique trained over 50, 000k 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.

4.2 Experiments

Initially, we will look at the overall impact of the various algorithms and experts. Subsequently, we narrow our focus on MCTS expert, analysing the experiments in greater detail. Finally, we analyze the impact of the expert's weight, by testing several weights, and trying to observe the impact when the expert is called less often.

4.2.1 Overall analysis. In Figure 1, we observe the overall performance on the IQM and Optimality Gap metrics. We observed that using AZ expert allows A2C agents to obtain better performances on both metrics. More precisely, the A2C-AZ with fixed weight outperforms the A2C agent by more than 0.3 on the IQM and 0.1 on the Optimality Gap. As expected, fine-tuning the weight provides better performance, IQM increases from 1.3 to 1.5 and goes from 0.28 to 0.24 for the Optimality Gap.

4.2.2 MCTS as an expert. In Figure 2, we observe the percentage improvement of A2C-AZ*/A2C-AZ over AlphaZero and A2C, and



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

AlphaZero over A2C. In Figure 3, we present a series of learning curves for the A2C-AZ,A2C and AlphaZero, which serve as the basis for our subsequent analysis.

We begin our analysis with AlphaZero and A2C agent alone. Each figure represents distinct scenarios: one where AlphaZero outperforms A2C (Figure 3b) and reversely (Figure 3a). These figures provide an initial glimpse into the broader picture. More generally, A2C demonstrates superior performance in 12 games, AlphaZero surpasses A2C in 8 games, and 1 equivalent. Despite A2C's general advantage, it is essential to highlight instances where A2C falls short, indicating the potential benefits of integrating AZ as experts.

When considering the incorporation of AZ as an expert, several critical questions arise: can this 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 expert lead to a weakened agent?

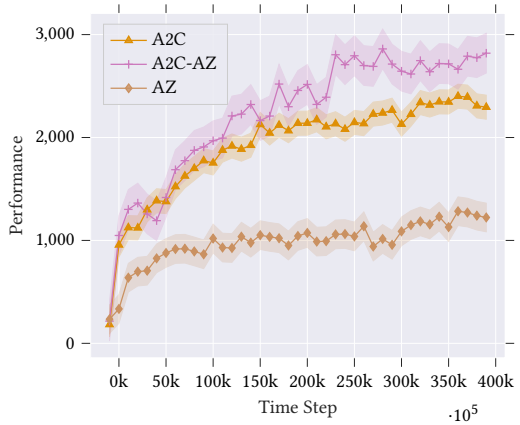
In our analysis across various games, we find that 12/17 games show performance improvements, 4/2 show equivalent, when using the combined approach (A2C-AZ/A2C-AZ*) compared to the A2C agent alone. Interestingly, in the subset of 8 games where AlphaZero outperformed A2C in isolation, integrating AZ as an expert resulted in superior performance in 6/7 of those instances.

Although not explicitly visible in the figure, our observations indicate that the combined algorithm outperforms both individually 9/11 times in the majority of cases, achieves the performance of the better of the two methods(7/9 times), is lower than the best but bounded by the two algorithms 4/1 times, and lower performance than both in only 1/0 instance.

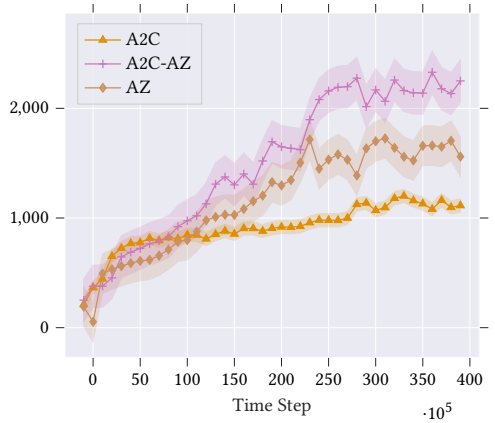
4.2.3 Hyperparameters. In Figure 4, we observe the impact of the weight λ^A on the performance. To do so, we compare multiple fixed weights λ^A ranging from $[0.1, 0.3, 0.5, 0.7]$, and the optimal weight that take the best λ^A for each game. We denote A2C-AZ-X where X denoted the weight used *i.e.*, A2C-AZ-0.3 uses the MCTS expert with a weight of 0.3. As can be observed, the best fixed λ^A find is 0.7 with a IQM of 1.25 and Optimality Gap of 0.28. As one can expected, reducing too much the weight leads to performances that closely resemble those of A2C.

4.2.4 Cost of expert. In practice, there are a number of MCTS methods that can significantly reduce the cost of it, *i.e.* by using batch MCTS [8], parallelization (leaf [9] / root [11] / tree [10]). In addition, many implementation often use several large computational resources to better distribute the workload, as an example, the basic version of AlphaGo uses 40 search threads, 48 CPUs, and 8 GPUs.

In the following experiment, we also show another way to reduce the cost of adding an expert. In addition, the idea is quite natural in our situation. Presently, the expert is executed at every iteration;



(a) Asterix



(b) Assault

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

Table 2: Runtime on Atari100k Benchmarks.

Algorithm	A2C	A2C-AZ-1	A2C-AZ-3	A2C-AZ-5
Time	4H	18H	8H30	6h30

however, our ultimate goal is to avoid deploying the expert in every encountered situation. Our aim is to activate the expert only when deemed necessary—specifically, in scenarios where our RL agent faces challenges or when the expert is known to excel.

In Figure 5, we observe the impact of using less often the expert *i.e.*, instead of using the expert at each iteration, we use it at every N iterations. We introduce the notation of A2C-AZ-X where X indicates how often the expert is called *i.e.*, A2C-AZ-3 uses the MCTS expert every 3 training step. Additionally, we observe in Table 2, the overhead cost of using AZ as an expert according to N. As observed, we can reduce the overhead of using MCTS but while maintaining the improvement performance.

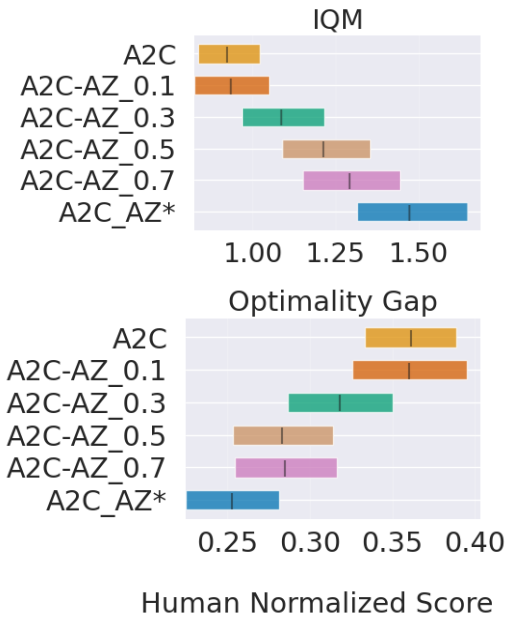


Figure 4: Aggregate performance metrics according to the weight. The shaded area shows 95% stratified bootstrap confidence interval.

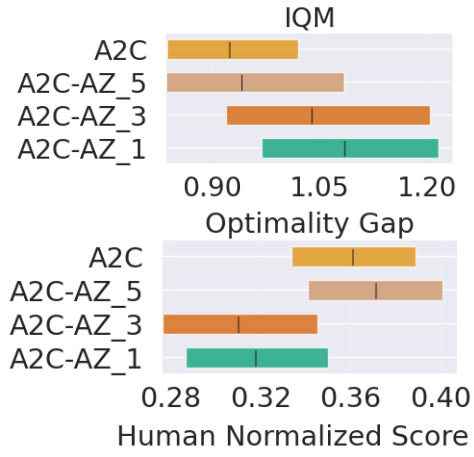


Figure 5: Aggregate performance metrics according to the weight when reducing the number of call to the expert. The shaded area shows 95% stratified bootstrap confidence interval.

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. And therefore, many of

the methods available can be applied in our situation, especially, offline RL methods generally rely on staying close to the data. Depending on how this proximity is implemented, there are several approaches to achieve this goal, including estimating the behavior policy and regularization to closely align with it [16, 24], ensuring that the learning of the policy is in-distribution [15, 34], or recently, estimating the Q-value of the behavior policy and regularization for proximity toward it [40].

Within the realm of regularization methods, there exist several avenues to explore. These include penalties applied within the reward function [48] or regularization penalties applied after its computation of the loss [28, 29]. Additionally, the calculation of the penalty is accomplished by using various functions, including KL divergence, Maximum Mean Discrepancy [48], or even Fisher information [28].

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 offline neural networks to boost its performance [41–43], 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 guiding expert while retaining the RL module. The closest related study we found is [26], where the authors use A3C with K workers being MCTS among N workers, this integration notably led to performance improvements. Nonetheless, unlike their approach, we employ MCTS as an expert in each state and take into account the value returned by MCTS to enhance learning.

6 CONCLUSION

In this paper, we study the influence of leveraging online algorithms as expert guides 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 expert. Our approach involves regularising the loss functions for both the actor and the critic to enforce our loss to take into account the information given by the expert.

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 an expert yields superior results compared to employing either of the two methods in isolation. Moreover, performance gains can be extended with the incorporation of an adaptive weight, contributing to enhanced outcomes while ensuring cost-effectiveness. Our study is based on the Atari100k benchmarks.

In the future, there exist promising avenues for further exploration. Experimenting with diverse hyperparameters, such as alternative distance functions, different planning algorithms or different reinforcement learning algorithms, could illuminate nuanced insights. Additionally, extending our experiments to encompass a broader spectrum of benchmarks holds potential for expanding the applicability and robustness of our approach.

REFERENCES

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. 2021. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems* 34 (2021).

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866* (2017).

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47 (2002), 235–256.

[4] Anton Bakhtin, David J Wu, Adam Lerer, Jonathan Gray, Athul Paul Jacob, Gabriele Farina, Alexander H Miller, and Noam Brown. 2022. Mastering the game of no-press Diplomacy via human-regularized reinforcement learning and planning. *arXiv preprint arXiv:2210.05492* (2022).

[5] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In *International conference on machine learning*. PMLR, 449–458.

[6] Marc G Bellemare, Will Dabney, and Mark Rowland. 2023. *Distributional reinforcement learning*. MIT Press.

[7] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4 (2012), 1–43.

[8] Tristan Cazenave. 2021. Batch Monte Carlo Tree Search. *ArXiv* abs/2104.04278 (2021).

[9] Tristan Cazenave and Nicolas Jouandeau. 2007. On the parallelization of UCT. In *Computer games workshop*.

[10] Tristan Cazenave and Nicolas Jouandeau. 2008. 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*. Springer, 72–80.

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

[12] Quentin Cohen-Solal and Tristan Cazenave. 2020. Minimax strikes back. *arXiv preprint arXiv:2012.10700* (2020).

[13] Peter I. Cowling, Edward Jack Powley, and Daniel Whitehouse. 2012. Information Set Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games* 4 (2012), 120–143.

[14] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. 2024. Stop Regressing: Training Value Functions via Classification for Scalable Deep RL. *arXiv preprint arXiv:2403.03950* (2024).

[15] Scott Fujimoto and Shixiang Shane Gu. 2021. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems* 34 (2021), 20132–20145.

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

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

[18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[19] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2020. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193* (2020).

[20] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2019. Dream to Control: Learning Behaviors by Latent Imagination. *CoRR* abs/1912.01603 (2019). arXiv:1912.01603 http://arxiv.org/abs/1912.01603

[21] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2023. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104* (2023).

[22] Zhiyu Huang, Jingda Wu, and Chen Lv. 2022. Efficient deep reinforcement learning with imitative expert priors for autonomous driving. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[23] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 1–35.

[24] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. 2019. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456* (2019).

[25] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. 2019. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374* (2019).

[26] Bilal Kartal, Pablo Hernandez-Leal, and Matthew E Taylor. 2019. Action guidance with MCTS for deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, Vol. 15. 153–159.

[27] Donald E. Knuth and Ronald W. Moore. 1975. An analysis of alpha-beta pruning. *Artificial Intelligence* 6, 4 (1975), 293–326. https://doi.org/10.1016/0004-3702(75)90019-3

[28] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. 2021. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*. PMLR, 5774–5783.

[29] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 1179–1191.

[30] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).

[31] Jeffrey Richard Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. 2010. Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search. In *AAAI*.

[32] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

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

[34] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. 2019. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177* (2019).

[35] Rafael Figueiredo Prudencio, Marcos ROA Maximo, and Esther Luna Colombini. 2023. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[36] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

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

[38] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[40] Laixi Shi, Robert Dadashi, Yuejie Chi, Pablo Samuel Castro, and Matthieu Geist. 2023. Offline Reinforcement Learning with On-Policy Q-Function Regularization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 455–471.

[41] D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2016), 484–489.

[42] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv* abs/1712.01815 (2017).

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

[44] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press, Cambridge, MA, USA.

[45] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2023. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review* 56, 3 (2023), 2497–2562.

[46] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8 (1992), 229–256.

[47] Jingda Wu, Zhiyu Huang, Zhongxu Hu, and Chen Lv. 2023. Toward human-in-the-loop AI: Enhancing deep reinforcement learning via real-time human guidance for autonomous driving. *Engineering* 21 (2023), 75–91.

[48] Yifan Wu, George Tucker, and Ofir Nachum. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361* abs/1911.11361 (2019). https://api.semanticscholar.org/CorpusID:208291277

[49] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. 2021. Mastering atari games with limited data. *Advances in Neural Information Processing Systems* 34 (2021), 25476–25488.